EXPLORING LOW-DIMENSIONAL STRUCTURES IN IMAGES USING DEEP FOURIER MACHINES

BEHNAM ASADI

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING & COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

NOVEMBER 2022

Abstract

The ground-breaking results achieved by Deep Generative Models, when given merely a dataset representing the desired distribution of generated images have caught the interest of scholars. In this work, we introduce a novel structure designed for image generation utilizing the idea behind Fourier Series and Deep Learning function composition. By composing low-dimensional structures, we will first compress a high-dimensional image, and then we will use this latent space to generate fake images. Our compression algorithm gives comparable results to the JPEG algorithm and even, in some cases, outperforms it. Also, our image generation model can generate decent fake images on MNIST and CIFAR-10 datasets and can surpass the first generation of Variational Autoencoders.

Acknowledgements

I would like to express my appreciation to my supervisor, Professor Hui Jiang, for developing the central idea for this work and providing constant financial support throughout the conduct of this research. I also acknowledge my examining committee, Professor Gene Cheung and Professor Amy Wu, for their time and advice on the thesis in advance. Lastly, I would like to express my gratitude to my colleagues at the Laboratory for Neural Computing and Machine Learning (NCML) for their invaluable advice.

Table of Contents

List of Tables

List of Figures

Abbreviations

Table 1: Abbreviations list

Abbreviation	**Full form**
JPEG	Joint Photographic Experts Group
DFT	Discrete Fourier Transform
AI	Artificial Intelligence
ANN	Artificial Neural Networks
NN	Neural Networks
GAN	Generative Adversarial Networks
VAE	Variational Autoencoder
NF	Normalizing Flow
CNN	Convolutional Neural Networks
MLP	Multi Layer Perceptron
RGB	Red Green Blue (image contex)

Continued on next page

Table 1 – continued from previous page

Abbreviation	Full form
FID	Fréchet Inception Distance
IS	Inception Score
KL-divergence	Kullback–Leibler divergence
EM-distance	Earth Mover's Distance
DCT	Discrete Cosine Transform
CPU	Central Processing Unit
GPU	Graphical Processing Unit
i.i.d	Independent and Identically Distributed
FC	Fully Connected
BN	Batch Norm
LN	Layer Norm
ReLU	Rectified Linear Unit

Chapter 1

Introduction

A generative model models the generation of data in the real world. It explains how data is created in terms of a probabilistic model. Image editing, visual domain adaptation, data augmentation for discriminative models, and assisted creative production are all examples of generative models. To be specific, datasets often have fewer data points in some sections of their domain, which might reduce model performance if not handled appropriately. Generative models may be used to modify datasets and upsample low-density areas. This is particularly beneficial for skewed datasets and simulated data settings. In addition to that, in a wide range of mathematics and engineering areas, high-dimensional probability distributions are crucial elements. As in many domains, such as visual data, most distribution content is constrained to a small region of space, we can safely compress it to a space with a lower dimension. A great test of our capacity to represent and work with high-dimensional probability distributions is the training and sampling of generative models that use the same approach. While significant

progress has been achieved, there are still issues that must be addressed. For example, high-quality synthesis for complicated scenes or multi-class datasets is still a long way off. There is a lack of universal objective criteria for assessing the quality of produced samples. While image production has delivered remarkable results, there is still a significant possibility for advancement in new areas such as cross-modal generation and video generation. In this work, we are trying to propose a new network in order to explicitly incorporate the frequency representation in the process of image generation in order to create a stable novel image generative model. The backbone of our model is a composition of low-frequency low-dimensional structures, which is designed to store high-dimensional images in a low-dimensional space in the frequency domain. If all images in a dataset could be assigned to separate points in our proposed latent space, we could get new samples from this space to generate fake images. We have two main contributions in this thesis:

- Introducing a unique Fourier-based algorithm for the lossy image compression task with an adjustable compression rate.
- Proposing a novel image generative model which relies on the same idea behind the compression task.

Chapter 2

Backgrounds

In this section, we first start with deep learning models, the reason behind their success, and their main building blocks. Then we introduce some deep generative model structures. We describe their structure, the core mathematical notion behind them, how they are designed to operate intuitively and mention their pros and cons. In the end, we give a brief review of methods to evaluate the quality of image generative models.

2.1 Deep Learning

In recent years, Deep Learning has revolutionized Artificial Intelligence (AI) techniques. From self-driving cars to language understanding services, almost all recent advances in AL are indebted to deep learning methods. As it appears from its application, the term “Deep Leaning" is used to name a broad set of algorithms which have totally different structures, but all these structures, which are usually called networks, work on the same principles. They

all consist of a composition of multiple functions, known as layers, each of which has an input-output relationship based on their learnable parameters, known as weights. The aim of the learning process is to find a set of weights for these layers so that the input-output relationship of the whole network estimates a target function. In practice, instead of a target function, we have a set of input-output pairs and we want to estimate the output of the target function for new inputs. It turned out that if we use a specific learning algorithm, known as Stochastic Gradient Descent (SGD), to minimize the error of these networks on a subset of large enough data, the predicted outputs for unseen data will be close enough to the real outputs for many different networks in different applications, although the minimization problem is not convex, which SGD is initially designed to work on. In the following paragraphs, we introduce some of the most common layers in Deep Learning in order to make it easier to understand the structure of our network.

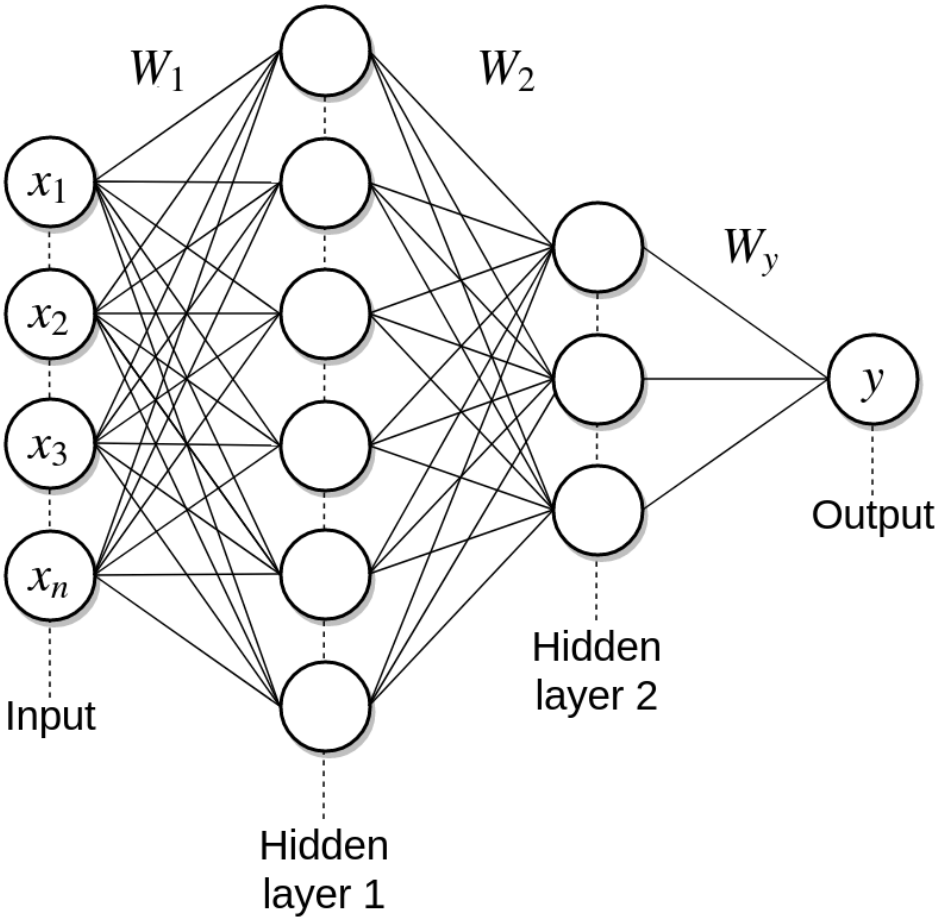
Multilayer Perceptron MLP or fully connected network is the most simple Neural Network and a building block for many Deep Learning models. It consists of multiple layers, each of which consists of a linear function and a pointwise nonlinear function, known as the activation function. The two most famous activation functions are ReLU [1] and Sigmoid functions. We can show this network as $\mathbf{MLP}_{\mathbf{w}}(\mathbf{x}) = f_d \circ f_{d-1} \circ \cdots \circ f_1(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^n$ is the input of the network and f_i is the i-th layer:

$$f_i(\mathbf{z}) = \phi(\mathbf{W}_i\mathbf{z} + \mathbf{w}_i) \tag{2.1}$$

where $f_i : \mathbb{R}^{n_{i-1}} \mapsto \mathbb{R}^{n_i}$ is a real function, $\mathbf{z} \in \mathbb{R}^{n_{i-1}}$ is the input of the layer, $\mathbf{W}_i \in \mathbb{R}^{n_i \times n_{i-1}}$

is a $n_i \times n_{i-1}$ real matrix, $\mathbf{w}^i \in \mathbb{R}^{n_i}$ is a n_i-dimensional real vector and $\phi : \mathbb{R}^{n_i} \mapsto \mathbb{R}^{n_i}$ is a pointwise real nonlinear function, e.g. ReLU or Sigmoid functions. We can also consider each layer of MLP as a set of nodes connected to nodes of the previous and next layer shown in figure 2.1. Here the value of each node represents each element of a vector in a layer. As you will see in the next chapters we are more interested in the first interpretation of an MLP.

Figure 2.1: A fully connected neural network with two hidden layers.



The training process of MLP - similar to other Neural Network models is based on Gradient Descent and Backpropagation algorithm. Considering the error function and its

divergence, these algorithms change the weights of the network so that the error of the output of the network decrease. This process continues until the error of the network becomes small enough. In practice, instead of Gradient Descent, Deep Learning models use SGD where the algorithm works on a small subset of the dataset, known as the batch in each iteration, rather than the whole dataset.

Convolutional Neural Networks A convolutional neural network (CNN) is another form of artificial neural network widely used in image recognition and processing that is specifically designed to process pixel data. The backbone of CNN is the convolutional layer which is a special kind of fully connected layer where most connections have been removed and many of the remaining connections share the same weight. This layer is designed in a way to satisfy two important properties. Locality modelling and shift-invariance. The reason behind the first property is that only a few close nodes (pixels) determine the value of a specific output node. The main idea is that close pixels usually represent the same object and share the same features, so each convolutional layer tries to extract these features into the next layer. The second property means when the input of the convolutional layer shifts by a vector, the output of the layer remains the same except for a shift by the same vector. This property ensures that changing the location of objects in the input image would result in a small change in the location of the output feature map, not more.

Although convolutional layers play a critical role in CNNs, many other layers are essential to get the best possible results from these models. Pooling layers and normalization layers are one of the most common layers in CNNs. Pooling layers are applied to compres the output of

convolutional layers and normalization layers try to smooth the loss function which is the input for the SGD algorithm.

2.2 Generative Models

Generally, there are two main machine learning models. Generative models vs discriminative models. In discriminative models, the objective is to train a model to figure out the relationship between features and labels so that in presence of features of new data (test set) model can predict the labels. On the other hand, in a generative task, models are usually trained to mimic the features (or even labels) and learn the distribution of the dataset in the space of all possible features. Image generative models are a subset of these models that try to generate images that are drawn from the same distribution as the images of the dataset. For example, if the dataset consists of images of human faces, the goal of a generative model is to generate new human faces that are not in the dataset.

The core idea behind image generative models is that we often have or observe really high dimensional data, in this case, images where there might be millions of dimensions in the data space, but images are actually not that high dimensional at the core. In other words, there is a smaller number of causal factors that explain the images that we observe which are much more compact than the actual signal itself and all image generative models use this idea and try to find this lower dimension space and sample new images from it.

There are many different types of probabilistic generative models starting from classical approaches like (Gaussian) Mixture Models and Energy-Based Models to the state-of-the-art

Generative Adversarial Networks, but most deep image generative models can be categorized into four different groups. Variational Autoencoder (VAEs), Generative Adversarial Networks (GANs), Normalizing Flows and Diffusion Models. In the next few paragraphs, we will discuss each of these models and their pros and cons.

