AUTOMATIC IMAGE RECOGNITION OF RAPID MALARIA EMERGENCY DIAGNOSIS: A DEEP NEURAL NETWORK APPROACH

ZHAOHUI LIANG

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND TECHNOLOGY

YORK UNIVERSITY

TORONTO, ONTARIO

June 2017

Abstract

Deep learning is the state-of-the-art artificial intelligence (AI) method for visual pattern detection and automated diagnosis. This paper describes the application of convolutional neural network (CNN), the deep learning model for visual recognition, to automatic detection of plasmodium parasitized red blood cells for malaria field screening and rapid diagnosis. The malaria thin blood smears are from Bangladesh and initially labeled by a specialist. 27,578 red blood cell images are segmented (raw set). The images are rotated clockwise three times to generate an augmented dataset with 110,312 red blood cell images. A 12-layer and an 18-layer CNN-based Malaria Net models are applied to classify both the raw data set and the augmented dataset. The performance is evaluated by ten-fold cross-validation and compared to a transfer learning model. In the ten-fold cross-validation test for Malaria Net, the average accuracy is 97.37% (18-layer) and 96.09% (12-layer) with the raw set, and is 97.93% and 96.75% with the augmented set, in comparison to 91.99% with the raw set and 94.26% with the augmented set in transfer learning. In addition, the two CNN models show superiority over transfer learning in all performance indicators such as sensitivity, specificity, precision, F1 score, and Matthews correlation coefficient. The Malaria Net can accurately detect malaria-infected red blood cells. A CNN model trained by domain-specific data shows superior performance over the transfer-learning method. Automatic image classification powered by deep learning offers not only an accurate method for the malaria field screening and rapid diagnosis but also a new solution for malaria control especially in resource-poor regions.

Acknowledgements

I would like to express my gratitude to Prof. Jimmy Huang, my supervisor of my graduate study in York University, for providing the most helpful advice to my study in information technology and computer programing, and opportunities to utilize my knowledge and skill in reality. And I would like to thank Dr. George R. Thoma and Dr. Stefan R. Jaeger from Lister Hill National Center for Biomedical Communications (LHNCBC), National Library of Medicine (NLM), National Institute of Health (NIH) for providing me with an excellent opportunity to apply my knowledge in both medicine and information technology to biomedicine and to contribute to the global health and disease control.

I would like to thank my wife for her continuous support for my study in York University and her hard work to care our children when I worked in NIH in the U.S. in the 2016 summer.

This research is funded by a Discovery grant by the Natural Sciences and Engineering Research Council (NSERC) of Canada, a NSERC CREATE award in Advanced Disaster, Emergency and Rapid-response Simulation 1 (ADERSIM 1), and an ORF-RE (Ontario Research Fund - Research Excellence) award in BRAIN (Big Data Research and Analytics Information Network) Alliance 2. The malaria automated diagnosis project is funded by the US HHS (United States Department of Health Human Services) Innovation Ventures Fund Program 2015-2016. The Master candidate is jointly trained by York University and by the US Department of Energy program for future scientist in North America. The deep learning research is also funded by the National Natural Science Foundation of China (No.81573827).

Contents

Abstract **ii**

Acknowledgements **iii**

Contents **iv**

1 Introduction **1**

1.1 Global Control of Malaria 1
1.2 Machine Learning for Automated Diagnosis 1
1.3 Deep Learning for Automated Diagnosis 3
1.4 Research Background 4

2 Literature Review **5**

2.1 Automatic Diagnosis by Machine Learning 5
2.2 Application of Machine Learning to Image Retrieval 8
2.3 Automatic Diagnosis of Malaria and Machine Learning 9
2.4 Deep learning and convolutional neural network 11
2.5 Advantages of CNN Compared to Other Methods 11
2.6 Challenges to CNN Applications 12

3 Our Proposed Method **14**

3.1 General Procedure of Deep Learning 14
3.2 Convolutional Layer 16
3.3 Pooling layer 17
3.4 Activation Layers 18
3.5 Output Layers 20
3.6 Model Optimization and Fine-tuning 21
3.7 CNN Architecture for Malaria Classification 27

4 Model Evaluation **30**

4.1 Data Source 30
4.2 Experimental Environment 31
4.3 Data Preprocessing 31
4.4 CNN Model Training 34
4.5 Performance Evaluation 36

5 Results **38**

5.1 Training Procedure 38
5.2 Confusion Matrices of the Malaria Net Evaluation 38
5.3 The Cross Validations of Different Models 40
5.4 Stability and Robustness of the CNN Classifiers 46

6 Discussion 49
6.1 The Global Burden of Malaria 49
6.2 Application of Deep Learning to Automated Diagnosis 50
6.3 Convolutional Neural Network for Image-based Diagnosis 50
6.4 CNN Applications to Malaria Diagnosis 52
6.5 Expectations of Deep Learning Applications 53

7 Conclusions and Future Work 55
7.1 Conclusions 55
7.2 Future Work 56

Bibliography 59

Appendix A Implementation of Deep Learning in MATLAB 68
A.1 General Description 68
A.2 Install the MatConvNet Toolbox 68
A.3 Compiling the MatConvNet Toolbox 69

Appendix B Preparation of the Image Data Object 71
B.1 Instructions 71
B.2 MATLAB Script to Resize Image 72
B.3 MATLAB Script to Image Transfer and Partition 72

Appendix C Image Data Preprocessing for Deep Learning 76
C.1 Instructions 76
C.2 MATLAB Script to Create the imdb Data Object with All Data Preprocessing Steps 76

Appendix D CNN Model Configuration 82
D.1 Instructions 82
D.2 The 12-layer Malaria Net 83
D.3 The 18-layer Malaria Net 85

Appendix E MATLAB Script for CNN Model Training 89
E.1 Instructions 89
E.2 MATLAB Script to Implement CNN Training 89

Appendix F MATLAB Script to Implement Ten-fold Cross Validation 107
F.1 Instructions 107
F.2 CNN Training with Cross Validation 108

Appendix G MATLAB Script for Model Evaluation 111

G.1 Instructions 111

G.2 MATLAB Script for Model Evaluation 112

Appendix H MATLAB Code for Malaria Net Cross Validation 113

H.1 Description 113

H.2 MATLAB Script for Model Evaluation 113

Appendix I Publications during the MAIST Program Study 115

I.1 Publication in Conference Proceedings 115

I.2 Publication in Journal 116

Chapter 1

Introduction

1.1 Global Control of Malaria

Malaria is one of the most common vulnerable diseases that affects the world population. Malaria is caused by parasites that are transmitted through the bites of infected mosquitoes. The pathogen of malaria is genus plasmodium. It is a kind of parasitic protozoans which can invade the erythrocyte of human and cause a serial of symptoms. According to WHO report, 214 million people were affected by malaria in 2015 with an estimated 438,000 deaths (See Figure 1.1)[1]. The disease burden is estimated up to 12 billion per year [2]. Malaria is a deadly infectious disease that has been present throughout human history. Although effective medications have been developed to fight malaria, it remains a major burden on global health in the 21st century, particularly in tropical and subtropical regions and in the resource poor regions such as Africa, South America, and South and South-eastern Asia. Most deaths caused by malaria occur among the children in Africa.

1.2 Machine Learning for Automated Diagnosis

Malaria can be prevented, effectively controlled, and cured if an accurate and efficient diagnosis method would be available. The standard diagnosis of malaria is to investigate the patients' blood smears under light microscope to count the infected erythrocytes by qualified pathological technicians. This method is inefficient and the quality of the

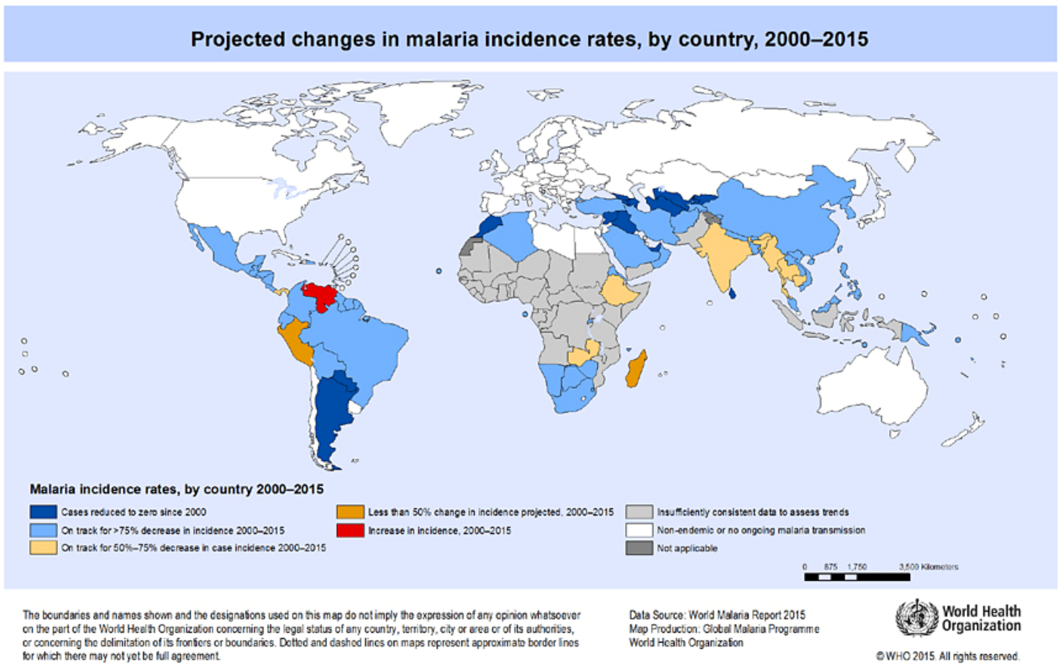

Projected changes in malaria incidence rates, by country, 2000–2015
Malaria incidence rates, by country 2000–2015
Cases reduced to zero since 2000
On track for >75% decrease in incidence 2000–2015
On track for 50%–75% decrease in case incidence 2000–2015
Less than 50% change in incidence projected, 2000–2015
Increase in incidence, 2000–2015
Insufficiently consistent data to assess trends
Non-endemic or no ongoing malaria transmission
Not applicable
0 875 1,750 3,500 Kilometers
The boundaries and names shown and the designations used on this map do not imply the expression of any opinion whatsoever on the part of the World Health Organization concerning the legal status of any country, territory, city or area or of its authorities, or concerning the delimitation of its frontiers or boundaries. Dotted and dashed lines on maps represent approximate border lines for which there may not yet be full agreement.
Data Source: World Malaria Report 2015
Map Production: Global Malaria Programme
World Health Organization
World Health Organization
©WHO 2015. All rights reserved.

Figure 1.1: The global burden of malaria in 2015

diagnosis depends on the experience and knowledge of the medical specialists. The improvement in diagnosis can certainly enhance the global control of malaria. However, this method is labor-intensive and heavily dependent on the experience and knowledge of the microscopists. In addition, it is estimated that a total of 170 million blood smears are examined annually, imposing a large burden on health resources [3]. Automatic image recognition technologies based on machine learning and big data have been applied to malaria blood smears diagnosis since 2005 [4]. An early application of machine learning to blood cell classification is to detect parasitemia in images of Giemsa-stained blood smears. In this early study, the correlation of automatic and manual parasitemia detection was compared, but cells were not classified as infected or uninfected [5].

Depending on the blood smear type, algorithms for automatic recognition of plasmodium-infected erythrocytes can be divided into two categories: classifiers for thick blood smears and classifiers for thin blood smears. Elter et al. proposed a model based on Bayesian pixel-classification which uses a kNN classifier to reduce false-positive detections. The model was evaluated and it has better performance compared to support vector machine (SVM) classifiers [6]. On the other hand, Das et al. reported that thin films of blood smear are more appropriate for automatic image diagnosis because fewer layers of blood cells provide clearer and more reliable patterns. Thus they propose a model using F-statistics and Information gain for feature selection and Naïve Bayes, RBF Neural network, and Logistic regression as the main method for pattern classification. This method is satisfactory for recognizing plasmodium inside the red blood cell, however, its performance will be weakened by overlapping cells [7].

Based on the current research progress, there is no robust model or algorithm available so far for automatic image diagnosis of malaria because the patterns for differentiating infected red blood cells from normal cells are not static. The image characteristics of plasmodium inside the erythrocytes change depending on the stage of infection. In addition, color intensities may vary due to staining variability, which can significantly weaken the reliability of conventional image recognition algorithms.

1.3 Deep Learning for Automated Diagnosis

A deep learning model is a neural network with more than two hidden layers. This feature allows it to learn representations of data with highly-dimensional abstraction. A convolutional neural network (CNN) is a deep learning model suitable for two-dimensional data such as images and videos. CNNs are motivated by the study on the neurophysiological activities in the vision cortex of the brain. In 1959, Hubel and Wiesel discover that a cat recognizes an object by memorizing its edges observed in multiple directions [8]. Based on this finding, one can argue that a pattern recognition model should mimic this information processing mechanism of the brain [9]. The advantage of a CNN model is that its hierarchical structure of learning layers can be trained in a robust manner once the topology of the model fits the feature input. The model can efficiently leverage the spatial relations of the visual patterns (e.g. the edges in an image) to reduce the number of parameters that need to be learned. This improves the accuracy of the feedforward-and-backpropagation training procedure. In addition, a CNN can be fed with raw data input and automatically discover high-dimensional complex pattern representations. Small local regions in an image are captured by the convolutional filters and treated as inputs to the lower layers of the deep network structure. The information then propagates layer by layer through the whole network. When the feature representations are passed to the next convolutional layer, the convolutional filters capture the salient features of the input which originate at the edges and corners of the image. The extracted simple pattern can be reassembled to complex visual patterns such as shapes and color textures. Since deep learning can model very complex features, a CNN provides a general-purpose learning framework not requiring feature extraction and fine-tuning beforehand.

The advantage of CNN is that its hierarchy structure of learning layers can be trained in a robust manner once the topology of the model fits the target features. The model can efficiently leverage the spatial relations of the visual patterns (e.g. the edges in an image) to reduce the number of parameters which are needed to be learned and thus it improves the subsequent general feed-forward back propagation training to achieve extreme accuracy. Theoretically CNN models can learn very complex features by the above thorough transformation. It is an expert-independent learning model that saves time and manpower for data preprocessing. Based on the literature review, we find CNN is the best solution for the malaria blood smear classification task.

1.4 Research Background

In my research, a new CNN-based neural network model is designed, trained, and applied as a highly sensitive and reliable diagnostic model for malaria. After sufficient training, the CNN model can automatically classify the malaria infected erythrocytes from normal uninfected blood cells from thin blood smears. The CNN model will be training be by a balanced dataset (i.e. the ratio of infected blood cells and normal blood cells is 1:1) acquired from Chittagong Medical College Hospital in Bangladesh. This research project is funded by the 2015 HHS Ventures Fund project "Watch it, Parasite!" and the NIH intramural research funding. The work is organized by the Image Processing Group at the Lister Hill National Center for Biomedical Communications (LHNCBC), a branch of National Library of Medicine (NLM), and National Institute of Allergy and Infectious Diseases (NIAID) affiliated to National Institutes of Health (NIH), at Bethesda, MD, USA. The thesis candidate received the funding by the U.S. Department of Energy (DOE) and worked for NIH in the 2016 Summer Internship program from June to August, 2016. The NIH project primary investigator, Dr. Stefan Jaeger, has agreed to grant the access of the image data to this master thesis for all relevant research purposes.

Chapter 2

Literature Review

2.1 Automatic Diagnosis by Machine Learning

The application of machine learning for visual diagnosis started in the late 1980s, when Kersten et. al. proposed that 2-D optical images collected from medical research and healthcare practice can be represented by neural network models for recognition. The idea was constrained at that time by limitation of statistical modeling, algorithms, and computer software [10]. Marr believes that the computer image processing is to mimic the visual information cognitive procedure of human [11]. Taylor et. al. point out that the goal of computer image processing is not only to represent the visual information, but also to recognize or discover the pattern inside them. They also illustrate major difficulty in image processing for medical image interpretation and face recognition in the 1990s is that the available algorithms were incapable of sufficiently modelling the natural vision [12]. The application of machine learning for automatic diagnosis is traced back to the 1970s and it is developed and improved with the advance and maturity of the infrastructure of information technology (IT) in the healthcare system [13]. For example, a predictive model named COMPASS (Computerized Decision Aid for Stroke thrombolysis) is developed to decide whether the acute stroke patients should have the thrombolytic treatment based on the available clinical information by balancing the overall benefits and risks [14]. A smart EMR (electronic medical record) information retrieval system based on the language model is developed to extract meaningful information from two independent databases in England to support the planning of clinical trial protocols [15].

In addition to the applications based on numeric and text data, machine learning is also applied to image based medical diagnosis and decision making. Nemoto et. al. review the available machine learning algorithms for medical image processing. They conclude that the machine learning approach provide a crucial method for computer-aided detection (CADe) to detect the clinical significant patterns from massive medical images, which eventually perform computer-aided diagnosis (CADx) [16]. De Bruijne points out that machine learning is increasingly successful method for image-based diagnosis, prognosis and risk assessment. The current challenges are the need of more robust model architecture, data accessibility, and effective technology translation [17]. The applications of machine learning to image-based medical diagnosis are generally divided into two categories. At first, this technology is used to detect and classify the gray-scale image patterns on radiology images generated by CT (computed tomography), MRI (magnetic resonance imaging), PET (positron emission tomography) scans. Depending on different purposes, the applications can be further divided to six categories: (1) medical image segmentation, (2) registration, (3) computer aided detection and diagnosis, (4) brain function or activity analysis and neurological disease diagnosis, (5) content-base3d image retrieval systems, and (6) natural language understanding [18]. Wang et. al. reports a dual-dictionary learning model to reconstruct the CT and MRI images to improve the pattern quality and the model classification capacity due to insufficient training [19]. Nouretdinov et. al. proposes a transductive conformal predictor (TCP) classification model to predict the significant visual patterns in MRI images as diagnostic and prognostic markers in depression [20]. Cheng et. al. introduces a deep learning architecture to discriminate the benign and malignant pulmonary nodules on CT images [21]. All these studies believe a promising future for applying machine learning in radiology image-based diagnosis and significant clinical pattern or marker detection. However, numerous issues such as insufficient training, the uncertainty of model robustness and capacity are the barriers from applying such technology to meet the clinical and industrial requirement.

On the other hand, machine learning technology is also used in histopathological patterns detection and classification on tissue images for clinical applications. The main difference between the histopathological image and radiological image is that the former are usually color images instead of gray-scale images. In the aspect of data processing, the color images are represented by the 3-dimension matrices or tensors with 3 color channels. Thus, to apply machine learning models to represent image data, we need more complex models and more intensive computation to tune these models compared to the above applications for radiology models. The use of Computer-assisted diagnosis (CAD) and machine learning to histopathological image analysis begins in the late 1990s. In accordance with Gurcan et. al., histopathological tissue analysis by CAD and machine learning can be categorized to two classes: to detect the pathologically significant visual patterns as the evidence for the presence or absence of diseases, and to grade the severity of diseases or to quantitatively measure the disease progression [22]. Lessmann et. al. proposes a content-based image retrieval (CBIR) model based on Discrete Wavelet Transform to overcome the information gap between the visual representation and semantic meaning. They applied this method to discriminate the histopathological images of meningioma at the accuracy at 79% and pointed out the main difficulty is the computational cost and to use local subsets of features to represent the overall features [23]. Vanderbeck et. al. proposes a supervised machine learning method to classify histological features of non-alcoholic fatty liver disease (NAFLD). Their experiment reports that the classification algorithm has the overall accuracy at 89%, but the performance to different types of historical tissues varies [24]. Svensson et al. uses a native Bayesian model to detect tumor cells in the blood circulation system. The experiment compared the Bayesian model with manual classification, support vector machine and generative mixture model. And conclude the new method can attain the accuracy to ROI (region of interest) at 99% and 75% to raw images [25]. Gopinath and Shanthi develop an automated diagnosis system to detect thyroid tumor cells by multiple algorithms including decision tree (DT), k-nearest neighbor (k-NN), Elman neural network (ENN) and support vector machine (SVM). It implements multi-class classification with the overall accuracy at 90% [26]. Gertych et. al. used a machine learning model consisting of support vector machine (SVM) and random forest (RF) to detect the pathological patterns on digital images of prostate tissues. The result shows that the more complex the model is, the better performance it yields but the model robustness still needs to be improved [27]. A review on histopathological image analysis points out that computer-assisted diagnosis (CAD) algorithms becomes a complementary option for disease detection, diagnosis, and prognosis prediction for pathology. It also predicts that the development fields of CAD in the future are multi-modal data fusion / registration, correlation between histological signatures with protein and gene expression, exploratory histopathology image analysis, and computer-aided prognosis [22].

2.2 Application of Machine Learning to Image Retrieval

The application of machine learning to automatic image-based diagnosis can be traced back to the artificial technology for image retrieval. Image retrieval is one of the most active research fields in information retrieval (IR) since the 1990s when the focuses were general-purposed applications such as indexing methods, searching models, and text searching of the image annotations [28, 29]. For example, Tan et. al applied a Bayesian model combined with stochastic sampling to enhance document ranking [30]. Ayadi et al. introduced a Bayesian network model based thesaurus to link the semantic meaning to the target images to enhance medical image retrieval by ranking the list of specific medical features such as image modality and image dimensionality. The result is compared with the classic information retrieval model BM25 [31] and shows superiority [32].

The renowned applications of image retrieval include Query By Image Content (QBIC) by IBM [33] and the Virage image and video retrieval system adopted by CNN [34]. The image IR systems aims to extract the image features from the digital image such as pixel distribution, edges, color, etc., and then to rebuild the content representations of the sematic meaning. A typical content-based image retrieval (CBIR) system is built based on a retrieval engine supported by the visual feature 7extraction algorithms, the distance and similarity measuring algorithms, the appropriate storage and access channels and the user-friendly interface [35]. The visual features or pattern in image retrieval are divided into two categories: the primitive features such as color, shape, and edges, and the logical features such as the identity of the combination of an object and a specific background [35]. Most image IR systems refer primitive features by applying strategies such as segmentation and local feature extraction to acquire the desired knowledge domain. However, the sematic meaning of such knowledge is difficult to represent. Some current studies propose to use the captions and annotated text accompanied with the image to refill the sematic gap, but the available methods are only capable of rendering general-purposed information with questionable reliability [36].

Image IR has a broad application scope in the biomedical industry, because compared to the expensive decision making by human, a reliable image IR system provides an economical method for diagnosis and relevant decision makings in both hospital or clinic content

[37]. The common applications include automatic radiologic diagnosis and histopathological diagnosis [35, 38]. By a serial of appropriate classifiers, the IR system is able to extract a variety of useful features from both the visual features and the annotated texts [39]. If the visual patterns can be extracted from the image and combined with the most relevant semantic information, we can perform an automated diagnosis based on the medical image data[29, 40]. For example, a spatially constrained convolutional neural network (SC-CNN) is applied to detect the nuclei of the cancerous cells from the whole-slide histological images of colon cancer [41]. Another study annotates the target image with multiple keywords, and then classifies them by a combined model consisting of wavelet-based center symmetric and local binary patterns (WCS–LBP). In the retrieval part, a confidence score will be assigned to each key word based on the feedback of the machine learning model to reduce semantic gap between the user and the IR system. This model can reduce the error rate by approximately 10% compared to the modified support vector machine (MSVM) model [42]. Another attempt by Morioka et al. who applied the ConText-based algorithm to classify ultrasonic radiology reports reports that the algorithm is effective for contextual feature identification [39]. In addition, many information retrieval technologies are proposed to enhance the IR in medical information systems such as the association rules algorithms [43, 44], the probabilistic models [45], and Bayesian models [46] etc. They show some promising features in different aspects of the healthcare information systems, but there is no an algorithm of IR models that has superiority over other methods.

More recent studies focus on improving the semantic linkage between the images and the text such as captions and annotations, or to apply automatic image annotation methods based on machine learning models to enhance CBIR performance [47]. Therefore, the application of the state-of-the-art deep learning algorithm has a promising future in the application of image processing and classification.

2.3 Automatic Diagnosis of Malaria and Machine Learning

The application of machine learning for malaria blood smear classification started in the 2000s when a software named MalariaCount was used to detect the parasitemia images of Giemsa- stained blood smears [6]. However, the study simply compared the correlation of the machine and the manual parasitemia detection and did not classify the infected red blood cells from the normal ones.

In 2009, a study by Diaz et. al. applied support vector machine (SVM) to classify preprocessed blood smear images to detected infected erythrocytes. The proposed algorithm has good performance in both specificity and sensitivity with a small dataset of 450 malaria images. Unfortunately, the model performance decreases when it is applied exclusively to blood images at the infection stage [26]. Verma et. al. uses a SVM model to classify the infected red blood cells by detecting the proteins secreted by malaria parasite into erythroncyte. The model achieves the accuracy at 86.20% and 88.22% respectively in spliting amino acid and dipeptides composition [48]. Kuang et. al. applies a SVM augmented by profile kernel model (PF-SVM) to detect the malaria degradomes as the method to predict the anti-malaria drug resistance [49]. Another attempt is made in 2011 when a new preprocessing algorithm is used to convert the color image to monochrome image, the method is believed to be able to enhance the accuracy of the SVM classifier with a data set of 266 images at a sensitivity of 0.97 with a mean number of 0.8 false-positive detections per image [6].

Except for SVM, other machine learning models are also applied to malaria automated diagnosis. For example, Wicht et. al. uses the Bayesian model to detect the β-haematin inhibiting compounds in anti-malarial drug discovery [50]. Yin et. al. proposes a tree-like Bayesian Structure Learning Algorithm (TL-BSLA) model to classify the high-throughput transcriptomic data in malaria genomic studies, and believe Bayesian method can analyze massive genomic data in the future malaria studies [51]. Scotti et. al. develops the artificial neural network models (ANNs) as the analytic approach for new discovery of anti-malaria drugs [52]. Das et. al. used a machine model based on Bayesian learning and support vector machine (SVM) to detect malaria infected patterns from the light microscopic images of malaria infected blood smears. The model accuracy is reported to be 84% at the cross validation. However, the model requires a complicated data preprocessing stage that significantly increases the manual workload [53].

Therefore, we can conclude that the available classification models inevitably require image preprocessing procedures and are only evaluated by small data sets. As mentioned in the review by Zinszer et. al., the application of machine learning model can improve the quality of malaria forecasting [54]. Though the reported outcomes are good, the factors such as the data representability and robustness of algorithm performance are questionable.

