GENERATIVE ADVERSARIAL NETWORK (GAN) FOR MEDICAL IMAGE SYNTHESIS AND AUGMENTATION

ZHAOHUI LIANG

A DISSERTATION SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, CANADA

September, 2022

Abstract

Medical image processing aided by artificial intelligence (AI) and machine learning (ML) significantly improves medical diagnosis and decision making. However, the difficulty to access well-annotated medical images becomes one of the main constraints on further improving this technology.

Generative adversarial network (GAN) is a DNN framework for data synthetization, which provides a practical solution for medical image augmentation and translation. In this study, we first perform a quantitative survey on the published studies on GAN for medical image processing since 2017. Then a novel adaptive cycle-consistent adversarial network (Ad CycleGAN) is proposed. We respectively use a malaria blood cell dataset (19,578 images) and a COVID-19 chest X-ray dataset (2,347 images) to test the new Ad CycleGAN. The quantitative metrics include mean squared error (MSE), root mean squared error (RMSE), peak signal-to-noise ratio (PSNR), universal image quality index (UIQI), spatial correlation coefficient (SCC), spectral angle mapper (SAM), visual information fidelity (VIF), Frechet inception distance (FID), and the classification accuracy of the synthetic images. The CycleGAN and variant autoencoder (VAE) are also implemented and evaluated as comparison.

The experiment results on malaria blood cell images indicate that the Ad CycleGAN generates more valid images compared to CycleGAN or VAE. The synthetic images by Ad CycleGAN or CycleGAN have better quality than those by VAE. The synthetic images by Ad CycleGAN have the highest accuracy of 99.61%. In the experiment on COVID-19 chest X-ray, the synthetic images by Ad CycleGAN or CycleGAN have higher quality than those generated by variant autoencoder (VAE). However, the synthetic images generated through the homogenous image augmentation process have better quality than those synthesized through the image translation process. The synthetic images by Ad CycleGAN have higher accuracy of 95.31% compared to the accuracy of the images by CycleGAN of 93.75%.

In conclusion, the proposed Ad CycleGAN provides a new path to synthesize medical images with desired diagnostic or pathological patterns. It is considered a new approach of conditional GAN with effective control power upon the synthetic image domain. The findings offer a new path to improve the deep neural network performance in medical image processing.

Acknowledgements

First, I would like to express my sincere gratitude to my supervisor Prof. Jimmy Xiangji Huang for his guidance, motivation, enthusiasm, and rigorous attitude. During my Ph.D. study with Prof. Jimmy Xiangji Huang, I have the opportunity to extend my knowledge and skills to one the most advanced fields in computer science: artificial intelligence and deep learning, and to become a researcher in the cross-disciplinary research area of biomedical informatics. With his patience, I finished my master's degree in Information System and Technologies and successfully transferred from a healthcare researcher to a Ph.D. in Computer Science. Prof. Jimmy Xiangji Huang also provided me a lot of opportunities to conferences and introduced me to the most renowned scholars in the academia community of computer science. Thanks to these, I am able to be close to the cutting edge of science. I am grateful to have Prof. Jimmy Xiangji Huang as my Ph.D. supervisor.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Augustine Wong, Prof. George Georgopolous, and Prof. Aijun An, for their advice, knowledge, and insightful comments. I also would like to thank my dissertation external examiner, Prof. Huiru Jane Zheng. She provided many insightful and detailed suggestions for improving my thesis.

In addition, I offer my regards and blessings to all best friends in the academic community, particular Dr. Stefan Jaeger, Dr. George R. Thoma, and Dr. Sameer K Antani from Lister Hill National Center for Biomedical Communications (LHNCBC), NLM, NIH, USA, for their generosity of providing me with the wonder internship in NIH, which inspires me to concentrate my research on the innovative deep neural network for medical image processing. It is no exaggeration to say that the excellent experience with LHNCBC is the origin of my thesis.

Last but not the least, I would like to thank my family. My beloved father Jianglun Liang and mother Yusui Huang always take care of me and support me in my whole life. My

wife, Eva Yuan, always encourages my study and research. Because of their support, I can sit down and write my papers and this thesis with a peaceful mind.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1 Introduction and Motivation

1.1 Motivation

Computer vision is one of the most promising fields of applying the deep learning AI technologies to biomedical and health informatics in the last decade. The technologies of machine learning (ML) and AI is playing an increasing role on medical image analysis because of their powerful image processing capacity in many applications such as image-based diagnosis, image interpretation, pathological pattern segmentation, image-guided surgery, and image retrieval and analysis, etc. This technology facilitates and improve the working efficiency of the whole healthcare delivery from diagnosis, prognosis prediction and disease prevention. In medical research, the genome sequencing technology highly relies on the accurate recognition of the light patterns from the fluorescent immunoassay reaction to detect the variations of the gene or peptide sequence, which is the basis of the current genomic analytic technology. These ML technologies range from the conventional algorithms such as linear regression, k-nearest neighbors (KNN), Bayes classifier, support vector machine (SVM), to the state-of-the-art deep neural networks (DNNs), or deep learning methods, such as convolutional neural network (CNN), recurrent neural network (RNN), and deep belief network (DBN), etc. The conventional models are usually restricted by some drawbacks such as the dependency of the expertise for data pre-processing, which requires more time and effort to tune the feature to be learnable by the ML models. The new advance of deep learning algorithms effectively reduces the cost of data pre-processing. The deep learning algorithms has been applied to medical informatics for multiple domains such as image processing, electric health record (EHR) entity recognition, computer-aided diagnosis, medical genomics, and drug discovery, etc.

Deep learning is a branch of machine learning algorithms where a trained model will automatically yield a decision given the experience from the seen training data. Deep learning is a machine learning model based on an artificial neural network with multiple

hidden layers. Compared to the conventional machine learning methods, deep learning has four significant strengths.

i. **Automatic feature extraction**: The conventional machine learning algorithms that are limited in processing data in the raw form. If the raw data cannot provide learnable and distinguishable patterns, the trained model will have poor performance. Therefore, the conventional machine learning technology requires a complex feature extraction procedure that involves the collaboration of the expertise of a domain and computer science on developing a proper feature representation and extraction paradigm to maximize the system performance. In comparison, the deep learning methods apply a hierarchic architecture to the data patterns through multiple levels of data representations captured by the deep hidden layer model connected with "simple but non-linear modules" to form a complex but learnable function. By the deep architecture, the data patterns are effectively separated and amplified through the deep architecture. When trained adequately, different neurons in the deep neural network are optimized to be capable of capturing some specific pattern passed through the whole network and the corresponding weights are tuned to perform the due response. [1]

ii. **End-to-end models**: the deep learning model is composed of neural network with multiple layers of neurons with different functionality given a particular task. The functions between layers are different from each other by their tasks. The layers are connected by the weights produced by the activation functions attached to each neuron. The whole architecture of a deep learning model is demonstrated by a directed graph, where the nodes represent the neurons holding the optimized weights and the edges represent the output by the activation functions of the neurons to the next layers. Therefore, to generate a deep learning network, we simply define the architecture of the whole graph composed of the pre-configured layers given the tasks. There is no need to embed multiple algorithms into the whole model therefore the overall solution will be robust.

iii. **Superior performance**: deep learning usually has better performance compared to other algorithms given sufficient training. In another word, deep learning is a data greedy algorithm become of the complexity of its network architecture. On the other hand, the highly complex hierarchic structure of the deep network provides powerful modeling capacity to memorize almost all possible mappings from the large training set after sufficient training, and the trained deep learning model usually gives robust and intelligent prediction by making proper interpolation or extrapolation to new or unseen instances.

iv. **General-purpose method**: the deep learning models have the capacity to extend the applications to various tasks by fine-tuning the weights at different layers of the network. The common way is called transfer learning. The basic idea of transfer learning is to use a pre-trained network to solve a different problem. However, transfer learning is feasible giving the patterns learning from the pre-trained dataset can be generalize to the new dataset [2], which is impractical in medical images so far. For example, we cannot transfer a pre-trained model trained by an X-ray image dataset to an MRI (magnetic resonance imaging) image set, or the patterns learned from a histological image set cannot applied to another ultrasound image set. In general, the transfer learning strategy is an optimal solution for implementing deep learning models (e.g., convolutional neural network, CNN) to medical image analysis, but we are still at the stage of data congregation or the formation of the big data platform that can provide sufficient pre-trained model to extend deep learning to all fields of medical images.

Besides the above strengths, DNNs have two common drawbacks. First, all the DNNs are considered as data greedy algorithms. The final performance of DNNs mainly relies on the abundance of training examples, or the accessibility of big data samples that fully represent the characteristics of the entire data domain. However, big image datasets with good annotations are difficult to acquire. This problem is even worse for the acquisition of medical image datasets because high quality annotation must be performed

by medical experts. The cost for such datasets is more expensive than general-purposed datasets. If the medical images are annotated by non-medical persons, the quality of the image data is suspicious due to the lack of expertise. Thus, if the technology of artificial intelligence (AI) such as DNNs can provide a method to automatically annotate the medial images, the cost not only for the medical image processing but also for the relevant medical service will be significantly reduced. This becomes the first motivation of this research.

Second, the specific medical image patterns are different from general-purposed images such as those in the ImageNet dataset [2]. When using transfer learning with DNN models trained by the ImageNet to fine tune a new model for the medical images, the pretrained feature extractors usually cannot effectively capture the medical significant patterns through the complex architecture but simply develop meaningless combinations for the final decision. In our previous work on CNN for the malaria blood cell image classification, the transfer learning approach has lower accuracy (91.99%) than the randomly initialized CNN (97.37%) [3]. In addition, a new study reveals that the seemly high-performance DNN models for COVID-19 chest X-Ray image detection are vulnerable from network attacks [4]. The common strategy to improve DNN performance is to enhance the diversity of training data by multiple augmentation techniques such as random rotation, flipping, and jittering. However, the conventional augmentation methods are unsuitable for most medical images like images of histological cells and tissues, or X-Ray photography. The image-based medical diagnosis usually requires structure completeness and correct image alignment because the diagnosis is usually based on the comparison between normal and abnormal structure. The random augmentation techniques are likely to break the structure completeness or position alignment. As a result, the trained DNNs are likely to capture wrong combination of patterns or artifacts instead of the correct ones meeting the human knowledge. Therefore, our second motivation is to explore a new method to synthesize homogenous images to preserve the meaningful patterns and meanwhile to reduce the DNN vulnerability for medical image processing.

1.2 Main Contributions

This thesis proposes a new generative adversarial network (GAN) architecture, namely adaptive generative adversarial network (Ad CycleGAN) based on the state-of-the-art cycle-consistent adversarial network (CycleGAN). In addition to the two generator-discriminator pairs of the original CycleGAN, a pertained classifier is added to the Ad CycleGAN architecture as an internal criterion to further control the output synthetic images not only belong to the target domain, but also to the due image class. Medical images are mainly for diagnostic and disease prognosis purposes. The diagnostically or pathologically significant patterns in the medical images are likely to be considered as the acceptable diversity of the small image domain. Therefore, it is necessary to enhance the current Cycle GAN architecture by adding external criterion to ensure that the generated synthetic images belong to both the correct image domain and the correct diagnosis class. This design will be easy to extend to other medical or non-medical data synthetization applications.

Another contribution of the thesis is that we propose a new term loss term, namely classification loss to the GAN composite loss objective function during model optimization. Unlike the original loss objective design, the new loss term needs to periodically inject extra loss weight to the total generator loss term, and decays as the optimization epoch increases to prevent from generating unnecessary artifacts to the synthetic images.

To evaluate the performance of the new Ad CycleGAN, we implemented the original Cycle and the convolutional variational autoencoder (CVA) for comparison. Multiple quantitative measurements are applied to measure the similarity of the real medical images and the corresponding synthetic images, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Peak Signal-to-Noise Ratio (PSNR), Universal Image Quality Index (UIQI), Spatial Correlation Coefficient (SCC), Spectral Angle Mapper (SAM), and Visual Information Fidelity (VIF). Furthermore, the Frechet Inception Distance (FID) is

applied to quantitatively measure the generated images by different generative models. The details of these measurements will be discussed in the following chapters.

1.3 Outline

This thesis has eight chapters in total. They are organized as follows. In chapter 1, the general idea is computer vision, and the relevant technologies are introduced. Then the main challenges for improving the current deep learning performance in biomedical research is presented with our proposed solution. The novel Ad CycleGAN architecture is also introduced in Chapter 1 with the corresponding evaluation metrics. Chapter 2 covers the related work of artificial neural network, its basic components and architecture, the classical DNNs, the typical applications of DNNs, the advanced technique of DNN and the typical application of DNN in biomedical research and practice, the basic knowledge of GAN and typical GAN models. Chapter 3 is a quantitative survey of GAN for biomedical applications. Chapter 4 introduces the basic knowledge and components of the novel Adaptive Cycle-Consistent Adversarial Network (Ad CycleGAN). Chapter 5 and Chapter 6 respectively describe the Application of Ad CycleGAN for histology and radiology image synthesis, as well as the relevant experiments and result interpretation. Chapter 7 summarizes the experiments and concludes the findings of this thesis. Chapter 8 concludes this thesis and explores the possibilities of future work.

Chapter 2 Related Work

2.1 Artificial Neural Network and Deep Learning

The tasks of machine learning (ML) are general divided into two categories: supervised learning and unsupervised learning. Supervised learning is to infer a mapping function $y = f(x)$ from the input x to the output y. The tasks of classification and regression are examples of supervised learning. In contrast, unsupervised learning is to learn the patterns from the distribution of the data x without pre-defined assumptions. Typical examples of unsupervised learning are clustering and density estimation.

On the other hand, all ML methods have one fundamental goal to extract the latent feature for the training data as the representation of the inputs to establish the optimized mapping from the input to the output. Unlike the conventional ML models requiring additional feature extractors for a complex ML pipeline, the deep learning ML models usually accommodate an automatic data-oriented feature extractor inside the deep learning model architecture. Therefore, the deep learning ML models can learn optimal patterns directly from the training data without the human expertise intervention. This capacity of automatic pattern discovery lets the deep learning techniques unveil the unknown or hidden feature possibly neglected by human. The complex data representation in deep learning consists of simple representations captured by the deep learning architecture during training. For example, the recognition of a medical pattern such as a malignant tumor involves the finding of multiple visual feature such as the edges, contours, and corners in a specific special composition, which can be learned in an unsupervised manner and later pipelined to a supervised learning task by the following layers of the deep learning model.

The major deep learning algorithms are built upon the framework of the artificial neural network. Artificial Neural Networks, or ANN is a computational model for machine learning. The structure of an ANN is to simulate the functions of the signal transmitting mechanism of the neural system in biology. Unlike other machine learning algorithm like support vector machine (SVM) that seeks a unique optimized solution, an ANN is a

nonlinear statistical model with complex functional mapping relations between the inputs and outputs throughout the multiple hidden layers in the middle. And this complexity makes the ANN model become deep learning and acquire the learning capacity as a mimic of a biological neuron.

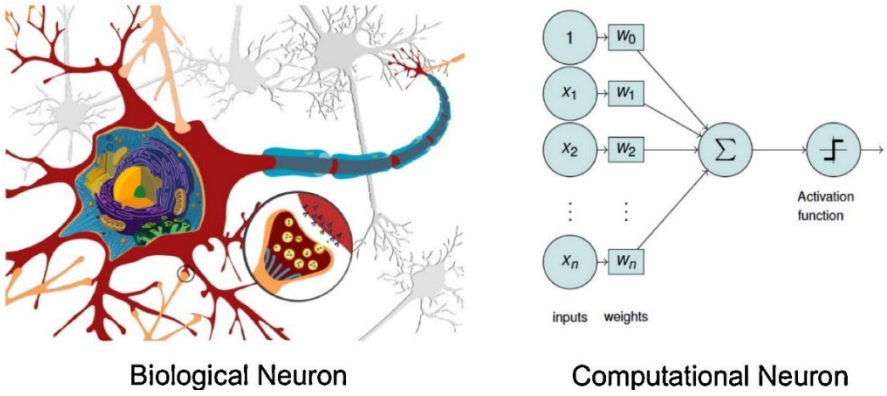


Figure 1: Comparison of Biological Neuron and Computational Neuron [5].

An ANN model is composed of multiple computational neurons or nodes, which are arranged as layers from the input to the output of the model. The layers of nodes that do not belong to the input or the output layers are called hidden layers. The nodes in the hidden layers have a set of weights that will be updated during training. And the weights of the whole ANN model are optimized by minimizing the loss function. For example, if a negative logarithm function is used as the loss function, the update will be presented as:

$$\mathcal{L}(\theta, D) = -\sum_{i=0}^{D}[logP(Y = y_i|x_i, \theta)] + \lambda\|\theta\|_p \quad (2.1)$$

where D is the entire training set, θ is the set of the model parameters that are updated by minimizing the p-norm loss function $\mathcal{L}$, and λ is the regularization term to prevent overfitting and to improve the model ability to generalize to new unseen data. Deep learning mainly uses the backpropagation method to minimize the loss from the last layer reversely throughout the network model [7].

There are many open-source implementation of the deep learning algorithms in multiple programming languages, including TensorFlow, Theano, PyTorch, Caffe, MXNet, Deeplearning4J, and ML.Net. The current deep learning models used in medical informatics is illustrated in Figure 2.

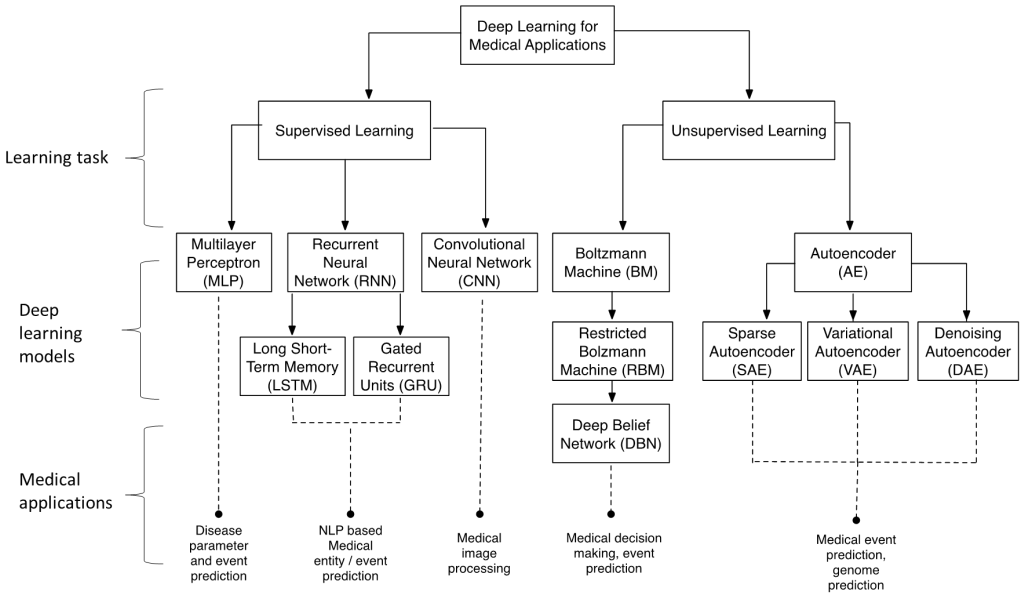


Figure 2: Current Deep Learning Models for Medical Informatics [5].

2.1.1 Artificial Neural Network and Deep Learning

In mathematics, a computational neuron is the basic unit of an ANN that receives a vector of weights $w = (w_1, \dots, w_n)$ and a bias $w_0$as the parameters $\theta = (w_0, w_1, \dots, w_n)$ to seek for a model decision by the mapping function:

$$\hat{f}(x) = h(w^T x + w_0) \tag{2.2}$$

By a certain non-linear function called activation $h(x)$. Therefore, a single computational neuron can be used as a classifier if the function of the activation is monotonic, bounded, and continuous [5]. Therefore, different types of activations may be chosen for different tasks. For example, Rosenblatt respectively used the sigmoid function $sign(x)$, the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, or the hyperbolic tangent function $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ for the primitive ANN models, the perceptron algorithms in the 1950's. the main demerit of using a single computational neuron for ML problems is that it cannot solve XOR problem because on the linearity of its decision boundary (Figure 3).

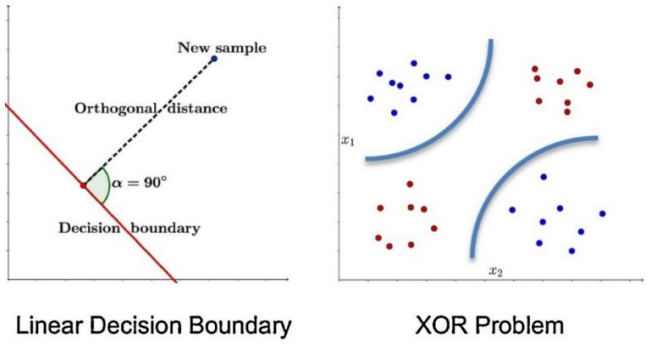


Figure 3: Limitation of Linearity of the Decision Boundary [5].

To overcome the demerit of the single perceptron, the idea is extended to the multilayer perceptron (MLP) algorithm. An MLP is an ANN composed of multiple hidden layers where the nodes (or neurons) in layer i are fully connected to the nodes in the next layer $i + 1$ (Figure 4). Given an MLP has i hidden layers and each layer has j nodes, the predicted output, or the hypothesis of the MLP is presented as:

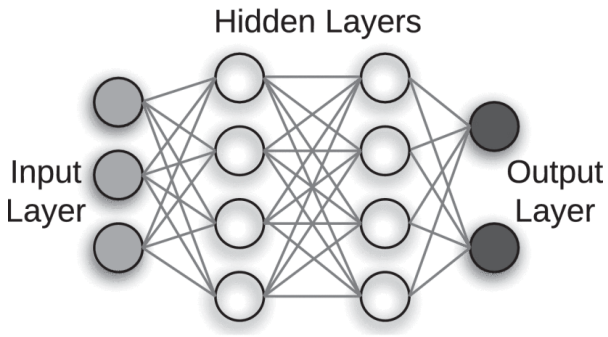


Figure 4: Multilayer Perceptron (MLP) [16].

$$\widehat{f}_i(x_i) = h_i = \sigma\left(\sum_{j=1}^{d} x_j w_{ij} + b_{ij}\right) \tag{2.3}$$

where $h_i$is the output (hypothesis) computed by each node which is the mapping of the weighted sum of the outputs from the previous layer $i - 1$, by a nonlinear activation σ. The conventional activation functions are the sigmoid or the hyperbolic tangent (tanh), but the modern ANN model prefers to choose simple functions such as rectified linear units (ReLU) to reduce the computing cost. The MLP algorithm is considered as a primitive model for deep learning with its extensible hidden layer architecture. When adequate

hidden layers are added to the MLP model, the weights are optimized by the training data and yield a reasonable mapping between the input and the output.

From the MLP model, deep learning advances by introducing new layers to improve its capacity of data-driven feature learning and extraction when the dataset goes through its deep architecture during training. The significant change of the current deep learning algorithms compared to the conventional MLP model is the data representation driven by training. The traditional machine learning methods highly relies on the hand-crafted training data, which means the learnable patterns from the raw data must be extracted by expertise. This feature extraction procedure is usually laborious and time consuming thus it is considered as the "black art" of machine learning [8], and partly makes the performance of machine learning unreliable. In contrast, deep learning uses a part of its architecture (e.g., the convolutional layers for image pattern learning) to learn optimal features directly from the raw data with minimum processing and without human intervention. By this automatic latent data pattern discovery capacity, deep learning techniques provider the state-of-the-art pattern extraction method to build the high-level complex data representation from for basic, simple representations. For example, medical image recognition needs to extract the representation of edges, contours, and corners of the interested pixel patterns from the raw image, and these basic features can be eventually combined in the complex representation to let the machine to discriminate the significant medical patterns in a certain use case.

2.1.2 Optimization of Deep Neural Network

Based on the topology of the artificial neural network (ANN) and the perceptron algorithm, a trained single layer network with N neurons can predict an output by the linear combination of the neurons:

$$\hat{f}(x) = v_i\sigma\left(\sum_{i=1}^{N-1}\left(w_i^T x + w_{0,i}\right)\right) = \sum_{i=1}^{N-1} v_i\sigma\left(w_i^T x + w_{0,i}\right) \tag{2.4}$$

where v_i represents the weights of N neurons. And all trainable parameters of the whole network can be summarized as:

$$\theta = \left(v_0, w_{0,0}, w_0, \dots, v_N, w_{0,N}, w_N\right)^T \tag{2.5}$$

where θ represents all trainable parameters in the network. The training goal of the network is to is to find the optimal θ to minimize the difference of the predicted value $\hat{f}(x)$ and the ground true value $f(x)$ bounded by $\left|f(x) - \hat{f}(x)\right| < \epsilon$. The optimization method for deep learning is gradient descent. To compute the gradient, we define a loss function $\mathcal{L}(\theta)$ to optimizing the parameter set θ by minimizing $\mathcal{L}(\theta) \sim \left|f(x) - \hat{f}(x)\right|$ with the back-propagation algorithm.

Given a layer with a linear activation $\hat{y} = \hat{f}(x) = Wx$ as a matrix multiplication where $y \in \mathbb{R}^m$, we use a L2 loss function presented as:

$$\mathcal{L}(\theta) = \frac{1}{2}\left\|\hat{f}(x) - y\right\|_2^2 = \frac{1}{2}\|Wx - y\|_2^2 \tag{2.6}$$

To update the parameter set $\theta = W$, we take the partial derivative $\mathcal{L}$ over W:

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial \hat{f}} \frac{\partial \hat{f}}{\partial W} = (Wx - y)(x^T) \tag{2.7}$$

using the chain rule. Then the final weight update is obtained as:

$$W^{j+1} = W^j + \eta(W^j x - y)x^T \tag{2.8}$$

where η is the learning rate to determine how much the update will be, j is the index of the iteration when the update of the weights occur.

If we extend the network structure from one layer to three layers, and we still use the linear activations in each layer, the predicted output is presented as:

$$\hat{y} = \hat{f}_3\left(\hat{f}_2\left(\hat{f}_1(x)\right)\right) = W_3 W_2 W_1 x \tag{2.9}$$

and the corresponding loss function is revised as:

$$\mathcal{L}(\theta) = \frac{1}{2}\left\|\hat{f}_3\left(\hat{f}_2\left(\hat{f}_1(x)\right)\right) - y\right\|_2^2 = \frac{1}{2}\|W_3 W_2 W_1 x - y\|_2^2 \tag{2.10}$$

We collapse the weights of all three layers into a single notation θ representing all trainable parameters $\theta = \{W_1, W_2, W_3\}$. When the network is trained, the θ can be updated by gradient descent using backpropagation, i.e., to compute the gradient from the output layer towards the input layer. At first, we compute the gradient of the last layer W_3:

$$\frac{\partial \mathcal{L}}{\partial W_3} = \frac{\partial \mathcal{L}}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_3} = (W_3 W_2 W_1 x - y)(W_2 W_1 x)^T \tag{2.11}$$

then we compute the gradient of the second layer W_2 by applying the chain rule twice:

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{f}_3}\frac{\partial \hat{f}_3}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{f}_3}\frac{\partial \hat{f}_3}{\partial \hat{f}_2}\frac{\partial \hat{f}_2}{\partial W_2} = W_3^T(W_3W_2W_1x - y)(W_1x)^T \tag{2.12}$$

finally, we compute the gradient of the first layer W1 by applying the chain rule three times:

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial \hat{f}_3}\frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial \hat{f}_3}\frac{\partial \hat{f}_3}{\partial \hat{f}_2}\frac{\partial \hat{f}_2}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial \hat{f}_3}\frac{\partial \hat{f}_3}{\partial \hat{f}_2}\frac{\partial \hat{f}_2}{\partial \widehat{f_1}}\frac{\partial \widehat{f_1}}{\partial W_1} = W_2^T W_3^T(W_3W_2W_1x - y)(x)^T \tag{2.13}$$

Note that the above back-propagation procedure for computing the gradients of each layer is also applicable to non-linear activation functions, which is illustrated in Figure 5.

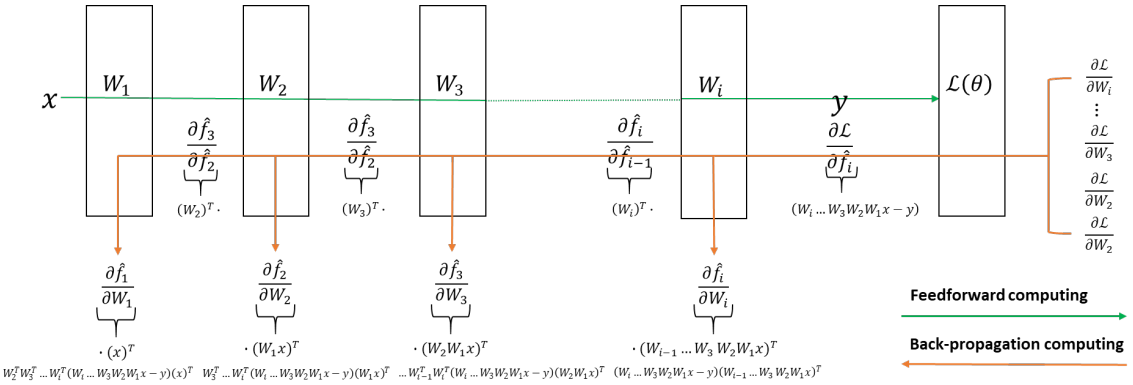


Figure 5: Graphical Overview of a DNN.

A neural network can be trained by the feedforward and back-propagation procedure illustrated in Figure 4. However, a deep neural network (i.e., a neural network with many hidden layers) cannot be effective trained unless some technical issues are solved. One important factor is the selection of activation functions. In deep learning, unlike the classical bounded activations such as the sigmoid function (σ(x)), the hyperbolic tangent function (tanh (x)), and the sign function, the typical examples of the activations for deep learning are the rectified linear unit function (ReLU) and Leaky ReLU [5].

$$\text{ReLU(x)} = \begin{cases} x & if\ x \geq 0 \\ 0 & else \end{cases}$$

$$\text{LReLU(x)} = \begin{cases} x & if\ x \geq 0 \\ \alpha x & else \end{cases} \tag{2.14}$$

The ranges and boundaries of these activation functions are illustrated in Figure 6.

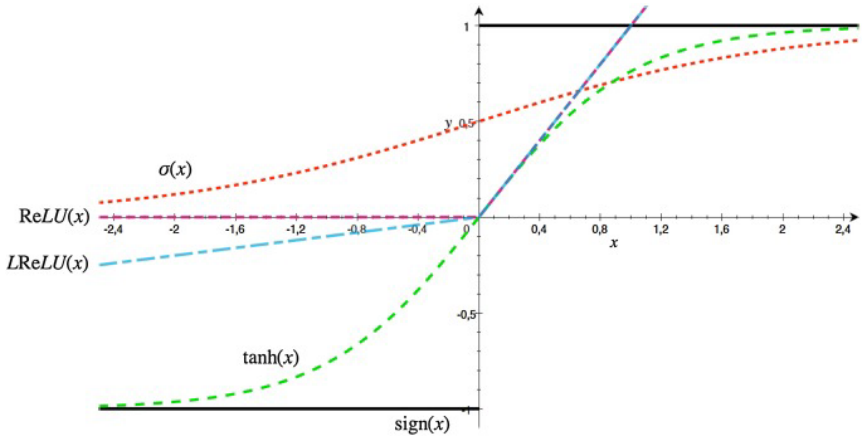


Figure 6: Comparison of Different Activations [5].

As shown in Figure 6, the new activations for deep learning (e.g., ReLU and LReLU) are convex and have a large range with non-zero derivatives for the convenience to compute the gradients through the deep hidden layers during back-propagation. This design provides a feasible solution for the gradient descent optimization by repeatedly applying the chain rule of many multiplications of partial derivatives (Figure 2.5) through a deep network architecture. One typical problem of gradient descent optimization is that if the gradients at several layers are less than 1, it will cause a cascade effect on the entire gradient decays dramatically, or gradient vanishing. If too many gradients from the neuron become zero, the corresponding neurons will lose the capacity as a classifier if the decision boundary is set to zero. This numeric computation issue used to be a main obstacle for deep neural network optimization until the non-saturating derivatives are introduced. Note that the universal approximation theorem still holds for a single hidden layer with ReLU as the activation [9].

A defect of using ReLU as the activation is that the function output is not differentiable throughout the entire domain. When $x = 0$, the function does not have a unique gradient. For the gradient descent optimization, an important property of the gradient is that it will point towards the direction of the steepest ascent. In other words, the optimization algorithm will follow the opposite direction to minimize the function. If the

function is differentiable, this direction is θ unique. However, if the constraint is released, i.e., we allow multiple directions leading to the extremum, we can apply the sub-gradient theory [10] (i.e., at least one sub-gradient towards the optimum) that the gradient descent algorithms are still applicable to the optimizing of the deep networks. Furthermore, the sub-gradient theory provides the basis of many gradient descent algorithms such as stochastic gradient descent (SGD), Nesterov accelerated gradient (NAG), RMSprop, Adagrad, and Adam [11].

- Stochastic gradient descent: stochastic gradient descent or SGD is the basic optimization algorithm for deep learning. It is derived from the batch gradient descent or Vanilla gradient descent algorithm to improve computation efficiency. Let θ be all the trainable parameters or the weights, batch gradient descent uses the entire training set to a single update of the weights:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.15}$$

where η is the learning rate, and $\nabla_\theta J(\theta)$ is the inverse of the objective function to compute the gradient using backpropagation. It is obvious that this algorithm is inefficient and expensive given the training set is big. In comparison, stochastic gradient descent uses a sample from the entire training set to perform update each time. Given the training set is divided into i samples, the weights are update by sample i is presented as:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.16}$$

Compared with the batch gradient descent, SGD just uses a small portion of the training data to update the weights. It performs frequent updates with much higher variance during the update with a heavily fluctuating way. SGD allows the function to jump to a new and better local minimum with the risk to complicate the convergence to the exact minima. Therefore, choosing the proper learning rate η is important to SGD but given a complex, non-convex function, it is likely to miss the

true minima value by such a fluctuating process. An improvement called mini-batch gradient descent is introduce:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \quad (2.17)$$

where the update is performed by n training example from a mini batch. The mini-batch gradient descent algorithm reduces the variance of each update and makes the convergence smooth and efficient.

In general, the SGD algorithms face many challenges. First, it is difficult to find the proper learning rate. Deep neural networks are not convex function. The objective function is to find the optimal local minima instead of the global minimum. A too small learning rate will lead to slow convergence, while a too large learning rate will let the function miss the minimum or even let the function diverge. Second, the learning rate cannot be a constant value given different stage of the training. The deep learning loss function is non-convex therefore it is vulnerable to some suboptimal local minima. The research by Dauphin et al. argues that it is more difficult to let the function to walk out from saddle points than from local minima when it reaches a point surrounded by a plateau of the same loss [12]. As a fixed learning rate is notorious to SGD, a learning schedule with a series of values is set before the optimization for different stage of learning when a threshold is triggered. However, a pre-defined set of values is unable to adapt to all new dataset characteristics.

- Nesterov accelerated gradient (NAG): Since SGD is vulnerable to walk out from ravines where the surface curves in a particular direction than others. This is the typical situation on a function surface where the curve decreases more rapidly in one dimension than the others [13]. To solve this problem, we can add the momentum method to let the gradient partly keep its previous direction. Thus, the optimization algorithm can be speeded up in the proper directions with a gentle but consistent gradient. The gradient descent with momentum can be present as:

$$v_{t+1} = \mu v_t - \eta \Delta \ell(\theta) \quad (2.18)$$

The effect of the gradient (the last term on the right) is to increment the previous velocity. In a standard SGD, the optimization yields a very large gradient at the beginning, so the momentum is small. Once the gradient becomes very small or disappears, the learning parameters or weights are likely to be stuck in a ravine and the momentum can be smoothly raised to a large value (e.g., 0.9 or more). The SGD with momentum can help the learning continue to converge by crossing the ravine which can cause divergent oscillations with a standard SGD. However, the standard momentum method first computes the gradient at the current location and then takes a big jump in the direction of the updated accumulated gradient. To improve the SGD momentum method, a new method called gradient descent with Nesterov momentum is proposed to further correct the gradient vector direction.

$$v_{t+1} = \mu v_t - \eta \nabla \ell(\theta + \mu v_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1} \quad (2.19)$$

where the gradient is not only determined by the current parameter θ_t, but is also adjusted by a correction term. The NAG method helps to keep the gradient term in the right direction.

- Adaptive gradient algorithm (AdaGrad): Adaptive gradient algorithm or AdaGrad is a modified stochastic gradient descent algorithm that can adaptively change the learning rate for each parameter [14]. In general, the AdaGrad algorithm follows two principles: to increase the learning rate if the parameters are sparser, and to decrease the learning rate if the parameters are less sparse. By adapting the learning rate with this strategy, AdaGrad improves the learning convergence compared to standard SGD particularly when the data is sparse, and the sparse parameters contain more information. The AdaGrad algorithm sets a base learning rate η, and η is multiplied with the elements of a vector $\{G_{j,j}\}$. And $\{G_{j,j}\}$ is the diagonal of the outer produce matrix $\mathrm{G} = \sum_{\tau=1}^{t} g_\tau g_\tau^T$, where $g_\tau = \nabla Q_i(w)$ is the gradient at the

iteration τ. The diagonal is given by $G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2$. The weight is updated after each iteration based on both the gradient and the diagonal vector.

$$\mathrm{w} \coloneqq \mathrm{w} - \eta \mathrm{diag}(\mathrm{G})^{-\frac{1}{2}} \circ g \tag{2.20}$$

and the update of each weight is written as:

$$\mathrm{w}_j \coloneqq \mathrm{w}_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j \tag{2.21}$$

From the above formula, each $\{G_{j,j}\}$ yields a scaling factor for the learning rate that applies to a single weight w_i, $\sqrt{G_i} = \sqrt{\sum_{\tau=1}^{t} g_\tau^2}$ is the ℓ^2 norm of the previous gradients, so extreme weight updates will be suppressed, while small weight updates will be assigned to a higher learning rate.

- Root mean square propagation (RMSProp): Root Mean Square Propagation or RMSProp is another method to adaptively change the magnitude of the gradient during learning. If we optimize the neural network by the full batch learning form, we can measure the sign of the gradient by the resilient backpropagation or RProp algorithm, where the learning rate can be adjusted by observing the sign of the gradient, so that the learning can escapes from the plateaus with tiny gradients quickly. The drawback is that the weights are updated with the same magnitude. The learning is likely to diverge too early thus it results in underfitting. To overcome this limitation, we can adapt the learning rate by the root mean square propagation or RMSProp. RMSProp divides the learning rate for each of the parameters by evaluating the weight of each parameter by comparing with the root mean square (RMS) of the magnitudes of recent gradients. The parameters of the current step (*t*) are determined by the weights of last step and a forgetting factor:

$$v(w,t) \coloneqq \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2 \tag{2.22}$$

and the parameters are updated as:

$$w \coloneqq w - \frac{\eta}{\sqrt{v(w,t)}} \tag{2.23}$$

Therefore, RMSProp can effectively adapt the learning rate for both full-batch learning or mini-batch learning as it overcomes the limit of resilient backpropagation (or RProp) only by the sign of the gradient.

- Adaptive Moment Estimation (Adam): Adaptive moment estimation, or Adam, is an improved optimization method based on RMSProp proposed by Diederik Kingma and Jimmy Ba [15]. Adam uses both the averages of the gradients and the second moments of the gradients to estimate the learning rate for the next iteration. Given the learning weights $w^{(t)}$ and the loss function $L^{(t)}$ at the t iteration, the weight update by Adam is given by:

$$m_w^{(t+1)} \coloneqq \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}$$
$$v_w^{(t+1)} \coloneqq \beta_2 \mathrm{v}_w^{(t)} + (1-\beta_2)\left(\nabla_w L^{(t)}\right)^2 \quad (2.24)$$

where the estimate of gradient and the second moment is given by:

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-\beta_1^{t+1}} \text{ and } \hat{v}_w = \frac{v_w^{(t+1)}}{1-\beta_2^{t+1}} \quad (2.25)$$

And the update of the weight in the next iteration (t+1) is given by:

$$w^{(t+1)} \coloneqq w^{(t)} - \mu \frac{\hat{m}_w}{\sqrt{\hat{v}_w}+\epsilon} \quad (2.26)$$

where ϵ is a small scalar to prevent the denominator from becoming 0, and β_1 and β_2 are the forgetting factors respectively for the gradient vector and for the gradient vector of the second moments. The Adam optimizer is widely used for multiple learning tasks such as image learning and natural language learning.

Other optimization algorithms include natural gradient descent, Kalman-based stochastic gradient descent (kSGD) and second-order methods, etc. [9, 16], but they are mainly used for research purposes.

2.1.3 Convolutional Neural Network

Convolutional neural networks (CNN) are the deep learning architecture of image processing. The most important structure of CNN includes the convolutional layer and the pooling layer. A convolutional layer is the basic component of the CNN that mainly performs feature extraction by combining the linear operation (convolution) and nonlinear operation (activation function). A CNN can have one or many convolutional layers given different needs of the task. It uses local connectivity on the raw data to reduce the storage memory by decreasing the number of parameters [6]. For example, the 50 X 50 image has 2,500 pixels. When we apply the convolutional kernels (or filters) on the raw image, only the meaningful features are extracted as a collection of local pixel patches. In additional, the convolution operation can be applied to one-dimensional time series data as a collection of local signal segments. The one-dimensional convolution is presented below, where x is the input signal and w are the convolutional kernel.

$$C_{1d} = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.27}$$

For image processing, we use the two-dimensional convolution. The input X is a 2-D grid recording the value of pixels of the image and K as the convolutional kernel (or filter). Then we use one or more filters as smaller grids to scan the entire input to extract feature maps.

$$C_{2d} = \sum_m \sum_n X(m,n)K(i-m,j-n) \tag{2.28}$$

Note that the convolution operation also improves parameter sharing because all filters slide across the whole input with the same strike. The output of each convolution operation is the summation of the element-wise product of the filter and the sub-matrix of the input scanned by the filter. By repeating this procedure with different convolutional filter, the image patterns such as edges, contours, and corners are extracted by different filters to form the feature representation and to skip the irrelevant spatial noise such as the background image and the location of the relevant patterns.

A CNN usually connects the output of one or several convolution layers to a pooling layer. A pooling layer performs subsampling to aggregate the extracted features to reduce the network parameters. Figure 6 illustrates the classical CNN architecture

originated from AlexNet, the breakthrough of CNN-based architecture for image recognition in 2012 [17].

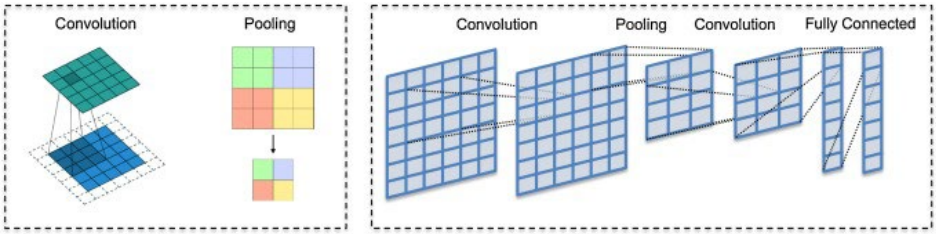


Figure 7: Convolutional Operation by a 3 X 3 Filter [5].

When comparing the CNN architecture in Figure 6 with the MLP in Figure 2.3, it is easy to find that the fully connected layers in a CNN have MLPs. On the top of the fully connected layers, CNN applies the convolution layer + pooling layer structure for automatic feature capturing. This CNN architecture lets the deep learning model capable of go through the whole machine learning pipeline from feature extraction to prediction / classification. In medical image, this full-stack deep learning model provide the "expertise-free" solution by saving the time and effort to hire medical specialist to extract learnable features for the downstream machine learning. Hence after CNN, many researchers believe the "hand-crafting" feature extraction step have become unnecessary and the machine learning process has turned to a totally data-driven manner in the age of deep learning.

- Convolutional layers in CNN: In image processing, convolutional layers are the main component for a convolutional neural network, or CNN. A convolution layer applies a series of filters to perform the summation of the products of the element-wise multiplication between a filter and the corresponding image patch over the image. Though the convolution technique is designed for two-dimensional input data, it can easily extend to the multi-dimensional data. If the convolution filter is designed to detect a particular type of image patterns such as edges and angles, the systematically use a serial of convolutional filters can effectively collect the common features of a special image patterns and then pipeline these features to

fully connected neural layers for multiple machine learning tasks such as classification, labeling, object detection, and image pattern segmentation, etc. Note that the convolutional layers are not only applied to the input data (e.g., two-dimensional gray scale image, or three-dimensional RGB color image), but they can also be applied to the activation output of the lower layers. The stacking of the convolutional layers helps to build a model for the hierarchical decomposition of the input image. The convolutional filters on the input raw pixels extract the low-level features, such as lines and edges, then the convolutional filters in the deep layers may extract and combine the lower-level features such as features that comprise multiple lines to express shapes. This cascading process continues until the complex image patterns are captured. The abstraction of features to high and higher orders as the depth of the CNN network is increased. In a particular convolutional layer l, the parameters include the convolutional filter size f, the padding p, and stride s that determines how far the convolutional filter should move, can the channel of the filter n_c. Given both the input and output are square matrices, the output dimension of is given by:

$$n^{(l)} = \left\lfloor \frac{n^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} + 1 \right\rfloor \tag{2.29}$$

An example of the convolutional layer operation by a 3x3 convolutional filter on a 9x9 image in a patch of the image is shown in Figure 2.9. Note that the there are two types of convolutions for image processing: valid convolution and same convolution. The valid convolution allows the first two dimensions of the matrices to reduce as the data passed through the convolutional layers. On the contrary, the same convolutional applied padding to keep the first two dimensions unchanged. The advantage of using same convolution is that it can simplify the computation and improve the effect of learning for a very deep CNN architecture.

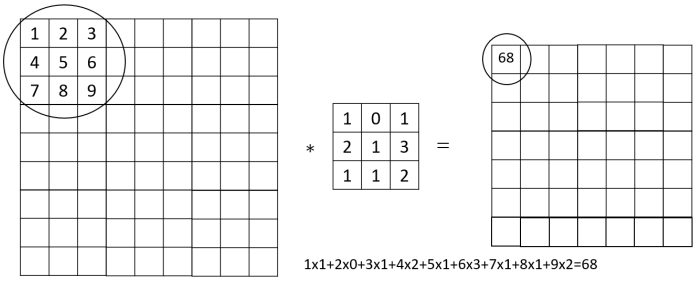


Figure 8: Convolution Operation on an Image Patch.

Figure 9 demonstrates the use of a 3-by-3 convolutional filter $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ to extract the vertical edges from a gray scale image, and another 3-by-3 convolutional filter $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ to extract the horizontal edges.

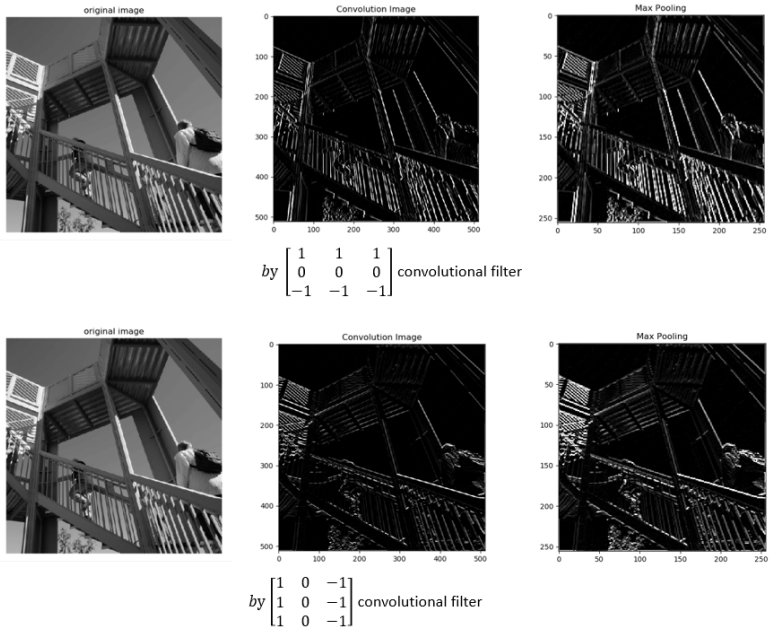


Figure 9: Extract Edges by Convolutional Layers.

- Pooling layers in CNN: The main function of using pooling layers is to perform a merging operation to condense the learnable features in a smaller region of the

matrices. There are two common pooling methods: maximum pooling and average pooling. Maximum pooling is used more frequently because it simplifies find the maximum value among the patch to same runtime. The hyperparameters of the pooling operation includes the filter size f and the stride s, and given the input is a square matrix, the dimension of the output can be computed by $\left\lfloor \frac{n^{(l-1)}-f^{(l)}}{s^{(l)}} + 1 \right\rfloor \times \left\lfloor \frac{n^{(l-1)}-f^{(l)}}{s^{(l)}} + 1 \right\rfloor \times n_c$.

Starting from the LeNet by Yann LeCun [18], the combination of the convolutional layers and pooling layers forms the basic structure to reduce the dimension of the feature maps that eventually leads to the desired dimension for different machine learning tasks. We will discuss further in the following section when different types of convolutional networks are introduced. Figure 10 illustrates the output from different convolutional layers of a CNN trained by eight epochs. The model is trained for classify the images of horses. By permutating the convolution layer + pooling layers blocks to form a CNN model, the output from the lower layers highlights the edges as visual patterns, and these patterns are decomposed and abstracted by the higher-level layers to form the discriminative features for the fully connected layers, then are pipelined to the final output for classification or regression.

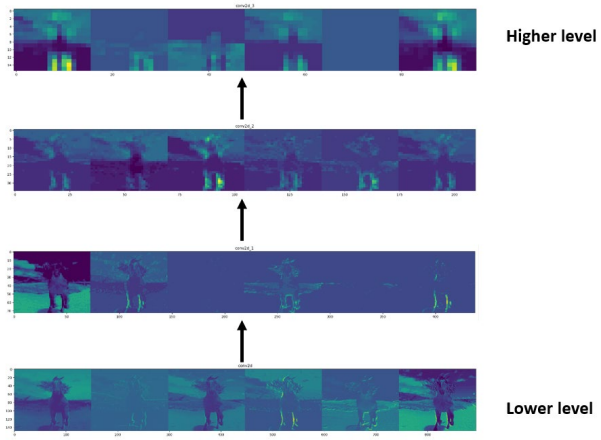


Figure 10: Outputs of Convolutional Layers.

- Activation functions in CNN: The activation function is a component of the artificial neural networks to define the output of a node, or a layer of the network given a set of inputs to the layer. For deep learning, we need to use nonlinear functions as activations for the outputs of deep neural networks, because given a linear function $\phi(v_i) = \mu v_i$, where μ is the slope, the node or layer attached by $\phi(v_i)$ is very difficult to fire if μ is negative. The common activation functions for classical neural networks are the sign function ($\mathrm{sign(x)}$), the logistic function ($\mathrm{sigmoid(x)}$), and the hyperbolic tangent function (tanh(x)). Their expressions and the corresponding derivative functions are listed below:

$$\mathrm{sign(x)} = \mathrm{f(x)} \coloneqq \begin{cases} -1, if\ x < 0 \\ 0, if\ x = 0 \\ 1, if\ x > 0 \end{cases}, or\ sign(x) = f(x) = \frac{|x|}{x}\ (x \neq 0), f'(x) = f(x) \quad (2.30)$$

$$\mathrm{sigmoid(x)} = f(x) = \frac{1}{1+e^{-x}},\ f'(x) = f(x)(1 - f(x)) \quad (2.31)$$

$$\tanh(\mathrm{x}) = \mathrm{f(x)} = \frac{(e^{x} - e^{-x})}{e^{x} + e^{-x}},\ f'(x) = 1 - f(x)^2 \quad (2.32)$$

In addition, the CNNs for image processing often uses new activation functions such as Rectified Linear Unit (ReLU) and Leaky ReLU, which are given by:

$$\mathrm{ReLU(x)} = \mathrm{f(x)} = \begin{cases} 0, for\ x \leq 0 \\ x, for\ x > 0 \end{cases}, f'(x) = \begin{cases} 0, for\ x \leq 0 \\ 1, for\ x > 0 \end{cases} \quad (2.33)$$

$$\text{Leaky ReLU(x)} = \mathrm{f(x)} = \begin{cases} 0.01x, for\ x < 0 \\ x, for\ x \geq 0 \end{cases}, f'(x) = \begin{cases} 0.01, for\ x < 0 \\ 1, for\ x \geq 0 \end{cases} \quad (2.34)$$

Compared to the classical activation functions (i.e., sign, sigmoid, tanh, etc.), the ReLU and LeakyReLU are convex functions with large ranges with non-zero derivatives, which are suitable for using the gradient based optimization algorithms. The computation of the gradient of the layers in a deep neural network using the chain rule needs serial of multiplications of partial derivatives. If the operation along the chain has many values smaller than one, the final gradient will decay rapidly when it goes down to the following layers. Therefore, the non-saturating derivatives play an important role in solving the numerical issues such as vanishing

gradients in very deep network architectures. Furthermore, the neurons in the middle of the network do not loose their interpretation as a classifier if zero is considered as the decision boundary. The universal approximation theorem still holds even for a single-layer network using ReLU as the activation function [9]. On the other hand, using ReLU as the activation has some disadvantages. The ReLU function is not differentiable over its entire domain. When the input x=0, a broken point is found where the gradient cannot yield a unique value. When the network is optimized, one important property of the function gradient is that it points to the direction of the steepest ascent, thus the function value can be minimized by following the opposite direction of the gradient. When the function is differentiable, the direction is unique. If we relax this constraint and allow the gradient point to multiple directions leading to an extremum, we need to apply the sub-gradient theory [10]. The sub-gradient theory allows us to continue using the gradient descent algorithms to optimize the network as far as we can determine a sub-gradient or find at least one instance pointing to the optimum. The ReLU function converts any value between 0 and -1, and x=0 to 0 as the function output for the descent operation, therefore the convergence is guaranteed for convex problems by using specific optimization programs like a fixed step size in the gradient descent [16]. This setting enables the running of the backpropagation for optimization with the non-differentiable functions as activations. The above activation functions are plotted on Figure 11.

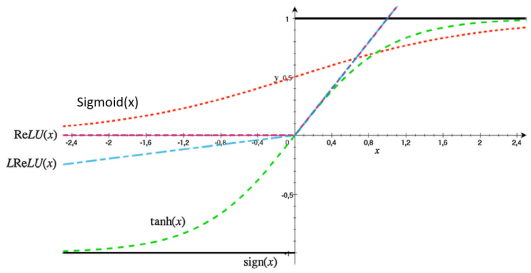


Figure 11: Activation Functions for Neural Network.