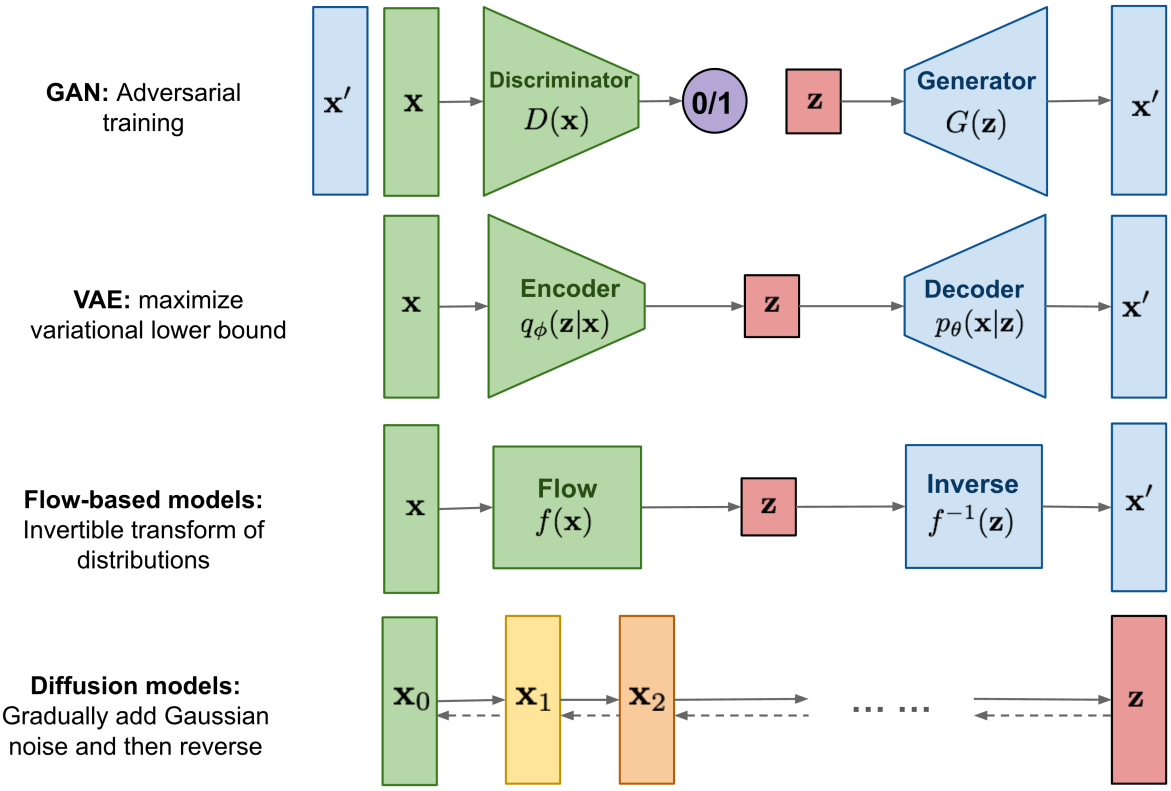


Figure 2.2: An overview of different deep generative model [2].

Generative Adversarial Networks GAN [3] is a brilliant way of training a generative model by framing the unsupervised learning problem as a supervised problem with two sub-modules: a generator $\mathcal{G}$, and a discriminator $\mathcal{D}$. Training these two networks together leads to a model which can generate images that are drawn from the same distribution as

the training set images. The generator is trained to produce realistic fake images from any random input. The fake examples produced by the generator are used as negative examples for training the discriminator. The discriminator is trained to distinguish fake data from realistic data. If the generator produces implausible results the discriminator penalizes the generator. If the result of the generator is good enough to deceive the discriminator, the discriminator gets penalized. This process continues until these two agents get experts in generating and discriminating fake images respectively.

Training a GAN requires finding a Nash equilibrium which is a saddle point and there is no guarantee that SGD-based algorithms can do it. In practice, finding a saddle point is way harder than a near-global optimum in a non-convex problem - at least in deep leaning models. This leads to an unstable training process in GANs compared to other deep generative models. Also sampling from GANs is easy and straightforward however evaluating the density is not possible.

Variational Autoencoder Autoencoders are a class of networks, that consist of a bottleneck which keeps a compressed version of input data. Half of the network, starting from input to bottleneck is called encoder and it attempts to encode the input and the other half, starting from the bottleneck to output is called decoder which tries to reconstruct the original data from the encoding, stored in the bottleneck. As a result, the model throws out some information, but because it still wants to reconstruct the signal, it has to preserve what's the important information for reconstructing the signal, as opposed to just preserving everything right. Autoencoders can be used as a compressing tool to decrease the dimension

of an input image but since they do not define a distribution of encoded data to be sampled from, they cannot operate as generative models. The issue of non-regularized latent space in autoencoders can be solved using Variational Autoencoders [4] so they could be used as generative models. In VAEs encoder outputs means and variances of a joint independent normal distribution in the latent space for every input instead of outputting a vector in the latent space. This constraint makes the latent space regularized.

An advantage of VAEs is that there is a straightforward way to evaluate the quality of the model (log-likelihood, either estimated by importance sampling or lower-bounded) compare to GANs which is not obvious how to compare them except by some other metrics like FID and Inception Score, to name a few, which they all have their own problems. Here in VAEs sampling from the distribution is simple and only approximate evaluation of density function is possible. Also, the outputs of VAE models are usually not as sharp as GANs.

Normalizing Flows Basically, Normalizing Flows [5] are image generative models built on invertible transformations. Based on change of variable rule in probability, given $p_{\mathbf{x}}(\mathbf{x})$ and a fixed normal distribution $p_{\mathbf{z}}(\mathbf{z})$, Normalizing Flows aim to find an invertible and differentiable function $\mathbf{f}(\mathbf{x})$ such that:

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{z}}(\mathbf{f}(\mathbf{x}))|\det D\mathbf{f}(\mathbf{x})|. \tag{2.2}$$

where $p_{\mathbf{z}}(\mathbf{z})$ is typically selected as $\mathcal{N}(\mathbf{z}|0, \mathbf{I})$. Now if we somehow find function $\mathbf{f}(\mathbf{x})$, as it is invertible, we would have $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{z})$ and as $p_{\mathbf{z}}(\mathbf{z})$ is a normal distribution, we have an exact estimation of $p_{\mathbf{x}}(\mathbf{x})$.

Using Normalizing Flows it is possible to get an exact estimate of the likelihood of the samples, as well as in the reverse direction but NFs are harder to train and are usually larger for the same quality.

Diffusion Models Diffusion probabilistic models (DPMs) [6], or diffusion models for short, can be viewed as a form of VAE, whose structure and loss function enable effective training of arbitrarily large models [7]. Non-equilibrium thermodynamics serves as the basis for diffusion models. They learn to reverse the diffusion process to create desired data samples from the noise after constructing a Markov chain of diffusion steps to gradually introduce random noise to data. Diffusion models, in contrast to VAE models, are trained using a fixed process, and the latent variable has a high degree of dimension, same as the original data. These models are consist of two process. A forward process (also known as a diffusion process), in which a picture is gradually noised, and a reverse process (also known as a reverse diffusion process), in which the noisy images is changed back into a sample from the target distribution.

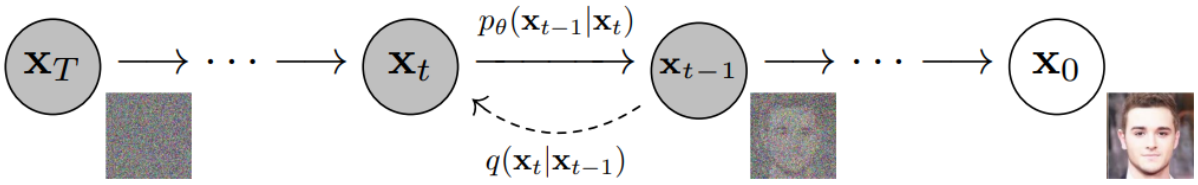


Figure 2.3: The Gaussian conditional chain transitions are the product of the joint distribution of the latent variables under the Markov assumption [8].

Same as Normalizing Flows, Diffusion models are both analytically tractable and it is possible to get an exact estimate of the likelihood of the samples but the training process

may be extremely time and compute intensive since they depend on a long Markov chain of diffusion processes. Despite the fact that new techniques have been offered to speed up the process significantly, sampling still takes more time than GANs.

2.3 Evaluating Quality of Image Generative Models

There are several metrics to measure the performance of machine learning methods on supervised tasks, either classification or regression, but proposing a measure in unsupervised tasks, especially image generation is not as easy as it seems as there is no label on the data. It even gets worse in the case of GANs where the loss of the model tells us nothing not only about the quality of models but the quality of the generated images of one model over the training time. This leads to a lack of consensus on the way of evaluating a given image generative model. While many measures have been introduced, there is no general agreement as to which measure best captures the strengths and limitations of models and should be used for fair model comparison.

Manual Evaluation Many generative model practitioners fall back on the evaluation of them via the manual assessment of generated images. This involves using the generator model to create a batch of synthetic images, then evaluating the quality and diversity of the images in relation to the target domain. The model is trained iteratively over many training epochs. For GANs it is not even straightforward when the training process should stop and when a final model should be saved for later use, as there is no objective measure of model

performance. Therefore, it is common to use the current state of the model during training to generate a large number of synthetic images and to save the current state of the generator used to generate the images. This allows for the evaluation of each saved generator model using its generated images.

Inception Score Inception Score considers two main concepts. First, each image should belong to a class. It means that the image of a dog belongs to the dog class, and there is no doubt about putting it in another class. In other words, it is the confidence of the conditional class predictions for each synthetic image $p(y|\mathbf{x})$, where $\mathbf{x}$ is the generated image and y is its class. Second, the generated images should be diverse and include all classes which are the integral of the marginal probability of the predicted classes $p(y)$. It means that $p(y|\mathbf{x})$ should have low entropy and $p(y)$ should have high entropy, so if we put them in a KL divergence formula a higher number shows two highly different distributions which will satisfy both metrics.

$$\mathbf{IS}(\mathcal{G}) = \exp(\mathbb{E}_{\mathbf{x}\sim P_{\mathcal{G}}} D_{KL}(p(y|\mathbf{x})||p(y)) \tag{2.3}$$

In practice, researchers use Inception v3 network [9], pre-trained on the Imagenet dataset, as a classifier to calculate this KL-divergence.

Fréchet Inception Distance Fréchet Distance can be used to calculate the distance between two multivariate normal distributions. In the context of computer vision, FID [10] is a measure of similarity between two image datasets. Similar to Inception Score, it uses

Inception v3 network [9], pre-trained on the Imagenet dataset, but this time not the output but the feature map after the third pooling layer of the model. If we assume this feature map for each dataset has a multivariate Gaussian distribution, we can compare these datasets via their distributions. To be precise, we can estimate the mean vector and the covariance matrix for both real and synthetic data distributions, μ_r, μ_g, Σ_r and Σ_g respectively. Then FID score can be computed using the Fréchet Distance formula:

$$\mathbf{FID}(r, g) = ||\mu_r - \mu_g||_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}}) \tag{2.4}$$

where $Tr(\Sigma)$ is trace - sum of the diagonal elements - of the matrix Σ. Note that unlike the Inception Score where we wanted to have two different distributions in a KL-divergence formula therefore a higher score was a sign of a better model, here a more realistic generated dataset should have the same mean and covariance as the real dataset so a lower score is an indication of a better model.

Chapter 3

Literature Review

In this section, we will review four main image generative models: Generative Adversarial Networks, Variational Autoencoder, Normalizing Flows, and Diffusion Models. Due to the enormous number of papers that have been published in this area, we have to only mention the most important works.

3.1 Generative Adversarial Networks

Goodfellow et al. [3] have shown that procedure of generating and discriminating images can be formulated as following minimax game:

$$\min_{\theta_g} \max_{\theta_d} \left(\mathbb{E}_{\mathbf{x} \sim P_{\mathbf{x}}(\mathbf{x})}[\log \mathcal{D}_{\theta_d}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim P_{\mathbf{z}}(\mathbf{z})}[\log(1 - \mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\mathbf{z})))] \right). \tag{3.1}$$

where θ_g and θ_d are the parameters of the generator and discriminator respectively. $\mathbf{z}$ is random input noise to the generator and $P_{\mathbf{z}}(\mathbf{z})$ is the distribution over it. $\mathbf{x}$ is an image from

data distribution $P_{\mathbf{x}}(\mathbf{x})$. $\mathcal{G}_{\theta_g}(\mathbf{z})$ is the generator which get the random variable $\mathbf{z}$ and gives an image as its output and $\mathcal{D}_{\theta_d}(\mathbf{x})$ is the discriminator which get the - fake or real - image $\mathbf{x}$ and give the probability of $\mathbf{x}$ coming from the data distribution rather than generator distribution. It can be shown that the optimal solution for this minimax problem is where the generator output distribution matches the data distribution.