2.4 Deep learning and convolutional neural network

Deep learning is an artificial neural network (ANN) model with multiple hidden layers. This technology has applied to medical research and recently given the maturity of high performance general purpose processors [55]. Convolutional neural network (CNN) is a series of deep learning architecture particularly used for image and vision recognition. A CNN model processes input data by its multiple layers by the four key ideas: local connections, shared weights, pooling, and the use of many layers [56]. The early applications of CNN can be traced back to the 1990s for speech recognition [57] and text recognition [58]. Its use is then extended to handwriting recognition [59] and later to natural image recognition [60]. However, the performance of the CNN models for natural image classification has not improved until the introduction of ImageNet by Alex Krizevsky (thus also named as AlexNet) in 2012. The AlexNet is considered the breakthrough of the application of CNN for multi-categorical classification. In the ILSVRC-2012 competition, the ImageNet composed of seven convolutional layers successfully classified the ILSVRC-2012 validation and test set with 10,184 categories and 8.9 million images with the top-5 error at 15.3% [61]. The record of top-5 error is renewed to 14.8% by the ZFNet [62], to 7.5% by the VggNet [63], to 6.7% by GoogLeNet in 2014 [64], and to 3.6% by ResNet in 2015 [65].

2.5 Advantages of CNN Compared to Other Methods

The advantage of CNN for visual pattern classification and detection is the complex topological structure that performs multilayer back-propagation. A set of convolutional layers are applied on the top of the CNN architecture that can effectively learn complex, high-dimensional, non-linear mappings by the back-propagation process and use the first few convolutional layers as an appropriate feature extractor for the following fully-connected multilayer networks as the classifier. [66]. Compared to the conventional artificial neural network models and other traditional machine learning such as k-nearest neighbor (KNN) and Bayesian model that rely on the independent feature extraction algorithms to extract features [67], the CNN models can be fed with “raw” image inputs with the minimal preprocessing such as pixel-normalizing and centering. One of the major defects of traditional unstructured neural networks for image recognition is the lack of built-in invariance with respect to translations or local distortions of the inputs. The CNN model applies a particular topological structure to force the extraction of local features by restricting the receptive fields of hidden units to be local. The mechanism is based on the physiological study on the cats’ visual recognition neural system by Hubel et. Al in 1959. [8] They discover a visual object is remembered by locally-sensitive, orientation-selective neurons in the neural system. A neural system can extract the basic visual patterns such as oriented edges, end-points, corners and integrate them on in the higher layers of the neural system. In a CNN model, the same mechanism is stimulated by applying a serial of convolutional kernels to detect the local visual features and to represent them as feature maps. As the result, all feature maps are assembled on in a convolutional layer and passed to the next layer. In general, a CNN use three methods to ensure some degree of shift and distortion invariance: local receptive fields by a serial of convolution operations, shared weights by activation functions, and spatial or temporal subsampling by the average or max pooling layers to improve computing efficiency. Therefore, the CNN model provides a more straightforward solution for visual pattern detection and classification. The CNN usually requires higher standard hardware implementations because it needs to compute numerous vector weights and matrix operations. Taking the advantage of the continuous decreasing of storage cost and the increasing of cost-effect of cloud computing resource. CNN becomes the state-of-the-art AI solution for image processing at present.

2.6 Challenges to CNN Applications

The performance of a CNN network is affected by multiple factors. In a simple CNN architecture, the model starts with a few small convolutional kernels (or filters) in combination with a deep network architecture to capture the discernable image features as much as possible. In a more complex CNN, the architecture will inevitably increase the demand for more powerful computing resources. New technologies such as GPU and cluster computing can effectively improve the training efficiency but they are unable to ensure classification performance. Other factors such as data preprocessing and size of the training dataset also affect the classification accuracy. Since the accuracy depends on the amount of training data, small datasets such as those used in earlier approaches are not large enough for training a deep model with its many parameters.

In order the solve the problem of insufficiency of training data, a compromised method called transfer learning is introduced where a pre-trained is used for feature extraction and these features will be used to fine-tune a conventional classifier for final outcomes [68]. Transfer learning can be used as a shortcut of deep learning where we can save time for training at the cost of relatively low but still acceptable performance. This method can be used as a temporary replacement when large training is inaccessible. In this study, we will implement deep learning by both training a newly configured CNN model and applying transfer learning in order to evaluate their feasibility for malaria blood smear classification.

Chapter 3

Our Proposed Method

3.1 General Procedure of Deep Learning

As the one of the most important of deep learning, the architecture of the CNN model will largely determine the final performance. The basic mechanism of deep learning is to apply a multi-layer network to distort the input space and to transform it by the hidden nodes. By a serial of transformations, the model can learn the patterns of the input data by back-propagation. By this procedure, the partial derivative or gradient of the input parameters are computed from the partial derivative of the output (See Figure 3.1 Figure 3.2) by the chain rule. Thus the changes from one layer can be computed by measuring the changes of other layers connected to it.

The learning of the CNN model is acquired by two inverse computations: the feed-forward and the back-propagation [56]. The feed-forward is to compute the output of one layer from all units in this layer where a non-linear activation function $f()$ is applied to the sum of weight z from the lower layer. The function can be a rectified linear unit (ReLU), hyperbolic tangent (tanh), logistic function etc. On the other hand, backward propagation is applied to fine-tuning the deep network by computing the parameters of each layer inversely. As shown in Figure 3.3, given the loss function for the unit l in the output layer is $0.5 \times (y_l - t_l)$ where t_l is the output, the error derivative $y_l - t_l$ of the output can be converted to the error derivative of the total by multiplying it by the partial derivative of $f(z)$.

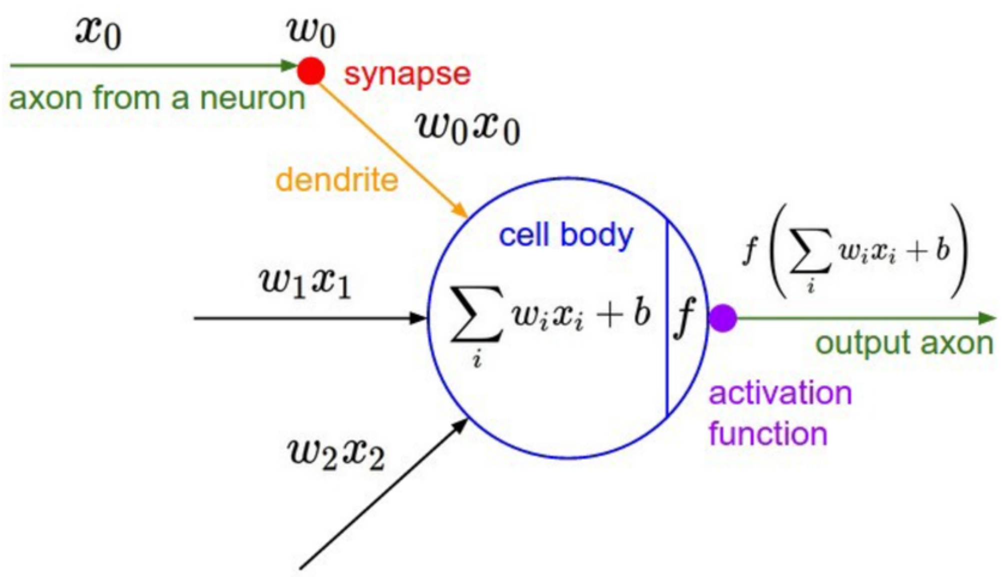
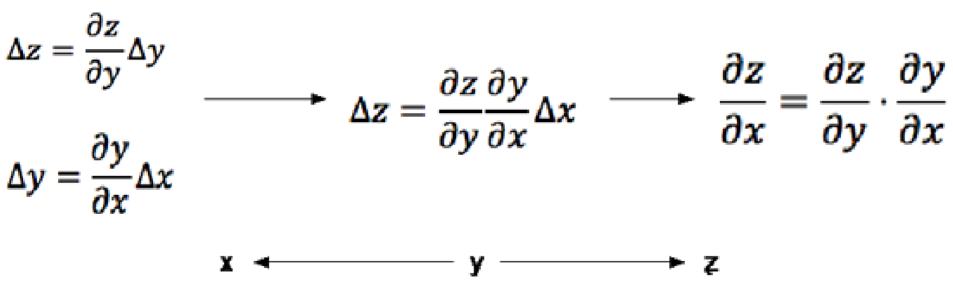
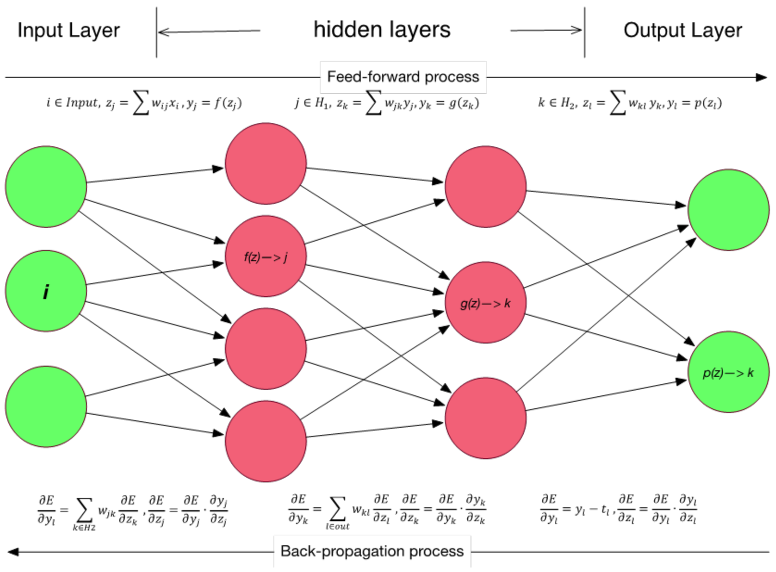


FIGURE 3.1: A Neuron in a Neural Network



$\Delta z = \frac{\partial z}{\partial y}\Delta y$
$\Delta y = \frac{\partial y}{\partial x}\Delta x$
$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$
$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\cdot\frac{\partial y}{\partial x}$
x
y
z

FIGURE 3.2: Gradient Descent in Backward Propagation



Input Layer
hidden layers
Output Layer
Feed-forward process
$i \in Input,\ z_j = \sum w_{ij}x_i, y_j = f(z_j)$
$j \in H_1,\ z_k = \sum w_{jk}y_j, y_k = g(z_k)$
$k \in H_2,\ z_l = \sum w_{kl}\,y_k, y_l = p(z_l)$
i
f(z)—> j
g(z)—> k
p(z)—> k
$\frac{\partial E}{\partial y_l} = \sum_{k \in H2} w_{jk}\frac{\partial E}{\partial z_k}, \frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\cdot\frac{\partial y_j}{\partial z_j}$
$\frac{\partial E}{\partial y_k} = \sum_{l \in out} w_{kl}\frac{\partial E}{\partial z_l}, \frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\cdot\frac{\partial y_k}{\partial z_k}$
$\frac{\partial E}{\partial y_l} = y_l - t_l, \frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\cdot\frac{\partial y_l}{\partial z_l}$
Back-propagation process

FIGURE 3.3: Feed-forward and Backward Propagation in a Deep Neural Network

By the above operation, the CNN model acquires learning when sufficient raw data enter and go through the whole network. In general a typical CNN architecture consists of four type of layers: convolutional layer, pooling layer, activation layer, fully connected layer, and Softmax layer, which perform different tasks in the training procedure.

3.2 Convolutional Layer

The convolutional layer is the most important component of the CNN architecture. It consists of a set of filters which are feature maps can are connected to the local patches in the feature maps of the previous layer by a set of weights.

In mathematics, the convolution of two vectors, u and v, represents the magnitude of overlap when the filter vector v slides across u. Let m be the length of vector u and n be the length of the filter vector v, then the vector w of the convolution of u and v is typically denoted as $w = u \otimes v$, where $w(k)$ is the convolutional product of the kth entry in vector u, as computed by $Formula 3.1$.

$$w(k) = \sum_j u(j)v(k - j + 1) \tag{3.1}$$

In the image processing through a convolutional neural network, an image is represented by a 3-dimensional matrix where the first two dimensions represent the pixel values and the third dimension represents a specific color channel (e.g. Red, Green, and Blue). Accordingly, the convolutional operation on an image can be written in algebraic notation as:

$$h(x) = f(x) \otimes g(x) = \int_{-\infty}^{+\infty} f(\tau)g(x - \tau)d\tau \tag{3.2}$$

Where $f(x)$ represents the input image and $g(x)$ is the function of the filter (or kernel). Then the convolution product is the integral of the dot product of the input image and the filter, where the filtered output is nonzero.

The local sum of weights of each unit is then passed to a non-linearity such as a ReLU or a sigmoid function. All units belonging to a feature map share the same set of weights or filter bank, while different feature maps use different filter banks for feature representation [57]. The reason for this design is based on two considerations. First, the images are fed into the network as array data which represent the features of the local pixels. Since local pixels are assumed highly correlated, forming distinctive local motifs are easier to be detected. Second, images signals are invariant to location, thus the change of location will not divert the feature of the identical pattern on the other location. In mathematics, this layer performs filtering operations by a feature map in a discrete convolution. As illustrated in Figure 3.3, a 32×32 pixel color image (i.e. RGB, 3 channel) is scanned by six filters (5×5 pixel, 3 channels) and output as a set of $28 \times 28 \times 6$ activation maps.

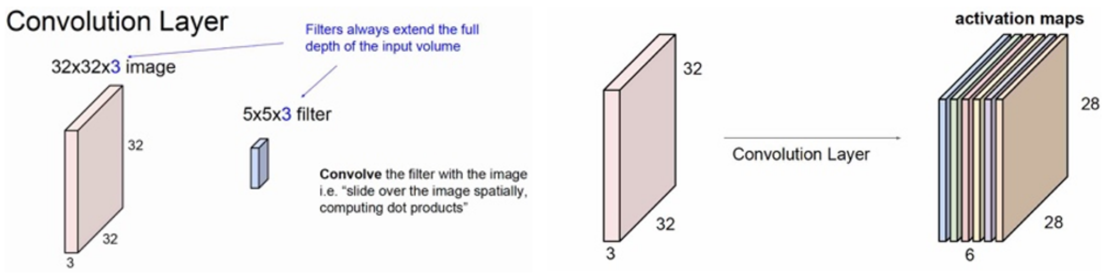

Convolution Layer
32x32x3 image
Filters always extend the full depth of the input volume
5x5x3 filter
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"
32
32
3
Convolution Layer
activation maps
28
28
6

FIGURE 3.4: Information Representation in a Convolutional Layer

3.3 Pooling layer

Unlike the convolutional layer to detect local conjunctions of features passed from the previous layer, the main function of a pooling layer is to perform a merging operation in order to reduce the size of the feature map. The pooling layers can merge the local semantically similar features into a more concise representation. Because the motif formed by the relative positions of the features can vary, the pooling operation can coarse-grain the position of each feature with the motif by a particular function. The common operations include computing the maximum of the local patches of units in the corresponding feature map or maps where the peripheral units in those patches are shifted by more than one row or column. As the result, the dimension of the feature representation is reduced and replaced by an invariance formed by the small shifts and distortions of the pooling operation.

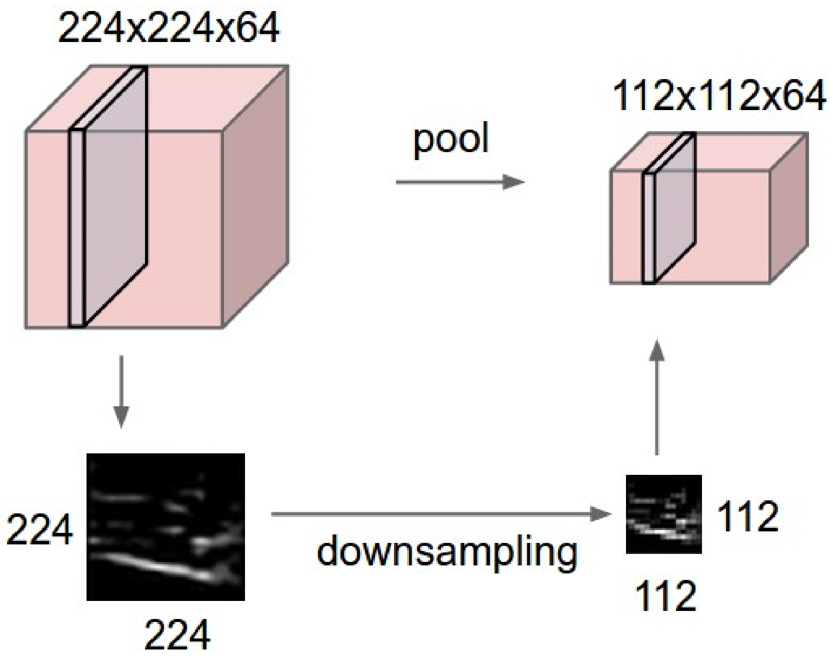


Figure 3.5: Information Downsampling in a Pooling Layer

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 classification by the Softmax function.

3.4 Activation Layers

The activation layers in the CNN architecture are non-linear functions to generate learned knowledge. The typical structure of a CNN is formed by stacking two or three stages of convolution, non-linearity, and pooling, and then followed by more convolutional and fully-connected layer [56]. The learned patterns are generated in the networks when backpropagating gradients go through the CNN model that is trained by repeatedly recomputing the weights in all the filters banks. There are three types of activation layer used in CNN for image classification: the rectified linear unit (ReLU) layer, the sigmoid layer, and the cross-channel normalization layers.

The ReLU layer can greatly accelerate the convergence of stochastic gradient with the low-cost operation where all the positive values are kept and all the negative values are reset to zero (See $Formula$3.3). However, the overuse of ReLU will easily suppress all tiny non-zero values to zero which will never be reactivated [62]. On the other hand, the sigmoid layer squashes a real-valued number into the range between 0 and 1 (See $Formula 3.4$) leading to the increase of representation density. However, the sigmoid layer will saturate and eventually kill the gradients when the gradient values turn to small as the absolute values of the input increase.

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{3.3}$$

$$f(x, a, c) = \frac{1}{1 + e^{-a(x-c)}} \tag{3.4}$$

The current trend for the activation functions is to simply use ReLU layers for generating learning. The state-of-art CNN models is composed of 10 to 20 ReLU layers to compute hundreds of millions of weights between billions of neuron connections [10], which takes extremely long time and requires continuous upgrades in hardware (e.g. GPU), software (e.g. CUDA), and algorithms.

Another choice of the activation layer is the cross-channel normalization layer which returns the local response normalization of each input element (See $Formula 3.5$)

$$x^{'} = \frac{x}{\kappa + \frac{\alpha \times ss}{windowChannelSize}} \tag{3.5}$$

where κ, α, and β are the hyperparameters, and ss is the sum of squares of the elements in the normalization window [15]. The normalization scheme can improve feature generalization by the activity of a neuron $a^i_{x,y}$ computed by the kernel i and position (x, y) and then applying the ReLU nonlinearity. The response-normalized activity $b^i_{x,y}$ is given by the $Formula$ 3.6.

$$b^i_{x,y} = \frac{a^i_{x,y}}{(\kappa + \alpha \sum_{max(0,i-\frac{n}{2})}^{min(N-1,i+\frac{n}{2})} (a^j_{x,y})^2)^{\beta}} \tag{3.6}$$

where the sum of n "adjacent" kernel maps at the same spatial position is computed and N is the total number of kernels in the layer. The normalization layer performs the local contrast normalization scheme which can improve learning [68]. However, since we perform a global contrast normalization as a part of data preprocessing, this normalization layer is not used in our CNN architecture. The comparison of the three activation functions is illustrated in Figure 3.6.

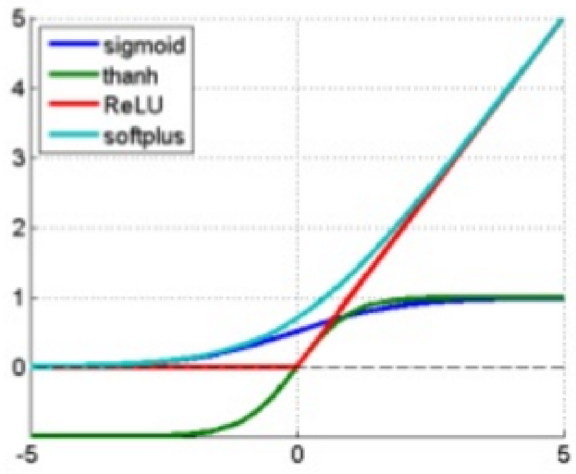


FIGURE 3.6: Comparison of Different Activation Functions

3.5 Output Layers

At the end of a CNN architecture, the structure usually has a few fully connected layers connected to a Softmax function for outputs, which are the typical outputs layers. The fully connected layers contain neurons that connect to the entire input volume as other neural networks. Note that we can still add ReLU layers between the fully connected layers to generate learning but this practice is not common. The Softmax layer returns the probability regarding the conditional probability of the given class. It is also known as the normalized exponential and can be considered as the multi-class generalization of the logistic sigmoid function (See $Formula$3.7 Figure 3.7) [68].

$$P(C_r \mid x) = \frac{P(x \mid C_r)P(C_r)}{\sum_{j=1}^{k} P(x \mid C_j)P(C_j)} = \frac{exp(a_r)}{\sum_{j=1}^{k} exp(a_j)} \tag{3.7}$$

$$(0 \leq P(C_r \mid x) \leq 1 \quad and \quad \textstyle\sum_{j=1}^{k} P(c_j \mid x) = 1)$$

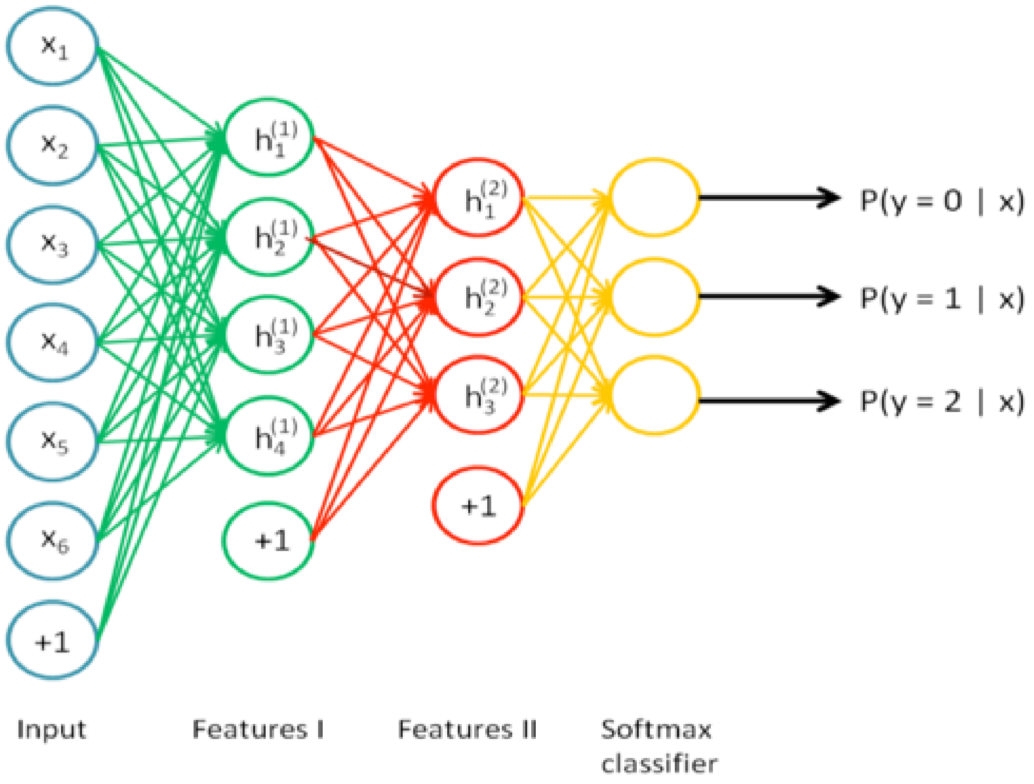


FIGURE 3.7: Output of the Softmax Layer

3.6 Model Optimization and Fine-tuning

As a common issue of machine learning, model optimization and fine-tuning play a crucial role to the final performance of the algorithm. In general, the goal of training a machine learning model (i.e. supervised learning) is to approximate a representation function to map the input variables (Xs) to an output variable (Y). When a model is trained, its parameters (the weights of the neurons in deep learning) are adjusted by the input data patterns. When the training data is a sample with good representation to the general, the trained model acquires the capacity of generalization to solve the unseen case with the acquired knowledge, which means a well-tuned machine learning model can learn and generalize the real-world data. Conversely, if a trained model cannot solve the unseen problems, there are two cases: over-fitting and under-fitting.

Over-fitting means the model is trained too well by the sample data to solve unseen analogous problems. It happens when the model learns both the due patterns of the training data and the noise generated by random fluctuations. As a result, the function curve of the over-fitting model is so close to the sample data to generalize that it loses the capacity to approximate the common patterns of the homologous data. (See Figure 3.8).

On the other hand, under-fitting means a trained model can neither solve the training data nor generalize the unseen new data. An under-fitting model can be produced when a unsuitable model is chosen or the wrong architecture is applied. In practice, it is easy to detect and solve the issue of under-fitting by the alternative modelling. (See Figure 3.8)

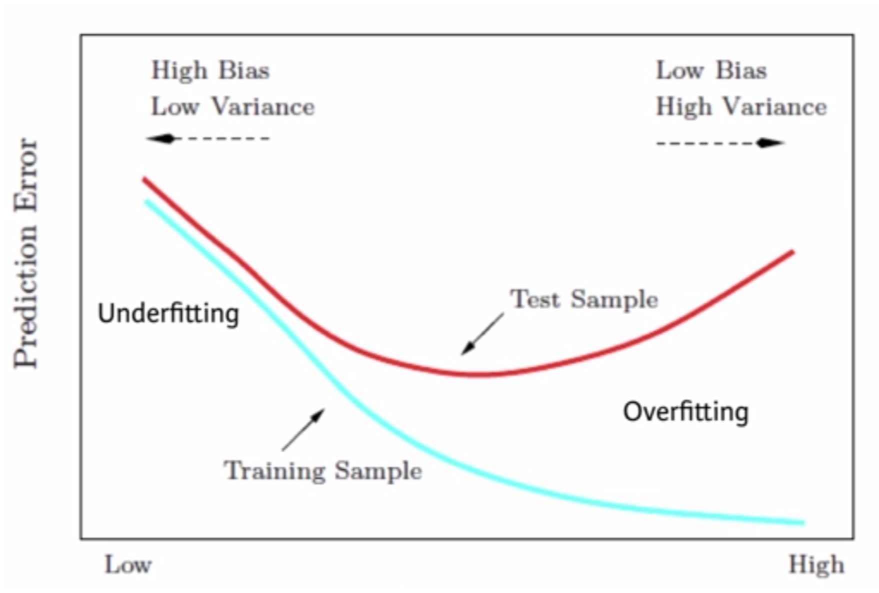


FIGURE 3.8: Under-fitting v.s. Over-fitting

Over-fitting is one of the major issues in the model optimization in deep learning. There are many methods to optimize the deep learning models. Many methods are proposed to minimize over-fitting and to provide the trained models with more generalized cognitive capacity. The common methods include alternating the activation functions, adjusting the learning rate based on the change of the outcomes of the loss function after each training epoch, using a random drop out to discard the over-tuned neurons, and to apply an appropriate back-propagation algorithm to experdite the learning effect at better cost-effect. Stochastic gradient descent is an optimization method for back-propagation by computing the partial derivative or gradients of a series of small subsets instead of the whole dataset with an appropriate learning rate in different training epoches. [69, 70]

Stochastic gradient descent (SGD) is a series of predominant optimization methods for deep learning. [70] In comparison with batch gradient descent (BGD) that computes the gradient of the whole training set, SGD (also known as incremental gradient descent) is a stochastic approximation of the gradient descent optimization method to minimize the target function in the form of a sum of differentiable functions by seeking the minima of the loss function in each iteration of computing the deep neural network. In mathematics, the difference of the ground true value and the approximated value can be present as:

$$Q_{train}(w) = \frac{1}{2m}\sum_{i=1}^{m}(h_w(x^{(i)} - y^{(i)})x_j^{(i)})^2 \tag{3.8}$$

where m is the total number of data points in the training set. The gradient computed in the *jth* epoch can be expressed as:

$$\frac{\partial}{\partial w_j}Q_{train}(w) \tag{3.9}$$

The minimizing of a function can be written in the form of summation:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n}Q_i(w) \tag{3.10}$$

where Q_i is the *ith* observation of the training dataset. $Q_i(w)$ is the value of the loss function at the *ith* data point and $Q(w)$ is the empirical risk (i.e. theoretical bounds of the function). To minimize the total loss, a standard batch gradient descent method can be performed by the following iterations:

$$w := w - \eta\nabla Q(w) = w - \eta\sum_{i=1}^{n}\frac{\nabla Q_i(w)}{n} \tag{3.11}$$

where η is the learning rate to approximate the loss. Note that the batch gradient descent algorithm requires to compute the gradient of the whole training set. Therefore, the computing cost is extremely high in big datasets.

