Heaven's Light is Our Guide

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING

Rajshahi University of Engineering & Technology, Bangladesh

Visualizing CNN-Based Performance Analysis for Malaria Cell Image

Author/Authors

Mahadi Hasan Biplob

Roll No. 1504022

Supervised By

Jannatul Robaiat Mou

Assistant Professor

Department of Electronics & Telecommunication Engineering

Rajshahi University of Engineering & Technology

ACKNOWLEDGEMENT

This thesis has been submitted to the Department of Electronics & Telecommunication Engineering of Rajshahi University of Engineering & Technology (RUET), Rajshahi-6204, Bangladesh, for the partial fulfillment of the requirements for the degree of B.Sc. in Electronics & Telecommunication Engineering. Thesis title regards to "**Visualizing CNN-Based Performance Analysis for Malaria Cell Image**".

First and foremost, I offer my sincere gratitude and in debtness to my thesis supervisor, Jannatul Robaiat Mou, Assistant Professor, Department of Electronics & Telecommunication Engineering who has supported me throughout my thesis with her patience and knowledge. I shall ever remain grateful to her for her valuable guidance, advice, encouragement, cordial and amiable contribution to my thesis.

Finally, I want to thank the most important and the closest persons of my life, my parents for giving a big support to me.

12 December, 2020 Mahadi Hasan Biplob

RUET, Rajshahi

Heaven's Light is Our Guide

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING

Rajshahi University of Engineering & Technology, Bangladesh

CERTIFICATE

Supervisor

Jannatul Robaiat Mou

Assistant Professor
Department of Electronics & Telecommunication Engineering
Rajshahi University of Engineering & Technology
Rajshahi-6204

External Examiner

(External Examiner Name)

(Supervisor's Designation)
Department of Electronics & Telecommunication Engineering
Rajshahi University of Engineering & Technology
Rajshahi-6204

Head

Dr. Md Munjure Mowla

Associate Professor
Department of Electronics & Telecommunication Engineering
Rajshahi University of Engineering & Technology
Rajshahi-6204

ABSTRACT

This paper studied automatic identification of malaria infected cells using deep learning methods. We used whole slide images of thin blood stains to compile an dataset of malaria-infected red blood cells and non-infected cells, as labeled by a bunch of 4 pathologists. We evaluated three types of well-known convolutional neural networks, including the LeNet, AlexNet and GoogLeNet. Simulation results showed that all these deep convolution neural networks achieved classification accuracies of over 96%, above the accuracy of about 92% attainable by using the support vector machine method. And also the validation accuracy is over 94% for using CNN in deep machine learning. Moreover, the deep learning methods have the advantage of having the ability to automatically learn the features from the computer file, thereby requiring minimal inputs from human experts for automated malaria diagnosis.