After Goodfellow et al. [3] proposed GAN in 2014, it has become one of the hottest research fields in machine learning and computer vision. Different types of GAN models have been designed in recent years, mostly to improve and stabilize the main model. GAN has the benefit of not relying on expected data distribution for input; the produced samples can be more realistic by employing direct sampling to input random noise Z. Simultaneously, GAN-generated samples are too free to regulate and cannot focus on a given class. Mehdi Mirza et al. introduced Conditional Generative Adversarial Nets or CGANs [11] where the label of each image was fed into the GAN generator and discriminator.

$$\min_{\theta_g} \max_{\theta_d} \left(\mathbb{E}_{\mathbf{x} \sim P_{\mathbf{x}}(\mathbf{x})} [\log \mathcal{D}_{\theta_d}(\mathbf{x}|y)] + \mathbb{E}_{\mathbf{z} \sim P_{\mathbf{z}}(\mathbf{z})} [\log(1 - \mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\mathbf{z}|y)))] \right). \tag{3.2}$$

In ACGAN [12] the access of the discriminator to the labels has been blocked but instead the discriminator has been trained to output two different variables. By using the idea of [13], the classification results of samples are added to the classical output of the discriminator which resulted in better performance. Researchers initially attempted to incorporate CNN in GAN, but no positive results were obtained [14]. In 2015, Alec Radford et al. published DCGAN paper [15], which enhanced the network structure of GAN and significantly raised the quality

of GAN-generated images as well as the stability of training. DCGAN removed the fully connected layers because of their inefficient training process on the images. Additionally, DCGAN employs batch normalization, which stops the generator from collapsing and allows deeper gradient propagation. The Wasserstein GAN or WGAN [16], is an extension that enhances model stability while training and gives a loss function that corresponds with image quality. The WGAN has a profound mathematical motivation, but in reality, just a few modest tweaks to DCGAN, are required. To stabilize the discriminator's training, [17] suggested spectral normalization, a unique weight normalization approach. Their normalizing approach is computationally light and simple to implement in current systems. They demonstrated that spectrally normalized GAN (SNGAN) may generate pictures of comparable or higher quality than earlier training stabilizing approaches. SAGAN or Self-Attention Generative Adversarial Network [18], enables attention-driven, long-term dependency modelling for image generation challenges. In lower-resolution feature maps, traditional convolutional GANs create high-resolution features as a function of only spatially local points. Details in SAGAN may be created utilizing inputs from all feature locations. Furthermore, the discriminator can ensure that highly detailed elements in different parts of the image are consistent with one another. With orthogonal regularisation and a truncation method, [19] continues its efforts to scale up the dataset. BigGAN altered the discriminator by including an additional path for encoded generated images, proposing a unique representation learning technique for training GANs. And lastly, [20] and [21] suggested the Affine Transformation and Adaptive Instance Normalization (AdaIN). Also, they modified the generator of the model in a way

that the noise input goes through an MLP to produce a style vector w, which is then fed into the generator architecture at various stages as opposed to the classical approach where the input noise z was fed into the generator directly.

3.2 Variational Autoencoder

As we mentioned in Background section encoder of VAEs outputs means and variances of a joint independent normal distribution in the latent space for every input instead of outputting a vector in that space. In practice, instead of maximizing the probability of getting a real-like image which leads to an intractable integral, VAEs objective is to maximize a lower bound on it, called Evidence Lower Bound or ELBO.

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log\Big[\frac{p_\theta(\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})}\Big]\Big] + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})). \tag{3.3}$$

Since Kullback–Leibler divergence is always positive, VAE would maximize $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log\Big[\frac{p_\theta(\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})}\Big]\Big]$ as a proxy for $\log p_\theta(\mathbf{x})$.

In 2014, [4] and [22] presented a scalable stochastic variational inference and learning technique for large datasets. The approach even works in the intractable scenario with minor differentiability requirements. Posterior inference may be made extremely efficient for i.i.d. datasets with continuous latent variables per data point. Two years later, [23] investigated the feasibility of adding latent random variables into the state of a recurrent neural network (RNN). They suggested that the variational RNN can simulate the type of variability seen in highly structured sequential data such as real speech by using high-level latent variables. In

2017, [24] proposed β-VAE which uses a hyperparameter that may be adjusted to reconcile latent channel capacity and independence limitations with reconstruction accuracy. They showed that β-VAE with correctly adjusted $\beta > 1$ outperforms VAE ($\beta = 1$) qualitatively. In order to quantify the trade-off between compression and reconstruction accuracy, [25] developed variational lower and upper bounds on the mutual information between the input and the latent variable. Then they utilize these bounds to generate a rate-distortion curve. With the help of this framework, they show that there is a family of models with the same ELBO but various quantitative and qualitative features. Additionally, their approach proposed a straightforward technique to prevent latent variable models with potent stochastic decoders from ignoring their latent code. In 2018, [26] suggested adding a new prior type to the VAE architecture, named “Variational Mixture of Posteriors" prior, or VampPrior. A blend of Gaussians distribution with components provided by variational posteriors conditional on learnable pseudo-inputs makes up the VampPrior. They demonstrated that this architecture, which has a coupled prior and posterior, learns noticeably superior models by extending this prior to a two-layer hierarchical model. The model also stays clear of the typical local optima problems caused by useless latent dimensions. In the paper [27], authors suggested Vector Quantised- Variational AutoEncoder (VQ-VAE) which the encoder network produces discrete outputs rather than continuous ones, in contrast to vanilla VAEs. Additionally, the prior is learned rather than static. The model can avoid posterior collapse issues where latents are disregarded when they are combined with a powerful autoregressive decoder, by using the vector quantization approach (VQ). In 2020, [28] presented Nouveau VAE (NVAE),

a deep hierarchical VAE designed for picture generation employing batch normalization and depth-wise separable convolutions. The training of NVAE is stabilized by spectral regularisation, and it has a residual parameterization of Normal distributions. VD-VAE [29] a hierarchical extremely deep VAE obtain greater likelihoods than the PixelCNN [30], employs fewer parameters, produces samples tens of thousands of times quicker, and is simpler to use with high-resolution pictures. This is because the VAE develops effective hierarchical visual representations, according to their qualitative research.

3.3 Normalizing Flows

Training with normalizing flows is most commonly done with maximum likelihood and because of the fact we can compute the density function exactly we can write out the log-likelihood of the set of datapoints $\mathbf{x}_i$:

$$\max_{\theta} \sum_{i=1}^{N} (\log p_{\mathbf{z}}(\mathbf{f}(\mathbf{x}_i|\theta) + \log |\det D\mathbf{f}(\mathbf{x}_i|\theta)|). \tag{3.4}$$

where θ are the parameters of the flow $\mathbf{f}(\mathbf{x}_i|\theta)$ which we aim to learn and $p_{\mathbf{z}}$ is a Gaussian density. Note that to make Normalizing Flows practically useful it has to be efficient to compute the Jacobian determinant $\det D\mathbf{f}(\mathbf{x})$. Designing and constructing these flows is the core research problem for Normalizing Flows. To do so researchers rely on the fact that invertible, differentiable functions are closed under *composition*. This allows us to build up complex flows from the composition of simpler flows - the same idea that we are going to use in building our model.

$$\mathbf{f} = \mathbf{f}_d \circ \mathbf{f}_{d-1} \circ \cdots \circ \mathbf{f}_1 \tag{3.5}$$

where each of $\mathbf{f}_1$ to $\mathbf{f}_d$ are invertible and differentiable. As the Jacobian of products is equal to the product of Jacobians, the log-likelihood problem we might use can be rewritten as:

$$\max_{\theta} \sum_{i=1}^{N} (\log p_{\mathbf{z}}(\mathbf{f}(\mathbf{x}_i|\theta) + \sum_{k=1}^{d} \log |\det D\mathbf{f}_k(\mathbf{x}_i|\theta)|). \tag{3.6}$$

There are many choices for $\mathbf{f}_i$ function. E.g., Elementwise Flows [31][32], Diagonal Flows [33][34], Triangular Flows [35], Permutation Flows [33][36], Orthogonal Flows [37], Factorizations [36][38], Convolution [36][38], Planar Flows [5], Radial Flows [5], Coupling Flows [39][33], Autoregressive Flows [40], Residual Flows [41][42][43], and Continuous Flows [44][45][46].

[39] provided a deep learning approach termed Non-linear Independent Component Estimation for modelling intricate high-dimensional densities (NICE). They trained a non-linear transformation of the data that maps it to a latent space and causes it to conform to a factorized distribution, producing independent latent variables. They parameterized this transformation in order to make it straightforward to calculate the determinant of the Jacobian and inverse Jacobian while retaining the capacity to learn complicated non-linear transformations using a combination of basic building blocks. Their proposed coupling layers are still one of the most utilized layers in normalizing flows. By defining a class of invertible functions (Real NVP) with a tractable Jacobian determinant, [33] made it possible to evaluate, infer, and sample log-likelihood with precision and tractability. They demonstrated that

both in terms of sample quality and log-likelihood, this class of generative models provides competitive results. Kingma and Dhariwal in [36] suggested Glow, a generative flow that makes use of an invertible 1×1 convolution. They showed a considerable improvement in log-likelihood on common benchmarks using their model. Most impressively, they showed that a flow-based generative network that is optimized for the basic log-likelihood objective is capable of producing realistic large-scale pictures with efficiency. [45] introduced FFJORD, a high-dimensional reversible generative model for data that leverages continuous-time dynamics to generate a generative model that is parameterized by an unconstrained neural network. Black-box ODE solvers, Hutchinson's trace estimator, and automatic differentiation may all be used to calculate the necessary values for training and sampling. Dequantization, flow design, and conditioning architecture design were three particular aspects of design concepts for flow models that [47] took into consideration and which bridged the performance gap between flow models and autoregressive models. Employing a "Russian roulette" estimator, [48] provided a tractable unbiased estimate of the log density and lowered the training memory need by using a different infinite series for the gradient. Additionally, by suggesting the usage of activation functions that prevent derivative saturation and extending the Lipschitz criterion to induced mixed norms, They enhanced invertible residual blocks. The resultant method, known as Residual Flows, performs better in joint generative and discriminative modelling than networks that have coupling blocks. SurVAE Flows [49] use surjective transformations to bridge the gap between normalizing flows and VAEs. The transformations are stochastic in the reverse direction, offering a lower bound on the likelihood, and deterministic in the forward

direction, enabling accurate likelihood calculation. They demonstrated that a number of new techniques, such as dequantization and enhanced normalizing flows, may be described as SurVAE Flows. Finally, they presented common operations like the absolute value, maximum value, stochastic permutation, and sorting as composable layers.

3.4 Diffusion Models

When the noise level is low enough, the forward process's sampling chain transitions can be shown as conditional Gaussians. This fact combined with the Markov hypothesis results in a parameterization of the forward process:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \Pi_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \Pi_{t=1}^{T} \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{3.7}$$

where $\mathbf{x}_1$ to $\mathbf{x}_T$ are the latent variables with the same dimensionality as $\mathbf{x}_0$, $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ is approximate posterior, and β_1 to β_T is an either learned or fixed variance schedule which ensures that $\mathbf{x}_T$ is as close as possible to an isotropic Gaussian for sufficiently large T.

But diffusion models' magic lies in the reverse procedure. The model learns to reverse the diffusion process during training in order to produce the output. Model starts with an isotropic Gaussian noise $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$ and learns the joint distribution $p(\mathbf{x}_{0:T})$ as below:

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_0)\Pi_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = p(\mathbf{x}_0)\Pi_{t=1}^{T} \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{3.8}$$

where $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$ can be learned by a deep learning model. Diffusion Models are trained by the reverse Markov transitions which maximize the likelihood of the training data. Here like VAEs training consists of minimizing the variational upper bound on the negative log-likelihood.

$$\mathbb{E}[-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_q[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}] = L_{vlb} = \sum_{t=0}^{T} L_t \tag{3.9}$$

Now we can rewrite L_t's as below:

$$L_0 = -\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)$$

$$L_t = D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))$$

$$L_T = D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))$$

Thanks to the fact that each KL term above compares two Gaussian distributions, they can all be calculated in closed form. Because $\mathbf{x}_T$ is a Gaussian noise and q has no learnable parameter, L_T is constant and may be disregarded during training. Note that the reverse diffusion process, $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$, requires us to learn a neural network to approximate the conditional normal distributions but instead of predicting less noisy image from the noisy one we can just predict the added noise itself. After some reparameterizations, we can train a model to predict $\mathbf{z}_\theta(\mathbf{x}_t, t)$ from $\mathbf{x}_t$ and t then plug it into the following formula to predict $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\mathbf{z}_\theta(\mathbf{x}_t, t)) \tag{3.10}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. According to experiments, parameterizing the model by the added noise's mean results in more stable training and better image synthesis. This outcome could be the result of the simpler distribution of the noise process compare to the natural image distribution function.