By contrast, stochastic gradient descent (SGD) does not compute the true gradient of $Q(w)$, but the gradient is approximated by the summation of the gradient of a small subset randomly select from the whole batch. The weight of the *ith* observed data point is updated by:

$$cost(w, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_w(x^{(i)}, y^{(i)}))^2 \tag{3.12}$$

then the gradient of the randomly chosen subset is:

$$Q_{train}(w) = \frac{1}{n}\sum_{i=1}^{n} cost(w, (x^{(i)}, y^{(i)})) \tag{3.13}$$

or the above formula can be simplified as:

$$w := w - \eta \nabla Q_i(w) \tag{3.14}$$

where i represents the *ith* randomly selected subset in an epoch of back-propagation. An computing epoch in a SGD algorithm means that the algorithm go through the whole training dataset. An iteration means that when a random subset is randomly chosen, the SGD algorithm goes through the small subset to compute a local summation of gradient.

As the algorithm goes through the whole training set, it updates the value of w by each training example until the algorithm converges. A typical SGD algorithm is implemented by the following steps:

```
Choose an initial parameter vector $w$ and learning rate $\eta$
Repeat until an approximate minimum is achieved:
    Randomize the data examples in the training set
    For $i = 1, 2, \ldots, n$ $do$:
        $w := w - \eta_i(w)$
```

In order to minimize the difference between the true gradient and the gradient computed from a single observed data example, we can compute the gradient against more than one training example (or mini-batch) at each step. This method is called mini-batch gradient descent: $w := w - \eta \nabla Q_j(w_{i:i+n})$ where j is the *jth* batch of the whole training set and n is the batch size. In this study, we apply the classic SGD method with a batch size of 256 images.

In addition to the classic SGD method, there a series method to optimize the SGD algorithm because the limits of the conventional stochastic gradient descent method: first, it is difficult to choose the best learning rate arbitrarily. And an improper learning rate might lead to slow convergence or even divergence. Second, when the algorithm encounter a saddle point in stead of a local minima. The conventional SGD algorithm is difficult to escape from the plateau of the saddle point as the surround gradient is close to zero in all dimensions.

In order to solve the above problem and accelerate the gradient descent procedure, many gradient descent optimization method are respectively proposed:

Momentum is a method to accelerate the SGD in the most relevant direction and to suppress the unnecessary oscillations. [71] In addition to compute the gradient, the momentum method applies a fraction γ from the last iteration to update the current gradient: $v_t = \gamma v_{i-1} + \eta \nabla Q_i(w)$, then $w := w - v_t$. To further expedite the SGD based on the momentum method, Nesterov proposed a revised method called Nesterov accelerated gradient (NAG) [72] where an extra γv_{i-1} term is added to compute the gradient of the *ith* step.

Adaptive gradient algorithm (AdaGrad) is a revised SGD algorithm with per-parameter learning rate. It is an effective SGD algorithm for the training datasets with sparse parameters. [73] In an AdaGrad, the basic learning rate η is multiplied by the vector $\{G_{i,j}\}$ which is the diagonal of the outer product matrix.

$$G = \sum_{\tau=1}^{t} g_\tau g_\tau^T \tag{3.15}$$

where $g_\tau = \nabla Q_i(w)$ is the gradient at the τth iteration. Then the diagonal is given by:

$$G_{i,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2 \tag{3.16}$$

Then $\{G_{i,j}\}$ is updated after each iteration by the formula:

$$w := w - \eta diag(G)^{-\frac{1}{2}} \circ g \tag{3.17}$$

where $\circ$ is the notation of element-wise product. So the update of each parameter is computed by:

$$w_j := w_j - \frac{\eta}{\sqrt{G_{i,j}}} g_j \tag{3.18}$$

Since the denominator of the fraction $\sqrt{G_i} = \sqrt{\sum_{r=1}^{t} g_\tau^2}$ is the ℓ_2 norm of the former derivatives, extreme parameter updates will be suppressed while small updates will be given a higher learning rate η. The AdaGrad method is reported as the best method for non-convex optimization. [73]

Similarly, Zeiler [74] proposed another SGD optimization method called AdaDelta which is consider an revised version of the AdaGrad algorithm. Compared to the AdaGrad method, the AdaDelta method applied a restricted window to accumulate the former gradients to enter the computation for new gradient. Therefore, it offers a less aggressive optimizing method that can adjust the learning rate dynamically instead of decreasing the learning rate monotonically.

The last group of optimizing method is called Root Mean Square Propagation (RMSProp) introduced by Tieleman and Hinton. [75] RMSProp aims to adjust the learning rate for each parameter by dividing the learning rate for the average decaying squared gradients. First, the average is computed by:

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2 \tag{3.19}$$

where γ is the forgetting factor (Hinton sugguests that γ is set to be 0.9), then the updates of the parameters are:

$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w) \tag{3.20}$$

According to Hinton et al. [75], the RMSProp method has excellent capacity to tune the learning rate for different applications. A revised optimizing algorithm called Adam (Adaptive Moment Estimation) is proposed in 2014, which uses both the gradients and the second moments of the gradients. It is considered as the most flexible and universal SGD optimizer at present, So it is used in this study. [76] $Figure$3.9 illustrates the different computing efficiencies of the above mentioned model optimizing methods.

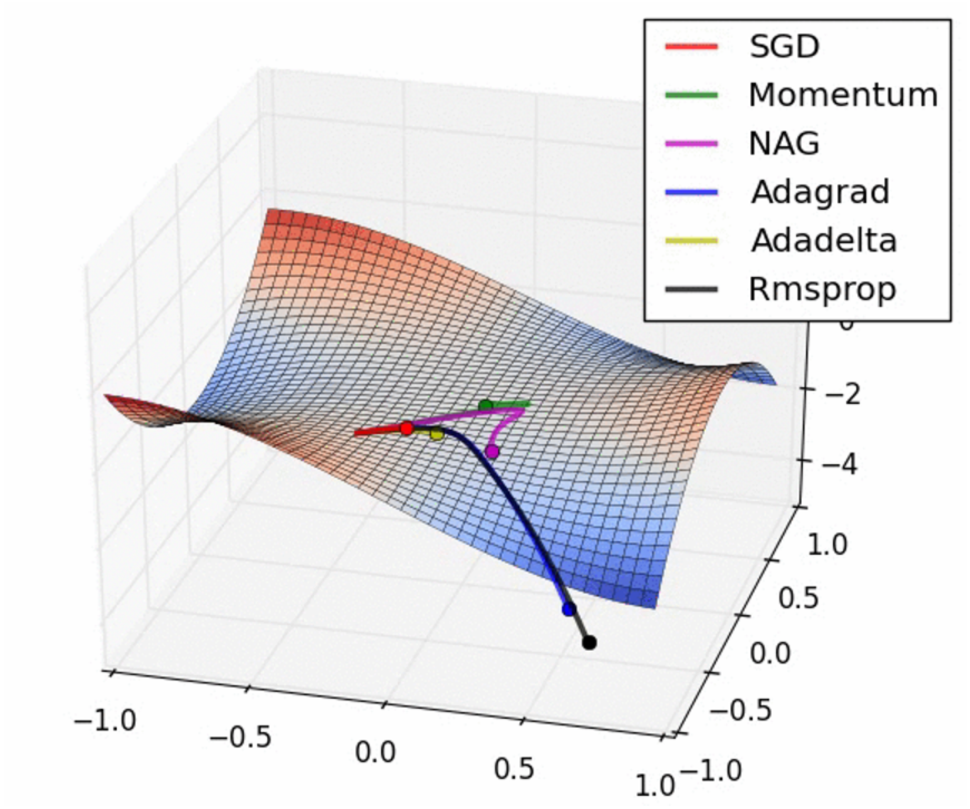


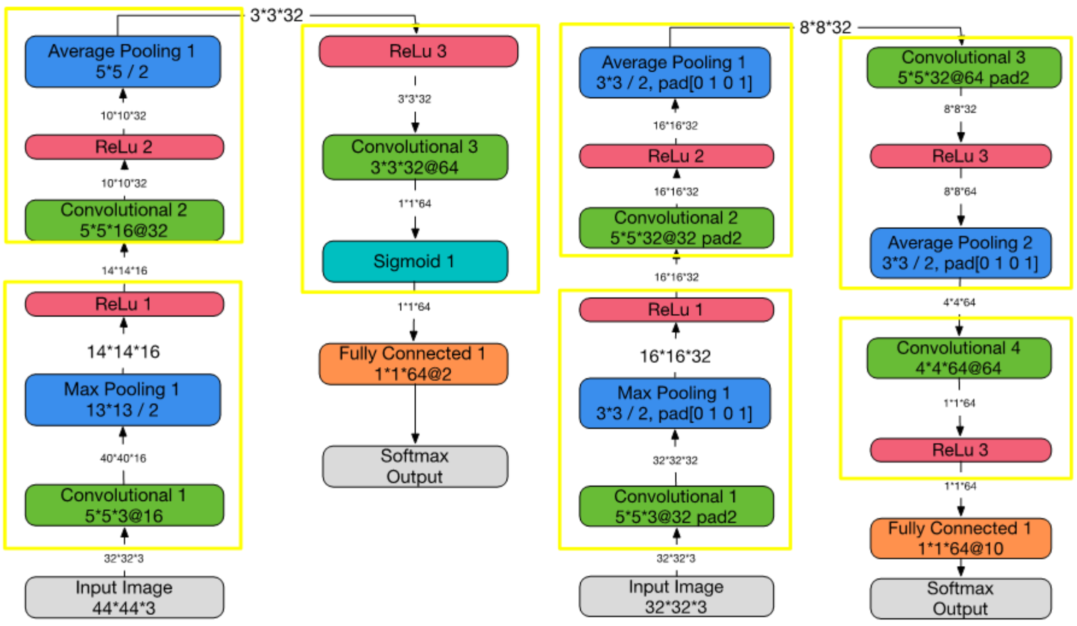
FIGURE 3.9: Comparison of the Efficiencies of Different SGD Optimizers

3.7 CNN Architecture for Malaria Classification

Based on the above discussions, two CNN models named Malaria Net are designed for the malaria image classification task. The first model (Figure 3.10, Left) has 12 layers based on the ImageNet classifier (Figure 3.6, Right) for the CIFAR-10 data set(https://www.cs.toronto.edu/ kriz/cifar.html) of MatConvNet toolbox for MATLAB [77].

As illustrated in Figure 3.10, the models follow the block design pattern, where one convolutional layer, one pooling layer (maximum pooling or average pooling), and one activation layer (ReLU or Sigmoid) are stacked as a single learning block. The whole model is formed by three learning blocks, followed by one fully connected layer, and then following by one Softmax layer to render the probabilities of each category for classification.

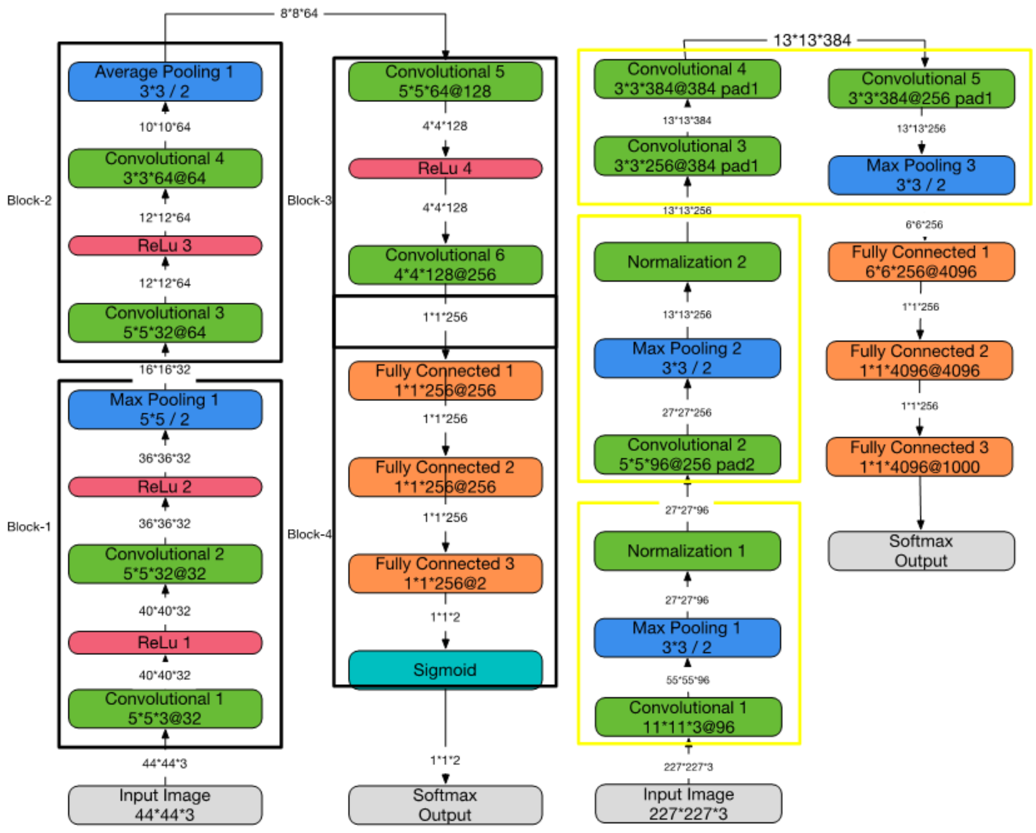
The second model is following the design pattern of AlexNet proposed by Alex Krizhevsky in 2012 [61], where the network structure also follows the block design. As shown in



12-layer Malaria Net for malaria image classification CNN for the CIFAR-10 data set

FIGURE 3.10: CIFAR-like CNN architecture

Figure 3.11 on the left, the 18-layer CNN Malaria Net uses a sandwich design. A single learning block starts with a convolutional layer and ends with a pooling layer. Inside each block, a convolutional layer is followed by a ReLU layer to enhance learning. The model is finalized by three fully connected layers to scale the output to the due input for the Softmax layer to render the probability for each category. The difference of the Malaria Net and the original AlexNet (see Figure 3.11, on the right) is that we replace the normalization layers with ReLU or Sigmoid layers because they can improve learning more efficiently [64, 65].



18-layer Malaria Net for malaria image classification AlexNet

FIGURE 3.11: AlexNet-like CNN architecture

Chapter 4

Model Evaluation

4.1 Data Source

The data set in this study is from National Library of Medicine (NLM), NIH, Bethesda, USA sponsored by the 2015 HHS Ventures Fund project. The images are acquired from the archived blood smear images acquired from Chittagong Medical College Hospital, Bangladesh. The visual region of the erythrocytes is segmented from the original images as raw input data. (See Figure 4.1).

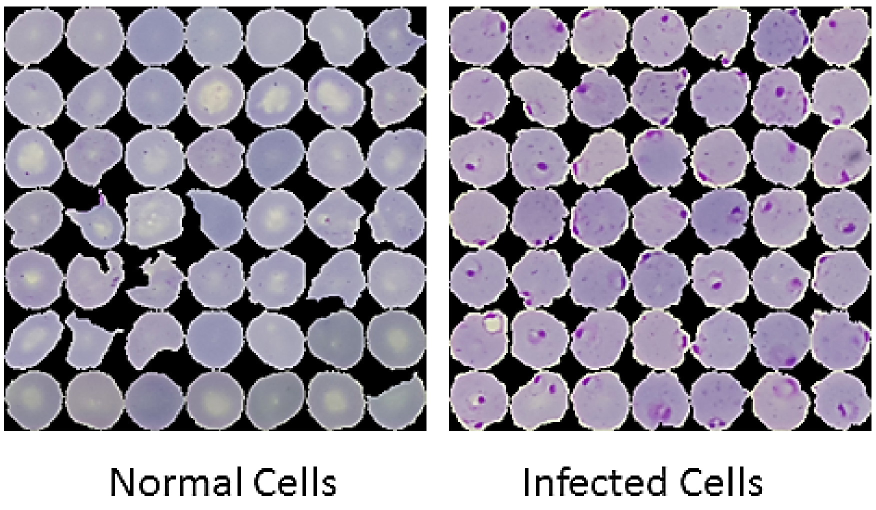


FIGURE 4.1: Raw Malaria Images

The training and test set of this study come from a cell image balanced data set composed of 27,578 erythrocyte images which the ratio of infected cells and uninfected cells is 1:1. Since the resolution of the images is not equal, we chose the median of width and height of the images as the input resolution. All image are resized to 44×44, 3 channel for the training and classification process. In addition, we rotate the images clockwise 4 times to generate a secondary data set with 110,312 images to increase the amount of data and to verify whether larger data set can enhance the CNN classifier performance.

4.2 Experimental Environment

The CNN training and cross-validation is performed on a Desktop computer with NVIDIA GeForce GTX 750 Ti GPU (with 2 GB RAM). The software environment is MATLAB R2016a with Neural Network Toolbox, Parallel Computing Toolbox, Statistics and Machine Learning Toolbox and the MatConvNet community Toolbox [77].

4.3 Data Preprocessing

All image data are read by MATLAB and converted to a 4-dimension data object. The images are resized to the resolution 44×44, 3 channel and then concatenated in the forth dimension. Before the data passed to the CNN model, two steps of preprocessing are applied. First, we perform a normalization step to improve local brightness and contrast. After normalizing the image matrix, the entire dataset is whitened by the eigenvalue decomposition (EVD) operation of the covariance matrix, where x in the original input matrix and $\tilde{x}$ is the transformed whitened matrix. Since the components of the input and output are uncorrelated and their variances equal unity, the covariance matrix of $\tilde{x}$ is equal to the identity matrix.

$$E\left\{\tilde{x} \cdot \tilde{x}^T\right\} = I \tag{4.1}$$

The covariance matrix is $E\left\{\tilde{x} \cdot \tilde{x}^T\right\} = EDE^T$ where E is the orthogonal matrix of eigenvectors of $E\left\{x \cdot x^T\right\}$ and D is the diagonal matrix of its eigenvalues, $D = diag(d_1, \ldots, d_n)$ [77, 78].

In our experiment, the data preprocessing is separated into three steps.

First, the raw images are linked to an ImageDatastore object. The ImageDatastore class is a subclass of the Datastore Class of MATLAB [79]. A Datastore object is a data repository for the collection of big data which are too large to store in the RAM of the machine. By a Datastore object, we can read and process a set of files as a single entity. As a subclass, an ImageDatastore object is a data repository for image manipulation. It provides pointers that respectively point to each image file in the designated directory [80]. By using an ImageDatastore object, we can process a collection of image files that cannot fit the memory with acceptable efficiency. The ImageDatastore also provides an abstract method called 'ReadFcn'. It can be used as a function handle and implemented by a user-defined concrete function. In this study, the raw images are resized to the median width and median height of the whole image dataset, i.e. $44 pixels \times 44$ pixels, by the bicubic interpolation method, where the output pixel value is a weighted average of pixels in the nearest 4-by-4 neighborhood [81]. The following MATLAB code is the implementation of this step.

```

%% define the readAndPreprocessImage function read and resize the raw
    image files

function Iout = readAndPreprocessImage(filename)

    I = imread(filename);

    if ismatrix(I)
        I = cat(3,I,I,I);
    end

    % Resize the image as required for the CNN.
    Iout = imresize(I, [44 44]);

end


%% Create the imageDatastore image data object
% path - the root folder of the images, the name of the classes
% should be the next level of the path
% group - the cell array containing the names of the classes
rootFolder = path;

```

```
% define the two-level nominal levels
categories = group;

% create a imageDatastore Object to manage a collection of image files
imds = imageDatastore(fullfile(rootFolder, categories), 'LabelSource',
    'foldernames');

% define the specify the 'ReadFcn' parameter of the object
% by the ReadFcn, all read-in image are resized to 44*44*3
imds.ReadFcn = @(filename)readAndPreprocessImage(filename, 44, 44);

% Load the image raw data to a 4-dimension matrix
all_images = readall(imds)
```

The Second step is to propare the imdb image data object in the format of the ImageNet image database [82].

The third step is to perform contrast normalizing of the image matrix (i.e. imdb) and to whiten the pixels by computing the diagonal eigenvalue of the image matrix. The MATLAB code for this step is presented below.

```

% img_matrix is the 4-dimension image pixel matrix to store
% resized input images from the newImgArray
img_matrix = newImgArray;
labels = newLabels;
setCell = newSetCell;
clear newImgArray;

% building imdb
imdb = struct();

%% Constrast Normalization
z = reshape(img_matrix,[],size(img_matrix,4)) ;
z = bsxfun(@minus, z, mean(z,1)) ;
n = std(z,0,1) ;
z = bsxfun(@times, z, mean(n) ./ max(n, 40)) ;
img_matrix = reshape(z, 44, 44, 3, []) ;

%%whiten data
```

```
z = reshape(img_matrix,[],size(img_matrix,4)) ;
W = z*z'/size(img_matrix,4) ;
[V,D] = eig(W) ;
the scale is selected to approximately preserve the norm of W
d2 = diag(D) ;
en = sqrt(mean(d2)) ;
z = V*diag(en./max(sqrt(d2), 10))*V'*z ;
img_matrix = reshape(z, 44, 44, 3, []) ;

% IMPORTANT: the image matrix must be single
img_matrix = single(img_matrix);
```

4.4 CNN Model Training

In order to train and evaluate the above CNN model, we implement a ten-fold cross-validation for the whole data set, where 90% of the images are used for training, and 10% are used for the test. In model training, 10% of the data are separated from the training set for back-propagation validation and the rest 90% are used for training. The evaluation procedure of the ten-fold cross validation is shown in Figure 4.2.