CONTENTS

Acknowledgement **i**

Certificate **ii**

Abstract **iii**

List of Figures **vi**

List of Tables **vii**

Introduction **1**

1.1 Motivation 2

1.2 Proposed Work 3

1.3 Contribution 3

1.4 Thesis Outline 4

Background And Preliminaries **5**

2.1 Literature Review 6

2.1.1 Evaluations of Deep Convolutional Neural Network for Automatic Identification of Malaria Infected Cells 6

2.1.2 CNN-Based Image Analysis for Malaria Diagnosis 7

2.1.3 Performance Analysis of Machine Learning And Deep Learning Architectures for Malaria Detection on Cell Images 8

2.2 Convolutional Neural Network 10

2.2.1 Criteria of CNN 11

2.2.2 Importance of CNN 12

2.2.3 Applications of CNN 14

2.3 Challenges for CNN Applications 15

Materials & Configuration of CNN **17**

3.1 The Architecture of Visual Cortex 17

3.2 Convolutional Layer 18

3.2.1 Filters 20

3.2.2 Stacking Multiple Features Maps 21

3.2.3 Memory Requirements 22

3.3 Pooling Layer 23

3.4 CNN Architectures 25

3.4.1 LeNet-5 27
3.4.2 AlexNet 28
3.4.3 GoogleNet 28
3.4.4 VGGNet 30
3.4.5 ResNet 30
3.4.6 Xception 33
3.4.7 SENet 35
3.5 Semantic Segmentation 37
Implementation & Result Evaluation **40**
4.1 Visualizing The Features 40
4.2 Visualizing The Activations 41
4.3 Model Evaluation 42
4.3.1 Data Source 42
4.3.2 Data Preprocessing 42
4.3.3 Dataset Compilation 42
4.3.4 CNN Model Training 42
4.4 Working Process 43
4.4.1 Importing Necessary Libraries 43
4.4.2 Displaying Uninfected and Infected Cell Tissues 43
4.4.3 Set Height Width in PX 44
4.4.4 Dividing Dataset into Two Folders Train and Test 44
4.4.5 Preparing Train and Test Image Generator 44
4.4.6 Preparing The Model 45
4.4.7 Summary of Model 45
4.4.8 Model Compilation 46
4.4.9 Displaying Epoch for Best Result 46
4.4.10 Plot The Result 47
4.5 Simulation Result and Analysis 48
Conclusion & Future Work **50**
5.1 Conclusion 50
5.2 Future Work 50
Reference **51**

LIST OF FIGURES

2.1 Typical CNN Architecture 10
2.2 Filter Pruning Methods Manually Select Criterion And Apply to All Layers 12
2.3 Pruning ResNets on the ImageNet dataset 13
3.1 Local receptive fields in the visual cortex 17
3.2 CNN layers with rectangular local receptive fields 19
3.3 Connections between layers and zero padding 19
3.4 Reducing dimensionality using a stride of 2 20
3.5 Applying two different filters to get two feature maps 21
3.6 Convolution layers with multiple feature maps, and images with three color channels . . . 22
3.7 Max pooling layer (2 × 2 pooling kernel, stride 2, no padding) 24
3.8 Invariance to small translations 24
3.9 Typical CNN architecture 25
3.10 Inception module 29
3.11 GoogLeNet architecture 30
3.12 Residual learning 31
3.13 Regular deep neural network (left) and deep residual network (right) 32
3.14 ResNet architecture 32
3.15 Skip connection when changing feature map size and depth 33
3.16 Depthwise Separable Convolutional Layer 34
3.17 SE-Inception Module (left) and SE-ResNet Unit (right) 35
3.18 An SE Block Performs Feature Map Recalibration 36
3.19 SE Block Architecture 36
3.20 Semantic segmentation 37
3.21 Upsampling Using a Transpose Convolutional Layer 38
3.22 Skip layers recover some spatial resolution from lower layers 39
4.1 Displaying Uninfected and Infected Cell tissues 44
4.2 Simulation Result for Accuracy of Training and Validation 48
4.3 Simulation Result for Loss of Training and Validation 48

LIST OF TABLES

4.1 Summary of Sequential Model 45
4.2 First Seven Epoch for Best Result 46

INTRODUCTION

Malaria is an communicable disease which is caused by a parasitic protozoan of the protoctist genus [1] which infects human red somatic cell and causes the disease. it's five differing kinds of species namely P.falciparum, P.vivax, P.malariae, P.ovale and P.knowlesi [1]. Among all P.falciparum is that the most dangerous parasite. per World Health Organization (WHO) report, there have been 216 million cases of malaria detected and causalities reached to 4,45,000 in 2019 & 2020 [2]. Malaria could be a curable & preventable disease if it's diagnosed at an early stage. Light micros-copy based diagnostic method is taken into account to be the gold standard method to detect different types of malaria. Thin or thick Giemsa stained blood smears are examined with magnification factor of 100x objective and 10x ocular lens under a microscope [3]. It is inferred from blood smear screen report that any person is also affected with more than one Plasmodium vivax simultaneously [2]. Fig. 1 shows mixed stages of the parasite within the same blood smear. It demands the necessity of viewing the entire specimen to spot mixed species [4]. Hence, any blood smear sample per malaria screening must be examined completely. Complete scanning of a smear through microscope requires examination of roughly 6050 non-overlapping fields of view for a specimen of size 2x1cm with 1000 magnification factor. It takes an outsized amount of your time to finish the diagnosis process. it's a watch straining and over exhaustive work for pathologists once they examine voluminous samples [4]. When the spread of disease is more in endemic regions, sufficient number of pathologists may not be available, because they're delegated for a hard and fast size of population and may not cover the complete endemic region. Also, there exists an instan-taneous relation between high incidents of malaria disease and alter in weather condition. High volume of cases in malaria epidemic localities and lack of skilled technicians, often ends up in delay in reporting the result which can be critical for malaria treatment [5]. The delay in diagnosis prevents prompt treatment of disease and further results in chronic stage, at substantial cost to the individual and large cost to society. Moreover, there are five different species of malaria parasites and every species undergoes a different life cycle. Every stage of the life cycle undergoes a change in its shape, size, morphology, color and etc. These stages are named as ring, trophozoite, schizont, and gametocyte [1]. Samples of such microscopic views of malaria specimen are shown in Fig. 2. to spot and understand the proper stages per specific types of malaria, skilled and well trained pathologists are needed. Pathologists who lack experience may find it difficult to segregate

such reasonably stages [3]. These reasons motivated us to style a cybernetic system; it could simulate the skilled knowledge of human intelligence in recognizing infected malaria of falciparum.