After [6] introduced the diffusion models in 2015, it took a long time for researchers to get a near-state-of-the-art performance. The main contribution to modifying these networks was from [8] in 2020, which added a number of model enhancements to the original DPM, which yielded remarkable results in the quality of generated images. They also discovered connections between diffusion models and variational inference and showed diffusion models excellent inductive biases for image data. Additional improvements were suggested by [50] which resulted in better likelihood scores. Furthermore, they discovered that learning the reverse diffusion process' variances, $\mathbf{\Sigma}_\theta(\mathbf{x}_t, t)$, enables sampling with an order of magnitude fewer forward passes and barely any difference in sample quality, which is crucial for the practical application of these models. Also, they demonstrated that these models are easily scalable due to the smooth scaling of their sample quality and likelihood with model capacity and training computation. [51] demonstrated how diffusion may be effectively utilized to optimize energy-based models (EBMs) towards an accurate approximation of the log-likelihood target, producing high-fidelity samples even after extensive Markov chains. [52] demonstrated that diffusion models provide image sample quality that is better than the most advanced generative models currently available. With classifier guidance, they further enhance sample quality for conditional image synthesis. They used a simple, computationally

efficient technique for balancing variety and fidelity using gradients from a classifier. With as little as 25 forward passes per sample, they got the same performance as BigGAN-deep. Finally, they demonstrate that classifier guidance enhances FID by combining well with upsampling diffusion models.

Chapter 4

Deep Fourier Machines (DFM)

All deep image generation models suffer from their own issues. GANs are unstable and generate samples with little diversity and because they are not trained using the log-likelihood method there is no obvious measure to compare them. Outputs of VAEs are usually not as sharp as other models. Normalizing Flows are huge models that need powerful computational power to train. The same problem occurs in diffusion models, which are computationally inefficient. In this work, we will try to propose a model that can remediate these problems.

4.1 Frequency Analysis

The Frequency Domain is an analytic space in which mathematical functions are displayed in terms of their frequency content, rather than time or space. For example, where a spatial image may display changes in the signal over space, a frequency-domain image displays how much of the signal is present in each given frequency band. It is possible to convert

information from a spatial domain to a frequency domain and vice versa. An example of such a transformation is the Fourier transform, which is usually used to convert continuous signals between the time and frequency domain. The frequency domain has close relationships with sine and cosine functions as they only contain one frequency in their spectrum. In fact, the Fourier transform is nothing but rewriting functions as an integral of a coefficient multiplied by the sine and cosine functions. Although there has been much work in the frequency domain in image processing, deep learning has limited the application of frequency analysis on images for a while. The frequency domain represents valuable information about the image in a different form, which in many applications can be helpful. Different types of representations have been introduced to manifest the frequency domain in either the continuous or discrete domain. Here we will have a quick review of the Fourier Series.

Fourier Series Fourier Series is a transformation from periodic continuous signals to the discrete frequency domain. Fundamentally, Fourier Series is just rewriting a periodic function as a weighted sum of orthogonal bases, in this case $f_k(x) = e^{2\pi ikx}$. Suppose we have a periodic function $\mathbf{h}\colon \mathbb{R}^m \to \mathbb{R}$. Under some loose constraints, we can calculate the Fourier Series for this function $\mathbf{c_k} \in \mathbb{C}^n$ as below:

$$\mathbf{h}(\mathbf{x}) = \sum_{k_1=-\infty}^{\infty} \cdots \sum_{k_m=-\infty}^{\infty} \mathbf{c_k} e^{2\pi i \mathbf{k}.\mathbf{x}} \tag{4.1}$$

where $\mathbf{k} = (k_1, \ldots, k_m) \in \mathbb{N}^m$. Now based on Euler's formula $e^{ix} = \cos x + i \sin x$ we can propose an equivalent form as below:

$$\mathbf{h}(\mathbf{x}) = \sum_{k_1=0}^{\infty} \cdots \sum_{k_m=0}^{\infty} \mathbf{a_k} \cos(2\pi \mathbf{k} \cdot \mathbf{x}) + \mathbf{b_k} \sin(2\pi \mathbf{k} \cdot \mathbf{x}) \tag{4.2}$$

where $\mathbf{a_k}, \mathbf{b_k} \in \mathbb{R}^n$. Based on Dirichlet Conditions what this equation exactly means is if we define error signal $\mathbf{e}_{l_1 \dots l_m}(\mathbf{x})$ as below:

$$\mathbf{e}_{l_1 \dots l_m}(\mathbf{x}) = \mathbf{h}(\mathbf{x}) - \sum_{k_1=0}^{l_1} \cdots \sum_{k_m=0}^{l_m} \mathbf{a_k} \cos(2\pi \mathbf{k} \cdot \mathbf{x}) + \mathbf{b_k} \sin(2\pi \mathbf{k} \cdot \mathbf{x}) \tag{4.3}$$

energy of the error signal $\mathbf{e}_{l_1 \dots l_m}(\mathbf{x})$ is zero when $l_1, \dots, l_m \to \infty$. So based on definition of limit, for every ϵ there is a sequence of $l_1, \dots, l_m$ which power of the error is less than ϵ which means that we can estimate $\mathbf{h}(\mathbf{x})$ based on limited number of elements:

$$\mathbf{h}(\mathbf{x}) \approx \sum_{k_1=0}^{l_1} \cdots \sum_{k_m=0}^{l_m} \mathbf{a_k} \cos(2\pi \mathbf{k} \cdot \mathbf{x}) + \mathbf{b_k} \sin(2\pi \mathbf{k} \cdot \mathbf{x}) \tag{4.4}$$

which is normally called truncated Fourier Series.

4.2 Deep Fourier Machines (DFM)

If we want to use Fourier Series as a signal compression or generation tool, the main problem with it is that if the original signal has large high-frequency components $l_1, \dots, l_m$ in equation 4.4 would be large numbers which means we need to save many $\mathbf{a_k}$'s and $\mathbf{b_k}$'s to preserve a decent copy of the signal. Our solution to this problem is to cascade many low-frequency $\mathbf{h}$'s and hope that this composition of functions would reconstruct important frequencies better, whether they are high or low. Note that here having a low-frequency signal is equivalent to small l_i's in equation 4.4 which is analogous to small number of $\mathbf{a_k}$'s and $\mathbf{b_k}$'s.

4.2.1 Image Compression

In the first step, we will show that it is possible to compress images in a network that consists of a composition of many low-frequency sine and cosine functions in a relatively low dimensional space. This network is trained on just one image. It treats each image as a 2-dimensional function. It gets the position of each pixel of an image and outputs the corresponding pixel value —either a scalar for a black-and-white image or an RGB value for a colour image. After the training process is done, the weights of the network are a compressed copy of the input image, which means that instead of the image itself, we can save the network and use it to reconstruct the image. Also, not that reconstruction is fast and can be done in parallel for each pixel. To reconstruct the image, we give the position of each pixel as input of the model and output is the value of that pixel. Specifically, we can show the input-output relationship of the layer i, $\mathbf{h}_i(\mathbf{x})$, of this network as follows:

$$\mathbf{y} = \mathbf{h}_i(\mathbf{x}) = \sum_{\mathbf{k}} \mathbf{a}_{i,\mathbf{k}} \cos(2\pi \mathbf{k} \cdot \mathbf{x}) + \sum_{\mathbf{k}} \mathbf{b}_{i,\mathbf{k}} \sin(2\pi \mathbf{k} \cdot \mathbf{x}) + \mathbf{x} \tag{4.5}$$

where $\mathbf{h}_i : \mathbb{R}^m \to \mathbb{R}^n$ represents i-th layer of the network from m to n dimensions, $\mathbf{x} = (x_1, x_2, \ldots, x_m) \in \mathbb{R}^m$ is an m-dimensional input, $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$ is an n-dimensional output, $\mathbf{k} \in (\mathbb{N} \cup \{0\})^m$ is all the possible frequencies in a 1×1 image (without considering quantization and discreteness effects), and $\mathbf{a}_{i,\mathbf{k}}, \mathbf{b}_{i,\mathbf{k}} \in \mathbb{R}^n$ are learnable parameters of the layer i. Note that we can always rescale any images to 1×1 unit square. In order to explore low-dimensional structures, we try to make this construction depend on a small number of free parameters: First, the construction relies on composing a small number of low-frequency

function. Second, each low-frequency depends on a few free Fourier coefficients. In other word, $\mathbf{h}_i$ should be low-frequency function, so in our experiments we choose $\mathbf{k} \in \{0,1\}^m$, which means that we have 2^m different $\mathbf{k}$'s in each layer. For example, in the first layer of the network, $\mathbf{h}_1$, because input is the position of each pixel in a image, (x_1, x_2) where $x_1, x_2 \in [0,1]$, dimension of the input, m, is equal to 2 so we have 4 different $\mathbf{k}$'s: $\mathbf{k}_1 = (0,0)$ constant value, $\mathbf{k}_2 = (1,0)$ one period of a sinusoidal function starting from left side of the image to right side, $\mathbf{k}_3 = (0,1)$ one period of a sinusoidal function starting from upper side of the image to lower side, and $\mathbf{k}_3 = (1,1)$ two periods of a sinusoidal function starting from one edge of the image to another edge. It can be shown that the composition of two functions with maximum frequency of $||\mathbf{f}||$ can have up to $||\mathbf{f}||^2$ frequencies. We keep only a small number of $\mathbf{k}$'s in hope of getting higher frequencies of the composition of these layers. Also, we use the idea proposed in [53] to mitigate the vanishing gradient problem and add input $\mathbf{x}$ to the output whenever the dimension of the input and output is the same.

We compose these layers and randomly initialize $\mathbf{a_k}$'s and $\mathbf{b_k}$'s of each layer from a Gaussian distribution:

$$\mathbf{h} = \mathbf{h}_d \circ \mathbf{h}_{d-1} \circ \cdots \circ \mathbf{h}_1 \tag{4.6}$$

Again, note that the input of this network is the position of each pixel (x_1, x_2), and its output is a scalar value for black-and-white images and an RGB value, (r, g, b), for colour images and loss function is a mean square error between real image and constructed one using our

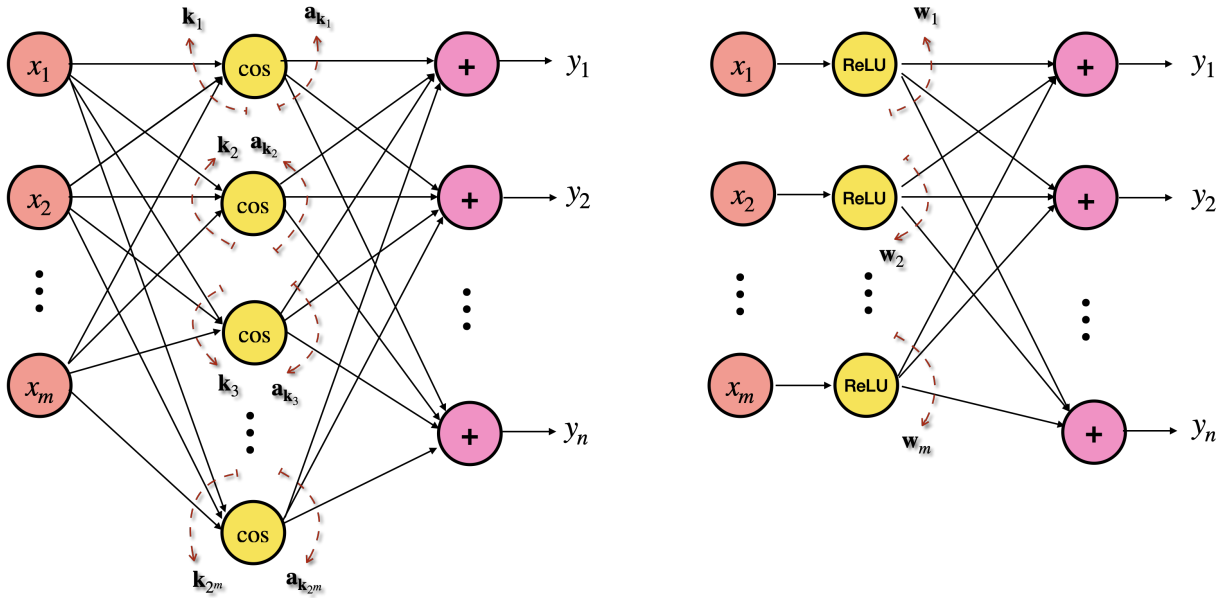


Figure 4.1: The cos part of one layer of our model compare to one layer of Perceptron. Note that sin's and the residual connections are not shown here. The only differences for the sin are that noo-linearity is the sin function and name of the trainable weights are $\mathbf{b}_{\mathbf{k}_i}$. Also, in Perceptron model, ReLU non-linearity is applied before the weight multiplication to make it easier to compare it with our model. Note that $\mathbf{k}_i$'s unlike $\mathbf{w}_i$'s and $\mathbf{a}_{\mathbf{k}_i}$'s are constants, not learnable weights. Here, in our model: $\mathbf{y} = \sum_{\mathbf{k}} \mathbf{a}_{\mathbf{k}} \cos(2\pi \mathbf{k} \cdot \mathbf{x})$ and in Perceptron: $\mathbf{y} = \sum_i \mathbf{w}_i ReLU(x_i)$