- One-by-one convolution: The pooling operation is the main method to down-sample the content of the feature maps for the CNN model architecture and to reduce the computation cost. Another problem for deep convolutional operations is to adjust the number of the feature maps by changing the depth of the network to reduce the runtime for the network computing. The one-by-one convolution is an ideal choice for channel-wise pooling, which decreases the number of feature maps whilst keeping their salient feature by manipulating the number of convolutional layers. For example, when a 9-by-9-by-256 feature map tensor is convolved with 64 1-by-1 convolutional filter, by the formula:

$$n^{(l)} = \left\lfloor \frac{n^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} + 1 \right\rfloor \tag{2.35}$$

We can calculate the output tensor dimension is 9-by-9-by-64, thus the channels of the feature map are reduced from 256 to 64, and the number of parameters is reduced from $9 \times 9 \times 256 = 20{,}736$ to $(9 + 9 + 1) \times 64 = 5{,}248$. (Figure 12)

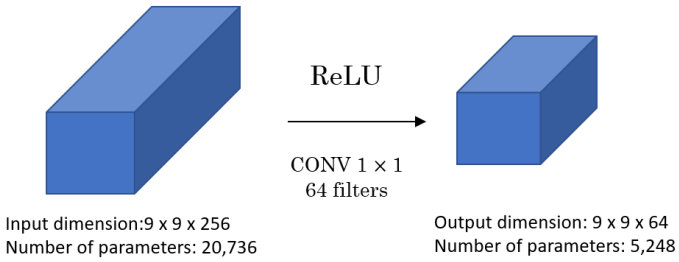


Figure 12: One-by-one Convolution.

The above one-by-one convolutional effectively reduces the computation complexity of the deep neural network, which is a common challenge to fine tune a CNN from high performance on prediction [19]. To learning complex features for sophisticated real-world problems, we need a CNN with high architecture. The depth of the input or the number of filters in the convolutional layers often increases the depth of the neural network thus leads to a dramatical growth of the number of

feature maps at the higher layers of the network. As the neural network becomes deeper, the deep feature channels make it difficult to optimize. Furthermore, the advanced network architectures such as the inception network [20] and the residual network [21] uses the one-by-one convolutional operation to concatenate the output feature maps from multiple convolutional layers, which will be discussed in the coming section.

2.1.4 Classical Architecture of Convolutional Neural Network

- LeNet: There are many classic architectures for building deep convolutional neural network architecture (CNN). CNN for image pattern learning can be traced back to the LeNet proposed by LeCun, Yann [22] for hand-written digit recognition. The LeNet is composed of five hidden layers with two convolutional + pooling modules and two fully connected layer respectively with 120 nodes and 84 nodes for abstract patterns learning. The LeNet uses a softmax layer to predict the probability of each possible digit by the activation output from the fully connected layer. The LeNet architecture is shown in Figure 13.

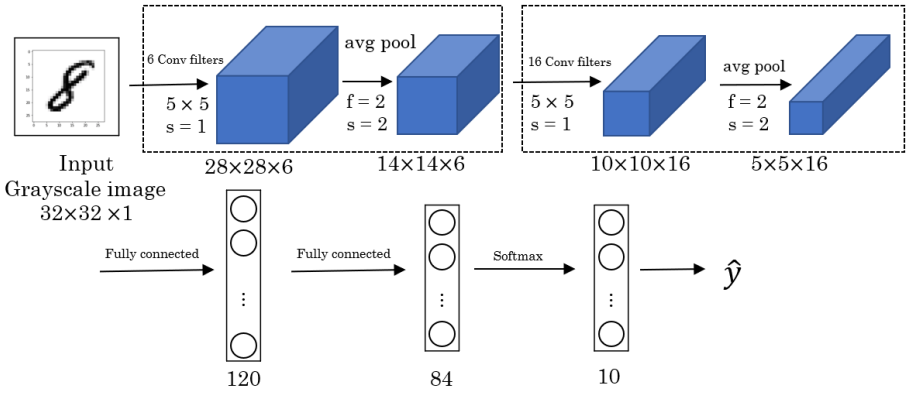


Figure 13: Architecture of LeNet [23].

- Alex Net: The breakthrough of CNN for image pattern classification started from Alex Net by Alex Krizhevsky et al. in the 2012 ImageNet Large Scale Visual Recognition Challenge (LSVRC-2012) [17]. ImageNet is an image dataset

containing 14 million images [23]. The images have been hand-annotated with labels regarding the classes of the objects in the images and the corresponding bounding boxes to the relevant pixels of the objects. It contains more than 20,000 classes of image. Each class has several hundred images. The ImageNet Large Scale Visual Recognition Challenge began in 2010. And in 2012, the Alex Net proposed by Alex Krizhevsky from University of Toronto achieved the top-5 classification accuracy of 84.7% (or top-5 error of 15.3%) compared to the accuracy of 73.8% achieved by the second-best entry [17]. The high classification accuracy makes the CNN-based network become the-state-of-the-art artificial intelligence technology for computer image processing.

The architecture of Alex Net follows the convolutional layer design pattern of LeNet, where a convolutional layer is followed by a maximum pooling layer to from a block to build the feature maps. In addition, Alex Net also uses the same convolution technique, where zero paddings are added to the outputs of the convolution to maintain the original height and width of the tensor. The same convolution simplifies the CNN network computation. In the forward propagation, we keep the first two dimension unchanged and simply add more channels to the output tensor, which helps to expedite the computation for both forward and back propagation. The architecture of AlexNet is shown in Figure 14.

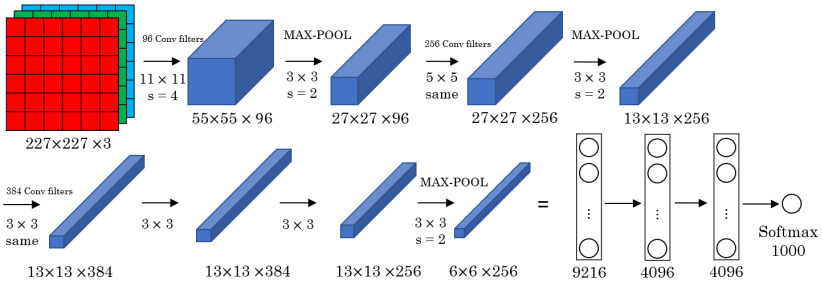


Figure 14: Architecture of Alex Net [17].

Besides AlexNet, a similar CNN architecture call VGG (Visual Geometry Group) Net is also proposed by Karen Simonyan and Andrew Zisserman from

Oxford University in England in 2015 [24]. The VGG Net applies similar design strategy as Alex Net, and it also unifies the configuration for all convolutional layers and pooling layers. For the convolutional filters, VGG Net uses 3-by-3 convolutional filters with stride=1 and same padding. For the pooling, it uses 2-by-2 filters with stride=2. (Figure 15) The advantage of the VGG Net architecture is that it further simplified the CNN design by filters with the same size. According to the report, the performance of VGG Net on the ImageNet dataset is superior to Alex Net with the top-5 error of 7.5% ~ 7.3% [24]. The success of VGG Net let the CNN become better with less learning parameters leading to lower computation cost. Another improvement in VGG Net is that it uses the one-by-one convolutional filters to reduce the depth of the tensor such that the network computation can be simplified (Figure 15).

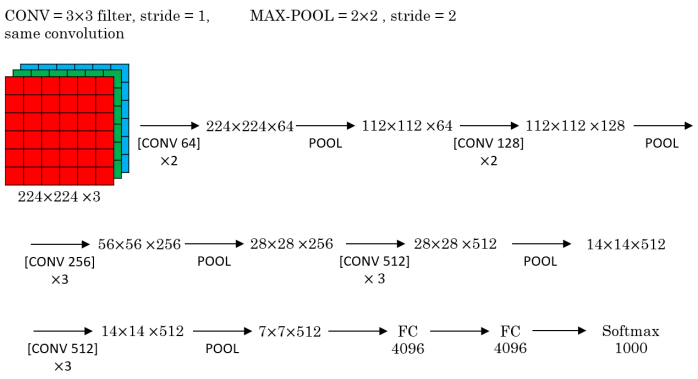


Figure 15: Architecture of VGG-16 [24].

- Inception Network: From LeNet to Alex Net then to VGG Net, the performance of the CNN network improves as the network architecture becomes deeper, i.e., the more convolutional layers with more filters are attached to the CNN, the higher performance will be expected. On the other hand, with the support from parallel computing and GPU, the hardware improvement makes the computing for very deep neural network less expensive and convenient both in the view of economy and technology. Under this background, Christian Szegedy et al. proposed the Google Net architecture in 2015 based on the inception network architecture [20].

The inception network uses a special inception block that is composed of convolutional layers with different filter sizes by the same convolution format with paddings to maintain the identical first two dimensions (i.e., height and width) of tensors going through the network. In addition, the inception block also uses the one-by-one convolution to manipulate the channels of the tensor so that not only can the shape of the tensors in different layers be effectively controlled, but also the computation cost of the network can be constrained in the acceptable scale. The inception block ends with a special channel concatenating layer, which concatenates the activation outputs of all convolutional layers in various sizes and passes the output to the next inception block. Another advantage of the inception architecture is that it can be attached with multiple Softmax layers to yield the CNN classification outputs given different tasks. This feature gives the inception network the flexibility to perform multiple labeling for the images of more than one interested pattern for various tasks such as object detection, motion detection, image segmentation, etc. The general inception network architecture is illustrated in Figure 16.

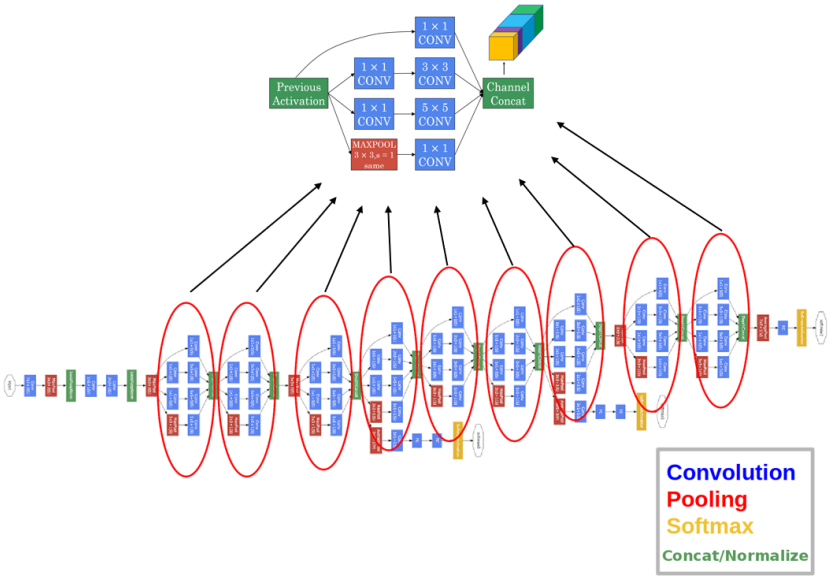


Figure 16: Architecture of Inception Network [20].

- Residual Network: The residual network is proposed by He K. et al. in 2015 to enable to train the very deep neural network [21]. As the neural network becomes very deep (e.g., more than 30 to 50 layers), the gradient flow across the layers becomes numerically unstable. Therefore, the training of a very deep neural networking is easy to diverge, which results in under-fitting. To alleviate the rapid decay of the gradient flow, the residual network adds a residual block to add extract input from more than one previous layer (Figure 17). Given the current layer l, the activation output a[l+1] of sequential network is presented as:

$$a^{[l+1]} = g(z^{[l+1]}) = g(W^{[l+1]}a^{[l]} + b^{[l+1]}) \quad (2.36)$$

where W is the model parameters, $a^{[l]}$ is the activation output of the previous layer (l-1), and b is the training bias. In a residual block, the output contains not only the output from the previous layer l-1, but also contains the output from the two previous layers, i.e., l-1 and l-2. So, the activation output $a^{[l+1]}$ for a residual block is revised as:

$$a^{[l+1]} = g(z^{[l+1]} + a^{l-1}) = g(W^{[l+1]}a^{[l]} + b^{[l+1]}) + g\left(W^{l}a^{[l-1]} + b^{[l]}\right) \quad (2.37)$$

The advantage of the residual architecture is that it parallelizes the outputs of two previous layers to improve the gradient flow for optimizing the whole network when running backpropagation. According to the research by He K. et. al, they improved the top-5 error from 7.89% by Google Net to 4.49% by a 152-layer ResNet [21]. The drawback of the deep residual network is that it enormously increases the computation cost of network optimization, which makes the use of GPU and cluster computing necessary to a neural network from scratch. However, the ResNet can accommodate a huge number of convolutional filters throughout its deep structure, thus it provides a handy solution for transfer learning. (Figure 17)

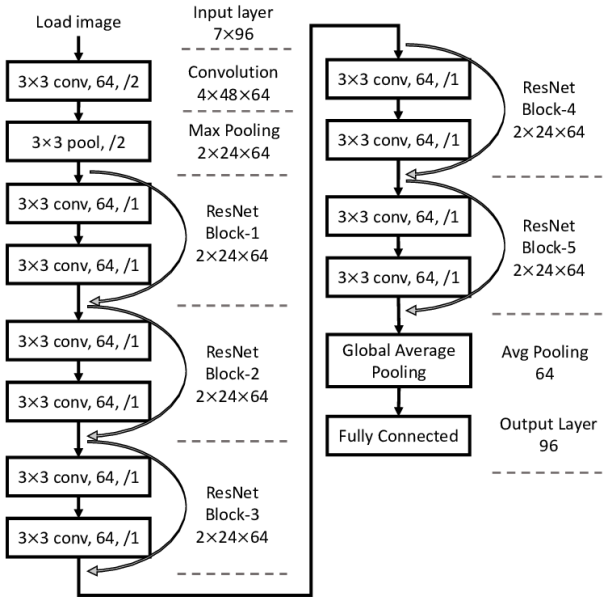


Figure 17: Architecture of Residual Network [21].

- Autoencoder: Autoencoder uses a semi-supervised learning method to find the representations of the input from a lower dimensionality [25]. The autoencoder model first learns the features from the raw data input. By this strategy, it does not need to annotation of the training dataset, but to use the unsupervised learning (e.g., clustering) to add labels to the training data. Then the network is trained to make the prediction based on the result of unsupervised learning by optimizing the loss function such as $\mathcal{L}(\theta) = \left\| \hat{f}(x) - x \right\|_2^2$. The ideal of autoencoder is successfully applied to CNN for motion detection [25-27].
- Generative Adversarial Network: Generative adversarial network, or GAN, is introduced by Ian Goodfellow et al. in 2014 [7]. It applies two neural networks to learn a representative distribution from the training dataset in competition with each other. The first network is called generative network or the generator (*G*) to generate the new data from a noise input. Given a training set *X* (e.g., images, documents, etc.), the generator *G(X)* takes the random noise input to produce fake data similar to those in the training set (real data) and pipeline the fake data to the second network, the discriminator (*D*). The task of the discriminator is to distinguish

the real data *X* against the fake data *G(X)* by the generator. In an ideal situation, the discriminator needs to learn the patterns from the real dataset X and not to be cheated by the generator, while the generator attempts to learn the true distribution of the training set *X* to provide similar fake data close to the true distribution. The two networks follow the game theory to compete and gradually improve their own capacity (Figure 18).

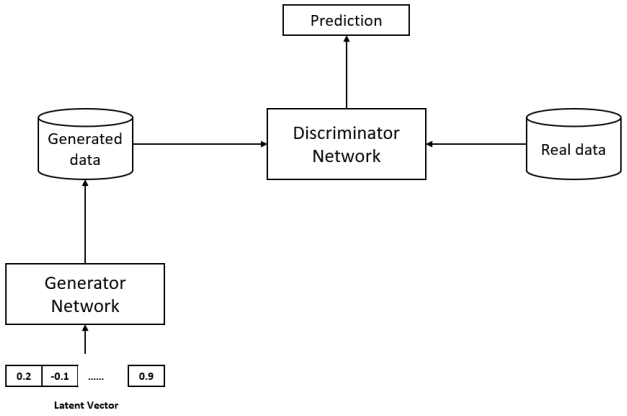


Figure 18: Architecture of GAN.

The well-trained GANs can render plausible and real-looking images, which is a potential approach to improve deep learning for medical images which the sample images are usually inaccessible [28]. In addition, a conditional GAN allows to encode particular patterns in the training process such that the images with desired properties can be generated [29]. Cycle GANs improve the GAN data generation to a particular domain, so that the newly generated data G(X) does not need to correspond to the images in the training set [30].

- Recurrent Neural Network: Recurrent neural network, or RNN, is the neural network to process data sequences with long term dependencies [31]. The recurrent networks introduce state variables that allow the cells to carry memory and essentially model any finite state machine. RNN is a multilayer perceptron (MLP) network connected by many recurrent loops to add feedback and memory to the

networks over time. It learns and generalizes across sequences of inputs rather than individual patterns. The recurrent loops hold the memory and allow an RNN to learn and generalize across sequences of inputs rather than individual patterns. An RNN uses the same set of weights at all time steps. In a specific hidden layer at step t, the hypothesis value is estimated by:

$$h_t = \sigma(W^{(hh)} h_{t-1} + W^{(hx)} x_{|t|}) \tag{2.38}$$

where $W^{(hh)}$ is the weight at the previous stage and $W^{(hx)}$ is the weight at the current stage, and the probability of the next stage is estimated by:

$$\hat{P}\left(x_{t+1} = v_j \middle| x_t, \dots, x_1\right) = \hat{y}_{t,j} \tag{2.39}$$

The extensions of RNN include long-short-term memory (LSTM) networks [32] and gated recurrent units (GRU) [33], which can model the explicit memory reading and writing memory transactions like a computer.

- U-Net: U-net is a DNN architecture proposed by Olaf Ronneberger et al. for biomedical image segmentation in 2015 [34], and it currently becomes the common architecture for semantic segmentation. U-net is an enhanced fully convolutional network (or FCN) for semantic segmentation by J Long et al. in 2014 [35]. Unlike the conventional DNN using a sequential structure, U-net uses a U-shape architecture with an encoder network on one side and a decoder network on the other side, which are connected by a bridge component (Figure 19). In addition, the corresponding components of encoder side and the decoder side with the same output dimension have the skip connections. The skip connections provide extra information to the decoder to generate better semantic outputs. They also act as shortcuts to produce indirect flow of gradients to the lower layers in the backpropagation for network optimization thus help the whole architecture to learn better feature representation. The U-net architecture can be easily adjusted to fit different input and output requirements. Therefore, U-net becomes the common architecture for generative models like GAN. In our study, we mainly use the U-net architecture to implement the generator of the proposed Ad Cycle GAN model.

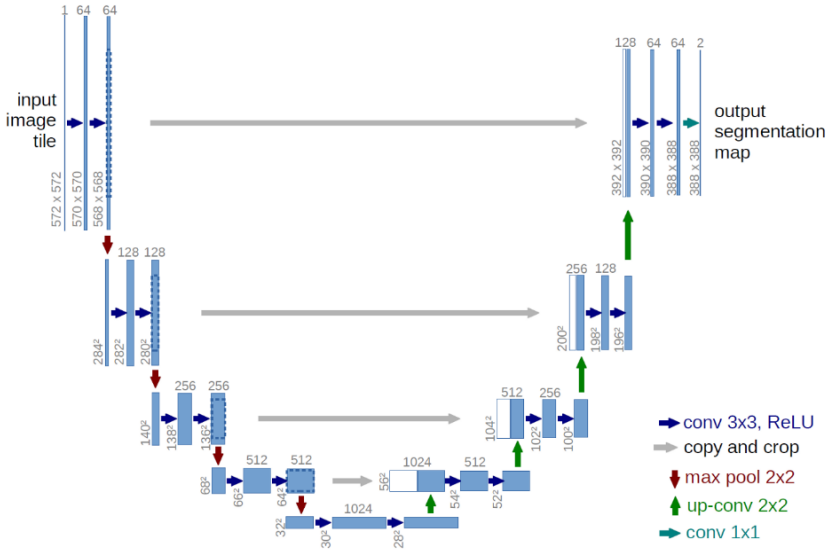


Figure 19: Architecture of U-net [36].

- PatchGAN: PatchGAN is a DNN model introduced as a discriminator component for the Pix-2-Pix image translation GAN model 2016 [36]. Instead of computing a scalar as the output of the DNN, PatchGAN computes a 2D matrix as the output. The advantage of using PatchGAN as the discriminator is that this model produces the prediction on the local image patches instead of a single prediction for the whole images. Therefore, when it is optimized as the discriminator adversarially with generators, the more local details of multiple image patches improve the learning procedure leading to better results. Note the PatchGAN is usually combined with the residual network [21], thus the outputs of the features from the patches provide more gradients for backpropagation optimization. In our study, the PatchGAN consisting of residual modules is used as the basic structure of the discriminator.

2.1.5 Advanced Techniques of Neural Networks

In addition to the network architecture, there are some useful concepts and techniques to improve the robustness and performance of deep neural networks. Some of them are listed below.

- Data Augmentation: Data augmentation is a good technique to add extra variations to the training set given the common features of the sample data. The typical variation includes noise, changes in contrast, rotations, and translations etc. This technique can increase the number of data points in the training set, which is particularly helpful for infrequent observations. In medical image processing, it is commonly difficult to acquire a large dataset with well-annotated labels. For example, Ronneberger et al. successfully applied the U-net with augmented data for classification and segmentation on a non-rigid deformation image dataset [34]. Recently, the new GANs are powerful to generated highly similarly data to improve the CNN performance on rear image datasets such as radiological images and histological images [37].
- Precision Learning: Precision learning is a machine learning strategy that includes known operators in the learning process [38]. It is not a conventional learning strategy which aims to learn the optimal representation. However, this approach is useful for signal processing in which a prior distribution is known, and an operator must be applied to the data processing pipeline. When the operator is embedded in the network architecture, not only is the training error further reduced, but also the required number of training samples can be decreased. It is particularly useful for data-greedy machine learning algorithms like the deep neural networks. This technique effectively extends the use range of deep learning for most signal processing tasks given the gradient or sub-gradient is kept in the training.
- Adversarial Examples: The adversarial examples can be generated by a well-trained GAN network. The adversarial examples can fix some weak spots which can by exploited by an attacker to lower the model performance [39]. In general, an attacker finds a perturbation e such that the network prediction $\hat{f}(x+e)$ is diverted to a different class other than the true y. while keeping the magnitude of e low. e.g., minimizing $\|e\|_2^2$, and the attack is related to the type of the objective function. The

attack either generates noise that misleads the training the deep model, or to exert extra weights to the patterns guiding the model to the wrong direction [40].

- GAN Evaluation: Unlike other deep learning neural networks which are trained with a loss function until convergence, a GAN generator model is trained using a discriminator that learns to classify images as real or generated. Both the generator and the discriminator model are trained together to maintain an equilibrium. Therefore, there is no objective loss function used to train the generator, nor a way to objectively assess the progress of the training or the quality of the model from the loss. Instead, the GAN model uses a set of qualitative and quantitative methods to evaluate the performance of the GAN models based on the quality and diversity of the generated synthetic images [41]. The qualitative measures include rapid scene categorization by human judgement and nearest neighbors. And there are many quantitative methods developed by different researchers: average log-likelihood, coverage metric, inception score (IS), modified inception score (m-IS), mode score, AM score, Frechet inception distance (FID), maximum mean discrepancy (MMD), the Wasserstein critic, birthday paradox test, classifier two-sample tests (C2ST), classification performance, boundary distortion, number of statistically-different bins (NDB), image retrieval performance, generative adversarial metric (GAM), tournament win rate and skill rating, normalized relative discriminative score (NRDS), adversarial accuracy and adversarial divergence, geometry score, reconstruction error, image quality measure, and low-level image statistics.

  In the original GAN paper by Goodfellow et al., they used the average Log-likelihood method, which was referred as kernel estimation or Parzen density estimation, to summarize the quality of the generated images [7]. This method estimates how well the generator captures the probability distribution of images in the domain. This method is eventually found not to be effective for GAN evaluation [41]. Alternatively, there are two common methods for GAN evaluation: inception score (IS), and Frechet inception distance (FID).

Inception score (IS) is proposed by Tim Salimans, et al. in 2016. It is an objective metric for evaluating the quality of generated images, specifically synthetic images output by GAN models [42]. The IS score uses the pretrained Inception v3 model to classify the generated images and calculate the probability of the prediction for each class. The outputs are conditional probabilities given the generated images. Images that are classified strongly as one class over all other classes indicate a high quality. The conditional probability of all generated images in the collection should have a low entropy. The entropy is calculated as the negative sum of each observed probability multiplied by the log of the probability. The intuition here is that large probabilities have less information than small probabilities [43].

$$entropy = -\sum p_i \times \log(p_i) \tag{2.40}$$

The conditional probability captures the interest in image quality. The marginal probability is applied to represent the probability distribution of all generated images. If generating various images is preferred, we use the marginal integral $p(y|x = G(z))dz$ as the entropy. These elements are combined by calculating the Kullback-Leibler divergence, or KL divergence: $KL(C||M)$. It calculates the divergence between two distributions C and M, where C is the conditional distribution and M is the marginal distribution.

$$KL\ divergence = p(y|x) \times (\log(p(y|x)) - \log(p(y))) \tag{2.41}$$

where $p(y|x)$ is the conditional probability for each image and $p(y)$ is the marginal probability. We can summate the KL divergence over all images and average over all classes. Then the exponent of the result is calculated to give the final score.

Another commonly used metric is the Frechet inception distance, or FID. FID is a metric that calculates the distance between feature vectors calculated for real and generated images. The score summarizes how similar the two groups are in terms of statistics on computer vision features of the raw images calculated using a pretrained image classifier. A Lower score indicates the two groups of images are

more similar or have more similar statistics. A 0 score indicates the two groups of images are identical. The FID score was proposed and used by Martin Heusel, et al., which is considered as an improvement of IS. The FID algorithm summates the activations of the coding layer of the pretrained network as a multivariate Gaussian based on the mean and covariance of the images. The outputs are used to compute the activations across the collection of real and general images. The Frechet distance (or Wasserstein-2 distance) represents the distance between the two distributions. A lower FID indicates better-quality images; conversely, a higher score indicates a lower-quality image, and the relationship may be linear. The FID score is computed with the formula below:

$$d^2 = \|mu_1 - mu_2\|^2 + Tr(C_1 + C_2 - 2 \times \sqrt{C_1 \times C_2}) \quad (2.42)$$

where d^2 is the FID score, mu_1 and mu_2 refer to the feature-wise means of the real and generated images. C_1 and C_2 are the covariance matrix for the real and generated feature vectors.

The above discussion indicates that the DNNs currently offer many techniques for different problems on medical image processing, particularly the perceptual tasks. In the upcoming section, we will discuss the tasks of deep learning for medical imaging.

2.2 Tasks of Deep Learning for Medical Imaging

In this section, we will discuss the tasks of deep learning in medical imaging, which includes image pattern detection and recognition, image segmentation, image registration, computer-aided diagnosis, and image reconstruction.

2.2.1 Image Pattern Detection and Recognition

Image detection and recognition relates to the problem about detecting a certain pattern in a medical image. The conventional methods need a complex preprocessing pipeline to parse the volumetric patterns from the images before passing to the classifier. For example, a popular method before deep learning is marginal space learning by a

probabilistic boosting tree model [44]. The introduction of the deep neural makes this procedure more efficient for the convolutional layer of a neural network model can boost the cascade and simplify the model architecture.

The current commonly used CNN-based algorithms for image pattern detection and recognition include the region-based convolutional neural networks (R-CNN) and the YOLO (You Only Look Once). The R-CNN is a family of techniques for addressing object localization and recognition. It was proposed by Ross Girshick in 2014, which covers R-CNN, Fast R-CNN, and Faster-RCNN designed for object localization and object recognition [45]. The original R-CNN model is comprised of three modules:

- The region proposal is used to generate and extract category-independent region proposals. e.g., the candidate bounding boxes.
- The feature extractor applies a deep convolutional neural network to extract features from each candidate region.
- The classifier is used to predict the features as one of the known classes.

A computer vision technique is used to propose candidate regions or bounding boxes of potential objects in the image called selective search. The feature extractor used by the model is a pretrained AlexNet CNN. When the input image passes through the pretrained CNN model, it generates a feature map in a 4,096 vector that is fed to an SVM classifier for the final classification. The R-CNN is a straightforward application of CNN to object localization and recognition, but its cost for runtime is very high because the CNN-based feature extractor passes on each of the candidate regions generated by the region proposal algorithm. Thus later, the author further proposed the Fast R-CNN and Faster R-CNN to improve the original R-CNN performance. The Fast R-CNN is proposed as a single model instead of a pipeline to learn and output regions and classifications directly. The architecture of the model takes the photograph and a set of region proposals as input that are passed through a deep convolutional neural network. The output of the CNN is then interpreted by a fully connected layer then the model bifurcates into two outputs: one for the class prediction via a softmax layer, and another with a linear output for the bounding

box. The model is significantly faster to train and to make predictions, yet still requires a set of candidate regions to be proposed along with each input image [46,47]. For further improvement, the Faster R-CNN uses the architecture that accommodates both proposing and refining region proposals as part of the training process, referred as Region Proposal Network, or RPN. These regions are then used in concert with a Fast R-CNN model in a single model design. These improvements both reduce the number of region proposals and accelerate the test-time operation of the model to near real-time with then state-of-the-art performance.

Another popular deep learning-based algorithm for object recognition is the YOLO family, or You Only Look Once by Joseph Redmon in 2016 [48]. The YOLO approach involves a single neural network trained end-to-end that takes a photograph as input and predicts bounding boxes and class labels for each bounding box directly. The technique involves a single deep convolutional neural network that splits the input into a grid of cells and each cell directly predicts a bounding box and object classification. The output is a series of candidate bounding boxes that are consolidated into a final prediction by a post-processing step. After it was firstly introduced in 2016, there are three versions of the YOLO algorithm (i.e., YOLOv1, YOLOv2, and YOLOv3). YOLOv1 proposed the general architecture of the model. YOLOv2 refined the design and made use of predefined anchor boxes to improve bounding box proposal. Finally, YOLOv3 further refined the model architecture and training process. Although the accuracy of the models is close but not as good as Region-Based Convolutional Neural Networks (R-CNNs), they are popular for object detection because of their detection speed, often demonstrated in real-time on video or with camera feed input.

The above object detection and classification algorithms have been successfully applied many research domains of medical imaging. For example, in the study domain of radiology image processing, Bier et al. use the deep neural network model to detect the invariant anatomical landmark for pelvic trauma surgery. They conclude that the deep reinforcement learning model can more efficiently detect the anatomical landmarks than the conventional search process. The new method can detect hundreds of landmarks in a

complete CT volume in a few seconds [49]. In another work, they use the region proposal convolutional neural network (R-CNN) to robustly detect tumors in mammographic images [50].

The deep learning-based detection and recognition are also applied in many other fields of medical imaging, such as histology and endoscopy diagnosis. For example, Aubreville et al. applies guided spatial transformer networks that refine the detection before the actual classification. In their next study, they use the convolutional neural network model to automatically detect images containing motion artifacts in confocal laser-endoscopy images [51]. Another popular application filed is on the historical images. In digital pathology, the image data are usually generated with a certain staining and present significant challenges towards image pattern detections. The difficulties include background clutter, inhomogeneous intensity, touching or overlapping nuclei/cells, etc. [52-54].

2.2.2 Image Segmentation

The image segmentation technology is an extension of the image recognition and classification in that it is the process of partitioning the images into multiple segments based on the output predictions of the classifiers. It is the process to give a label to every region of interest (ROI) in the whole image such that the ROIs with the shared label have some common characteristics. The goal of image segmentation is to simplify and or change the representation of the whole image into smaller pieces of pattern given the knowledge learned and represented by the machine learning classifiers (e.g., the deep neural networks).

The image segmentation technology is greatly benefited from the recent development in deep learning. In the research of medical image segmentation, our purpose is to determine the outline of an organ, or some anatomical or histological structure as accurately as possible. Currently, the image segmentation is also dominated by convolutional neural networks (CNNs) because the CNN architecture is effective in capturing intricate structural patterns that is even difficult to well-trained experts. For

example, Holger Roth et al. proposed a CNN-based model called DeepOrgan for the segmentation of MRI images. They concluded that the DeepOrgan model is efficient and allows to detect organs robustly [55]. Florin Ghesu et al. propose a marginal space deep learning (MSDL) to build a processing pipeline to detect and segment for volumetric image parsing. The MSDL can perform anatomical pose estimation and boundary delineation together. It is more efficient because its probabilistic boosting trees are replaced using a neural network-based boosting cascade. Furthermore, the MSDL drives efficiency even further by replacing the search process by an artificial agent that follows anatomy to detect anatomical landmarks using deep reinforcement learning. The authors concluded that it could detect hundreds of landmarks in a complete CT volume in few seconds [56]. In other study by Moeskops et al. used a CNN model to perform MRI (Magnetic resonance imaging) image segmentation for different anatomical region of human brain [57]. Chen et al. applied a fully convolutional neural network to segment different organs from the dual energy computed tomography (DECT) data. By evaluation the segmentation performance on four abdominal organs (liver, spleen, left and right kidneys), the average Dice coefficient is 93% for the liver, 90% for the spleen, 91% for the right kidney and 89% for the left kidney [58]. Jeffrey Nirschl et al. used an AlexNet-based CNN to perform pixel-level segmentation for cardiac histopathologic tissues and proved that the deep learning model outperforms the random forest model [59].

Early in 2004, Middleton et al. did an experiment with a hybridized model of combined with the neural networks and the active contour models to segment MRI images. They found that the deep neural network can effectively improve other models as the feature extractor. This finding suggests that revising conventional segmentation methods and fusing the available algorithms with deep learning in the end-to-end manner is the promising way to the 2D MRI images [60]. Fu et al. follow a similar idea by mapping the Frangi's vesselness into a neural network. They demonstrate that their solution can adjust the convolution kernels in the first step of the algorithm towards the specific task of vessel segmentation in in ophthalmic fundus imaging [61].

In addition, another interesting class of segmentation algorithm is the use of recurrent networks (RNN) for medical image segmentation. Poudel et al. applies a series of recurrent fully convolutional neural networks for multi-slice MRI cardiac image segmentation [62]. Andermatt et al. reports a successful application of multi-dimensional gated recurrent units for the 3D brain MRI image segmentation [63].

2.2.3 Image Registration

While the perceptual tasks of image detection and classification have been receiving a lot of attention to the applications of deep learning, image registration has not seen this large boost yet. Fortunately, there are several promising studies found in the latest literature that implies there are some opportunities to use deep learning for image registration.

Image registration is the process to transform different sets of image data into an identical coordinate system [64]. In medical imaging, this process is often used as a preliminary step in other image processing applications. For example, it applies geometric transformations or local displacements to align medical images captured with different diagnostic modalities, such as MRI and SPECT to the reference image. Image registration enables people to compare common features in different images. For example, by designating different images captured by various diagnostic devices, the doctors can determine whether a tumor is visible in an MRI or SPECT image.

One typical problem in point-based registration is to find good feature descriptors that allow correct identification of corresponding points. Wu et al. proposed an unsupervised framework using autoencoder by deep neural network to mine useful features from the MRI images of brains [64]. Compared to the existing image registration methods, the deep learning framework provides an end-to-end solution for fast processing with relatively low development cost. Schaffert R. et al. improve the deep learning-based unsupervised method by using the registration metric as loss function for learning good feature representations for training [65]. Miao et al. used deep convolutional neural network (CNN) for both 2D and 3D radiological images by estimating the 3D pose directly from the 2D point features, and concluded that CNN can significantly improve the

robustness, capture range and computational runtime as an intensity-based method for image registration [66].

In addition, deep learning provides an effective solution for full volumetric registration. Yang et al. used a CNN-based deep learning model called the quicksilver algorithm to model a deformable registration (large deformation diffeomorphic metric mapping, LDDMM) and used a patch-wise prediction directly from the image appearance [67]. Another report by Liao et al. modeled the registration problem as a control problem. Their strategy is to use an agent and reinforcement learning for rigid registration of predicting the next optimal movement to align the new incoming image to the old ones [68]. This approach can also be applied to non-rigid registration with a statistical deformation model. In this use case, the actions are movements in the vector space of the deformation model. Obviously, agent-based approaches are applicable for point-based registration problems. Zhong et al. reported a study belonging to this type of application for intra-operative brain shift using imitation learning. This method can be easily further applied to the landmark detection of different brain regions in the MRI images [69].

2.2.4 Computer-aided Diagnosis

Computer-aided diagnosis, or CAD in medical imaging, refers to the type of computer systems that assist the medical doctors in the interpretation of medical images. The digital images generated by X-ray, MRI, and ultrasound diagnostics provide a massive volume of objective information for the radiologist or other medical professionals to make the accurate diagnosis. On the other hand, the medical experts must analyze and evaluate the whole information collection comprehensively in a short time. The CAD systems can efficiently process the large number of digital images for typical appearances. It can also highlight the conspicuous sections or regions, such as the possible diseases and historical trauma, as the concrete evidence for the clinical decision made by the medical professionals. CAD is regarded as one of the most challenging problems in the field of medical image processing and deep learning. The success of deep learning in many pattern recognition applications gives excellent expectation for CAD. In computer-aided

diagnosis, we use the technology of artificial intelligence (AI) not merely as a supportive role to acquire and quantify clinical findings and evidence to the diagnosis but also to predict the complete diagnosis. Therefore, the decision procedure must be rigorous, and the result be reliable. A typical CAD system is comprised of four stages: pre-processing, feature extraction, feature selection, and classification. The three promising domains of deep learning for CAD include ultrasonic diagnosis, radiological diagnosis (MRI, CT, PET and X-ray), and procedure management and prediction [70].

Radiology diagnosis is the most common application in Computer-aided diagnosis. Diamant et al. used a transfer learning CNN to analyze chest radiographs consists of a large amount of routine radiologist information to classify 6 different pathologies with robust ROC AUC close to 90% [71]. Antropova N. used a CNN with the MIP (maximum intensity projections) images as input to perform automated MRI diagnosis [72]. Other CAD applications reported by current literature includes the diagnosis for fatty liver [73], prostate cancer [74], dry eye [75], Alzheimer disease [76], and breast cancer [77]. Salam et al. used the hybrid features combined with color and texture of the optic disc and the optic cup as the feature map to detect glaucoma in fundus images [78]. The optic disc is localized by employing support vector machine trained using local features extracted from the vessels in the eye [79]. For the automatic diagnosis of Alzheimer disease, a hybrid pattern comprised of clinical finding and image features was used for the multi-class classification for different types of the disease. The method was tested and evaluated by the Alzheimer disease neuro-image initiative (ADNI) dataset with promising accuracy [80].

For ultrasound-based diagnosis, Zhang et al. proposed deep neural network called BIRADS-SSDL that integrates clinically approved breast lesion characteristics as inputs for the semi-supervised deep learning (SSDL) for the automatic diagnosis based on a small dataset of ultrasound images. The model classification accuracy reaches 94.23% and is higher than the conventional model for ultrasound image classification (84.38%) [81]. Another study on the CADx for breast lesion used convolutional neural network (CNN) to detect and then classify the lesion region of interests (ROI). A dataset comprised of 579

benign breast lesion images and 464 malignant breast lesion images are used to evaluate the CNN model performance with satisfactory results [82].

Positron-emission tomography, or PET, is a technology of nuclear medicine that uses functional imaging technique to observe the metabolic processes in the body. PET is a state-of-art technology for the aid to the diagnosis of disease based on digital images, thus it also becomes one promising domain for deep learning. For example, to predict the prognosis of oropharyngeal squamous cell carcinoma, Fujima et al. proposed a CNN model to differentiate FDG-PET images between the human papillomavirus (HPV) and negative oropharyngeal squamous cell carcinomas. An image dataset with 2160 FDG-PET images was used to evaluate the model with the overall accuracy of 83% [83].

The above examples show that deep learning has become the state-of-the-art solution for computer aided diagnosis (CAD) because its flexible architecture is very good at parsing complex tasks.

2.2.5 Medical Image Retrieval

Digital medical imaging is widespread in most every aspect of practice in hospitals. As the volume of medical image repositories is increasing, it is in urgent need for image retrieval technology to effectively manage and query of the large image databases. The development of an effective medical image retrieval system can aid the clinicians in browsing these large datasets in an efficient way. Algorithms for automatic analysis of medical images have been introduced by many researchers [84-86]. Content-based image retrieval, or CBIR, is the computer vision technology to search for digital images by analyzing the contents of the image rather than the metadata such as keywords, tags, or descriptions associated with the image. The term “content-based” refers to the color, shape, texture, or any other pattern that can be derived from the image. The strength of CBIR is that it does not rely on the annotation quality and completeness. In stead, the search by CBIR is based on the latent patterns of the images such as the similarity of the query images.

Content based medical image retrieval (CBMIR) systems offer an effective way to support clinical diagnosis and provide the basis for the treatment of various diseases [87]. In addition, CBMIR is an efficient management tool to access, manage, and extract the relevant information from the large collections of complex medical images. The conventional medical image retrieval technology based on textual information such as tags and manual annotation is inefficient and with high cost of manpower, expertise, and preprocessing time. In comparison, CBMIR with deep learning can automatically classify and retrieve images based on feature representations extracted from the medical images. The technology will support not only the healthcare decision process, but also facilitate the medical research and clinical studies in finding relevant information through the large medical image repositories.

Deep learning methods have been applied for CBMIR in recent studies after the breakthrough of deep learning methods in bridging the semantic gap between meaning and the image. In particularly, the convolutional neural network (CNN) is adapted for learning feature representations for different imaging modalities and body organs. In general, the 3D volumetric medical image is obtained consisting of a series of 2D slices acquired from the target body organ. For example, the digital radiological images from different body parts were divided into separate classes with respective body part label. In this way, the supervision is very weak and requires very less time for labeling, hence decreasing the annotation effort required in training phase. The annotation of the medical image usually requires high-cost expert knowledge, if we can train a series of CNN classifiers to categorize the medical images in the first phase, then the workload of the following retrieval tasks can be decreased. Anthimopoulos et al. introduced a CNN based system for classification of Interstitial Lung Diseases (ILDs). The study dataset consists of 7 classes including 6 different ILD patterns and a healthy tissue class. The CNN model achieved a classification performance of 85.5% in characterizing lungs patterns [88]. Van Tulder et al. used a Boltzmann machine-based network for lung computed tomography (CT) image analysis in the semi-supervised learning manner. The research presented two approaches for two datasets: the first one is for lung texture classification, and the second one is for

airway detection [89]. Jiji et al. introduced a CBIR system for skin lesion images using reduced feature vector, classification and regression tree [85]. To integrate the semantic information to image retrieval, Brahmi et al. used a Bag of Visual Word (BoVWs) method along with scale invariant feature transform (SIFT) for the diagnosis of Alzheimer disease (AD) [90]. Rahman et al. proposed a supervised learning framework for biomedical image retrieval, where the predicted class label from classifier for retrieval [86].

The above discussion indicates that the image retrieval technology is often integrated into the computer aided diagnosis (CAD), where the deep learning models are used for image pattern extraction in either the supervised or semi-supervised manner.

2.2.6 Physical Simulation

Simulation is a common technology for performance tuning or optimizing. In the medical domain, physical simulation is frequently used for the diagnosis and prognosis prediction of radiology and surgery. Physical simulation becomes a new field of deep learning to support the medical engineering modeling. Wu et al. proposed a deep learning-based method to perceive the physical object properties, which can be exploited in the gaming industry to compute realistically appearing physics engines [91]. Chu and Thuerey used a convolutional neural network as a data-driven model for real-time smoke stimulation [92]. Meister et al. applied brought deep learning to biomedical modeling, using a neural network to learn the underlying biomechanics and to predict vertex-wise accelerations of biophysics solvers [93].

In addition to the application for pure theoretical studies, researchers also started to use such methods for medical imaging. For example, Maier et al. trained a deep convolutional neural network to reproduce the output of Monte Carlo simulations for the deep scatter estimation of CT images [94]. Unberath et al. used a deep neural network called DeepDRR for the simulation of fluoroscopy and digital radiography from CT images. They found that the deep learning model can be generalize to unseen clinically acquired data without the need for re-training or domain adaptation [95]. Horger et al. used a deep neural network to learn different noise distributions to a certain direction.

They concluded that the deep learning is a good alternative sampling method for simulation [96].

Except for the above samples, physical processes have been investigated using deep learning. Maier et al. introduced precision learning with prior physical operator to simulate material decomposition. They used the X-ray material decomposition as an example, in which the deep learning-based model is incorporated with additional prior knowledge. The new method is found to be able to improve prediction quality with SSIM (Structural Similarity) values from 0.54 to 0.88 [38]. Han proposed to use a deep convolutional neural network (DCNN) with 27 convolutional layers to synthesize MRI images to CT images to simplify the clinical decision process for soft tissue imaging. The new method is evaluated by the mean absolute error (MAE) and is superior to the atlas-based methods [97]. Stimpel et al. went further to use MRI projection images to predict X-ray projections [98]. And Schiffers et al. reported that the application of cycle GANs can create appearing fluorescence images from fundus images in ophthalmology [99]. However, some study indicates that deep learning can produce undesired effect such as mapping drusen onto micro aneurysms. Cohen et al. reported that hallucinate features can be produced when they attempted to create radiological patterns for cancers by the modality-to-modality mapping [100]. Therefore, the simulation by deep learning approaches must be used carefully.

2.2.7 Image Reconstruction

Image reconstruction refers to using the iterative algorithms to reconstruct 2D and 3D images given certain imaging techniques. The reconstruction of an image from the acquired data is an inverse problem. Usually, it is not possible to exactly solve the inverse problem directly. Alternatively, given an algorithm can approximate the solution, it might cause visible reconstruction artifacts in the image. Iterative algorithms approach the correct solution using multiple iteration steps, which allows to obtain a better reconstruction at the cost of a higher computation time. Medical image reconstruction is one of the common domains for this technology. For example, in computed tomography

or CT, an image is reconstructed from projections of an object. The iterative reconstruction methods usually have a better, but more expensive alternation to the common filtered back projection (FBP) method [101].

The deep learning-based iterative reconstruction algorithm uses the convolutional neural network (CNN) to learn the visual patterns from training data and combines the patterns with the image formation model. This method gives faster and higher quality reconstructions, which has been applied to both CT and MRI reconstructions [102-103]. A recent review concludes that using the deep learning models can omit the actual problem of reconstruction and formulate the inverse as image-to-image transforms with different initialization techniques [104]. Zhang et al. used the deep learning model called DenseNet with the deconvolution layers (inverse operation of convolution) to reconstruct sparse-view CT images. The study found the DenseNet can produce better reconstructed image with higher structure similarity (SSIM) and much lower root mean square error (RMSE) compared to other reconstruction methods [105]. Another work by Kofler et al. used a convolution neural network named U-net to reconstruct sparse view computed tomography images by a cascade of U-nets and data consistency layers. The study found that this model produces superior visual results and better preserves the overall image structure and diagnostic details [106]. Zhu et al. introduce a semi-supervised learning model using a structure like an autoencoder to reduce the dimensions of raw data and reconstruct the domain knowledge for image reconstruction. The entire model is linked by a non-linear correlation model, which can be combined as a single network and be trained in an end-to-end manner. The entire model was tested by reconstructing 2D MRI and PET (Positron Emission Tomography) images and showed better performance compared to traditional approaches [107].

A challenge for applying deep learning to image reconstruction is that the data-driven model has the risk to produce undesired effects [108]. Therefore, it needs to integrate prior knowledge and operators to improve the overall learning outcome. For example, Ye et al. introduce the concept of deep convolutional framelets by adding a multi-scale transform into the encoder and decoder of the U-net like network. This design is believed to bring

consistent improvement to the CNN models [109]. Another technique is to use wavelets for the multi-scale transforms. One successful case by Kang et al. added the wavelets to the residual net-based model to reduce undesired artifacts [110]. In another case by Han et al., a U-net-based model was embedded with the wavelets to reduce the noise of the reconstructed images from sparse-view CT images [111].

From the above discussion, we are inspired that the variational networks derived from the classical deep neural network architecture are useful to tackle various type of image reconstruction tasks when combined with the iterative algorithms by minimizing an energy function step by step. With a limit number of iterations, we can use the deep learning framework to map almost all iterative reconstruction algorithms onto the deep neural networks. One impressive work by Hammernik et al. used a variational network for MRI image reconstruction. It concluded that the reconstructed images can well preserve both the natural appearance and the pathological patterns of the original MRI images [103]. Vishnevskiy et al. used the variational network model to reconstructed ultrasound images and found it was able to produce high-quality rapidly reconstructed ultrasound images [112]. Adler et al. moved further to use this type of neural network for the entire primal-dual reconstruction of CT images and found it can significantly reduce noise [102]. In the work by Würfl et al., they followed the idea of using prior operators to improve the reconstructed CT images [113,114]. They designed the neural network based on the classical filtered back-projection which can be retrained to improve the approximation of limited angle geometries. This is very difficult to be solved by classical analytic inversion methods. In addition, the following research revealed that this end-to-end design manner can intrinsically correct the errors produced by the filter discretization or initialization during the learning process [115]. Their later study showed that their proposed deep neural network with prior operator model is compatible with other methods, such as learning an additional de-streaking sparsifying transform [116]. Syben et al. went further to demonstrate this concept can be extended to precision learning and derived a neural network structure [117]. In the above work, the researchers assumed that an expensive matrix inverse is a circulant matrix. Thus, it can be replaced by a convolution operation.

This method led to the derivation of a previously unknown filtering, back-projection, re-projection-style rebinning algorithm that intrinsically suffers less from resolution loss than the traditional interpolation-based rebinning methods.

Since all the neural networks are prone to adversarial attacks when the trained networks are trained, Huang et al. demonstrated their research that a conventional model mixed with incorrect noise can distort the entire image, but the network still constructs visually pleasing results [108]. These reconstructed images with plausible artifacts cannot be easily identified by classical methods [114]. A possible solution for this issue is to use the precision learning paradigm and fix the network as much as possible, so that the network model can be analyzed with classical methods [118]. An alternation is the Bayesian deep learning, where the network output has two aspects: the reconstructed image, and a confidence map on the measure of the accuracy [119].

In summary, deep learning can contribute to the suppression of artifacts as mentioned in the work by Zhang et al. where the deep learning model was successful to reduce metal artifacts [120]. Another study by Bier et al. shows the deep learning-based method is feasible for motion tracking and it is a feasible solution for motion compensated reconstruction [120].

2.3 Generative Adversarial Learning for Medical Imaging

As mentioned above, the generative adversarial network, or GAN, is introduced by Ian Goodfellow et al. in 2014 [7]. It applies two neural network models to learn a representative distribution from the training dataset in competition with each other. The first network called the generative network, or the generator (G) is to generate the new data from a noise input. Then the generated fake data is pipelined to the second network, the discriminator (D) to distinguish the real data X against the fake data $G(X)$ by the generator.

The adversarial optimization scheme for the GAN models has gained great interest in both the academia domain and the industrial domain because of its potential applications for counteracting domain shift, and the effectiveness in generating new image samples. The GAN models have achieved state-of-the-art performance on many fields of computer

imaging tasks such as text-to-images synthesis [121], improving image resolution [122], and image-to-image-translation [30]. The original GAN model introduced by Goodfellow et al. in 2014 [7] is a generative model designed for directly drawing samples from the desired data distribution without the need to explicitly model the underlying probability density function. The input z to the generator (G) is pure random noise sampled from a prior distribution $p(z)$ chosen from a distribution such as Gaussian distribution or uniform distribution. The output of the generator (G) x_g is expected to have visual similarity with the real sample x_r, drawn from the real data distribution $p_r(x)$. We denote the non-linear mapping function learned by G parameterized by θ_g as $x_g = G(z; \theta_g)$. The input to the discriminator (D) is either a real or generated sample. The output $y_1$of D is a single value indicating the probability of the input being a real or fake sample. The mapping learned by D parametrized by θ_d is denoted as $y_1 = D(x; \theta_d)$. The generated samples form a distribution $p_g(x)$ which is desired to approximate $p_r(x)$ after the GAN model is successfully optimized. The objective of the discriminator (D) is to differentiate these two groups of images whereas the generator (G) is optimized to confuse the discriminator (D) as much as possible. Intuitively, G could be viewed as a forger trying to produce some quality counterfeit material, and D can be considered as the detective trying to find out the fake items. In other words, we can consider G will receive a reward from D based on whether the generated data is correct or not. The gradient information is back propagated from D to G, so G updates its parameters to yield a better output image in the next iteration to cheat D. Mathematically, the training goal of the discriminator (D) and the generator (G) can be presented as:

$$\text{Discriminator: } \mathcal{L}_D^{GAN} = max_D \mathbb{E}_{x_r \sim p_r(x)}[logD(x_r)] + \mathbb{E}_{x_g \sim p_g(x)}[\log(1 - D(x_g))]$$

$$\text{Generator: } \mathcal{L}_G^{GAN} = min_G \mathbb{E}_{x_g \sim p_g(x)}\left[\log(1 - D(x_g))\right] \quad (2.43)$$

From the above formula, D is simply a binary classifier with a maximum log likelihood objective. If the discriminator D is trained to optimality before the next update of the generator G, then minimizing $\mathcal{L}_G^{GAN}$ is proven to be equivalent to minimizing the

Jensen-Shannon (JS) divergence between $p_r(x)$ and $p_g(x)$ [30]. The expected outcome after training is that the samples formed by x_g should properly approximate the real data distribution $p_r(x)$ of the real input images.

2.3.1 Deep Convolutional Generative Adversarial Network

A deep convolutional generative adversarial network, or DCGAN, is a GAN model built based on two CNNs: a CNN-based generator and a CNN-based discriminator. The model is optimized by the competition of the generator and the discriminator following the game theory. The generator is updated by loss of the low-quality images it produces and those are classified as 'fake' by the discriminator. On the other hand, the discriminator is updated by the loss from the error of misclassifying the read images and the 'fake' images yielded by the generator. The optimization goal is that the generator attempts to fool the discriminator with plausible generated images after each update, and the discriminator tries to correctly distinguish the generated images by the generator from the real images acquired from the real train set. This optimization strategy follows the game theory and is considered the most promising method to overcome the current threshold of deep learning [123]. Johnson et al. applied a conditional DCGAN to predict artifact-free brain images from motion-corrupted MRI images. And finds that the images generated by the conditional GAN have improved image quality [124].