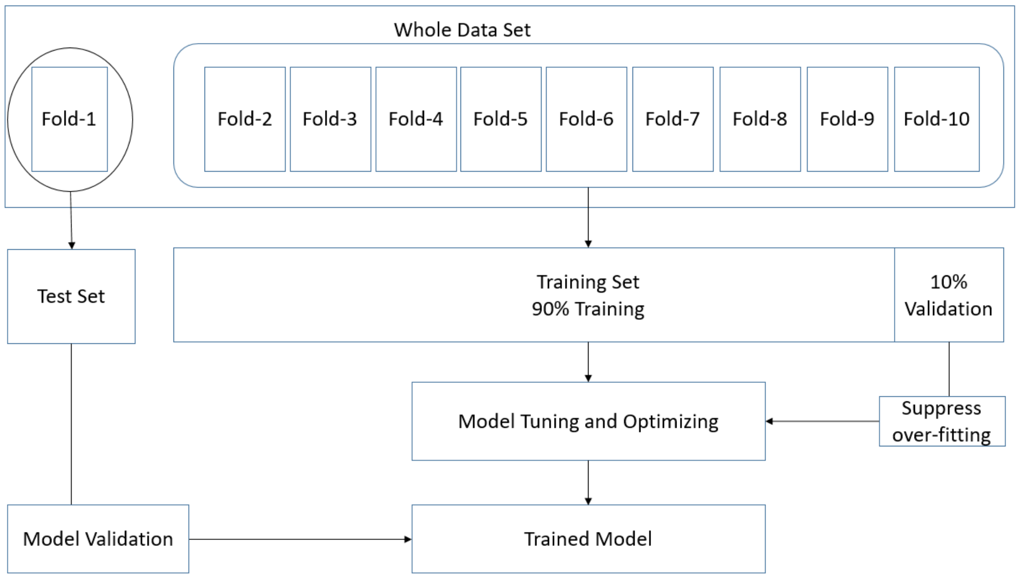


FIGURE 4.2: Ten-fold Cross Validation Flowchart

For comparison, we apply a pre-trained AlexNet which is trained by the CIFAR-100 data set for feature extraction and train the extracted features through the pre-trained model with a conventional support vector machine (SVM) classifier. Furthermore, it is believed that a larger data set can improve the CNN classifier performance. The images in the data set are rotated 90 degrees clockwise for four times thus a larger data set with 110,312 images are formed. The two data sets are used to trained the above two CNN CNN model and the transfer learning model respectively. The results of the ten-fold cross-validations will be compared in the end. The training procedure is illustrated by the flowchart below. (See Figure 4.3)

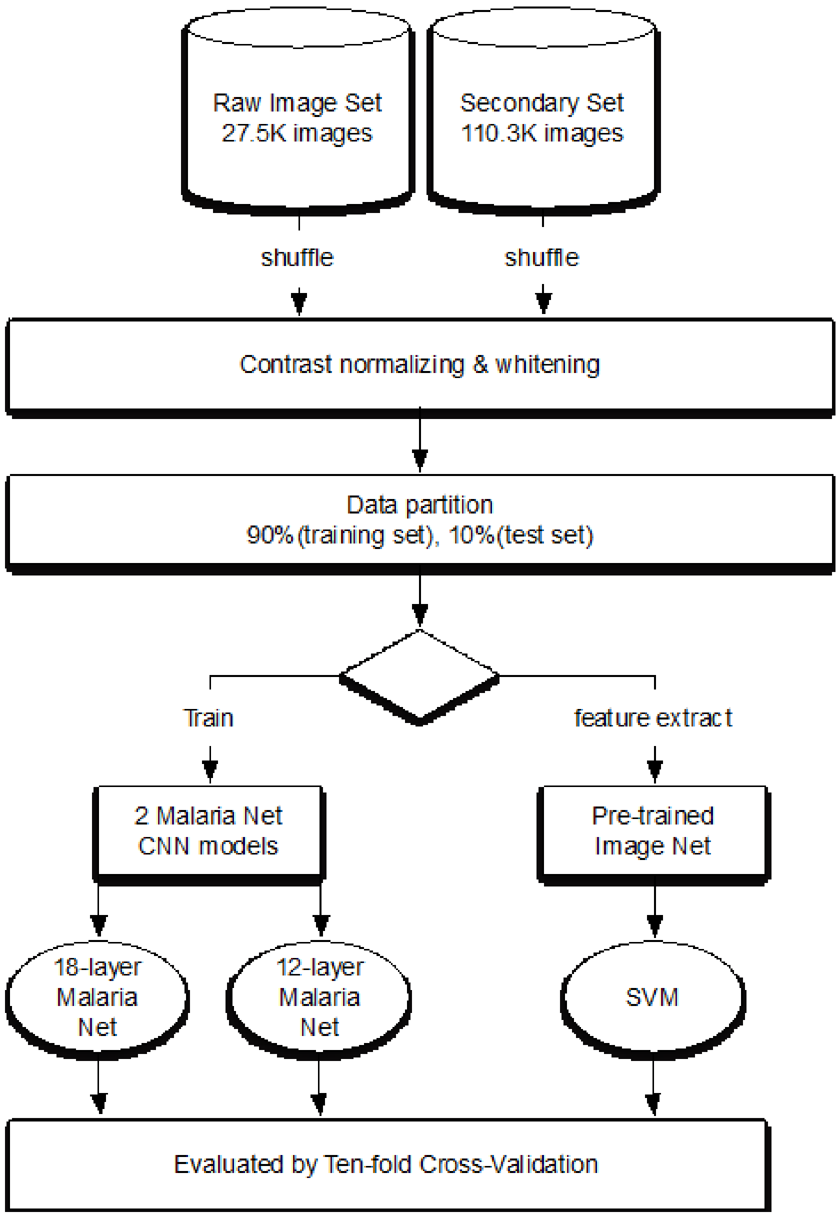
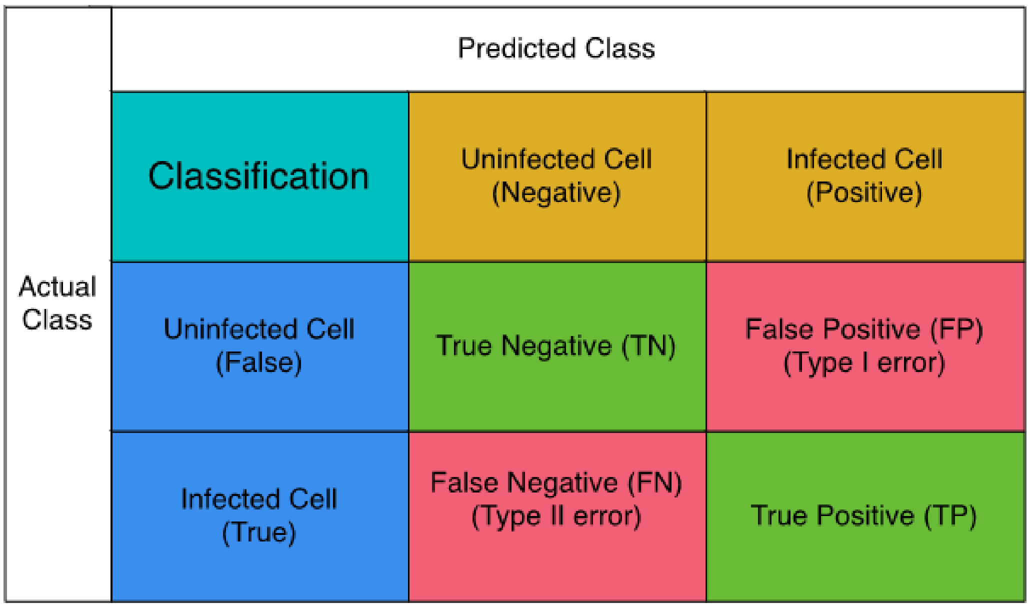


Figure 4.3: Model Training Evaluation Procedure

4.5 Performance Evaluation

The performance of the new CNN models is evaluated by the averages of accuracy, sensitivity, specificity, precision $F1$ score and Matthews correlation coefficient (MCC) computed from each iteration of the ten-fold cross-validation. The parameters are calculated based on the confusion matrices of the classifying. The cell true negative contains the uninfected erythrocyte images that are correctly classified as negative images; the cell false negative contains the infected erythrocyte images that are wrongly classified as negative images; the cell false positive contains the uninfected erythrocyte images that are wrongly classified as positive images; and the cell true positive contains the infected erythrocyte images that are correctly classified as positive images.(See Figure 4.4)



		Predicted Class	
Actual Class	Classification	Uninfected Cell (Negative)	Infected Cell (Positive)
	Uninfected Cell (False)	True Negative (TN)	False Positive (FP) (Type I error)
	Infected Cell (True)	False Negative (FN) (Type II error)	True Positive (TP)

$$Accuracy\ (Acc) = \frac{\sum TN + \sum TP}{\sum TN + \sum TP + \sum FN + \sum FP}$$

$$Sensitivity = \frac{\sum TP}{\sum TP + \sum FN}\ , Sepcificity = \frac{\sum TN}{\sum FP + \sum TN}$$

$$Precision = \frac{\sum TP}{\sum TP + \sum FP}\ , F1\ Score = \frac{2\sum TP}{2\sum TP + \sum FP + \sum FN}$$

$$Matthews\ correlation\ coefficient\ (MCC) = \frac{\sum TP \cdot \sum TN - \sum FP \cdot \sum FN}{\sqrt{(\sum TP + \sum FP)(\sum TP + \sum FN)(\sum TN + \sum FP)(\sum TN + \sum FN)}}$$

FIGURE 4.4: Confusion Matrix

Accuracy (ACC) is the degree of closeness of measurements of a quantity to that quantity's true value, which is defined as $ACC = \frac{\sum TN + \sum TP}{\sum TN + \sum TP + \sum FN + \sum FP}$. Sensitivity (also called the true positive rate, TPR, or the recall) is used to measure the proportion of positives that are correctly classified, which is defined as $TPR = \frac{\sum TP}{\sum TP + \sum FN}$. Specificity (or true negative rate, TNR) is the proportion of the correctly classified negative results, which is defined as $TNR = \frac{\sum TN}{\sum FP + \sum TN}$. Precision (or positive predictive value, PPV) is the the proportion of the correctly classified positive result, which is defined as $PPV = \frac{\sum TP}{\sum TP + \sum FP}$. $F1$ score is the harmonic mean of precision and sensitivity, which is defined as $F1 = \frac{2\sum TP}{2\sum TP + \sum FP + \sum FN}$. In this study, F1 score reflects how the infected erythrocytes are correctly classified. Matthews correlation coefficient (MCC) is a balanced measure for binary classifications when the two categories are unbalanced (i.e. in different size). It is defined as $MCC = \frac{\sum TP \cdot \sum TN - \sum FP \cdot \sum FN}{\sqrt{(\sum TP + \sum FP)(\sum TP + \sum FN)(\sum TN + \sum FP)(\sum TN + \sum FN)}}$, where $-1 \leq MCC \leq 1$. When MCC approaches to 1, it represents a perfect prediction; when $MCC = 0$, it implies the prediction is no better than random guessing; when MCC approaches to -1, it indicates the great disagreement between the prediction and the true result.

Chapter 5

Results

5.1 Training Procedure

In the training experiments, the CNN-based Malaria Net models are respectively trained by the raw image data set and by the secondary rotated data set with 50 epochs. Figure 5.1 illustrates the training curve of the 18-layer Malaria Net model by the large image dataset with 110.3 thousand images. Figure 5.2 illustrates the training curve of the 12-layer Malaria Net model by the small image dataset with 27.5 thousand images. By comparing the training procedure of the complex CNN model (18 layers) with the large dataset to the relative simple CNN model (12 layers) with the small dataset, we can easily conclude that the more complex 18-layer Malaria Net learns faster and generates better cognitive capacity than the 12-layer CNN model with higher accuracy in both the training set and the validation set. This conclusion is further verified by the comparison of the parameters of the confusion matrix.

5.2 Confusion Matrices of the Malaria Net Evaluation

All the parameters of the CNN model performance can be illustrated illustrated by the confusion matrix. The vertical axis represents the predicted labels by the trained CNN model (i.e. the 18-layer Malaria Net). The horizontal axis represents the actual labels of the images. The label '1' represents that the images belong to the infected red blood cells. The label '2' represents that the images belong to the uninfected normal red blood

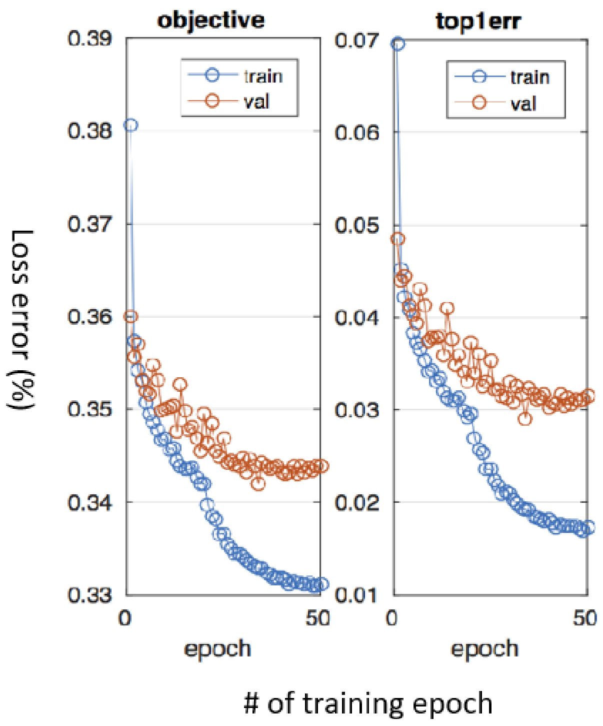


FIGURE 5.1: Training curve of the 18-layer Malaria Net by the 110.3K (images) dataset

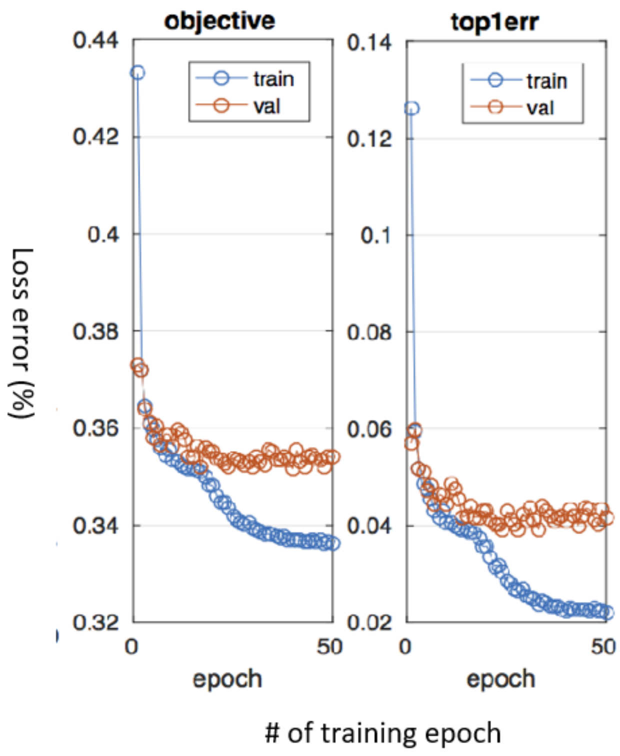


FIGURE 5.2: Training curve of the 12-layer Malaria Net by the 27.5K (images) dataset

cells. When a raw image is classified, if the predicted label is consistent to the actual label, the prediction is correct. The confusion matrices of the ten-fold cross validation of the 18-layer Malaria Net is illustrated below.

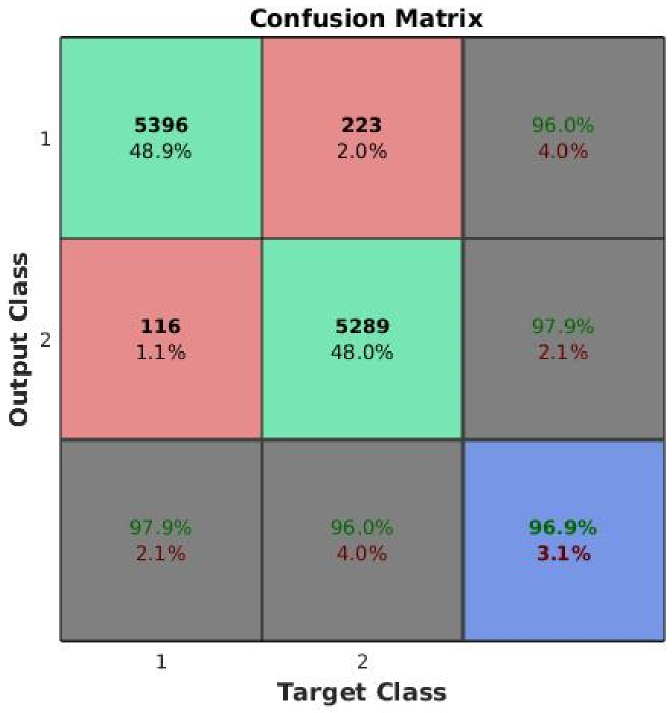


Figure 5.3: Confusion Matrix of the 1st fold Cross Validation

5.3 The Cross Validations of Different Models

The results of the ten-fold cross validations by the CNN based Malaria Net are respectively presented in Table 5.1 and Table 5.2. It indicates that the highest average accuracy of the CNN model is 97.93% attained by the 18-layer Malaria Net with the secondary data set with 110,312 images. When trained by the 27,578 raw image data set, the 18-layer model can still reach the accuracy at 97.37%. The accuracy of 12-layer Malaria Net can reach 96.75% at the 50th epoch when trained by with the secondary data set with rotated images and reach 96.09% when trained by the with the raw data set with 27.5 K images. Note that all the rest parameters such as sensitivity, specificity, and precision all reach very high levels with the trained CNN models. (See Table 5.1 and Table 5.2)

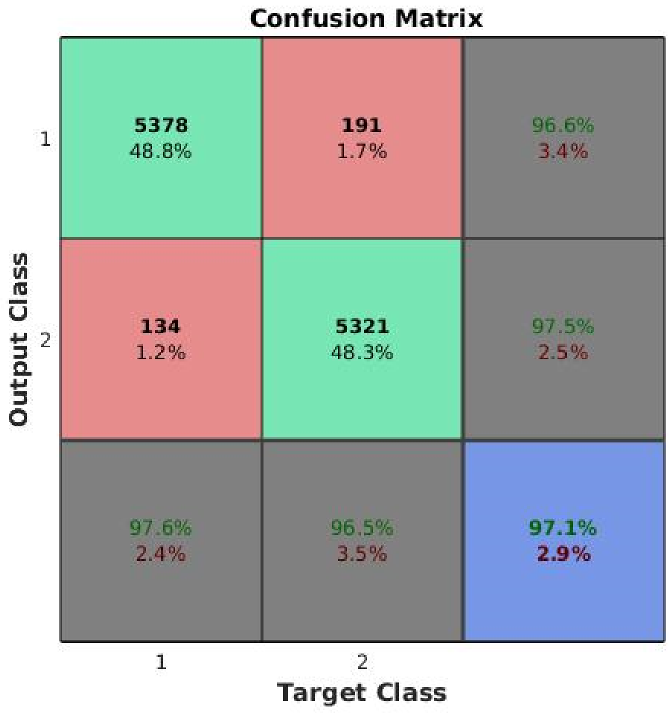


FIGURE 5.4: Confusion Matrix of the 2nd fold Cross Validation

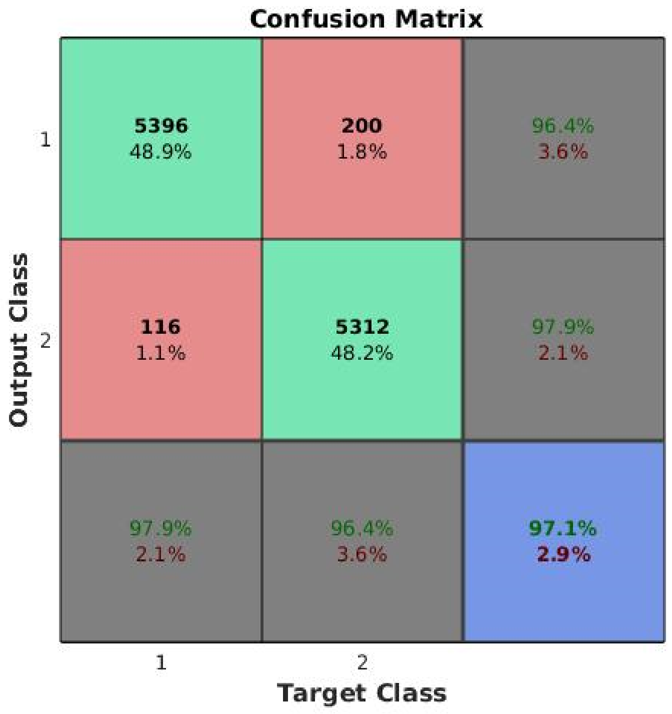


FIGURE 5.5: Confusion Matrix of the 3rd fold Cross Validation

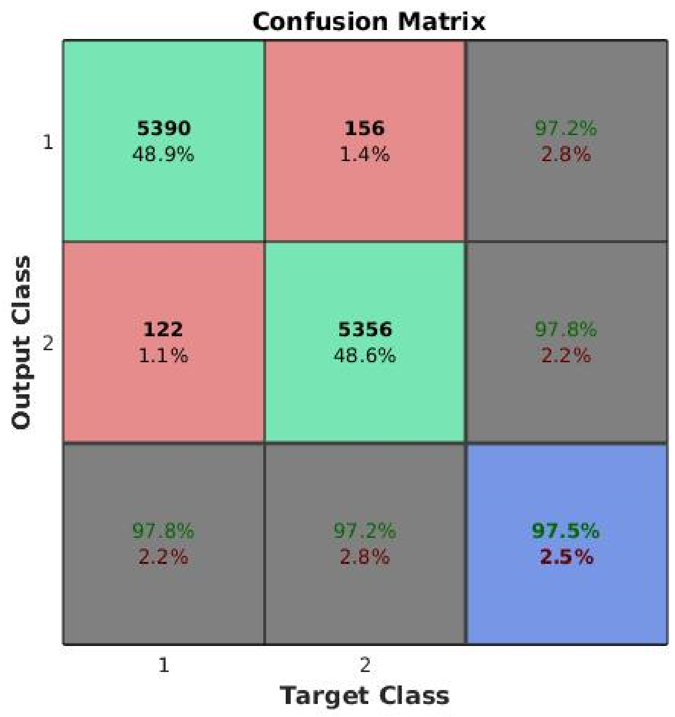


FIGURE 5.6: Confusion Matrix of the 4th fold Cross Validation

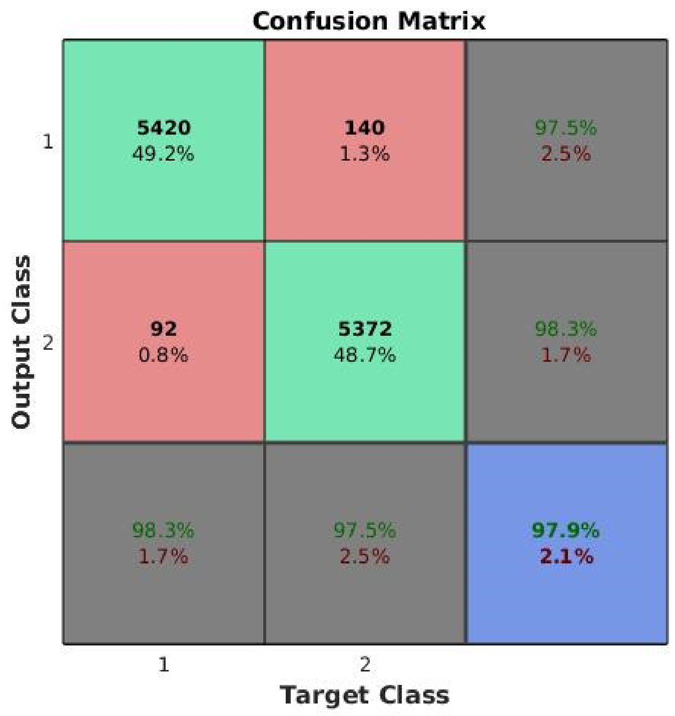


FIGURE 5.7: Confusion Matrix of the 5th fold Cross Validation

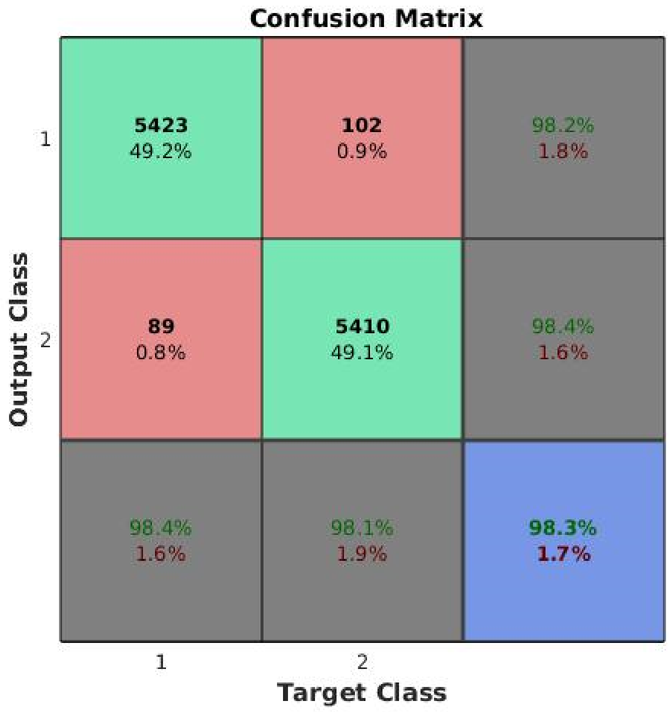


FIGURE 5.8: Confusion Matrix of the 6th fold Cross Validation

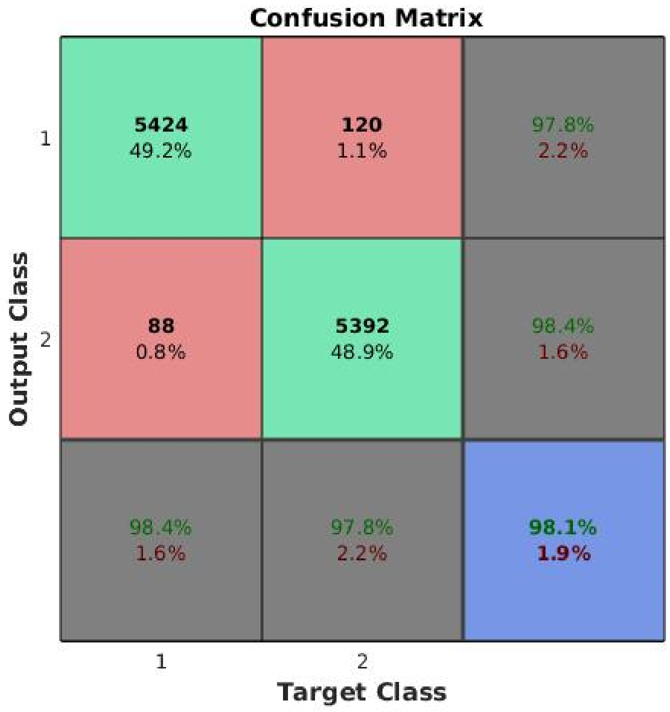


FIGURE 5.9: Confusion Matrix of the 7th fold Cross Validation

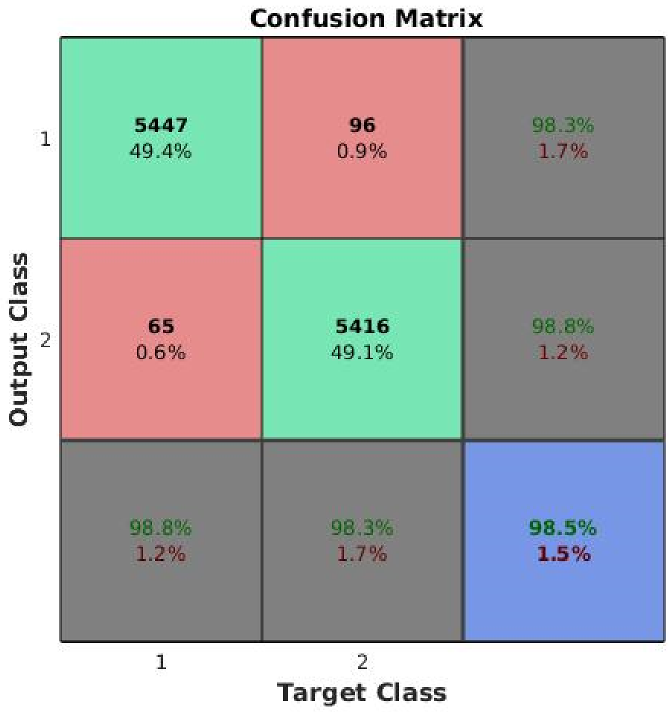


FIGURE 5.10: Confusion Matrix of the 8th fold Cross Validation

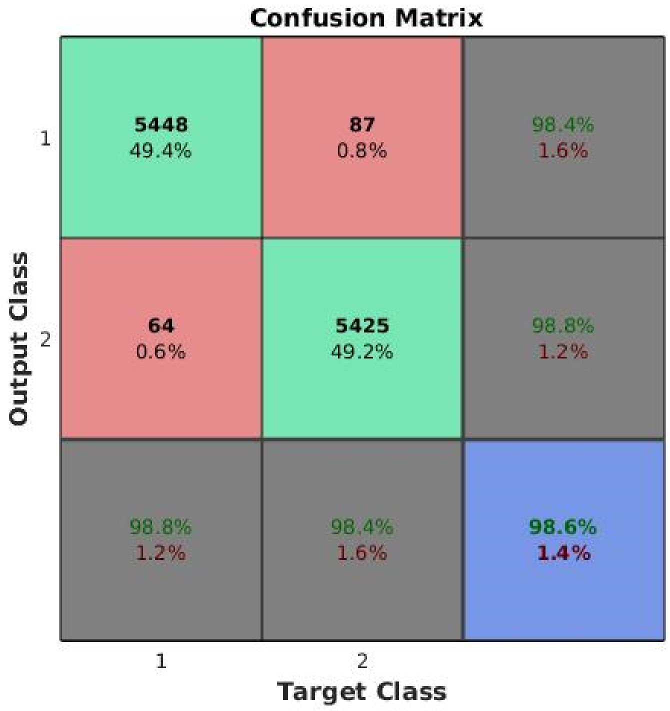


FIGURE 5.11: Confusion Matrix of the 9th fold Cross Validation

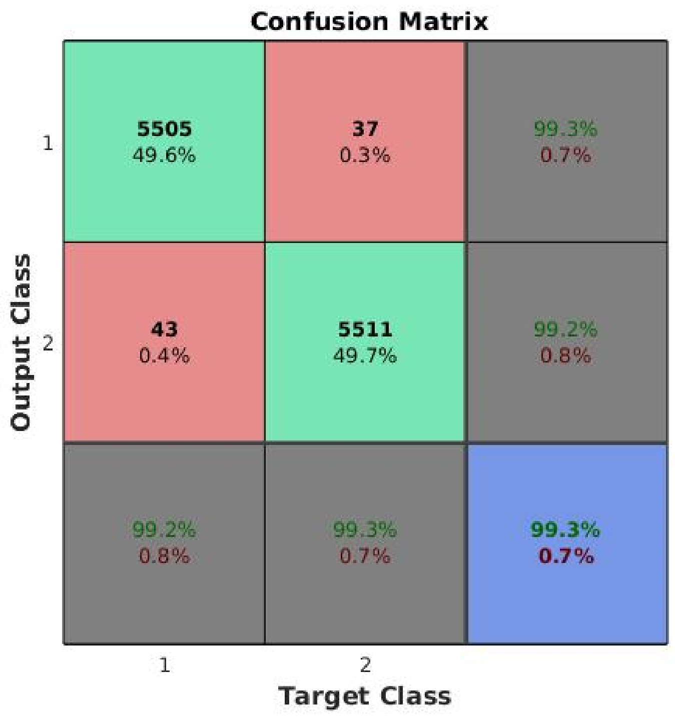


FIGURE 5.12: Confusion Matrix of the 10th fold Cross Validation

TABLE 5.1: Average Outcome of the 18-layer Malaria Net Cross Validation

	27.5K raw data set	110.3K secondary data set
Accuracy	97.37%	97.93%
Sensitivity	96.99%	97.55%
Specificity	97.75%	98.32%
Precision	97.73%	98.30%
F1 score	97.36%	97.92%
MCC	94.75%	95.87%

In comparison, a transfer learning model is also applied to be trained by the same malaria image data. The transfer learning model consists of a pre-trained CNN model as the feature extractor, and a linear SVM model as the classifier. Note that the pre-trained CNN is trained by irrelevant images to tune the model parameters for achieving cognitive capacity. The experiment shows that the transfer learning model has lower performance compared to the outcomes by the original Malaria Net models which are trained by the relevant malaria image data. Table 3 shows that the transfer learning model achieves the overall accuracy at 91.99% at the 50th epoch of the training by the 27.5 K raw image dataset, and its average accuracy reaches 94.26% at the 50th epoch of the training by the 110.3 K secondary dataset with the rotated images. (See Table 5.3).