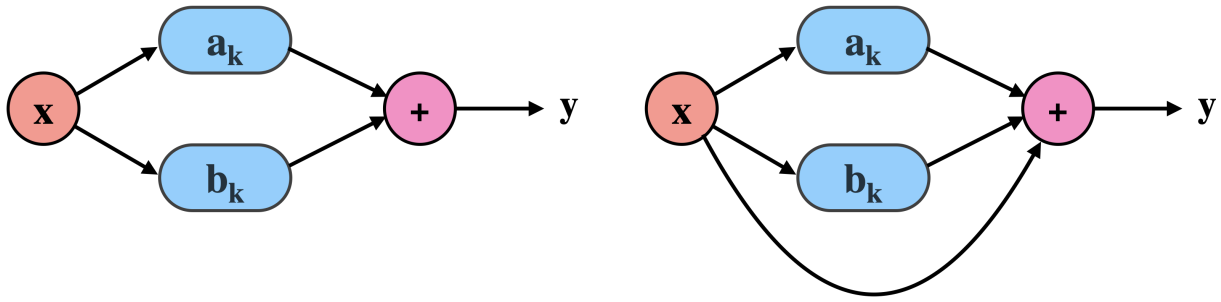


Figure 4.2: High-level overview of a layer of the Model. Our network without (left image) and with (right image) the residual connection. Unless input and output have different dimensions, we use the residual layer. $\mathbf{a_k}$'s and $\mathbf{b_k}$'s are the coefficients of the cos and sin functions, respectfully.

network.

$$\mathbf{MSE} = \frac{1}{N}\sum_{i=0}^{N}\|\mathbf{y}_i - \hat{\mathbf{y}}_i\|^2 \tag{4.7}$$

where $\mathbf{y}_i$ is the original value of i-th pixel of the image, $\hat{\mathbf{y}}_i$ is the predicted value of i-th pixel of the image and N is the number of pixels of the image.

To fully use the residual blocks, except for the dimension of the input and output of the network, which are two and three—for colour images—respectively, we keep the dimension of the middle layers the same and call it the width of our network. Furthermore, the number of layers of the network would be called depth. We pass the output of the model from a tanh function between -1 and 1. We also normalize pixel values between these numbers as the target value. We train multiple networks with different depths and widths to evaluate the power of our model to compress images. After training these networks, we found that images can be compressed by up to 85% with a small reconstruction error.

Algorithm 1 The forward propagation of image compression algorithm

Input: Index of each pixel of an image

Output: Value of each pixel of an image

1: **for** each batch of pixels **do**

2: $pixel\_value \leftarrow \mathbf{h}(pixel\_index)$

3: **end for**

4: **return** $pixel\_value$'s

4.2.2 Image Generation

After we found out that it is possible to reduce the dimension of images with this network without losing much quality, we tried to use it in an image generation model. Note that the main idea of many image generation models is to compress high-dimensional images into low-dimensional spaces. In our generation model, we try to further explore the lower-dimensional structure in natural images. We use another deep neural network as a decoder to generate all the above Fourier coefficients, i.e., $\mathbf{a_k}$'s and $\mathbf{b_k}$'s, from an even smaller set of parameters in a low-dimensional latent space. Different images that have different embeddings will have different decoder outputs, which would lead to different reconstructed images. If the latent space is regulated enough, any point there could potentially be mapped to a realistic image, and after the model is trained, we can use any point in the low-dimensional latent space to generate a new image in the original high-dimensional space. We use an MLP as our decoder, which means we start from the latent representation of images, which for now is

just a learnable vector in about 120-dimensional space. Then we feed each of them to an MLP, which sends the representation to a higher and higher dimension in each layer. The output of this decoder will have the same dimension as the coefficient of a compressed image and will be used as the Fourier Net's $\mathbf{a_k}$ and $\mathbf{b_k}$. Then this network gets the position of all pixels as an input and is supposed to predict the value of each pixel in the corresponding image. Again, we use square error as the loss of our model. We also add the log-likelihood of each latent representation coming from a Gaussian distribution time by a constant—which is a hyperparameter—to the loss function to make the latent space normal. This term simply adds $\left\|\mathbf{z}^2\right\|$ to the loss function, causing the network to keep latent representations as close to the origin of space as possible.

$$\textbf{Loss} = \frac{1}{MN}\sum_{i=0}^{M}\sum_{j=0}^{N}\left\|\mathbf{y}_{i,j} - \hat{\mathbf{y}}_{i,j}\right\|^2 + \frac{\lambda}{M}\sum_{i=0}^{M}\|\mathbf{z}_i\|^2 \tag{4.8}$$

where $\mathbf{y}_{i,j}$ is the original value of j-th pixel of i-th image of dataset, $\hat{\mathbf{y}}_{i,j}$ is the predicted value of j-th pixel of i-th image of dataset, $\mathbf{z}_i$ is the latent representation of i-th image, M is the number of images of the dataset, N is the number of pixels of each image and λ is a hyperparameter.

Algorithm 2 The forward propagation of image generation algorithm w/o encoder

Input: Index of each pixel of an image and index of all images

Output: Value of each pixel of all images

for each batch of images **do**
    find the corresponding embedding $\mathbf{z}$ for each index of the image of batch from a table
    $\mathbf{A}, \mathbf{B} \leftarrow \mathbf{Decoder}(\mathbf{z})$
    for each batch of pixels **do**
        $pixel\_value \leftarrow \mathbf{h}_{\mathbf{A},\mathbf{B}}(pixel\_index)$
    end for
end for
return *pixel_value*'s

In the training process, we found out that large networks are unstable to train. Our solution to this problem was to add an encoder before the latent representation. Instead of randomly initializing $\mathbf{z}$, we pass each image to an encoder then we use the output as the latent representation of the image. Using this structure, not only large networks are stabilized but also similar images have similar latent representations even at the beginning of training, which speeds up the training process. Similar to the decoder, we used an MLP as the encoder of our model.

Algorithm 3 The forward propagation of image generation algorithm with encoder

Input: Index and value of each pixel of all image

Output: Value of each pixel of all images

for each batch of images **do**
    $\mathbf{z} \leftarrow \mathbf{Encoder}(image)$
    $\mathbf{A}, \mathbf{B} \leftarrow \mathbf{Decoder}(\mathbf{z})$
    for each batch of pixels **do**
        $pixel\_value \leftarrow \mathbf{h}_{\mathbf{A},\mathbf{B}}(pixel\_index)$
    end for
end for
return $pixel\_value$'s

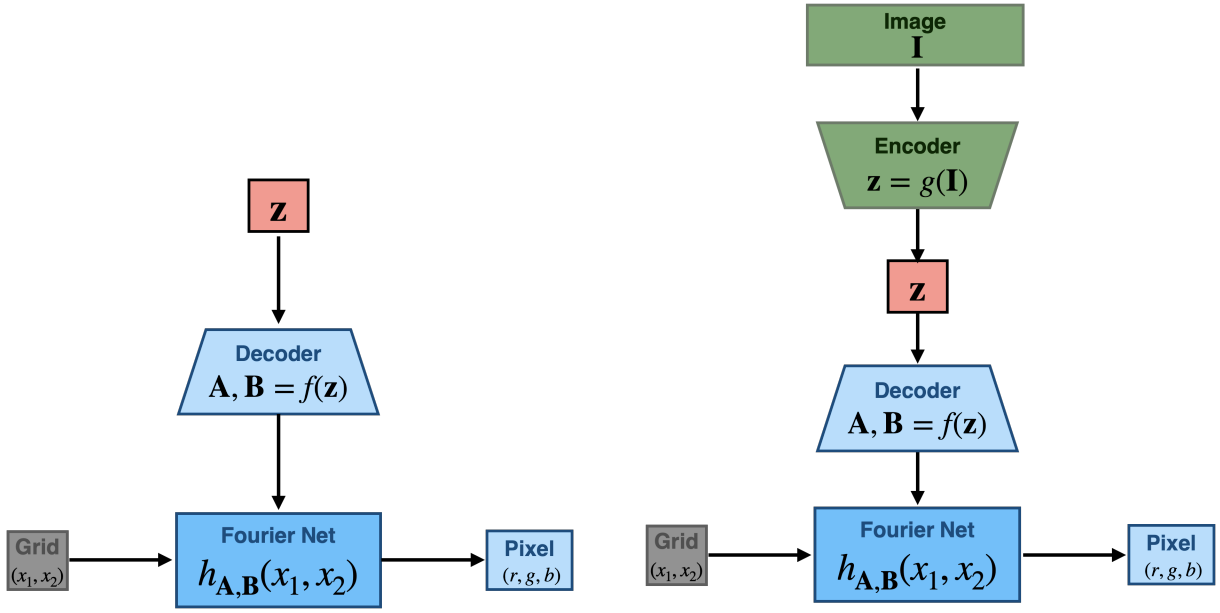


Figure 4.3: Our Model without (left image) and with (right image) an encoder. $\mathbf{z}$ is the latent representation of each image, $\mathbf{A}$ and $\mathbf{B}$ are the set of all $\mathbf{a_k}$'s and $\mathbf{b_k}$'s of each image, (x_1, x_2) is the input position of each pixel, and (r, g, b) is output of network for each pixel of each image.

Chapter 5

Experiments

5.1 Experimental Setup

5.1.1 Servers

The servers that are used in this study have been provided by Laboratory for Neural Computing for Machine Learning. Please refer to table 5.1 for more details. As our model was the first of its kind, we spend hundreds of hours investigating different structures and tuning the hyperparameters.

The experiments are all conducted with Python version 3.6.13, JAX [54] version 0.2.13 utilized with CUDA 11.1. JAX is a framework that is designed by Google for high-performance numerical computing, especially machine learning research. We have designed the model layer by layer using this library and used its tools for high-speed computations.

Server List			
System	**CPU**	**Memory**	**GPUs**
Data	10-core CPU (i9-7900X)	128GB	4 × GTX1080 Ti, 11 GB memory
Knowledge	10-core CPU (i9-7900X)	128GB	4 × GTX1080 Ti, 11 GB memory
Text	6-core CPU (i7-5820K)	64GB	4 × TITAN X, 12GB memory

Table 5.1: List of systems used for the experiments

5.1.2 Datasets

We used three datasets in this work, MNIST [55], CIFAR10 [56] and ImageNet [57]. MNIST contains 70000, 28×28 black-and-white images of handwritten digits between 0 and 9. 60000 images are gathered as the training set and the remaining are test set but because of the nature of unsupervised tasks, we use the whole dataset to train our generative model. The objective of our algorithm is to generate realistic synthetic handwritten digits. The second dataset, CIFAR10, contains 60000, 32×32 colour images, 50000 as the training set and 10000 as the test set, of 10 different classes. We trained our generative model on this dataset to output images from the same classes. Finally, we use ImageNet [57] to show the ability of our model to compress separate images. ImageNet is a huge dataset containing 1000 classes. We only use a very limited number of images to train our model.