There are generally two application approaches of GANs in medical imaging. The first is focused on the generative aspect, which can help in exploring and discovering the underlying structure of training data and learning to generate new images. This property makes GANs very promising in coping with data scarcity and patient privacy. The second focuses on the discriminative aspect, where the discriminator D can be regarded as a learned prior for normal images so that it can be used as a regularizer or a detector when presented with abnormal images. Furthermore, the constraints in clinical environment such as radiation dose and patient comfort, the diagnostic quality of acquired medical images may be limited by noise and artifacts. In the last decade, we witness the change of the clinical paradigm in reconstruction methods changing from analytic to iterative and now

to machine learning based methods. These data-driven learning-based methods either learn to transfer raw sensory inputs directly to output images or serve as a post processing step for reducing image noise and removing artifacts. Most of the methods reviewed in this section are borrowed directly from the computer vision literature that formulate post-processing as an image-to-image translation problem where the conditioned inputs of cGANs are compromised in certain forms, such as low spatial resolution, noise contamination, under-sampling, or aliasing. One exception is for MRI images where the Fourier transform is used to incorporate the raw K-space data into the reconstruction. In the next sections, we will discuss different types of GAN models that are originated from the GAN by Goodfellow et al. in 2014 [7].

2.3.2 Conditional Generative Adversarial Network

The conditional generative adversarial network, or cGAN, is an extension of the basic GAN that involves the conditional generation of images by a generator model. Image generation can be conditional on a specific, which allows the cGAN model to render generated images of a given class [122]. GANs are effective at image synthesis, that is, generating new examples of images for a target dataset. There are two motivations for using of the class label information in a GAN model: to improve the GAN performance, and to generate to target images. Additional information that is correlated with the input images, such as class labels, can be used to improve the GAN, which can benefit with stable training, faster training, and the generation of better images. The information of the label can be used for the deliberate or targeted generation of the images of the given label. One of the limitations of a basic GAN is that it generates a random image from the domain. The relationship between the points in the latent space to the generated image is complex and cannot by precisely mapped. An alternative approach to control the generated image is adding constraints to both the generator and the discriminator such that the whole model is conditional to a particular class label. When the trained generator model is used as a standalone model to generate images in the domain, it only produces images of a given type or class. The idea of conditional GAN is usually combined with deep convolutional

GAN or DCGAN, which is referred to as cDCGAN. There are many methods to incorporate the class information into the GAN models. One of the best methods proposed by Denton et al. is to add an embedding layer on the top of the CNN architecture and then it is followed by a fully connected layer with a linear activation that scales the embedding to the size of the image before concatenating it in the model as an additional channel [125]. Note that a GAN can be conditioned not only on the class label as a class-conditional GAN, but it can also be conditioned on other inputs, such as an image, in the case where a GAN is used for image-to-image translation tasks.

2.3.3 Information Maximizing Generative Adversarial Network

The information maximizing GAN, or InfoGAN, is another extension to the GAN architecture. An InfoGAN introduces control variables that are automatically learned by the architecture and allow control over the generated image, such as style, thickness, and type of generated images. A classic GAN consists of two models: the discriminator and the generator. While the whole model is optimized, the discriminator and the generator compete in a zero-sum game such that convergence of the training process involves finding a balance between the generator's skill in generating convincing images and the discriminator's in being able to distinguish them. The generator model takes as input a random point from a latent space us and a series of random numbers from a Gaussian distribution. The generator applies a unique meaning to the points in the latent space via training and maps points to specific output synthetic images. This means that though the latent space is structured by the generator model, it has no control over the generated images.

The latent space is used to explored and generated images and to be compared to the learned mapping function that the generator was learned. One approach is to use the conditional generative adversarial network, or cGAN to alter the generation process with a conditioned layer (e.g., using a class representation). Alternatively, we can provide control variables as input to the generator along with the point in the latent space. Then the generator is trained with the control variables to influence specific properties of the

generated images. This method is called information maximizing generative adversarial network, or InfoGAN [125].

The InfoGAN is motivated by the desire to disentangle the properties of generated images. For example, the properties of generating a face can be disentangled and controlled, such as the shape of the face, hair color, hairstyle, and so on. Control variables are provided along with the noise as input to the generator and the model is trained via a mutual information loss function. Mutual information refers to the amount of information learned about one variable given another variable. In information theory, mutual information between X and Y, I(X; Y) measures the amount of information learned from knowledge of random variable Y about the other random variable X. The Mutual Information (MI) is the conditional entropy of the control variables *(c)* given the new image (created by the generator (G) from the noise (z) and the control variable (c) subtracted from the marginal entropy of the control variables (c):

$$\text{MI} = \text{Entropy}(\text{c}) - Entropy(c; G(z, c)) \tag{2.44}$$

The computation of the true mutual information is often intractable, but we can use a simplified method called variational information maximization where the entropy is kept constant. Thus, training the generator via mutual information is achieved using a new model, referred to as Q or the auxiliary model [126], where the model shares all the same weights as the discriminator model for interpreting an input image. Unlike the discriminator model that predicts whether the image is real or fake, the auxiliary model predicts the mutual information used to generate the image. Both the discriminator model and the auxiliary model are used to update the generator model. The discriminator improves the likelihood of generating images to confuse the discriminator model, and the auxiliary model improves the mutual information. The result is that the generator model is regularized via mutual information loss such that it captures salient properties of the generated images, and, in turn, it can be used to control the image generation process.

2.3.4 Auxiliary Classifier Generative Adversarial Network

The auxiliary classifier generative adversarial network, or AC-GAN, is a new implementation of the conditional model (cGAN) whose focus switch from the generator network to the discriminator network. The classic cGAN adds the class information to the generator as the input to constrain the image generation process. As a result, the trained generator model will generate images of a given specific type. The auxiliary classifier generative adversarial network (AC-GAN) was introduced by Augustus Odena et al. in 2017 [127]. Its generator uses similar mechanism of the classic GAN where a random vector from the latent space and the class label are provided as input. In addition, the discriminator of the AC-GAN renders two predictions: first, whether the given image is a real one or a generated fake image; and second, the class of the given image. Therefore, the AC-GAN architecture requires that the discriminator (predicting real or fake) and the auxiliary classifier (predicting class label) are considered as separated models but sharing the same set of weights of the network. In practice, the discriminator and auxiliary classifier can be merged into a single neural network model with two outputs: first, an output by a sigmoid activation to predict whether the input is a real or a generated image, second, an output by a softmax activation to the probability of each class. Since the discriminator and the auxiliary classifier have similar architecture, they can share the model weights with the model is trained. During the optimization process, the objective function of the discriminator has two parts: the log-likelihood of the correct source (*LS*), and the log-likelihood of the correct class (*LC*). Thus, the discriminator (D) is optimized to maximize *LS*+*LC* while the generator (*G*) is optimized to maximise LC-LS [127]. As a result, the generator learns a latent space representation that is independent of the class label, which is different from the classic cGAN that aims to embed the information of the class label to the input of the generator.

2.3.5 Semi-Supervised GAN

Semi-supervised learning refers to a machine learning problem where a predictive model is attained from a training set of few labeled examples and many unlabeled

examples. A common example is a classification predictive modeling problem in which there is a large dataset only with a small fraction of data examples having target labels. The model is mainly optimized by the small set of labeled examples but is somehow affected by the unlabeled examples in the whole dataset in order to bed generalized to new data in the future. The Semi-Supervised GAN, or SGAN, is an extension of the Generative Adversarial Network architecture for solving the semi-supervised learning problems by utilizing the additional unlabeled examples [42]. The discriminator (*D*) in a classic GAN predicts whether a given input image is real or fake (generated), which allows *D* to learn the features from the unlabeled image data. Thus, the discriminator can be used via transfer learning as a start point for the unlabeled data in the same dataset. This allows the model optimization process in the supervised learning manner to benefit from the unsupervised training of the GAN. In the SGAN, the discriminator network is updated to predict $N+1$ classes, where N is the number of the classes in the prediction problem and the extra one class label is added for a new fake class. The training of the SGAN involves the simultaneous optimization for both the unsupervised task and the supervised task [128]. The discriminator of the SGAN is trained in two modes: in the unsupervised training mode, the discriminator is trained in the same way as in the classic GAN to predict whether a new input is real or fake; in the supervised training mode, the discriminator is trained to predict the class label of real examples.

The optimization in the unsupervised mode allows the network to learn useful feature extraction capabilities from a large unlabeled dataset, whereas the training in the supervised mode allows the network to use the extracted features and apply class labels. The result of this dual training mode provides a model that can achieve state-of-the-art results with very few labeled examples. In the work by Odena et al., they compared the SGAN model with a conventional CNN on the MNIST dataset from a series of examples from 1,000 labeled images to 25 labeled images only. The result shows that the SGAN has stable performance even with 25 labeled images, and the performance of the CNN drops significantly when the training set contains 25 images [128], the similar conclusion was also found in the research by Salimans et al. [42].

2.3.6 GAN Optimization

The basic GAN architecture introduced by Goodfellow et al. in 2014 contains two neural networks respectively called generator and discriminator. The optimization of the GAN architecture is considered as a zero-sum game regarding the competing against these two networks. During the optimization procedure, the objective of the generator is to learn mapping from a random distribution to a particular domain belonging to the distribution of the training dataset. On the other hand, the discriminator learns to classify the real data samples from the fake ones generated by the generator. When we apply the optimization algorithms such as stochastic gradient descent, the generator G aims to minimize the objective function meanwhile the discriminator D aims to maximize the objective function:

$$G^*, D^* = \arg min_G \, max_D \mathcal{L}(G, D) = E_x[\log(D(x))] + E_z[\log(1 - D(G(z))] \quad (2.45)$$

Formula 2.45 is the original GAN loss function proposed by Goodfellow et al. [7], which can be further divided into the generator loss and the discriminator loss during model optimization.

$$Generator\ loss: \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))$$

$$Discriminator\ loss: \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [\log D(x^{(i)}) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)] \quad (2.46)$$

The initial GAN has some weakness points during the optimization by the original loss objective. The first one is mode collapse, which means the generator keep producing monotonous outputs in stead of diverse generated examples representing the whole distribution of the target domain. This phenomenon is like overfitting in common machine learning problems where the generator is stuck in some local minimal, but it succeeds in confusing the discriminator. The second one is vanishing gradients when the discriminator outperforms the generator, leading to generator saturation and eventually causing gradient vanishing. In this case, the adversarial equilibrium cannot be maintained and the whole GAN cannot be further optimized. This causes the convergence of the GAN training where both the generator and the discriminator become stabilized and keep producing consistent outputs.

There are different strategies to solve the above problems in GAN optimization. For example, we can revise the loss function for smoother and more stable optimization. Wasserstein GAN or WGAN is one of the most successful examples for this pathway [28]. WGAN uses a linear function to compute the continuous Wasserstein distance as the loss for the discriminator instead of estimating the probability between 0 to 1 for telling whether the input data is true or fake. Thus, the discriminator is also called the critique to rate the quality of the generated data example. The optimization objective is to maximize the output difference between the critique and the generator. Another strategy to improve the GAN optimization is using conditional GAN or cGAN architecture, where the labels of the data examples are encoded as part of the training inputs. cGAN is an effective way to exert controls to the outputs of GAN generator with the labels as auxiliary information for the optimization. There are multiple variations for the cGAN implementation as we discussed above. In practice, we can add various labels either by directly encoding into the data and the GAN components, or by adjusting the GAN architecture by building multiple generators and discriminator for different modalities inside the GAN.

2.4 Summary

In this chapter, we go through the theory and concepts of deep neural network for artificial intelligence. Then we introduced the types of deep neural network and their application to various medical imaging tasks. In particular, the deep learning-based algorithms are successful to solve many tasks such as object detection, image recognition, and medical pattern segmentation. All these tasks are clearly linked to perception without any presenting of prior knowledge. The current state-of-the-art deep learning architectures applied in other fields, such as computer vision, are often easily adopted to biomedical tasks, such as radiological imaging analysis (e.g., X-ray, CT, and MRI, etc.), histological and pathological imaging analysis, and the science of omics. In order to improve the understanding of the black box, the combined model like reinforcement learning and modeling of artificial agents are the promising solution in the near future.

In the field of image registration, deep learning is not commonly used so far. However, research efforts have brought interesting applications that aims to either predict the deformations directly from the input image, or to use reinforced learning-based methods for the registration as on optimal control problem. An additional benefit of deep neural network is to obtain pattern representations which are either done in an unsupervised manner or using the registration metrics.

Computer-aided diagnosis is one of the most popular domains addressed by many studies recently. The objective of the application is to simplify the procedure of the high workload of routine tasks that the medical practitioners will prioritize in their work. Unfortunately, the deep learning approach cannot perform well in complex diagnosis tasks because the deep neural network cannot effectively represent the internal relation of the evidence. One possible method is to link the observations to evidence and then use a model to construct a line of argument towards a decision. Many authors concluded that the deep learning approach can bring an impact to computer-aided diagnosis if only the network can achieve all the evidence-based decision-making process.

Physical simulation is an application field of deep learning with accelerated improvement with realistic outcomes as with the support of advanced graphical processing in both software and hardware. As the result, this kind of applications is highly relevant to interventional applications, particularly to those requiring real-time processing. First approaches exist, yet there is considerable room for more new developments. Particularly, precision learning and variational networks seem to be suitable for such tasks, which provide some promising improvements to the prediction outcomes. We can expect that there will be some new progress in both the academia and the industry soon, especially in radiation therapy and real-time interventional dose tracking.

Reconstruction based on data-driven methods with deep learning has significant outcomes, too. However, these methods such as the GAN networks still suffer from a “new kind” of deep learning artifacts that need further improvement. As shown in the work by Huang et al. in 2018, the robustness of the deep learning-based graphical reconstruction still needs upcoming efforts in detail [30]. Both precision learning and Bayesian

approaches seem well suited to tackle the problem in the future. Another concern is that the researchers are still unclear how to benefit best from the data-driven methods while maintaining intuitive and safe image reading.

The generative adversarial networks (GANs) are a new and promising deep neural network model since 2016 [7,42], which have achieved remarkable results and become the state-of-the-art in generative modeling. On the other hand, the nascent GAN models are still based on empirical findings without concrete theory about how to implement and configure them. The current research progress is exciting because the achieved findings are significant. However, we have only scratched the surface on the capabilities of the GAN methods. The challenge is that the GAN models are fussy and prone to failure modes even after careful consideration on the model architecture, model configuration hyperparameters and data preparation.

A great advantage of all the deep learning methods is that they are inherently compatible to each other and to many classical approaches. The major difference between deep neural learning with image input and deep neural network learning with feature input is the direct use of pixel values with the deep learning architecture. In other words, the classic neural network for machine learning methods for classification, object detection, or segmentation require the pre-processing step for feature calculation, while the deep learning-based models accept direct raw data input. Therefore, deep learning for image processing can avoid errors caused by inaccurate feature calculation and segmentation. The performance of the deep learning models is higher than that of conventional feature-based classifiers. In the medical image processing tasks, the visual patterns with significant are usually sparsely located in the small region. The deep neural networks learn pixel data directly, thus all information on the pixels can be preserved when them are entered into the model. In comparison, the classic feature-based ML methods learn the features extracted from segmented lesions and thus important information is likely to be lost during the extraction because of their special sparsity. Errors are also to occur from inaccurate segmentation for complicated patterns. In addition, the development and implementation of segmentation and feature calculation, and selection of feature are unnecessary given the

deep learning models where feature calculation is not required. This feature of deep learning offers fast and efficient development for AI solutions for medical image processing. The characteristics of the deep learning models with direct image input would generally differ from those of ordinary feature-based models (with extracted feature input). As the result, the combination a classic feature-based model with the deep learning model by direct raw data input can provide a higher performance than using either of them alone. In the above studies on medical pattern detection or classification, we have witnessed these combinations generate the state-of-the-art outcomes.

In general, this fusion of the image processing pipeline by deep learning will spark many new developments in the future. In particular, the fusion on network-level using either the direct connection of networks or precision learning allows end-to-end training of algorithms. The only requirement for this deep fusion is that each operation in the hybrid net has a gradient or sub-gradient for the optimization. In fact, there are already efforts to design whole programming languages to be compatible with this kind of differential programming [108]. With such integrated networks, multi-task learning can be implemented by a single architecture. For example, Wang conjectured that the training of networks that deliver optimal reconstruction quality and the best volumetric overlap of the resulting segmentation at the same time can be integrated in a single deep learning model [129]. This advantage can hopefully be expanded to computer-aided diagnosis or patient benefit.

In general, we observe that the CNN-based architectures that emerge from deep learning are computationally efficient. The deep neural networks usually find solutions comparable or even better than many classic algorithms. However, the computational cost for inference time is often much lower than other algorithms in typical domains of medical imaging in detection, segmentation, registration, reconstruction, and physical simulation tasks. This benefit at runtime comes at high computational cost of model optimization even with the support of high-performance GPU clusters. Given an appropriate problem domain and training setup, we can thus exploit this effect to save runtime at the with relative cost of additional training time.

On the other hand, however, deep learning is an extremely data hungry ML approach. This is one of the main factors that limits the model performance with logarithmically with the amount of training data [130]. Other weakly supervised learning approaches can partially compensate this gap [131]. Hence in the clinical setting, one hospital or a group of researchers are unlikely to collect a competitive amount of data in a short period of time. Therefore, the industry needs initiatives such as hackathon challenges or medical data donors and hope that they will be successful in the near future.

Chapter 3 Survey of GAN on Medical Image Processing

In this chapter, we perform a quantitative review on the available literature on the applications of GAN for various medical image processing tasks since 2017.

The papers are retrieved from PubMed (https://www.ncbi.nlm.nih.gov/), the most recognized database for medical research. The search was performed in August 2022. The search keywords include “GAN” and “Generative Adversarial Network”.

The inclusion criteria are:

1. The research is related to computer medical image processing.
2. The research is about generative adversarial networks.

The exclusion criteria are:

1. The research is not directly related to medical image.
2. The research is not related to GAN.
3. The paper is not about original research.
4. It is a review paper.

After searching on PubMed, 952 papers were found. Then the retrieved papers are reviewed by manually by the researcher to determine whether the content is directly related to the research topic. Eventually 132 papers are included for quantitative analysis.

The applications of the GAN models for medical image processing started in 2017 after sophisticated GAN models with stable and reliable synthetic image outputs in multiple industrial domains. Examples of the most successful GAN architecture include Cycle-Consistent Adversarial Network (CycleGAN) by Zhu JY in 2017 [30], pix2pix by Isola P et al. in 2017 [36], StyleGAN by Karras T et al. in 2019 [132]. In the search with the keyword “GAN” or “generative adversarial network” on PubMed, the world class search engine to the MEDLINE database hosted by NCBI (National Center for Biotechnology Information), NLM (National Library of Medicine), USA, there are 952 relevant research papers are found since the year 2017. The detailed numbers of the relevant search papers are shown in Figure 20. It indicates an increasing trend to apply

GAN models to medical research as one of the most promising methods for data augmentation and synthesis that hopefully solve the "data greedy" bottle-net problem to improve DNN performance. Note that the current number of publications regarding GAN for medical research in 2022 remains lower than that in 2021, but the search was performed in the middle of the year 2022, we can predict there will be more GAN related research published in 2022 than in 2021 given this tendency.

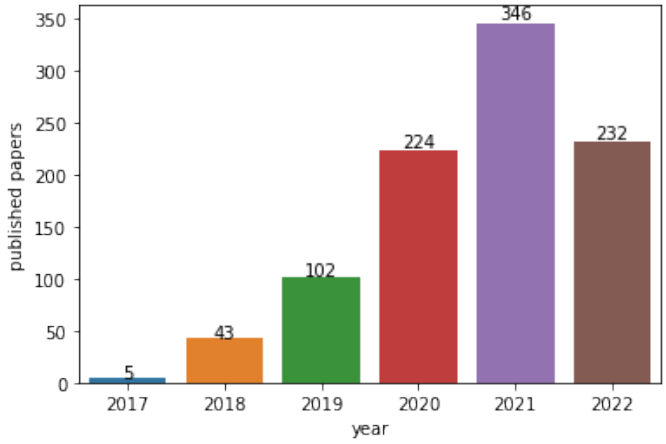


Figure 20: Publication of GAN Related Medical Research.

3.1 Overview of GAN on Medical Images

The retrieved papers are later processed by manual analysis to filter the irrelevant contents. Eventually we achieved 169 highly relevant research paper, which will be further analyzed by different topics and themes. In Table 3.1, we summarize the image category used for GAN studies in the retrieved research papers. From Table 3.1, we find that the majority of GAN applications for medical imaging falls in the radiology images such as computed tomography (CT) images, magnetic resonance images (MRI), histological images, endoscopic images, retinal images, etc. In addition, GAN can be also applied to analyze non-image data, particularly the sequential data in biomedicine. The typical applications include synthesizing electrocardiogram (ECG) and electroencephalogram (EEG) data [277-280], and synthesizing genomic or proteomic sequence [281, 293, 294]. These types of research help to exploration the new domains of GAN applications but they

are beyond the discussion of this thesis. Figure 21 illustrates comparison of the GAN applications to different types of medical images. It indicates that the studies on radiologic images occupy 75.93% (123 out of 162) of the published papers. This phenomenon has several reasons. First, the radiologic images are usually saved in the gray-scale format. Though there are research using 3D radiologic images as the data sample, each slide of the 3D data can be represented by a single channel of 2D array like the gray-scale image. Therefore, the complexity of processing this image data can be effectively simplified and less expensive compared to the 3-channel color image data. Second, the pixel brightness intensity in different types of radiologic images (i.e., CT, MR, PET, or X-Ray) has different meaning given their signal handling mechanism. GAN models can effectively learn the spatial patterns after sufficiently exposed to data samples and generate plausible synthetic images to augment the imbalanced dataset. All the factors lead to the widely use of GAN for radiologic image processing just in a few years.

In general, the research purposes or the tasks for the GANs are divided into five categories: image reconstruction and enhancement, image synthesis or augmentation, image translation, image segmentation. In addition, there are marginal applications such as dosage prediction, latent feature representation, network attack, etc. These applications can be considered as extensions to the above four categories.

Image category	Publication number
Computed tomography (CT) images	46
Magnetic resonance (MR) images	54
Positron emission tomography (PET) images	14
X-Ray images	9
Ultrasound Images	10
Cell, endoscopic and histologic image	19
Retina image	10
Medical sequential data	7

Table 1: GAN Research Category.

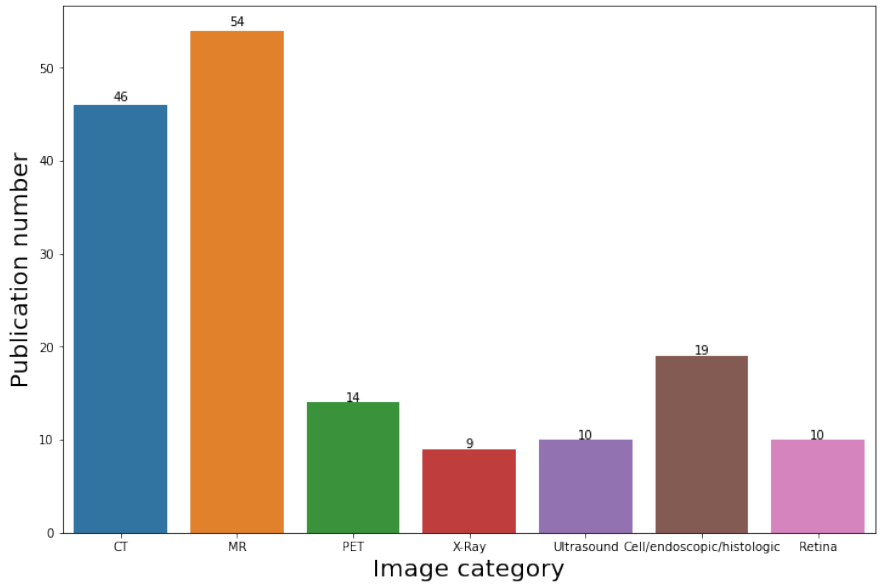


Figure 21: Publication Number in Different Medical Image Category.

In the rest sections of this chapter, we will analyze the retrieved GAN research papers respectively under these four categories of topics.

3.2 GAN on Medical Image Reconstruction and Enhancement

The tasks of image reconstruction and enhancement include converting images from low resolution to high resolution, reducing noise and artifacts caused by low-dose radiation, and reconstructing image from inconsistent signals. Compared to conventional methods like pixel interpolation, the GAN models can produce much more meaningful details to reconstructed or enhanced images which facilitates the clinical diagnosis and prognosis prediction for radiologists. Figure 22 shows the number of research on medical image reconstruction and enhancement in each image category. Note that the majority of research are about CT or MR image reconstruction and enhancement. One possible reason is that the quality of the CT or MR images mainly relies on the radiation dose received by the patients. If GAN can generate high quality radiological images or improve the image resolution while preserved the fidelity, it provides an effective method for the trade-off of patient safety and diagnostic performance.

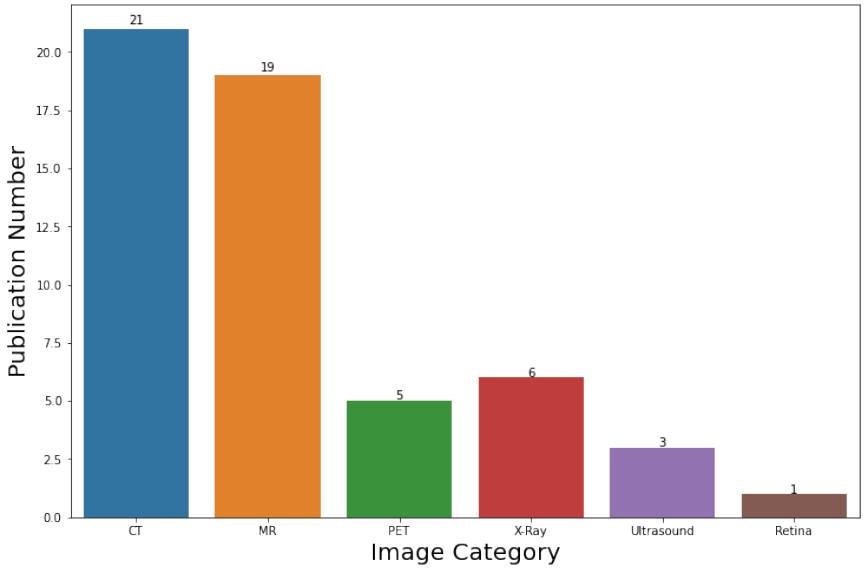


Figure 22: Publications on Image Reconstruction and Enhancement Research.

For example, the research by Wolterink JM et al. [135], Yang Q et al. [136], Yi X et al. [137], Choi K et al. [138], Hu Z et al. [139], and Tang C et al. [140] used DCGAN-based models as a new denoising technique for CT images. Wang J et al. [141], Podgorsak AR et al. [146], and Huang Z et al. [186] used DCGAN models to remove artifacts from low-dose CT images. Hsieh KY et al. [159], Janssens N et al. [160], and Usui K et al. [202] respectively used DCGAN and CycleGAN to convert low resolution CT images to high resolution CT images.

Regarding the research on MR images, Yang G et al. proposed the DAGAN model to remove the aliasing artefacts from MR images [163]. Johnson PM et al. used a conditional GAN model to perform MR image motion correction [124]. Yuan Z et al. [165] and Liu X et al. [190] used GAN models to perform MR image reconstruction and denoising. Mardani M et al. [284], Zhao M et al. [289], Luo S et al. [191], Ota J et al. [195], Zhang K et al. [213], and Zhao M et al. [216] used GAN models to improve MR image resolution. Similar GAN applications are also used for PET, regular X-Ray, ultrasound, and retina images. The research under this topic is summarised in Table 2.

Image category	Authors
CT images	Mosser L et al. [133], Bai J et al. [134], Hsieh KY et al. [159], Janssens N et al. [160], Wolterink JM et al. [135], Yang Q et al. [136], Yi X et al. [137], Choi K et al. [138], Hu Z et al. [139], Tang C et al. [140], Wang J et al. [141], Funama Y et al. [142], Harms J et al. [143], Podgorsak AR et al. [146], Tien HJ et al. [147], Wang Y et al. [148], Deng L[180], Huang Z [185], Huang Z et al. [186], Nakamoto A et al. [192], Tan C et al. [199]
MR images	Quan TM et al. [162], Yang G et al. [163], Hamghalam M et al. [164], Yuan Z et al. [165], Johnson PM et al. [124], Gomi T et al. [183], Li Z et al. [189], Liu X et al. [190], Luo S et al. [191], Ota J et al. [195], Ueki W et al. [201], Wang C et al. [204], Wei H et al. [205], Xie H et al. [206], Zhang K et al. [213], Zhao M et al. [216], Mardani M et al. [284], Zhao M et al. [289], Cui J et al. [292]
PET images	Wang Y et al. [229], Lei Y et al. [231], Armanious K et al. [232], Jeong YJ et al. [237], Xue H et al. [238]
X-Ray images	Galbusera F et al. [238], Sun Y et al. [239], Ahn G et al. [242], Bae K et al. [243], Yang CJ et al. [244], Zhou Y et al. [245]
Ultrasound Images	Goudarzi S et al. [220], Zhou Z et al. [221], Zhang L et al. [222]
Retina images	Mahapatra D et al. [265]

Table 2: GAN Research on Image Reconstruction and Enhancement.

3.3 GAN on Medical Image Synthesis or Augmentation

Another common research purpose of GAN is image synthesis or augmentation. Theoretically, when the GAN models are fully optimized, they can generate synthetic images belonging to target domain with good fidelity and diversity [295]. Fidelity means the generated images should be looked like the real images; while diversity means the generated images should have reasonable random change within the due feature domain. If the GAN models are successfully optimized, the generated images are good enough to

replace the real ones. This will solve the data-greedy restriction for deep neural networks. As a new image augmentation method, the GANs are ideal to solve the imbalanced training of medical data, where the data samples of abnormal cases or rare disease are usually difficult to collect and be well annotated. Therefore, the applications of GAN for image synthesis or augmentation are usually combined with disease diagnosis. Figure 23 shows the number of research on medical image synthesis or augmentation in each image category.

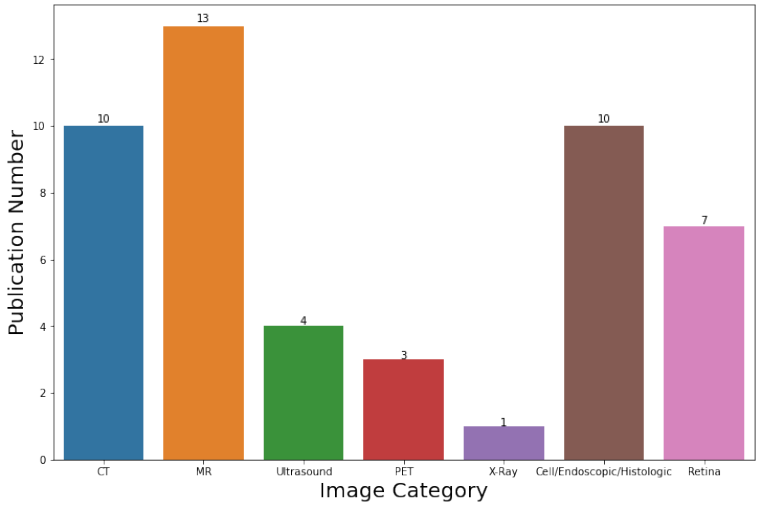


Figure 23: Publications on Image Synthesis & Augmentation Research.

Figure 23 indicates that using GAN for image synthesis or data augmentation to improve image-based diagnosis and prognosis is common for all image categories though the numbers of published research vary among them. It reflects that the research topic is of common interest for both gray-scale images (e.g., radiological images) and colored images (e.g., cell, endoscopic and histologic images).

Among the studies on radiological images, Onishi Y et al. used DCGAN to improve the classification accuracy of pulmonary nodule from CT images [154]; Babier A et al. introduced the KBPGAN as a new data augmentation method to measure the CT scan dose [156]. Nneji GU et al. [194] and Roy R et al. [198] used DCGAN as to augment the CT image classification performance optimized by low-quality CT image datasets. Dai X et al. used a unified GAN model to synthesize MR images [286]. Decourt C et al. [287] and

Delannoy Q et al. [173] both proposed to use DCGAN models as a new data augmentation approach for semi-supervised learning for insufficiently annotated MR image datasets.

In addition, many studies on colored medical images such as cell, endoscopic, histologic, or retina images also apply the GAN models for data synthesis and augmentation. For example, Hu B et al. [254], Levine AB et al. [256], and Lorencin I et al. [259] used DCGAN to improve histological diagnosis. Rau A et al. [248] and de Souza LA et al. [249] used DCGAN to enhance endoscopic image diagnosis. Iqbal T et al. [266], Zhao H et al. [267], Zheng R et al. [268], and Schlegl T et al. [269] used pix2pix or DCGAN models to synthesize the key structures (e.g., retinal fundus and neuronal) that are important for medical diagnosis. The relevant studies under this topic are summarised in Table 3.

Image category	Authors
CT images	Onishi Y et al. [154], Sandfort V et al. [155], Babier A et al. [156], Kawahara D et al. [157], Klages P et al. [158], Nneji GU et al. [194], Roy R et al. [198], Xiong YT et al. [207], Zhang Y et al. [215], Zhou H et al. [218]
MR images	Dar SU et al. [169], Gao J et al. [182], Huang P et al. [185], Zhang H et al. [211], Zheng C et al. [217], Torrents-Barrena J et al. [252], Kazuhiro K et al. [282], Yu B et al. [285], Dai X et al. [286], Decourt C et al. [287], Huang Y et al. [288], Yurt M et al. [290], Ahmad B et al. [291]
PET images	Islam J et al. [234], Kimura Y et al. [235]
X-Ray images	Guan S et al. [240]
Ultrasound Images	Fujioka T et al. [225], Zhang Q et al. [226], Zhao J et al. [227], Fujioka T et al. [228]
Retina images	Iqbal T et al. [266], Zhao H et al. [267], Zheng R et al. [268], Schlegl T et al. [269], Yu Z et al. [271], Lazaridis G et al. [272], Zhou Y et al. [274]

Cell, endoscopic or histologic images	Dirvanauskas D et al. [247], Rau A et al. [248], de Souza LA et al. [249], Hu B et al. [254], Levine AB et al. [256], Teramoto A et al. [257], Lorencin I et al. [258], Chen YI et al. [260], Theagarajan R et al. [262], Hussain S et al. [264]

Table 3: GAN Research on Image Synthesis & Augmentation.

3.4 GAN on Medical Image Translation

Image translation is another common task for radiology to convert images from one type to another. The appearance of radiologic images is like gray-scale image. However, the images are acquired by different computational mechanisms, thus the anatomical and pathological patterns usually demonstrate different among different methods such as computed tomography (CT), magnetic resonance (MR), and Positron emission tomography (PET). As a DNN architecture, GAN can learn the detailed mapping between two medical imaging pattern domains. With an optimized GAN, the digital images acquired from different methods (i.e., CT, MR, or PET) can be effectively converted to each other. This function helps the radiologists to maximize their performance to interpret the clinical findings without asking the patients to do all types of examinations. Furthermore, it lowers the radiational dose for the examination to protect the patients while keeping the best diagnostic performance. Figure 24 summarises the publications on medical image translation in each image category.

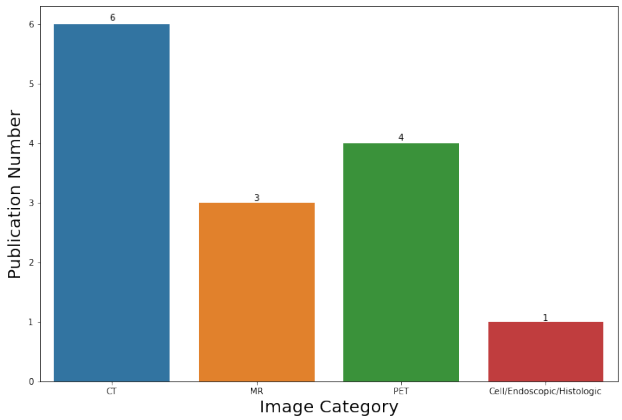


Figure 24: Publications on Image Translation Research.

The most common application of medical images translation is to convert the radiologic images between CT and MR images. For example, Fu J et al. introduced the sCTcycleGAN to convert MR images to CT images [150]. Lee JH et al. [145] and Hu N et al. [184] used cGAN models to perform conversion of CT image to MR images to acquire more detail information. Conversely, Nie D et al. [167] and Emami H et al. [168] used serialized GAN models to implement MR images to CT images conversions. Another type of image translation is to convert PET images to CT images, such as the study by Hu Z et al. using a WGAN model to perform attenuation correction and to convert PET to pseudo-CT images [233]. Bazangani F et al. introduced an E-GAN for translating 3D FDG-PET image to MR image. And Burlingame EA et al. reported to use the pix2pix GAN model to translate H&E stain histopathological images to immunofluorescence (IF) images [246].

The main purpose of the above image translation is to convert the radiologic images from a complex format to relatively simple format, such as from PET to MR, then from MR to CT, because the latter are easier to be interpret by empirical medical expertise. On the other hand, one interesting topic of image translation is seldom involved, i.e., to perform the image domain translation from the normal domain to a certain disease domain. This idea is intuitive because a medical image with some morbid abnormality can be interpreted as: normal patterns + disease patterns. Thus, it becomes the principle of our experiment designed. The relevant studies under the image translation topic are summarised in Table 4.

Image category	Authors
CT images	Maspero M et al. [144], Lee JH et al. [145], Cai J et al. [149], Fu J et al. [150], Hu N et al. [184], Wang CC et al. [204]
MR images	Pan Y et al. [166], Nie D et al. [167], Emami H et al. [168]
PET images	Bazangani F et al. [178], Armanious K et al. [230], Hu Z et al. [233], Shiyam Sundar LK et al. [236]

Cell, endoscopic or histologic images	Burlingame EA et al. [246]

Table 4: GAN Research on Image Translation.

3.5 GAN on Medical Image Segmentation

Image segmentation is also a common research topic for GAN since 2017. The research objective of using GAN to generate the landmarks or masks for the target organs or tissue structures. Compared with whole image synthesis, generating landmarks or image masks are easier and requires less computation power, but this functionality provides an ideal low-manual-labor solution for medical image segmentation. There are many successful studies on GAN for medical image segmentation since 2018, first in the application domain of radiologic image segmentation, later extending to other research domains such as endoscopic images and retina images. Figure 25 summarises the publications on GAN for medical image segmentation in each image category.

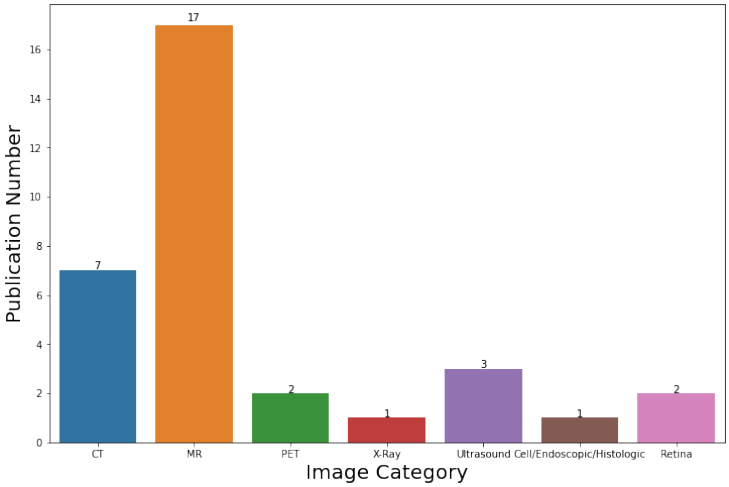


Figure 25: Publications on Image Segmentation Research.

As shown in Figure 25, the segmentation research mainly focuses on the MR images, which accounts for 51.5% (17/33) of the available studies, followed by the applications of CT images with 21.2% (7/33). It can be explained by the widely use of CT and MR images both for the diagnosis of internal medicine and surgery plan making. The segmentation

task by GAN is like image synthesis and augmentation, where the GAN generator is mainly used to produce valid mask or landmark for the real images. The image masks / landmarks were mainly produced by manual annotation. It was time consuming and usually expensive because it requires medical expertise. The introduction of the GAN to this task is hopefully solved this difficulty and meanwhile significantly improves the DNN performance for medical image segmentation.

MR images is produced by the signals of the energy released from stimulated protons of the human body in a strong magnetic field and the time the protons are realigned in the magnetic field. Therefore, the pixel intensity cannot be simply explained by the density of human tissue, but it is determined by the characteristics of different body tissues. The features of the difference between organ or tissue borders can be captured and learned by the GAN. So, the GAN models can generate clear border landmarks or shape masks for the target organs or tissues in the MR images. For example, Huo Y et al. applied the DCGAN to segment splenomegaly with MR images [283]. Shi Y et al. [170], Siddiquee MMR et al. [171], Carver EN et al. [172], Delannoy Q et al. [173], Conte GM et al. [175] and De Asis-Cruz J et al. [179] and Duman EA et al. [181] applied conditional GAN or variant GAN architecture to performance brain MR images segmentation. Gaj S et al. used conditional GAN to segment the knee structure in MR images [174]. Regarding the segmentation research on CT images, Sandfort V et al. used CycleGAN for organ segmentation in CT scans [155]. He R et al. applied DCGAN for 3D liver segmentation from multiple layers of CT scans [151]. Zhang G et al. used DCGAN to segment artery stenosis from CT images [152]. Zhang T et al. proposed an architecture called NAGAN for both CT and ultrasound image segmentation [153]. And Tyagi S et al. applied a conditional GAN to segment lung nodule from CT scans.

In addition to radiologic images, the GAN-based segmentation is extended to segment endoscopic and retina images. Poorneshwaran JM et al. [250] and Yoon D et al. [251] both reported to use GAN to segment polyp from endoscopic images. Son J et al. applied the pix2pix model to segment retinal vessels and the optic disc from retina images

[270]. Liu J et al. proposed a GAN model called A-GAN to segment cell from histologic retina images [273].

These studies reflect that GAN is improving the DNN performance not only on radiological image segmentation but also on multiple sources of medical images. The segmentation tasks vary from organs or tissues, to estimate the histologic border of heterogeneous structure. All these achievements hopefully improve the DNN performance for medical diagnosis and clinical decision making. The relevant studies under the image segmentation topic are summarised in Table 5.

Image category	Authors
CT images	Cai J et al. [149], He R et al. [151], Zhang G et al. [152], Zhang T et al. [153], Tyagi S et al. [200], van Voorst H et al. [203], Zhang L et al. [214]
MR images	Shi Y et al. [170], Siddiquee MMR et al. [171], Carver EN et al. [172], Delannoy Q et al. [173], Gaj S et al. [174], Conte GM et al. [175], Kossen T et al. [176], Wang W et al. [177], De Asis-Cruz J et al. [179], Duman EA et al. [181], Kawahara D et al. [187], Kossen T et al. [188], Niu K et al. [193], Quintana-Quintana OJ et al. [197], Yang M et al. [208], Zhu L et al. [219], Huo Y et al. [283]
PET images	Islam J et al. [234]
X-Ray images	Zhang Y et al. [241]
Ultrasound images	Han L et al. [223], Torrents-Barrena J et al. [224], Ye H et al. [209]
Cell, endoscopic or histologic images	Poorneshwaran JM et al. [250], Yoon D et al. [251], Gadermayr M et al. [263], Lei B et al. [275], Aida S et al. [276]
Retina Image	Son J et al. [270], Liu J et al. [273]

Table 5: GAN Research on Image Segmentation.

3.6 Summary

In this chapter, we perform a quantitative review on the research of GAN for medical image processing. GAN was a new DNN architecture introduced in 2014 [7], however, hundreds of GAN studies for multiple purposes have been performed in the world. The survey in this chapter performs a quantitative content analysis respectively with five topics: GAN for medical image reconstruction and enhancement with 55 papers, GAN for medical image synthesis or augmentation with 48 papers, GAN for medical image translation with 14 papers, GAN for medical image segmentation with 33 papers, and GAN for other medical applications with 15 papers.

The survey indicates the current research effort focuses on image quality enhancement, artifact removal and denoising, etc. These applications are based on the GAN models like StyleGAN [132]. Other research topic focusing on image synthesis, augmentation, translation is based on the GAN models for image-to-image translation such as pix2pix [36] and CycleGAN [30]. Last but the most important research topic within the latest three years is GAN for medical image segmentation, which mainly achievement by the combination of the DCGAN and the U-Net [34,35]. In conclusion, the GAN models have successfully expanded the scope of DNN applications and enhanced performance to a variety of medical image processing problems. Given the findings of our survey, we can expect that the medical research based on GANs models will keep increasing and new finding and breakthroughs will be achieved in the near future.

In the next three chapters of this thesis, we will introduce our new GAN architecture, adaptive cycle-consistent adversarial network, or Ad CycleGAN, a new variant of the state-of-the-art GAN for unpaired image translation to convert medical images from normal domain to a desired target disease domain. It provides a new and reliable resolution to synthesize medial images with new or rare diseases, which will eventually improve the computer-aid diagnosis and healthcare decision to a new level.

Chapter 4 Adaptive Cycle-Consistent Adversarial Network

In this chapter, we introduce a new GAN architecture named adaptive cycle-consistent adversarial network, or Ad CycleGAN to perform medical image translation from the normal domain to a particular disease domain. Image translation is considered as a sub-task of image synthesis and augmentation. Its goal is to synthesize the medical images carrying the designated disease patterns from normal images. This function satisfies the need of disease images generation because the acquisition of medical images with disease patterns is more difficult than normal images.

The image translation tasks can be done by the generative networks. In the next section, we will introduce the two mainstream generative networks: variant autoencoder (VAE) and generative adversarial networks (GAN). Then we will go into details of the cycle-consistent adversarial network (CycleGAN), the state-of-the-art GAN architecture for unpaired images translation. Next, the pretrained criterion mechanism and its role in CycleGAN optimization will be discussed which leads to the overall architecture of the new Ad CycleGAN. Finally, we will end up with a further discussion on the evaluation metric for the synthetic images.

4.1 Generative Networks for Image Synthesis

The generative models are a series of machine learning algorithms that can learn the data distribution patterns from the training dataset in the unsupervised manner, then they can generate new data with reasonable variations given the learned distribution. Deep generative learning or generative DNN, is an unsupervised learning approach to learn the training data distribution with a given DNN architecture, then it can generate new data points belonging to the learned distribution with random variance. However, the generative DNN cannot either explicitly or implicitly learn the identical distribution of the training data, but it can approximate the true parameters by different modeling techniques. There are two main methods for generative DNN: variational autoencoder (VAE) and generative adversarial networks (GAN).

4.1.1 Variational Autoencoder

Variational autoencoder, or VAE is a generative model introduced in 2013 [295]. Given the observed dataset $X = \{x^{(1)}, x^{(2)}, \dots, x^{(i)}\}$, a VAE is composed of two networks. The encoder is a DNN parameterized by ϕ to estimate the posterior distribution of the latent variable z given X: $q_\phi(z|X)$, where the training data points are taken as observations to estimate the parameters of the conditional distribution of the latent representation Z. The decoder is another DNN parameterized by θ to estimate the conditional distribution of the observed data $p_\theta(X|z)$, where the input is a sample z (usually the outputs from the encoder). The optimization objective of a VAE can be written as:

$$-\mathcal{L}(\theta, \phi; X) = \omega \cdot D_{KL}(q_\phi(z|X)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|X)}[-logp_\theta(X|z)] \quad (4.1)$$

where the reverse Kullback-Leibler (KL) divergence is to measure the distance between posterior distribution of z ($q_\phi(z|X)$) parameterized by the encoder and the prior distribution of z ($p_\theta(z)$) parametrized by the decoder. The second term of the right side of Formula 4.1 is the expected negative log-likelihood to measure the expected error of reconstructing the data points belonging to X from the latent space Z. We aim to maximize the log-likelihood of $logp_\theta(x) \geq ELBO$, where ELBO is the evidence lower bound. We let $ELBO = \mathcal{L}(\theta, \phi; X)$. The $logp_\theta(x)$ will be maximized when the negative ELBO is minimized. There are two methods to solve the ELBO minimization problem: calculating the analytic KL divergence or using the reparameterization trick. Given the GPU support in our experiments, we choose to compute the analytic KL divergence for the solution. In addition, the weight (ω) of the KL divergence term is a crucial hyperparameter for VAE performance. A too small ω cannot effectively regularizing the $q_\phi(z|X)$ term so the z sampled from $q_\phi(z)$ will be from a very low-density position of $q_\phi(z|X)$. On the contrary, when ω is too large, the distance between the posterior distribution and prior distribution, resulting in the loss of diversity. In our experiments, we set the ω=0.01 as the KL divergence weight or VAE optimization. For image generation, we use 2D convolutional layers to down-sampling (stride=2) the feature maps for the VAE encoder and use 2D

transpose convolutional layers to up-sampling (stride=2) the latent variables back to input images dimension for the VAE decoder.

4.1.2 Cycle-consistent Adversarial Network (CycleGAN)

Cycle-consistent adversarial network, or CycleGAN is the state-of-the-art conditional generative adversarial network (CGAN) for unpaired image to image translation. A typical Cycle GAN uses two generators and two discriminators to learn the mapping of two distributions by optimizing with a complex objective and reaching a state of adversarial equilibrium. The general architecture of a CycleGAN is illustrated in Figure 26.

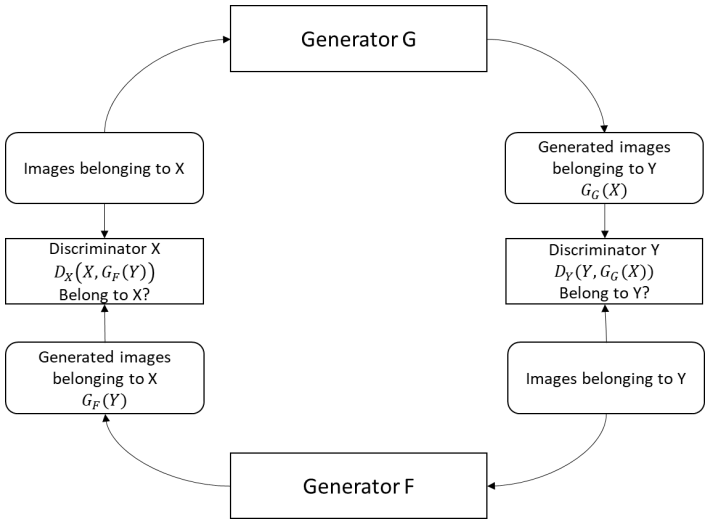


Figure 26: CycleGAN Architecture.

During optimization, the objective of the CycleGAN has three components: the adversarial loss, the cycle consistency loss, and the identity loss. The adversarial loss follows the original GAN design to measure the difference of the generated images and the target images. As shown in Figure 4.1, there are two pairs of generators and discriminators in our general model. G and D_Y (Fig. 1) aim to adversarially generate and distinguish the images belong to domain X and the images belonging to domain Y, i.e., $min_G max_{D_Y} \mathcal{L}_{GAN}(G, D_Y, X, Y)$. The optimization objective is written as:

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log (1 - D_Y(G(x))] \quad (4.2)$$

where the data distribution of X and Y are denoted as $x \sim p_{data}(x)$, and $y \sim p_{data}(y)$. On the other hand, F and D_X (Figure4.1) aim to adversarially generate and distinguish the generated and real images belonging to domain Y, i.e. $min_F max_{D_X} \mathcal{L}_{GAN}(F, D_X, Y, X)$, the corresponding objective is writing as:

$$\mathcal{L}_{GAN}(F, D_X, Y, X) = \mathbb{E}_{x \sim p_{data}(x)}[\log D_X(x)] + \mathbb{E}_{y \sim p_{data}(y)}[\log (1 - D_X(F(y))] \quad (4.3)$$

Thus, the total adversarial loss during a single iteration is summation of the loss from Formula 4.2 and formula 4.3, i.e., $\mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X)$. The adversarial optimization can theoretically learn the mappings of G and F to produce identical images. However, this goal is unrealistic in two folds: first, given the ideal situation, the generator networks can randomly map an input image from the source domain to a random image in the target domain, and it is not our desired outcome. Second, we need to guarantee that the generated images have valid shapes and other uncommon elements within a reasonable scope of the real images. We add the cycle-consistent losses to reversely translate the images back to their original domains, i.e., $x \to G(x) \to F(G(x)) \approx x$ (forward cycle consistency), and $y \to F(y) \to G(F(y)) \approx y$ (backward cycle consistency). The total cycle-consistent loss is written as:

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}\left[\left\|F(G(x)) - x\right\|_1\right] + \mathbb{E}_{y \sim p_{data}(y)}[\left\|G(F(y)) - y\right\|_1] \quad (4.4)$$

Furthermore, an identity loss is added to measure how close the generated image to the real image itself if the real image goes through the CycleGAN generator, i.e., $x \to F(x) \approx x$, and $y \to G(y) \approx y$. Adding to identity loss to the total loss of the generator can help to preserve the original color. The identity loss can be expressed as:

$$\mathcal{L}_{iden}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\|F(x) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G(y) - y\|_1] \quad (4.5)$$

The full generator loss of the CycleGAN is written as the summation of the above three loss functions:

$$\mathcal{L}_{total} = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F) + \sigma \mathcal{L}_{iden}(G, F) \quad (4.6)$$

where λ and σ are the parameters to respectively adjust the importance of the cycle-consistent loss, and the identity loss during the model optimization. Thus, the objective for the whole model optimization is to solve:

$$G^*, F^* = \arg min_{G,F}\, max_{D_X,D_Y} \mathcal{L}(G, F, D_X, D_Y) \quad (4.7)$$

In the experiments we find that though the original CycleGAN models can produce visually acceptable synthetic images, but the quality of the images by CycleGAN is lower than the new Ad CycleGAN based on the quantitative evaluation. The pseudo code of the optimization algorithm for the CycleGAN model is presented in Algorithm 4.1

Algorithm 4.1 CycleGAN optimization

for number of epochs **do**

  for number of batches **do**

    Sample minibatch $\leftarrow \{x^{(i)}\}_{i=1}^{m} \in X$

    Sample minibatch $\leftarrow \{y^{(j)}\}_{j=1}^{m} \in Y$

    Generate m synthetic samples of $G(x) and\ F(y)$

      $synthetic\ X: X \rightarrow G(x)$

      $synthetic\ Y: Y \rightarrow F(y)$

    Compute the Adversarial loss

      $\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y\sim p(y^{(m)})}[\log D_Y(y)] + \mathbb{E}_{x\sim p(x^{(m)})}[\log(1 - D_Y(G(x)))]$

      $\mathcal{L}_{GAN}(F, D_X, Y, X) = \mathbb{E}_{x\sim p_{data}(x)}[\log D_X(x)] + \mathbb{E}_{y\sim p_{data}(y)}[\log(1 - D_X(F(y)))]$

    Generate m cycle sample of $F(G(x)) and\ G(F(y))$

      $Cycle\ X: G(x) \rightarrow F(G(x))$

      $Cycle\ Y: F(y) \rightarrow G(F(y))$

    Compute the Cycle loss

      $\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x\sim p(x^{(m)})}\left[\|F(G(x)) - x\|_1\right] + \mathbb{E}_{y\sim p(y^{(m)})}[\|G(F(y)) - y\|_1]$

    Generate m identical sample of $F(x) and\ G(y)$

      $identical\ X: X \rightarrow F(x)$

      $identical\ Y: Y \rightarrow G(y)$

    Compute the identity loss

      $\mathcal{L}_{iden}(G, F) = \mathbb{E}_{x\sim p(x^{(m)})}[\|F(x) - x\|_1] + \mathbb{E}_{y\sim p(y^{(m)})}[\|G(y) - y\|_1]$

    Compute the total generator loss

22: $\mathcal{L}_{total} = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda\mathcal{L}_{cyc}(G, F) + \sigma\mathcal{L}_{iden}(G, F)$

23: Update the Discriminator D_X *and* D_Y

24: $max_{D_X} L_{GAN}(F, D_X, X, Y)$

25: $max_{D_Y} L_{GAN}(G, D_y, X, Y)$

26: Update the Generators G, F

27: $min_{G,F} L(G, F, D_X, D_Y)$

28: **end do**

29: **end do**

4.2 Role of the Criterion in Cycle GAN Optimization

The term criterion originates from the concept of critics introduced by Arjovsky M et al. for the optimization of their Wasserstein GAN (WGAN) model [28]. Its strategy is to use the discriminator as a critic to evaluate the quality of the generated images against the real ones instead of estimating the probability of the generated images being fake. To extend this idea, we can add extra criterion to evaluate the generated images from multiple aspects except simply classifying as real or fake. The errors from different criteria can be finally congregated as classification or criterion loss as a new component of the total generator loss.