Table 5.2: Average Outcome of the 12-layer Malaria Net Cross Validation

	27.5K raw data set	110.3K secondary data set
Accuracy	96.09%	96.75%
Sensitivity	95.57%	96.44%
Specificity	96.61%	97.06%
Precision	96.59%	97.05%
F1 score	96.06%	96.75%
MCC	92.22%	93.50%

Table 5.3: Average Outcome of the Cross Validation of Transfer Learning

	27.5K raw data set	110.3K secondary data set
Accuracy	91.99%	94.26%
Sensitivity	89.00%	95.04%
Specificity	94.98%	93.49%
Precision	95.12%	93.70%
F1 score	90.24%	94.30%
MCC	85.25%	88.66%

In addition, the $F1$ score and the Matthews correlation coefficient (MCC) of the trained CNN based Malaria Net models are both more than 7% larger than the numbers achieved by the transfer learning model. This reflects that the trained CNN model is a much better representation of the training images compared to the transfer learning model, which relies on feature extraction from a pre-trained model trained on an entirely different image set.

5.4 Stability and Robustness of the CNN Classifiers

The stability and robustness of the CNN classifier can be measured by the accuracy of the classification in each fold in the cross validation. In our study, both the 18-layer Malaria Net and the 12-layer Malaria Net show stable performance (from the highest at 99.28% by the 18-layer CNN to the lowest at 92.85% by the 12-layer CNN) in the malaria image classification. This result reflects that the convolutional neural network model are robust classifiers for microscopic image classification. (Figure 5.3 to Figure 5.6).

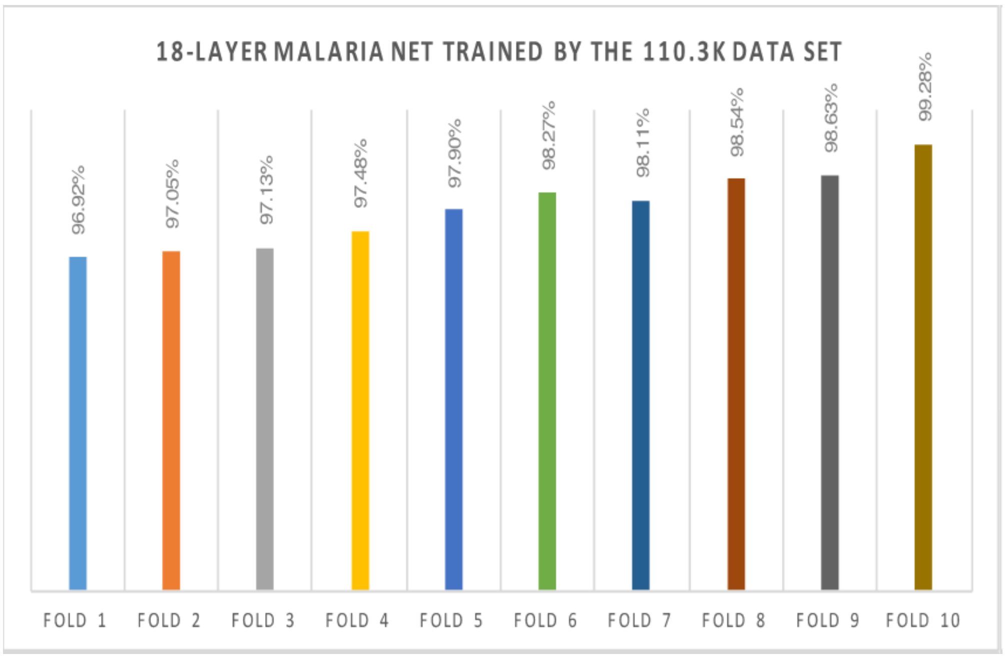


FIGURE 5.13: Performance Stability of the 18-layer Malaria Net by the 110.3K (images) Dataset

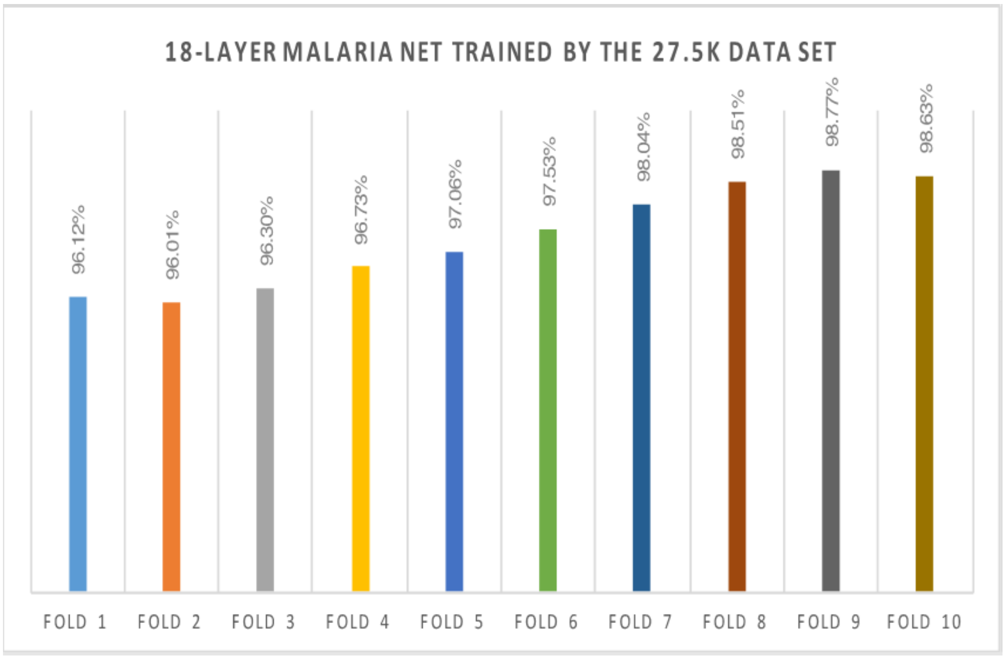


FIGURE 5.14: Performance Stability of the 18-layer Malaria Net by the 27.5K (images) Dataset

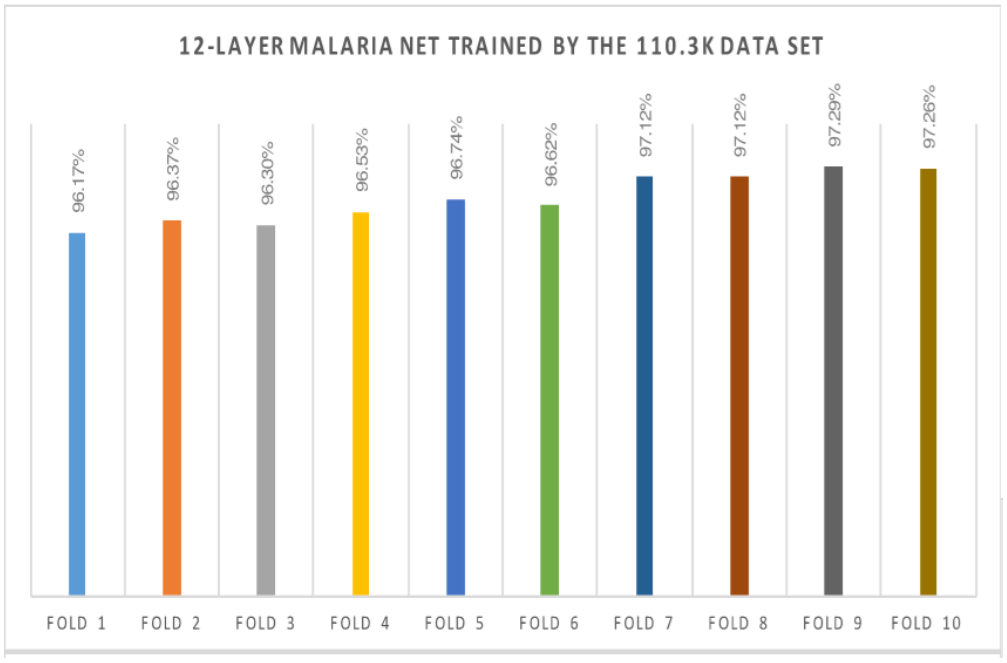


FIGURE 5.15: Performance Stability of the 12-layer Malaria Net by the 110.3K (images) Dataset

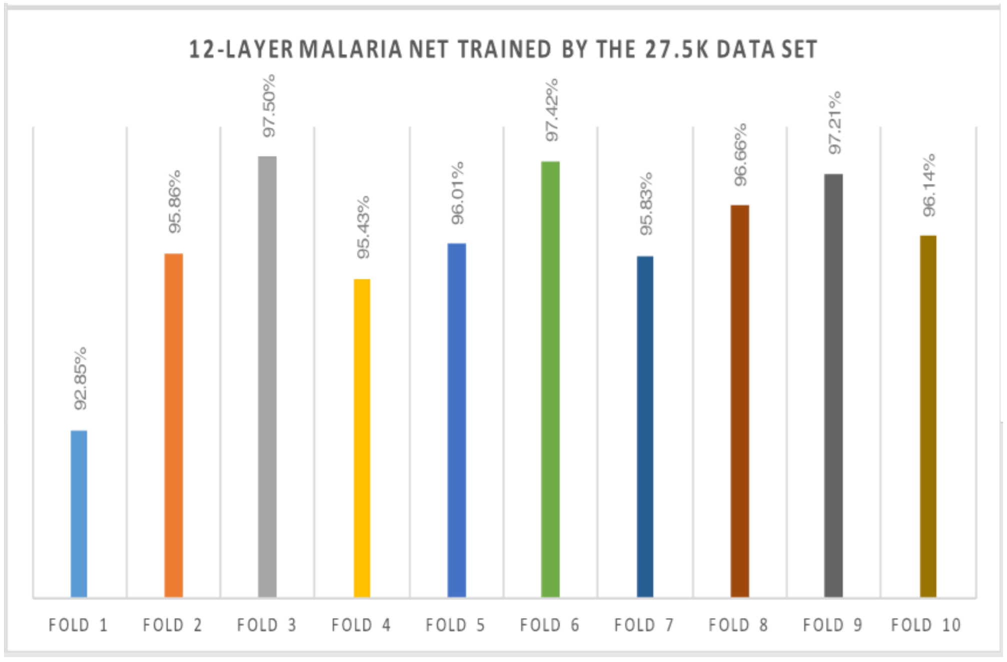


FIGURE 5.16: Performance Stability of the 12-layer Malaria Net by the 27.5K (images) Dataset

Chapter 6

Discussion

6.1 The Global Burden of Malaria

Malaria is a major threat to global health. Most deaths by malaria occur among children resource-poor regions. The 2015 WHO report estimates that malaria causes 438,000 deaths annually among which 69% are in children. The statistics implies that a child dies every two minutes from malaria in our world. Therefore it is a leading cause of childhood neuro-disability [1].

Though the existing drugs such as quinine and artemisinin make malaria a curable disease, inadequate diagnostics and emerging drug resistance are the major barriers for successful mortality reduction. Therefore, a fast and reliable diagnostic test is one of the most promising ways of controlling malaria, together with more timely and effective treatment, development of new malaria vaccines, and mosquito control.

The current standard diagnosis method for malaria is the light microscopic counting of malaria infected red blood cells from blood smears. About 170 million blood samples are to be examined every year for field diagnosis for malaria thus the accuracy of parasite counts is essential to correct diagnosis, drug-resistance testing, and drug efficacy assessment. This non-standardized method depends heavily on the experience and skill of the microscopists. However, the training and working environment for microscopists is poor in malaria epidemic regions. These places are generally in underdeveloped countries and resource-poor regions which are incapable of providing reliable and rigorous system for malaria diagnosis. This status leads to incorrect diagnosis in the field and the consequential healthcare decisions: the false negative cases lead to unnecessary use of antibiotics, a second consultation, lost days of work, and progression to severe syndromes; the false positive cases cause the misdiagnosis leading to unnecessary use of anti-malaria drugs and the potential adverse drug reactions (ADRs), and sometimes severe complications.

6.2 Application of Deep Learning to Automated Diagnosis

Deep learning provides a solution to enhance healthcare decision making by implementing the state-of-the-art technology of artificial intelligence (AI) to the healthcare information system. The unique strength of deep learning is that it can automatically discover complex and abstract general patterns from the raw data with minimal pre-processing through learning via their multiple layers of neurons connected by the non-linear activation function layers between them.

Taking the advantage of the complex artificial neural networks, a deep learning model can interpret complicated real-world instances via the complex functions by thorough transformations through a general-purpose procedure. This feature makes deep learning a relatively expert-independent machine learning method with extremely high prediction accuracy. At present, the deep learning technology offers a series of deep neural network models suitable for various kinds of big data.

6.3 Convolutional Neural Network for Image-based Diagnosis

Image-based diagnosis methods such as histopathology diagnosis by biopsy tissue images, radiology diagnosis by X-ray, CT (computed tomography), MRI (magnetic resonance imaging), and PETS (positron emission tomography) images are commonly applied for the medical decision making process in Ontario healthcare providers and research institutions, which usually play a key role in the most critical cases such as cancer, cerebrovascular accidents, and severe accidental trauma, etc. These kinds of Image-based diagnosis are currently heavily relied on the expertise and experience of the specialists who are easily interfered by external factors such as working environment, pressure, and fatigue etc. Thus, a misdiagnosis is likely to be made and to brings unnecessary loss in proper treatment and unnecessary healthcare expense.

Automatic diagnosis based on AI is a perfect solution for the above issue. However, the conventional machine learning models such as SVM, k-nearest neighbor (kNN), and shallow neural networks with limit hidden neurons / layers are incapable of effectively representing all latent features of the complexity of image pixels and end up with the underfeeding issue. On the other hand, a complex ensemble learning approach which integrates multiple learning models will generate an unsolvable overfeeding issue. Therefore, the available AI technology based on shallow learning models cannot reach an acceptable performance for industrial application, especially for image-based diagnosis.

Convolutional neural networks (CNN) is the state-of-the-art deep learning algorithm that has been successfully applied to many industries such as automatic handwriting recognition, text recognition, natural image recognition and vision recognition with extremely accurate predictions. Compared to other conventional machine learning models, the CNN can be fed with the raw images and automatically discover high and abstract representations by non-linear activation function layers. A CNN model with multiple layers can learn complex functions by thorough transformations through a general-purpose procedure. Therefore, a CNN learning model is expert-independent and renders extremely high classification accuracy with relatively simple data preprocessing procedure. In comparison, the conventional machine learning models are incapable of processing natural image data directly. The complexity of raw image lets the feature extraction difficult, time consuming, and expert knowledge dependent. (See Figure 6.1 and Figure 6.2)

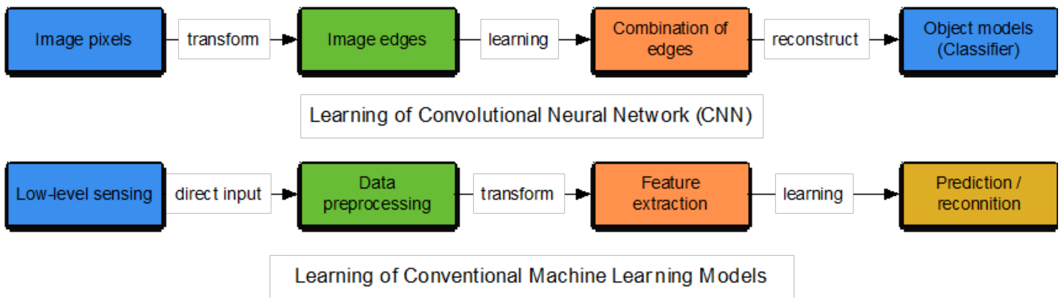
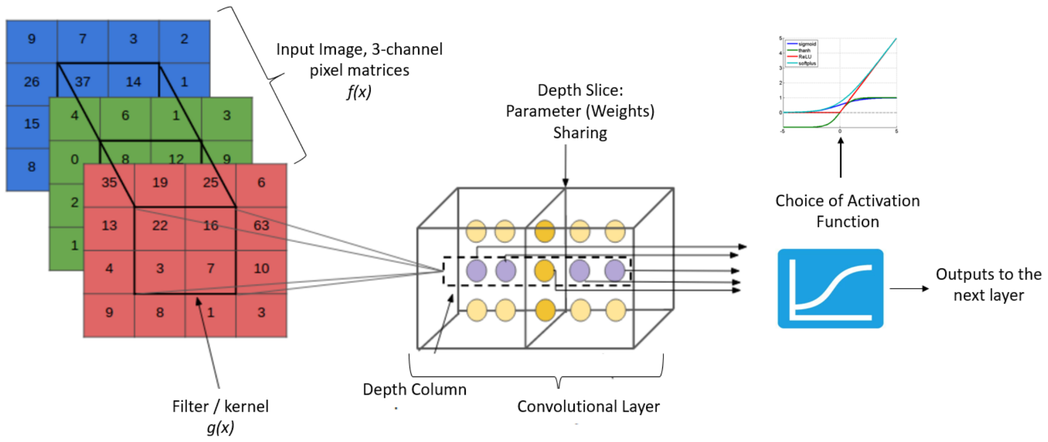


FIGURE 6.1: Comparison between CNN and Shallow learning models



Input Image, 3-channel pixel matrices
f(x)
Depth Slice: Parameter (Weights) Sharing
Choice of Activation Function
Outputs to the next layer
Filter / kernel
g(x)
Depth Column
Convolutional Layer

Figure 6.2: Information Transfer in a CNN Model

6.4 CNN Applications to Malaria Diagnosis

The above merits of CNN make the implementation of CNN a promising solution for the automatic diagnosis of malaria and other epidemic infectious disease whose diagnosis heavily replies on image observation. In our study, an android App with a machine learning classifier is being developed with a CNN classifier trained by the massive images in the image achieve of National Library of Medicine (NLM). A smartphone equipped with the APP can be connected to a light microscope to perform automatic blood cell segmentation, classify the infected and uninfected cells and render the counting directly to the EHR system. This system hopefully provides an efficient and inexpensive solution for malaria field screening and diagnosis.

Since the training set in the study contains over 27 thousand original blood cell images which outnumber all the previous studies [4, 5, 7], the newly trained CNN classifier is expected to have good performance compared to the previous studies. In addition, the CNN architecture also affects the performance. A reasonably complex design helps to improve the total classification performance. In the above experiment, the 18-layer AlexNet-like Malaria Net model has an average classification accuracy at 97.37% in the 27.5K raw image data set and 97.93% in the 110.3K secondary data set with rotated images. In comparison, the 12-layer CIFAR-like Malaria Net model has an average classification accuracy at 96.09% in the 27.5K raw image data set and 96.75% in the 110.3K secondary data set with rotated images. The results imply that the complexity of the CNN model affects the CNN classification performance and a more complex model with more convolutional layers can render more accurate predictions

Deep learning is widely applied in many fields of biomedical research. The major merit of deep learning is that it can discover intricate structure in large data sets and provide extremely accurate predictions. The previous work on the application of deep learning to genomics studies [83] and electronic medical information retrieval [84] also indicates deep learning is applicable for many health-related issues. Therefore, the models designed in this study will be hopefully applied to the other deep learning based applications and computer software to provide efficiency and accurate automatic diagnosis.

In the comparison of an originally trained CNN model and a transfer learning model, the overall prediction accuracy of the originally trained CNN model is approximately 5% higher than the transfer learning model. This implies the homogeneity of the training set directly affects the deep learning accuracy. A CNN classifier trained by homogeneous images works much better than a transfer learning model which is composed of a pre-trained model by heterogeneous images and a conventional machine learning model as the classifier.

Therefore, we confirm from this experiment that CNN provides a good solution for the image based automatic diagnosis. CNN is the state-of-art model for image classification algorithm and is able to provide extremely accurate prediction for relevant image classification with high efficiency.

6.5 Expectations of Deep Learning Applications

Deep learning is the state-of-the-art machine learning model of artificial intelligence (AI) to mimic the information processing of human brain. The advantages of deep learning are its accuracy and reliability after sufficient training, and the overwhelmingly rapid data processing capacity backed by parallel computing over human brain.

The current restriction of deep learning is that the deep learning models need to be trained by a large amount of data to develop good cognitive capacity. Small datasets will be insufficient to train a deep model with its many parameters. As a compromise, the method of transfer learning is introduced where a pre-trained network model is used to extract features that a conventional classifier (e.g. support vector machine, or SVM) can use for fine-tuned classification.

Transfer learning can be used as a shortcut to deep learning where the time for training is saved at the cost of performance, which might be lower but still acceptable. It may be used as a temporary replacement when large training data is not immediately available .The merits of transfer learning include less training time, less computation, and simpler data preprocessing. The experiment outcome shows that using a transfer learning model consisting of a pre-train CNN feature extractor and a linear classifier can still achieve an acceptable performance that can satisfy most of the industrial requirements. Therefore, transfer learning provides an effective strategy for deep learning if massive data is unavailable.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this study, a convolutional neural network model consisting of six convolutional layers is designed to detect the visual patterns of infected red blood cells (RBC) in thin blood smear images. The deep learning model is evaluated by a original malaria RBC image dataset containing 27,578 raw images and a secondary image dataset with 110,312 images. The training set in this experiment outnumbers all the previously reported studies. The model performance is tested by the ten-fold cross validation and compared to a transfer learning model which is trained by the same two datasets.

The result implies that the amount of images of the training set has a direct influence on the performance of the trained classifier. A larger data set certainly improves the accuracy and other performances of the CNN classifier. In addition, the CNN architecture also affects the performance. A reasonably complex design is likely to improve the total classification performance. However, we cannot conclude the more complex the model, the higher accuracy we shall expect because it is also affected by other factors such as the combination of different kinds of layers and the steps for data preprocessing.

The training results of transfer learning indicate that transfer learning can achieve good performance (over 90% of accuracy) that can fulfill most of the industrial requirements. The feature extractor of the transfer learning model is a pre-trained AlexNet trained by the ImageNet from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), it has good cognitive capacity by the sufficient training thus it can act as a pattern encoder for a relatively simple classifier. We expect the performance of transfer learning can be improved when a more appropriate classifier is attached to a fine-tuned CNN feature extractor.

Based on the above findings, we conclude that we successfully develop an automated diagnosis solution based on a deep convolutional neural network model named Malaria Net. The ten-fold cross validation result confirms that the newly developed model has better performance in detecting the malaria infected blood cells than the machine learning model proposed by the US National Library of Medicine in 2015 [85]. A initial result paper on the Malaria Net design has been published on the IEEE international conference in 2016. [86]

7.2 Future Work

Deep learning is the machine learning framework requires high performance computing. New technology such as cloud computing and parallel computing provide one of the ideal solution to implementing deep learning to processing big data. The big data has the four V features: volume, velocity, variety, and veracity.

Deep learning provides the best trade-off for high performance and accuracy, thus it renders accurate results after trained by large volume and variety of data. However, to let a deep learning model acquire high-performance cognitive capacity requires powerful technology and hardware support such as parallel computing, multi-thread processing and GPUs.