5.1.3 Optimization and Hyperparameters Settings

We trained the model using the Adam optimizer both for image compression and image generation tasks. We used a constant learning rate equal to 4×10^{-4} for the image compression task and a linear schedule starting from 10^{-3} and ending with 10^{-7} for the decoder of the image generation task. On the MNIST dataset, where the model does not have an encoder, we set the learning rate for the latent representation 10^6 times smaller than the decoder learning rate; whereas, on the CIFAR10 dataset, the encoder learning rate is from 10 to 1000 times smaller than the decoder, a linear schedule through the whole training process. The batch size for the compression task is set to 2^{14} where each element of the batch is a pixel. For the generation tasks on MNIST and CIFAR10, the batch size is set to 64 and 512, respectively, where each element is an image. All of the compression task's learnable weights are drawn from a Gaussian distribution with a variance of 2×10^{-10}. The same initialization has been used in the generation task with 2×10^{-2} and 3×10^{-2} respectively on MNIST and CIFAR10, except for bias weights, which are initialized to 0. On the MNIST and CIFAR10 datasets, we set the constant hyperparameter, which is multiplied by the log-likelihood of each latent representation derived from a Gaussian distribution, $\left\|\mathbf{z}^2\right\|$, to 0.5 and 200, respectively.

5.2 Models Evaluation

5.2.1 Image compression on ImageNet

We use a 400×400 image from the ImageNet dataset [57] to train our network five times and get the average final loss on these five networks. Then we show this averaged final loss in figure 5.1 using four different colours. The darker the colour, the better the image is preserved. Also, the number in each box of the table shows the percentage of the size of the compressed image. For example, 100 indicates that the network is the same size as the training image, and 1 indicates that it is $\frac{1}{100}$ the size of the image. To get a better sense of the colour of each box, we also display the corresponding image to each loss interval in figure 5.2. In this experiment we used 32-bit floating point numbers, which means the number in each box can be calculated as below:

$$\frac{\#\ of\ learnable\ weights\ of\ the\ net}{400 \times 400 \times 3} \times \frac{32}{8} \times 100$$

where 400 is the height and length of the original image, 3 is the number of channels in a colour image (RGB), 32 is the number of bits required to represent a 32-bit float, and 8 is the number of bits required to represent one channel of a pixel—a number between 0 and 255.

We also trained one specific network with about 82% compression rate — compressed image is 18% of the original image — on other images of the ImageNet dataset to find out how our approach can generalize on other images. Also, the same images are compressed with almost the same compression rate using the JPEG algorithm to compare them with our

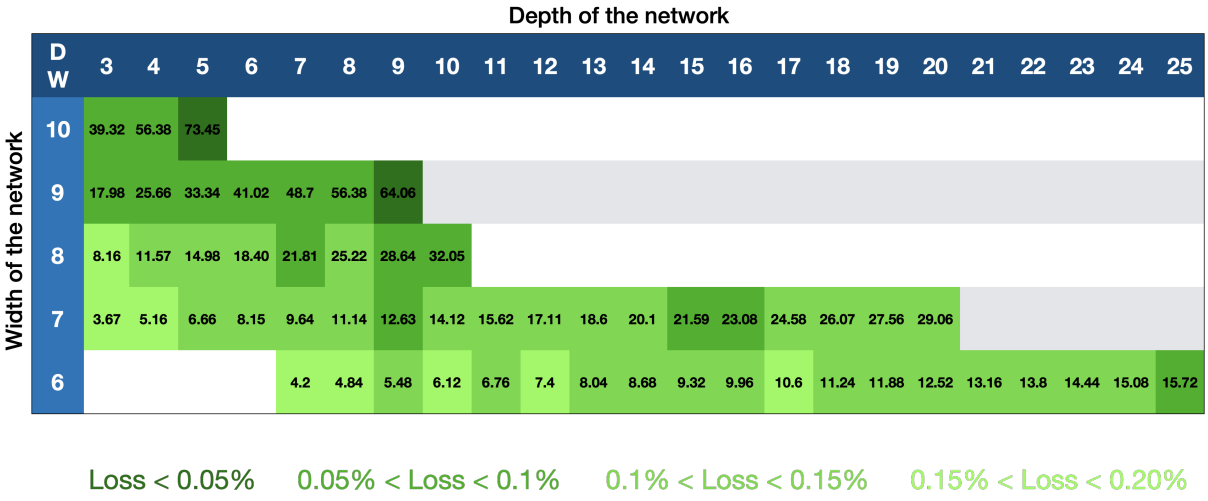


Figure 5.1: Percent size of the network compare to the original image and corresponding loss interval.

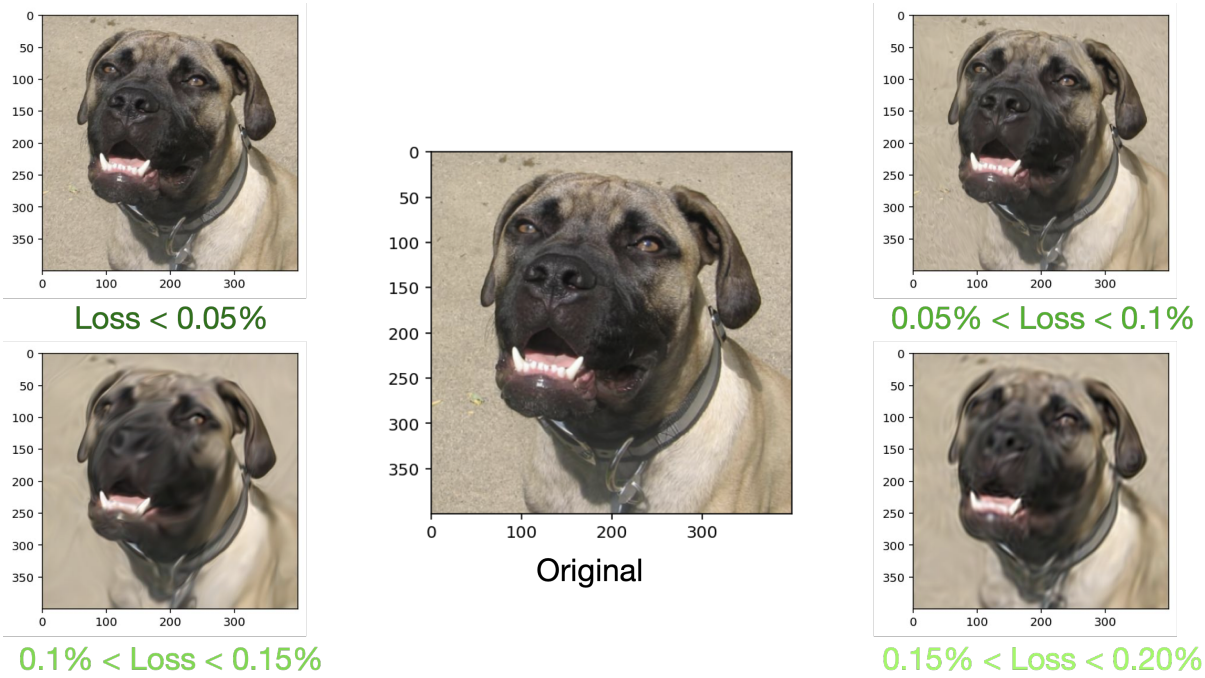


Figure 5.2: Compressing the same image with different final loss. Use this figure to get a better sense how a sample image would look like in the table 5.1.

model's output. Comparing results presented in figure 5.3, our network can compress images as well as the JPEG algorithm and even outperform it mostly in cases where the original image is low-frequency dominant, for example, the speaker image. The only disadvantage of our model compared to the JPEG algorithm is its training time, which is a lot longer than the compression time of the JPEG algorithm.

5.2.2 Image generation on MNIST

We used the MNIST dataset to train our model so that it embeds images in a lower-dimensional space and uses that space to generate fake images. We trained the model both with just one class—to generate one specific fake number—and with all ten classes. The width and depth of the Fourier Net we used were 4 and 6, respectively. The latent space size is 55, and the decoder is a fully connected network with a size of (55, 170, 220, 352), where 352 is the number of learnable weights of the Fourier Net.

In figure 5.4, we present reconstructed images from the dataset. We can see that after embedding images into the latent space, they can be reconstructed almost perfectly via our network. In figure 5.5, our networks are trained on one class of the MNIST dataset to generate fake numbers from the same class. In figure 5.6, we trained the same network on all images of the MNIST dataset and sampled random images from the network. In figure 5.7, we try to explore latent space by moving from one embedded image to another to generate images that are similar to both. In some rows, for instance, the interpolation between 7 and 8, the network generates images which are similar to other numbers—in this case, 9. If this

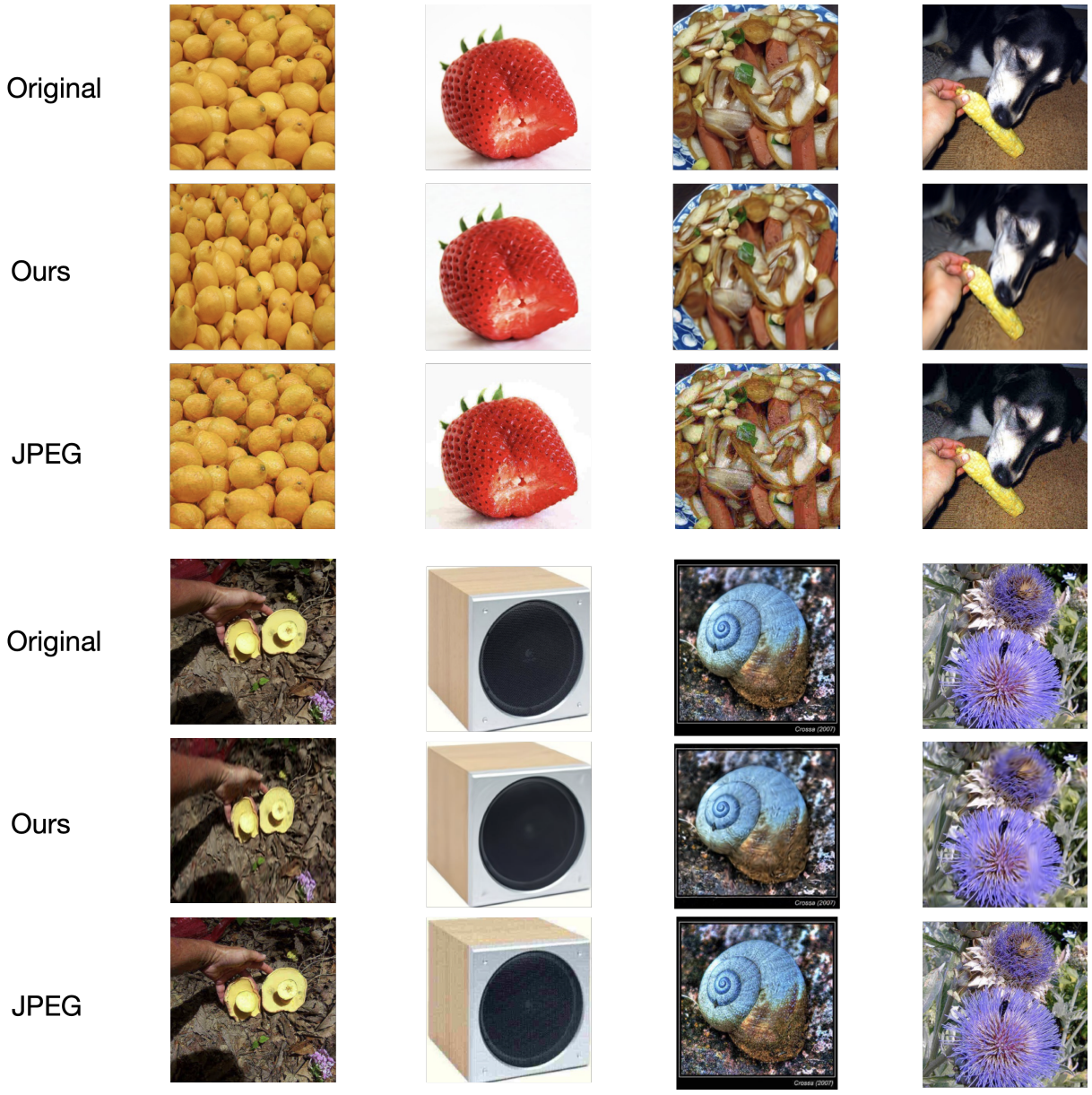


Figure 5.3: Comparison between our model and JPEG algorithm. The width and depth of our model in this experiment are 8 and 6 respectively. A lossy JPEG algorithm with almost the same compression rate has been used to compress the original image.

number is visually between the start and end numbers, it is a sign that the latent space is well-regulated and similar numbers are being embedded near each other.