In our new model, we introduce two loss terms: the cycle criterion loss and the identity criterion loss. Both are estimated by a pretrained residual network to assess the likelihood of the generated images belong to the correct class. In other word, the pretrained network is classifier to evaluate the binary cross entropy from the output of the last layer activation. The joint critics loss is written as:

$$\mathcal{L}_{critics} = \mathcal{L}_{c-cycle} + \mathcal{L}_{c-identity} \tag{4.8}$$

where the cycle criterion loss is to measure the similarity of $x \sim F(G(x))$ and of $y \sim G(F(y))$, and the identity criterion loss is to measure the similarity of $x \sim F(x)$ and $y \sim G(y)$. In other words, like the cycle loss and identity loss, the cycle criterion loss $\mathcal{L}_{c-cycle}$ quantitatively measures whether of the back-translated images are still be

classified as the original class, and the identity criterion loss $\mathcal{L}_{c-identity}$ quantitatively measures whether the trained generators can produce real images from a real observed sample that still consistent to the same class.

In addition, when the Cycle GAN training reaches an adversarial equilibrium, the critics loss can periodically add an extra oscillation momentum to the stable condition to push the generator progress to learn more details. The new $\mathcal{L}_{critics}$ term is considered as a regularization method to prevent the saturated status of the GAN optimization because it provides a method to make the GAN training controllable to a certain degree. However, we need to add an empirical decay factor to the critics loss term to control its side effect of breaking the adversarial equilibrium leading the GAN model to learn the loss patterns again through more iterations.

4.3 Ad CycleGAN

Based on the above discussion, the newly proposed Ad Cycle GAN consists of two generators and discriminators to learning the mapping between the image domain, and a pretrained classifier as the criterion to ensure the generated images containing the key discriminative patterns which are likely to be ignored due to the homogenous or similarity of the two image domains. The total loss function of the generators in the Ad Cycle GAN consists of four parts:

- Adversarial loss: $\mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X)$
- Cycle consistency loss: $\mathcal{L}_{cycle}(G, F)$
- Identity loss: $\mathcal{L}_{iden}(G, F)$
- Criterion Loss: $\mathcal{L}_{critics} = \mathcal{L}_{c-cycle} + \mathcal{L}_{c-identity}$

The total generator loss of the original CycleGAN in Formula 4.6 is revised as:

$$\mathcal{L}_{total} = [\mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X)] + \lambda\mathcal{L}_{cycle}(G, F) + \lambda\mathcal{L}_{iden}(G, F) + \kappa(\varphi\mathcal{L}_{critics}) \quad (4.9)$$

In the Ad CycleGAN architecture, the generators follow the U-Net architecture with skip connections to reduce the input feature size from 64 by 64 to 1 by 1 then to restore to 64 by 64. The discriminators follow the PatchGAN architecture with an output of 4-by-4-by-1 feature map (given the low resolution of our dataset) to determine with the images are

real or fake. We choose to use the binary cross entropy as the objective function for the discriminator loss and the adversarial loss terms for the generators. The cycle consistency loss and the identity loss use the mean of absolute error (MAE) function as the objective. The terms of the criterion loss are measured by the sparse categorical cross entropy as the same method as how the pretrained criterion was optimized. Though some studies recommend using the unbounded smooth loss function such as to optimize the GAN models such as Wasserstein loss or least square loss (MSE) [28, 296]. Empirically, the choice of loss functions is mainly based on the components of the total loss objective. If all errors can be measured within similar scales, using the unbounded loss functions is straightforward and easier for the overall GAN optimization. However, if the GAN architecture consists of many components like this case, using hypermeters to adjust the importance of different terms or to determine the frequency of loss injection to the total loss can provide a more flexible option for GAN optimization as described in Formula 4.9. The Ad Cycle GAN architecture is illustrated in Figure 27. And the pseudo code of the optimization algorithm for the Ad CycleGAN model is presented in Algorithm 4.2.

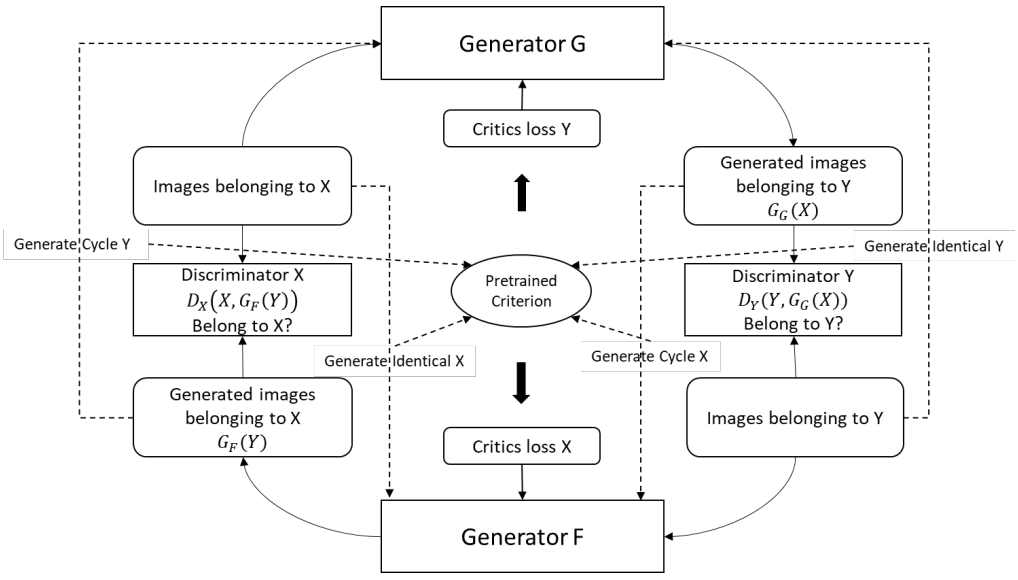


Figure 27: Ad CycleGAN Architecture.

Algorithm 4.2 Ad CycleGAN optimization

for number of epochs **do**

  for number of batches **do**

    Sample minibatch ← $\{x^{(i)}\}_{i=1}^{m} \in X$

    Sample minibatch ← $\{y^{(j)}\}_{j=1}^{m} \in Y$

    Generate m synthetic samples of $G(x) and\ F(y)$

      $synthetic\ X: X \rightarrow G(x)$

      $synthetic\ Y: Y \rightarrow F(y)$

    Compute the Adversarial loss

      $\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p(y^{(m)})}[\log D_Y(y)] + \mathbb{E}_{x \sim p(x^{(m)})}[\log(1 - D_Y(G(x)))]$

      $\mathcal{L}_{GAN}(F, D_X, Y, X) = \mathbb{E}_{x \sim p_{data}(x)}[\log D_X(x)] + \mathbb{E}_{y \sim p_{data}(y)}[\log(1 - D_X(F(y)))]$

    Generate m cycle sample of $F(G(x)) and\ G(F(y))$

      $Cycle\ X: G(x) \rightarrow F(G(x))$

      $Cycle\ Y: F(y) \rightarrow G(F(y))$

    Compute the Cycle loss

      $\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p(x^{(m)})}\left[\|F(G(x)) - x\|_1\right] + \mathbb{E}_{y \sim p(y^{(m)})}[\|G(F(y)) - y\|_1]$

    Generate m identical sample of $F(x) and\ G(y)$

      $identical\ X: X \rightarrow F(x)$

      $identical\ Y: Y \rightarrow G(y)$

    Compute the identity loss

      $\mathcal{L}_{iden}(G, F) = \mathbb{E}_{x \sim p(x^{(m)})}[\|F(x) - x\|_1] + \mathbb{E}_{y \sim p(y^{(m)})}[\|G(y) - y\|_1]$

    Compute the criterion loss for cycle sample: $\mathcal{L}_{c-cycle}$

    Compute the criterion loss for identical sample: $\mathcal{L}_{c-identity}$

    Compute the total generator loss

      $\mathcal{L}_{total} = [\mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X)] + \lambda \mathcal{L}_{cycle}(G, F) + \lambda \mathcal{L}_{iden}(G, F) + \kappa(\varphi \mathcal{L}_{critics})$

    Update the Discriminator $D_X\ and\ D_Y$

      $max_{D_X} L_{GAN}(F, D_X, X, Y)$

      $max_{D_Y} L_{GAN}(G, D_y, X, Y)$

28: Update the Generators G, F

29: $min_{G,F} L(G, F, D_X, D_Y)$

30: **end do**

31: **end do**

4.4 Evaluation Metrics

In this section, we will discuss the evaluation metrics for the generated images. The performance evaluation of GAN networks is usually subjective and remains as an open problem [42]. Since our research objective is to generated synthetic medical images with good fidelity and diversity, we need to measure both the quality of the images and ensure the generated images belonging to the correct class, i.e., carrying the diagnostic significant patterns. The latter task is simple and straightforward because we can reuse the high-performance criterion to estimate the accuracy of the classification. As to measure the quality of the synthetic images, some quantitative metrics are selected.

The assessment of image quality means to quantitatively measures the degradation of the target images. There are generally two types of methods: subjective evaluation and objective evaluation. Subjective evaluation requires human expertise, and it is time consuming and difficult to replication. Therefore, we apply the objective metrics to compare the synthetic images and the generated images with the assumption that the high-quality synthetic images have higher degrees of the similarity to the real images. We also use the Frechet Inception Distance (FID) as a common acceptable metric to compare the quality of the images synthesized by different generative models.

According to the literature review for GAN evaluation measure [41], there are currently 24 quantitative and 5 qualitative measures for GAN evaluation. However, the measures are respectively introduced in different times, so some new methods such as Average Log-likelihood and Coverage Metric are questioned by the GAN research community. One of the typical measures is Inception score (IS) [42]. is a commonly accepted measure by the GAN research community based on the features output by a pre-

trained Inception v3 model with the ImageNet dataset. There are many studies based on the IS method and a series of improved methods based on IS are introduced such as Modified Inception Score (m-IS), Mode Score, AM Score etc. However, these following methods cannot provide better quantitative measure for GAN. The Frechet inception distance or FID was introduced as an enhanced method based on IS. FID also uses a pre-trained Inception v3 model how similar the synthesis images to the input real images. FID is considered a robust measure for GAN performance. It overcomes some drawbacks of IS and IS related measures. For example, FID is more consistent with human subjective judgements, more robust to noise, and it can detect intra-class mode dropping [42]. However, FID still has limitation for medical image evaluation. One problem is that medical images should be similar for the normal tissues and meanwhile should have the significant patterns for diagnosis. In my study, the objective is to use an optimized Ad CycleGAN to translate normal images to disease-positive images. In the ideal condition, the synthetic images should be real-looking, similar to the input images, and contain the target disease patterns which are not in the input images. Therefore, the use case of medical GAN is different from the general-purpose GAN, but the fidelity and diversity of the synthetic images are still important. Using FID as one of the evaluation metrics can provide a robust and quantitative comparison among different generative models (Ad CycleGAN, Cycle GAN, and VAE in my use case), but the magnitude of the FID score has different meaning compared to the general-purpose GAN. Other quantitative metrics include MSE, RMSE, PSNR, UIQI, SCC, SAM and VIF are commonly used in medical image analysis, especially for grayscale images such as X-Ray, CT, and MR images. Therefore, I include these metrics in my GAN evaluation.

The quantitative evaluation metrics for our experiments include:

- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Peak Signal-to-Noise Ratio (PSNR)
- Universal Image Quality Index (UIQI)
- Spatial Correlation Coefficient (SCC)

- Spectral Angle Mapper (SAM)
- Visual Information Fidelity (VIF)

MSE, RMSE and PSNR are metrics to measure the pixel difference between the synthetic images and the real images. MSE is the accumulated mean squared error of two images with the 2D size $M \times N$. And RMSE is the accumulated root mean square error of the two images.

$$MSE = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(X_{m,n} - Y_{m,n})^2 \tag{4.10}$$

$$RMSE = \sqrt{\frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(X_{m,n}-Y_{m,n})^2}{MN}} \tag{4.11}$$

Peak Signal-to-Noise Ratio (PSNR) is a measure for image quality by Wang Z et al. [313] in 2004 based on the pixel difference between the synthetic image and the real image given the following formula:

$$PSNR = 10log\frac{s^2}{MSE} \tag{4.12}$$

where $s = 255$ as the range of the pixel value is from 0 to 255.

Universal image quality index (UIQI or UQI) was also introduced by Wang Z and Bovik AC [314] in 2002 based on summarizing the attributes of human vision. The synthetic images and the real images are compared in three aspects: luminance, contrast, and structure. Given μ_x and μ_y are the mean pixel value of the real image and the synthetic image, $\sigma_x, \sigma_y, and\ \sigma_{xy}$ are the standard deviation and covariance of the real image and the synthetic image, the luminance, contrast, and structure comparison of the two images are defined as:

$$luminance: l(x,y) = \frac{2\mu_x^2\mu_y^2}{\mu_x^2+\mu_y^2},\ contrast: c(x,y) = \frac{2\sigma_x\sigma_y}{\sigma_x^2+\sigma_y^2},\ structure: s(x,y) = \frac{2\sigma_{xy}}{\sigma_x+\sigma_y} \tag{4.13}$$

Based on these three comparisons, the UIQI is defined as:

$$UIQI(x,y) = l(x,y)\cdot c(x,y)\cdot s(x,y) = \frac{4\mu_x\mu_y\mu_{xy}}{(\mu_x^2+\mu_y^2)(\sigma_x^2+\sigma_y^2)} \tag{4.14}$$

Spatial Correlation Coefficient (SCC) is a correlation-based measure to compare the difference of image quality. Spectral Angle Mapper (SAM) [315, 316] are spectral distance-based measures to quantify the image difference using discrete Fourier transform.

The difference of the Fourier magnitude reflects the image quality. Visual Information Fidelity (VIF) is another measure based on human visual perception. VIF use information theoretic criterion to quantify image fidelity measurement. Under this framework, the difference of the information extracted from the real image and the information loss to the synthetic image by human brain is quantified as the VIF score using visual natural scene statistics (NSS), human visual system (HVS) and an image distortion model. The VIF score is represented as a numeric value between 0 and 1, where higher value indicates better image fidelity and quality. The detail of the Frechet Inception Distance (FID) has been discussed in section 2.1.5

4.5 Summary

In this chapter, we discuss the rationale, architecture of variational autoencoder (VAE), cycle-consistent adversarial network (CycleGAN) and the proposed adaptive cycle-consistent adversarial network (Ad CycleGAN), as well as the optimization algorithms for CycleGAN and Ad CycleGAN. Then we briefly introduce the quantitative metrics for the synthetic images. These components form the framework for the following experiments.

In Chapter 5, we will present the experiments of using these DNN models to synthesize blood cell images which are either normal or infected by malaria plasmodium. Then the synthetic images will be evaluated and compared by the quantitative metrics.

Chapter 5 Ad CycleGAN for Histology Image Synthesis

In this chapter, we present the series of experiments on using the newly proposed Ad CycleGAN to synthesize histological images. The experiment dataset contains normal and malaria infected blood cell images. Malaria is a tropical infectious disease threating global health. In our study in 2016, a convolution neural network (CNN) with 6 convolutional layers was implemented for classification of malaria infected blood cells. The CNN was trained with a dataset with 27,578 blood cell images (ration: 1:1) and the average accuracy is 97.37% [3]. The following studies also report extremely high classification accuracy [302, 303]. However, these results are all achieved based on a large, well annotated dataset for training the CNN models. In most cases, big annotated medical image datasets are difficult to acquired. If the medical images are annotated by non-medical persons, the quality of the image data is suspicious due to the lack of expertise. Therefore, we should seek for a solution to minimize the human expertise intervention to the deep neural network (DNN) optimization.

Another drawback of DNN is that the specific medical image patterns are different from general-purposed images such as those in the ImageNet dataset. As the result, when using transfer learning with DNN models trained by the ImageNet to fine tune a new model for the medical images, the pretrained feature extractors usually cannot effectively capture the medical significant patterns through the complex architecture but simply develop meaningless combinations for the final decision. In our previous work on CNN for the malaria blood cell image classification, the transfer learning approach has lower accuracy (91.99%) than the randomly initialized CNN (97.37%) [3]. In addition, the study by Hirano H et al. reveals that the seemly high-performance DNN models for medical images are vulnerable from network attacks [4].

These become the motivation of the experiments. In the following experiments, we will sequentially use dataset to optimize variant autoencoder (VAE), cycle-consistent adversarial network (CycleGAN), and adaptive cycle-consistent adversarial network (Ad CycleGAN). Then we will compare the quality of the synthetic images respectively

generated by these three models with the above-mentioned quantitative metrics. Finally, the results will be interpreted and summarized.

5.1 Material and Methods

We use an open-source dataset containing 24 thousand parasitemic (malaria positive) and normal (malaria negative) segmented blood cell images (ratio 1:1) hosted by National Library of Medicine (NLM) as we did our previous work [3]. The dataset is accessible at ftp://lhcftp.nlm.nih.gov/Open-Access-Datasets/Malaria/NIH-NLM-ThinBloodSmearsPf/ for the development of an Android based automatic malaria screener [303]. One benefit of using the Cycle GAN architecture is that the model can be optimized by a relatively small dataset (e.g., hundreds of images). To save the runtime, we randomly choose 18,000 images (9,000 from each class) for the Cycle GAN optimization and the rest images for the following tests. Given the original image size, they are resized to 32-by-32-by-3 to fit the model input. And the models are respectively optimized by 600 epochs on the Google Colab Pro Cloud GPU support. The average optimization runtime of a single epoch is 2 seconds for the VAE model, 35 seconds for the CycleGAN model, and 40 seconds for the Ad CycleGAN model. A sample of the real blood cell images is illustrated in Figure 28.

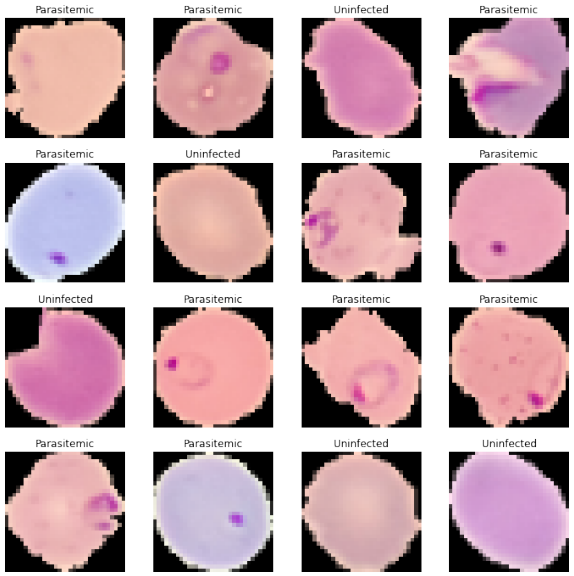


Figure 28: Original Blood Cell Images.

From Figure 28, we find that it is difficult to discriminate the uninfected blood cells (malaria negative) from the parasitemic cells (malaria positive) without medical expertise because the images from both classes have similar background color and randomly dyed dots inside the cells. Since our hypothesis is that the malaria positive cell images and the malaria negative cell images belongs to two separable distribution domains. Therefore, we can use the VAE to learn the distribution parameters of the malaria positive images, and we can also use the CycleGAN or Ad CycleGAN to learn the mapping parameters between the two domains.

5.2 Experiment Results and Interpretation

We implement the VAE models as mentioned above and optimized them with different weights of KL divergence. Finally, we conclude the VAE generates the best synthetic images when the KL divergence weight is 0.01. The model used the Adam (adaptive moment estimation) optimizer with the initial learning rate started with 2×10^{-4}. The VAE model is optimized with 600 epochs with the mini-batch size of 512. The synthetic images generated by the trained VAE are shown in Figure 29.

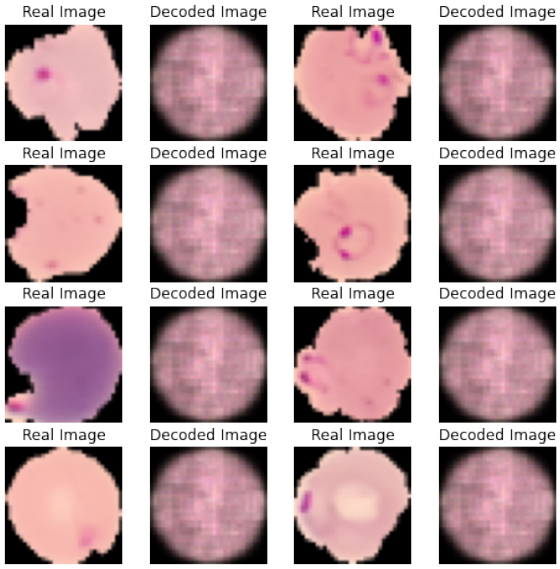


Figure 29: Synthetic Blood Cell Images by VAE.

In the next experiments, we respectively optimized the CycleGAN and the Ad CycleGAN, with λ=80.0, σ=60.0, φ=0.1, and κ=20, which means the criterion loss is added to the total generator loss term very 20 steps during the model optimization. The models are optimized by the Adam optimizer with the initial learning rate of 2×10^{-4} for 600 epochs. The mini-batch size is 128. The synthetic blood cell images respectively generated by the CycleGAN and by the Ad CycleGAN are shown in Figure 30 and Figure 31.

The quantitative measures for the quality of the synthetic images are listed in Table 6, the FID score and the classification accuracy to the due category of the synthetic images are listed in Table 7.

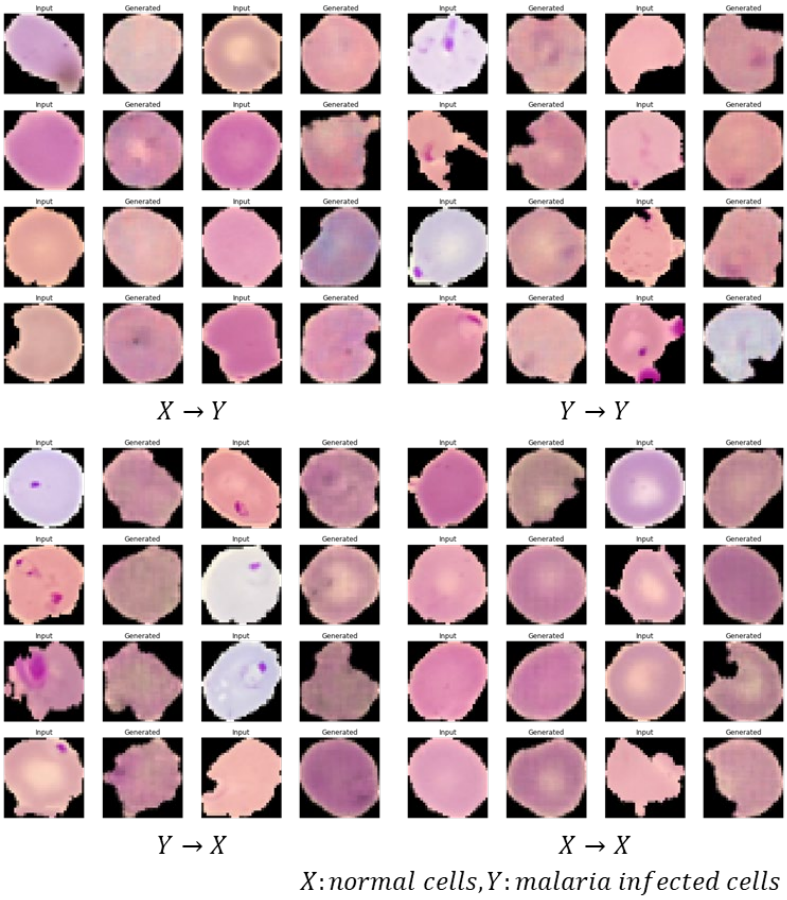


Figure 30: Synthetic Blood Cell Images by CycleGAN.

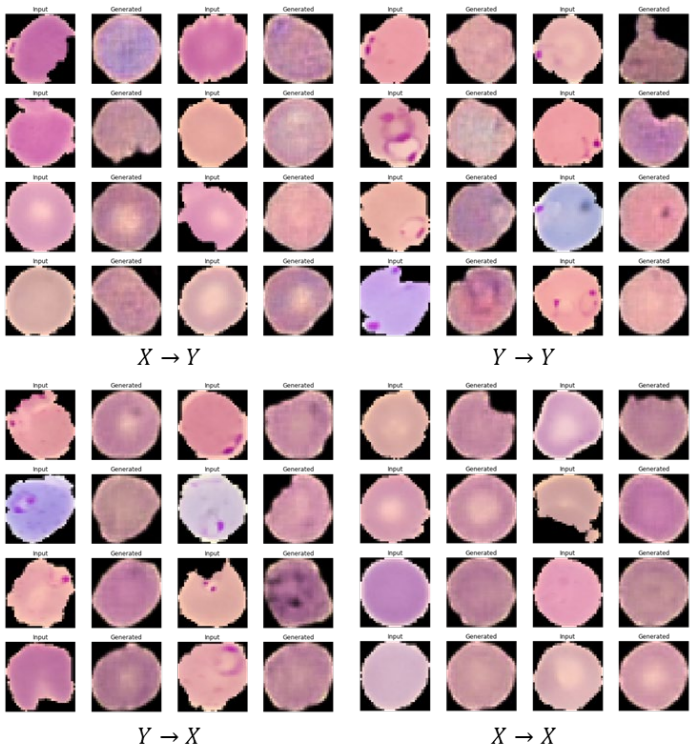


Figure 31: Synthetic Blood Cell Images by Ad CycleGAN.

Image source	MSE (±std)	RMSE (±std)	PSNR (±std)	UIQI (±std)	SCC (±std)	SAM (±std)	VIF (±std)
VAE	7288.789 (2213.888)	84.401 (12.854)	9.707 (1.353)	0.678 (0.071)	0.100 (0.051)	0.390 (0.095)	0.070 (0.007)
CycleGAN ($X \rightarrow Y$)	1660.815 (873.898)	39.45 (10.206)	16.496 (2.238)	0.937 (0.045)	0.335 (0.065)	0.174 (0.029)	0.262 (0.049)
CycleGAN ($Y \rightarrow Y$)	619.545 (531.475)	23.425 (8.413)	21.189 (2.685)	0.981 (0.022)	0.609 (0.044)	0.109 (0.018)	0.421 (0.057)
Ad CycleGAN ($X \rightarrow Y$)	598.751 (358.173)	23.709 (6.049)	20.871 (1.968)	0.980 (0.022)	0.524 (0.051)	0.136 (0.028)	0.304 (0.083)
Ad CycleGAN ($Y \rightarrow Y$)	710.698 (578.58)	25.280 (8.462)	20.466 (2.492)	0.977 (0.038)	0.556 (0.056)	0.128 (0.024)	0.346 (0.057)

Table 6: Quantitative Measure of the Blood Cell Synthetic Images.

Image source	FID (±std)	Accuracy
VAE	8.552×10^{-5} (7.989×10^{-6})	0.0
CycleGAN **($X \rightarrow Y$)**	2.356×10^{-6} (1.828×10^{-7})	0.7929
CycleGAN **($Y \rightarrow Y$)**	4.443×10^{-6} (4.402×10^{-7})	0.9570
Ad CycleGAN **($X \rightarrow Y$)**	4.241×10^{-6} (1.948×10^{-7})	0.9961
Ad CycleGAN **($Y \rightarrow Y$)**	9.324×10^{-6} (2.991×10^{-7})	1.0

Table 7: FID score of Classification Accuracy of the Blood Cell Synthetic Images.

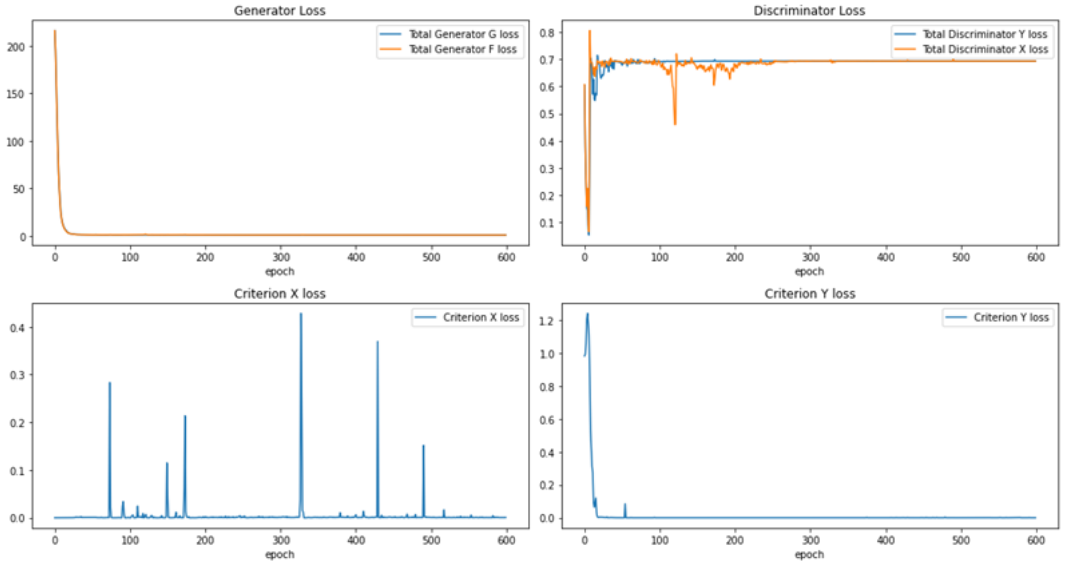
From Figure 30 and Figure 31, we observe that both the conventional CycleGAN and the new Ad CycleGAN can synthesize high quality blood cell images with good visual fidelity and diversity compared to those generated by variant autoencoder (VAE). When taking a closer look, the images generated by CycleGAN seem to contain less artifacts compared those generated by Ad CycleGAN, but this subjective observation is reversed by the quantitative measures in Table 7

The quantitative scores in Table 7 also confirm that the quality of the synthetic images by CycleGAN and Ad CycleGAN is superior to those by VAE, where the mean squared error (MSE) and root mean squared error (RMSE) between the input real images and the output synthetic images by the CycleGAN and Ad CycleGAN is much smaller than the VAE. Among the rest quantitative measures including Peak Signal-to-Noise Ratio (PSNR), Universal Quality Image Index (UIQI), Spatial Correlation Coefficient (SCC), Spectral Angle Mapper (SAM), and Visual Information Fidelity (VIF), the scores to the synthetic images by CycleGAN and Ad CycleGAN are much higher than those by VAE. All these indicate that the GAN models (CycleGAN and Ad CycleGAN) generate much better synthetic images compared to VAE. Another observation on the synthetic images respectively by the two GAN models finds that the new Ad CycleGAN has more stable image quality output compared to the original CycleGAN model. We used both the normal

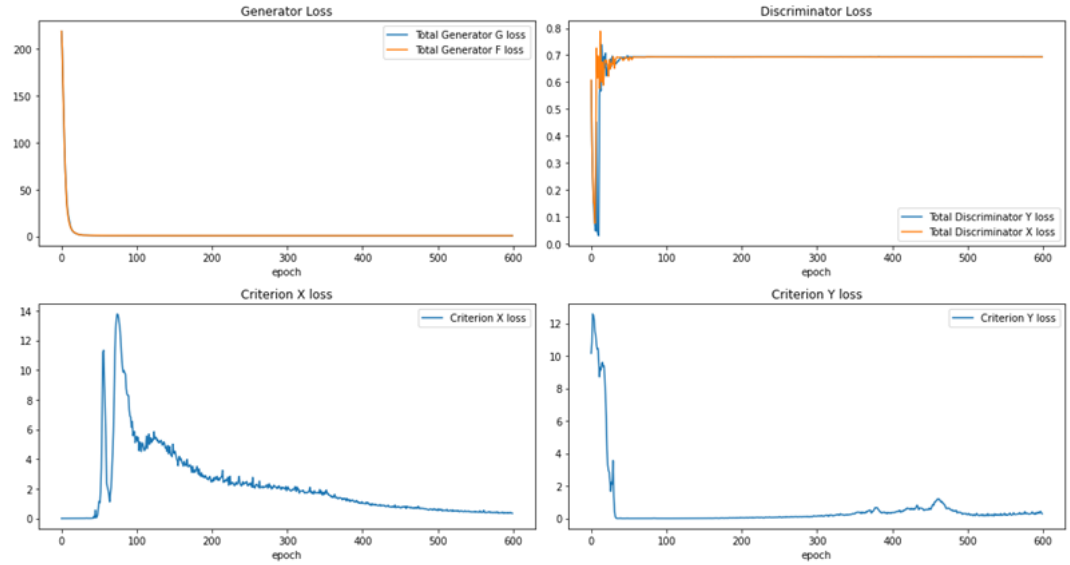
images (belong to domain X) and malaria infected images (belong to domain Y) to synthetize the malaria infected blood cell images. The new Ad CycleGAN generates better synthetic images when we use the normal images to synthesize malaria infected images, i.e., to perform $X \rightarrow Y$ translation, though the synthetic image by the original CycleGAN through real malaria infected images has comparable quality compared to those by Ad CycleGAN.

The FID score is a harmonious measure for both image fidelity and diversity by GAN models as we have discussed in Chapter 2 section 2.1.5. From Table 8 we observe the FID score of the synthetic images by the GAN models lower than those by the VAE. This phenomenon does not tell us the images by the VAE are better. However, it indicates the images by the GAN models are more homogenous to the real malaria infected images. The images synthesized by the Ad CycleGAN from normal images (i.e., to perform $X \rightarrow Y$ translation) and those synthesized by the original CycleGAN from malaria infected images (i.e., to perform $Y \rightarrow Y$ identical conversion) both have high FID compared to the rest two groups by GANs. This result indicates that the Ad CycleGAN has superior performance for image translation that the original CycleGAN. In addition, the classification accuracy of the synthetic images by Ad CycleGAN is higher than both the CycleGAN and VAE. This result clearly proves that the newly proposed Ad CycleGAN has superior capacity to perform image translation to the target category compared to both the original CycleGAN and VAE.

Next, we compare the optimization process of the new Ad CycleGAN with the original CycleGAN. We add a periodic criterion loss to the total generator loss once every 20 steps during the optimization. At the top of Figure 32. we can observe a clear fluctuation from the beginning to approximate epoch 300, where the criterion loss acts as extra momentum for the discriminator loss. The criterion loss for X also has periodic surges which indicates the quality of the synthetic images are not stable at the beginning. Fortunately, the trend changes to be stable when the decay factor controls the influence of the criterion loss to a proper range.



Optimization of Ad CycleGAN



Optimization of CycleGAN

Figure 32: Optimization Process of Ad CycleGAN and CycleGAN for Blood Cell Images.

On the other hand, the discriminator loss of the origin CycleGAN becomes stable early at approximately epoch 100. It implies the magnitude of gradients for GAN update become low at the early stage thus the entire GAN optimization is slowed down. The effect of the so-called adversarial equilibrium has two folds. First, when the training reaches the

adversarial equilibrium, the gradients remain low thus the optimization process becomes very slow. The GAN mainly learns the details of the image patterns under a stand profile. In this experiment, the GAN learns the blood cell contours approximately at the first 100 epochs, and after that its updates is slow down and start to focus on the details inside the blood cell. However, the low gradients after the adversarial equilibrium become so low that if some errors occur, it is very difficult for the GAN to walk out the wrong path to further learn the correct patterns. This phenomenon is reflected in the change of the criterion loss X and Y at the bottom of Figure 32 where the original CycleGAN is optimized without the loss from the pretrained criterion. The criterion loss cannot be effective controlled throughout the whole optimization and sometimes it divergent because there is no control to ensure the synthetic images to go to the correct category. On the other hand, the criterion loss has its side effect to the GAN optimization if its magnitude cannot be controlled is an acceptable range as shown in the optimization process of the Ad CycleGAN. Therefore, it is crucial to set a proper decay factor φ combining with the periodic factor κ to control the weight of the criterion loss throughout the Ad CycleGAN optimization process.

5.3 Summary

In this chapter, we present the experiment of Ad CycleGAN for malaria infected blood cell synthesis and image translation between normal blood cells and malaria infected blood cells. The results are compared with the synthetic images by the original CycleGAN and the VAE. Except for the subjective evaluation by human eyes, the quantitative metrics including MSE, RMSE, PSNR, UIQI, SCC, SAM, and VIF are used to measure the fidelity and quality of the synthetic images using the real input images as reference. The MSE, RMSE, and SAM of the images by the Ad CycleGAN and the original CycleGAN is significantly lower than the MSE and RMSE of those by VAE ($p < 0.01$), and the rest scores (PSNR, UIQI, SCC, VIF) of the images by Ad CycleGAN and CycleGAN are higher than those by VAE ($p < 0.01$). These results confirm that GAN including the new Ad

CycleGAN and the original CycleGAN produce synthetic malaria infected blood cell images with superior quality over those by VAE.

When we compare the images respectively by the new Ad CycleGAN and the original CycleGAN, we find that the synthetic images by Ad CycleGAN have comparable image quality with the original CycleGAN for image synthesis. Particularly, when the new Ad CycleGAN model perform image domain translation from normal blood cell images to malaria infected blood cell images, i.e., $X \rightarrow Y$ translation, the output has the lowest MSE and comparable effects in other qualitative metrics image quality with those simply doing identical translation within the same domain, i.e., $Y \rightarrow Y$ translation. In addition, 99.61% synthetic images by the Ad CycleGAN from $X \rightarrow Y$ translation are correctly classified, which is the highest accuracy compared to the outputs by the original CycleGAN. This finding is also supported by the optimization procession shown in Figure 32, where the criterion loss values are better convergent than those in the training of the original CycleGAN in the late stage of GAN optimization.

Finally, we need to explain the difference between the $X \rightarrow Y$ image translation and the $Y \rightarrow Y$ image augmentation. The former one is the optimization objective of the GAN to let the architecture to learn the mapping between two image domains. In this experiment, when the GAN receives real images of normal blood cell as inputs, it should produce synthetic images belonging to the malaria infected blood cells with both high fidelity and good diversity. This process covers both the image synthesis process and the image translation process. On the other hand, the $Y \rightarrow Y$ image augmentation only receives real images of malaria blood cells and then synthetize the images to the same image domain, i.e., also belonging to the malaria infected images. Thus, the $Y \rightarrow Y$ augmentation process only the image synthesis task. On the above experiments in this chapter, the new Ad CycleGAN model well perform both the image translation and augmentation given the pretrained criterion to ensure the output synthetic images falling into the target category.

Chapter 6 Ad CycleGAN for Radiologic Image Synthesis

In this chapter, we present the series of experiments on newly proposed Ad CycleGAN to synthesize radiologic images. Compared to histologic images as color images, radiologic images are considered as grey-scale images with complex visual patterns and texture presented by the pixel tensity. Chest radiography is perhaps one of the simplest radiologic images because the pixel intensity only presents the density of the human tissue when the X-Ray goes through the human body. The advantage of X-Ray images is the easy accessibility of the regular X-Ray equipment. The GAN research on X-Ray images is still beneficial for rare disease and emergency management. For example, when new disease and conditions occur, the GAN model can act as a reliable method for data augmentation for imbalanced medical image dataset. As we have discussed in previous chapters, using general purpose dataset such as ImageNet cannot well train the low-level filters of the DNNs to capture the visual significant patterns from medical images. New approach such as introducing augmented dataset with high fidelity synthetic images can be a feasible alternation to improve the DNN performance.

In our experiments in this chapter, we implement the Ad CycleGAN to synthesize COVID-19 positive chest X-ray images. Chest X-ray radiography with real-time polymerase chain reaction (RT-PCR) test are commonly used fast screening methods for coronavirus disease 2019 (COVID-19) since the start of the pandemic [304]. The COVID-19 positive cases have special bilateral or unilateral multiple mottling and ground-glass opacity patterns on the chest X-ray and CT images [305]. These patterns can be detected by deep neural network (DNN) image classifiers integrated to the compute-aided health systems for early screening. Since the RT-PCR usually takes a few hours for the result, using the DNN image detection method can separate the highly suspicious cases earlier and reduce the risk of secondary infection. Many successful studies on developing the DNN models either for COVID-19 chest X-ray or CT image detection have been published [306]. However, a recent study revealed the seemly high-performance DNN models for COVID-19 chest X-Ray image detection are vulnerable to network attacks4. One

constraint is that the DNNs are optimized by extremely imbalanced datasets where the COVID-19 images only occupy 5% to 6% of the total image samples [306]. Another drawback is that the specific medical image patterns are different from general-purposed images such as those in the ImageNet dataset. As the result, when using transfer learning with DNN models trained by the ImageNet to fine tune a new model for the radiography images, the pretrained feature extractors usually cannot effectively capture the medical significant patterns through the complex architecture but simply develop meaningless combinations for the final decision. These factors all contribute to the vulnerability of the current DNN models.

6.1 Material and Methods

As a new solution for the DNN training with imbalanced datasets, we use the Ad CycleGAN to generate synthetic COVID-19 chest X-ray images from normal images. Cycle GAN is the state-of-the-art conditional generative adversarial network (CGAN) for unpaired image to image translation. The new Ad CycleGAN uses a pretrained criterion to further control the synthetic images falling into the correct category (i.e., COVID-19 positive X-ray images). We use a COVID-19 image dataset acquired from the Kaggle COVID-19 Radiography Database:
https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database

It consists of 219 COVID-19 positive images, and 1,064 normal chest X-ray for the experiments described in this chapter. Figure 33 illustrates an example of chest X-ray images collection in the dataset.

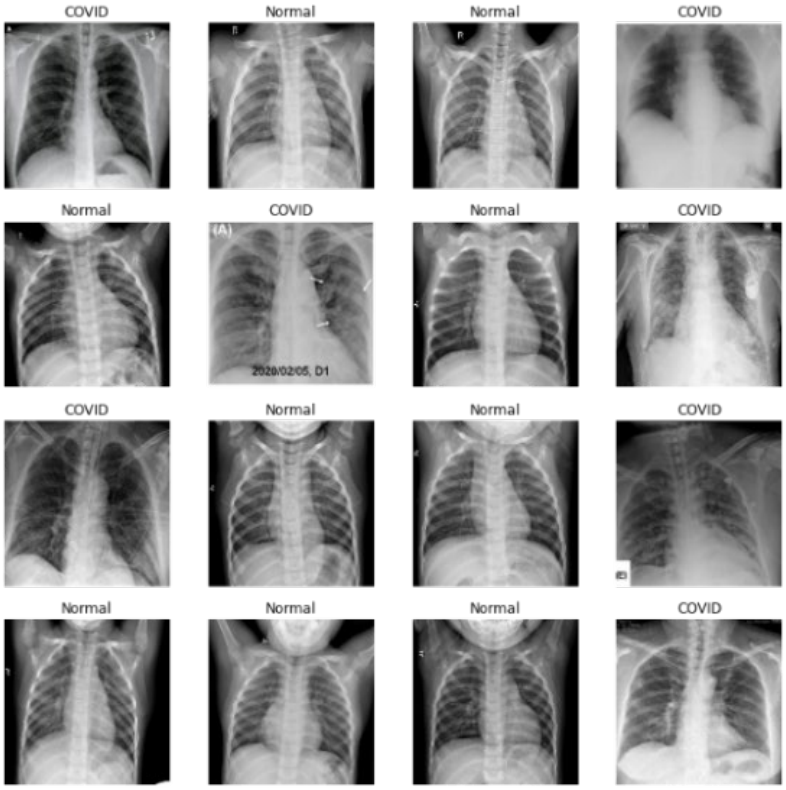


Figure 33: Original Chest X-ray Images.

Note that the original chest X-ray dataset contains three categories of images: normal chest X-ray images (labeled as Normal), Covid-19 positive chest X-ray images (labeled as COVID), and viral pneumonia chest X-ray images (labeled as Viral). Our task is to implement the generative models to synthesize COVID-19 positive images and to perform images translation from normal chest X-ray images to COVID-19 positive images, therefore, we discard the viral pneumonia images during model optimization. Three models are optimized in the experiments as we do in Chapter 5: a variant autoencoder (VAE), the Cycle-Consistent Adversarial Network (CycleGAN) and the newly proposed Adaptive Ad Cycle-Consistent Adversarial Network (Ad CycleGAN) with pretrained criterion.

Given the hardware condition, the images are resized to 64-by-64, 3 channel as the input dimensions. Because we have only 219 real COVID-19 X-ray images, 50 of the COVID-19 images are randomly selected and withheld for testing, the rest 169 real images

are duplicated 6 times to match 1,014 normal X-ray images for model optimization. The VAE, CycleGAN, and Ad CycleGAN models are respectively optimized by 600 epochs on the Google Colab platform with GPU. The average runtime is about 30 seconds per epoch with the mini-batch size of 64.

6.2 Experiment Results and Interpretation

The synthetic X-ray images generated by the trained VAE after trained by 600 epochs with the KL divergence weight of 0.01 are shown in Figure 34. It indicates that the VAE learns the overall contour of the thoracic cavity with clear landmarks of the important organs such as the heart, the lungs, and the diaphragm.

In addition, the VAE also performs image conversion. For example, some input images aligned on the sagittal plane will be automatically converted to coronal axis alignment as shown on the bottom right of Figure 34. However, the synthetic image quality by the VAE is low by human eyes, which also reflects in the quantitative metrics.

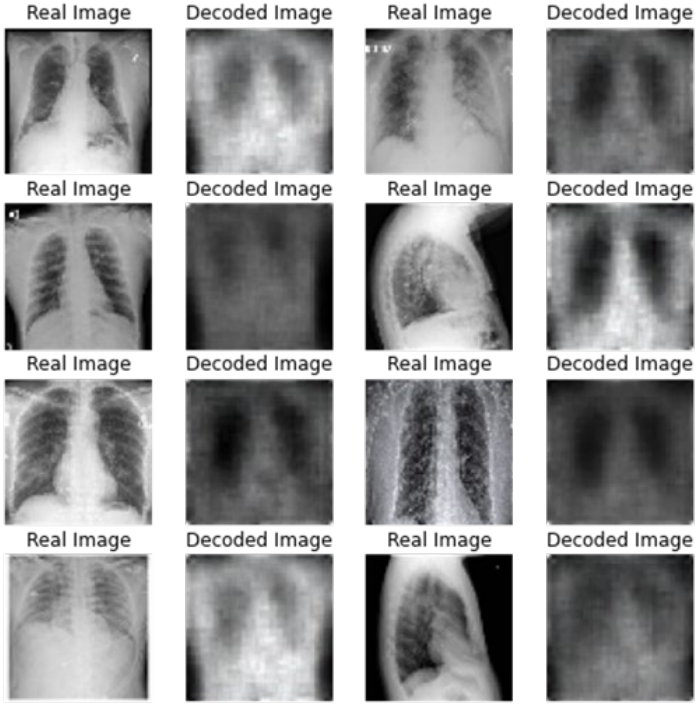


Figure 34: Synthetic COVID-19 Chest X-ray Images by VAE.

In the next experiments, we respectively optimized the CycleGAN and the Ad CycleGAN with similar parameter configurations as in Chapter 5, with $\lambda = 80.0$, σ =60.0, $\varphi = 0.1$, and $\kappa = 20$. It means that the criterion loss is added to the total generator loss term very 20 steps during the model optimization. The models are optimized by the Adam optimizer with the initial learning rate of 2×10^{-4} for 600 epochs. The mini-batch size is 64. The synthetic chest X-ray images respectively generated by the CycleGAN and by the Ad CycleGAN are shown in Figure 35 and Figure 36. The quantitative measures for the quality of the synthetic images are listed in Table 8, the FID score, and the classification accuracy to the due category of the synthetic images are listed in Table 9.

Figure 35: Synthetic Chest X-ray Images by CycleGAN.

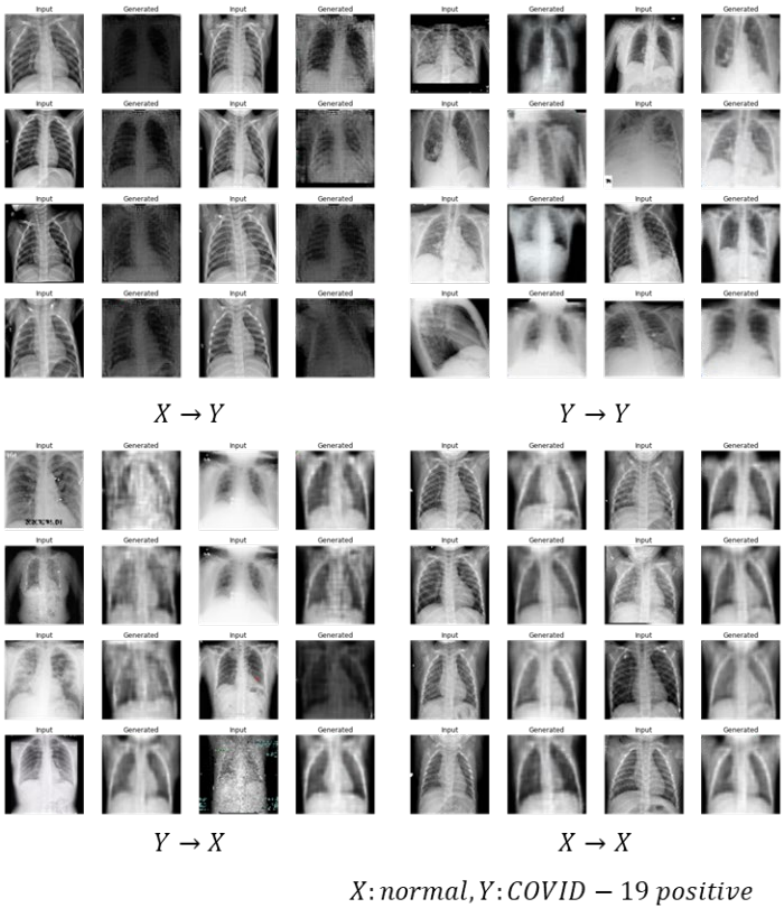


Figure 36: Synthetic Chest X-ray Images by Ad CycleGAN.

Image source	MSE (±std)	RMSE (±std)	PSNR (±std)	UIQI (±std)	SCC (±std)	SAM (±std)	VIF (±std)
VAE	5102.752 (4957.601)	63.103 (33.476)	13.414 (4.813)	0.715 (0.225)	0.003 (0.017)	0.185 (0.035)	0.189 (0.050)
CycleGAN ($X \rightarrow Y$)	3608.976 (1398.376)	58.776 (12.420)	12.976 (2.106)	0.816 (0.086)	0.052 (0.027)	0.386 (0.088)	0.120 (0.051)
CycleGAN ($Y \rightarrow Y$)	409.005 (495.043)	17.558 (10.035)	24.436 (4.398)	0.975 (0.029)	0.410 (0.086)	0.081 (0.038)	0.558 (0.050)
Ad CycleGAN ($X \rightarrow Y$)	3744.764 (1561.575)	59.969 (12.183)	12.747 (1.740)	0.813 (0.095)	0.017 (0.024)	0.422 (0.095)	0.075 (0.040)
Ad CycleGAN ($Y \rightarrow Y$)	435.849 (461.599)	18.731 (9.218)	23.590 (3.863)	0.973 (0.036)	0.443 (0.083)	0.093 (0.039)	0.525 (0.056)

Table 8: Quantitative Measure of the Chest X-ray Synthetic Images.

Image source	FID (±std)	Accuracy
VAE	1.568×10^{-3} (1.024×10^{-3})	0.0
CycleGAN ($X \rightarrow Y$)	5.267×10^{-5} (9.605×10^{-6})	0.9375
CycleGAN ($Y \rightarrow Y$)	6.311×10^{-4} (1.251×10^{-4})	1.0
Ad CycleGAN ($X \rightarrow Y$)	3.603×10^{-4} (6.446×10^{-5})	0.9531
Ad CycleGAN ($Y \rightarrow Y$)	1.191×10^{-5} (4.162×10^{-6})	1.0

Table 9: FID score of Classification Accuracy of the Chest X-ray Synthetic Images.

From Figure 35 and Figure 36, we observe that both the orginal CycleGAN and the new Ad CycleGAN can synthesize high quality chest X-ray images with superior visual fidelity and diversity over those generated by variant autoencoder (VAE). Another finding is that both CycleGAN and Ad CycleGAN not only perform the image synthesis / translation, but also convert the input images aligned on the sagittal axis to the synthetic images aligned on the coronal axis. When compared with the quantitative metrics, the synthetic images generated either by the CycleGAN or by the Ad CycleGAN have lower MSE and RMSE between the input real images and the output synthetic images than those by VAE. While comparing the image quality, the synthetic images through the image augmentation process (i.e., $Y \rightarrow Y$) have higher scores in the PSNR, UIQI, and VIF than those by VAE, which indicates the GANs have better capacity for image synthesis and augmentation. However, the synthetic images through image translation process (i.e., $X \rightarrow Y$) have lower scores and some of them even worse than VAE. It implies the GANs cannot translate high quality synthetic images probably due to insufficient training samples.

When we look at the FID measure, we find that the synthetic images by VAE have higher FID score than those by GANs. As we have discussed in Chapter 5, the FID score is a harmonious measure for both image fidelity and diversity. Lower FID score actually

implies the images generated by the GANs are more homogenous and are considered belonging to the same category by the pretrained DNN. When comparing the classification accuracy, the synthetic images by the Ad CycleGAN has higher accuracy of 95.31% than those by the original CycleGAN with the accuracy of 93.75% ($p < 0.05$). This result indicates that the criterion in the Ad CycleGAN architecture is effective to exert control on the synthetic images falling into the correct category. In general, the synthetic images through the image augmentation process (i.e., $Y \rightarrow Y$) have better quality than the images synthesized through the image translation process (i.e., $X \rightarrow Y$). It can be caused by the limit number of real COVID-19 images in the training dataset so that both the criterion and the Ad CycleGAN architecture cannot be thoroughly optimized.