In this work, it takes about 30 hours to train the 18-layer Malaria Net with the GPU. However, if using the CPU only, the same training will take a whole week. Thus it indicates that deep learning require good computing resource. Another solution is the application of cloud computing platforms such as Amazon AWS, Microsoft Azure, IBM Watson, and Google Cloud etc. Therefore, a could computing platform with large memory (including GPU RAM) can significantly improve the training performance of deep learning, because more data points can be read in for model training and tuning in large batch size. Larger RAM can save the massive I/O time for data searching and reading. The complexity of deep learning requires to send the data to the trained model by batch mode. Large memory system can effectively reduce the number of batches and increase the batch size, which can also improve the training efficiency of deep learning

On the other hand, deep learning is new AI technology. Thus, it still stays in the academia circle. The development of deep learning products is restricted by giant companies such as Google, IBM, Microsoft, etc. For example, Google publish their deep learning library TensorFlow, which provides various APIs including C++ and Java for software development. Microsoft also produces its python-based cognitive toolkit (CNTK) to implement deep learning on the Windows platform, and provides C library and APIs of CNTK that can be applied to its Visual Studio development platform. The governments of many countries believe that AI will be one of the key driving forces for the next industrial revolution. For example, both the Canadian federal government and the Ontario provincial government believe that AI supported by deep learning and other machine learning methods is one of the pillars of the economic growth strategy of Canada in the short future. The new booming industry is in urgent need of highly qualified personnel trained by both academia and industry.

Based on the research of Malaria Net which is proved to be perform highly reliable classification of malaria infected red blood cells. Our future work will focus on applying the trained malaria net model with high classification accuracy to construct a region-based convolutional neural network (R-CNN) model for object detection and region of interest (ROI) segmentation. [87] The R-CNN model can automatically detect, classify, and count the ROIs in the raw images with multiple interested visual features. (See $Figure$7.1) By counting the total ROIs in a certain amount of image sample and combined with other state-of-the-art knowledge representation technology [88, 89], this new technology will provide a good solution for automatic medical diagnosis and other visual information management and retrieval applications.

Deep learning for visual pattern recognition has a massive demand from both the commercial market and academic communities. For example, face recognition is widely used in government agencies (e.g. military, intelligence, border costumes, etc.) and commercial organizations. Object recognition is commonly applied in image retrieval (e.g. Google image) and automations (e.g. automatic driving). In addition, the available pre-trained models are initially trained by common images such as animals, human, and common objects (e.g. cars). These heterogeneous models are obviously unsuitable to be

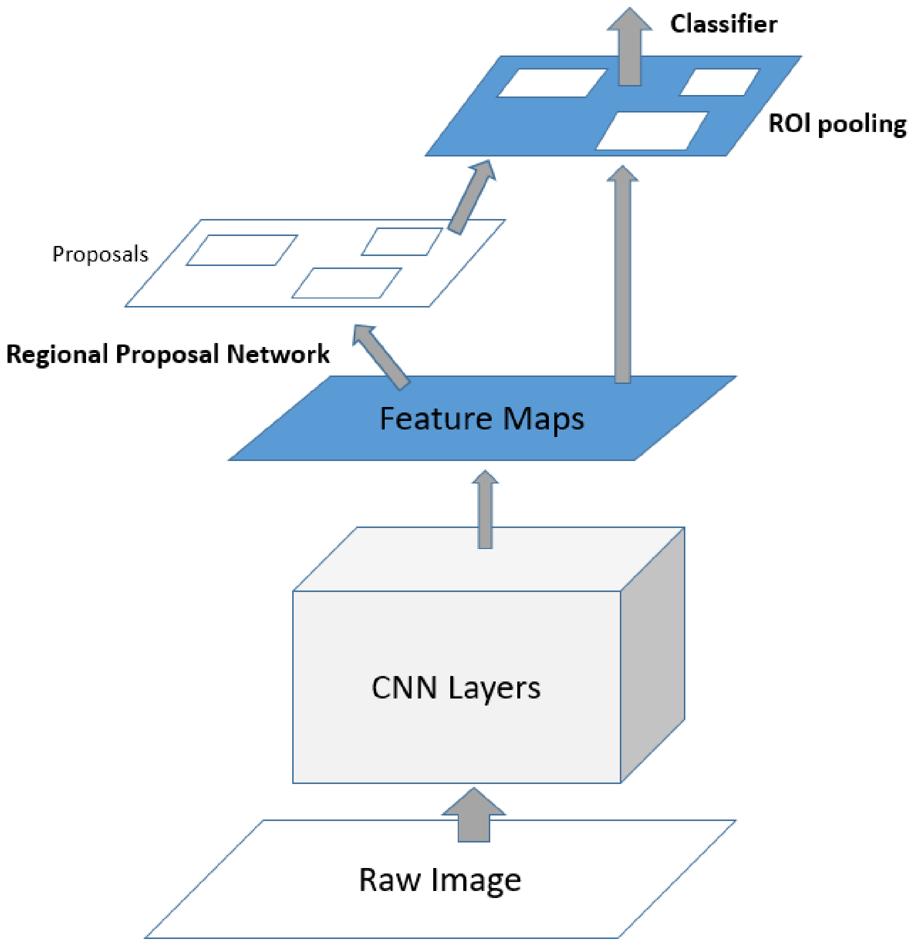


Figure 7.1: A Fast R-CNN model

apply for medical and health proposes. As the response to the above demands, we will initially train a series of CNN models / classifiers for health-related problems, especially for histology and pathology diagnosis and the relevant health decision making. This work will provide strong support for the implementation of deep learning and artificial intelligence to medical informatics and the relevant AI research.

From the year 2016, a group of scientists led by François Chollet from Google develop the Python-based Keras library which acts as the universal interface running on the top of many deep learning libraries including MxNet in R, Deeplearning4J in Java, and Python libraries including Tensorflow by Google, CNTK by Microsoft, and Theano by the University of Montreal. [90] The Keras library provides a bridge to integrate the implementation of deep learning on different platforms with the same coding syntactic style in Python. Since 2017, many institutions and individual developers keep publishing new libraries in different programming languages that support to transfer the trained deep learning model with Keras to different development environments including Java and C. In addition, Google publishes their Android library based on Java to implement deep learning in the Android based smart phones. For example, to transfer the deep learning technology to the most popular mobile platform, we can apply the Xamarin technology with the C-based cross-platform development environment to simultaneously develop mobile applications on the Android, iOS, and Windows phone platform. All the above software provides the necessary technical to implement deep learning in the short future.

Bibliography

[1] WHO. World malaria report 2015, 2015. URL http://www.who.int/malaria/publications/world-malaria-report-2015/en/.

[2] Sonja Mali, S. Patrick Kachur, and Paul M. Arguin. Malaria surveillance—united states, 2010. *MMWR Surveillance Summary*, 61(2):1–17, 2012.

[3] Michael L. Wilson. Malaria rapid diagnostic tests. *Clinical Infectious Diseases: an official publication of the Infectious Diseases Society of America*, 54(11):1637–41, 2012.

[4] Fuyuki Tokumasu, Rick M. Fairhurst, Graciela R. Ostera, Nathaniel J. Brittain, Jeeseong Hwang, Thomas E Wellems, and James A. Dvorak. Band 3 modifications in plasmodium falciparum-infected aa and cc erythrocytes assayed by autocorrelation analysis using quantum dots. *Journal of Cell Science*, 118(5):1091–1098, 2005.

[5] Selena WS Sio, Weiling Sun, Saravana Kumar, Bin Wong, Zeng, Tan Soon, Shan, Ong Sim, Heng, Haruhisa Kikuchi, Yoshiteru Oshima, and Kevin SW Tan. Malariacount: an image analysis-based program for the accurate determination of parasitemia. *Journal of Microbiological Methods*, 68(1):11–18, 2007.

[6] Matthias Elter, Erik Haßlmeyer, and Thorsten Zerfaß. Detection of malaria parasites in thick blood films. In *Proceedings of Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5140–5144. IEEE, 2011.

[7] D.K. Das, A.K. Maiti, and C. Chakraborty. Automated system for characterization and classification of malaria-infected stages using light microscopic images of thin blood smears. *Journal of Microscopy*, 257(3):238–252, 2015.

[8] David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[9] Itamar Arel, Derek C. Rose, and Thomas P. Karnowski. Deep machine learning-a new frontier in artificial intelligence research. *IEEE Computational Intelligence Magazine*, 5(4):13–18, 2010.

[10] Daniel Kersten, Alice J. O'toole, Margaret E. Sereno, David C. Knill, and James A. Anderson. Associative learning of scene parameters from images. *Applied Optics*, 26(23):4999–5006, 1987.

[11] David Marr. Visual information processing: the structure and creation of visual representations. *Philosophical Transactions of Royal Society of London. Series B, Biological Sciences*, 290(1038): 199–218, 1980.

[12] CJ Taylor, TF Cootes, A Lanitis, G Edwards, P Smyth, and AC Kotcheff. Model-based interpretation of complex and variable images. *Philosophical Transactions of Royal Society of London. Series B, Biological Sciences*, 352(1358):1267–1274, 1997.

[13] Sholom M. Weiss, Kulikowski Casimir A., Saul Amarel, and Aran Safir. A model-based method for computer-aided medical decision-making. *Artificial Intelligence*, 11(1-2):145–172, 1978.

[14] Darren Flynn, Daniel J. Nesbitt, Gary A. Ford, Peter McMeekin, Helen Rodgers, Christopher Price, Christian Kray, and Richard G. Thomson. Development of a computerised decision aid for thrombolysis in acute stroke care. *BMC Medical Informatics and Decision Making*, 15(127), 2015.

[15] Hanaa F. Elkhenini, Kourtney J. Davis, Norman D. Stein, John P. New, Mark R. Delderfield, Martin Gibson, Jorgen Vestbo, Ashley Woodcock, and Diar Bakerly, Nawar. Using an electronic medical record (emr) to conduct clinical trials: Salford lung study feasibility. *BMC Medical Informatics and Decision Making*, 15(8), 2015.

[16] M. Nemoto, Y. Masutani, Y. Nomura, S. Hanaoka, S. Miki, T. Yoshikawa, N. Hayashi, and K. Ootomo. Machine learning for computer-aided diagnosis. *Japanese Journal of Medical Physics: an Official Journal of Japan Society of Medical Physics*, 36(1):29–34, 2016.

[17] Marleen de Bruijne. Machine learning approaches in medical image analysis: From detection to diagnosis. *Medical Image Analysis*, 33:94–97, 2016.

[18] Shijun Wang and Ronald M. Summers. Machine learning and radiology. *Medical Image Analysis*, 16(5):933–951, 2012.

[19] Bigong Wang and Li Liang. Recent development of dual-dictionary learning approach in medical image analysis and reconstruction. *Computational and Mathematical Methods in Medicine*, 2015 (152693), 2015.

[20] Ilia Nouretdinov, Sergi G. Costafreda, Gammerman Alexander, Alexey Chervonenkis, Vladimir Vovk, Vladimir Vapnik, and HY Fu, Cynthia. Machine learning classification with confidence: application of transductive conformal predictors to mri-based diagnostic and prognostic markers in depression. *Neuroimage*, 56(2):809–813, 2011.

[21] Jie-Zhi Cheng, Dong Ni, Yi-Hong Chou, Jing Qin, Chang Yeun-Chung Tiu, Chui-Mei, Chiun-Sheng Huang, Dinggang Shen, and Chung-Ming Chen. Computer-aided diagnosis with deep learning architecture: Applications to breast lesions in us images and pulmonary nodules in ct scans. *Scientific Reports*, 6(24454), 2016.

[22] Metin N. Gurcan, Laura E. Boucheron, Ali Can, Anant Madabhushi, Nasir M. Rajpoot, and Bulent Yener. Histopathological image analysis: a review. *IEEE reviews in biomedical engineering*, 2:147–171, 2009.

[23] Birgit Lessmann, Tim W. Nattkemper, Volkmar H. Hans, and Andreas Degenhard. A method for linking computed image features to histological semantics in neuropathology. *Journal of Biomedical Informatics*, 40(6):631–641, 2007.

[24] Scott Vanderbeck, Joseph Bockhorst, Richard Komorowski, David E. Kleiner, and Samer Gawrieh. Automatic classification of white regions in liver biopsies by supervised machine learning. *Human Pathology*, 45(4):785–792, 2014.

[25] Magnus Svensson, Carl, Solveigh Krusekopf, Jörg Lücke, and Marc Thilo, Figge. Automated detection of circulating tumor cells with naive bayesian classifiers. *Cytometry. Part A: the Journal of the International Socieity for Analytical Cytology*, 85(6):501–511, 2014.

[26] B. Gopinath and N. Shanthi. Development of an automated medical diagnosis system for classifying thyroid tumor cells using multiple classifier fusion. *Technology Cencer Research Treatment*, 14(5): 653–662, 2015.

[27] Arkadiusz Gertych, Nathan Ing, Zhaoxuan Ma, Thomas J. Fuchs, Sadri Salman, Sambit Mohanty, Sanica Bhele, Adriana Velásquez-Vacca, Mahul B. Amin, and Beatrice S. Knudsen. Machine learning approaches to analyze histological images of tissues from radical prostatectomies. *Computerized Medical Imaging and Graphics: the Official Journal of the Computerized Medical Imaging Society*, 46:197–208, 2015.

[28] Amarnath Gupta and Ramesh Jain. Visual information retrieval. *Communications of the ACM*, 40 (5):70–79, 1997.

[29] Zheng Ye and Jimmy Xiangji Huang. A learning to rank approach for quality-aware pseudo-relevance feedback. *Journal of the Association for Information Science and Technology*, 67(4): 942–959, 2016.

[30] Xing Tan, Jimmy Xiangji Huang, and Aijun An. Ranking documents through stochastic sampling on bayesian network-based models: A pilot study. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 961–964. ACM, 2016.

[31] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[32] Hajer Ayadi, Mouna Torjmen Khemakhem, Jimmy Xiangji Huang, Mariam Daoud, and Maher Ben Jemaa. Learning to re-rank medical images using a bayesian network-based thesaurus. In *European Conference on Information Retrieval*, pages 160–172. Springer, 2017.

[33] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, et al. Query by image and video content: The qbic system. *Computer*, 28(9):23–32, 1995.

[34] Jeffrey R Bach, Charles Fuller, Amarnath Gupta, Arun Hampapur, Bradley Horowitz, Rich Humphrey, Ramesh C Jain, and Chiao-Fe Shu. Virage image search engine: an open framework

for image management. In *Electronic Imaging: Science & Technology*, pages 76–87. International Society for Optics and Photonics, 1996.

[35] Henning Müller, Nicolas Michoux, David Bandon, and Antoine Geissbuhler. A review of content-based image retrieval systems in medical applications—clinical benefits and future directions. *International Journal of Medical Informatics*, 73(1):1–23, 2004.

[36] Yansong Feng and Mirella Lapata. Automatic caption generation for news images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(4):797–812, 2013.

[37] C Kulikowski, E Ammenwerth, A Bohne, K Ganser, R Haux, P Knaup, C Maier, A Michel, R Singer, and AC Wolff. Medical imaging informatics and medical informatics: Opportunities and constraints. *Methods of Information in Medicine-Methodik der Information in der Medizin*, 41(2): 183, 2002.

[38] Arnold WM Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.

[39] Craig Morioka, Frank Meng, Ricky Taira, James Sayre, Peter Zimmerman, David Ishimitsu, Jimmy Huang, Luyao Shen, and Suzie El-Saden. Automatic classification of ultrasound screening examinations of the abdominal aorta. *Journal of digital imaging*, 29(6):742–748, 2016.

[40] Jun Miao, Jimmy Xiangji Huang, and Jiashu Zhao. Topprf: A probabilistic framework for integrating topic space into pseudo relevance feedback. *ACM Transactions on Information Systems (TOIS)*, 34(4):22, 2016.

[41] Korsuk Sirinukunwattana, Shan E Ahmed Raza, Yee-Wah Tsang, David RJ Snead, Ian A Cree, and Nasir M Rajpoot. Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images. *IEEE Transactions on Medical Imaging*, 35(5):1196–1206, 2016.

[42] Byoung Chul Ko, JiHyeon Lee, and Jae-Yeal Nam. Automatic medical image annotation and keyword-based image retrieval using relevance feedback. *Journal of Digital Imaging*, 25(4):454–465, 2012.

[43] Hajer Ayadi, Mouna Torjmen, Mariam Daoud, Maher Ben Jemaa, and Jimmy Xiangji Huang. Correlating medical-dependent query features with image retrieval models using association rules. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 299–308. ACM, 2013.

[44] Atanaz Babashzadeh, Mariam Daoud, and Jimmy Huang. Using semantic-based association rule mining for improving clinical text retrieval. In *International Conference on Health Information Science*, pages 186–197. Springer, 2013.

[45] Xiangdong An and Jimmy Xiangji Huang. Boosting novelty for biomedical information retrieval through probabilistic latent semantic analysis. In *Proceedings of the 36th international ACM SIGIR conference on research and development in information retrieval*, pages 829–832. ACM, 2013.

[46] Jiashu Zhao, Jimmy Xiangji Huang, Xiaohua Hu, Joseph Kurian, and William Melek. A bayesian-based prediction model for personalized medical health care. In *Bioinformatics and Biomedicine (BIBM), 2012 IEEE International Conference on*, pages 1–4. IEEE, 2012.

[47] Ashnil Kumar, Shane Dyer, Jinman Kim, Changyang Li, Philip HW Leong, Michael Fulham, and Dagan Feng. Adapting content-based image retrieval techniques for the semantic annotation of medical images. *Computerized Medical Imaging and Graphics*, 49:37–45, 2016.

[48] Ruchi Verma, Ajit Tiwari, Sukhwinder Kaur, Grish C. Varshney, and PS Raghava, Gajendra. Identification of proteins secreted by malaria parasite into erythrocyte using svm and pssm profiles. *BMC Bioinformatics*, 16(9):201, 2008.

[49] Rui Kuang, Jianying Gu, Hong Cai, and Yufeng Wang. Improved prediction of malaria degradomes by supervised learning with svm and profile kernel. *Genetica*, 136(1):189–209, 2009.

[50] Kathryn J. Wicht, Jill M. Combrinck, Peter J. Smith, and Timothy J. Egan. Bayesian models trained with hts data for predicting β-haematin inhibition and in vitro antimalarial activity. *Bioorganic Medicinal Chemistry*, 23(16):5210–5217, 2015.

[51] Weiwei Yin, Swetha Garimalla, Alberto Moreno, Mary R. Galinski, and Mark P. Styczynski. A tree-like bayesian structure learning algorithm for small-sample datasets from complex biological model systems. *BMC Systems Biology*, 9(49), 2015.

[52] Luciana Scotti, Hamilton Ishiki, Francisco JB Mendonca, Junior, Marcelo S da Silva, and Marcus T Scotti. Artificial neural network methods applied to drug discovery for neglected diseases. *Combinatorial chemistry high throughput screening*, 18(8):819–829, 2015.

[53] Kumar Das, Dev, Madhumala Ghosh, Mallika Pal, K. Maiti, Asok, and Chandan Chakraborty. Machine learning approach for automated screening of malaria parasite using light microscopic images. *Micron*, 45:97–106, 2013.

[54] Kate Zinszer, Aman D. Verma, Katia Charland, Timothy F. Brewer, John S. Brownstein, Zhuoyu Sun, and David L. Buckeridge. A scoping review of malaria forecasting: past work and future directions. *BMJ Open*, 2:e001992, 2012.

[55] Lok-Won Kim. Deepx: Deep learning accelerator for restricted boltzmann machine artificial neural networks. *IEEE transactions on neural networks and learning systems*, 2017.

[56] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[57] Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37:328–339, 1989.

[58] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324. IEEE, 1998.

[59] Patrice Y. Simard, David Steinkraus, and John C. Platt. Best practices for convolutional neural networks applied to visual document analysis. *In ICDAR*, 3:958–962, 2003.

[60] Régis Vaillant, Christophe Monrocq, and Yann LeCun. Original approach for the localisation of objects in images. In *IEEE Proceedings of Vision, Image and Signal Processing*, pages 245–250. IEEE, 1994.

[61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012.

[62] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proceedings of European conference on computer vision*, pages 818–833. Springer International Publishing, 2014.

[63] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Learning local feature descriptors using convex optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8): 1573–1585, 2014.

[64] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9. IEEE, 2015.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE, 2016.

[66] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[67] Gang Zhang, Jian Yin, Ziping Li, Xiangyang Su, Guozheng Li, and Honglai Zhang. Automated skin biopsy histopathological image annotation using multi-instance representation and learning. *BMC medical genomics*, 6(3):S10, 2013.

[68] Mathworks. Deep learning, 2016. URL http://www.mathworks.com/discovery/deep-learning.html.

[69] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[70] Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011.

[71] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12 (1):145–151, 1999.

[72] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/k2). In *Doklady an SSSR*, volume 269, pages 543–547, 1983.

[73] Maya R Gupta, Samy Bengio, and Jason Weston. Training highly multiclass classifiers. *Journal of Machine Learning Research*, 15(1):1461–1492, 2014.

[74] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[75] T Tieleman and G Hinton. Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. Technical report, Technical report, 2012. 31.

[76] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[77] A. Vedaldi, K. Lenc, and A. Gupta. Matconvnet convolutional neural networks for matlab, 2016. URL http://www.mathworks.com/discovery/deep-learning.html.

[78] Kevin Jarrett, Koray Kavukcuoglu, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Proceedings of International Conference on Computer Vision*, pages 2146–2153, 2009.

[79] MathWorks. Datastore, 2017. URL https://www.mathworks.com/help/matlab/datastore.html.

[80] MathWorks. Imagedatastore, 2017. URL https://www.mathworks.com/help/matlab/ref/imagedatastore-object.html.

[81] MathWorks. Imresize, 2017. URL https://www.mathworks.com/help/images/ref/imresize.html.

[82] Stanford Vision Lab. About imagenet, 2016. URL http://image-net.org/about-overview.

[83] Zhaohui Liang, Xiangji Huang, Jimmy, Xing Zeng, and Gang Zhang. Dl-adr: a novel deep learning model for classifying genomic variants into adverse drug reactions. *BMC Medical Genomics*, 9(2):48, 2016.

[84] Zhaohui Liang, Gang Zhang, Xiangji Huang, Jimmy, and Vivian Hu, Qmming. Deep learning for healthcare decision making with emrs. In *Proceedings of International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 556–559. IEEE, 2014.

[85] Mahdieh Poostchi, Ilker Ersoy, Emile Gordon, Abhisheka Bansal, Kannappan Palaniappan, Susan Pierce, Sameer Antani, George Thoma, and Stefan Jaeger. Image analysis of blood slides for automatic malaria diagnosis. In *Conference Proceedings of 2015 Healthcare Innovations and Point-of-Care Technologies Conference (HICPT 15).*

[86] Zhaohui Liang, Andrew Powell, Ilker Ersoy, Mahdieh Poostchi, Kamolrat Silamut, Kannappan Palaniappan, Peng Guo, Md Amir Hossain, Antani Sameer, Richard James Maude, et al. Cnn-based image analysis for malaria diagnosis. In *Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on*, pages 493–496. IEEE, 2016.

[87] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[88] Guangyou Zhou and Jimmy Xiangji Huang. Modeling and learning distributed word representation with metadata for question retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 29 (6):1226–1239, 2017.

[89] Hajer Ayadi, Mouna Torjmen-Khemakhem, Mariam Daoud, Jimmy Xiangji Huang, and Maher Ben Jemaa. Mining correlations between medically dependent features and image retrieval models for query classification. *Journal of the Association for Information Science and Technology*, 68(5): 1323–1334, 2017.

[90] François Chollet et al. Keras, 2015. URL https://keras.io/.

Appendix A

Implementation of Deep Learning in MATLAB

A.1 General Description

The convolutional neural network (CNN) models (i.e. the 12-layer and the 18-layer Malaria) and the machine learning experiments are implemented by MATLAB 2016a (License: 40458891). The CNN algorithms is implemented via the MatConvNet Toolbox and supported by the Computer Vision System Toolbox, Neural Network Toolbox, and Parallel Computing Toolbox. In this session, the relevant codes are presented in serial of MATLAB scripts (most are in the form of MATLAB functions).

A.2 Install the MatConvNet Toolbox

In this study, we implement the deep learning experiment on the MATLAB platform. To install deep learning with MATLAB, we need to install the necessary software by the following steps:

1. Visit MathWorks via the URL: https://www.mathworks.com/
2. Purchase a MATLAB software package with the Computer Vision System Toolbox, Neural Network Toolbox, and Parallel Computing Toolbox.

3. Visit the MatConvNet community toolbox from the URL:
   http://http://www.vlfeat.org/matconvnet/

4. This instruction assumes you use a WINDOWS machine.

5. Download the package from http://www.vlfeat.org/matconvnet/ . It is recommended to download and install the latest version. The current version is 1.0-beta24.

6. Use the commandline window in MATLAB, create a directory for the MatConvNet toolbox inside the MATLAB directory. In our case, we create a directory called 'MatConvNet' inside the MATLAB directory, then unzip the package.

7. Open MATLAB and use the command-line window.

8. Type run $< MatConvNet > /matlab/vl_setupnn$

9. Run vl_testnn to check if the compiling is successful.

10. To test GPU support (assuming the GPU and the CUDA library have been installed successfully), use the command: $vl_testnn('gpu', true)$

A.3 Compiling the MatConvNet Toolbox

1. To compile for CPU support, use the command: $vl_compilenn$. The use the command: $mex - setup$ to set up the mex files.

2. To compile for GPU support, use the command:
   $vl_compilenn('enableGpu', true)$. If you have multiple versions of the CUDA toolkit, or you want to specify the path to the CUDA toolkit explicitly, then use the command:
   $vl_compilenn('enableGpu', true,' cudaRoot',' /path/NVIDIA/CUDA - 8.0')$

3. To compile for CUDA and cuDNN library support, use the command:
   vl$_compilenn('enableGpu', true,' cudaMethod',' nvcc', ...$
   $'cudaRoot',' /Developer/NVIDIA/CUDA - 6.5', ...$
   $'enableCudnn', true,' cudnnRoot',' local/cudnn - rc2')$

Now the MATLAB for deep learning environment is established. The next step is to create the image data object and set up the deep convolution neural network architecture.

The compiled functions are located in the directory "$/matconvnet-version/matlab/$".

Appendix B

Preparation of the Image Data Object

B.1 Instructions

The Image Data Object is a 4-dimensional matrix created from the imageDatastore Class. The imageDatastore Class can read image from the designated directories can assign the names of the directories as data labels for each category. Since an imageDatastore object only stores the pointers that point to each image inside the collect, it provides an elastic approach to manage the image files, in which the images only read into memory when it is needed.

To prepare the image data for deep learning, we need to implement the following data preprocessing steps:

1. Create an imageDatastore object to read all images from the designated directory, and save them into a 4-dimensional matrix, where the first dimension is the image indices, the second dimension is the width, the third dimension is the height, and the forth dimension is the channel.
2. After the images are converted to the the matrices where the pixels are represented by single integers, we need to resize the image resolution to 44×44, 3 channels. This step can be performed by the ReadFcn method of the imageDataStore Class, where a pre-defined function is implemented.

3. The read-in image are randomized by the shuffle method of the imageDataStore Class. Then the image data are separated into ten folds by an iteration.
4. Finally, the image data separated into ten folds are saved into ten .mat files.
5. The raw image data object will be further pre-processed to generate the imdb data object for training the deep learning model.