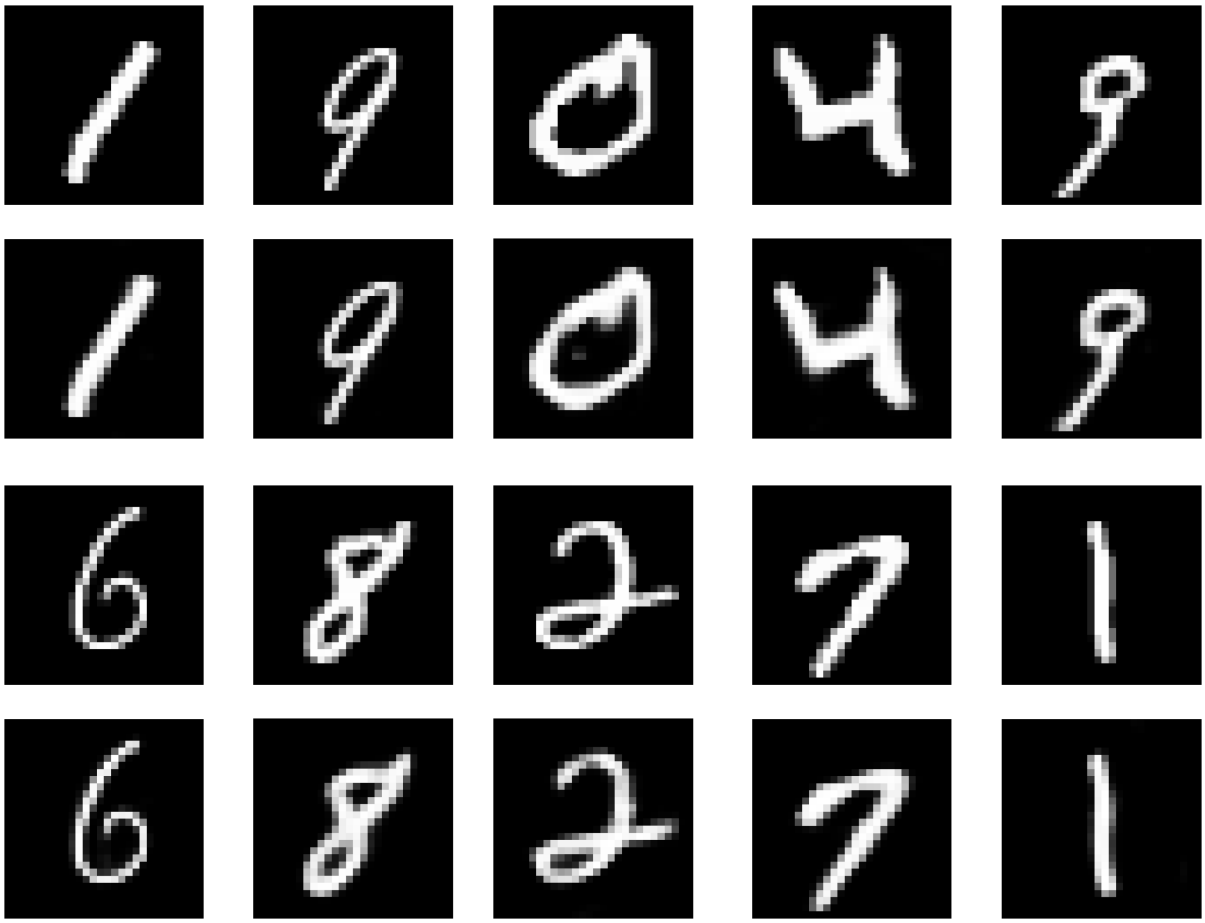


Figure 5.4: Original images and their reconstructed version using the our model trained on all MNIST dataset. We can see that images has been embedded in the latent space almost prefectly which means that square loss of the training is close to zero - excluding log-likelihood part.

(a)

(b)

Figure 5.5: Random images generated using our model trained on (a) number 2 and (b) number 7 images of that dataset.

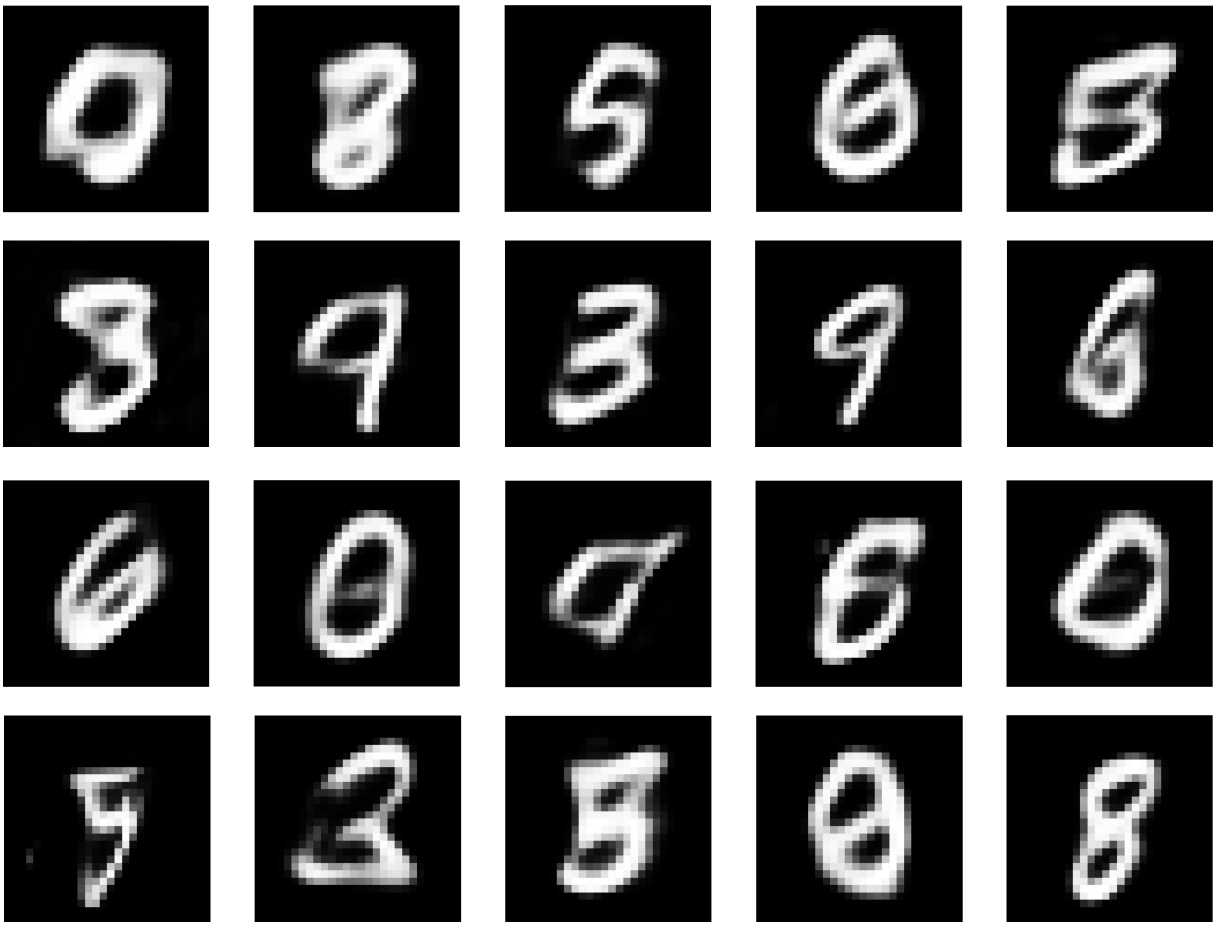


Figure 5.6: Random images sampled from our model trained on full MNIST dataset.

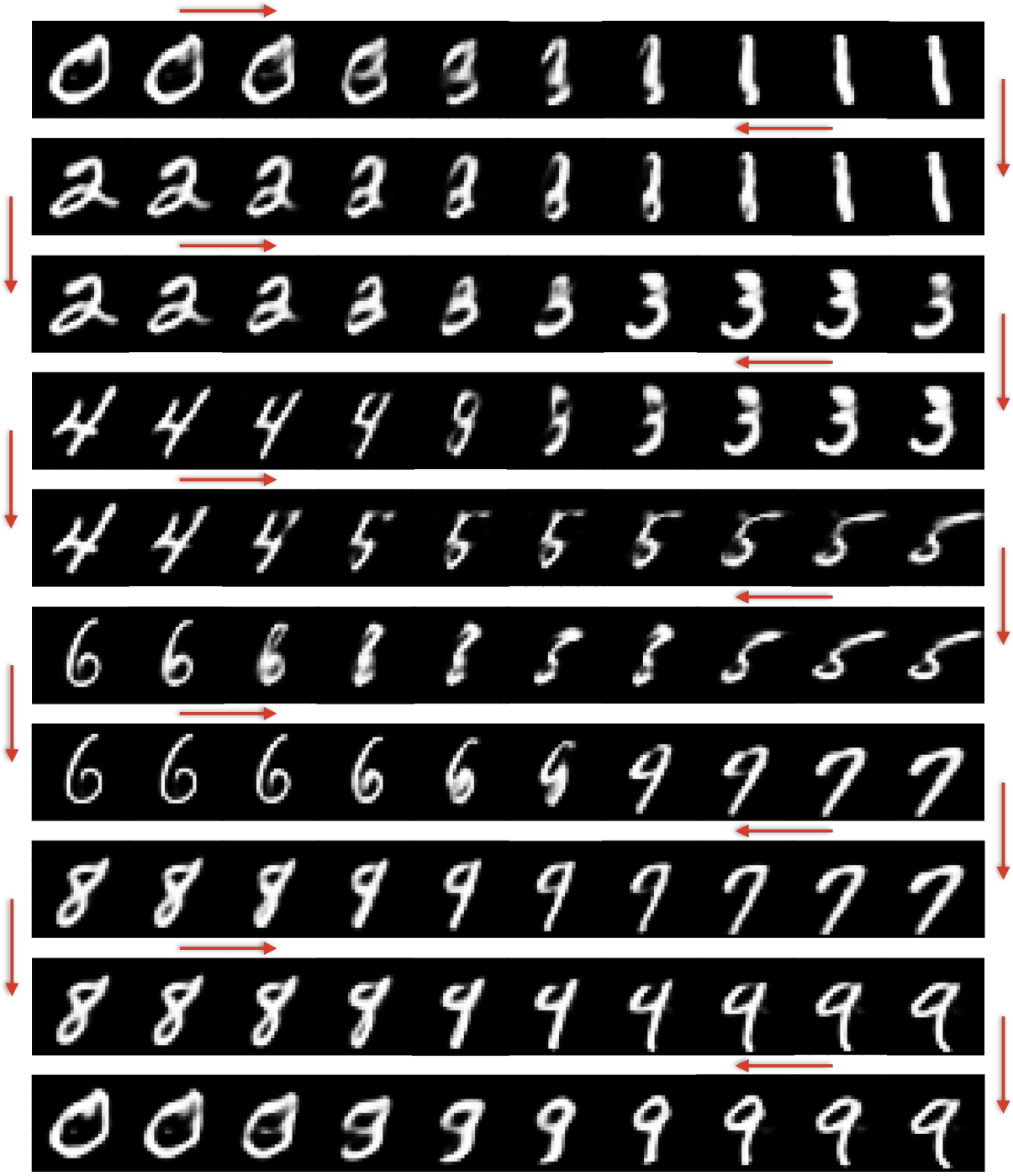


Figure 5.7: Interpolation between different integers from 0 to 9. We start from one image latent representation and as we go toward the next one in the latent space, we feed that to the decoder and get the output from Fourier Net.

5.2.3 Image generation on CIFAR10

We trained our model on the CIFAR10 dataset to measure its capability to generate colour images. Because the FID score uses the Inception v3 network trained on the ImageNet dataset which contains colour images, we can get quantitative results from this experiment. Throughout this experiment, the width and depth of the Fourier Net were 6 and 10 respectively, the dimension of the latent space was 110, and the encoder and decoder were (3072, 1200, 600, 110) and (110, 500, 1000, 3672) fully connected networks, where 3072 is the number of RGB pixels in the CIFAR10 dataset and 3672 is the number of learnable weights of the Fourier Net.