Next, we compare the optimization process of the new Ad CycleGAN with the original CycleGAN. We add a periodic criterion loss to the total generator loss once every 20 steps during the optimization. At the top of Figure 37. we can observe a clear fluctuation from the beginning to approximate epoch 400, where the criterion loss acts as extra momentum for the discriminator loss. The discriminator loss of the Ad CycleGAN has a surging peak at about 430 epochs, and it is fortunate to reduce soon after a few epochs. It can be attributed to the decay factor that controls the influence of the criterion loss to a proper range. In comparison, the optimization of the CycleGAN is relatively smooth, where the loss value becomes stable after about 200 epochs of optimization.

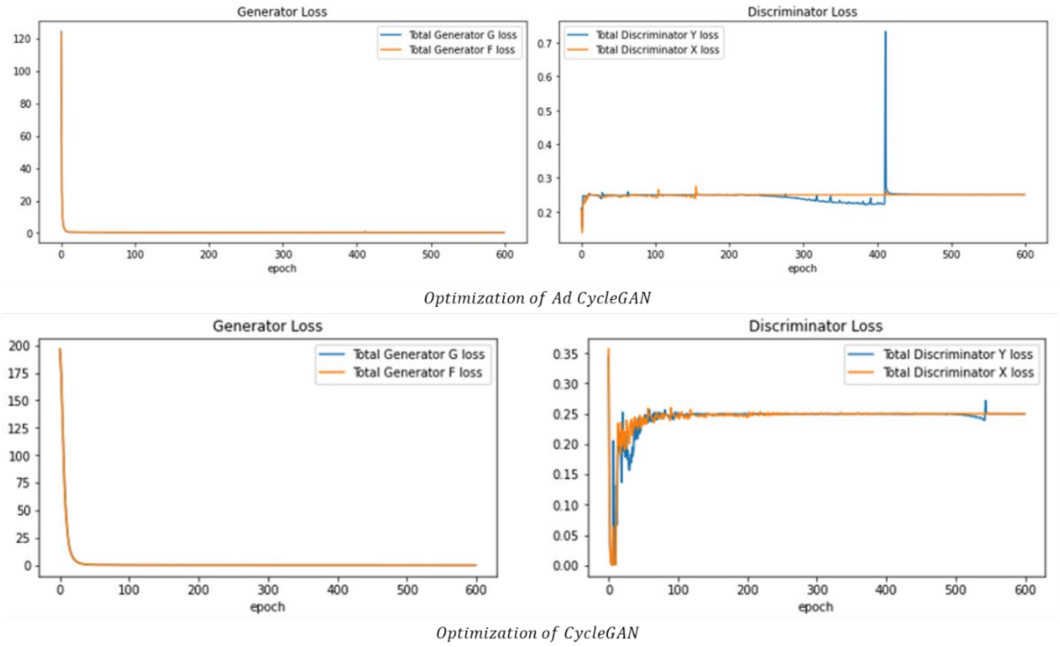


Figure 37: Optimization Process of Ad CycleGAN and CycleGAN for X-ray Images.

6.3 Summary

In this chapter, we present the experiment of Ad CycleGAN for COVID-19 positive chest X-ray synthesis and image translation between normal images and COVID-19 positive images. The results are compared with the synthetic images by the original CycleGAN and the VAE. The experiments are performed in the similar procedure as the experiments for synthesis malaria blood cell images in Chapter 5.

The experiment results have the same conclusion as in Chapter 5 that the GAN models produce higher quality synthetic images than VAE. However, the images synthesized from real COVID-19 X-ray images through the image augmentation process (i.e., $Y \rightarrow Y$) have better quality than those synthesized from normal X-ray images through the image translation process (i.e., $X \rightarrow Y$). This result is inconsistent with the findings from the experiments in Chapter 5. It can be explained by the insufficiency of real COVID-19 image samples in the training dataset and we have done some image replication to match the data sample number of the normal images.

On the other hand, the high accuracy of the synthetic images by the Ad CycleGAN model confirms that the adaptive criterion design can effectively control the image category given the context of how the criterion was optimized. The Ad CycleGAN can be considered as a new approach of conditional GAN which can extend the control power upon the synthetic image domain.

Chapter 7 Summaries

This thesis feasibility of applying the state-of-the-art DNN-based generative model for medical image synthesis. At the beginning, we introduce the research motivation contribution of our work. Then we introduce the basic knowledge of deep neural networks (DNNs), the concepts of multiple type of DNN variations particularly the DNNs for image processing. Following the introduction of DNN, we further explain the main applications of DNN for medical images, including medical significant pattern detection and recognition, image segmentation, image registration and automatic alignment, computer-aided diagnosis and medical decision making, image retrieval, physical stimulation, and image reconstruction. Based on understanding DNN for medical use, we move to the topics of generative adversarial network, or GAN, different type of GAN models, and the main tasks that GAN perform in medical research.

After the introduction of the background knowledge, we focus on the details of GAN for medical image synthesis. The GANs for image synthesis or translation nowadays include Pix2Pix and CycleGAN, or any customized GANs with similar architecture. In order to analyze the research progress of GAN for medical image applications, we conduct a quantitative survey on GAN for medical image processing where 165 highly relevant research papers are collected and categorized into five content topics: image reconstruction and enhancement, image synthesis and augmentation, image translation, image segmentation, and other medical applications. The survey provides an outline of GAN research since the year 2017. It concludes that there are great achievements on GAN in the above five research aspects, and it also finds that there is no similar research on using adaptive pretrained criterion combined with CycleGAN to control the synthetic images falling into the desired category. Unlike the general-purpose image synthesis and translation, the difference between normal medical images and images with significant diagnostic patterns is so trivial that it is difficult to be captured by GANs. Thus, the newly proposed Ad CycleGAN architecture has the potential to extend the GAN medical applications to a new domain. This is the novelty and main contribution of this work to the

research community and to the industry in the future. Two experiments are then performed to compare the performance of Ad CycleGAN with both the original CycleGAN and variant autoencoder (VAE). The details of the experiments and the results interpretations are discussed in Chapter 5 and Chapter 6.

7.1 Practical Impact of the Proposed Approaches

The Ad CycleGAN is an improved architecture based on the CycleGAN. The Ad CycleGAN follows the GAN optimization strategy originated from the Wasserstein GAN (WGAN) [28], where the objective function is not to estimate the probability of the synthetic images considered as real images, but to "rate" the synthetic images as a critic. Under this framework, the GAN optimization process can be combined with the opinions by multiple critics from different aspects, thus the zero-sum adversarial game rule proposed by Goodfellow et al. [7] has been changed to a multi-domain task. Furthermore, the Ad CycleGAN does not need to encode the labels into the training data therefore it simplifies the computation runtime for GAN optimizations. In our experiments in Chapter 5 and Chapter 6, a single training epoch of the GAN optimization takes less than one minute, and the Ad CycleGAN models can produce high quality synthetic images compared to the VAE. The most significant merit of the Ad CycleGAN is that it improves the image classification accuracy to the target image category. In our literature review, most of the applications of GAN is to use this generative model for image data augmentation, but there is no guarantee that the generated images falling into the correct category. Therefore, the introduction of the pretrained criterion becomes a unique impact of the Ad CycleGAN to the GAN based architectures.

On the other hand, the Ad CycleGAN can perform both image augmentation and image translation. Image augmentation means the input real images belongs to the same category as the expect synthetic outputs, e.g., from normal images to normal images with acceptable diversity, or from disease positive images to disease positive images with acceptable diversity. From the quantitative survey, we find that most of the GAN studies on medical images are focusing on image augmentation. The GAN models generate

multiple synthetic samples including synthetic images for direct DNN optimization, image mask for improving image segmentation, and image feature maps for medical diagnosis and decision making. The applications of image translation are mainly for converting the images from one format to another format, such as from MR images to CT images, from ultrasound images to CT images, or between two histological dyeing methods. However, all the available applications have not explored the task of converting images from the normal / healthy domain to a specific disease domain, which is crucial in medical research. The experiment result described in Chapter 5 concludes that the Ad CycleGAN can synthesize high quality malaria infected blood cell images which are superior to those synthesized by original CycleGAN and the VAE generative model. The malaria infected blood cell images can be regarded as normal blood cell with a malaria plasmodium inside, or the visual patterns of normal blood cell plus the malaria patterns. Therefore, the images from both domains are considered homogenous as human blood cells which are likely to be classified to the identical category by some general-purpose models. This feature is confirmed by the low FID score of the synthetic images. The experiment result described in Chapter 6 on the other hand, shows the Ad CyleGAN can only translate better quality images than the VAE, but they are not as good as those by the image augmentation path (i.e., $Y \rightarrow Y$). This outcome is confirmed by all the quantitative metrics where the output synthetic images through image augmentation (i.e., $Y \rightarrow Y$) by either Ad CycleGAN or original CycleGAN are in better quality. However, the synthetic images through the image translation path (i.e., $X \rightarrow Y$) has higher classification accuracy than those translated by the original CycleGAN.

The findings in the experiments and the discussion above all indicates than the proposed Ad CycleGAN can well perform the medical image translation tasks. This unique feature is hopefully to solve the common class imbalance issues because the medical images containing rare or new disease information are both difficult to acquire and expensive for expert annotation. A typical example is the COVID-19 pandemic, when a large amount of GAN based studies are published since 2020. The successful applications

of GAN for COVID-19 pattern detection and segmentation shows the feasibility to widely use DNN for computer assisted disease diagnosis and public health management.

7.2 Study Limitations

Though the experiment results presented in Chapter 5 and Chapter 6 indicates that the proposed Ad CycleGAN can effectively synthesize medical images to the designated categories, this new model still has some limitations.

First, the optimization of Ad CycleGAN is computationally intensive. The development of Ad CycleGAN requires good hardware support such as high-performance GPU and RAM. In the experiments described above, a single optimization of an Ad CycleGAN model need at least 8 to 10 hours even with GPU support. The requirement of RAM depends on the size of the training dataset, but the minimum requirement for RAM is at least 10 GB. Since the optimization is mainly on the cloud platforms such as Google Cloud Platform (GCP) and Amazon Web Service (AWS), and the requirement of RAM usually increments as it progresses, there is always the risk of environment crash during the optimization.

Second, the tuning of the total loss function for the Ad CycleGAN is intuitive and experience based. Efforts to develop a mathematically explainable algorithm to dynamically adjust the corresponding weights for different terms of the total loss is needed to further improve the robustness of the optimization procedure.

Third, the performance of the Ad CycleGAN is highly relied on the size of total training samples. It is verified by the experiment results where the Ad CycleGAN for blood cell image synthesis is much better than the chest X-Ray images synthesis, because the training dataset of blood cell images is much larger.

The above limitations are to be improved by future work.

7.3 Summary of Ad CycleGAN

The proposed adaptive cycle-consistent adversarial network, or Ad CycleGAN in this thesis is a new extension of the cycle-consistent adversarial network (CycleGAN).

CycleGAN is one type of the conditional GANs provides state-of-the-art image translation between two image domains. The advantage of CycleGAN is that it does not require paired images to learn the mapping between two fixed images like in the Pix2Pix architecture. In other words, it provides the flexibility for capturing the images patterns randomly. Another advantage of the CycleGAN architecture is that the model does not need to synthesize images from a latent space of random distribution but based on the sample distribution of real images. This feature is particularly helpful for medical images.

On the other hand, the normal medical images and the images containing disease or pathological patterns can be considered homogenous. In other words, the disease images can be interpreted as normal image plus disease specific patterns. The original design of the CycleGAN model uses the adversarial loss, the cycle loss and the identity loss as the objectives to respectively estimate the difference of the two images domains, the image shape and contour, and the image colors. It is effective for general images where the visual difference between the two domains is obvious. However, the homogenesis makes the difference between two medical image domains so close that the original CycleGAN model will suffer from insufficient gradients or gradient vanishing problem during the optimization. The new Ad CycleGAN model can effectively solve this problem as the new source of loss objective functions are added. In our experience for Ad CycleGAN, two criterion loss terms are introduced and combined as the total criterion loss. The total criterion loss will be added to the total adversarial loss for GAN optimization. The idea of the external criterion loss originates from the Wasserstein metric introduced by Arjovsky M et al. with the Wasserstein GAN or WGAN architecture. The distance between the probability distributions of the image domains can be measured by the earth mover's distance (EMD). Statistically, the distributions of the two image domains can be represented by two clusters of points p_i and q_j. The distance between the two clusters $\{p_i\}$ and $\{q_j\}$ over the region D defined by $\mathbb{R}^d$. With the earth mover's language, we can interpret the distributions as two ways of piling up a certain amount of earth over the region D. The EMD is the optimal way to move one pile of the earth to another pile (i.e.,

converting the distribution of $\{p_i\}$ to $\{q_j\}$ through D or vice versa). The EMD is defined as the distance magnitude between the two image domains normalized by the total flow F:

$$EMD(P,Q) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j}d_{i,j}}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j}} = inf_{\gamma\in\Pi(P,Q)}\mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|] \quad (7.1)$$

where $f_{i,j}$ is the flow between p_i and q_j, and $d_{i,j}$ is the ground distance between p_i and q_j. $\Pi(\mathrm{P},\mathrm{Q})$ is the set of all joint distributions with the marginals P and Q. The introduction of EMD as the objective, the GAN optimization process is converted to a linear optimization problem as:

$$argmin_F \sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j}d_{i,j} \quad (7.2)$$

where the task of the GAN discriminators is to act as a critique to evaluate the quality of the generated images by referring to the real images. Following this idea, we can easily add multiple sources of critics to the total EMD to control the parameters of the joint distribution $\Pi(\mathrm{P},\mathrm{Q})$.

The Ad CycleGAN is based on the above strategy by adding a pretrained criterion C, where is a pretrained DNN to evaluate whether the synthetic images belonging to the due category. The EMD objective provides the flexibility to congregate multiple critics to evaluate not only the quality but also the context of the images given any practical settings. The experiments described in Chapter 5 and Chapter 6 both confirm that the proposed Ad CycleGAN with external pretrained criterion can improve the classification accuracy of the synthetic images to satisfy the individualized requirements imposed by clinical circumstances. Therefore, we conclude that the proposed Ad CycleGAN architecture provides an ideal solution for medical images synthesis and image translation. It is particularly helpful to improve the performance of DNN for screening, automatic detection, and computer aided diagnosis and decision making for rare and new disease when the relevant image data are difficult to collect and annotated.

Chapter 8 Conclusions and Future Work

This thesis presents the state-of-the-art research on generative adversarial network (GAN) on medical image processing. It proposed the adaptive cycle-consistent adversarial network, or Ad CycleGAN as a new GAN architecture for medical image synthesis and image translation. The Ad CycleGAN is an extension of the cycle-consistent adversarial network (CycleGAN), which is commonly used GAN architecture to perform image synthesis and image translation between unpaired images. The original CycleGAN can synthesize high quality images for general purposes, however, it becomes unreliable to synthesize medical images with significant diagnostic patterns because of the complexity of medical diagnosis. The new Ad CycleGAN significantly improves the synthetic accuracy of disease specific medical images from normal medical images by a pretrained external criterion to exert extra gradient to break the adversarial equilibrium during the GAN optimization. When the weight of the criterion is properly controlled by the factors such as the periodic factor and decay factor during the GAN optimization, the new Ad CycleGAN model can generate high quality synthetic medical images in multiple formats with higher accuracy compared to the other state-of-the-art generative DNN models such as CycleGAN and VAE.

The future work on Ad CycleGAN has two folds. First, we need to further improve the EMD objective to ensure more control on the optimization process and minimize the side effects on the external criterion from synthesizing high quality images like reducing the occurrence of artifacts on the synthetic images. More extra criterion can also be added to the EMD to further control the characteristics of the generated images to the due domain. Second, we can develop more sophisticated GAN architecture to extend the Ad CycleGAN design for more tasks. For example, we can combine Ad CycleGAN with StyleGAN [307] to improve the synthetic image resolution alongside with image translation process. The Ad CycleGAN can be optimized with the Pix2Pix architecture to precisely allocate the location of the synthetic patterns. Therefore, the application of the Ad CycleGAN can be extended to medical image segmentation.

In addition, we also find some promising research topic GAN for medical usage. For example, Sarrut D et al. [161] and Zhan B et al. [210] respectively used GAN models based on the CT images to estimate the optimal dose of radiotherapy to individual patients. Qiang N et al. [196] and Matsui T et al. [253] used the GAN models to generate MR image features that facilitate diagnosis. Chen MT et al. used GAN generated histological image features to estimate optical properties [255]. Vu QD et al. used GAN generated histological image features as parameters to quantify cancer tissue characteristics [259]. Das A et al. applied GAN generated histological image features to estimate breast cancer prognosis [261]. All these attempts open new perspectives to further apply GAN models to improve medical practices.

Except for the applications to medical images, the GAN models are widely applied to analyze medical sequential data. The typical applications include synthesizing electrocardiogram (ECG) and electroencephalogram (EEG) data [277-280], and synthesizing genomic or proteomic sequence [281, 293, 294]. These types of research help to exploration the new domains of GAN applications but they are beyond the discussion of this thesis. Table 6 summaries the GAN applications mentioned in this section. Table 6 summarise these GAN studies.

In conclusion, GAN provides a promising solution for the data greedy feature of deep neural network. The proposed Ad CycleGAN model provides more authentic image to augment the training of DNN with high performance and robust. We believe this new technology will promote DNN related technologies for medical diagnosis and decision making, and it will ultimately help to enhance high-quality healthcare delivery.

Bibliography

[1] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015 May; 521: 436-444.

[2] Recht B, Roelofs R, Schmidt L, Shankar V. Do imagenet classifiers generalize to imagenet?. In Proceedings of International Conference on Machine Learning. 2019 May; PMLR. 5389-5400.

[3] Liang Z, Powell A, Ersoy I, Poostchi M, Silamut K, Palaniappan K, Guo P, Hossain MA, Sameer A, Maude RJ, Huang JX. CNN-based image analysis for malaria diagnosis. In Proceedings of 2016 IEEE international conference on bioinformatics and biomedicine (BIBM). 2016; pp.493-496.

[4] Hirano H, Koga K, Takemoto K. Vulnerability of deep neural networks for detecting COVID-19 cases from chest X-ray images to universal adversarial attacks. PLoS One. 2020; 15(12): e0243963.

[5] Maier A, Syben C, Lasser T, Riess C. A gentle introduction to deep learning in medical image processing. Zeitschrift für Medizinische Physik. 2019; pii: S0939-3889(18)30120-X.

[6] Shickel B, Tighe PJ, Bihorac A, Rashidi P. Deep EHR: a survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. IEEE journal of biomedical and health informatics. 2018; 22(5):1589-604.

[7] Goodfellow I, Bengio Y, Courville A. Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[8] Domingos P. A few useful things to know about machine learning. Communications of the ACM. 2012; 55(10): 78-87.

[9] Sonoda S, Murata N. Neural network with unbounded activation functions is universal approximator. Applied and Computational Harmonic Analysis. 2017; 43(2): 233-268.

[10] Rockafellar R. Convex analysis, Princeton landmarks in mathematics and physics. Princeton University Press; 1970.

[11] Ding S, Li H, Su C, Yu J, Jin F. Evolutionary artificial neural networks: a review. Artificial Intelligence Review. 2013; 39(3): 251-60.

[12] Dauphin YN, Pascanu R, Gulcehre C, Cho K, Ganguli S, Bengio Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. Advances in neural information processing systems. 2014; 2933-2941.

[13] Rosenblatt F. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory; 1957.

[14] Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research. 2011; 12: 2121-59.

[15] Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. 2014 Dec 22.

[16] Bertsekas DP, Scientific A. Convex optimization algorithms. Athena Scientific Belmont; 2015.

[17] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems. 2012; 1097-1105.

[18] Al-Jawfi R. Handwriting Arabic character recognition LeNet using neural network. Int. Arab J. Inf. Technol. 2009; 6(3): 304-9.

[19] Lin M, Chen Q, Yan S. Network in network. arXiv preprint arXiv:1312.4400. 2013 Dec 16.

[20] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2015; pp. 1-9.

[21] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2016; pp. 770-778.

[22] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998; 86(11): 2278-324.

[23] Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L. Imagenet: A large-scale hierarchical image database. In Proceedings of 2009 IEEE conference on computer vision and pattern recognition. 2009 Jun 20; pp. 248-255.

[23] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556. 2014.

[25] Vincent P, Larochelle H, Bengio Y, Manzagol PA. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on machine learning, ACM. 2008; pp. 1096-1103.
[26] Holden D, Saito J, Komura T, Joyce T. Learning motion manifolds with convolutional autoencoders. SIGGRAPH Asia 2015 Technical Briefs, ACM. 2015; 18.
[27] Vincent P, Larochelle H, Bengio Y, Manzagol PA. Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research. 2010; 3371-3408.
[28] Arjovsky M, Chintala S, Bottou L. Wasserstein generative adversarial networks. In Proceedings of International conference on machine learning. 2017; pp. 214–23.
[29] Gauthier J. Conditional generative adversarial nets for convolutional face generation. In Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester 2014; 2014: 2.
[30] Zhu JY, Park T, Isola P, Efros AA. Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE international conference on computer vision. 2017; pp. 2223-2232.
[31] Mandic DP, Chambers J. Recurrent neural networks for prediction: learning algorithms, architectures and stability. John Wiley & Sons, Inc.; 2001.
[32] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997; 9: 1735–80.
[33] Chung J, Gulcehre C, Cho K, Bengio Y. Gated feedback recurrent neural networks. In Proceedings of International conference on machine learning. 2015 Jun; pp.2067-2075.
[34] Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation. In Proceedings of International Conference on Medical image computing and computer-assisted intervention. 2015 Oct 5; pp. 234-241.
[35] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2015; pp. 3431-3440.

[36] Isola P, Zhu JY, Zhou T, Efros AA. Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2017; pp. 1125-1134.

[37] Frid-Adar M, Diamant I, Klang E, Amitai M, Goldberger J, Greenspan H. GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. Neurocomputing. 2018 Dec 10; 321: 321-31.

[38] Maier A, Schebesch F, Syben C, Würfl T, Steidl S, Choi J-H, et al. Precision learning: towards use of known operators in neural networks. In: Tan JKT, editor. In Proceedings of 24th International Conference on Pattern Recognition (ICPR). 2018; pp. 183–188.

[39] Yuan X, He P, Zhu Q, Bhat RR, Li X. Adversarial examples: attacks and defenses for deep learning; 2017 arXiv:1712.07107.

[40] Brown TB, Mané D, Roy A, Abadi M, Gilmer J. Adversarial patch; 2017; arXiv:1712.09665.

[41] Borji A. Pros and cons of gan evaluation measures. Computer Vision and Image Understanding. 2019; 179: 41-65.

[42] Salimans T, Goodfellow I, Zaremba W, Cheung V, Radford A, Chen X. Improved techniques for training gans. In Proceedings of Advances in neural information processing systems. 2016; pp. 2234-2242.

[43] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2016; pp. 2818-2826.

[44] Zheng Y, Comaniciu D. Marginal space learning. In Proceedings of Marginal space learning for medical image analysis. Springer; 2014; 25–65.

[45] Girshick R, Donahue J, Darrell T, Malik J. Region-based convolutional networks for accurate object detection and segmentation. IEEE transactions on pattern analysis and machine intelligence. 2015 May 25; 38(1): 142-58.

[46] Girshick R. Fast r-cnn. In Proceedings of the IEEE international conference on computer vision. 2015; pp. 1440-1448.

[47] Ren S, He K, Girshick R, Sun J. Faster r-cnn: Towards real-time object detection with region proposal networks. In Proceedings of Advances in neural information processing systems. 2015; pp. 91-99.
[48] Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2016; pp. 779-788.
[49] Bier B, Unberath M, Zaech JN, Fotouhi J, Armand M, Osgood G, Navab N, Maier A. X-ray-transform invariant anatomical landmark detection for pelvic trauma surgery. In Proceedings of Medical Image Computing and Computer Assisted Intervention (MICCAI) 2018. Cham: Springer International Publishing; 2018. pp. 55–63.
[50] Akselrod-Ballin A, Karlinsky L, Alpert S, Hasoul S, Ben-Ari R,Barkan E. A region based convolutional network for tumor detection and classification in breast mammography. In Proceedings of Deep learning and data labeling for medical applications. Springer; 2016. pp. 197–205.
[51] Aubreville M, Stoeve M, Oetter N, Goncalves M, Knipfer C, Neumann H, Bohr C, Stelzle F, Maier A. Deep learning-based detection of motion artifacts in probe-based confocal laser endomicroscopy images. Int J Comput Assist Radiol Surg. 2019; 14(1): 31-42.
[52] Xing F, Xie Y, Su H, Liu F, Yang L. Deep learning in microscopy image analysis: A survey. IEEE Transactions on Neural Networks and Learning Systems. 2017; 29(10): 4550-68.
[53] Gurcan MN, Boucheron LE, Can A, Madabushi A, Rajpoot NM, Yener B. Histopathological image analysis: A review. IEEE Rev. Biomed. Eng. 2009; 2: 147–171.
[54] McCann MT, Ozolek JA, Castro CA, Parvin B, Kovacevic J. Automated histology analysis: Opportunities for signal processing. IEEE Signal Process. Mag. 2015; 32(1): 78–87.
[55] Roth HR, Lu L, Farag A, Shin HC, Liu J, Turkbey EB, Summers RM. DeepOrgan: multi-level deep convolutional networks for automated pancreas segmentation. In

Proceedings of International conference on medical image computing, computer-assisted intervention. Springer. 2015; pp. 556–564.
[56] Ghesu FC, Krubasik E, Georgescu B, Singh V, Zheng Y, Hornegger J, et al. Marginal space deep learning: efficient architecture for volumetric image parsing. IEEE Trans Med Imaging. 2016; 35: 1217–28.
[57] Moeskops P, Viergever MA, Mendrik AM, de Vries LS, BendersMJ, Iˇsgum I. Automatic segmentation of MR brain images with a convolutional neural network. IEEE Trans Med Imaging. 2016; 35: 1252-61.
[58] Chen S, Zhong X, Hu S, Dorn S, Kachelrieß M, Lell M, Maier A. Automatic multi-organ segmentation in dual-energy CT (DECT) with dedicated 3D fully convolutional DECT networks. Med Phys. 2020; 47(2): 552-562.
[59] Nirschl JJ, Janowczyk A, Peyster EG, Frank R, Margulies KB, Feldman MD, Madabhushi A. Deep learning tissue segmentation in cardiac histopathology images. Deep learning for medical image analysis. 2017; pp. 179-195.
[60] Middleton I, Damper RI. Segmentation of magnetic resonance images using a combination of neural networks and active contour models. Med Eng Phys 2004; 26: 71–86.
[61] Fu W, Breininger K, Würfl T, Ravikumar N, Schaffert R, Maier A. Frangi-Net: a neural network approach to vessel segmentation. Bildverarbeitung für die Medizin. 2018; pp. 341–6.
[62] Andermatt S, Pezold S, Cattin P. Multi-dimensional gated recurrent units for the segmentation of biomedical 3D-data. In Proceedings of Deep learning and data labeling for medical applications. Springer; 2016 Oct 21. pp. 142–51.
[63] Peng H, Chung P, Long F, Qu L, Jenett A, Seeds AM, Myers EW, Simpson JH. Brain Aligner: 3D registration atlases of Drosophila brains. Nature methods. 2011; 8(6): 493-8.
[64] Wu G, Kim M, Wang Q, Munsell BC, Shen D. Scalable high-performance image registration framework by unsupervised deep feature representations learning. IEEE Trans Biomed Eng. 2016; 63:1505–16.

[65] Schaffert R, Wang J, Fischer P, Borsdorf A, Maier A. Metric-driven learning of correspondence weighting for 2-D/3-D image registration. In Proceedings of German Conference on Pattern Recognition (GCPR). Springer. 2018 Oct 9. pp. 140-152.

[66] Miao S, Wang JZ, Liao R. Convolutional neural networks for robust and real-time 2-D/3-D registration. In Proceedings of Deep learning for medical image analysis. 2017 Jan 1. pp. 271-296.

[67] Yang X, Kwitt R, Styner M, Niethammer M. Quicksilver: fast predictive image registration – a deep learning approach. Neuroimage. 2017; 158: 378–396.

[68] Liao R, Miao S, de Tournemire P, Grbic S, Kamen A, Mansi T, Comaniciu D. An artificial agent for robust image registration. In Proceedings of the AAAI conference on artificial intelligence. 2017 Feb 12; 31(1): 4168–75.

[69] Krebs J, Mansi T, Delingette H, Zhang L, Ghesu FC, Miao S, Maier AK, Ayache N, Liao R, Kamen A. Robust non-rigid registration through agent-based action learning. In Proceedings of Medical Image Computing and Computer-Assisted Intervention. Springer. 2017 Sep 10; pp. 344–52.

[70] Anwar SM, Majid M, Qayyum A, Awais M, Alnowami M, Khan MK. Medical image analysis using convolutional neural networks: a review. Journal of medical systems. 2018; 42(11): 226.

[71] Diamant I, Bar Y, Geva O, Wolf L, Zimmerman G, Lieberman S, Konen E, Greenspan H. Chest radiograph pathology categorization via transfer learning. In Proceedings of Deep learning for medical image analysis. Elsevier. 2017 Jan 1; pp. 299–320.

[72] Antropova N, Abe H, Giger ML. Use of clinical MRI maximum intensity projections for improved breast lesion classification with deep convolutional neural networks. J Med Imaging (Bellingham). 2018; 5: 014503.

[73] Ma HY, Zhou Z, Wu S, Wan YL, Tsui PH. A computer-aided diagnosis scheme for detection of fatty liver in vivo based on ultrasound kurtosis imaging. J. Med. Syst. 2016; 40(1):33.

[74] Mosquera-Lopez C, Agaian S, Velez-Hoyos A, Thompson I. Computer-aided prostate cancer diagnosis from digitized histopathology: a review on texture-based systems. IEEE reviews in biomedical engineering. 2014; 8: 98-113.
[75] Remeseiro B, Mosquera A, Penedo MG. CASDES: a computer-aided system to support dry eye diagnosis based on tear film maps. IEEE journal of biomedical and health informatics. 2015; 20(3): 936-43.
[76] Torrents-Barrena J, Lazar P, Jayapathy R, Rathnam MR, Mohandhas B, Puig D. Complex wavelet algorithm for computer-aided diagnosis of Alzheimer's disease. Electronics Letters. 2015; 51(20): 1566-8.
[77] Saha M, Mukherjee R, Chakraborty C. Computer-aided diagnosis of breast cancer using cytological images: A systematic review. Tissue and Cell. 2016; 48(5): 461-74.
[78] Salam AA, Akram MU, Wazir K, Anwar SM, Majid M. Autonomous Glaucoma detection from fundus image using cup to disc ratio and hybrid features. In Proceedings of 2015 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT). 2015 Dec 7; pp. 370-374.
[79] Salam AA, Akram MU, Abbas S, Anwar SM. Optic disc localization using local vessel based features and support vector machine. In Proceedings of 2015 IEEE 15th International Conference on Bioinformatics and Bioengineering (BIBE). 2015 Nov 2; pp. 1-6.
[80] Altaf T, Anwar SM, Gul N, Majeed MN, Majid M. Multi-class Alzheimer's disease classification using image and clinical features. Biomedical Signal Processing and Control. 2018; 43: 64-74.
[81] Zhang E, Seiler S, Chen M, Lu W, Gu X. BIRADS features-oriented semi-supervised deep learning for breast ultrasound computer-aided diagnosis. Phys Med Biol. 2020; 65(12): 125005.
[82] Cao Z, Duan L, Yang G, Yue T, Chen Q. An experimental study on breast lesion detection and classification from ultrasound images using deep learning architectures. BMC medical imaging. 2019; 19(1): 51.
[83] Fujima N, Andreu-Arasa VC, Meibom SK, Mercier GA, Truong MT, Sakai O. Prediction of the human papillomavirus status in patients with oropharyngeal squamous

cell carcinoma by FDG-PET imaging dataset using deep learning analysis: A hypothesis-generating study. European Journal of Radiology. 2020; 126: 108936.

[84] Mizotin M, Benois-Pineau J, Allard M, Catheline G. Feature-based brain MRI retrieval for Alzheimer disease diagnosis. In Proceedings of the 19th IEEE International Conference on Image Processing. 2012 Sep 30; pp. 1241-1244.

[85] Jiji GW, Raj PSJD. Content-based image retrieval in dermatology using intelligent technique. IET Image Processing. 2015; 9(4): 306-317.

[86] Rahman MM, Antani SK, Thoma GR. A learning-based similarity fusion and filtering approach for biomedical image retrieval using SVM classification and relevance feedback. IEEE Trans. Inf. Technol. Biomed. 2011; 15 (4): 640-646.

[87] Zhang F, Song Y, Cai W, Hauptmann AG, Liu S, Pujol S, Kikinis R, Fulham MJ, Feng DD, Chen M. Dictionary pruning with visual word significance for medical image retrieval. Neurocomputing. 2016; 177: 75-88.

[88] Anthimopoulos M, Christodoulidis S, Ebner L, Christe A, Mougiakakou S. Lung pattern classification for interstitial lung diseases using a deep convolutional neural network. IEEE transactions on medical imaging. 2016; 35(5): 1207-16.

[89] van Tulder G, de Bruijne M. Combining generative and discriminative representation learning for lung CT analysis with convolutional restricted boltzmann machines. IEEE transactions on medical imaging. 2016; 35(5): 1262-72.

[90] Brahmi D, Ziou D. Improving CBIR systems by integrating semantic features. In Proceedings of First Canadian Conference on Computer and Robot Vision, 2004. IEEE. 2004 May 17; pp. 233-240.

[91] Wu J, Yildirim I, Lim JJ, Freeman B, Tenenbaum J. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In Proceedings of Advances in neural information processing systems. 2015; pp. 127-135.

[92] Chu M, Thuerey N. Data-driven synthesis of smoke flows with CNN-based feature descriptors. ACM Transactions on Graphics (TOG). 2017; 36(4): 1-4.

[93] Meister F, Passerini T, Mihalef V, Tuysuzoglu A, Maier A, Mansi T. Towards fast biomechanical modeling of soft tissue using neural networks. In Proceedings of Medical

Imaging workshop at 32nd conference on Neural Information Processing Systems (NeurIPS), 2018 Dec: arXiv-1812.
[94] Maier J, Berker Y, Sawall S, Kachelrieß M. Deep scatter estimation (DSE): feasibility of using a deep convolutional neural network for real-time x-ray scatter prediction in cone-beam CT. In Proceedings of Medical imaging 2018: physics of medical imaging. 2018 Mar 9; vol. 10573, pp. 393-398.
[95] Unberath M, Zaech JN, Lee SC, Bier B, Fotouhi J, Armand M, Navab N. DeepDRR–a catalyst for machine learning in fluoroscopy-guided procedures. In Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention. 2018 Sep 16; pp. 98-106.
[96] Horger F, Würfl T, Christlein V, Maier A. Towards Arbitrary Noise Augmentation-Deep Learning for Sampling from Arbitrary Probability Distributions. In Proceedings of International Workshop on Machine Learning for Medical Image Reconstruction. 2018 Sep 16; pp. 129-137.
[97] Han X. MR-based synthetic CT generation using a deep convolutional neural network method. Medical physics. 2017; 44(4): 1408-19.
[98] Stimpel B, Syben C, Würfl T, Mentl K, Dörfler A, Maier A. MR to X-ray projection image synthesis. In Proceedings of the 5th international conference on image formation in X-ray computed tomography (CT-meeting). 2018. pp. 435-438.
[99] Schiffers F, Yu Z, Arguin S, Maier A, Ren Q. Synthetic fundus fluorescein angiography using deep neural networks. In Proceedings of Bildverarbeitung für die Medizin 2018. Berlin, Heidelberg. Springer. 2018. pp. 234-238.
[100] Cohen JP, Luck M, Honari S. Distribution matching losses can hallucinate features in medical image translation. In Proceedings of International Conference on Medical Image Computing and Computer-assisted Intervention. Springer. 2018 Sep 16; pp.529-536.
[101] Herman GT, Fundamentals of computerized tomography: Image reconstruction from projection, 2nd edition, Springer, 2009, pp. 12-15.
[102] Adler J, Öktem O. Learned Primal-dual Reconstruction. IEEE Transactions on Medical Imaging. 2018; 37(6): 1322-1332.

[103] Hammernik K, Klatzer T, Kobler E, Recht MP, Sodickson DK, Pock T, Knoll F. Learning a variational network for reconstruction of accelerated MRI data. Magnetic resonance in medicine. 2018; 79(6): 3055-3071.

[104] McCann MT, Jin KH, Unser M. A review of convolutional neural networks for inverse problems in imaging. 2017; arXiv:1710.04011.

[105] Zhang Z, Liang X, Dong X, Xie Y, Cao G. A sparse-view CT reconstruction method based on combination of DenseNet and deconvolution. IEEE Trans Med Imaging 2018; 37(6):1407-1417.

[106] Kofler A, Haltmeier M, Kolbitsch C, Kachelrieß M, Dewey M. A U-Nets cascade for sparse view computed tomography. In Proceedings of International workshop on machine learning for medical image reconstruction. Springer; 2018 Sep 16; pp. 91-99.

[107] Zhu B, Liu JZ, Cauley SF, Rosen BR, Rosen MS. Image reconstruction by domain-transform manifold learning. Nature. 2018; 555(7697):487-492.

[108] Huang Y, Würfl T, Breininger K, Liu L, Lauritsch G, Maier A. Some investigations on robustness of deep learning in limited angle tomography. In Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer. 2018; pp.145-153.

[109] Ye JC, Han Y, Cha E. Deep convolutional framelets: a general deep learning framework for inverse problems. SIAM Journal on Imaging Sciences. 2018; 11(2): 991-1048.

[110] Kang E, Chang W, Yoo J, Ye JC. Deep convolutional framelet denoising for low-dose CT via wavelet residual network. IEEE Trans Med Imaging. 2018; 37(6):1358-1369.

[111] Han Y, Ye JC. Framing U-Net via deep convolutional framelets: application to sparse-view CT. IEEE Trans Med Imaging. 2018; 37(6):1418-1429.

[112] Vishnevskiy V, Sanabria SJ, Goksel O. Image reconstruction via variational network for real-time hand-held sound-speed imaging. In Proceedings of International workshop on machine learning for medical image reconstruction. Springer. 2018 Sep 16; pp. 120-128.

[113] Würfl T, Ghesu FC, Christlein V, Maier A. Deep learning computed tomography. In Proceedings of International conference on medical image computing and computer-assisted intervention. Springer; 2016 Oct 17; pp. 432-440.

[114] Würfl T, Hoffmann M, Christlein V, Breininger K, Huang Y, Unberath M, Maier AK. Deep learning computed tomography: learning projection-domain weights from image domain in limited angle problems. IEEE Trans Med Imaging 2018; 37(6):1454-63.

[115] Syben C, Stimpel B, Breininger K, Würfl T, Fahrig R, DörflerA, Maier A. Precision learning: Reconstruction filter kernel discretization. In Proceedings of the Fifth International Conference on Image Formation in X-Ray Computed Tomography. 2018. pp.386-90.

[116] Hammernik K, Würfl T, Pock T, Maier A. A deep learning architecture for limited angle computed tomography reconstruction. In Proceedings of Bildverarbeitung für die Medi-zin 2017. Berlin, Heidelberg. Springer. 2017. pp. 92-97.

[117] Syben C, Stimpel B, Lommen J, Würfl T, Dörfler Ae, Maier A. Deriving neural network architectures using precision learning: parallel-to-fan beam conversion. In Proceedings of German Conference on Pattern Recognition (GCPR). Springer. 2018 Oct 9; pp. 503-517.

[118] Schlemper J, Castro DC, Bai W, Qin C, Oktay O, Duan J, Price AN, Hajnal J, Rueckert D. Bayesian deep learning for accelerated MR image reconstruction. In Proceedings of International Workshop on Machine Learning for Medical Image Reconstruction. Springer. 2018 Sep 16; pp. 64-71.

[119] Zhang Y, Yu H. Convolutional neural network based metal artifact reduction in X-ray computed tomography. IEEE Trans Med Imaging, 2018; 37(6):1370-1381.

[120] Bier B, Aschoff K, Syben C, Unberath M, Levenston M, Gold G, Fahrig R, Maier A. Detecting anatomical landmarks for motion estimation in weight-bearing imaging of knees. In Proceedings of International workshop on machine learning for medical image reconstruction. Springer; 2018 Sep 16; pp. 83–90.

[121] Xu T, Zhang P, Huang Q, Zhang H, Gan Z, Huang X, He X. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2018; pp. 1316-1324.
[122] Ledig C, Theis L, Huszár F, Caballero J, Cunningham A, Acosta A, Aitken A, Tejani A, Totz J, Wang Z, Shi W. Photo-realistic single image super-resolution using a generative adversarial network. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2017; pp. 4681-4690.
[123] Zhao J, Mathieu M, LeCun Y. Energy-based generative adversarial network, 5th International Conference on Learning Representations, 2019. DOI: arXiv:1702.01691v2.
[124] Johnson PM, Drangova M, Conditional generative adversarial network for 3D rigid-body motion correction in MRI. Magn Reson Med. 2019; 82(3): 901-910.
[125] Denton EL, Chintala S, Fergus R. Deep generative image models using a Laplacian pyramid of adversarial networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems. 2015; vol 1. pp.1486-1494.
[126] Chen X, Duan Y, Houthooft R, Schulman J, Sutskever I, Abbeel P. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. Advances in neural information processing systems. 2016; 29.
[127] Odena A, Olah C, Shlens J. Conditional image synthesis with auxiliary classifier gans. In Proceedings of the 34th International Conference on Machine Learning. 2017; vol. 70. pp. 2642-2651.
[128] Odena A. Semi-supervised learning with generative adversarial networks. arXiv preprint arXiv:1606.01583. 2016 Jun 5.
[129] Wang G. A perspective on deep imaging. IEEE Access 2016; 4: 8914–24.
[130] Sun C, Shrivastava A, Singh S, Gupta A. Revisiting unreasonable effectiveness of data in deep learning era; 2017. p. 1 arXiv:1707.02968.
[131] Oquab M, Bottou L, Laptev I, Sivic J. Is object localization for free? Weakly-supervised learning with convolutional neural networks. In Proceedings of the IEEE conference on computer vision and patterner cognition. 2015; pp. 685–94.

[132] Karras T, Laine S, Aila T. A style-based generator architecture for generative adversarial networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019; pp. 4401-4410.
[133] Mosser L, Dubrule O, Blunt MJ. Reconstruction of three-dimensional porous media using generative adversarial neural networks. Phys Rev E. 2017; 96(4-1): 043309.
[134] Bai J, Dai X, Wu Q, Xie L. Limited-view CT Reconstruction Based on Autoencoder-like Generative Adversarial Networks with Joint Loss. Annu Int Conf IEEE Eng Med Biol Soc. 2018; 2018: 5570-4.
[135] Wolterink JM, Leiner T, Viergever MA, Isgum I. Generative Adversarial Networks for Noise Reduction in Low-Dose CT. IEEE Trans Med Imaging. 2017; 36(12): 2536-45.
[136] Yang Q, Yan P, Zhang Y, Yu H, Shi Y, Mou X, Kalra MK, Zhang Y, Sun L, Wang G. Low-Dose CT Image Denoising Using a Generative Adversarial Network With Wasserstein Distance and Perceptual Loss. IEEE Trans Med Imaging. 2018; 37(6): 1348-1357.
[137] Yi X, Babyn P. Sharpness-Aware Low-Dose CT Denoising Using Conditional Generative Adversarial Network. J Digit Imaging. 2018; 31(5): 655-69.
[138] Choi K, Vania M, Kim S. Semi-Supervised Learning for Low-Dose CT Image Restoration with Hierarchical Deep Generative Adversarial Network (HD-GAN). Annu Int Conf IEEE Eng Med Biol Soc. 2019; 2019: 2683-6.
[139] Hu Z, Jiang C, Sun F, Zhang Q, Ge Y, Yang Y, et al. Artifact correction in low-dose dental CT imaging using Wasserstein generative adversarial networks. Med Phys. 2019; 46(4): 1686-96.
[140] Tang C, Li J, Wang L, Li Z, Jiang L, Cai A, Zhang W, Liang N, Li L, Yan B. Unpaired Low-Dose CT Denoising Network Based on Cycle-Consistent Generative Adversarial Network with Prior Image Information. Comput Math Methods Med. 2019; 2019:8639825.
[141] Wang J, Zhao Y, Noble JH, Dawant BM. Conditional Generative Adversarial Networks for Metal Artifact Reduction in CT Images of the Ear. Med Image Comput Comput Assist Interv. 2018; 11070:3-11.

[142] Funama Y, Oda S, Kidoh M, Nagayama Y, Goto M, Sakabe D, Nakaura T. Conditional generative adversarial networks to generate pseudo low monoenergetic CT image from a single-tube voltage CT scanner. Phys Med. 2021; 83:46-51.

[143] Harms J, Lei Y, Wang T, Zhang R, Zhou J, Tang X, Curran WJ, Liu T, Yang X. Paired cycle-GAN-based image correction for quantitative cone-beam computed tomography. Med Phys. 2019; 46(9): 3998-4009.

[144] Maspero M, Savenije MH, Dinkla AM, Seevinck PR, Intven MP, Jurgenliemk-Schulz IM, Kerkmeijer LG, van den Berg CA. Dose evaluation of fast synthetic-CT generation using a generative adversarial network for general pelvis MR-only radiotherapy. Phys Med Biol. 2018; 63(18): 185001.

[145] Lee JH, Han IH, Kim DH, Yu S, Lee IS, Song YS, Joo S, Jin CB, Kim H. Spine Computed Tomography to Magnetic Resonance Image Synthesis Using Generative Adversarial Networks: A Preliminary Study. J Korean Neurosurg Soc. 2020; 63(3): 386-396.

[146] Podgorsak AR, Shiraz Bhurwani MM, Ionita CN. CT artifact correction for sparse and truncated projection data using generative adversarial networks. Med Phys. 2021; 48(2): 615-626.

[147] Tien HJ, Yang HC, Shueng PW, Chen JC. Cone-beam CT image quality improvement using Cycle-Deblur consistent adversarial networks (Cycle-Deblur GAN) for chest CT imaging in breast cancer patients. Sci Rep. 2021; 11(1): 1133.

[148] Wang Y, Zhang W, Cai A, Wang L, Tang C, Feng Z, Li L, Liang N, Yan B. An effective sinogram inpainting for complementary limited-angle dual-energy computed tomography imaging using generative adversarial networks. J Xray Sci Technol. 2021; 29(1): 37-61.

[149] Cai J, Zhang Z, Cui L, Zheng Y, Yang L. Towards cross-modal organ translation and segmentation: A cycle- and shape-consistent generative adversarial network. Med Image Anal. 2019; 52: 174-84.

[150] Fu J, Singhrao K, Cao M, Yu V, Santhanam AP, Yang Y, et al. Generation of abdominal synthetic CTs from 0.35T MR images using generative adversarial networks for MR-only liver radiotherapy. Biomed Phys Eng Express. 2020;6(1):015033.
[151] Fu J, Singhrao K, Cao M, Yu V, Santhanam AP, Yang Y, Guo M, Raldow AC, Ruan D, Lewis JH. Three-Dimensional Liver Image Segmentation Using Generative Adversarial Networks Based on Feature Restoration. Front Med (Lausanne). 2021; 8: 794969.
[152] Zhang G, Mao Y, Li M, Peng L, Ling Y, Zhou X. The Optimal Tetralogy of Fallot Repair Using Generative Adversarial Networks. Front Physiol. 2021; 12: 613330.
[153] Zhang T, Cheng J, Fu H, Gu Z, Xiao Y, Zhou K, Gao S, Zheng R, Liu J. Noise Adaptation Generative Adversarial Network for Medical Image Analysis. IEEE Trans Med Imaging. 2020;39(4):1149-59.
[154] Onishi Y, Teramoto A, Tsujimoto M, Tsukamoto T, Saito K, Toyama H, Imaizumi K, Fujita H. Automated Pulmonary Nodule Classification in Computed Tomography Images Using a Deep Convolutional Neural Network Trained by Generative Adversarial Networks. Biomed Res Int. 2019; 2019: 6051939.
[155] Sandfort V, Yan K, Pickhardt PJ, Summers RM. Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks. Sci Rep. 2019; 9(1): 16884.
[156] Babier A, Mahmood R, McNiven AL, Diamant A, Chan TCY. Knowledge-based automated planning with three-dimensional generative adversarial networks. Med Phys. 2020; 47(2): 297-306.
[157] Kawahara D, Ozawa S, Kimura T, Nagata Y. Image synthesis of monoenergetic CT image in dual-energy CT using kilovoltage CT with deep convolutional generative adversarial networks. J Appl Clin Med Phys. 2021; 22(4):184-92.
[158] Klages P, Benslimane I, Riyahi S, Jiang J, Hunt M, Deasy JO, Veeraraghavan H, Tyagi N. Patch-based generative adversarial neural network models for head and neck MR-only planning. Med Phys. 2020; 47(2): 626-42.

[159] Hsieh KY, Tsai HC, Chen GY. Generation of High-resolution Lung Computed Tomography Images using Generative Adversarial Networks. Annu Int Conf IEEE Eng Med Biol Soc. 2020; 2020: 2400-3.
[160] Janssens N, Huysmans M, Swennen R. Computed Tomography 3D Super-Resolution with Generative Adversarial Neural Networks: Implications on Unsaturated and Two-Phase Fluid Flow. Materials (Basel). 2020;13(6): 1397.
[161] Sarrut D, Krah N, Letang JM. Generative adversarial networks (GAN) for compact beam source modelling in Monte Carlo simulations. Phys Med Biol. 2019; 64(21): 215004.
[162] Quan TM, Nguyen-Duc T, Jeong WK. Compressed Sensing MRI Reconstruction Using a Generative Adversarial Network with a Cyclic Loss. IEEE Trans Med Imaging. 2018; 37(6): 1488-97.
[163] Yang G, Yu S, Dong H, Slabaugh G, Dragotti PL, Ye X, Liu F, Arridge S, Keegan J, Guo Y, Firmin D. DAGAN: Deep De-Aliasing Generative Adversarial Networks for Fast Compressed Sensing MRI Reconstruction. IEEE Trans Med Imaging. 2018; 37(6): 1310-21.
[164] Hamghalam M, Wang T, Lei B. High tissue contrast image synthesis via multistage attention-GAN: Application to segmenting brain MR scans. Neural Netw. 2020; 132: 43-52.
[165] Yuan Z, Jiang M, Wang Y, Wei B, Li Y, Wang P, Menpes-Smith W, Niu Z, Yang G. SARA-GAN: Self-Attention and Relative Average Discriminator Based Generative Adversarial Networks for Fast Compressed Sensing MRI Reconstruction. Front Neuroinform. 2020; 14: 611666.
[166] Pan Y, Liu M, Lian C, Zhou T, Xia Y, Shen D. Synthesizing Missing PET from MRI with Cycle-consistent Generative Adversarial Networks for Alzheimer's Disease Diagnosis. Med Image Comput Comput Assist Interv. 2018; 11072: 455-63.
[167] Nie D, Trullo R, Lian J, Petitjean C, Ruan S, Wang Q, et al. Medical Image Synthesis with Context-Aware Generative Adversarial Networks. Med Image Comput Comput Assist Interv. 2017; 10435: 417-25.

[168] Emami H, Dong M, Nejad-Davarani SP, Glide-Hurst CK. Generating synthetic CTs from magnetic resonance images using generative adversarial networks. Med Phys. 2018.
[169] Dar SU, Yurt M, Karacan L, Erdem A, Erdem E, Cukur T. Image Synthesis in Multi-Contrast MRI With Conditional Generative Adversarial Networks. IEEE Trans Med Imaging. 2019; 38(10): 2375-88.
[170] Shi Y, Cheng K, Liu Z. Hippocampal subfields segmentation in brain MR images using generative adversarial networks. Biomed Eng Online. 2019; 18(1): 5.
[171] Siddiquee MMR, Zhou Z, Tajbakhsh N, Feng R, Gotway MB, Bengio Y, Liang J. Learning Fixed Points in Generative Adversarial Networks: From Image-to-Image Translation to Disease Detection and Localization. Proc IEEE Int Conf Comput Vis. 2019; 2019:191-200.
[172] Carver EN, Dai Z, Liang E, Snyder J, Wen N. Improvement of Multiparametric MR Image Segmentation by Augmenting the Data with Generative Adversarial Networks for Glioma Patients. Front Comput Neurosci. 2020; 14:495075.
[173] Delannoy Q, Pham CH, Cazorla C, Tor-Díez C, Dollé G, Meunier H, Bednarek N, Fablet R, Passat N, Rousseau F. SegSRGAN: Super-resolution and segmentation using generative adversarial networks - Application to neonatal brain MRI. Comput Biol Med. 2020; 120:103755.
[174] Gaj S, Yang M, Nakamura K, Li X. Automated cartilage and meniscus segmentation of knee MRI with conditional generative adversarial networks. Magn Reson Med. 2020; 84(1): 437-49.
[175] Conte GM, Weston AD, Vogelsang DC, Philbrick KA, Cai JC, Barbera M, Sanvito F, Lachance DH, Jenkins RB, Tobin WO, Eckel-Passow JE. Generative Adversarial Networks to Synthesize Missing T1 and FLAIR MRI Sequences for Use in a Multisequence Brain Tumor Segmentation Model. Radiology. 2021; 299(2): 313-23.
[176] Kossen T, Subramaniam P, Madai VI, Hennemuth A, Hildebrand K, Hilbert A, Sobesky J, Livne M, Galinovic I, Khalil AA, Fiebach JB. Synthesizing anonymized and labeled TOF-MRA patches for brain vessel segmentation using generative adversarial networks. Comput Biol Med. 2021; 131:104254.