B.2 MATLAB Script to Resize Image

```

function Iout = readAndPreprocessImage(filename)

    I = imread(filename);

    % Some images may be grayscale. Replicate the image 3 times to
    % create an RGB image.
    if ismatrix(I)
        I = cat(3,I,I,I);
    end

    % Resize the image as required for the CNN.
    Iout = imresize(I, [44 44]);

    % Note that the aspect ratio is not preserved. In Caltech 101, the
    % object of interest is centered in the image and occupies a
    % majority of the image scene. Therefore, preserving the aspect
    % ratio is not critical. However, for other data sets, it may prove
    % beneficial to preserve the aspect ratio of the original image
    % when resizing.
end
```

B.3 MATLAB Script to Image Transfer and Partition

```

function img_load_mat(path, group)
% Author: Zhaohui Liang, York University, 2016
% use the imageDatastore object to load all the image from the path to a
% .mat image array

% path - the root folder of the images, the name of the classes should be
% the next level of the path
% group - the cell array containing the names of the classes
rootFolder = path;

% define the two-level nominal levels
categories = group;

% create a imageDatastore Object to manage a collection of image files
imds = imageDatastore(fullfile(rootFolder, categories), 'LabelSource',
    'foldernames');

% define the specify the 'ReadFcn' parameter of the object
% by the ReadFcn, all read-in image are resized to 44*44*3
imds.ReadFcn = @(filename)readAndPreprocessImage(filename, 44, 44);

% adjust to balance the number of images in each class
tbl = countEachLabel(imds);
minSetCount = min(tbl{:,2});

% Use splitEachLabel method to trim the set
% the two categories of images are in random locations
imds = splitEachLabel(imds, minSetCount, 'randomize');
imds = shuffle(imds);

% randomly partition the whole data set by 80/20
% trainingSet contains 80% of the whole image set
% testSet contains 20% of the whole image set
% save the data set to a .mat file

% alldata_x is a vector for the image raw data
% alldata_y is a vector for the image labels

% Load the image raw data
all_images = readall(imds)
% Turn the images raw data into vectors and put them in a matrix
alldata_x = zeros(3*44*44,numel(all_images));
for i = 1:numel(all_images)
    alldata_x(:,i) = all_images{i}(:);
end
```

```

% Load the labels of the images
img_number = numel(imds.Files);

alldata_y = zeros(2,img_number);
for i = 1:2
    for j=1:img_number
        if imds.Labels(j) == 'Parasitemic'
            alldata_y(i,j)=1;
        else
            alldata_y(i,j)=0;
        end
    end
end

% Save the raw data and labels into a .mat file
save('Cells_alldata.mat','alldata_x','alldata_y','-v7.3');

% form a 10-fold partition for cross-validation
index_x = randperm(size(alldata_x,1));
index_y = randperm(size(alldata_y,1));

foldSize_x = floor(size(index_x,2) / 10);
foldSize_y = floor(size(index_y,2) / 10);

for k = 1 : 10
    train_x = [];
    train_y = [];
  test_x = [];
  test_y = [];

    testIndex_x = (k-1)*foldSize_x+1:k*foldSize_x;
    testIndex_y = (k-1)*foldSize_y+1:k*foldSize_y;
    test_x = alldata_x(index(1,testIndex_x),:);
    test_y = alldata_y(index(1,testIndex_y),:);

    train_x = alldata_x;
  train_y = alldata_y;

    train_x(index(1,testIndex_x),:) = [];

    train_y(index(1,testIndex_y),:) = [];

    save(strcat('./data/Cells_fold-',num2str(k),'.mat'),...
    'train_x','train_y','test_x','test_y','-v7.3');

end
```

Appendix C

Image Data Preprocessing for Deep Learning

C.1 Instructions

Before the image data matrices are sent to training, we need to perform three steps of data preprocessing:

1. Randomly partition and label the image into the training set, the validation set, and the test set.
2. Perform contrast normalization for the image matrices.
3. Perform data whitening to compute the diagonal eigenvalue of the image matrices.

C.2 MATLAB Script to Create the imdb Data Object with All Data Preprocessing Steps

```
% Date: July 20, 2016
% Author: Zhaohui Liang
% create an imdb object for ten-fold validation

%%load image
disp('loading images from disk...');
```

```
% total number of all images
total_img = 0;
% lables for all images
labels = [];
% labels for each of the 10 fold data set
setCell = {};
% the image matrix holding all images
img_matrix = [];
% global index for the whole data set
index = 1;
disp('preparing image data ...');

for k = 1:10
  dataset{k} =
    load(fullfile('/home/liangz2/MATLAB/matconvnet, ...
    image_preprocess/raw_data_small_rotated', ...
    strcat('img_small_rotated-fold-',num2str(k),'.mat')));
  data{k} = dataset{k}.xImg_matrix;

  % the image array in 44*44*3*N
  % the set of labels in each 1/10 set

  labels_subset = single(dataset{k}.tImages);

  % get the total image number in the current fold
  num = size(data{k},1);
  img_matrix=cat(4,img_matrix,data{k});

%   for i = 1:num
%       img_matrix(:,:,:,index) = data{k}(i,:,:,:);
%% change the image matrix to 44*44*3*N
%       index = index + 1;
%   end

  labels = cat(2, labels, labels_subset);

end

total_img = size(img_matrix,4);

labels = double(labels);
```

```

% add the set labels to the data - setCell
% train - 1, test - 2, validation - 3
%% add labels

disp('adding image set labels ... ');

for k = 1 : 10
      subtotal = size(data{k},1);
      % the total number of the current subset
      % must be changed when k=10
      setCell{k} = zeros(1,total_img);
      % because of the rounding
      % from fold 1 to fold 8, the training set will be greater than 80%
      % in fold 9, the validation set will be greater than 10%
      % in fold 10, the test set will be greater than 10%
      if k==1
         % label the test set
         setCell{k}(1,1:subtotal) = 2;
         % label the validation set
         setCell{k}(1,(subtotal+1):(subtotal*2)) = 3;
         % label the training set
         setCell{k}(1,(subtotal*2+1):total_img)= 1;

      elseif k==2
          setCell{k}(1,(subtotal+1):(subtotal*2)) = 2;
          setCell{k}(1,(subtotal*2+1):(subtotal*3)) = 3;
          setCell{k}(1,(subtotal*3+1):total_img)= 1;
          setCell{k}(1,1:subtotal)= 1;

      elseif k==3
          setCell{k}(1,(subtotal*2+1):(subtotal*3)) = 2;
          setCell{k}(1,(subtotal*3+1):(subtotal*4)) = 3;
          setCell{k}(1,(subtotal*4+1):total_img)= 1;
          setCell{k}(1,1:(subtotal*2))= 1;

      elseif k==4
          setCell{k}(1,(subtotal*3+1):(subtotal*4)) = 2;
          setCell{k}(1,(subtotal*4+1):(subtotal*5)) = 3;
          setCell{k}(1,(subtotal*5+1):total_img)= 1;
          setCell{k}(1,1:(subtotal*3))= 1;
```

```

    elseif k==5
        setCell{k}(1,(subtotal*4+1):(subtotal*5)) = 2;
        setCell{k}(1,(subtotal*5+1):(subtotal*6)) = 3;
        setCell{k}(1,(subtotal*6+1):total_img)= 1;
        setCell{k}(1,1:(subtotal*4))= 1;

    elseif k==6
        setCell{k}(1,(subtotal*5+1):(subtotal*6)) = 2;
        setCell{k}(1,(subtotal*6+1):(subtotal*7)) = 3;
        setCell{k}(1,(subtotal*7+1):total_img)= 1;
        setCell{k}(1,1:(subtotal*5))= 1;

    elseif k==7
        setCell{k}(1,(subtotal*6+1):(subtotal*7)) = 2;
        setCell{k}(1,(subtotal*7+1):(subtotal*8)) = 3;
        setCell{k}(1,(subtotal*8+1):total_img)= 1;
        setCell{k}(1,1:(subtotal*6))= 1;

    elseif k==8
        setCell{k}(1,(subtotal*7+1):(subtotal*8)) = 2;
        setCell{k}(1,(subtotal*8+1):(subtotal*9)) = 3;
        setCell{k}(1,(subtotal*9+1):total_img)= 1;
        setCell{k}(1,1:(subtotal*7))= 1;

    elseif k==9
        setCell{k}(1,(subtotal*8+1):(subtotal*9)) = 2;
        setCell{k}(1,(subtotal*9+1):total_img) = 3;
        setCell{k}(1,1:(subtotal*8))= 1;

    else
    %very important to prevent logical error
        subtotal=size(data{k-1},1);
    %bigger test set
        setCell{k}(1,(subtotal*9+1):total_img) = 2;
        setCell{k}(1,1:subtotal) = 3;
        setCell{k}(1,(subtotal+1):(subtotal*9))=1;
    end
end

%% randomize the image matrix
```

```
disp('randomizing the image set ...');
imageCount = size(img_matrix,4);
ranNum = randperm(imageCount);
newImgArray = [];
newLabels = zeros(1,imageCount);
newSetCell = [];

% free RAM
clear dataset;
clear data;

for i = 1 : 10
    newSetCell{i} = zeros(1,imageCount);
end

for i = 1 : imageCount
        newImgArray(:,:,:,i) = img_matrix(:,:,:,ranNum(1));
        newLabels(i) = labels(ranNum(1));
    for j = 1 : 10
           newSetCell{j}(i)=setCell{j}(ranNum(1));
    end
        ranNum(1)=[];
end

img_matrix = newImgArray;
labels = newLabels;
setCell = newSetCell;
clear newImgArray;
% building imdb
imdb = struct();

%% Constrast Normalization
z = reshape(img_matrix,[],size(img_matrix,4)) ;
z = bsxfun(@minus, z, mean(z,1)) ;
n = std(z,0,1) ;
z = bsxfun(@times, z, mean(n) ./ max(n, 40)) ;
img_matrix = reshape(z, 44, 44, 3, []) ;

%%whiten data
z = reshape(img_matrix,[],size(img_matrix,4)) ;
W = z*z'/size(img_matrix,4) ;
```

```
[V,D] = eig(W) ;
the scale is selected to approximately preserve the norm of W
d2 = diag(D) ;
en = sqrt(mean(d2)) ;
z = V*diag(en./max(sqrt(d2), 10))*V'*z ;
img_matrix = reshape(z, 44, 44, 3, []) ;

% IMPORTANT: the image matrix must be single
img_matrix = single(img_matrix);
imdb.images.set = setCell;
imdb.images.data = img_matrix;
imdb.images.labels = labels;
imdb.meta.sets = {'train', 'test', 'val'} ;
imdb.meta.classes = {'Uninfected','Parasitemic'}';
save('imdb-preprocess-rotated.mat','-struct', 'imdb','-v7.3') ;
```

Appendix D

CNN Model Configuration

D.1 Instructions

To initiate a convolutional neural network (CNN) architecture, we use the MATLAB structure (struct) to defined a net structure. Depending on the types of the layer, we need to defined different parameters in the net structure.

1. The parameters for a convolution layer: name, type, weights, dimension and number of the convolutional kernels, learning rate, stride.
2. The parameters for a pooling layer: name, type, method of pooling, stride.
3. The parameters for a activation (function) layer: name, type.
4. The parameters for a softmax layer: name, type (set to softmaxloss for computing the loss value).
5. The parameters for a fully connected layer: name, type (the same as a convolutional layer), dimension of convolutional kernel is set to be 1×1, the stride is set to 1.
6. call the $vl_simplenn_tidy$ function to convert the configured structure to a CNN model: $vl_simplenn_tidy(net)$.

D.2 The 12-layer Malaria Net

```

function net = cnn_init_18malarianet(varargin)
opts.networkType = 'simplenn' ;
%opts.networkType = 'dagnn' ;
opts = vl_argparse(opts, varargin) ;

disp('Calling 18-layer Malaria Net ...');

lr = [.1 2] ;

% Define network Cells-quick
net.layers = {} ;

% Block 1
net.layers{end+1} = struct('name', 'conv1', ...
                           'type', 'conv', ...
                           'weights', {{0.01*randn(5,5,3,...
                           16, 'single'), zeros(1, 16,...
                           'single')}}, ...
                           'learningRate', lr, ...
                           'stride', 1) ;
% 40*40*16
net.layers{end+1} = struct('name', 'pool_1', ...
                           'type', 'pool', ...
                           'method', 'max', ...
                           'pool', [13 13], ...
                           'stride', 2) ;
%(40-13+1) / 2 = 14  14*14*16
net.layers{end+1} = struct('name', 'relu1', 'type', 'relu') ;

% Block 2
net.layers{end+1} = struct('name', 'conv2', ...
                           'type', 'conv', ...
                           'weights', {{0.01*randn(5,5,16,32,...
                           'single'), zeros(1, 32,...
                           'single')}}, ...
                           'learningRate', lr, ...
                           'stride', 1) ;
% 10*10*32
```

```
net.layers{end+1} = struct('name', 'relu2', 'type', 'relu') ;
net.layers{end+1} = struct('name', 'pool_2', ...
                           'type', 'pool', ...
                           'method', 'average', ...
                           'pool', [5 5], ...
                           'stride', 2) ;
%(10-5+1) / 2 = 3  3*3*32

% Block 3
net.layers{end+1} = struct('name', 'relu3', 'type', 'relu') ;
net.layers{end+1} = struct('name', 'conv3', ...
                           'type', 'conv', ...
                           'weights', {{0.05*randn(3,3,32,64,...
                           'single'), zeros(1,64,'single')}},...
                           'learningRate', lr, ...
                           'stride', 1) ;
% 1*1*64
net.layers{end+1} = struct('name', 'sig1','type', 'sigmoid') ;

% Block 4
net.layers{end+1} = struct('name', 'conv4', ...
                           'type', 'conv', ...
                           'weights', {{0.05*randn(1,1,64,2,...
                           'single'), zeros(1,2,'single')}},...
                           'learningRate', lr, ...
                           'stride', 1) ;
% 1*1*64

% Loss layer
net.layers{end+1} = struct('name', 'softmax', 'type',...
                          'softmaxloss') ;

% Meta parameters
net.meta.inputSize = [44 44 3] ;
net.meta.trainOpts.learningRate = [0.05*ones(1,20) 0.03*ones(1,3)
    0.02*ones(1,2) 0.01*ones(1,5) 0.008*ones(1,5) 0.004*ones(1,5)
    0.002*ones(1,5) 0.001*ones(1,5)] ;
%  net.meta.trainOpts.learningRate = [0.05*ones(1,25) 0.01*ones(1,10)
    0.005*ones(1,10) 0.0005*ones(1,5)] ;
net.meta.trainOpts.weightDecay = 0.0001 ;
net.meta.trainOpts.batchSize = 100 ;
net.meta.trainOpts.numEpochs = numel(net.meta.trainOpts.learningRate) ;

% Fill in default values
net = vl_simplenn_tidy(net) ;

% Switch to DagNN if requested
switch lower(opts.networkType)
  case 'simplenn'
    % done
  case 'dagnn'
    net = dagnn.DagNN.fromSimpleNN(net, 'canonicalNames', true) ;
    net.addLayer('error', dagnn.Loss('loss', 'classerror'), ...
              {'prediction','label'}, 'error') ;
  otherwise
    assert(false) ;
end
```

D.3 The 18-layer Malaria Net

```

function net = cnn_init_18malarianet(varargin)
opts.networkType = 'simplenn' ;
%opts.networkType = 'dagnn' ;
opts = vl_argparse(opts, varargin) ;

disp('Calling 18-layer Malaria Net ...');

lr = [.1 2] ;

% Define network Cells-quick
net.layers = {} ;

% Block 1
net.layers{end+1} = struct('name', 'conv1', ...
                           'type', 'conv', ...
                           'weights', {{0.01*randn(5,5,3,...
                           32, 'single'), zeros(1, 32,...
                           'single')}}, ...
                           'learningRate', lr, ...
                           'stride', 1) ;
```

```
% 40*40*32
net.layers{end+1} = struct('name', 'relu1', 'type', 'relu') ;
net.layers{end+1} = struct('name', 'conv2', ...
                           'type', 'conv', ...
                           'weights', {{0.01*randn(5,5,32,32,...
                           'single'), zeros(1, 32,...
                           'single')}},...
                           'learningRate', lr, ...
                           'stride', 1) ;
% 36*36*32
net.layers{end+1} = struct('name', 'relu2', 'type', 'relu') ;
net.layers{end+1} = struct('name', 'poo1_1', ...
                           'type', 'pool', ...
                           'method', 'max', ...
                           'pool', [5 5], ...
                           'stride', 2) ;
%(36-2+1) / 2 = 16  16*16*32

% Block 2
net.layers{end+1} = struct('name', 'conv3', ...
                           'type', 'conv', ...
                           'weights', {{0.05*randn(5,5,32,64,...
                           'single'), zeros(1,64,'single')}},...
                           'learningRate', lr, ...
                           'stride', 1) ;
% 12*12*64
net.layers{end+1} = struct('name', 'relu3','type', 'relu') ;
net.layers{end+1} = struct('name', 'conv4', ...
                           'type', 'conv', ...
                           'weights', {{0.05*randn(3,3,64,64,...
                           'single'), zeros(1,64,'single')}},...
                           'learningRate', lr, ...
                           'stride', 1) ;
% 10*10*64
net.layers{end+1} = struct('name', 'pool2_3', ...
                           'type', 'pool', ...
                           'method', 'avg', ...
                           'pool', [3 3], ...
                           'stride', 1) ;
% 10-3+1 = 8,  8*8*64

% Block 3
net.layers{end+1} = struct('name', 'conv5', ...
                          'type', 'conv', ...
                          'weights', {{0.05*randn(5,5,64,128,..
                          'single'), zeros(1,128,'single')}},...
                           'learningRate', lr, ...
                           'stride', 1) ;
% 8 - 5 + 1 = 4   4*4*128
net.layers{end+1} = struct('name', 'relu4', 'type', 'relu') ;
net.layers{end+1} = struct('name', 'conv6', ...
                         'type', 'conv', ...
                         'weights', {{0.05*randn(4,4,128,256,...
                         'single'), zeros(1,256,'single')}},...
                         'learningRate', lr, ...
                         'stride', 1) ;
% 4-4+1=1 1*1*256

% Block 4
net.layers{end+1} = struct('name', 'fc1', ...
                         'type', 'conv', ...
                         'weights', {{0.05*randn(1,1,256,256,...
                         'single'), zeros(1,256,'single')}}, ...
                         'learningRate', .1*lr, ...
                         'stride', 1) ;
net.layers{end+1} = struct('name', 'fc2', ...
                         'type', 'conv', ...
                         'weights', {{0.05*randn(1,1,256,256,...
                         'single'), zeros(1,256,'single')}},...
                         'learningRate', .1*lr, ...
                         'stride', 1) ;
net.layers{end+1} = struct('name', 'fc3', ...
                         'type', 'conv', ...
                         'weights', {{0.05*randn(1,1,256,2,...
                         'single'), zeros(1,2,'single')}}, ...
                         'learningRate', .1*lr, ...
                         'stride', 1) ;
net.layers{end+1} = struct('name', 'sigmoid1','type',...
                         'sigmoid') ;

% Loss layer
net.layers{end+1} = struct('name', 'softmax', 'type',...
                           'softmaxloss') ;

% Meta parameters
net.meta.inputSize = [44 44 3] ;
net.meta.trainOpts.learningRate = [0.05*ones(1,20) 0.03*ones(1,3)
    0.02*ones(1,2) 0.01*ones(1,5) 0.008*ones(1,5) 0.004*ones(1,5)
    0.002*ones(1,5) 0.001*ones(1,5)] ;
%  net.meta.trainOpts.learningRate = [0.05*ones(1,25) 0.01*ones(1,10)
    0.005*ones(1,10) 0.0005*ones(1,5)] ;
net.meta.trainOpts.weightDecay = 0.0001 ;
net.meta.trainOpts.batchSize = 100 ;
net.meta.trainOpts.numEpochs = numel(net.meta.trainOpts.learningRate) ;

% Fill in default values
net = vl_simplenn_tidy(net) ;

% Switch to DagNN if requested
switch lower(opts.networkType)
  case 'simplenn'
    % done
  case 'dagnn'
    net = dagnn.DagNN.fromSimpleNN(net, 'canonicalNames', true) ;
    net.addLayer('error', dagnn.Loss('loss', 'classerror'), ...
             {'prediction','label'}, 'error') ;
  otherwise
    assert(false) ;
end
```

Appendix E

MATLAB Script for CNN Model Training

E.1 Instructions

We use the compiled functions in the MatConvNet toolbox to implement training for the configured CNN model. The CNN network optimization method is stochastic gradient descent (SGD). The initial learning rate starts from 0.001, the maximal number of training epochs is set to 300. The training batch size is set to 512 images given the dimension of the images is $44 \times 44 \times 3$ and the GPU has 2 GB RAM. During the training process, the loss error computed from each training epoch will be displayed on a line plot to show the learning curve of the model compared with the validation loss error.