Figure 5.8 shows the quality of images after reconstructing them using the trained model. It is worth mentioning that the best model with respect to the quality of generated images is not the same model that can reconstruct the training images the best. We can achieve better image reconstructions by training larger networks, but their generated images will not have the same quality and FID score as the presented outputs here. As our goal is to generate fake images, we present reconstructed images of our best model in the image generation task. In figure 5.9, we sample fake images from the network. In table 5.2, we compare the FID score of our model with the first generation of different image generative models. Note that DCGAN [15] is a second-generation GAN model that utilizes CNN layers, and unlike our model, evaluating the probability density function is not possible there. Also, although the FID score is probably the most popular metric to compare all generative models, it has many shortcomings, and it is better to not rank generative models only based on it.

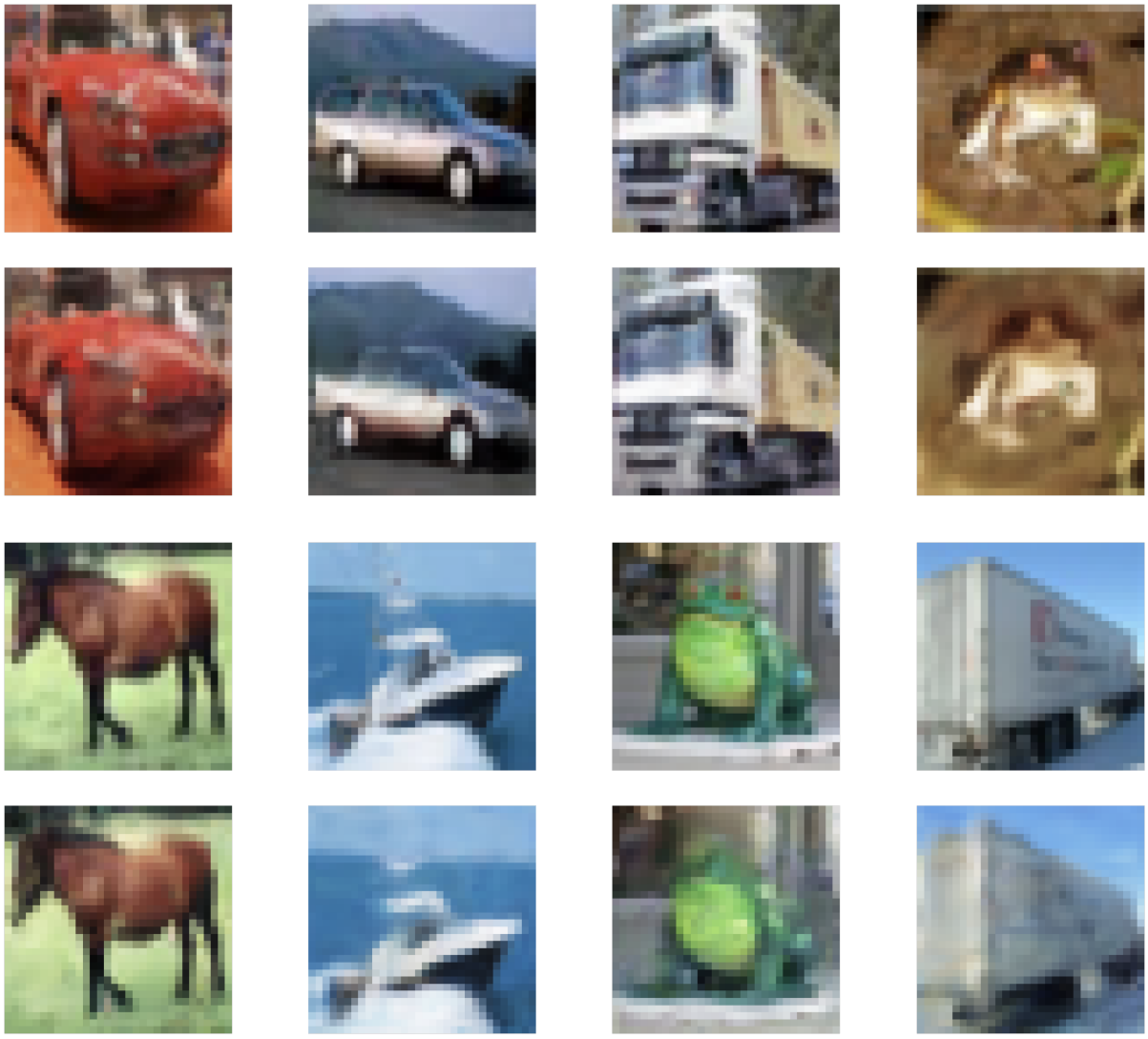


Figure 5.8: Original images and their reconstructed version using the our model trained on CIFAR10 dataset. Images have been embedded in the latent space with a small quality loss.

Figure 5.9: Random images sampled from our model trained on CIFAR10 dataset.

Model	**FID**
VAE [22]	106.0
WAE [58]	80.9
DCGAN [15]	30.9
Fourier Net (ours)	**80.5**

Table 5.2: FID comparison between different models trained on CIFAR10 dataset

Chapter 6

Conclusion

6.1 Summary

In this work, we began by discussing deep learning models, their success factors, and their fundamental components. We present some complex generative model architectures. We discussed the fundamental mathematical idea that underlies them, how they are intended to function intuitively, and their advantages and disadvantages. Then we provided a brief overview of the frequency domain. We gave an overview of Variational Autoencoder, Generative Adversarial Networks, Normalizing Flows, and Diffusion Models and discussed their advantages and disadvantages. Then, to overcome these issues, we proposed our network, which is capable of compressing images using the frequency domain.

On the MNIST and CIFAR-10 datasets, our model produces acceptable fake images. Images have been encoded in the latent space and can be reconstructed well, and the whole

latent space is regulated enough to generate high-quality images.

6.2 Future Works

The future steps for the current research are listed as follow:

- Using 16-bit float on image compression task. We used 32-bit floating point numbers for the whole experiment. Trying 16-bit float will probably give better results on the image compression task but since here our main focus was on the image generation task, we present it as future work.

- Using the Gaussian Mixture Model (GMM) instead of a Gaussian Distribution as the log-likelihood of latent spaces. When there are multiple classes in the dataset, GMM usually works better as the distribution of the likelihood in image generation tasks.

- Increasing the training speed (decreasing epoch). We tried many modules and tricks to speed up the training process but most of them were unsuccessful. Batch norm, Layer norm and dropout were the most famous ones.

- Training this model on larger datasets. We need to evaluate this model on larger image datasets.

Bibliography

[1] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10, Haifa, Israel: Omnipress, 2010, pp. 807–814, ISBN: 9781605589077.

[2] L. Weng, "What are diffusion models?" *lilianweng.github.io*, 2021. [Online]. Available: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[4] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[5] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proceedings of the 32nd International Conference on International Conference on*

Machine Learning - Volume 37, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 1530–1538.

[6] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, Jul. 2015, pp. 2256–2265.

[7] D. Kingma, T. Salimans, B. Poole, and J. Ho, "Variational diffusion models," *Advances in neural information processing systems*, vol. 34, pp. 21 696–21 707, 2021.

[8] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016. DOI: 10.1109/cvpr.2016.308. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2016.308.

[10] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, *Gans trained by a two time-scale update rule converge to a local nash equilibrium*, 2017. arXiv: 1706.08500 [cs.LG].

[11] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, 2014. arXiv: 1411.1784.

[12] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *International conference on machine learning*, PMLR, 2017, pp. 2642–2651.

[13] A. Odena, "Semi-supervised learning with generative adversarial networks," *arXiv preprint arXiv:1606.01583*, 2016.

[14] E. L. Denton, S. Chintala, R. Fergus, *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," *Advances in neural information processing systems*, vol. 28, 2015.

[15] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[16] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, PMLR, 2017, pp. 214–223.

[17] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, *Spectral normalization for generative adversarial networks*, 2018. arXiv: `1802.05957 [cs.LG]`.

[18] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *International conference on machine learning*, PMLR, 2019, pp. 7354–7363.

[19] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[20] T. Karras, S. Laine, and T. Aila, "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.

[21] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "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.

[22] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *International conference on machine learning*, PMLR, 2014, pp. 1278–1286.

[23] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," *Advances in neural information processing systems*, vol. 28, 2015.

[24] I. Higgins, L. Matthey, A. Pal, *et al.*, "Beta-VAE: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations*, 2017. [Online]. Available: `https://openreview.net/forum?id=Sy2fzU9gl`.

[25] A. Alemi, B. Poole, I. Fischer, J. Dillon, R. A. Saurous, and K. Murphy, "Fixing a broken elbo," in *International Conference on Machine Learning*, PMLR, 2018, pp. 159–168.

[26] J. Tomczak and M. Welling, "Vae with a vampprior," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2018, pp. 1214–1223.

[27] A. van den Oord, O. Vinyals, and k. kavukcuoglu koray, "Neural discrete representation learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017.

[28] A. Vahdat and J. Kautz, "Nvae: A deep hierarchical variational autoencoder," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 667–19 679, 2020.

[29] R. Child, "Very deep vaes generalize autoregressive models and can outperform them on images," *arXiv preprint arXiv:2011.10650*, 2020.

[30] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional image generation with pixelcnn decoders," *Advances in neural information processing systems*, vol. 29, 2016.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123.

[32] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[33] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=HkpbnH9lx.

[34] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference*

on International Conference on Machine Learning - Volume 37, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 448–456.

[35] C.-W. Huang, S. Tan, A. Lacoste, and A. Courville, "Improving explorability in variational inference with annealed variational objectives," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, Montréal, Canada: Curran Associates Inc., 2018, pp. 9724–9734.

[36] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018. [Online]. Available: `https://proceedings.neurips.cc/paper/2018/file/d139db6a236200b21cc7f752979132d0-Paper.pdf`.

[37] J. M. Tomczak and M. Welling, "Improving variational auto-encoders using householder flow," *arXiv preprint arXiv:1611.09630*, 2016.

[38] E. Hoogeboom, R. van den Berg, and M. Welling, "Emerging convolutions for generative normalizing flows," in *International conference on machine learning*, 2019. [Online]. Available: `https://arxiv.org/abs/1901.11137`.

[39] L. Dinh, D. Krueger, and Y. Bengio, "NICE: non-linear independent components estimation," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: `http://arxiv.org/abs/1410.8516`.

[40] D. P. Kingma, T. Salimans, and M. Welling, "Improving variational inference with inverse autoregressive flow," *CoRR*, vol. abs/1606.04934, 2016. arXiv: 1606.04934. [Online]. Available: http://arxiv.org/abs/1606.04934.

[41] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, *et al.*, Eds., 2017, pp. 2214–2224. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/f9be311e65d81a9ad8150a60844bb94c-Abstract.html.

[42] J.-H. Jacobsen, A. W. Smeulders, and E. Oyallon, "I-revnet: Deep invertible networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=HJsjkMb0Z.

[43] J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen, "Invertible residual networks," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Sep. 2019, pp. 573–582. [Online]. Available: https://proceedings.mlr.press/v97/behrmann19a.html.

[44] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31,

Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

[45] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "Ffjord: Free-form continuous dynamics for scalable reversible generative models," *International Conference on Learning Representations*, 2019.

[46] E. Dupont, A. Doucet, and Y. W. Teh, "Augmented neural odes," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf.

[47] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Flow++: Improving flow-based generative models with variational dequantization and architecture design," in *International Conference on Machine Learning*, PMLR, 2019, pp. 2722–2730.

[48] R. T. Chen, J. Behrmann, D. K. Duvenaud, and J.-H. Jacobsen, "Residual flows for invertible generative modeling," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[49] D. Nielsen, P. Jaini, E. Hoogeboom, O. Winther, and M. Welling, "Survae flows: Surjections to bridge the gap between vaes and flows," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 685–12 696, 2020.

[50] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 18–24 Jul 2021, pp. 8162–8171.

[51] R. Gao, Y. Song, B. Poole, Y. N. Wu, and D. P. Kingma, "Learning energy-based models by diffusion recovery likelihood," *arXiv preprint arXiv:2012.08125*, 2020.

[52] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8780–8794, 2021.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[54] J. Bradbury, R. Frostig, P. Hawkins, *et al.*, *JAX: Composable transformations of Python+NumPy programs*, version 0.3.13, 2018. [Online]. Available: `http://github.com/google/jax`.

[55] Y. LeCun and C. Cortes, "The mnist database of handwritten digits," 1998.

[56] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[57] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[58] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, "Wasserstein auto-encoders," *arXiv preprint arXiv:1711.01558*, 2017.