[177] Wang W, Wang G, Wu X, Ding X, Cao X, Wang L, Zhang J, Wang P. Automatic segmentation of prostate magnetic resonance imaging using generative adversarial networks. Clin Imaging. 2021; 70:1-9.
[178] Bazangani F, Richard FJP, Ghattas B, Guedj E. Alzheimer's Disease Neuroimaging I. FDG-PET to T1 Weighted MRI Translation with 3D Elicit Generative Adversarial Network (E-GAN). Sensors (Basel). 2022; 22(12): 4640.
[179] De Asis-Cruz J, Krishnamurthy D, Jose C, Cook KM, Limperopoulos C. FetalGAN: Automated Segmentation of Fetal Functional Brain MRI Using Deep Generative Adversarial Learning and Multi-Scale 3D U-Net. Front Neurosci. 2022; 16:887634.
[180] Deng L, Zhang M, Wang J, Huang S, Yang X. Improving cone-beam CT quality using a cycle-residual connection with a dilated convolution-consistent generative adversarial network. Phys Med Biol. 2022; 67(14): 145010.
[181] Duman EA, Sagiroglu S, Celtikci P, Demirezen MU, Borcek AO, Emmez H, Celtikci E. Utilizing Deep Convolutional Generative Adversarial Networks for Automatic Segmentation of Gliomas: An Artificial Intelligence Study. Turk Neurosurg. 2022; 32(1): 16-21.
[182] Gao J, Zhao W, Li P, Huang W, Chen Z. LEGAN: A Light and Effective Generative Adversarial Network for medical image synthesis. Comput Biol Med. 2022; 148:105878.
[183] Gomi T, Kijima Y, Kobayashi T, Koibuchi Y. Evaluation of a Generative Adversarial Network to Improve Image Quality and Reduce Radiation-Dose during Digital Breast Tomosynthesis. Diagnostics (Basel). 2022; 12(2): 495.
[184] Hu N, Zhang T, Wu Y, Tang B, Li M, Song B, Gong Q, Wu M, Gu S, Lui S. Detecting brain lesions in suspected acute ischemic stroke with CT-based synthetic MRI using generative adversarial networks. Ann Transl Med. 2022;10(2): 35.
[185] Huang P, Li D, Jiao Z, Wei D, Cao B, Mo Z, Wang Q, Zhang H, Shen D. Common feature learning for brain tumor MRI synthesis by context-aware generative adversarial network. Med Image Anal. 2022; 79: 102472.

[186] Huang Z, Zhang G, Lin J, Pang Y, Wang H, Bai T, Zhong L. Multi-modal feature-fusion for CT metal artifact reduction using edge-enhanced generative adversarial networks. Comput Methods Programs Biomed. 2022; 217: 106700.
[187] Kawahara D, Tsuneda M, Ozawa S, Okamoto H, Nakamura M, Nishio T, Nagata Y. Deep learning-based auto segmentation using generative adversarial network on magnetic resonance images obtained for head and neck cancer patients. J Appl Clin Med Phys. 2022; 23(5): e13579.
[188] Kossen T, Hirzel MA, Madai VI, Boenisch F, Hennemuth A, Hildebrand K, Pokutta S, Sharma K, Hilbert A, Sobesky J, Galinovic I. Toward Sharing Brain Images: Differentially Private TOF-MRA Images with Segmentation Labels Using Generative Adversarial Networks. Front Artif Intell. 2022; 5: 813842.
[189] Li Z, Tian Q, Ngamsombat C, Cartmell S, Conklin J, Filho AL, Lo WC, Wang G, Ying K, Setsompop K, Fan Q. High-fidelity fast volumetric brain MRI using synergistic wave-controlled aliasing in parallel imaging and a hybrid denoising generative adversarial network (HDnGAN). Med Phys. 2022; 49(2): 1000-14.
[190] Liu X, Du H, Xu J, Qiu B. DBGAN: A dual-branch generative adversarial network for undersampled MRI reconstruction. Magn Reson Imaging. 2022; 89: 77-91.
[191] Luo S, Zhou J, Yang Z, Wei H, Fu Y. Diffusion MRI super-resolution reconstruction via sub-pixel convolution generative adversarial network. Magn Reson Imaging. 2022; 88: 101-7.
[192] Nakamoto A, Hori M, Onishi H, Ota T, Fukui H, Ogawa K, Masumoto J, Kudo A, Kitamura Y, Kido S, Tomiyama N. Three-dimensional conditional generative adversarial network-based virtual thin-slice technique for the morphological evaluation of the spine. Sci Rep. 2022; 12(1): 1-8.
[193] Niu K, Li X, Zhang L, Yan Z, Yu W, Liang P, Wang Y, Lin CP, Zhang H, Guo C, Li K. Improving segmentation reliability of multi-scanner brain images using a generative adversarial network. Quant Imaging Med Surg. 2022; 12(3): 1775-86.
[194] Nneji GU, Deng J, Monday HN, Hossin MA, Obiora S, Nahar S, Cai J. COVID-19 Identification from Low-Quality Computed Tomography Using a Modified Enhanced

Super-Resolution Generative Adversarial Network Plus and Siamese Capsule Network. Healthcare (Basel). 2022; 10(2):403.

[195] Ota J, Umehara K, Kershaw J, Kishimoto R, Hirano Y, Tachibana Y, Ohba H, Obata T. Super-resolution generative adversarial networks with static T2*WI-based subject-specific learning to improve spatial difference sensitivity in fMRI activation. Sci Rep. 2022; 12(1): 1-9.

[196] Qiang N, Dong Q, Liang H, Li J, Zhang S, Zhang C, Ge B, Sun Y, Gao J, Liu T, Yue H. Learning brain representation using recurrent Wasserstein generative adversarial net. Comput Methods Programs Biomed. 2022; 223: 106979.

[197] Quintana-Quintana OJ, De Leon-Cuevas A, Gonzalez-Gutierrez A, Gorrostieta-Hurtado E, Tovar-Arriaga S. Dual U-Net-Based Conditional Generative Adversarial Network for Blood Vessel Segmentation with Reduced Cerebral MR Training Volumes. Micromachines (Basel). 2022;13(6): 823.

[198] Roy R, Mazumdar S, Chowdhury AS. ADGAN: Attribute-Driven Generative Adversarial Network for Synthesis and Multiclass Classification of Pulmonary Nodules. IEEE Trans Neural Netw Learn Syst. 2022; PP.

[199] Tan C, Yang M, You Z, Chen H, Zhang Y. A selective kernel-based cycle-consistent generative adversarial network for unpaired low-dose CT denoising. Precis Clin Med. 2022; 5(2): pbac011.

[200] Tyagi S, Talbar SN. CSE-GAN: A 3D conditional generative adversarial network with concurrent squeeze-and-excitation blocks for lung nodule segmentation. Comput Biol Med. 2022; 147: 105781.

[201] Ueki W, Nishii T, Umehara K, Ota J, Higuchi S, Ohta Y, Nagai Y, Murakawa K, Ishida T, Fukuda T. Generative adversarial network-based post-processed image super-resolution technology for accelerating brain MRI: comparison with compressed sensing. Acta Radiol. 2022:2841851221076330.

[202] Usui K, Ogawa K, Goto M, Sakano Y, Kyougoku S, Daida H. A cycle generative adversarial network for improving the quality of four-dimensional cone-beam computed tomography images. Radiat Oncol. 2022; 17(1): 69.

[203] Van Voorst H, Konduri PR, van Poppel LM, van der Steen W, van der Sluijs PM, Slot EM, Emmer BJ, van Zwam WH, Roos YB, Majoie CB, Zaharchuk G. Unsupervised Deep Learning for Stroke Lesion Segmentation on Follow-up CT Based on Generative Adversarial Networks. AJNR Am J Neuroradiol. 2022; 43(8):1107-1114.

[204] Wang CC, Wu PH, Lin G, Huang YL, Lin YC, Chang YE, et al. Magnetic Resonance-Based Synthetic Computed Tomography Using Generative Adversarial Networks for Intracranial Tumor Radiotherapy Treatment Planning. J Pers Med. 2022;12(3): 361.

[205] Wei H, Li Z, Wang S, Li R. Undersampled Multi-contrast MRI Reconstruction Based on Double-domain Generative Adversarial Network. IEEE J Biomed Health Inform. 2022; 26(9): 4371-4377.

[206] Xie H, Lei Y, Wang T, Roper J, Dhabaan AH, Bradley JD, Liu T, Mao H, Yang X. Synthesizing high-resolution magnetic resonance imaging using parallel cycle-consistent generative adversarial networks for fast magnetic resonance imaging. Med Phys. 2022; 49(1): 357-69.

[207] Xiong YT, Zeng W, Xu L, Guo JX, Liu C, Chen JT, Du XY, Tang W. Virtual reconstruction of midfacial bone defect based on generative adversarial network. Head Face Med. 2022; 18(1): 19.

[208] Yang M, Colak C, Chundru KK, Gaj S, Nanavati A, Jones MH, Winalski CS, Subhas N, Li X. Automated knee cartilage segmentation for heterogeneous clinical MRI using generative adversarial networks with transfer learning. Quant Imaging Med Surg. 2022; 12(5): 2620-33.

[209] Ye H, Yang Y, Mao K, Wang Y, Hu Y, Xu Y, Fei P, Lyv J, Chen L, Zhao P, Zheng C. Generating Synthesized Ultrasound Biomicroscopy Images from Anterior Segment Optical Coherent Tomography Images by Generative Adversarial Networks for Iridociliary Assessment. Ophthalmol Ther. 2022; 11(5):1817-31.

[210] Zhan B, Xiao J, Cao C, Peng X, Zu C, Zhou J, Wang Y. Multi-constraint generative adversarial network for dose prediction in radiotherapy. Med Image Anal. 2022; 77: 102339.

[211] Zhang H, Li H, Dillman JR, Parikh NA, He L. Multi-Contrast MRI Image Synthesis Using Switchable Cycle-Consistent Generative Adversarial Networks. Diagnostics (Basel). 2022;12(4): 816.

[212] Zhang J, Qiu S, Cui X, Liang T. A Pulmonary Nodule Spiculation Recognition Algorithm Based on Generative Adversarial Networks. Biomed Res Int. 2022; 2022: 3341924.

[213] Zhang K, Hu H, Philbrick K, Conte GM, Sobek JD, Rouzrokh P, Erickson BJ. SOUP-GAN: Super-Resolution MRI Using Generative Adversarial Networks. Tomography. 2022; 8(2): 905-19.

[214] Zhang L, Jiang B, Chen Q, Wang L, Zhao K, Zhang Y, Vliegenthart R, Xie X. Motion artifact removal in coronary CT angiography based on generative adversarial networks. Eur Radiol. 2022.

[215] Zhang Y, Ding SG, Gong XC, Yuan XX, Lin JF, Chen Q, Li JG. Generating synthesized computed tomography from CBCT using a conditional generative adversarial network for head and neck cancer patients. Technol Cancer Res Treat. 2022; 21:15330338221085358.

[216] Zhao M, Wei Y, Wong KKL. A Generative Adversarial Network technique for high-quality super-resolution reconstruction of cardiac magnetic resonance images. Magn Reson Imaging. 2022; 85: 153-60.

[217] Zheng C, Ye H, Yang J, Fei P, Qiu Y, Xie X, Wang Z, Chen J, Zhao P. Development and Clinical Validation of Semi-Supervised Generative Adversarial Networks for Detection of Retinal Disorders in Optical Coherence Tomography Images Using Small Dataset. Asia Pac J Ophthalmol (Phila). 2022; 11(3): 219-26.

[218] Zhou H, Liu X, Wang H, Chen Q, Wang R, Pang ZF, Zhang Y, Hu Z. The synthesis of high-energy CT images from low-energy CT images using an improved cycle generative adversarial network. Quant Imaging Med Surg. 2022; 12(1): 28-42.

[219] Zhu L, He Q, Huang Y, Zhang Z, Zeng J, Lu L, Kong W, Zhou F. DualMMP-GAN: Dual-scale multi-modality perceptual generative adversarial network for medical image segmentation. Comput Biol Med. 2022; 144: 105387.

[220] Goudarzi S, Asif A, Rivaz H. High Frequency Ultrasound Image Recovery Using Tight Frame Generative Adversarial Networks. Annu Int Conf IEEE Eng Med Biol Soc. 2020; 2020: 2035-8.
[221] Zhou Z, Wang Y, Guo Y, Jiang X, Qi Y. Ultrafast Plane Wave Imaging With Line-Scan-Quality Using an Ultrasound-Transfer Generative Adversarial Network. IEEE J Biomed Health Inform. 2020; 24(4): 943-56.
[222] Zhang L, Zhang J. Ultrasound image denoising using generative adversarial networks with residual dense connectivity and weighted joint loss. PeerJ Comput Sci. 2022; 8: e873.
[223] Han L, Huang Y, Dou H, Wang S, Ahamad S, Luo H, Liu Q, Fan J, Zhang J. Semi-supervised segmentation of lesion from breast ultrasound images with attentional generative adversarial network. Comput Methods Programs Biomed. 2020; 189: 105275.
[224] Torrents-Barrena J, Piella G, Valenzuela-Alcaraz B, Gratacos E, Eixarch E, Ceresa M, Ballester MA. TTTS-STgan: Stacked Generative Adversarial Networks for TTTS Fetal Surgery Planning Based on 3D Ultrasound. IEEE Trans Med Imaging. 2020; 39(11): 3595-606.
[225] Fujioka T, Kubota K, Mori M, Katsuta L, Kikuchi Y, Kimura K, Kimura M, Adachi M, Oda G, Nakagawa T, Kitazume Y. Virtual Interpolation Images of Tumor Development and Growth on Breast Ultrasound Image Synthesis With Deep Convolutional Generative Adversarial Networks. J Ultrasound Med. 2021; 40(1): 61-9.
[226] Zhang Q, Zhao J, Long X, Luo Q, Wang R, Ding X, Shen C. AUE-Net: Automated Generation of Ultrasound Elastography Using Generative Adversarial Network. Diagnostics (Basel). 2022; 12(2): 253.
[227] Zhao J, Zhou X, Shi G, Xiao N, Song K, Zhao J, Hao R, Li K. Semantic consistency generative adversarial network for cross-modality domain adaptation in ultrasound thyroid nodule classification. Appl Intell (Dordr). 2022; 52(9): 10369-10383.
[228] Fujioka T, Mori M, Kubota K, Kikuchi Y, Katsuta L, Adachi M, Oda G, Nakagawa T, Kitazume Y, Tateishi U. Breast Ultrasound Image Synthesis using Deep Convolutional Generative Adversarial Networks. Diagnostics (Basel). 2019; 9(4): 176.

[229] Wang Y, Yu B, Wang L, Zu C, Lalush DS, Lin W, Wu X, Zhou J, Shen D, Zhou L. 3D conditional generative adversarial networks for high-quality PET image estimation at low dose. Neuroimage. 2018; 174: 550-62.
[230] Armanious K, Kustner T, Reimold M, Nikolaou K, La Fougere C, Yang B, Gatidis S. Independent brain (18)F-FDG PET attenuation correction using a deep learning approach with Generative Adversarial Networks. Hell J Nucl Med. 2019; 22(3): 179-186.
[231] Lei Y, Dong X, Wang T, Higgins K, Liu T, Curran WJ, Mao H, Nye JA, Yang X. Whole-body PET estimation from low count statistics using cycle-consistent generative adversarial networks. Phys Med Biol. 2019; 64(21): 215017.
[232] Armanious K, Hepp T, Kustner T, Dittmann H, Nikolaou K, La Fougere C, Yang B, Gatidis S. Independent attenuation correction of whole body [(18)F]FDG-PET using a deep learning approach with Generative Adversarial Networks. EJNMMI Res. 2020; 10(1): 53.
[233] Hu Z, Li Y, Zou S, Xue H, Sang Z, Liu X, Yang Y, Zhu X, Liang D, Zheng H. Obtaining PET/CT images from non-attenuation corrected PET images in a single PET system using Wasserstein generative adversarial networks. Phys Med Biol. 2020; 65(21): 215010.
[234] Islam J, Zhang Y. GAN-based synthetic brain PET image generation. Brain Inform. 2020; 7(1): 3.
[235] Kimura Y, Watanabe A, Yamada T, Watanabe S, Nagaoka T, Nemoto M, Miyazaki K, Hanaoka K, Kaida H, Ishii K. AI approach of cycle-consistent generative adversarial networks to synthesize PET images to train computer-aided diagnosis algorithm for dementia. Ann Nucl Med. 2020; 34(7): 512-5.
[236] Sundar LK, Iommi D, Muzik O, Chalampalakis Z, Klebermass EM, Hienert M, Rischka L, Lanzenberger R, Hahn A, Pataraia E, Traub-Weidinger T. Conditional Generative Adversarial Networks (cGANs) aided motion correction of dynamic (18)F-FDG PET brain studies. J Nucl Med. 2020; 62(6):871-9.
[237] Jeong YJ, Park HS, Jeong JE, Yoon HJ, Jeon K, Cho K, Kang DY. Restoration of amyloid PET images obtained with short-time data using a generative adversarial networks framework. Sci Rep. 2021;11(1):4825.

[238] Galbusera F, Niemeyer F, Seyfried M, Bassani T, Casaroli G, Kienle A, Wilke HJ. Exploring the Potential of Generative Adversarial Networks for Synthesizing Radiological Images of the Spine to be Used in In Silico Trials. Front Bioeng Biotechnol. 2018; 6: 53.
[239] Sun Y, Liu X, Cong P, Li L, Zhao Z. Digital radiography image denoising using a generative adversarial network. J Xray Sci Technol. 2018; 26(4): 523-34.
[240] Guan S, Loew M. Breast cancer detection using synthetic mammograms from generative adversarial networks in convolutional neural networks. J Med Imaging (Bellingham). 2019; 6(3): 031411.
[241] Zhang Y, Miao S, Mansi T, Liao R. Unsupervised X-ray image segmentation with task driven generative adversarial networks. Med Image Anal. 2020; 62: 101664.
[242] Ahn G, Choi BS, Ko S, Jo C, Han HS, Lee MC, Ro DH. High-Resolution Knee Plain Radiography Image Synthesis Using Style Generative Adversarial Network Adaptive Discriminator Augmentation. J Orthop Res. 2022; 41(1): 84-93.
[243] Bae K, Oh DY, Yun ID, Jeon KN. Bone Suppression on Chest Radiographs for Pulmonary Nodule Detection: Comparison between a Generative Adversarial Network and Dual-Energy Subtraction. Korean J Radiol. 2022; 23(1): 139-49.
[244] Yang CJ, Lin CL, Wang CK, Wang JY, Chen CC, Su FC, Lee YJ, Lui CC, Yeh LR, Fang YH. Generative Adversarial Network (GAN) for Automatic Reconstruction of the 3D Spine Structure by Using Simulated Bi-Planar X-ray Images. Diagnostics (Basel). 2022; 12(5): 1121.
[245] Zhou Y, Wei J, Wu D, Zhang Y. Generating Full-Field Digital Mammogram From Digitized Screen-Film Mammogram for Breast Cancer Screening With High-Resolution Generative Adversarial Network. Front Oncol. 2022; 12:868257.
[246] Burlingame EA, Margolin AA, Gray JW, Chang YH. SHIFT: speedy histopathological-to-immunofluorescent translation of whole slide images using conditional generative adversarial networks. Proc SPIE Int Soc Opt Eng. 2018; 10581.
[247] Dirvanauskas D, Maskeliunas R, Raudonis V, Damasevicius R, Scherer R. HEMIGEN: Human Embryo Image Generator Based on Generative Adversarial Networks. Sensors (Basel). 2019; 19(16): 3578.

[248] Rau A, Edwards PJE, Ahmad OF, Riordan P, Janatka M, Lovat LB, Stoyanov D. Implicit domain adaptation with conditional generative adversarial networks for depth prediction in endoscopy. Int J Comput Assist Radiol Surg. 2019; 14(7): 1167-76.
[249] de Souza LA, Jr., Passos LA, Mendel R, Ebigbo A, Probst A, Messmann H, Palm C, Papa JP. Assisting Barrett's esophagus identification using endoscopic data augmentation based on Generative Adversarial Networks. Comput Biol Med. 2020; 126: 104029.
[250] Poorneshwaran JM, Santhosh Kumar S, Ram K, Joseph J, Sivaprakasam M. Polyp Segmentation using Generative Adversarial Network. Annu Int Conf IEEE Eng Med Biol Soc. 2019; 2019:7201-4.
[251] Yoon D, Kong HJ, Kim BS, Cho WS, Lee JC, Cho M, Lim MH, Yang SY, Lim SH, Lee J, Song JH. Colonoscopic image synthesis with generative adversarial network for enhanced detection of sessile serrated lesions using convolutional neural network. Sci Rep. 2022; 12(1): 261.
[252] Torrents-Barrena J, Piella G, Masoller N, Gratacos E, Eixarch E, Ceresa M, Ballester MA. Fetal MRI Synthesis via Balanced Auto-Encoder Based Generative Adversarial Networks. Annu Int Conf IEEE Eng Med Biol Soc. 2018; 2018: 2599-602.
[253] Matsui T, Taki M, Pham TQ, Chikazoe J, Jimura K. Counterfactual Explanation of Brain Activity Classifiers Using Image-To-Image Transfer by Generative Adversarial Network. Front Neuroinform. 2021; 15: 802938.
[254] Hu B, Tang Y, Chang EI, Fan Y, Lai M, Xu Y. Unsupervised Learning for Cell-Level Visual Representation in Histopathology Images With Generative Adversarial Networks. IEEE J Biomed Health Inform. 2019; 23(3): 1316-28.
[255] Chen MT, Mahmood F, Sweer JA, Durr NJ. GANPOP: Generative Adversarial Network Prediction of Optical Properties From Single Snapshot Wide-Field Images. IEEE Trans Med Imaging. 2020; 39(6): 1988-99.
[256] Levine AB, Peng J, Farnell D, Nursey M, Wang Y, Naso JR, Ren H, Farahani H, Chen C, Chiu D, Talhouk A. Synthesis of diagnostic quality cancer pathology images by generative adversarial networks. J Pathol. 2020; 252(2): 178-88.

[257] Teramoto A, Tsukamoto T, Yamada A, Kiriyama Y, Imaizumi K, Saito K, Fujita H. Deep learning approach to classification of lung cytological images: Two-step training using actual and synthesized images by progressive growing of generative adversarial networks. PLoS One. 2020; 15(3): e0229951.

[258] Lorencin I, Baressi Segota S, Andelic N, Mrzljak V, Cabov T, Spanjol J, Car Z. On Urinary Bladder Cancer Diagnosis: Utilization of Deep Convolutional Generative Adversarial Networks for Data Augmentation. Biology (Basel). 2021; 10(3): 175.

[259] Vu QD, Kim K, Kwak JT. Unsupervised Tumor Characterization via Conditional Generative Adversarial Networks. IEEE J Biomed Health Inform. 2021; 25(2): 348-57.

[260] Chen YI, Chang YJ, Liao SC, Nguyen TD, Yang J, Kuo YA, Hong S, Liu YL, Rylander HG, Santacruz SR, Yankeelov TE. Generative adversarial network enables rapid and robust fluorescence lifetime image analysis in live cells. Commun Biol. 2022; 5(1): 18.

[261] Das A, Devarampati VK, Nair MS. NAS-SGAN: A Semi-Supervised Generative Adversarial Network Model for Atypia Scoring of Breast Cancer Histopathological Images. IEEE J Biomed Health Inform. 2022; 26(5): 2276-87.

[262] Theagarajan R, Bhanu B. DeephESC 2.0: Deep Generative Multi Adversarial Networks for improving the classification of hESC. PLoS One. 2019; 14(3): e0212849.

[263] Gadermayr M, Gupta L, Appel V, Boor P, Klinkhammer BM, Merhof D. Generative Adversarial Networks for Facilitating Stain-Independent Supervised and Unsupervised Segmentation: A Study on Kidney Histology. IEEE Trans Med Imaging. 2019; 38(10): 2293-302.

[264] Hussain S, Anees A, Das A, Nguyen BP, Marzuki M, Lin S, Wright G, Singhal A. High-content image generation for drug discovery using generative adversarial networks. Neural Netw. 2020; 132: 353-63.

[265] Mahapatra D, Bozorgtabar B, Garnavi R. Image super-resolution using progressive generative adversarial networks for medical image analysis. Comput Med Imaging Graph. 2019; 71: 30-9.

[266] Iqbal T, Ali H. Generative Adversarial Network for Medical Images (MI-GAN). J Med Syst. 2018; 42(11): 231.
[267] Zhao H, Li H, Maurer-Stroh S, Cheng L. Synthesizing retinal and neuronal images with generative adversarial nets. Med Image Anal. 2018; 49: 14-26.
[268] Zheng R, Liu L, Zhang S, Zheng C, Bunyak F, Xu R, Li B, Sun M. Detection of exudates in fundus photographs with imbalanced learning using conditional generative adversarial network. Biomed Opt Express. 2018; 9(10): 4863-78.
[269] Schlegl T, Seebock P, Waldstein SM, Langs G, Schmidt-Erfurth U. f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks. Med Image Anal. 2019; 54: 30-44.
[270] Son J, Park SJ, Jung KH. Towards Accurate Segmentation of Retinal Vessels and the Optic Disc in Fundoscopic Images with Generative Adversarial Networks. J Digit Imaging. 2019; 32(3): 499-512.
[271] Yu Z, Xiang Q, Meng J, Kou C, Ren Q, Lu Y. Retinal image synthesis from multiple-landmarks input with generative adversarial networks. Biomed Eng Online. 2019; 18(1): 62.
[272] Lazaridis G, Lorenzi M, Ourselin S, Garway-Heath D. Improving statistical power of glaucoma clinical trials using an ensemble of cyclical generative adversarial networks. Med Image Anal. 2021; 68: 101906.
[273] Liu J, Shen C, Aguilera N, Cukras C, Hufnagel RB, Zein WM, Liu T, Tam J. Active Cell Appearance Model Induced Generative Adversarial Networks for Annotation-Efficient Cell Segmentation and Identification on Adaptive Optics Retinal Images. IEEE Trans Med Imaging. 2021; 40(10):2820-2831.
[274] Zhou Y, Wang B, He X, Cui S, Shao L. DR-GAN: Conditional Generative Adversarial Network for Fine-Grained Lesion Synthesis on Diabetic Retinopathy Images. IEEE J Biomed Health Inform. 2022; 26(1): 56-66.
[275] Lei B, Xia Z, Jiang F, Jiang X, Ge Z, Xu Y, Qin J, Chen S, Wang T, Wang S. Skin lesion segmentation via generative adversarial networks with dual discriminators. Med Image Anal. 2020; 64:101716.

[276] Aida S, Okugawa J, Fujisaka S, Kasai T, Kameda H, Sugiyama T. Deep Learning of Cancer Stem Cell Morphology Using Conditional Generative Adversarial Networks. Biomolecules. 2020;10(6): 931.

[277] Hazra D, Byun YC. SynSigGAN: Generative Adversarial Networks for Synthetic Biomedical Signal Generation. Biology (Basel). 2020; 9(12): 441.

[278] Haradal S, Hayashi H, Uchida S. Biosignal Data Augmentation Based on Generative Adversarial Networks. Annu Int Conf IEEE Eng Med Biol Soc. 2018; 2018: 368-71.

[279] Truong ND, Zhou L, Kavehei O. Semi-supervised Seizure Prediction with Generative Adversarial Networks. Annu Int Conf IEEE Eng Med Biol Soc. 2019; 2019: 2369-72.

[280] Fahimi F, Dosen S, Ang KK, Mrachacz-Kersting N, Guan C. Generative Adversarial Networks-Based Data Augmentation for Brain-Computer Interface. IEEE Trans Neural Netw Learn Syst. 2020; 32(9): 4039-4051.

[281] Karimi M, Zhu S, Cao Y, Shen Y. De Novo Protein Design for Novel Folds Using Guided Conditional Wasserstein Generative Adversarial Networks. J Chem Inf Model. 2020; 60(12): 5667-81.

[282] Kazuhiro K, Werner RA, Toriumi F, Javadi MS, Pomper MG, Solnes LB, Verde F, Higuchi T, Rowe SP. Generative Adversarial Networks for the Creation of Realistic Artificial Brain Magnetic Resonance Images. Tomography. 2018; 4(4): 159-63.

[283] Huo Y, Xu Z, Bao S, Bermudez C, Plassard AJ, Liu J, Yao Y, Assad A, Abramson RG, Landman BA. Splenomegaly Segmentation using Global Convolutional Kernels and Conditional Generative Adversarial Networks. In Proceedings of SPIE Medical Imaging. Houston, Texas, United States. 2018; vol. 10574, pp. 45-51.

[284] Mardani M, Gong E, Cheng JY, Vasanawala SS, Zaharchuk G, Xing L, Pauly JM. Deep Generative Adversarial Neural Networks for Compressive Sensing MRI. IEEE Trans Med Imaging. 2019; 38(1): 167-79.

[285] Yu B, Zhou L, Wang L, Shi Y, Fripp J, Bourgeat P. Ea-GANs: Edge-Aware Generative Adversarial Networks for Cross-Modality MR Image Synthesis. IEEE Trans Med Imaging. 2019; 38(7): 1750-62.

[286] Dai X, Lei Y, Fu Y, Curran WJ, Liu T, Mao H, Yang X. Multimodal MRI synthesis using unified generative adversarial networks. Med Phys. 2020; 47(12): 6343-54.
[287] Decourt C, Duong L. Semi-supervised generative adversarial networks for the segmentation of the left ventricle in pediatric MRI. Comput Biol Med. 2020; 123: 103884.
[288] Huang Y, Zheng F, Cong R, Huang W, Scott MR, Shao L. MCMT-GAN: Multi-Task Coherent Modality Transferable GAN for 3D Brain Image Synthesis. IEEE Trans Image Process. 2020; PP.
[289] Zhao M, Liu X, Liu H, Wong KKL. Super-resolution of cardiac magnetic resonance images using Laplacian Pyramid based on Generative Adversarial Networks. Comput Med Imaging Graph. 2020; 80: 101698.
[290] Yurt M, Dar SU, Erdem A, Erdem E, Oguz KK, Cukur T. Mustgan: Multi-stream generative adversarial networks for MR image synthesis. Med Image Anal. 2021; 70: 101944.
[291] Ahmad B, Sun J, You Q, Palade V, Mao Z. Brain Tumor Classification Using a Combination of Variational Autoencoders and Generative Adversarial Networks. Biomedicines. 2022;10(2): 223.
[292] Cui J, Gong K, Han P, Liu H, Li Q. Unsupervised arterial spin labeling image superresolution via multiscale generative adversarial network. Med Phys. 2022; 49(4): 2373-85.
[293] Wang X, Ghasedi Dizaji K, Huang H. Conditional generative adversarial network for gene expression inference. Bioinformatics. 2018;34(17): i603-i611.
[294] Seyyedsalehi SF, Soleymani M, Rabiee HR, Mofrad MRK. PFP-WGAN: Protein function prediction by discovering Gene Ontology term correlations with generative adversarial networks. PLoS One. 2021;16(2): e0244430.
[295] Kingma DP, Welling M. Auto-Encoding Variational Bayes. Stat. 2014 May; 1050:1.
[296] Li M, Huang H, Ma L, Liu W, Zhang T, Jiang Y. Unsupervised image-to-image translation with stacked cycle-consistent adversarial networks. In Proceedings of the European Conference on Computer Vision (ECCV). 2018: 184-199.

[297] Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing. 2004; 13(4): 600–612.
[298] Wang Z, Bovik AC. A universal image quality index. IEEE signal processing letters. 2002; 9(3): 81-4.
[299] González-Audícana M, Saleta JL, Catalán RG, García R. Fusion of multispectral and panchromatic images using improved IHS and PCA mergers based on wavelet decomposition. IEEE Transactions on Geoscience and Remote sensing. 2004; 42(6): 1291-9.
[300] Yuhas RH, Goetz AF, Boardman JW. Discrimination among semi-arid landscape endmembers using the spectral angle mapper (SAM) algorithm. In Proceedings of JPL, Summaries of the Third Annual JPL Airborne Geoscience Workshop. Volume 1: AVIRIS Workshop 1992 Jun 1.
[301] Sheikh HR, Bovik AC. Image information and visual quality. IEEE Transactions on image processing. 2006; 15(2): 430-44.
[302] Yang F, Poostchi M, Yu H, Zhou Z, Silamut K, Yu J, Maude RJ, Jaeger S, Antani S. Deep learning for smartphone-based malaria parasite detection in thick blood smears. IEEE J Biomed Health Inform. 2020; 24(5): 1427-1438.
[303] Yu H, Yang F, Rajaraman S, Ersoy I, Moallem G, Poostchi M, Palaniappan K, Antani S, Maude RJ, Jaeger S. Malaria Screener: a smartphone application for automated malaria screening. BMC Infect Dis. 2020; 20(1): 1-8.
[304] Fang Y, Zhang H, Xie J, Lin M, Ying L, Pang P, Ji W. Sensitivity of chest CT for COVID-19: comparison to RT-PCR. Radiology. 2020; 200432.
[305] Chen N, Zhou M, Dong X, Qu J, Gong F. Han Y, Qiu Y, Wang J, Liu Y, Wei Y, Yu T. Epidemiological and clinical characteristics of 99 cases of 2019 novel coronavirus pneumonia in Wuhan, China: a descriptive study. Lance. 2020; 395: 507-513.
[306] Minaee S, Kafieh R, Sonka M, Yazdani S, Jamalipour Soufi G. Deep-COVID: Predicting COVID-19 from chest X-ray images using deep transfer learning. Med Image Anal. 2020; 65: 101794.

[307] Karras T, Laine S, Aittala M, Hellsten J, Lehtinen J, Aila T. Analyzing and improving the image quality of stylegan. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition 2020, pp. 8110-8119.

Appendices

This section contains the list of my publications during the PhD study period, and the python source codes for the implementations of the residual network-based classifiers, the convolutional variation autoencoder (CVA) models, and the Ad CycleGAN models respectively for malaria parasitemic blood cell images and COVID-19 chest X-Ray images. All the source codes are originally written in the Jupyter Interactive Notebook format, which can be directly accessed at:
https://github.com/StanleyLiangYork/GAN_for_Medical_Image

and run directly on Google Colab platform.

There are seven appendices in this part.

- Appendix A – Publications during Study Period
- Appendix B - Residual Network for Malaria Parasitemic Blood Cell image Classification
- Appendix C - Residual Network for COVID-19 Chest X-Ray Image Classification
- Appendix D - Convolutional Variation Autoencoder for Malaria Parasitemic Blood Cell image Synthesis
- Appendix E - Convolutional Variation Autoencoder for COVID-19 Chest X-Ray Image Synthesis
- Appendix F - Ad Cycle GAN for Malaria Parasitemic Blood Cell image Synthesis
- Appendix G - Ad Cycle GAN for COVID-19 Chest X-Ray Image Synthesis

The readers who are interested in replicating the experiments are highly recommended run the GitHub version directly online with a Google Colab virtual machine with GPU support. Or you can also replicate all the experiments by accessing the datasets on the author's Google Cloud Storage bucket following the URL provide in the source code.

Appendix A: **Publications during Study Period**

Journal Articles

1. Liang Zhaohui, Liu Jun, Ou Aihua, Zhang Honglai, Li Ziping, Huang Jimmy Xiangji. Deep generative learning for automated EHR diagnosis of traditional Chinese medicine. Computer methods and programs in biomedicine. 2019; 174: 17-23.
2. Liang Zhaohui, Liu Jun, Huang Jimmy Xiangji, Zeng Xing. Fast Screening Technology for Drug Emergency Management: Predicting Suspicious SNPs for ADR with Information Theory-based Models. Combinatorial Chemistry & High Throughput Screening. 2018; 21(2): 93-99.
3. Liang Zhaohui, Huang Jimmy Xiangji, Antani Sameer. Image Translation by Ad CycleGAN for COVID-19 X-ray Images: A New Approach for Controllable GAN. Sensors. 2022, 22(24): 9628.

Conference Articles

1. Liang Zhaohui, Huang Jimmy Xiangji. Emergency Department Wait Time Prediction based on Cyclical Features by Deep Neural Networks. In Proceedings of AMIA 2022 Clinical Informatics Conference. 2022, May 24-26, Houston, USA.
2. Liang Zhaohui, Huang Jimmy Xiangji. CycleGAN with Dynamic Criterion for Malaria Blood Cell Image Synthetization. In Proceedings of AMIA 2022 Informatics Summit. 2022, Mar 21-24, Chicago. USA.
3. Liang Zhaohui, Huang Jimmy Xiangji. Adaptive Cycle-consistent Adversarial Network for Malaria Blood Cell Image Synthetization. In Proceedings of IEEE Applied Imagery Pattern Recognition Workshop (AIPR), 2021, Oct 12 -14, Washington DC, USA, pp.1-7.
4. Liang Zhaohui, Huang Jimmy Xiangji. Cycle-Consistent Adversarial network with criterion for COVID-19 Chest X-ray Generation. In Proceedings of AMIA 2021 Annual Symposium, 2021, Oct 30 – Nov 3, San Diego, pp. 1723.
5. Liang Zhaohui, Huang Jimmy Xiangji, Li Jun, Chan Stephen. Enhancing automated COVID-19 chest X-ray diagnosis by image-to-image GAN translation. In Proceedings of IEEE International Conference on Bioinformatics and Biomedicine (BIBM). 2020, Dec 16-19, pp. 1068-1071.
6. Liang Zhaohui, Liu Jun, Zhang Honglai, Huang Jimmy, Li Ziping, Chan Stephen. Patient Entity Recognition by Word Embedding Representation and Deep Learning. In Proceedings of IEEE International Conference on Bioinformatics and Biomedicine (BIBM). 2019, Nov 18-21, San Diego, CA, USA, pp. 1096-1099.

Appendix B: **Residual Network for Malaria Parasitemic Blood Cell image Classification**

```
# -*- coding: utf-8 -*-

"""Original file is located at

    https://github.com/StanleyLiangYork/GAN_for_Medical_Image/blob/main/Cycle_GAN_with_Criterion_malaria.ipynb

"""

!pip install tensorflow_addons


import os

import shutil

import random

import pandas as pd

import numpy as np

import seaborn as sns

import tensorflow as tf

import zipfile

import random

from PIL import Image

from matplotlib import pyplot as plt

import re

import tensorflow.keras as tfk
```

```
import tensorflow.keras.layers as tfkl
import tensorflow_hub as hub
import cv2
import tensorflow_addons as tfa
import tensorflow_probability as tfp
import pathlib

print(f'Tensorflow Version: {tf.__version__}')

"""Set a random seed for replication"""

tf.random.set_seed(1000)

"""Fetch the malaria dataset"""

if not os.path.exists('malaria.zip'):
  !wget https://storage.googleapis.com/pet-detect-239118/malaria.zip ./malaria.zip

with zipfile.ZipFile('malaria.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()
```

```
"""Set the folder for the different image classes"""

data_dir = './malaria'
data_dir = pathlib.Path(data_dir)
blood_imgs = list(data_dir.glob('*/*.png'))

print(f'There are {len(blood_imgs)} in total.')

positive_paths = []
negative_paths = []
for file in blood_imgs:
  file = str(file)
  parts = tf.strings.split(file, os.path.sep)
  if parts[-2] == 'Parasitemic':
    positive_paths.append('/content/'+file)
  else:
    negative_paths.append('/content/'+file)

"""randomly select 2000 images from each class for test, note that we will use the images
in the test dataset to train the GAN"""

total = len(positive_paths)
```

```
test_idx = np.random.choice(total, 2000, replace=False)

# move the images to their correspoding folders
positive_paths = np.array(positive_paths)
negative_paths = np.array(negative_paths)
test_positive = np.take(positive_paths, test_idx, axis=0)
train_positive = np.delete(positive_paths, test_idx, axis=0)
test_negative = np.take(negative_paths, test_idx, axis=0)
train_negative = np.delete(negative_paths, test_idx, axis=0)

train_images = np.concatenate((train_positive, train_negative), axis=0)
test_images = np.concatenate((test_positive, test_negative), axis=0)
print(train_images.shape)
print(test_images.shape)

"""Setup the folders for the following tasks"""

if not os.path.exists('train'):
  os.mkdir('train')

if not os.path.exists('test'):
  os.mkdir('test')
```

```
os.mkdir('train/Parasitemic')
os.mkdir('test/Parasitemic')
os.mkdir('train/Uninfected')
os.mkdir('test/Uninfected')

for file in train_images:
  parts = str.split(file, os.path.sep)
  cp_path = parts[-2]+'/'+parts[-1]
  root = '/content/train'+'/'
  dest = root+cp_path
  src = file
  shutil.copy2(src, dest)

for file in test_images:
  parts = str.split(file, os.path.sep)
  cp_path = parts[-2]+'/'+parts[-1]
  root = '/content/test'+'/'
  dest = root+cp_path
  src = file
  shutil.copy2(src, dest)
```

```
"""Set the helper functions for image processing"""


def decode_img(img):
  # convert the compressed string to a 3D uint8 tensor
  img = tf.image.decode_png(img, channels=3)
  # Use `convert_image_dtype` to convert to floats in the [0,1] range.
  img = tf.image.convert_image_dtype(img, tf.float32)
  # resize the image to the desired size.
  return tf.image.resize(img, [32, 32])


def get_label(file_path):
  parts = tf.strings.split(file_path, os.path.sep)
  if parts[-2] == 'Parasitemic':
    return tf.constant(1.0, dtype="float64")
  else:
    return tf.constant(0.0, dtype="float64")


def process_path(file_path):
  label = get_label(file_path)
  # load the raw data from the file as a string
  img = tf.io.read_file(file_path)
  img = decode_img(img)
```

```
  # rescale from (0,255) to (-1,1)
  img = (img - 127.5) / 127.5
  return img, label

# setup the dataset iterator for positive / negative
BATCH_SIZE = 512
train_BUFFER_SIZE = 20820
test_BUFFER_SIZE = 4000

train_dataset = tf.data.Dataset.list_files("/content/train/*/*.png")
train_dataset = train_dataset.map(process_path,
num_parallel_calls=tf.data.AUTOTUNE)
train_dataset = train_dataset.shuffle(train_BUFFER_SIZE).batch(BATCH_SIZE)

test_dataset = tf.data.Dataset.list_files("/content/test/*/*.png")
test_dataset = test_dataset.map(process_path, num_parallel_calls=tf.data.AUTOTUNE)
test_dataset = test_dataset.shuffle(test_BUFFER_SIZE).batch(BATCH_SIZE)

images, labels = next(iter(test_dataset))
print(images.shape)
print(labels.shape)
```

```
plt.figure(figsize=(10,10))
title=['Parasitemic', 'Uninfected']
for i in range(6 * 6):
  plt.subplot(6, 6, 1 + i)
  plt.axis(False)
  image = tf.keras.preprocessing.image.array_to_img(images[i])
  # plt.title(title[np.argmax(labels[i])])
  if labels[i] == 1:
    plt.title(title[0])
  else:
    plt.title(title[1])
  plt.imshow(image)
plt.show()

"""Define and train the classifier"""

# function for creating an identity or projection residual module
def residual_module(layer_in, n_filters):
  merge_input = layer_in
  # check if the number of filters needs to be increase, assumes channels last format
  if layer_in.shape[-1] != n_filters:
```

```
    merge_input = tfkl.Conv2D(n_filters, (1,1), padding='same', activation='relu',
kernel_initializer='he_normal')(layer_in)
  # conv1
  conv1 = tfkl.Conv2D(n_filters, (3,3), padding='same', activation='relu',
kernel_initializer='he_normal')(layer_in)
  # conv2
  conv2 = tfkl.Conv2D(n_filters, (3,3), padding='same', activation='linear',
kernel_initializer='he_normal')(conv1)
  # add filters, assumes filters/channels last
  layer_out = tfk.layers.Add()([conv2, merge_input])
  # activation function
  layer_out = tfkl.Activation('relu')(layer_out)
  return layer_out

def define_classifier(input_dim=(32,32,3)):
  input_layer = tfk.Input(shape=input_dim)
  layer = tfkl.Lambda(lambda x: x*127.5+127.5)(input_layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
  layer = tfkl.MaxPooling2D()(layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
```

```
  layer = tfkl.MaxPooling2D()(layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
  layer = tfkl.MaxPooling2D()(layer)
  layer = tfkl.Flatten()(layer)
  layer = tfkl.Dense(128, activation='tanh')(layer)
  layer = tfkl.Dropout(0.4)(layer)
  layer = tfkl.Dense(128, activation='tanh')(layer)
  layer = tfkl.Dropout(0.4)(layer)
  layer = tfkl.Dense(2)(layer)
  model = tfk.models.Model(inputs=input_layer, outputs=layer)
  return model

classifier = define_classifier()
classifier.summary()

tfk.utils.plot_model(classifier, show_shapes=True, dpi=64)

classifier.compile(
    optimizer=tfk.optimizers.Adam(learning_rate=1e-4),
    loss=tfk.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'])
```

```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=20)

history = classifier.fit(train_dataset, validation_data=test_dataset, epochs=100,
callbacks=[callback], verbose=2)

classifier.evaluate(test_dataset)

plt.figure(figsize=(10, 8))
plt.subplot(2, 1, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.ylim([0, 2.5])
plt.legend(loc='best')
plt.title('Training and Validation Loss')

plt.subplot(2, 1, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.ylim([0, 1.2])
plt.plot()
plt.legend(loc='best')
plt.title('Training and Validation Accuracy')
```

```
plt.xlabel('epoch')

"""Save the trained model, you can zip and copy it to you google drive by uncommenting
the code below"""

classifier.save('classify_malaria_32')

!zip -r classify_malaria_32.zip classify_malaria_32

from google.colab import drive
drive.mount('/content/drive')

import shutil

source = "/content/classify_malaria_64.zip"
destination = "/content/drive/MyDrive/classify_malaria_64.zip"
shutil.copy2(source, destination)
```

Appendix C: **Residual Network for COVID-19 Chest X-Ray Image Classification**

```
# -*- coding: utf-8 -*-

"""Original file is located at

    https://github.com/StanleyLiangYork/GAN_for_Medical_Image/blob/main/adaptive_c

ycle_gan_malaria.ipynb

"""

!pip install tensorflow_addons

!pip install sewar

from sewar.full_ref import mse, rmse, psnr, uqi, ssim, ergas, scc, rase, sam, msssim, vifp

import os

import shutil

import random

import pandas as pd

import numpy as np

import seaborn as sns

import tensorflow as tf

import zipfile

import random

from PIL import Image

from matplotlib import pyplot as plt

import re
```

```
import tensorflow.keras as tfk
import tensorflow.keras.layers as tfkl
import tensorflow_hub as hub
import cv2
import tensorflow_addons as tfa
import tensorflow_probability as tfp

print(f'Tensorflow Version: {tf.__version__}')

"""Set the random seed for replication"""

tf.random.set_seed(100)
AUTOTUNE = tf.data.AUTOTUNE

"""Fetch the COVID-19 X-Ray dataset"""

if not os.path.exists('covid_set.zip'):
  !wget https://storage.googleapis.com/pet-detect-239118/covid_set.zip ./covid_set.zip

with zipfile.ZipFile('covid_set.zip') as ZipObj:
  ZipObj.extractall()
```

```
"""Set the folders for the experiment"""

root_dir = '/content/covid_set'
paths = os.listdir(root_dir)

covid = re.compile("COVID")
normal = re.compile("NORMAL")
viral = re.compile("Viral")

covid_path = []
normal_path = []
viral_path = []

for path in paths:
  if covid.match(path):
    covid_path.append(path)
  if normal.match(path):
    normal_path.append(path)
  if viral.match(path):
    viral_path.append(path)

val_covid_path = covid_path[:50]
```

```
covid_path = covid_path[50:]
print(len(val_covid_path))
print(len(covid_path))

"""Since we have just a few COVID-19 X-ray images, we separate 50 images for
validation, and the rest 169 for training"""

for _ in range(5):
  random_items = random.sample(covid_path, 169)
  covid_path += random_items

print(len(covid_path))

"""Randomly resample the images

Build a balanced dataset, each class has 1014 images respectively
"""

for i, path in enumerate(covid_path):
  covid_path[i] = root_dir + '/' + path

for i, path in enumerate(normal_path):
```

```
  normal_path[i] = root_dir + '/' + path

for i, path in enumerate(viral_path):
  viral_path[i] = root_dir + '/' + path

# 1014 + 50 = 1064 -- need 50 extra images from normal and from viral classes for the validation dataset
covid_path = covid_path
normal_path = normal_path[:1064]
viral_path = viral_path[:1064]

print(len(covid_path))
print(len(normal_path))
print(len(viral_path))

"""The helper function the resize and rescale the images.<p>
labels: COVID-0, NORMAL-1, VIRAL-2
"""

def decode_img(img):
  # convert the compressed string to a 3D uint8 tensor
  img = tf.image.decode_png(img, channels=3)
```

```
  # Use `convert_image_dtype` to convert to floats in the [0,1] range.
  img = tf.image.convert_image_dtype(img, tf.float32)
  # resize the image to the desired size.
  return tf.image.resize(img, [64, 64])


def get_label(file_path):
  if tf.strings.regex_full_match(file_path, ".*COVID.*"):
    return tf.constant(0.0, dtype="float32")
  elif tf.strings.regex_full_match(file_path, ".*NORMAL.*"):
    return tf.constant(1.0, dtype="float32")
  else:
    return tf.constant(2.0, dtype="float32")


def process_path(file_path):
  label = get_label(file_path)
  # load the raw data from the file as a string
  img = tf.io.read_file(file_path)
  img = decode_img(img)


  # rescale from (0,255) to (0,1)
  # img = img / 255.0
  img = (img - 127.5) / 127.5
```

```
  return img, label


"""The three image datasets for each image class """


covid_ds = tf.data.Dataset.list_files(covid_path, shuffle=True)
normal_ds = tf.data.Dataset.list_files(normal_path[:1014], shuffle=True)
viral_ds = tf.data.Dataset.list_files(viral_path[:1014], shuffle=True)


BATCH_SIZE = 64
BUFFER_SIZE = 1014
AUTOTUNE = tf.data.AUTOTUNE


covid_ds = covid_ds.map(process_path,
num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
normal_ds = normal_ds.map(process_path,
num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
viral_ds = viral_ds.map(process_path,
num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)


train_path = covid_path + normal_path[:1014] + viral_path[:1014]
val_path = val_covid_path + normal_path[1014:] + viral_path[1014:]
print(len(train_path))
```

```
print(len(val_path))

train_ds = tf.data.Dataset.list_files(train_path, shuffle=True)
train_ds = train_ds.map(process_path,
num_parallel_calls=AUTOTUNE).shuffle(3042).batch(BATCH_SIZE)
val_ds = tf.data.Dataset.list_files(val_path, shuffle=True)
val_ds = val_ds.map(process_path,
num_parallel_calls=AUTOTUNE).shuffle(150).batch(BATCH_SIZE)

images, labels = next(iter(train_ds))

"""visualize the images"""

plt.figure(figsize=(12,12))

for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  image = tf.keras.preprocessing.image.array_to_img(images[i,:,:,:])
  plt.imshow(image)
  if labels[i] == 0.0:
    plt.title('COVID')
```

```
  if labels[i] == 1.0:
    plt.title("Normal")
  if labels[i] == 2.0:
    plt.title("Viral")


"""Train the classifier later as the criterion for the GAN"""


# function for creating an identity or projection residual module
def residual_module(layer_in, n_filters):
  merge_input = layer_in
  # check if the number of filters needs to be increase, assumes channels last format
  if layer_in.shape[-1] != n_filters:
    merge_input = tfkl.Conv2D(n_filters, (1,1), padding='same', activation='relu',
kernel_initializer='he_normal')(layer_in)
  # conv1
  conv1 = tfkl.Conv2D(n_filters, (3,3), padding='same', activation='relu',
kernel_initializer='he_normal')(layer_in)
  # conv2
  conv2 = tfkl.Conv2D(n_filters, (3,3), padding='same', activation='linear',
kernel_initializer='he_normal')(conv1)
  # add filters, assumes filters/channels last
  layer_out = tfk.layers.Add()([conv2, merge_input])
```

```
  # activation function
  layer_out = tfkl.Activation('relu')(layer_out)
  return layer_out


def define_classifier(input_dim=(64,64,3)):
  input_layer = tfk.Input(shape=input_dim)
  layer = tfkl.Lambda(lambda x: x*127.5+127.5)(input_layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
  layer = tfkl.MaxPooling2D()(layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
  layer = tfkl.MaxPooling2D()(layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
  layer = tfkl.MaxPooling2D()(layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
  layer = tfkl.MaxPooling2D()(layer)
  layer = tfkl.Flatten()(layer)
  layer = tfkl.Dense(128, activation='tanh')(layer)
  layer = tfkl.Dropout(0.4)(layer)
```

```
  layer = tfkl.Dense(3)(layer)
  model = tfk.models.Model(inputs=input_layer, outputs=layer)
  return model

classifier = define_classifier()
classifier.summary()

"""Compile the classifer DNN with sparce categorical crossentropy as the loss function,
the input label with shape (batch, 1), the DNN output with shape (batch, 3)"""

classifier.compile(
    optimizer=tfk.optimizers.Adam(learning_rate=1e-4),
    loss=tfk.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'])
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

"""Optimize with 100 epochs"""

history = classifier.fit(train_ds, validation_data=val_ds, epochs=100,
callbacks=[callback], verbose=2)

# visualize the training procedure
```

```
plt.figure(figsize=(12, 8))
plt.subplot(2, 1, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.ylim([0, 2.5])
plt.legend(loc='best')
plt.title('Training and Validation Loss')


plt.subplot(2, 1, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.ylim([0, 1.2])
plt.plot()
plt.legend(loc='best')
plt.title('Training and Validation Accuracy')
plt.xlabel('epoch')
plt.savefig('train.png')


# save and zip the classifier
classifier.save('covid_classifier_64')
!zip -r covid_classifier_64.zip covid_classifier_64
```

```
"""Now we setup the Cycle-GAN with criterion"""

from zipfile import ZipFile

with ZipFile('covid_classifier_64.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()

# delete and reload the pretrained classifier
# del classifier
classifier = tfk.models.load_model('covid_classifier_64')
classifier.trainable = False

classifier.evaluate(train_ds)

# check the pretrained classifier with the validation dataset
classifier.evaluate(val_ds)
```