E.2 MATLAB Script to Implement CNN Training

```

function [net, stats] = cnn_train(net, imdb, getBatch, varargin)
%CNN_TRAIN  An implementation of SGD for training CNNs
%    CNN_TRAIN() applies stochastic gradient descent with
%    momentum to train a CNN. It can be used
%    with different datasets and tasks by providing a suitable
%    getBatch function.
%
%    The function automatically restarts after each training epoch by
%     checkpointing.
%
%     The function supports training on CPU or on one or more GPUs
%     (specify the list of GPU IDs in the `gpus` option).

% This file is part of the VLFeat library and is made available under
% the terms of the BSD license (see the COPYING file).
addpath(fullfile(vl_rootnn, 'malaria'));
opts.expDir = fullfile('data','exp') ;
opts.continue = true ;
opts.batchSize = 256 ;
opts.numSubBatches = 1 ;
opts.train = [] ;
opts.val = [] ;
opts.gpus = [] ;
opts.epochSize = inf;
opts.prefetch = false ;
opts.numEpochs = 300 ;
opts.learningRate = 0.001 ;
opts.weightDecay = 0.0005 ;

opts.solver = [] ;  % Empty array means use the default SGD solver
[opts, varargin] = vl_argparse(opts, varargin) ;
if ~isempty(opts.solver)
  assert(isa(opts.solver, 'function_handle') && nargout(opts.solver) ==
     2,...
    'Invalid solver; expected a function handle with two outputs.') ;
  % Call without input arguments, to get default options
  opts.solverOpts = opts.solver() ;
end

opts.momentum = 0.9 ;
opts.saveSolverState = true ;
opts.nesterovUpdate = false ;
opts.randomSeed = 0 ;
opts.memoryMapFile = fullfile(tempdir, 'matconvnet.bin') ;
opts.profile = false ;
opts.parameterServer.method = 'mmap' ;
opts.parameterServer.prefix = 'mcn' ;

opts.conserveMemory = true ;
opts.backPropDepth = +inf ;
opts.sync = false ;
opts.cudnn = true ;
opts.errorFunction = 'multiclass' ;
opts.errorLabels = {} ;
opts.plotDiagnostics = false ;
opts.plotStatistics = true;
opts.postEpochFn = [] ;  % postEpochFn(net,params,state) called after each epoch; can return a new learning rate, 0 to stop, [] for no change
opts = vl_argparse(opts, varargin) ;

if ~exist(opts.expDir, 'dir'), mkdir(opts.expDir) ; end
if isempty(opts.train), opts.train = find(imdb.images.set==1) ; end
if isempty(opts.val), opts.val = find(imdb.images.set==2) ; end
if isscalar(opts.train) && isnumeric(opts.train) && isnan(opts.train)
  opts.train = [] ;
end
if isscalar(opts.val) && isnumeric(opts.val) && isnan(opts.val)
  opts.val = [] ;
end

% -------------------------------------------------------------------------
%                                                            Initialization
% -------------------------------------------------------------------------

net = vl_simplenn_tidy(net); % fill in some eventually missing values
net.layers{end-1}.precious = 1; % do not remove predictions, used for error
vl_simplenn_display(net, 'batchSize', opts.batchSize) ;

evaluateMode = isempty(opts.train) ;
if ~evaluateMode
  for i=1:numel(net.layers)
    J = numel(net.layers{i}.weights) ;
    if ~isfield(net.layers{i}, 'learningRate')
      net.layers{i}.learningRate = ones(1, J) ;
    end
    if ~isfield(net.layers{i}, 'weightDecay')
      net.layers{i}.weightDecay = ones(1, J) ;
    end
  end
end

% setup error calculation function
hasError = true ;
if isstr(opts.errorFunction)
  switch opts.errorFunction
    case 'none'
      opts.errorFunction = @error_none ;
      hasError = false ;
    case 'multiclass'
      opts.errorFunction = @error_multiclass ;
      if isempty(opts.errorLabels), opts.errorLabels = {'top1err', 'top5err'} ; end
    case 'binary'
      opts.errorFunction = @error_binary ;
      if isempty(opts.errorLabels), opts.errorLabels = {'binerr'} ; end
    otherwise
      error('Unknown error function ''%s''.', opts.errorFunction) ;
  end
end

state.getBatch = getBatch ;
stats = [] ;

% -------------------------------------------------------------------------
%                                                         Train and validate
% -------------------------------------------------------------------------

modelPath = @(ep) fullfile(opts.expDir, sprintf('net-epoch-%d.mat', ep));
modelFigPath = fullfile(opts.expDir, 'net-train.pdf') ;

start = opts.continue * findLastCheckpoint(opts.expDir) ;
if start >= 1
  fprintf('%s: resuming by loading epoch %d\n', mfilename, start) ;
  [net, state, stats] = loadState(modelPath(start)) ;
else
  state = [] ;
end

for epoch=start+1:opts.numEpochs

  % Set the random seed based on the epoch and opts.randomSeed.
  % This is important for reproducibility, including when training
  % is restarted from a checkpoint.

  rng(epoch + opts.randomSeed) ;
  prepareGPUs(opts, epoch == start+1) ;

  % Train for one epoch.
  params = opts ;
  params.epoch = epoch ;
  params.learningRate = opts.learningRate(min(epoch,
    numel(opts.learningRate))) ;
  params.train = opts.train(randperm(numel(opts.train))) ; % shuffle
  params.train = params.train(1:min(opts.epochSize, numel(opts.train)));
  params.val = opts.val(randperm(numel(opts.val))) ;
  params.imdb = imdb ;
  params.getBatch = getBatch ;

  if numel(params.gpus) <= 1
    [net, state] = processEpoch(net, state, params, 'train') ;
    [net, state] = processEpoch(net, state, params, 'val') ;
    if ~evaluateMode
      saveState(modelPath(epoch), net, state) ;
    end
    lastStats = state.stats ;
  else
    spmd
      [net, state] = processEpoch(net, state, params, 'train') ;
      [net, state] = processEpoch(net, state, params, 'val') ;
      if labindex == 1 && ~evaluateMode
        saveState(modelPath(epoch), net, state) ;
      end
      lastStats = state.stats ;
    end
    lastStats = accumulateStats(lastStats) ;
  end

  stats.train(epoch) = lastStats.train ;
  stats.val(epoch) = lastStats.val ;
  clear lastStats ;
  if ~evaluateMode
    saveStats(modelPath(epoch), stats) ;
  end

  if params.plotStatistics
    switchFigure(1) ; clf ;
    plots = setdiff(...
      cat(2,...
      fieldnames(stats.train)', ...
      fieldnames(stats.val)'), {'num', 'time'}) ;
    for p = plots
      p = char(p) ;
      values = zeros(0, epoch) ;
      leg = {} ;
      for f = {'train', 'val'}
        f = char(f) ;
        if isfield(stats.(f), p)
          tmp = [stats.(f).(p)] ;
          values(end+1,:) = tmp(1,:)' ;
          leg{end+1} = f ;
        end
      end
      subplot(1,numel(plots),find(strcmp(p,plots))) ;
      plot(1:epoch, values','o-') ;
      xlabel('epoch') ;
      title(p) ;
      legend(leg{:}) ;
      grid on ;
    end
    drawnow ;
    print(1, modelFigPath, '-dpdf') ;
  end

  if ~isempty(opts.postEpochFn)
    if nargout(opts.postEpochFn) == 0
```

```
      opts.postEpochFn(net, params, state) ;
    else
      lr = opts.postEpochFn(net, params, state) ;
      if ~isempty(lr), opts.learningRate = lr; end
      if opts.learningRate == 0, break; end
    end
  end
end

% With multiple GPUs, return one copy
if isa(net, 'Composite'), net = net{1} ; end

%
    -------------------------------------------------------------------------
function err = error_multiclass(params, labels, res)
%
    -------------------------------------------------------------------------
predictions = gather(res(end-1).x) ;
[~,predictions] = sort(predictions, 3, 'descend') ;

% be resilient to badly formatted labels
if numel(labels) == size(predictions, 4)
  labels = reshape(labels,1,1,1,[]) ;
end

% skip null labels
mass = single(labels(:,:,1,:) > 0) ;
if size(labels,3) == 2
  % if there is a second channel in labels, used it as weights
  mass = mass .* labels(:,:,2,:) ;
  labels(:,:,2,:) = [] ;
end

m = min(5, size(predictions,3)) ;

error = ~bsxfun(@eq, predictions, labels) ;
err(1,1) = sum(sum(sum(mass .* error(:,:,1,:)))) ;
err(2,1) = sum(sum(sum(mass .* min(error(:,:,1:m,:),[],3)))) ;

%
    -------------------------------------------------------------------------
```

```
function err = error_binary(params, labels, res)
% -------------------------------------------------------------------------
predictions = gather(res(end-1).x) ;
error = bsxfun(@times, predictions, labels) < 0 ;
err = sum(error(:)) ;

% -------------------------------------------------------------------------
function err = error_none(params, labels, res)
% -------------------------------------------------------------------------
err = zeros(0,1) ;

% -------------------------------------------------------------------------
function [net, state] = processEpoch(net, state, params, mode)
% -------------------------------------------------------------------------
% Note that net is not strictly needed as an output argument as net
% is a handle class. However, this fixes some aliasing issue in the
% spmd caller.

% initialize with momentum 0
if isempty(state) || isempty(state.solverState)
  for i = 1:numel(net.layers)
    state.solverState{i} = cell(1, numel(net.layers{i}.weights)) ;
    state.solverState{i}(:) = {0} ;
  end
end

% move CNN  to GPU as needed
numGpus = numel(params.gpus) ;
if numGpus >= 1
  net = vl_simplenn_move(net, 'gpu') ;
  for i = 1:numel(state.solverState)
    for j = 1:numel(state.solverState{i})
      s = state.solverState{i}{j} ;
      if isnumeric(s)
        state.solverState{i}{j} = gpuArray(s) ;
      elseif isstruct(s)
```

```
        state.solverState{i}{j} = structfun(@gpuArray, s,
      'UniformOutput', false) ;
      end
    end
  end
end
if numGpus > 1
  parserv = ParameterServer(params.parameterServer) ;
  vl_simplenn_start_parserv(net, parserv) ;
else
  parserv = [] ;
end

% profile
if params.profile
  if numGpus <= 1
    profile clear ;
    profile on ;
  else
    mpiprofile reset ;
    mpiprofile on ;
  end
end

subset = params.(mode) ;
num = 0 ;
stats.num = 0 ; % return something even if subset = []
stats.time = 0 ;
adjustTime = 0 ;
res = [] ;
error = [] ;

start = tic ;
for t=1:params.batchSize:numel(subset)
  fprintf('%s: epoch %02d: %3d/%3d:', mode, params.epoch, ...
          fix((t-1)/params.batchSize)+1,
      ceil(numel(subset)/params.batchSize)) ;
  batchSize = min(params.batchSize, numel(subset) - t + 1) ;

  for s=1:params.numSubBatches
    % get this image batch and prefetch the next
    batchStart = t + (labindex-1) + (s-1) * numlabs ;
    batchEnd = min(t+params.batchSize-1, numel(subset)) ;
    batch = subset(batchStart : params.numSubBatches * numlabs :
   batchEnd) ;
    num = num + numel(batch) ;
    if numel(batch) == 0, continue ; end

    [im, labels] = params.getBatch(params.imdb, batch) ;

    if params.prefetch
      if s == params.numSubBatches
        batchStart = t + (labindex-1) + params.batchSize ;
        batchEnd = min(t+2*params.batchSize-1, numel(subset)) ;
      else
        batchStart = batchStart + numlabs ;
      end
      nextBatch = subset(batchStart : params.numSubBatches * numlabs :
   batchEnd) ;
      params.getBatch(params.imdb, nextBatch) ;
    end

    if numGpus >= 1
      im = gpuArray(im) ;
    end

    if strcmp(mode, 'train')
      dzdy = 1 ;
      evalMode = 'normal' ;
    else
      dzdy = [] ;
      evalMode = 'test' ;
    end
    net.layers{end}.class = labels ;
    res = vl_simplenn(net, im, dzdy, res, ...
                      'accumulate', s ~= 1, ...
                      'mode', evalMode, ...
                      'conserveMemory', params.conserveMemory, ...
                      'backPropDepth', params.backPropDepth, ...
                      'sync', params.sync, ...
                      'cudnn', params.cudnn, ...
                      'parameterServer', parserv, ...
```

```
                        'holdOn', s < params.numSubBatches) ;

    % accumulate errors
    error = sum([error, [...
      sum(double(gather(res(end).x))) ;
      reshape(params.errorFunction(params, labels, res),[],1) ; ]],2) ;
  end

  % accumulate gradient
  if strcmp(mode, 'train')
    if ~isempty(parserv), parserv.sync() ; end
    [net, res, state] = accumulateGradients(net, res, state, params,
   batchSize, parserv) ;
  end

  % get statistics
  time = toc(start) + adjustTime ;
  batchTime = time - stats.time ;
  stats = extractStats(net, params, error / num) ;
  stats.num = num ;
  stats.time = time ;
  currentSpeed = batchSize / batchTime ;
  averageSpeed = (t + batchSize - 1) / time ;
  if t == 3*params.batchSize + 1
    % compensate for the first three iterations, which are outliers
    adjustTime = 4*batchTime - time ;
    stats.time = time + adjustTime ;
  end

  fprintf(' %.1f (%.1f) Hz', averageSpeed, currentSpeed) ;
  for f = setdiff(fieldnames(stats)', {'num', 'time'})
    f = char(f) ;
    fprintf(' %s: %.3f', f, stats.(f)) ;
  end
  fprintf('\n') ;

  % collect diagnostic statistics
  if strcmp(mode, 'train') && params.plotDiagnostics
    switchFigure(2) ; clf ;
    diagn = [res.stats] ;
    diagnvar = horzcat(diagn.variation) ;
    diagnpow = horzcat(diagn.power) ;
    subplot(2,2,1) ; barh(diagnvar) ;
    set(gca,'TickLabelInterpreter', 'none', ...
      'YTick', 1:numel(diagnvar), ...
      'YTickLabel',horzcat(diagn.label), ...
      'YDir', 'reverse', ...
      'XScale', 'log', ...
      'XLim', [1e-5 1], ...
      'XTick', 10.^(-5:1)) ;
    grid on ; title('Variation');
    subplot(2,2,2) ; barh(sqrt(diagnpow)) ;
    set(gca,'TickLabelInterpreter', 'none', ...
      'YTick', 1:numel(diagnpow), ...
      'YTickLabel',{diagn.powerLabel}, ...
      'YDir', 'reverse', ...
      'XScale', 'log', ...
      'XLim', [1e-5 1e5], ...
      'XTick', 10.^(-5:5)) ;
    grid on ; title('Power');
    subplot(2,2,3); plot(squeeze(res(end-1).x)) ;
    drawnow ;
  end
end

% Save back to state.
state.stats.(mode) = stats ;
if params.profile
  if numGpus <= 1
    state.prof.(mode) = profile('info') ;
    profile off ;
  else
    state.prof.(mode) = mpiprofile('info');
    mpiprofile off ;
  end
end
if ~params.saveSolverState
  state.solverState = [] ;
else
  for i = 1:numel(state.solverState)
    for j = 1:numel(state.solverState{i})
      s = state.solverState{i}{j} ;
      if isnumeric(s)
        state.solverState{i}{j} = gather(s) ;
      elseif isstruct(s)
        state.solverState{i}{j} = structfun(@gather, s, 'UniformOutput', false) ;
      end
    end
  end
end

net = vl_simplenn_move(net, 'cpu') ;

% -------------------------------------------------------------------------
function [net, res, state] = accumulateGradients(net, res, state, params, batchSize, parserv)
% -------------------------------------------------------------------------
numGpus = numel(params.gpus) ;
otherGpus = setdiff(1:numGpus, labindex) ;

for l=numel(net.layers):-1:1
  for j=numel(res(l).dzdw):-1:1

    if ~isempty(parserv)
      tag = sprintf('l%d_%d',l,j) ;
      parDer = parserv.pull(tag) ;
    else
      parDer = res(l).dzdw{j}  ;
    end

    if j == 3 && strcmp(net.layers{l}.type, 'bnorm')
      % special case for learning bnorm moments
      thisLR = net.layers{l}.learningRate(j) ;
      net.layers{l}.weights{j} = vl_taccum(...
        1 - thisLR, ...
        net.layers{l}.weights{j}, ...
        thisLR / batchSize, ...
        parDer) ;
    else
      % Standard gradient training.
      thisDecay = params.weightDecay * net.layers{l}.weightDecay(j) ;
      thisLR = params.learningRate * net.layers{l}.learningRate(j) ;

      if thisLR>0 || thisDecay>0
        % Normalize gradient and incorporate weight decay.
        parDer = vl_taccum(1/batchSize, parDer, ...
                           thisDecay, net.layers{l}.weights{j}) ;

        if isempty(params.solver)
          % Default solver is the optimised SGD.
          % Update momentum.
          state.solverState{l}{j} = vl_taccum(...
            params.momentum, state.solverState{l}{j}, ...
            -1, parDer) ;

          % Nesterov update (aka one step ahead).
          if params.nesterovUpdate
            delta = params.momentum * state.solverState{l}{j} - parDer ;
          else
            delta = state.solverState{l}{j} ;
          end

          % Update parameters.
          net.layers{l}.weights{j} = vl_taccum(...
            1, net.layers{l}.weights{j}, ...
            thisLR, delta) ;

        else
          % call solver function to update weights
          [net.layers{l}.weights{j}, state.solverState{l}{j}] = ...
            params.solver(net.layers{l}.weights{j}, 
state.solverState{l}{j}, ...
            parDer, params.solverOpts, thisLR) ;
        end
      end
    end

    % if requested, collect some useful stats for debugging
    if params.plotDiagnostics
      variation = [] ;
      label = '' ;
      switch net.layers{l}.type
        case {'conv','convt'}
          if isnumeric(state.solverState{l}{j})
            variation = thisLR * mean(abs(state.solverState{l}{j}(:))) ;
          end
          power = mean(res(l+1).x(:).^2) ;
          if j == 1 % fiters
            base = mean(net.layers{l}.weights{j}(:).^2) ;
            label = 'filters' ;
          else % biases
            base = sqrt(power) ;%mean(abs(res(l+1).x(:))) ;
            label = 'biases' ;
          end
          variation = variation / base ;
          label = sprintf('%s_%s', net.layers{l}.name, label) ;
      end
      res(l).stats.variation(j) = variation ;
      res(l).stats.power = power ;
      res(l).stats.powerLabel = net.layers{l}.name ;
      res(l).stats.label{j} = label ;
    end
  end
end

%
    -------------------------------------------------------------------------
function stats = accumulateStats(stats_)
%
    -------------------------------------------------------------------------

for s = {'train', 'val'}
  s = char(s) ;
  total = 0 ;

  % initialize stats stucture with same fields and same order as
  % stats_{1}
  stats__ = stats_{1} ;
  names = fieldnames(stats__.(s))' ;
  values = zeros(1, numel(names)) ;
  fields = cat(1, names, num2cell(values)) ;
  stats.(s) = struct(fields{:}) ;

  for g = 1:numel(stats_)
    stats__ = stats_{g} ;
    num__ = stats__.(s).num ;
    total = total + num__ ;

    for f = setdiff(fieldnames(stats__.(s))', 'num')
      f = char(f) ;
      stats.(s).(f) = stats.(s).(f) + stats__.(s).(f) * num__ ;

      if g == numel(stats_)
        stats.(s).(f) = stats.(s).(f) / total ;
      end
    end
  end
  stats.(s).num = total ;
end

%
    -------------------------------------------------------------------------
function stats = extractStats(net, params, errors)
%
    -------------------------------------------------------------------------
stats.objective = errors(1) ;
for i = 1:numel(params.errorLabels)
  stats.(params.errorLabels{i}) = errors(i+1) ;
end

%
    -------------------------------------------------------------------------
function saveState(fileName, net, state)
%
    -------------------------------------------------------------------------
save(fileName, 'net', 'state') ;

%
    -------------------------------------------------------------------------
function saveStats(fileName, stats)
%
    -------------------------------------------------------------------------
if exist(fileName)
```

```
  save(fileName, 'stats', '-append') ;
else
  save(fileName, 'stats') ;
end

% --------------------------------------------------------------------------
function [net, state, stats] = loadState(fileName)
% --------------------------------------------------------------------------
load(fileName, 'net', 'state', 'stats') ;
net = vl_simplenn_tidy(net) ;
if isempty(whos('stats'))
  error('Epoch ''%s'' was only partially saved. Delete this file and try
   again.', ...
        fileName) ;
end

% --------------------------------------------------------------------------
function epoch = findLastCheckpoint(modelDir)
% --------------------------------------------------------------------------
list = dir(fullfile(modelDir, 'net-epoch-*.mat')) ;
tokens = regexp({list.name}, 'net-epoch-([\d]+).mat', 'tokens') ;
epoch = cellfun(@(x) sscanf(x{1}{1}, '%d'), tokens) ;
epoch = max([epoch 0]) ;

% --------------------------------------------------------------------------
function switchFigure(n)
% --------------------------------------------------------------------------
if get(0,'CurrentFigure') ~= n
  try
    set(0,'CurrentFigure',n) ;
  catch
    figure(n) ;
  end
end

```

```
%
    --------------------------------------------------------------------------
function clearMex()
%
    --------------------------------------------------------------------------
%clear vl_tmove vl_imreadjpeg ;
disp('Clearing mex files') ;
clear mex ;
clear vl_tmove vl_imreadjpeg ;

%
    --------------------------------------------------------------------------
function prepareGPUs(params, cold)
%
    --------------------------------------------------------------------------
numGpus = numel(params.gpus) ;
if numGpus > 1
  % check parallel pool integrity as it could have timed out
  pool = gcp('nocreate') ;
  if ~isempty(pool) && pool.NumWorkers ~= numGpus
    delete(pool) ;
  end
  pool = gcp('nocreate') ;
  if isempty(pool)
    parpool('local', numGpus) ;
    cold = true ;
  end
end
if numGpus >= 1 && cold
  fprintf('%s: resetting GPU\n', mfilename) ;
  clearMex() ;
  if numGpus == 1
    disp(gpuDevice(params.gpus)) ;
  else
    spmd
      clearMex() ;
      disp(gpuDevice(params.gpus(labindex))) ;
    end
  end
end
```

Appendix F

MATLAB Script to Implement Ten-fold Cross Validation

F.1 Instructions

The imdb image data object has been separated into ten subsets in the data preprocessing stage. To implement the ten-fold cross validation, we can use the iteration code to extract the imdb.images.set label from the imdb object where '1' denotes the training set, '2' denotes the test set, and '3' denotes the validation set.

The training set is used to tune the CNN model. The validation set is used to suppress over-fitting. And the test set is used for trained model evaluation. Note that the test set will not be used during the training stage.

Before execute the train procedure, we need to define several parameters for the training:

1. To define the *opts.modelType* for which model will be trained.
2. To define the path to the imdb data object with the field *opts.imdbPath*.
3. To define the training model by the field *opts.networkType*, where the 'simplenn' is to use the SGD method, and the 'dagnn' is to use the directed acyclic graph (DAG) topology method.
4. To define the 'fold' iteration to control which fold of training will be performed.

F.2 CNN Training with Cross Validation

```

function [net, info] = cnn_cells_cross_validate(varargins)

varargins = {};

run(fullfile(fileparts(mfilename('fullpath')),...
  '..', '..', 'matlab', 'vl_setupnn.m')) ;

opts.modelType = 'malaria18' ;
[opts, varargins] = vl_argparse(opts, varargins) ;

opts.expDir = sprintf('.\data\%s', opts.modelType) ;
[opts, varargins] = vl_argparse(opts, varargins) ;

opts.imdbPath = 'imdb-small-unrotated.mat';     % give the full path of
    your imdb.mat file
opts.networkType = 'simplenn' ;
%opts.networkType = 'dagnn' ;
opts.train = struct() ;
opts = vl_argparse(opts, varargins) ;

if ~isfield(opts.train, 'gpus'), opts.train.gpus = [1]; end;

%
    -------------------------------------------------------------------------
%                                                        Prepare model and
    data
%
    -------------------------------------------------------------------------

switch opts.modelType
  case 'malaria12'
    net = cnn_init_12malarianet('networkType', opts.networkType);
  case 'malaria18'
    net = cnn_init_18malarianet('networkType', opts.networkType) ;
  otherwise
    error('Unknown model type ''%s''.', opts.modelType) ;
end

```

```
imdb = load(opts.imdbPath) ;

net.meta.classes.name = imdb.meta.classes(:)';

% ------------------------------------------------------------------------
% Train
% ------------------------------------------------------------------------

switch opts.networkType
  case 'simplenn', trainfn = @cnn_train ;
  case 'dagnn', trainfn = @cnn_train_dag ;
end

baseDir = opts.expDir;

setCells = imdb.images.set;

for fold = 1 : 1

  opts.expDir = strcat(baseDir,'\fold-',num2str(fold));

  if ~exist(opts.expDir)
        mkdir(opts.expDir);
    end

    imdb.images.set = setCells{fold};

  [net, info] = trainfn(net, imdb, getBatch(opts), ...
    'expDir', opts.expDir, ...
    net.meta.trainOpts, ...
    opts.train, ...
    'val', find(imdb.images.set == 3)) ;
end

imdb.images.set = setCells;

% ------------------------------------------------------------------------
```

```
function fn = getBatch(opts)
% --------------------------------------------------------------------------
switch lower(opts.networkType)
  case 'simplenn'
    fn = @(x,y) getSimpleNNBatch(x,y) ;
  case 'dagnn'
    bopts = struct('numGpus', numel(opts.train.gpus)) ;
    fn = @(x,y) getDagNNBatch(bopts,x,y) ;
end

% --------------------------------------------------------------------------
function [images, labels] = getSimpleNNBatch(imdb, batch)
% --------------------------------------------------------------------------
disp(strcat('batch',num2str(batch)));

images = imdb.images.data(:,:,:,batch) ;
labels = imdb.images.labels(1,batch) ;
if rand > 0.5, images=fliplr(images) ; end

% --------------------------------------------------------------------------
function inputs = getDagNNBatch(opts, imdb, batch)
% --------------------------------------------------------------------------
images = imdb.images.data(:,:,:,batch) ;
labels = imdb.images.labels(1,batch) ;
if rand > 0.5, images=fliplr(images) ; end
if opts.numGpus > 0
  images = gpuArray(images) ;
end
inputs = {'input', images, 'label', labels} ;
```

Appendix G

MATLAB Script for Model Evaluation

G.1 Instructions

This script is to perform the model evaluation of the trained CNN models. The trained CNN models from every training epoch are automatically saved as a .mat file. To use a trained to predict the label of a new image, we need to call the $vl_simplenn$ function with a load model by the $vl_simplenn_tidy$ function by the steps below:

1. Load the trained model from the .mat file.
2. Apply the $vl_simplenn_tidy$ function to extract the neural network.
3. Extract the raw image data and the corresponding labels.
4. replace the output layer from a $softmaxloss$ function to a $softmax$ function to compute the probability of each image label.
5. Use the $vl_simplenn$ function and the *squeeze* function to get the probability of each color channel of the image pixel matrices.
6. Get the highest value from the probabilities of all labels.
7. Plot the confusion matrix representing the overall outcome.

G.2 MATLAB Script for Model Evaluation

```

disp('loading imdb...');
% load the imdb here
imdb = load(fullfile('imdb-small-unrotated.mat'));
%% load trained model
disp('loading pretrained model...');
% run the toolbox
run(fullfile(fileparts(mfilename('fullpath')),...
  '..', '..', 'matlab', 'vl_setupnn.m')) ;
fold = 1;
%load the trained model here
model = load('net-epoch-50.mat');
%get the network for computing
model.net = vl_simplenn_tidy(model.net);
% get the indice of the test set
testSet = imdb.images.set{fold};
testImgs = imdb.images.data(:,:,:,testSet == 2);
testLabels = imdb.images.labels(:,testSet == 2);
% change from softmaxloss
model.net.layers{end}.type = 'softmax';
predictLabel = zeros(size(testImgs,4),1);
dummy_test = dummyvar(testLabels');
total = size(testImgs,4);
res = vl_simplenn(model.net, testImgs(:,:,:,1:total)) ;
scores = squeeze(gather(res(end).x)) ;
bestScores = max(scores);
%%
disp('performing predictions...');
for i = 1 : size(testImgs,4)
    maxScores = max(bestScores(:,:,:,i));
    [maxScore index] = max(maxScores);
    predictLabel(i) = index;
end

%%
dummy_predict = dummyvar(predictLabel);
figure
plotconfusion(dummy_test', dummy_predict')
```

Appendix H

MATLAB Code for Malaria Net Cross Validation

H.1 Description

This section contains the MATLAB code to to perform cross validation of the trained 18-layer Malaria Net. The code can be used to validate any convolutional neural network (CNN) models trained by the MATLAB MatConvNet Toolbox by small revision.

The code will generate the confusion matrix that illustrate the ground true labels on the X-axis (horizontal) versus the predicted labels on the Y-axis (vertical). The label '1' represents that the images belong to the infected red blood cells. The label '2' represents that the images belong to the uninfected normal red blood cells. When a raw image is classified, if the predicted label is consistent to the actual label, the prediction is correct.

H.2 MATLAB Script for Model Evaluation

```

disp('loading imdb...');
% load the imdb here
imdb = load(fullfile('imdb-small-unrotated.mat'));
%% load trained model
disp('loading pretrained model...');
% run the toolbox
```

```
run(fullfile(fileparts(mfilename('fullpath')),...
  '..', '..', 'matlab', 'vl_setupnn.m')) ;
fold = 1;
%load the trained model here
model = load('net-epoch-50.mat');
%get the network for computing
model.net = vl_simplenn_tidy(model.net);
% get the indice of the test set
testSet = imdb.images.set{fold};
testImgs = imdb.images.data(:,:,:,testSet == 2);
testLabels = imdb.images.labels(:,testSet == 2);
% change from softmaxloss
model.net.layers{end}.type = 'softmax';
predictLabel = zeros(size(testImgs,4),1);
dummy_test = dummyvar(testLabels');
total = size(testImgs,4);
res = vl_simplenn(model.net, testImgs(:,:,:,1:total)) ;
scores = squeeze(gather(res(end).x)) ;
bestScores = max(scores);
%%
disp('performing predictions...');
for i = 1 : size(testImgs,4)
    maxScores = max(bestScores(:,:,:,i));
    [maxScore index] = max(maxScores);
    predictLabel(i) = index;
end

%%
dummy_predict = dummyvar(predictLabel);
figure
plotconfusion(dummy_test', dummy_predict')
```

Appendix I

Publications during the MAIST Program Study

I.1 Publication in Conference Proceedings

1. Liang Zhaohui, Zhang Gang, Huang Jimmy Xiangji, Hu Qinming Vivian. Deep learning for healthcare decision making with EMRs. In Proceedings of IEEE International Conference on Bioinformatics and Biomedicine (BIBM). 2014, Nov 2-5, Belfast, UK, pp.556-559, DOI: 10.1109/BIBM.2014.6999219.

2. Liang Zhaohui, Zhang Gang, Huang Jimmy Xiangji. Discovery of the relations between genetic polymorphism and adverse drug reactions. In Proceedings of IEEE International Conference on Bioinformatics and Biomedicine (BIBM). 2015, Nov 9-12, Washington DC, USA, pp. 543-548, DOI: 10.1109/BIBM.2015.7359741.

3. Liang Zhaohui, Powell Andrew, Ersoy IIker, Poostchi Mahdieh, Silamut Kamolrat, Palaniappan Kannappan, Guo Peng, Hossain Md Amir, Sameer Antani, Maude Richard James, Huang Jimmy Xiangji, Jaeger Stefan, Thoma George. CNN-based image analysis for malaria diagnosis. In Proceedings of IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2016, Dec 15-18, Shenzhen, China, pp. 493-496, DOI:10.1109/BIBM.2016.7822567.

I.2 Publication in Journal

1. Liang Zhaohui, Huang Jimmy Xiangji, Zeng Xing, Zhang Gang. DL-ADR: a novel deep learning model for classifying genomic variants into adverse drug reactions. BMC Medical Genomics. 2016 Aug 10; 9(2):48, DOI: 10.1186/s12920-016-0207-4.

2. Zhaohui Liang, Honglai Zhang, Guozheng Li, Jimmy Xiangji Huang. Special Issue on Health and Clinical Informatics in Chinese Medicine (Editorial). International Journal of Computers in Healthcare. 2015; 2(2): 69-71.

3. Zhaohui Liang, Xiangji Huang, Byeongsang Oh, Josiah Poon. Bioinformatics/Medical Informatics in Traditional Medicine and Integrative Medicine (Editorial). The Scientific World Journal. 2015; 2015, Article ID 460490.