Appendix D: **Convolutional Variation Autoencoder for Malaria Parasitemic Blood Cell image Synthesis**

```
# -*- coding: utf-8 -*-

"""Original file is located at

    https://github.com/StanleyLiangYork/GAN_for_Medical_Image/blob/main/CVA_mal

aria.ipynb

"""



!pip install tensorflow_addons

!pip install -q tensorflow-probability



# to generate gifs

!pip install -q imageio

!pip install -q git+https://github.com/tensorflow/docs



from IPython import display

from IPython.display import clear_output

import os

from zipfile import ZipFile

import glob

import imageio

import matplotlib.pyplot as plt
```

```
import numpy as np

import PIL

import tensorflow as tf

import tensorflow_probability as tfp

import time

import tensorflow_addons as tfa

import pathlib

from tensorflow.keras import layers


tfd = tfp.distributions

tfpl = tfp.layers

tfk = tf.keras

tfkl = tf.keras.layers


AUTOTUNE = tf.data.AUTOTUNE

print(f'Tensorflow Version: {tf.__version__}')


"""load the malaria dataset"""


if not os.path.exists('malaria.zip'):

  !wget https://storage.googleapis.com/pet-detect-239118/malaria.zip ./malaria.zip
```

```
with ZipFile('malaria.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()

data_dir = './malaria'
data_dir = pathlib.Path(data_dir)
blood_imgs = list(data_dir.glob('*/*.png'))

print(f'There are {len(blood_imgs)} in total.')

file_list = list(data_dir.glob('*/*.png'))
positive_paths = []
negative_paths = []
for file in file_list:
  file = str(file)
  parts = tf.strings.split(file, os.path.sep)
  if parts[-2] == 'Parasitemic':
    positive_paths.append('/content/'+file)
  else:
    negative_paths.append('/content/'+file)

total = len(positive_paths)
```

```
test_idx = np.random.choice(total, 4000, replace=False)
print(f'Total image: {total*2}')

positive_paths = np.array(positive_paths)
negative_paths = np.array(negative_paths)
test_positive = np.take(positive_paths, test_idx, axis=0)
train_positive = np.delete(positive_paths, test_idx, axis=0)
test_negative = np.take(negative_paths, test_idx, axis=0)
train_negative = np.delete(negative_paths, test_idx, axis=0)

train_images = np.concatenate((train_positive, train_negative), axis=0)
test_images = np.concatenate((test_positive, test_negative), axis=0)
print(train_images.shape)
print(test_images.shape)

if not os.path.exists('train'):
  os.mkdir('train')

if not os.path.exists('test'):
  os.mkdir('test')

os.mkdir('train/Parasitemic')
```

```
os.mkdir('test/Parasitemic')
os.mkdir('train/Uninfected')
os.mkdir('test/Uninfected')

import shutil

for file in train_images:
  parts = str.split(file, os.path.sep)
  cp_path = parts[-2]+'/'+parts[-1]
  root = '/content/train'+'/'
  dest = root+cp_path
  src = file
  shutil.copy2(src, dest)

for file in test_images:
  parts = str.split(file, os.path.sep)
  cp_path = parts[-2]+'/'+parts[-1]
  root = '/content/test'+'/'
  dest = root+cp_path
  src = file
  shutil.copy2(src, dest)
```

```
# Run this to reset the VAE run
def decode_img(img):
  # convert the compressed string to a 3D uint8 tensor
  img = tf.image.decode_png(img, channels=3)
  # Use `convert_image_dtype` to convert to floats in the [0,1] range.
  img = tf.image.convert_image_dtype(img, tf.float32)
  # resize the image to the desired size.
  return tf.image.resize(img, [32, 32])


def get_label(file_path):
  parts = tf.strings.split(file_path, os.path.sep)
  if parts[-2] == 'Parasitemic':
    return tf.constant(1.0, dtype="float32")
  else:
    return tf.constant(0.0, dtype="float32")


def process_path(file_path):
  label = get_label(file_path)
  # load the raw data from the file as a string
  img = tf.io.read_file(file_path)
  img = decode_img(img)
  # rescale from (0,255) to (0,1)
```

```
  img = (img / 127.5) - 1
  # img = img / 255.0
  return img, label

def CVA_process_path(file_path):
  label = get_label(file_path)
  # load the raw data from the file as a string
  img = tf.io.read_file(file_path)
  img = decode_img(img)
  # rescale from (0,255) to (0,1)
  # img = (img / 127.5) - 1
  img = img / 255.0
  return img, label

"""Set up the datasets"""

BATCH_SIZE = 512
BUFFER_SIZE = 2000

train_dataset = tf.data.Dataset.list_files("/content/train/Parasitemic/*.png")
train_dataset = train_dataset.map(CVA_process_path,
num_parallel_calls=tf.data.AUTOTUNE)
```

```
# train_dataset = train_dataset.map(process_path,
num_parallel_calls=tf.data.AUTOTUNE)
train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

test_dataset = tf.data.Dataset.list_files("/content/test/Parasitemic/*.png")
test_dataset = test_dataset.map(CVA_process_path,
num_parallel_calls=tf.data.AUTOTUNE)
# test_dataset = test_dataset.map(process_path,
num_parallel_calls=tf.data.AUTOTUNE)
test_dataset = test_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

# VAE cost function as negative ELBO (Evidence Lower Bound Objective)

def normal_log_pdf(sample, mean, sd, raxis=1):
  log2pi = tf.math.log(2. * np.pi)
  logvar = np.log((np.square(sd)))
  return tf.reduce_sum(
      -.5 * ((sample - mean) ** 2. * tf.exp(-logvar) + logvar + log2pi),
      axis=raxis)

def vae_cost(x_true, model, analytic_kl=True, kl_weight=0.01):
  z_sample, mu, sd = model.encode(x_true)
```

```
  x_recons_logits = model.decoder(z_sample)
  # compute cross entropy loss for each dimension of every datapoint
  raw_cross_entropy = tf.nn.sigmoid_cross_entropy_with_logits(labels=x_true,
logits=x_recons_logits) # bs*128*128*3
  # compute cross entropy loss for all instances in mini-batch; shape=(batch_size,), the
first term of the objective
  neg_log_likelihood = tf.math.reduce_sum(raw_cross_entropy, axis=[1, 2, 3]) # the first
term of the objective

  # compute reverse KL divergence, either analytically or through MC approximation
with one sample, the second term of the objective
  if analytic_kl:
    kl_divergence = - 0.5 * tf.math.reduce_sum(1 + tf.math.log(tf.math.square(sd)) -
tf.math.square(mu) - tf.math.square(sd), axis=1)  # shape=(batch_size, )
  else:
    logpz = normal_log_pdf(z_sample, 0., 1.)  # shape=(batch_size,)
    logqz_x = normal_log_pdf(z_sample, mu, tf.math.square(sd))  # shape=(batch_size,)
    kl_divergence = logqz_x - logpz
  elbo = tf.math.reduce_mean(-kl_weight * kl_divergence - neg_log_likelihood)  #
shape=()
  return -elbo
```

```
# Adjust the KL divergence weight here
# @tf.function
def train_step(x_true, model, optimizer, analytic_kl=True, kl_weight=0.01):
  with tf.GradientTape() as tape:
    cost_mini_batch = vae_cost(x_true, model, analytic_kl, kl_weight)
  gradients = tape.gradient(cost_mini_batch, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))
  return cost_mini_batch


class Encoder_Z(tfk.layers.Layer):


    def __init__(self, dim_z, name="encoder", **kwargs):
        super(Encoder_Z, self).__init__(name=name, **kwargs)
        self.dim_x = (32, 32, 3)
        self.dim_z = dim_z


    def build(self):
        layers = [tfkl.InputLayer(input_shape=self.dim_x)]
        layers.append(tfkl.Conv2D(filters=32, kernel_size=4, strides=(2, 2),
padding='same')) # 16*16*32
        layers.append(tfkl.LeakyReLU())
```

```
        layers.append(tfkl.Conv2D(filters=64, kernel_size=4, strides=(2, 2),
padding='same')) # 8*8*64
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2D(filters=128, kernel_size=4, strides=(2, 2),
padding='same')) # 4*4*128
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Flatten())
        layers.append(tfkl.Dense(self.dim_z * 2,
                                 activation=None))  # *2 because number of parameters for both
mean and (raw) standard deviation
        return tfk.Sequential(layers)


class Decoder_X(tfk.layers.Layer):


    def __init__(self, dim_z, name="decoder", **kwargs):
        super(Decoder_X, self).__init__(name=name, **kwargs)
        self.dim_z = dim_z


    def build(self):
        layers = [tfkl.InputLayer(input_shape=(self.dim_z,))]
        layers.append(tfkl.Dense(4 * 4 * 16, activation=None))
        layers.append(tfkl.Reshape((4, 4, 16)))
```

```
        layers.append(tfkl.Conv2DTranspose(filters=32, kernel_size=4, strides=2,
padding='same')) # 8*8*32
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2DTranspose(filters=64, kernel_size=4, strides=2,
padding='same')) # 16*16*64
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2DTranspose(filters=128, kernel_size=4, strides=2,
padding='same')) # 32*32*128
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2DTranspose(filters=3, kernel_size=4, strides=1,
padding='same'))

        return tfk.Sequential(layers)

class VAE(tfk.Model):

    def __init__(self, dim_z, learning_rate, seed=2000, name="autoencoder", **kwargs):
        super(VAE, self).__init__(name=name, **kwargs)
        self.dim_x = (32, 32, 3)
        self.dim_z = dim_z
        self.learning_rate = learning_rate
        self.seed = seed
```

```
    self.encoder = Encoder_Z(dim_z=self.dim_z).build()
    self.decoder = Decoder_X(dim_z=self.dim_z).build()

@tf.function
def sample(self, eps=None):
  if eps is None:
    eps = tf.random.normal(shape=(100, self.dim_z))
  return self.decode(eps, apply_sigmoid=True)

def encode(self, x_input):
    mu, rho = tf.split(self.encoder(x_input), num_or_size_splits=2, axis=1)
    sd = tf.math.log(1 + tf.math.exp(rho))
    z_sample = mu + sd * tf.random.normal(shape=(self.dim_z,))
    return z_sample, mu, sd

def decode(self, z, apply_sigmoid=False):
    logits = self.decoder(z)
    if apply_sigmoid:
      probs = tf.sigmoid(logits)
      return probs
    return logits
```

```
autoencoder = VAE(128, 2e-4)
optimizer = tf.keras.optimizers.Adam(2e-4)

def generate_and_save_images(model, epoch, test_sample):
  z, mean, logvar = model.encode(test_sample)
  predictions = model.sample(z)
  fig = plt.figure(figsize=(8, 8))

  for i in range(16):
    plt.subplot(4, 4, i + 1)
    show_img = tf.keras.preprocessing.image.array_to_img(predictions[i, :, :, :])
    plt.imshow(show_img)
    plt.axis('off')

  # tight_layout minimizes the overlap between 2 sub-plots
  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()

images, labels = next(iter(test_dataset))
images.shape

z, mean, logvar = autoencoder.encode(images)
```

```
predictions = autoencoder.sample(z)
fig = plt.figure(figsize=(8, 8))

for i in range(16):
  plt.subplot(4, 4, i + 1)
  if i % 2 == 0:
    show_img = tf.keras.preprocessing.image.array_to_img(images[i, :, :, :])
    plt.title('Real Image')
    plt.imshow(show_img)
    plt.axis('off')
  else:
    show_img = tf.keras.preprocessing.image.array_to_img(predictions[i, :, :, :])
    plt.title('Decoded Image')
    plt.imshow(show_img)
    plt.axis(False)

"""Train the CVA"""

epochs = 600
images, labels = next(iter(test_dataset))
test_sample = images
```

```
generate_and_save_images(autoencoder, 0, test_sample)
vae_history = dict()
vae_history['loss'] = []
vae_history['val_loss'] = []

for epoch in range(1, epochs + 1):

 start_time = time.time()
 loss = []
 for train_x, _ in train_dataset:
  temp_loss = train_step(train_x, autoencoder, optimizer)
  loss.append(temp_loss)

 loss = np.array(loss)
 end_time = time.time()
 elbo = np.mean(loss)
 vae_history['loss'].append(elbo)
 print(f'epoch -- {epoch}: Loss: {elbo}')

 val_loss = tf.keras.metrics.Mean()
 for test_x, _ in test_dataset:
  val_loss(vae_cost(test_x, autoencoder))
```

```
  val_elbo = -val_loss.result()
  vae_history['val_loss'].append(val_elbo)
  display.clear_output(wait=False)
  print('Epoch: {}, Test set ELBO: {}, time elapse for current epoch: {}'
        .format(epoch, val_elbo, end_time - start_time))
  generate_and_save_images(autoencoder, epoch, test_sample)

"""save the trained encoder and decoder"""

autoencoder.encoder.save('CVA_encoder_32')
autoencoder.decoder.save('CVA_decoder_32')
!zip -r CVA_decoder_32.zip CVA_decoder_32
!zip -r CVA_encoder_32.zip CVA_encoder_32

plt.figure(figsize=(12, 8))
plt.plot(vae_history['loss'], label='Evidence Lower Bound loss')
# plt.plot(vae_history['val_loss'], label='Validation ELBO loss')
plt.legend(loc='best')
plt.title('CVA training')
plt.xlabel('epoch')

show_images, labels = next(iter(test_dataset))
```

```
plt.figure(figsize=(12,12))


for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  image = tf.keras.preprocessing.image.array_to_img(show_images[i,:,:,:])
  plt.imshow(image)
  if labels[i] == 1.0:
    plt.title('Parasitemic')
  else:
    plt.title('Uninfected')


def CVA_rescale(image_tensor):
  img = tf.multiply(image_tensor, 127.5)
  img = tf.add(img, 127.5)
  img = tf.divide(img, 255.0)
  return img


z, mean, logvar = autoencoder.encode(show_images)
predictions = autoencoder.sample(z)
fig = plt.figure(figsize=(8, 8))
```

```
for i in range(16):
  plt.subplot(4, 4, i + 1)
  if i % 2 == 0:
    show_img = tf.keras.preprocessing.image.array_to_img(show_images[i, :, :, :])
    plt.title('Real Image')
    plt.imshow(show_img)
    plt.axis('off')
  else:
    show_img = tf.keras.preprocessing.image.array_to_img(predictions[i, :, :, :])
    plt.title('Decoded Image')
    plt.imshow(show_img)
    plt.axis(False)


!pip install sewar


from sewar.full_ref import mse, rmse, psnr, uqi, ssim, ergas, scc, rase, sam, msssim, vifp


input_images = []
generated_images = []
for i in range(show_images.shape[0]):
```

```
  input_images.append(tf.keras.preprocessing.image.array_to_img(show_images[i, :, :,
:]).convert('L'))
  generated_images.append(tf.keras.preprocessing.image.array_to_img(predictions[i, :, :,
:]).convert('L'))

MSE = []
RMSE = []
PSNR = []
UQI = []
SCC = []
RASE =[]
SAM = []
VIF = []

for j in range(len(input_images)):
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
```

```
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
  VIF.append(vifp(gen, org))

MSE = np.array(MSE)
RMSE = np.array(RMSE)
PSNR = np.array(PSNR)
UQI = np.array(UQI)
SCC = np.array(SCC)
RASE = np.array(RASE)
SAM = np.array(SAM)
VIF = np.array(VIF)

print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")
print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")
print(f"PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ")
print(f"UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ")
print(f"SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ")
print(f"RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ")
print(f"SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ")
print(f"VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ")
```

```
def get_accuracy(g_true, preds):
  pred_idx = tf.argmax(preds, axis=1).numpy()
  count = 0
  for tab, pred in zip(g_true, pred_idx):
    if tab == pred:
      count += 1
  return count / preds.shape[0]

!wget https://storage.googleapis.com/pet-detect-239118/classify_malaria_32.zip
./classify_malaria_32.zip

with ZipFile('classify_malaria_32.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()

classifier = tfk.models.load_model('classify_malaria_32')
classifier.trainable = False

preds = classifier(predictions)

print(f"classify accuracy: {get_accuracy(labels, preds)}")
```

```
"""FID score"""

from numpy import cov
from numpy import trace
from numpy import iscomplexobj
from numpy import asarray
from numpy.random import shuffle
from scipy.linalg import sqrtm
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.datasets.mnist import load_data

# scale an array of images to a new size
def scale_images(images, new_shape):
 images_list = list()
 for image in images:
  # resize with nearest neighbor interpolation
  new_image = tf.image.resize(image, new_shape)
  # store
  images_list.append(new_image)
 return asarray(images_list)
```

```
# calculate frechet inception distance
def calculate_fid(model, images1, images2):
  # calculate activations
  act1 = model.predict(images1)
  act2 = model.predict(images2)
  # calculate mean and covariance statistics
  mu1, sigma1 = act1.mean(axis=0), cov(act1, rowvar=False)
  mu2, sigma2 = act2.mean(axis=0), cov(act2, rowvar=False)
  # calculate sum squared difference between means
  ssdiff = np.sum((mu1 - mu2)**2.0)
  # calculate sqrt of product between cov
  covmean = sqrtm(sigma1.dot(sigma2))
  # check and correct imaginary numbers from sqrt
  if iscomplexobj(covmean):
    covmean = covmean.real
  # calculate score
  fid = ssdiff + trace(sigma1 + sigma2 - 2.0 * covmean)
  return fid

model = InceptionV3(include_top=False, pooling='avg', input_shape=(299,299,3))
FID = []
for images, labels in test_dataset:
```

```
  images1 = images
  z, mean, logvar = autoencoder.encode(images1)
  images2 = autoencoder.sample(z)
  images1 = preprocess_input(images1)
  images2 = preprocess_input(images2)
  fid = calculate_fid(classifier, images1, images2)
  FID.append(fid)

FID = np.array(FID)
print("CVA Malaria------")
print(f'Adjusted FID Mean: {FID.mean()}, Std: {FID.std()}')
```

Appendix E: **Convolutional Variation Autoencoder for COVID-19 Chest X-Ray Image Synthesis**

```
# -*- coding: utf-8 -*-

"""Original file is located at

    https://github.com/StanleyLiangYork/GAN_for_Medical_Image/blob/main/CVA_mal

aria.ipynb

"""



!pip install tensorflow_addons

!pip install -q tensorflow-probability



# to generate gifs

!pip install -q imageio

!pip install -q git+https://github.com/tensorflow/docs



from IPython import display

from IPython.display import clear_output

import os

import glob

import imageio

import matplotlib.pyplot as plt

import numpy as np
```

```
import PIL
import tensorflow as tf
import tensorflow_probability as tfp
import time
import tensorflow_addons as tfa
import pathlib
from tensorflow.keras import layers
import shutil
import pandas as pd
import seaborn as sns
import tensorflow as tf
import random
from PIL import Image
from matplotlib import pyplot as plt
import re


tfd = tfp.distributions
tfpl = tfp.layers
tfk = tf.keras
tfkl = tf.keras.layers


AUTOTUNE = tf.data.AUTOTUNE
```

```
print(f'Tensorflow Version: {tf.__version__}')

tf.random.set_seed(100)

"""Fetch the COVID-19 data"""

from zipfile import ZipFile

if not os.path.exists('covid_set.zip'):
  !wget https://storage.googleapis.com/pet-detect-239118/covid_set.zip ./covid_set.zip

with ZipFile('covid_set.zip') as ZipObj:
  ZipObj.extractall()

"""Set up the folder for experiment"""

root_dir = '/content/covid_set'
paths = os.listdir(root_dir)

covid = re.compile("COVID")
normal = re.compile("NORMAL")
viral = re.compile("Viral")
```

```
covid_path = []
normal_path = []
viral_path = []

for path in paths:
  if covid.match(path):
    covid_path.append(path)
  if normal.match(path):
    normal_path.append(path)
  if viral.match(path):
    viral_path.append(path)

val_covid_path = covid_path[:50]
covid_path = covid_path[50:]
print(len(val_covid_path))
print(len(covid_path))

"""Build a balanced dataset, each class has 1014 images respectively <p>"""

for _ in range(5):
  random_items = random.sample(covid_path, 169)
```

```
  covid_path += random_items

print(len(covid_path))

for i, path in enumerate(covid_path):
  covid_path[i] = root_dir + '/' + path

for i, path in enumerate(normal_path):
  normal_path[i] = root_dir + '/' + path

for i, path in enumerate(viral_path):
  viral_path[i] = root_dir + '/' + path

# 1014 + 50 = 1064 -- need 50 extra images from normal and from viral classes for the
validation dataset
covid_path = covid_path
normal_path = normal_path[:1064]
viral_path = viral_path[:1064]

print(len(covid_path))
print(len(normal_path))
print(len(viral_path))
```

```
# Run this to reset the VAE run
def decode_img(img):
  # convert the compressed string to a 3D uint8 tensor
  img = tf.image.decode_png(img, channels=3)
  # Use `convert_image_dtype` to convert to floats in the [0,1] range.
  img = tf.image.convert_image_dtype(img, tf.float32)
  # resize the image to the desired size.
  return tf.image.resize(img, [64, 64])

def get_label(file_path):
  if tf.strings.regex_full_match(file_path, ".*COVID.*"):
    return tf.constant(0.0, dtype="float32")
  elif tf.strings.regex_full_match(file_path, ".*NORMAL.*"):
    return tf.constant(1.0, dtype="float32")
  else:
    return tf.constant(2.0, dtype="float32")

def process_path(file_path):
  label = get_label(file_path)
  # load the raw data from the file as a string
  img = tf.io.read_file(file_path)
```

```
  img = decode_img(img)
  # rescale from (0,255) to (0,1)
  img = (img - 127.5) / 127.5
  return img, label

def CVA_process_path(file_path):
  label = get_label(file_path)
  # load the raw data from the file as a string
  img = tf.io.read_file(file_path)
  img = decode_img(img)
  img = img / 255.0
  return img, label

covid_ds = tf.data.Dataset.list_files(covid_path, shuffle=True)
normal_ds = tf.data.Dataset.list_files(normal_path[:1014], shuffle=True)
viral_ds = tf.data.Dataset.list_files(viral_path[:1014], shuffle=True)

BATCH_SIZE = 64
BUFFER_SIZE = 1014

covid_ds = covid_ds.map(CVA_process_path,
num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

```
normal_ds = normal_ds.map(CVA_process_path,
num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
viral_ds = viral_ds.map(CVA_process_path,
num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

# VAE cost function as negative ELBO (Evidence Lower Bound Objective)

def normal_log_pdf(sample, mean, sd, raxis=1):
  log2pi = tf.math.log(2. * np.pi)
  logvar = np.log((np.square(sd)))
  return tf.reduce_sum(
      -.5 * ((sample - mean) ** 2. * tf.exp(-logvar) + logvar + log2pi),
      axis=raxis)

def vae_cost(x_true, model, analytic_kl=True, kl_weight=0.01):
  z_sample, mu, sd = model.encode(x_true)
  x_recons_logits = model.decoder(z_sample)
  # compute cross entropy loss for each dimension of every datapoint
  raw_cross_entropy = tf.nn.sigmoid_cross_entropy_with_logits(labels=x_true,
logits=x_recons_logits) # bs*128*128*3
  # compute cross entropy loss for all instances in mini-batch; shape=(batch_size,), the
first term of the objective
```

```
  neg_log_likelihood = tf.math.reduce_sum(raw_cross_entropy, axis=[1, 2, 3]) # the first term of the objective

  # compute reverse KL divergence, either analytically or through MC approximation with one sample, the second term of the objective
  if analytic_kl:
    kl_divergence = - 0.5 * tf.math.reduce_sum(1 + tf.math.log(tf.math.square(sd)) - tf.math.square(mu) - tf.math.square(sd), axis=1)  # shape=(batch_size, )
  else:
    logpz = normal_log_pdf(z_sample, 0., 1.)  # shape=(batch_size,)
    logqz_x = normal_log_pdf(z_sample, mu, tf.math.square(sd))  # shape=(batch_size,)
    kl_divergence = logqz_x - logpz
  elbo = tf.math.reduce_mean(-kl_weight * kl_divergence - neg_log_likelihood)  # shape=()
  return -elbo

# Adjust the KL divergence weight here
def train_step(x_true, model, optimizer, analytic_kl=True, kl_weight=0.01):
  with tf.GradientTape() as tape:
    cost_mini_batch = vae_cost(x_true, model, analytic_kl, kl_weight)
  gradients = tape.gradient(cost_mini_batch, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

```
  return cost_mini_batch


class Encoder_Z(tfk.layers.Layer):


    def __init__(self, dim_z, name="encoder", **kwargs):
        super(Encoder_Z, self).__init__(name=name, **kwargs)
        self.dim_x = (64, 64, 3)
        self.dim_z = dim_z


    def build(self):
        layers = [tfkl.InputLayer(input_shape=self.dim_x)]
        layers.append(tfkl.Conv2D(filters=32, kernel_size=4, strides=(2, 2),
padding='same')) # 32*32*32
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2D(filters=64, kernel_size=4, strides=(2, 2),
padding='same')) # 16*16*64
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2D(filters=128, kernel_size=4, strides=(2, 2),
padding='same')) # 8*8*128
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2D(filters=256, kernel_size=4, strides=(2, 2),
padding='same')) # 4*4*256
```

```
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Flatten())
        layers.append(tfkl.Dense(self.dim_z * 2,
                                 activation=None))  # *2 because number of parameters for both
mean and (raw) standard deviation
        return tfk.Sequential(layers)


class Decoder_X(tfk.layers.Layer):


    def __init__(self, dim_z, name="decoder", **kwargs):
        super(Decoder_X, self).__init__(name=name, **kwargs)
        self.dim_z = dim_z


    def build(self):
        layers = [tfkl.InputLayer(input_shape=(self.dim_z,))]
        layers.append(tfkl.Dense(4 * 4 * 16, activation=None))
        layers.append(tfkl.Reshape((4, 4, 16)))
        layers.append(tfkl.Conv2DTranspose(filters=32, kernel_size=4, strides=2,
padding='same')) # 8*8*32
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2DTranspose(filters=64, kernel_size=4, strides=2,
padding='same')) # 16*16*64
```

```
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2DTranspose(filters=128, kernel_size=4, strides=2,
padding='same')) # 32*32*128
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2DTranspose(filters=256, kernel_size=4, strides=2,
padding='same')) # 64*64*256
        layers.append(tfkl.LeakyReLU())
        layers.append(tfkl.Conv2DTranspose(filters=3, kernel_size=4, strides=1,
padding='same'))

        return tfk.Sequential(layers)

class VAE(tfk.Model):

    def __init__(self, dim_z, learning_rate, seed=2000, name="autoencoder", **kwargs):
        super(VAE, self).__init__(name=name, **kwargs)
        self.dim_x = (128, 128, 3)
        self.dim_z = dim_z
        self.learning_rate = learning_rate
        self.seed = seed
        self.encoder = Encoder_Z(dim_z=self.dim_z).build()
        self.decoder = Decoder_X(dim_z=self.dim_z).build()
```

```
    @tf.function
    def sample(self, eps=None):
      if eps is None:
        eps = tf.random.normal(shape=(100, self.dim_z))
      return self.decode(eps, apply_sigmoid=True)

    def encode(self, x_input):
        mu, rho = tf.split(self.encoder(x_input), num_or_size_splits=2, axis=1)
        sd = tf.math.log(1 + tf.math.exp(rho))
        z_sample = mu + sd * tf.random.normal(shape=(self.dim_z,))
        return z_sample, mu, sd

    def decode(self, z, apply_sigmoid=False):
        logits = self.decoder(z)
        if apply_sigmoid:
          probs = tf.sigmoid(logits)
          return probs
        return logits

autoencoder = VAE(128, 2e-4)
optimizer = tf.keras.optimizers.Adam(2e-4)
```

```
def generate_and_save_images(model, epoch, test_sample):
  z, mean, logvar = model.encode(test_sample)
  predictions = model.sample(z)
  fig = plt.figure(figsize=(8, 8))

  for i in range(16):
    plt.subplot(4, 4, i + 1)
    show_img = tf.keras.preprocessing.image.array_to_img(predictions[i, :, :, :])
    plt.imshow(show_img)
    plt.axis('off')

  # tight_layout minimizes the overlap between 2 sub-plots
  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()

images, labels = next(iter(covid_ds))
images.shape

z, mean, logvar = autoencoder.encode(images)
predictions = autoencoder.sample(z)
fig = plt.figure(figsize=(8, 8))
```

```
for i in range(16):
  plt.subplot(4, 4, i + 1)
  if i % 2 == 0:
    show_img = tf.keras.preprocessing.image.array_to_img(images[i, :, :, :])
    plt.title('Real Image')
    plt.imshow(show_img)
    plt.axis('off')
  else:
    show_img = tf.keras.preprocessing.image.array_to_img(predictions[i, :, :, :])
    plt.title('Decoded Image')
    plt.imshow(show_img)
    plt.axis(False)

"""Train the CVA"""

epochs = 600
images, labels = next(iter(covid_ds))
test_sample = images

generate_and_save_images(autoencoder, 0, test_sample)
vae_history = dict()
```

```
vae_history['loss'] = []
# vae_history['val_loss'] = []

for epoch in range(1, epochs + 1):

  start_time = time.time()
  loss = []
  for train_x, _ in covid_ds:
    temp_loss = train_step(train_x, autoencoder, optimizer)
    loss.append(temp_loss)

  loss = np.array(loss)
  end_time = time.time()
  elbo = np.mean(loss)
  vae_history['loss'].append(elbo)
  print(f'epoch -- {epoch}: Loss: {elbo}')

  display.clear_output(wait=False)
  print('Epoch: {}, ELBO: {}, time elapse for current epoch: {}'
        .format(epoch, elbo, end_time - start_time))
  generate_and_save_images(autoencoder, epoch, test_sample)
```

```
autoencoder.encoder.save('CVA_encoder_64_600')

autoencoder.decoder.save('CVA_decoder_64_600')

!zip -r CVA_decoder_64_600.zip CVA_decoder_64_600

!zip -r CVA_encoder_64_600.zip CVA_encoder_64_600


plt.figure(figsize=(12, 8))

plt.plot(vae_history['loss'], label='Evidence Lower Bound loss')

plt.legend(loc='best')

plt.title('CVA training')

plt.xlabel('epoch')


def CVA_rescale(image_tensor):

  img = tf.multiply(image_tensor, 127.5)

  img = tf.add(img, 127.5)

  img = tf.divide(img, 255.0)

  return img


z, mean, logvar = autoencoder.encode(images)

predictions = autoencoder.sample(z)

fig = plt.figure(figsize=(8, 8))


for i in range(16):
```

```
 plt.subplot(4, 4, i + 1)
 if i % 2 == 0:
   show_img = tf.keras.preprocessing.image.array_to_img(images[i, :, :, :])
   plt.title('Real Image')
   plt.imshow(show_img)
   plt.axis('off')
 else:
   show_img = tf.keras.preprocessing.image.array_to_img(predictions[i, :, :, :])
   plt.title('Decoded Image')
   plt.imshow(show_img)
   plt.axis(False)

!pip install sewar

from sewar.full_ref import mse, rmse, psnr, uqi, ssim, ergas, scc, rase, sam, msssim, vifp

input_images = []
generated_images = []
for i in range(images.shape[0]):
  input_images.append(tf.keras.preprocessing.image.array_to_img(images[i, :, :,
:]).convert('L'))
```

```
  generated_images.append(tf.keras.preprocessing.image.array_to_img(predictions[i, :, :,
:]).convert('L'))

MSE = []
RMSE = []
PSNR = []
UQI = []
SCC = []
RASE =[]
SAM = []
VIF = []

for j in range(len(images)):
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
```

```
  VIF.append(vifp(gen, org))


MSE = np.array(MSE)

RMSE = np.array(RMSE)

PSNR = np.array(PSNR)

UQI = np.array(UQI)

SCC = np.array(SCC)

RASE = np.array(RASE)

SAM = np.array(SAM)

VIF = np.array(VIF)


print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")

print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")

print(f"PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ")

print(f"UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ")

print(f"SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ")

print(f"RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ")

print(f"SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ")

print(f"VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ")


def get_accuracy(g_true, preds):

  pred_idx = tf.argmax(preds, axis=1).numpy()
```

```
 count = 0
 for tab, pred in zip(g_true, pred_idx):
  if tab == pred:
   count += 1
 return count / preds.shape[0]

with ZipFile('covid_classifier_64.zip', 'r') as zipObj:
  # Extract all the contents of zip file in current directory
  zipObj.extractall()

classifier = tfk.models.load_model('covid_classifier_64')
classifier.trainable = False

preds = classifier(predictions)

print(f"classify accuracy: {get_accuracy(labels, preds)}")

from numpy import cov
from numpy import trace
from numpy import iscomplexobj
from numpy import asarray
from numpy.random import shuffle
```

```
from scipy.linalg import sqrtm
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.datasets.mnist import load_data


# scale an array of images to a new size
def scale_images(images, new_shape):
  images_list = list()
  for image in images:
    # resize with nearest neighbor interpolation
    new_image = tf.image.resize(image, new_shape)
    # store
    images_list.append(new_image)
  return asarray(images_list)


# calculate frechet inception distance
def calculate_fid(model, images1, images2):
  # calculate activations
  act1 = model.predict(images1)
  act2 = model.predict(images2)
  # calculate mean and covariance statistics
  mu1, sigma1 = act1.mean(axis=0), cov(act1, rowvar=False)
```

```
  mu2, sigma2 = act2.mean(axis=0), cov(act2, rowvar=False)
  # calculate sum squared difference between means
  ssdiff = np.sum((mu1 - mu2)**2.0)
  # calculate sqrt of product between cov
  covmean = sqrtm(sigma1.dot(sigma2))
  # check and correct imaginary numbers from sqrt
  if iscomplexobj(covmean):
    covmean = covmean.real
  # calculate score
  fid = ssdiff + trace(sigma1 + sigma2 - 2.0 * covmean)
  return fid

model = InceptionV3(include_top=False, pooling='avg', input_shape=(299,299,3))

FID = []

for images, labels in covid_ds:
  images1 = images
  z, mean, logvar = autoencoder.encode(images1)
  images2 = autoencoder.sample(z)
  images1 = preprocess_input(images1)
  images2 = preprocess_input(images2)
```

```
  fid = calculate_fid(classifier, images1, images2)
  FID.append(fid)

FID = np.array(FID)
print("CVA Covid Chest X-ray ------")
print(f'Adjusted FID Mean: {FID.mean()}, Std: {FID.std()}')
```

Appendix F: **Ad Cycle GAN for Malaria Parasitemic Blood Cell image Synthesis**

```
# -*- coding: utf-8 -*-

"""Original file is located at

  https://github.com/StanleyLiangYork/GAN_for_Medical_Image/blob/main/adaptive_cy

cle_gan_malaria.ipynb


Install the additional Tensorflow extension packages

"""


!pip install -q tensorflow-probability

!pip install -q tensorflow_addons

!pip install sewar


import os

import time

import matplotlib.pyplot as plt

from IPython.display import clear_output

from IPython import display

from zipfile import ZipFile

import glob

import imageio
```

```
import numpy as np

import PIL

import tensorflow as tf

import tensorflow_probability as tfp

import tensorflow_addons as tfa

import pathlib

import shutil

from sewar.full_ref import mse, rmse, psnr, uqi, ssim, ergas, scc, rase, sam, msssim, vifp


tfk = tf.keras

tfkl = tf.keras.layers

tfd = tfp.distributions

tfpl = tfp.layers


AUTOTUNE = tf.data.AUTOTUNE


"""Prepare the malaria blood cell dataset"""


if not os.path.exists('malaria.zip'):

  !wget https://storage.googleapis.com/pet-detect-239118/malaria.zip ./malaria.zip


with ZipFile('malaria.zip', 'r') as zipObj:
```

```
  # Extract all the contents of zip file in current directory
  zipObj.extractall()

data_dir = './malaria'
data_dir = pathlib.Path(data_dir)
blood_imgs = list(data_dir.glob('*/*.png'))

print(f'There are {len(blood_imgs)} in total.')

"""Separate the positive and negative images"""

positive_paths = []
negative_paths = []
for file in blood_imgs:
  file = str(file)
  parts = tf.strings.split(file, os.path.sep)
  if parts[-2] == 'Parasitemic':
    positive_paths.append('/content/'+file)
  else:
    negative_paths.append('/content/'+file)

"""take 8,000 images from each class"""
```

```
positive_paths = positive_paths[:8000]
negative_paths = negative_paths[:8000]

total = len(positive_paths)
test_idx = np.random.choice(total, 1000, replace=False)

positive_paths = np.array(positive_paths)
negative_paths = np.array(negative_paths)
test_positive = np.take(positive_paths, test_idx, axis=0)
train_positive = np.delete(positive_paths, test_idx, axis=0)
test_negative = np.take(negative_paths, test_idx, axis=0)
train_negative = np.delete(negative_paths, test_idx, axis=0)

train_images = np.concatenate((train_positive, train_negative), axis=0)
test_images = np.concatenate((test_positive, test_negative), axis=0)
print(train_images.shape)
print(test_images.shape)

"""copy the images to the correct folder"""

if not os.path.exists('train'):
```

```
  os.mkdir('train')

if not os.path.exists('test'):
  os.mkdir('test')

os.mkdir('train/Parasitemic')
os.mkdir('test/Parasitemic')
os.mkdir('train/Uninfected')
os.mkdir('test/Uninfected')
os.mkdir('gen_images')

for file in train_images:
  parts = str.split(file, os.path.sep)
  cp_path = parts[-2]+'/'+parts[-1]
  root = '/content/train'+'/'
  dest = root+cp_path
  src = file
  shutil.copy2(src, dest)

for file in test_images:
  parts = str.split(file, os.path.sep)
  cp_path = parts[-2]+'/'+parts[-1]
```

```
  root = '/content/test'+'/'
  dest = root+cp_path
  src = file
  shutil.copy2(src, dest)

"""Helper functions for making the dataset"""

def decode_img(img):
  # convert the compressed string to a 3D uint8 tensor
  img = tf.image.decode_png(img, channels=3)
  # Use `convert_image_dtype` to convert to floats in the [0,1] range.
  img = tf.image.convert_image_dtype(img, tf.float32)
  # resize the image to the desired size.
  return tf.image.resize(img, [32, 32])

def get_label(file_path):
  parts = tf.strings.split(file_path, os.path.sep)
  if parts[-2] == 'Parasitemic':
    return tf.constant(1.0, dtype="float64")
  else:
    return tf.constant(0.0, dtype="float64")
```

```
def process_path(file_path):
  label = get_label(file_path)
  # load the raw data from the file as a string
  img = tf.io.read_file(file_path)
  img = decode_img(img)
  # rescale from (0,255) to (0,1)
  # img = img / 255.0
  img = (img - 127.5) / 127.5
  return img, label

"""Load the pretrained classifier"""

!wget https://storage.googleapis.com/pet-detect-239118/classify_malaria_32.zip
./classify_malaria_32.zip

with ZipFile('classify_malaria_32.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()

classifier = tfk.models.load_model('classify_malaria_32')
classifier.trainable = False
```

```
"""Build the classify dataset for classification"""

BATCH_SIZE = 256
BUFFER_SIZE = 3000

classify_dataset = tf.data.Dataset.list_files("/content/train/*/*.png")
classify_dataset = classify_dataset.map(process_path, num_parallel_calls=AUTOTUNE)
classify_dataset = classify_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

test_dataset = tf.data.Dataset.list_files("/content/test/*/*.png")
test_dataset = test_dataset.map(process_path, num_parallel_calls=AUTOTUNE)
test_dataset = test_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

show_images, labels = next(iter(classify_dataset))

plt.figure(figsize=(12,12))

for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  image = tf.keras.preprocessing.image.array_to_img(show_images[i,:,:,:])
  plt.imshow(image)
```

```
  if labels[i] == 1.0:
    plt.title('Parasitemic')
  else:
    plt.title('Uninfected')

classifier.evaluate(classify_dataset)

classifier.evaluate(test_dataset)

"""Load the pretrained VAE model """

if not os.path.exists('CVA_encoder2_32.zip'):
  !wget https://storage.googleapis.com/pet-detect-239118/CVA_encoder2_32.zip
./CVA_encoder2_32.zip

with ZipFile('CVA_encoder2_32.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()

if not os.path.exists('CVA_decoder2_32.zip'):
  !wget https://storage.googleapis.com/pet-detect-239118/CVA_decoder2_32.zip
./CVA_decoder2_32.zip
```

```
with ZipFile('CVA_decoder2_32.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()


# VAE cost function as negative ELBO (Evidence Lower Bound Objective)


def normal_log_pdf(sample, mean, sd, raxis=1):
  log2pi = tf.math.log(2. * np.pi)
  logvar = np.log((np.square(sd)))
  return tf.reduce_sum(
      -.5 * ((sample - mean) ** 2. * tf.exp(-logvar) + logvar + log2pi),
      axis=raxis)


def vae_cost(x_true, model, analytic_kl=True, kl_weight=0.01):
  z_sample, mu, sd = model.encode(x_true)
  x_recons_logits = model.decoder(z_sample)
  # compute cross entropy loss for each dimension of every datapoint
  raw_cross_entropy = tf.nn.sigmoid_cross_entropy_with_logits(labels=x_true,
logits=x_recons_logits) # bs*64*64*3
  # compute cross entropy loss for all instances in mini-batch; shape=(batch_size,), the
first term of the objective
```

```
  neg_log_likelihood = tf.math.reduce_sum(raw_cross_entropy, axis=[1, 2, 3]) # the first
term of the objective

  # compute reverse KL divergence, either analytically or through MC approximation
with one sample, the second term of the objective
  if analytic_kl:
    kl_divergence = - 0.5 * tf.math.reduce_sum(1 + tf.math.log(tf.math.square(sd)) -
tf.math.square(mu) - tf.math.square(sd), axis=1)  # shape=(batch_size, )
  else:
    logpz = normal_log_pdf(z_sample, 0., 1.)  # shape=(batch_size,)
    logqz_x = normal_log_pdf(z_sample, mu, tf.math.square(sd))  # shape=(batch_size,)
    kl_divergence = logqz_x - logpz
  elbo = tf.math.reduce_mean(-kl_weight * kl_divergence - neg_log_likelihood)  #
shape=()
  return -elbo

class VAE(tfk.Model):

    def __init__(self, dim_z, learning_rate, seed=2000, name="autoencoder", **kwargs):
        super(VAE, self).__init__(name=name, **kwargs)
        self.dim_x = (32, 32, 3)
        self.dim_z = dim_z
```

```
    self.learning_rate = learning_rate
    self.seed = seed

@tf.function
def sample(self, eps=None):
  if eps is None:
    eps = tf.random.normal(shape=(100, self.dim_z))
  return self.decode(eps, apply_sigmoid=True)

def encode(self, x_input):
    mu, rho = tf.split(self.encoder(x_input), num_or_size_splits=2, axis=1)
    sd = tf.math.log(1 + tf.math.exp(rho))
    z_sample = mu + sd * tf.random.normal(shape=(self.dim_z,))
    return z_sample, mu, sd

def decode(self, z, apply_sigmoid=False):
    logits = self.decoder(z)
    if apply_sigmoid:
      probs = tf.sigmoid(logits)
      return probs
    return logits
```

```
def generate_and_save_images(model, epoch, test_sample):
  z, mean, logvar = model.encode(test_sample)
  predictions = model.sample(z)
  fig = plt.figure(figsize=(8, 8))

  for i in range(16):
    plt.subplot(4, 4, i + 1)
    show_img = tf.keras.preprocessing.image.array_to_img(predictions[i, :, :, :])
    plt.imshow(show_img)
    plt.axis('off')

  # tight_layout minimizes the overlap between 2 sub-plots
  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()

CVA_model = VAE(128, 2e-4)
CVA_model.encoder = tfk.models.load_model('CVA_encoder2_32')
CVA_model.decoder = tfk.models.load_model('CVA_decoder2_32')

# rescale the image tensor from [-1,1] to [0,1]
# @tf.function
def CVA_rescale(image_tensor):
```

```
  img = tf.multiply(image_tensor, 127.5)
  img = tf.add(img, 127.5)
  img = tf.divide(img, 255.0)
  return img

"""A dataset with parasitemic images only"""

BATCH_SIZE = 256
BUFFER_SIZE = 8000

positive_dataset = tf.data.Dataset.list_files("/content/train/Parasitemic/*.png")
positive_dataset = positive_dataset.map(process_path, num_parallel_calls=AUTOTUNE)
positive_dataset = positive_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

show_images, labels = next(iter(positive_dataset))

plt.figure(figsize=(12,12))

for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  image = tf.keras.preprocessing.image.array_to_img(show_images[i,:,:,:])
```

```
  plt.imshow(image)
  if labels[i] == 1.0:
    plt.title('Parasitemic')
  else:
    plt.title('Uninfected')

input_CVA = CVA_rescale(show_images) # rescale to [0,1]
z, mean, logvar = CVA_model.encode(input_CVA)
predictions = CVA_model.sample(z)
fig = plt.figure(figsize=(8, 8))

for i in range(16):
  plt.subplot(4, 4, i + 1)
  if i % 2 == 0:
    show_img = tf.keras.preprocessing.image.array_to_img(input_CVA[i, :, :, :])
    plt.title('Real Image')
    plt.imshow(show_img)
    plt.axis('off')
  else:
    show_img = tf.keras.preprocessing.image.array_to_img(predictions[i, :, :, :])
    plt.title('Decoded Image')
    plt.imshow(show_img)
```

```
    plt.axis(False)


input_images = []
generated_images = []
for i in range(input_CVA.shape[0]):
  input_images.append(tf.keras.preprocessing.image.array_to_img(input_CVA[i, :, :,
:]).convert('L'))
  generated_images.append(tf.keras.preprocessing.image.array_to_img(predictions[i, :, :,
:]).convert('L'))


MSE = []
RMSE = []
PSNR = []
UQI = []
SCC = []
RASE =[]
SAM = []
VIF = []


for j in range(len(input_images)):
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
```

```
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
  VIF.append(vifp(gen, org))

MSE = np.array(MSE)
RMSE = np.array(RMSE)
PSNR = np.array(PSNR)
UQI = np.array(UQI)
SCC = np.array(SCC)
RASE = np.array(RASE)
SAM = np.array(SAM)
VIF = np.array(VIF)

print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")
print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")
print(f"PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ")
print(f"UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ")
```

```
print(f'SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ")

print(f'RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ")

print(f'SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ")

print(f'VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ")


def get_accuracy(g_true, preds):
  pred_idx = tf.argmax(preds, axis=1).numpy()
  count = 0
  for tab, pred in zip(g_true, pred_idx):
    if tab == pred:
      count += 1
  return count / preds.shape[0]


class_loss = tfk.losses.SparseCategoricalCrossentropy(from_logits=True)


"""Classify the CAV generated images"""


preds = classifier(predictions)

print(f'classify accuracy: {get_accuracy(labels, preds)}")


"""Classify the real images"""
```

```
images, labels = next(iter(classify_dataset))
preds = classifier(images)
print(f"classify accuracy: {get_accuracy(labels, preds)}")

"""Set the two image domains"""

def process_gan_path(file_path):
  img = tf.io.read_file(file_path)
  img = decode_img(img)
  img = (img - 127.5) / 127.5
  return img

y_dataset = tf.data.Dataset.list_files("/content/train/Parasitemic/*.png")
x_dataset = tf.data.Dataset.list_files("/content/train/Uninfected/*.png")

BUFFER_SIZE = 8000
BATCH_SIZE = 256

x_dataset = x_dataset.map(process_gan_path, num_parallel_calls=AUTOTUNE)
x_dataset = x_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
y_dataset = y_dataset.map(process_gan_path, num_parallel_calls=AUTOTUNE)
y_dataset = y_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

```
y_testset = tf.data.Dataset.list_files("/content/test/Parasitemic/*.png")
x_testset = tf.data.Dataset.list_files("/content/test/Uninfected/*.png")

y_testset = y_testset.map(process_gan_path, num_parallel_calls=AUTOTUNE)
y_testset = y_testset.shuffle(2000).batch(256)
x_testset = x_testset.map(process_gan_path, num_parallel_calls=AUTOTUNE)
x_testset = x_testset.shuffle(2000).batch(256)

"""# Set the Cycle GAN model

Define the Generator and Discriminator
"""

# define the discriminator model
def define_discriminator(image_shape=(32,32,3)):
  init = tf.keras.initializers.TruncatedNormal(mean=0.0, stddev=0.02)
  in_image = tf.keras.Input(shape=image_shape)
  d = tf.keras.layers.Conv2D(64, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(in_image) # 16*16*64
  d = tf.keras.layers.LeakyReLU(alpha=0.2)(d)
```

```
  d = tf.keras.layers.Conv2D(128, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(d) # 8*8*128
  d = tfa.layers.InstanceNormalization(axis=-1, center=True, scale=True,
beta_initializer="random_uniform", gamma_initializer="random_uniform")(d)
  d = tf.keras.layers.LeakyReLU(alpha=0.2)(d)
  d = tf.keras.layers.Conv2D(256, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(d) # 4*4*256
  d = tfa.layers.InstanceNormalization(axis=-1, center=True, scale=True,
beta_initializer="random_uniform", gamma_initializer="random_uniform")(d)
  d = tf.keras.layers.LeakyReLU(alpha=0.2)(d)
  d = tf.keras.layers.Conv2D(512, (4,4), padding='same', kernel_initializer=init)(d) #
4*4*512
  d = tfa.layers.InstanceNormalization(axis=-1, center=True, scale=True,
beta_initializer="random_uniform", gamma_initializer="random_uniform")(d)
  d = tf.keras.layers.LeakyReLU(alpha=0.2)(d)
  patch_out = tf.keras.layers.Conv2D(1, (4,4), padding='same', kernel_initializer=init)(d)
# 4*4*1
  model = tf.keras.Model(inputs=in_image, outputs=patch_out)
  return model


def downsample(filters, size, apply_batchnorm=True):
  initializer = tf.random_normal_initializer(0., 0.02)
```

```
  result = tf.keras.Sequential()
  result.add(
      tfkl.Conv2D(filters, size, strides=2, padding='same',
                              kernel_initializer=initializer, use_bias=False))

  if apply_batchnorm:
    result.add(tfa.layers.InstanceNormalization(axis=-1, center=True, scale=True,
beta_initializer="random_uniform", gamma_initializer="random_uniform"))

  result.add(tf.keras.layers.LeakyReLU())

  return result

def upsample(filters, size, apply_dropout=False):
  initializer = tf.random_normal_initializer(0., 0.02)

  result = tf.keras.Sequential()
  result.add(
    tfkl.Conv2DTranspose(filters, size, strides=2,
                                  padding='same',
                                  kernel_initializer=initializer,
```

```
                                    use_bias=False))

  result.add(tfa.layers.InstanceNormalization(axis=-1, center=True, scale=True,
beta_initializer="random_uniform", gamma_initializer="random_uniform"))

  if apply_dropout:
      result.add(tf.keras.layers.Dropout(0.5))

  result.add(tf.keras.layers.ReLU())

  return result

def define_generator():
  inputs = tf.keras.layers.Input(shape=[32, 32, 3])

  down_stack = [
    downsample(64, 4),  # (bs, 16, 16, 64)
    downsample(128, 4),  # (bs, 8, 8, 128)
    downsample(256, 4),  # (bs, 4, 4, 256)
    downsample(512, 4),  # (bs, 2, 2, 512)
    downsample(512, 4),  # (bs, 1, 1, 512)
  ]
```

```
up_stack = [
  upsample(512, 4),  # (bs, 2, 2, 512)
  upsample(256, 4),  # (bs, 4, 4, 256)
  upsample(128, 4),  # (bs, 8, 8, 256)
  upsample(64, 4),  # (bs, 16, 16, 128)

]

initializer = tf.random_normal_initializer(0., 0.02)
last = tf.keras.layers.Conv2DTranspose(3, 4,
                                       strides=2,
                                       padding='same',
                                       kernel_initializer=initializer,
                                       activation='tanh')  # (bs, 64, 64, 3)

x = inputs

# Downsampling through the model
skips = []
for down in down_stack:
  x = down(x)
```

```
    skips.append(x)

  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    x = tf.keras.layers.Concatenate()([x, skip])

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)

image_shape = (32,32,3)

generator_g = define_generator()
generator_f = define_generator()
discriminator_x = define_discriminator(image_shape)
discriminator_y = define_discriminator(image_shape)

discriminator_x.summary()
```

```
tf.keras.utils.plot_model(generator_g, show_shapes=True, dpi=64)

# x -> y: normal -> infected - generator_g
# y -> x: infected -> normal - generator_f

x_images = next(iter(x_dataset))
y_images = next(iter(y_dataset))

to_para = generator_g(x_images)
to_normal = generator_f(y_images)
plt.figure(figsize=(6, 6))

imgs = [x_images, to_para, y_images, to_normal]
title = ['Uninfected', 'To_parasitemic', 'parasitemic', 'To_Uninfected']

plt.imshow(tf.keras.preprocessing.image.array_to_img(imgs[0][0]))
plt.suptitle("Mapping of generators before training", fontsize=14)
for i in range(len(imgs)):
  plt.subplot(2, 2, i+1)
  plt.title(title[i])
  plt.axis(False)
  if i % 2 == 0:
```

```
    plt.imshow(tf.keras.preprocessing.image.array_to_img(imgs[i][0]))
  else:
    plt.imshow(tf.keras.preprocessing.image.array_to_img(imgs[i][0]))
plt.show()

# the hotmap of untrained discriminator
plt.figure(figsize=(8,8))
plt.subplot(121)
plt.title('discriminate parasitemic')
plt.axis(False)
plt.imshow(discriminator_y(y_images)[0, ..., -1], cmap='RdBu_r')

plt.subplot(122)
plt.title('discriminate uninfected')
plt.axis(False)
plt.imshow(discriminator_x(x_images)[0, ..., -1], cmap='RdBu_r')
plt.show()

"""Setup the folder for generated images

Define the loss functions for the GAN components
"""
```

```
LAMBDA = 80
# alternative: MSE
loss_obj = tf.keras.losses.BinaryCrossentropy(from_logits=True)
# loss_obj = tfk.losses.MeanSquaredError()

def discriminator_loss(real, generated):
  real_loss = loss_obj(tf.ones_like(real), real)
  generated_loss = loss_obj(tf.zeros_like(generated), generated)
  total_disc_loss = real_loss + generated_loss
  return total_disc_loss * 0.5

def generator_loss(generated):
  return loss_obj(tf.ones_like(generated), generated)

def calc_cycle_loss(real_image, cycled_image):
  loss1 = tf.reduce_mean(tf.abs(real_image - cycled_image))
  return LAMBDA * loss1

# prevser color
def identity_loss(real_image, same_image):
  loss = tf.reduce_mean(tf.abs(real_image - same_image))
```

```
  return LAMBDA * loss * 0.8

generator_g_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
generator_f_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)

discriminator_x_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_y_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)

checkpoint_path = "./checkpoints/train"

ckpt = tf.train.Checkpoint(generator_g=generator_g,
                           generator_f=generator_f,
                           discriminator_x=discriminator_x,
                           discriminator_y=discriminator_y,
                           generator_g_optimizer=generator_g_optimizer,
                           generator_f_optimizer=generator_f_optimizer,
                           discriminator_x_optimizer=discriminator_x_optimizer,
                           discriminator_y_optimizer=discriminator_y_optimizer)

ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=3)

# if a checkpoint exists, restore the latest checkpoint.
```

```
if ckpt_manager.latest_checkpoint:
  ckpt.restore(ckpt_manager.latest_checkpoint)
  print ('Latest checkpoint restored!!')

EPOCHS = 600

def generate_images(model, test_input, epoch):
  prediction = model(test_input)
  idx = int(np.random.choice(16, 1, replace=False))
  plt.figure(figsize=(10, 10))


  display_list = [test_input[idx], prediction[idx]]
  title = ['Input Image', 'Predicted Image']

  for i in range(2):
    plt.subplot(1, 2, i+1)
    plt.title(title[i])
    # getting the pixel values between [0, 1] to plot it.
    plt.imshow(tf.keras.preprocessing.image.array_to_img(display_list[i]))
    plt.axis('off')
  plt.savefig('./gen_images/image_at_epoch_{:04d}.png'.format(epoch))
```

```
  plt.show()


def classify_para_loss(y, batch_size):
  true_para = tf.ones([batch_size,1])
  pred = classifier(y)
  return class_loss(true_para, pred) * 0.5


def classify_normal_loss(x, batch_size):
  true_normal = tf.zeros([batch_size,1])
  pred = classifier(x)
  return class_loss(true_normal, pred) * 0.5


# @tf.function
def train_step(real_x, real_y, epoch, c_flag=True):
  # persistent is set to True because the tape is used more than
  # once to calculate the gradients.
  batch_size = real_x.shape[0]
  with tf.GradientTape(persistent=True) as tape:
    # Generator G translates X -> Y
    # Generator F translates Y -> X.


    fake_y = generator_g(real_x, training=True)
```

```
    cycled_x = generator_f(fake_y, training=True)

    fake_x = generator_f(real_y, training=True)
    cycled_y = generator_g(fake_x, training=True)

    # same_x and same_y are used for identity loss.
    same_x = generator_f(real_x, training=True)
    same_y = generator_g(real_y, training=True)

    disc_real_x = discriminator_x(real_x, training=True)
    disc_real_y = discriminator_y(real_y, training=True)

    disc_fake_x = discriminator_x(fake_x, training=True)
    disc_fake_y = discriminator_y(fake_y, training=True)

    # calculate the loss

    gen_g_loss = generator_loss(disc_fake_y)
    gen_f_loss = generator_loss(disc_fake_x)

    c_loss_x = classify_normal_loss(same_x, batch_size) +
classify_normal_loss(cycled_x, batch_size)
```

```
    c_loss_x = c_loss_x * 0.1
    c_loss_y = classify_para_loss(same_y, batch_size) + classify_para_loss(cycled_y,
batch_size)
    c_loss_y = c_loss_y * 0.1
    total_cycle_loss = calc_cycle_loss(real_x, cycled_x) + calc_cycle_loss(real_y,
cycled_y)

    # if epoch > 50 and epoch < 100:
    #   c_loss_x = c_loss_x * 1.0
    #   c_loss_y = c_loss_y * 1.0
    #   total_cycle_loss = total_cycle_loss * 1.0

    # if epoch >= 100 and epoch < 200:
    #   c_loss_x = c_loss_x * 0.4
    #   c_loss_y = c_loss_y * 0.4
    #   total_cycle_loss = total_cycle_loss * 1.5

    # if epoch >= 200 and epoch <= 400:
    #   c_loss_x = c_loss_x * 0.2
    #   c_loss_y = c_loss_y * 0.2
    #   total_cycle_loss = total_cycle_loss * 2.0
```

```
    # Total generator loss = adversarial loss + cycle loss
    if (c_flag):
      total_gen_g_loss = gen_g_loss + total_cycle_loss + identity_loss(real_y, same_y) +
c_loss_y
      total_gen_f_loss = gen_f_loss + total_cycle_loss + identity_loss(real_x, same_x) +
c_loss_x
    else:
      total_gen_g_loss = gen_g_loss + total_cycle_loss + identity_loss(real_y, same_y)
      total_gen_f_loss = gen_f_loss + total_cycle_loss + identity_loss(real_x, same_x)

    disc_x_loss = discriminator_loss(disc_real_x, disc_fake_x)
    disc_y_loss = discriminator_loss(disc_real_y, disc_fake_y)


  # Calculate the gradients for generator and discriminator
  generator_g_gradients = tape.gradient(total_gen_g_loss,
                                        generator_g.trainable_variables)
  generator_f_gradients = tape.gradient(total_gen_f_loss,
                                        generator_f.trainable_variables)

  discriminator_x_gradients = tape.gradient(disc_x_loss,
```

```
                                        discriminator_x.trainable_variables)
    discriminator_y_gradients = tape.gradient(disc_y_loss,
                                        discriminator_y.trainable_variables)

    # Apply the gradients to the optimizer
    generator_g_optimizer.apply_gradients(zip(generator_g_gradients,
                                        generator_g.trainable_variables))

    generator_f_optimizer.apply_gradients(zip(generator_f_gradients,
                                        generator_f.trainable_variables))

    discriminator_x_optimizer.apply_gradients(zip(discriminator_x_gradients,
                                          discriminator_x.trainable_variables))

    discriminator_y_optimizer.apply_gradients(zip(discriminator_y_gradients,
                                          discriminator_y.trainable_variables))

    return c_loss_x, c_loss_y, total_cycle_loss, total_gen_g_loss, total_gen_f_loss,
disc_x_loss, disc_y_loss

import time
from IPython.display import clear_output
```

```
from IPython import display

history = {}
history['class_loss_x'] = []
history['class_loss_y'] = []
history['cycle_loss'] = []
history['total_gen_g_loss'] = []
history['total_gen_f_loss'] = []
history['disc_loss_x'] = []
history['disc_loss_y'] = []
history['used_time'] = []

c_loss_x_mean = tfk.metrics.Mean()
c_loss_y_mean = tfk.metrics.Mean()
cycle_loss_mean = tfk.metrics.Mean()
total_gen_g_loss_mean = tfk.metrics.Mean()
total_gen_f_loss_mean = tfk.metrics.Mean()
disc_loss_x_mean = tfk.metrics.Mean()
disc_loss_y_mean = tfk.metrics.Mean()
uesed_time_mean = tfk.metrics.Mean()

for epoch in range(600):
```

```
  start = time.time()
  c_loss_x_mean.reset_state()
  c_loss_y_mean.reset_state()
  cycle_loss_mean.reset_state()
  total_gen_g_loss_mean.reset_state()
  total_gen_f_loss_mean.reset_state()
  disc_loss_x_mean.reset_state()
  disc_loss_y_mean.reset_state()
  uesed_time_mean.reset_state()

  n = 0
  for image_x, image_y in tf.data.Dataset.zip((x_dataset, y_dataset)):
    if (n % 10 == 0):
      c_loss_x, c_loss_y, total_cycle_loss, total_gen_g_loss, total_gen_f_loss, disc_x_loss,
disc_y_loss = train_step(image_x, image_y, epoch, c_flag=True) # chang c_flag to False
if want to remove the criterion
    else:
      c_loss_x, c_loss_y, total_cycle_loss, total_gen_g_loss, total_gen_f_loss, disc_x_loss,
disc_y_loss = train_step(image_x, image_y, epoch=epoch, c_flag=False)
    c_loss_x_mean.update_state(c_loss_x)
    c_loss_y_mean.update_state(c_loss_y)
    cycle_loss_mean.update_state(total_cycle_loss)
```

```
  total_gen_g_loss_mean.update_state(total_gen_g_loss)
  total_gen_f_loss_mean.update_state(total_gen_f_loss)
  disc_loss_x_mean.update_state(disc_x_loss)
  disc_loss_y_mean.update_state(disc_y_loss)
  uesed_time_mean.update_state(time.time()-start)

  if n % 10 == 0:
    print ('.', end='')
  n += 1

clear_output(wait=True)
# Using a consistent image (sample_horse) so that the progress of the model
# is clearly visible.
generate_images(generator_g, x_images, epoch)

if (epoch + 1) % 50 == 0:
  ckpt_save_path = ckpt_manager.save()
  print ('Saving checkpoint for epoch {} at {}'.format(epoch+1,
                                                       ckpt_save_path))
history['class_loss_x'].append(c_loss_x_mean.result().numpy())
history['class_loss_y'].append(c_loss_y_mean.result().numpy())
history['cycle_loss'].append(cycle_loss_mean.result().numpy())
```

```
  history['total_gen_g_loss'].append(total_gen_g_loss_mean.result().numpy())
  history['total_gen_f_loss'].append(total_gen_f_loss_mean.result().numpy())
  history['disc_loss_x'].append(disc_loss_x_mean.result().numpy())
  history['disc_loss_y'].append(disc_loss_y_mean.result().numpy())
  history['used_time'].append(uesed_time_mean.result().numpy())

  print ('Time taken for epoch {} is {} sec\n'.format(epoch + 1,
                                                      time.time()-start))

generator_g.save('malaria_CycleGAN')
!zip -r malaria_CycleGAN.zip malaria_CycleGAN

generator_f.save('normalcell_CycleGAN')
!zip -r normalcell_CycleGAN.zip normalcell_CycleGAN

!zip -r gen_images_cycleGAN.zip gen_images

plt.figure(figsize=(15, 8))

plt.subplot(2, 2, 1)
plt.plot(history['total_gen_g_loss'], label='Total Generator G loss')
plt.plot(history['total_gen_f_loss'], label='Total Generator F loss')
```

```
plt.legend(loc='best')
plt.title('Generator Loss')
plt.xlabel('epoch')

plt.subplot(2, 2, 2)

plt.plot(history['disc_loss_y'], label='Total Discriminator Y loss')
plt.plot(history['disc_loss_x'], label='Total Discriminator X loss')
plt.legend(loc='best')
plt.title('Discriminator Loss')
plt.xlabel('epoch')

plt.subplot(2, 2, 3)

plt.plot(history['class_loss_x'], label='Criterion X loss')
plt.legend(loc='best')
plt.title('Criterion X loss')
plt.xlabel('epoch')

plt.subplot(2, 2, 4)
plt.tight_layout()
plt.plot(history['class_loss_y'], label='Criterion Y loss')
```

```
plt.legend(loc='best')
plt.title('Criterion Y loss')
plt.xlabel('epoch')

plt.plot(history['cycle_loss'], label='Total cycle loss')
plt.legend(loc='best')
plt.title('Total cycle loss')
plt.xlabel('epoch')

plt.plot(history['used_time'], label='Runtime per epoch')
plt.legend(loc='best')
plt.title('Optimization runtime per epoch')
plt.xlabel('epoch')
plt.ylabel('second')

"""X --> Y by Gen G"""

n_images = next(iter(x_dataset))
gen_images = generator_g(n_images)

plt.figure(figsize=(12,12))
```

```
for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  if i % 2 == 0:
    image = tf.keras.preprocessing.image.array_to_img(n_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Input')
  else:
    image = tf.keras.preprocessing.image.array_to_img(gen_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Generated')


input_images = []
generated_images = []
for i in range(n_images.shape[0]):
  input_images.append(tf.keras.preprocessing.image.array_to_img(n_images[i, :, :,
:]).convert('L'))
  generated_images.append(tf.keras.preprocessing.image.array_to_img(gen_images[i, :, :,
:]).convert('L'))


MSE = []
RMSE = []
```

```
PSNR = []
UQI = []
SCC = []
RASE =[]
SAM = []
VIF = []

for j in range(len(input_images)):
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
  VIF.append(vifp(gen, org))

MSE = np.array(MSE)
RMSE = np.array(RMSE)
PSNR = np.array(PSNR)
```

```
UQI = np.array(UQI)

SCC = np.array(SCC)

RASE = np.array(RASE)

SAM = np.array(SAM)

VIF = np.array(VIF)


print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")

print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")

print(f"PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ")

print(f"UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ")

print(f"SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ")

print(f"RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ")

print(f"SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ")

print(f"VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ")


preds = classifier(gen_images)

true_labels = tf.ones([256,1])

print(f"classify accuracy: {get_accuracy(true_labels, preds)}")


"""Y --> Y by Gen G"""


n_images = next(iter(y_dataset))
```

```
gen_images = generator_g(n_images)


plt.figure(figsize=(12,12))


for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  if i % 2 == 0:
    image = tf.keras.preprocessing.image.array_to_img(n_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Input')
  else:
    image = tf.keras.preprocessing.image.array_to_img(gen_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Generated')


input_images = []
generated_images = []
for i in range(n_images.shape[0]):
  input_images.append(tf.keras.preprocessing.image.array_to_img(n_images[i, :, :,
:]).convert('L'))
```

```
  generated_images.append(tf.keras.preprocessing.image.array_to_img(gen_images[i, :, :,
:]).convert('L'))

MSE = []
RMSE = []
PSNR = []
UQI = []
SCC = []
RASE =[]
SAM = []
VIF = []

for j in range(len(input_images)):
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
```

```
  VIF.append(vifp(gen, org))


MSE = np.array(MSE)

RMSE = np.array(RMSE)

PSNR = np.array(PSNR)

UQI = np.array(UQI)

SCC = np.array(SCC)

RASE = np.array(RASE)

SAM = np.array(SAM)

VIF = np.array(VIF)


print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")

print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")

print(f"PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ")

print(f"UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ")

print(f"SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ")

print(f"RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ")

print(f"SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ")

print(f"VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ")


preds = classifier(gen_images)

true_labels = tf.ones([256,1])
```

```
print(f"classify accuracy: {get_accuracy(true_labels, preds)}")


"""Y --> X by Gen F"""


n_images = next(iter(y_dataset))
gen_images = generator_f(n_images)


plt.figure(figsize=(12,12))


for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  if i % 2 == 0:
    image = tf.keras.preprocessing.image.array_to_img(n_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Input')
  else:
    image = tf.keras.preprocessing.image.array_to_img(gen_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Generated')


input_images = []
```

```
generated_images = []
for i in range(n_images.shape[0]):
  input_images.append(tf.keras.preprocessing.image.array_to_img(n_images[i, :, :,
:]).convert('L'))
  generated_images.append(tf.keras.preprocessing.image.array_to_img(gen_images[i, :, :,
:]).convert('L'))

MSE = []
RMSE = []
PSNR = []
UQI = []
SCC = []
RASE =[]
SAM = []
VIF = []

for j in range(len(input_images)):
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
```

```
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
  VIF.append(vifp(gen, org))

MSE = np.array(MSE)
RMSE = np.array(RMSE)
PSNR = np.array(PSNR)
UQI = np.array(UQI)
SCC = np.array(SCC)
RASE = np.array(RASE)
SAM = np.array(SAM)
VIF = np.array(VIF)

print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")
print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")
print(f"PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ")
print(f"UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ")
print(f"SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ")
print(f"RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ")
print(f"SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ")
```

```
print(f'VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ')

preds = classifier(gen_images)
true_labels = tf.zeros([256,1])
print(f"classify accuracy: {get_accuracy(true_labels, preds)}")

"""X --> X by Gen F"""

n_images = next(iter(x_dataset))
gen_images = generator_f(n_images)

plt.figure(figsize=(12,12))

for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  if i % 2 == 0:
    image = tf.keras.preprocessing.image.array_to_img(n_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Input')
  else:
    image = tf.keras.preprocessing.image.array_to_img(gen_images[i,:,:,:])
```

```
    plt.imshow(image)
    plt.title('Generated')

input_images = []
generated_images = []
for i in range(n_images.shape[0]):
  input_images.append(tf.keras.preprocessing.image.array_to_img(n_images[i, :, :,
:]).convert('L'))
  generated_images.append(tf.keras.preprocessing.image.array_to_img(gen_images[i, :, :,
:]).convert('L'))

MSE = []
RMSE = []
PSNR = []
UQI = []
SCC = []
RASE =[]
SAM = []
VIF = []

for j in range(len(input_images)):
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
```

```
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
  VIF.append(vifp(gen, org))

MSE = np.array(MSE)
RMSE = np.array(RMSE)
PSNR = np.array(PSNR)
UQI = np.array(UQI)
SCC = np.array(SCC)
RASE = np.array(RASE)
SAM = np.array(SAM)
VIF = np.array(VIF)

print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")
print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")
print(f"PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ")
```

```
print(f'UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ')

print(f'SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ')

print(f'RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ')

print(f'SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ')

print(f'VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ')


preds = classifier(gen_images)

true_labels = tf.zeros([256,1])

print(f"classify accuracy: {get_accuracy(true_labels, preds)}")


"""FID score"""


from numpy import cov

from numpy import trace

from numpy import iscomplexobj

from numpy import asarray

from numpy.random import shuffle

from scipy.linalg import sqrtm

from tensorflow.keras.applications.inception_v3 import InceptionV3

from tensorflow.keras.applications.inception_v3 import preprocess_input

from tensorflow.keras.datasets.mnist import load_data

from tensorflow.keras.datasets import cifar10
```

```
# scale an array of images to a new size
def scale_images(images, new_shape):
  images_list = list()
  for image in images:
    # resize with nearest neighbor interpolation
    new_image = tf.image.resize(image, new_shape)
    # store
    images_list.append(new_image)
  return asarray(images_list)

# calculate frechet inception distance
def calculate_fid(model, images1, images2):
  # calculate activations
  act1 = model.predict(images1)
  act2 = model.predict(images2)
  # calculate mean and covariance statistics
  mu1, sigma1 = act1.mean(axis=0), cov(act1, rowvar=False)
  mu2, sigma2 = act2.mean(axis=0), cov(act2, rowvar=False)
  # calculate sum squared difference between means
  ssdiff = np.sum((mu1 - mu2)**2.0)
  # calculate sqrt of product between cov
```

```
  covmean = sqrtm(sigma1.dot(sigma2))
  # check and correct imaginary numbers from sqrt
  if iscomplexobj(covmean):
    covmean = covmean.real
  # calculate score
  fid = ssdiff + trace(sigma1 + sigma2 - 2.0 * covmean)
  return fid

model = InceptionV3(include_top=False, pooling='avg', input_shape=(299,299,3))

FID = []
for images in x_testset:
  images1 = images
  images2 = generator_g(images)
  images1 = preprocess_input(images1)
  images2 = preprocess_input(images2)
  fid = calculate_fid(classifier, images1, images2)
  FID.append(fid)

FID = np.array(FID)
print("Cycle GAN X to Y by Gen G ------")
print(f'Adjusted FID Mean: {FID.mean()}, Std: {FID.std()}')
```

```
FID = []
for images in y_testset:
  images1 = images
  images2 = generator_g(images)
  images1 = preprocess_input(images1)
  images2 = preprocess_input(images2)
  fid = calculate_fid(classifier, images1, images2)
  FID.append(fid)

FID = np.array(FID)
print("Cycle GAN Y to Y by Gen G ------")
print(f'Adjusted FID Mean: {FID.mean()}, Std: {FID.std()}')

FID = []
for images in y_testset:
  images1 = images
  images2 = generator_f(images)
  images1 = preprocess_input(images1)
  images2 = preprocess_input(images2)
  fid = calculate_fid(classifier, images1, images2)
  FID.append(fid)
```

```
FID = np.array(FID)
print("Cycle GAN Y to X by Gen F ------")
print(f'Adjusted FID Mean: {FID.mean()}, Std: {FID.std()}')

FID = []
for images in x_testset:
  images1 = images
  images2 = generator_f(images)
  images1 = preprocess_input(images1)
  images2 = preprocess_input(images2)
  fid = calculate_fid(classifier, images1, images2)
  FID.append(fid)

FID = np.array(FID)
print("Cycle GAN X to X by Gen F ------")
print(f'Adjusted FID Mean: {FID.mean()}, Std: {FID.std()}')
```

Appendix G: **Ad Cycle GAN for COVID-19 Chest X-Ray Synthesis**

```
# -*- coding: utf-8 -*-
"""
Original file is located at
    https://github.com/StanleyLiangYork/GAN_for_Medical_Image/blob/main/Cycle_GAN_with_Criterion_COVID_19.ipynb
"""
!pip install tensorflow_addons
!pip install sewar


from sewar.full_ref import mse, rmse, psnr, uqi, ssim, ergas, scc, rase, sam, msssim, vifp
import os
import shutil
import random
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as tf
import zipfile
import random
from PIL import Image
from matplotlib import pyplot as plt
```

```
import re
import tensorflow.keras as tfk
import tensorflow.keras.layers as tfkl
import tensorflow_hub as hub
import cv2
import tensorflow_addons as tfa
import tensorflow_probability as tfp

print(f'Tensorflow Version: {tf.__version__}')

"""Set the random seed for replication"""

tf.random.set_seed(100)
AUTOTUNE = tf.data.AUTOTUNE

"""Fetch the COVID-19 X-Ray dataset"""

if not os.path.exists('covid_set.zip'):
  !wget https://storage.googleapis.com/pet-detect-239118/covid_set.zip ./covid_set.zip

with zipfile.ZipFile('covid_set.zip') as ZipObj:
  ZipObj.extractall()
```

```
"""Set the folders for the experiment"""

root_dir = '/content/covid_set'
paths = os.listdir(root_dir)

covid = re.compile("COVID")
normal = re.compile("NORMAL")
viral = re.compile("Viral")

covid_path = []
normal_path = []
viral_path = []

for path in paths:
  if covid.match(path):
    covid_path.append(path)
  if normal.match(path):
    normal_path.append(path)
  if viral.match(path):
    viral_path.append(path)
```

```
val_covid_path = covid_path[:50]
covid_path = covid_path[50:]
print(len(val_covid_path))
print(len(covid_path))

"""Since we have just a few COVID-19 X-ray images, we separate 50 images for
validation, and the rest 169 for training"""

for _ in range(5):
  random_items = random.sample(covid_path, 169)
  covid_path += random_items

print(len(covid_path))

"""Randomly resample the images

Build a balanced dataset, each class has 1014 images respectively
"""

for i, path in enumerate(covid_path):
  covid_path[i] = root_dir + '/' + path
```

```
for i, path in enumerate(normal_path):
  normal_path[i] = root_dir + '/' + path

for i, path in enumerate(viral_path):
  viral_path[i] = root_dir + '/' + path

# 1014 + 50 = 1064 -- need 50 extra images from normal and from viral classes for the
validation dataset
covid_path = covid_path
normal_path = normal_path[:1064]
viral_path = viral_path[:1064]

print(len(covid_path))
print(len(normal_path))
print(len(viral_path))

"""The helper function the resize and rescale the images.<p>
labels: COVID-0, NORMAL-1, VIRAL-2
"""

def decode_img(img):
  # convert the compressed string to a 3D uint8 tensor
```

```
  img = tf.image.decode_png(img, channels=3)
  # Use `convert_image_dtype` to convert to floats in the [0,1] range.
  img = tf.image.convert_image_dtype(img, tf.float32)
  # resize the image to the desired size.
  return tf.image.resize(img, [64, 64])


def get_label(file_path):
  if tf.strings.regex_full_match(file_path, ".*COVID.*"):
    return tf.constant(0.0, dtype="float32")
  elif tf.strings.regex_full_match(file_path, ".*NORMAL.*"):
    return tf.constant(1.0, dtype="float32")
  else:
    return tf.constant(2.0, dtype="float32")


def process_path(file_path):
  label = get_label(file_path)
  # load the raw data from the file as a string
  img = tf.io.read_file(file_path)
  img = decode_img(img)


  # rescale from (0,255) to (0,1)
  # img = img / 255.0
```

```
  img = (img - 127.5) / 127.5
  return img, label

"""The three image datasets for each image class """

covid_ds = tf.data.Dataset.list_files(covid_path, shuffle=True)
normal_ds = tf.data.Dataset.list_files(normal_path[:1014], shuffle=True)
viral_ds = tf.data.Dataset.list_files(viral_path[:1014], shuffle=True)

BATCH_SIZE = 64
BUFFER_SIZE = 1014
AUTOTUNE = tf.data.AUTOTUNE

covid_ds = covid_ds.map(process_path,
num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
normal_ds = normal_ds.map(process_path,
num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
viral_ds = viral_ds.map(process_path,
num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

train_path = covid_path + normal_path[:1014] + viral_path[:1014]
val_path = val_covid_path + normal_path[1014:] + viral_path[1014:]
```

```
print(len(train_path))
print(len(val_path))

train_ds = tf.data.Dataset.list_files(train_path, shuffle=True)
train_ds = train_ds.map(process_path,
num_parallel_calls=AUTOTUNE).shuffle(3042).batch(BATCH_SIZE)
val_ds = tf.data.Dataset.list_files(val_path, shuffle=True)
val_ds = val_ds.map(process_path,
num_parallel_calls=AUTOTUNE).shuffle(150).batch(BATCH_SIZE)

images, labels = next(iter(train_ds))

"""visualize the images"""

plt.figure(figsize=(12,12))

for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  image = tf.keras.preprocessing.image.array_to_img(images[i,:,:,:])
  plt.imshow(image)
  if labels[i] == 0.0:
```

```
    plt.title('COVID')
  if labels[i] == 1.0:
    plt.title("Normal")
  if labels[i] == 2.0:
    plt.title("Viral")

"""Train the classifier later as the criterion for the GAN"""

# function for creating an identity or projection residual module
def residual_module(layer_in, n_filters):
  merge_input = layer_in
  # check if the number of filters needs to be increase, assumes channels last format
  if layer_in.shape[-1] != n_filters:
    merge_input = tfkl.Conv2D(n_filters, (1,1), padding='same', activation='relu',
kernel_initializer='he_normal')(layer_in)
  # conv1
  conv1 = tfkl.Conv2D(n_filters, (3,3), padding='same', activation='relu',
kernel_initializer='he_normal')(layer_in)
  # conv2
  conv2 = tfkl.Conv2D(n_filters, (3,3), padding='same', activation='linear',
kernel_initializer='he_normal')(conv1)
  # add filters, assumes filters/channels last
```

```
  layer_out = tfk.layers.Add()([conv2, merge_input])
  # activation function
  layer_out = tfkl.Activation('relu')(layer_out)
  return layer_out

def define_classifier(input_dim=(64,64,3)):
  input_layer = tfk.Input(shape=input_dim)
  layer = tfkl.Lambda(lambda x: x*127.5+127.5)(input_layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
  layer = tfkl.MaxPooling2D()(layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
  layer = tfkl.MaxPooling2D()(layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
  layer = tfkl.MaxPooling2D()(layer)
  layer = residual_module(layer, 64)
  layer = tfkl.BatchNormalization()(layer)
  layer = tfkl.MaxPooling2D()(layer)
  layer = tfkl.Flatten()(layer)
  layer = tfkl.Dense(128, activation='tanh')(layer)
```

```
  layer = tfkl.Dropout(0.4)(layer)
  layer = tfkl.Dense(3)(layer)
  model = tfk.models.Model(inputs=input_layer, outputs=layer)
  return model

classifier = define_classifier()
classifier.summary()

"""Compile the classifer DNN with sparce categorical crossentropy as the loss function,
the input label with shape (batch, 1), the DNN output with shape (batch, 3)"""

classifier.compile(
    optimizer=tfk.optimizers.Adam(learning_rate=1e-4),
    loss=tfk.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'])
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

"""Optimize with 100 epochs"""

history = classifier.fit(train_ds, validation_data=val_ds, epochs=100,
callbacks=[callback], verbose=2)
```

```
# visualize the training procedure
plt.figure(figsize=(12, 8))
plt.subplot(2, 1, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.ylim([0, 2.5])
plt.legend(loc='best')
plt.title('Training and Validation Loss')

plt.subplot(2, 1, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.ylim([0, 1.2])
plt.plot()
plt.legend(loc='best')
plt.title('Training and Validation Accuracy')
plt.xlabel('epoch')
plt.savefig('train.png')

# save and zip the classifier
classifier.save('covid_classifier_64')
!zip -r covid_classifier_64.zip covid_classifier_64
```

```
"""Now we setup the Cycle-GAN with criterion"""

from zipfile import ZipFile

with ZipFile('covid_classifier_64.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()

# delete and reload the pretrained classifier
# del classifier
classifier = tfk.models.load_model('covid_classifier_64')
classifier.trainable = False

classifier.evaluate(train_ds)

# check the pretrained classifier with the validation dataset
classifier.evaluate(val_ds)

# set up the criterion loss object
class_loss = tfk.losses.SparseCategoricalCrossentropy(from_logits=True)
```

```
def get_accuracy(g_true, preds):
  pred_idx = tf.argmax(preds, axis=1).numpy()
  count = 0
  for tab, pred in zip(g_true, pred_idx):
    if tab == pred:
      count += 1
  return count / preds.shape[0]

preds = classifier(images)

result = get_accuracy(labels, preds)
print(result)

loss = class_loss(labels, preds)
print(loss)

# define the discriminator model
def define_discriminator(image_shape=(64,64,3)):
  init = tf.keras.initializers.TruncatedNormal(mean=0.0, stddev=0.02)
  in_image = tf.keras.Input(shape=image_shape)
  d = tf.keras.layers.Conv2D(64, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(in_image) # 32*32*64
```

```
  d = tf.keras.layers.LeakyReLU(alpha=0.2)(d)
  d = tf.keras.layers.Conv2D(128, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(d) # 16*16*128
  d = tfa.layers.InstanceNormalization(axis=-1, center=True, scale=True,
beta_initializer="random_uniform", gamma_initializer="random_uniform")(d)
  d = tf.keras.layers.LeakyReLU(alpha=0.2)(d)
  d = tf.keras.layers.Conv2D(256, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(d) # 8*8*256
  d = tfa.layers.InstanceNormalization(axis=-1, center=True, scale=True,
beta_initializer="random_uniform", gamma_initializer="random_uniform")(d)
  d = tf.keras.layers.LeakyReLU(alpha=0.2)(d)
  d = tf.keras.layers.Conv2D(512, (4,4), strides=(2,2), padding='same',
kernel_initializer=init)(d) # 4*4*512
  d = tfa.layers.InstanceNormalization(axis=-1, center=True, scale=True,
beta_initializer="random_uniform", gamma_initializer="random_uniform")(d)
  d = tf.keras.layers.LeakyReLU(alpha=0.2)(d)
  patch_out = tf.keras.layers.Conv2D(1, (4,4), padding='same', kernel_initializer=init)(d)
# 4*4*1
  model = tf.keras.Model(inputs=in_image, outputs=patch_out)
  return model

def downsample(filters, size, apply_batchnorm=True):
```

```
  initializer = tf.random_normal_initializer(0., 0.02)

  result = tf.keras.Sequential()
  result.add(
      tfkl.Conv2D(filters, size, strides=2, padding='same',
                              kernel_initializer=initializer, use_bias=False))

  if apply_batchnorm:
    result.add(tfa.layers.InstanceNormalization(axis=-1, center=True, scale=True,
beta_initializer="random_uniform", gamma_initializer="random_uniform"))

  result.add(tf.keras.layers.LeakyReLU())

  return result

def upsample(filters, size, apply_dropout=False):
  initializer = tf.random_normal_initializer(0., 0.02)

  result = tf.keras.Sequential()
  result.add(
    tfkl.Conv2DTranspose(filters, size, strides=2,
                                  padding='same',
```

```
                                kernel_initializer=initializer,
                                use_bias=False))

  result.add(tfa.layers.InstanceNormalization(axis=-1, center=True, scale=True,
beta_initializer="random_uniform", gamma_initializer="random_uniform"))

  if apply_dropout:
      result.add(tf.keras.layers.Dropout(0.5))

  result.add(tf.keras.layers.ReLU())

  return result

def define_generator():
  inputs = tf.keras.layers.Input(shape=[64, 64, 3])

  down_stack = [
    downsample(64, 4),  # (bs, 32, 32, 64)
    downsample(128, 4),  # (bs, 16, 16, 128)
    downsample(256, 4),  # (bs, 8, 8, 256)
    downsample(512, 4),  # (bs, 4, 4, 512)
    downsample(512, 4),  # (bs, 2, 2, 512)
```

```
  downsample(512, 4),  # (bs, 1, 1, 512)
]

up_stack = [
  upsample(512, 4),  # (bs, 2, 2, 1024)
  upsample(512, 4),  # (bs, 4, 4, 1024)
  upsample(256, 4),  # (bs, 8, 8, 512)
  upsample(128, 4),  # (bs, 16, 16, 256)
  upsample(64, 4),  # (bs, 32, 32, 128)
]

initializer = tf.random_normal_initializer(0., 0.02)
last = tf.keras.layers.Conv2DTranspose(3, 4,
                                       strides=2,
                                       padding='same',
                                       kernel_initializer=initializer,
                                       activation='tanh')  # (bs, 128, 128, 3)

x = inputs

# Downsampling through the model
skips = []
```

```
  for down in down_stack:
    x = down(x)
    skips.append(x)

  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    x = tf.keras.layers.Concatenate()([x, skip])

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)

image_shape = (64,64,3)

generator_g = define_generator()
generator_f = define_generator()
discriminator_x = define_discriminator(image_shape)
discriminator_y = define_discriminator(image_shape)
```

```
discriminator_x.summary()

tfk.utils.plot_model(generator_g, show_shapes=True, dpi=64)

# x -> y: normal -> covid - generator_g
# y -> x: covid -> normal - generator_f

c_images, _ = next(iter(covid_ds))
n_images, _ = next(iter(normal_ds))

to_covid = generator_g(n_images)
to_normal = generator_f(c_images)
plt.figure(figsize=(6, 6))

imgs = [n_images, to_covid, c_images, to_normal]
title = ['Normal', 'To_Covid', 'Covid', 'To_normal']

plt.imshow(tf.keras.preprocessing.image.array_to_img(imgs[0][0]))
plt.suptitle("Mapping of generators before training", fontsize=14)
for i in range(len(imgs)):
  plt.subplot(2, 2, i+1)
  plt.title(title[i])
```

```
  plt.axis(False)
  if i % 2 == 0:
    plt.imshow(tf.keras.preprocessing.image.array_to_img(imgs[i][0]))
  else:
    plt.imshow(tf.keras.preprocessing.image.array_to_img(imgs[i][0]))
plt.show()

# the hotmap of untrained discriminator
plt.figure(figsize=(8,8))
plt.subplot(121)
plt.title('discriminate positive')
plt.axis(False)
plt.imshow(discriminator_y(c_images)[0, ..., -1], cmap='RdBu_r')

plt.subplot(122)
plt.title('discriminate negative')
plt.axis(False)
plt.imshow(discriminator_x(n_images)[0, ..., -1], cmap='RdBu_r')
plt.show()

"""Define the loss functions for the GAN components"""
```

```
LAMBDA = 80
# alternative: MSE
# loss_obj = tf.keras.losses.BinaryCrossentropy(from_logits=True)
loss_obj = tfk.losses.MeanSquaredError()

def discriminator_loss(real, generated):
  real_loss = loss_obj(tf.ones_like(real), real)
  generated_loss = loss_obj(tf.zeros_like(generated), generated)
  total_disc_loss = real_loss + generated_loss
  return total_disc_loss * 0.5

def generator_loss(generated):
  return loss_obj(tf.ones_like(generated), generated)

def calc_cycle_loss(real_image, cycled_image):
  loss1 = tf.reduce_mean(tf.abs(real_image - cycled_image))

  return LAMBDA * loss1

# prevser color
def identity_loss(real_image, same_image):
  loss = tf.reduce_mean(tf.abs(real_image - same_image))
```

```
  return LAMBDA * 0.5 * loss


"""Set up the solver for the GAN components"""


generator_g_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
generator_f_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)


discriminator_x_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_y_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)


checkpoint_path = "./checkpoints/train"


ckpt = tf.train.Checkpoint(generator_g=generator_g,
                           generator_f=generator_f,
                           discriminator_x=discriminator_x,
                           discriminator_y=discriminator_y,
                           generator_g_optimizer=generator_g_optimizer,
                           generator_f_optimizer=generator_f_optimizer,
                           discriminator_x_optimizer=discriminator_x_optimizer,
                           discriminator_y_optimizer=discriminator_y_optimizer)


ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)
```

```
# if a checkpoint exists, restore the latest checkpoint.
if ckpt_manager.latest_checkpoint:
  ckpt.restore(ckpt_manager.latest_checkpoint)
  print ('Latest checkpoint restored!!')


os.mkdir('gen_images')


EPOCHS = 600


def generate_images(model, test_input, epoch):
  prediction = model(test_input)
  idx = int(np.random.choice(16, 1, replace=False))
  plt.figure(figsize=(10, 10))



  display_list = [test_input[idx], prediction[idx]]
  title = ['Input Image', 'Predicted Image']


  for i in range(2):
    plt.subplot(1, 2, i+1)
    plt.title(title[i])
```

```
    # getting the pixel values between [0, 1] to plot it.
    plt.imshow(tf.keras.preprocessing.image.array_to_img(display_list[i]))
    plt.axis('off')
  plt.savefig('./gen_images/image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()

def classify_normal_loss(y, batch_size):
  true_normal = tf.ones([batch_size,1])
  pred_y = classifier(y)
  return class_loss(true_normal, pred_y)

def classify_covid_loss(x, batch_size):
  true_covid = tf.zeros([batch_size,1])
  pred_x = classifier(x)
  return class_loss(true_covid, pred_x)

def train_step(real_x, real_y, epoch, c_flag=True):
  # persistent is set to True because the tape is used more than
  # once to calculate the gradients.
  batch_size = real_x.shape[0]
  with tf.GradientTape(persistent=True) as tape:
    # Generator G translates X -> Y
```

```
# Generator F translates Y -> X.

fake_y = generator_g(real_x, training=True)
cycled_x = generator_f(fake_y, training=True)

fake_x = generator_f(real_y, training=True)
cycled_y = generator_g(fake_x, training=True)

# same_x and same_y are used for identity loss.
same_x = generator_f(real_x, training=True)
same_y = generator_g(real_y, training=True)

disc_real_x = discriminator_x(real_x, training=True)
disc_real_y = discriminator_y(real_y, training=True)

disc_fake_x = discriminator_x(fake_x, training=True)
disc_fake_y = discriminator_y(fake_y, training=True)

# calculate the loss
gen_g_loss = generator_loss(disc_fake_y)
gen_f_loss = generator_loss(disc_fake_x)
```

```
    # calculate classifier loss
    # true_normal = tf.ones([batch_size,])
    # true_covid = tf.zeros([batch_size,])
    pred_same_x = classifier(same_x)
    pred_same_y = classifier(same_y)
    pred_cycled_x = classifier(cycled_x)
    pred_cycled_y = classifier(cycled_y)

    c_loss_x = classify_normal_loss(same_x, batch_size) +
classify_normal_loss(cycled_x, batch_size)
    c_loss_x = c_loss_x * 0.05
    c_loss_y = classify_covid_loss(same_y, batch_size) + classify_covid_loss(cycled_y,
batch_size)
    c_loss_y = c_loss_y * 0.05
    total_cycle_loss = calc_cycle_loss(real_x, cycled_x) + calc_cycle_loss(real_y,
cycled_y)

    if (c_flag):
      total_gen_g_loss = gen_g_loss + total_cycle_loss + identity_loss(real_y, same_y) +
c_loss_y
      total_gen_f_loss = gen_f_loss + total_cycle_loss + identity_loss(real_x, same_x) +
c_loss_x
```

```
  else:
    total_gen_g_loss = gen_g_loss + total_cycle_loss + identity_loss(real_y, same_y)
    total_gen_f_loss = gen_f_loss + total_cycle_loss + identity_loss(real_x, same_x)

  disc_x_loss = discriminator_loss(disc_real_x, disc_fake_x)
  disc_y_loss = discriminator_loss(disc_real_y, disc_fake_y)

# Calculate the gradients for generator and discriminator
generator_g_gradients = tape.gradient(total_gen_g_loss,
                                      generator_g.trainable_variables)
generator_f_gradients = tape.gradient(total_gen_f_loss,
                                      generator_f.trainable_variables)

discriminator_x_gradients = tape.gradient(disc_x_loss,
                                          discriminator_x.trainable_variables)
discriminator_y_gradients = tape.gradient(disc_y_loss,
                                          discriminator_y.trainable_variables)

# Apply the gradients to the optimizer
generator_g_optimizer.apply_gradients(zip(generator_g_gradients,
                                          generator_g.trainable_variables))
```

```
  generator_f_optimizer.apply_gradients(zip(generator_f_gradients,
                                            generator_f.trainable_variables))

  discriminator_x_optimizer.apply_gradients(zip(discriminator_x_gradients,
                                                discriminator_x.trainable_variables))

  discriminator_y_optimizer.apply_gradients(zip(discriminator_y_gradients,
                                                discriminator_y.trainable_variables))

  return c_loss_x, c_loss_y, total_cycle_loss, total_gen_g_loss, total_gen_f_loss,
disc_x_loss, disc_y_loss

"""Build the dataset for the GAN training"""

def process_gan_path(file_path):
  img = tf.io.read_file(file_path)
  img = decode_img(img)
  img = (img - 127.5) / 127.5
  return img

covid_dataset = tf.data.Dataset.list_files(covid_path, shuffle=True)
normal_dataset = tf.data.Dataset.list_files(normal_path[:1014], shuffle=True)
```

```
BUFFER_SIZE = 1014

BATCH_SIZE = 64


covid_dataset = covid_dataset.map(process_gan_path,

num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

normal_dataset = normal_dataset.map(process_gan_path,

num_parallel_calls=AUTOTUNE).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)


import time

from IPython.display import clear_output

from IPython import display


history = {}

history['class_loss_x'] = []

history['class_loss_y'] = []

history['cycle_loss'] = []

history['total_gen_g_loss'] = []

history['total_gen_f_loss'] = []

history['disc_loss_x'] = []

history['disc_loss_y'] = []

history['used_time'] = []
```

```
c_loss_x_mean = tfk.metrics.Mean()

c_loss_y_mean = tfk.metrics.Mean()

cycle_loss_mean = tfk.metrics.Mean()

total_gen_g_loss_mean = tfk.metrics.Mean()

total_gen_f_loss_mean = tfk.metrics.Mean()

disc_loss_x_mean = tfk.metrics.Mean()

disc_loss_y_mean = tfk.metrics.Mean()

uesed_time_mean = tfk.metrics.Mean()


for epoch in range(600):

  start = time.time()

  c_loss_x_mean.reset_state()

  c_loss_y_mean.reset_state()

  cycle_loss_mean.reset_state()

  total_gen_g_loss_mean.reset_state()

  total_gen_f_loss_mean.reset_state()

  disc_loss_x_mean.reset_state()

  disc_loss_y_mean.reset_state()

  uesed_time_mean.reset_state()


  n = 0
```

```
  for image_x, image_y in tf.data.Dataset.zip((normal_dataset, covid_dataset)):
    if (n % 10 == 0):
      c_loss_x, c_loss_y, total_cycle_loss, total_gen_g_loss, total_gen_f_loss, disc_x_loss, disc_y_loss = train_step(image_x, image_y, epoch, c_flag=True) # chang c_flag to False if want to remove the criterion
    else:
      c_loss_x, c_loss_y, total_cycle_loss, total_gen_g_loss, total_gen_f_loss, disc_x_loss, disc_y_loss = train_step(image_x, image_y, epoch, c_flag=False)
    c_loss_x_mean.update_state(c_loss_x)
    c_loss_y_mean.update_state(c_loss_y)
    cycle_loss_mean.update_state(total_cycle_loss)
    total_gen_g_loss_mean.update_state(total_gen_g_loss)
    total_gen_f_loss_mean.update_state(total_gen_f_loss)
    disc_loss_x_mean.update_state(disc_x_loss)
    disc_loss_y_mean.update_state(disc_y_loss)
    uesed_time_mean.update_state(time.time()-start)

    if n % 10 == 0:
      print ('.', end='')
    n += 1

  clear_output(wait=True)
```

```
# Using a consistent image (sample_horse) so that the progress of the model
# is clearly visible.
if (epoch + 1) % 5 == 0:
  generate_images(generator_g, n_images, epoch)

if (epoch + 1) % 100 == 0:
  ckpt_save_path = ckpt_manager.save()
  print ('Saving checkpoint for epoch {} at {}'.format(epoch+1,
                                        ckpt_save_path))
history['class_loss_x'].append(c_loss_x_mean.result().numpy())
history['class_loss_y'].append(c_loss_y_mean.result().numpy())
history['cycle_loss'].append(cycle_loss_mean.result().numpy())
history['total_gen_g_loss'].append(total_gen_g_loss_mean.result().numpy())
history['total_gen_f_loss'].append(total_gen_f_loss_mean.result().numpy())
history['disc_loss_x'].append(disc_loss_x_mean.result().numpy())
history['disc_loss_y'].append(disc_loss_y_mean.result().numpy())
history['used_time'].append(uesed_time_mean.result().numpy())

print ('Time taken for epoch {} is {} sec\n'.format(epoch + 1,
                                        time.time()-start))

"""Visualize the loss values of the generators and discriminators"""
```

```
# save the trained
generator_g.save('covid_generator_64')
!zip -r covid_generator_64.zip covid_generator_64

# save the trained
generator_f.save('normal_generator_64')
!zip -r normal_generator_64.zip normal_generator_64

plt.figure(figsize=(15, 8))

plt.subplot(2, 2, 1)
plt.plot(history['total_gen_g_loss'], label='Total Generator G loss')
plt.plot(history['total_gen_f_loss'], label='Total Generator F loss')
plt.legend(loc='best')
plt.title('Generator Loss')
plt.xlabel('epoch')

plt.subplot(2, 2, 2)

plt.plot(history['disc_loss_y'], label='Total Discriminator Y loss')
plt.plot(history['disc_loss_x'], label='Total Discriminator X loss')
```

```
plt.legend(loc='best')
plt.title('Discriminator Loss')
plt.xlabel('epoch')

plt.subplot(2, 2, 3)

plt.plot(history['class_loss_x'], label='Criterion X loss')
plt.legend(loc='best')
plt.title('Criterion X loss')
plt.xlabel('epoch')

plt.subplot(2, 2, 4)
plt.tight_layout()
plt.plot(history['class_loss_y'], label='Criterion Y loss')
plt.legend(loc='best')
plt.title('Criterion Y loss')
plt.xlabel('epoch')

plt.plot(history['cycle_loss'], label='Total cycle loss')
plt.legend(loc='best')
plt.title('Total cycle loss')
plt.xlabel('epoch')
```

```
plt.plot(history['used_time'], label='Runtime per epoch')
plt.legend(loc='best')
plt.title('Optimization runtime per epoch')
plt.xlabel('epoch')
plt.ylabel('second')

"""X -- > Y"""

n_images = next(iter(normal_dataset))
gen_images = generator_g(n_images)

plt.figure(figsize=(12,12))

for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  if i % 2 == 0:
    image = tf.keras.preprocessing.image.array_to_img(n_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Input')
  else:
```

```
    image = tf.keras.preprocessing.image.array_to_img(gen_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Generated')

input_images = []
generated_images = []
for i in range(n_images.shape[0]):
  input_images.append(tf.keras.preprocessing.image.array_to_img(n_images[i, :, :,
:]).convert('L'))
  generated_images.append(tf.keras.preprocessing.image.array_to_img(gen_images[i, :, :,
:]).convert('L'))

MSE = []
RMSE = []
PSNR = []
UQI = []
SCC = []
RASE =[]
SAM = []
VIF = []

for j in range(len(input_images)):
```

```
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
  VIF.append(vifp(gen, org))

MSE = np.array(MSE)
RMSE = np.array(RMSE)
PSNR = np.array(PSNR)
UQI = np.array(UQI)
SCC = np.array(SCC)
RASE = np.array(RASE)
SAM = np.array(SAM)
VIF = np.array(VIF)

print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")
print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")
```

```
print(f'PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ')
print(f'UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ')
print(f'SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ')
print(f'RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ')
print(f'SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ')
print(f'VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ')

preds = classifier(gen_images)
true_labels = tf.zeros([64,1])
print(f'classify accuracy: {get_accuracy(true_labels, preds)}')

"""Y --> Y"""

n_images = next(iter(covid_dataset))
gen_images = generator_g(n_images)

plt.figure(figsize=(12,12))

for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  if i % 2 == 0:
```

```
    image = tf.keras.preprocessing.image.array_to_img(n_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Input')
  else:
    image = tf.keras.preprocessing.image.array_to_img(gen_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Generated')


input_images = []
generated_images = []
for i in range(n_images.shape[0]):
  input_images.append(tf.keras.preprocessing.image.array_to_img(n_images[i, :, :,
:]).convert('L'))
  generated_images.append(tf.keras.preprocessing.image.array_to_img(gen_images[i, :, :,
:]).convert('L'))


MSE = []
RMSE = []
PSNR = []
UQI = []
SCC = []
RASE =[]
```

```
SAM = []
VIF = []


for j in range(len(input_images)):
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
  VIF.append(vifp(gen, org))


MSE = np.array(MSE)
RMSE = np.array(RMSE)
PSNR = np.array(PSNR)
UQI = np.array(UQI)
SCC = np.array(SCC)
RASE = np.array(RASE)
SAM = np.array(SAM)
```

```
VIF = np.array(VIF)

print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")
print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")
print(f"PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ")
print(f"UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ")
print(f"SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ")
print(f"RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ")
print(f"SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ")
print(f"VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ")

preds = classifier(gen_images)
true_labels = tf.zeros([64,1])
print(f"classify accuracy: {get_accuracy(true_labels, preds)}")

"""Y -- X by GEN F"""

n_images = next(iter(covid_dataset))
gen_images = generator_f(n_images)

plt.figure(figsize=(12,12))
```

```
for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  if i % 2 == 0:
    image = tf.keras.preprocessing.image.array_to_img(n_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Input')
  else:
    image = tf.keras.preprocessing.image.array_to_img(gen_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Generated')


input_images = []
generated_images = []
for i in range(n_images.shape[0]):
  input_images.append(tf.keras.preprocessing.image.array_to_img(n_images[i, :, :,
:]).convert('L'))
  generated_images.append(tf.keras.preprocessing.image.array_to_img(gen_images[i, :, :,
:]).convert('L'))


MSE = []
RMSE = []
```

```
PSNR = []
UQI = []
SCC = []
RASE =[]
SAM = []
VIF = []

for j in range(len(input_images)):
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
  VIF.append(vifp(gen, org))

MSE = np.array(MSE)
RMSE = np.array(RMSE)
PSNR = np.array(PSNR)
```

```
UQI = np.array(UQI)

SCC = np.array(SCC)

RASE = np.array(RASE)

SAM = np.array(SAM)

VIF = np.array(VIF)


print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")

print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")

print(f"PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ")

print(f"UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ")

print(f"SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ")

print(f"RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ")

print(f"SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ")

print(f"VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ")


preds = classifier(gen_images)

true_labels = tf.zeros([64,1])

print(f"classify accuracy: {get_accuracy(true_labels, preds)}")


"""X --> X by GEN F"""


n_images = next(iter(normal_dataset))
```

```
gen_images = generator_f(n_images)


plt.figure(figsize=(12,12))


for i in range(4 * 4):
  plt.subplot(4, 4, 1+i)
  plt.axis(False)
  if i % 2 == 0:
    image = tf.keras.preprocessing.image.array_to_img(n_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Input')
  else:
    image = tf.keras.preprocessing.image.array_to_img(gen_images[i,:,:,:])
    plt.imshow(image)
    plt.title('Generated')


input_images = []
generated_images = []
for i in range(n_images.shape[0]):
  input_images.append(tf.keras.preprocessing.image.array_to_img(n_images[i, :, :,
:]).convert('L'))
```

```
  generated_images.append(tf.keras.preprocessing.image.array_to_img(gen_images[i, :, :,
:]).convert('L'))

MSE = []
RMSE = []
PSNR = []
UQI = []
SCC = []
RASE =[]
SAM = []
VIF = []

for j in range(len(input_images)):
  gen = tf.keras.preprocessing.image.img_to_array(generated_images[j]).astype('uint8')
  org = tf.keras.preprocessing.image.img_to_array(input_images[j]).astype('uint8')
  MSE.append(mse(gen,org))
  RMSE.append(rmse(gen, org))
  PSNR.append(psnr(gen, org))
  UQI.append(uqi(gen, org))
  SCC.append(scc(gen, org))
  RASE.append(rase(gen, org))
  SAM.append(sam(gen, org))
```

```
  VIF.append(vifp(gen, org))


MSE = np.array(MSE)

RMSE = np.array(RMSE)

PSNR = np.array(PSNR)

UQI = np.array(UQI)

SCC = np.array(SCC)

RASE = np.array(RASE)

SAM = np.array(SAM)

VIF = np.array(VIF)


print(f"MSE ---- mean: {MSE.mean()}, std: {MSE.std()} ")

print(f"RMSE: ---- mean: {RMSE.mean()}, std: {RMSE.std()} ")

print(f"PSNR: ---- mean: {PSNR.mean()}, std: {PSNR.std()} ")

print(f"UQI: ---- mean: {UQI.mean()}, std: {UQI.std()} ")

print(f"SCC: ---- mean: {SCC.mean()}, std: {SCC.std()} ")

print(f"RASE: ---- mean: {RASE.mean()}, std: {RASE.std()} ")

print(f"SAM: ---- mean: {SAM.mean()}, std: {SAM.std()} ")

print(f"VIF: ---- mean: {VIF.mean()}, std: {VIF.std()} ")


preds = classifier(gen_images)

true_labels = tf.zeros([64,1])
```

```
print(f"classify accuracy: {get_accuracy(true_labels, preds)}")


"""FID"""


from numpy import cov
from numpy import trace
from numpy import iscomplexobj
from numpy import asarray
from numpy.random import shuffle
from scipy.linalg import sqrtm
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.datasets.mnist import load_data
from tensorflow.keras.datasets import cifar10


# scale an array of images to a new size
def scale_images(images, new_shape):
  images_list = list()
  for image in images:
    # resize with nearest neighbor interpolation
    new_image = tf.image.resize(image, new_shape)
    # store
```

```
    images_list.append(new_image)
  return asarray(images_list)


# calculate frechet inception distance
def calculate_fid(model, images1, images2):
  # calculate activations
  act1 = model.predict(images1)
  act2 = model.predict(images2)
  # calculate mean and covariance statistics
  mu1, sigma1 = act1.mean(axis=0), cov(act1, rowvar=False)
  mu2, sigma2 = act2.mean(axis=0), cov(act2, rowvar=False)
  # calculate sum squared difference between means
  ssdiff = np.sum((mu1 - mu2)**2.0)
  # calculate sqrt of product between cov
  covmean = sqrtm(sigma1.dot(sigma2))
  # check and correct imaginary numbers from sqrt
  if iscomplexobj(covmean):
    covmean = covmean.real
  # calculate score
  fid = ssdiff + trace(sigma1 + sigma2 - 2.0 * covmean)
  return fid
```

```
model = InceptionV3(include_top=False, pooling='avg', input_shape=(299,299,3))

FID = []

for images in covid_dataset:
  images1 = images
  images2 = generator_g(images)
  images1 = preprocess_input(images1)
  images2 = preprocess_input(images2)
  fid = calculate_fid(classifier, images1, images2)
  FID.append(fid)

FID = np.array(FID)
print("Adaptive Cycle GAN Y to Y by Gen G------")
print(f'Adjusted FID Mean: {FID.mean()}, Std: {FID.std()}')

FID = []

for images in normal_dataset:
  images1 = images
  images2 = generator_g(images)
  images1 = preprocess_input(images1)
```

```
  images2 = preprocess_input(images2)
  fid = calculate_fid(classifier, images1, images2)
  FID.append(fid)

FID = np.array(FID)
print("Adaptive Cycle GAN X to Y by Gen G------")
print(f'Adjusted FID Mean: {FID.mean()}, Std: {FID.std()}')

FID = []

for images in normal_dataset:
  images1 = images
  images2 = generator_f(images)
  images1 = preprocess_input(images1)
  images2 = preprocess_input(images2)
  fid = calculate_fid(classifier, images1, images2)
  FID.append(fid)

FID = np.array(FID)
print("Adaptive Cycle GAN X to X by Gen F------")
print(f'Adjusted FID Mean: {FID.mean()}, Std: {FID.std()}')
```

```
FID = []


for images in covid_dataset:
  images1 = images
  images2 = generator_f(images)
  images1 = preprocess_input(images1)
  images2 = preprocess_input(images2)
  fid = calculate_fid(classifier, images1, images2)
  FID.append(fid)


FID = np.array(FID)
print("Adaptive Cycle GAN Y to X by Gen F------")
print(f'Adjusted FID Mean: {FID.mean()}, Std: {FID.std()}')
```