1.1 Motivation

Previous research fraternities have attempted various image processing techniques followed by machine learning to classify the infected erythrocyte from blood smear images [6]. Any image processing based malaria diagnosis method follows five functional stages namely image acquisition, image preprocessing, erythrocyte segmentation, feature extraction and classification [7]. one among the largest challenges in traditional feature crafting technique is to extract the features of microscopic image and apply an appropriate algorithm to map the extracted features to recognize infected malaria images [8]. This process is challenging, because the user must find an efficient algorithm to figure-out useful insights from malaria images. A recent study reveals that deep learning models are capable of learning knowledge like human-beings through experience. Convolutional Neural Network (CNN) is one such class of deep learning approach. It excels in image classification and mimics the substitute neural spec like human intelligence [9].
In this study, our contribution is to style a brand new VGG-SVM network using transfer learning approach for recognizing infected falciparum malaria. Here, a pre-trained VGG is taken into account as an expert learning model and it's targeted to classify 1000 classes. SVM is a domain specific classifier accustomed classify infected and non-infected malaria microscopic images. Thus, to unify these two learning models and to beat the mismatch of two different classes of target distributions, VGG-SVM is introduced with BTrain top layers and freeze out remainder of the layers^ transfer learning strategy. This unification facilitates a capability to utilize prior knowledge of VGG as model parameters to find out SVM for classifying infected and non-infected malaria images. VGG-SVM was trained using malaria image digital corpus and obtained 95% classification accuracy. This result demonstrates the potential of transfer learning in the field of medical image diagnosis, especially to malaria diagnosis.
Rest of the paper gives a close explanation of the method. Elaborates on the numerous literature works associated with identification and classification of Plasmodium vivax from micros-copic images. Provides basic details about deep learning, convolutional neural network, transfer learning and its types. Describes the design of the proposed malaria arrangement. Demonstrates the experimental results and performance evaluation. the ultimate section entails the conclusion.

1.2 Proposed Work

The architecture of a CNN will largely determine the ultimate net performance after training. the essential mechanism of deep learning is to use a multi-layer network to map the input space by transforming it at the hidden nodes. employing a series of transformations, the network tries to find out the optimal mapping of the input file through a process called back-propagation. In this process, the derivative or gradient of the input parameters is computed from the derivative of the output, given an objective function, by applying the chain rule. Thus the changes of 1 layer is computed recursively by measuring the changes of the subsequent layer connected to that. Learning of a CNN model consists of two inverse computations: the feed-forward and also the said back-propagation [10]. The feedforward propagation computes the output of all units in each layer, where for every unit a non-linear activation function f is applied to the weighted sum of all inputs from the preceding layer. The activation function may be a rectified long measure (ReLU), hyperbolic tangent (tanh), logistic function etc. Back- propagation is applied to coach or fine-tune the deep network by optimizing the parameters/weights of every layer. By applying the above propagation mechanisms, a CNN can create a model of the computer file when sufficient training data enters and propagates through the full network. The results show that the new CNN model incorporates a superior performance compared to the transfer learning model. The average classification accuracy of the CNN model is 96%, and the model sensitivity, specificity, and precision all reach the 94% level. The F1 score and also the Matthews correlation coefficient (MCC) of the trained CNN model are both quite 7% larger than the transfer learning model. This shows that the trained CNN model could be a far better representation of the training images than the transfer learning model, which relies on feature extraction from a pre-trained model trained on a wholly different image set.

1.3 Contribution

In this research, we've presented several deep learning-based classification approaches for CAD of Plasmodium. All the classification algorithms presented during this paper performed relatively well with minimum performance being 86% and 0.932 in terms of overall accuracy and AUC respectively. All the deep learning and transfer learning-based approaches performed better than bag-of-features and SVM based classification model. Implementation of CNN based architectures for categorizing the cell images as parasitized and uninfected is proven to be effective. Transfer learning based algorithms provided similar performance with VGG-16 performing the most effective in terms of AUC (0.993) and DenseNet being the most effective in terms of overall accuracy

(96.6%). Performance of transfer learning approaches clearly reiterates the very fact that CNN based classification models are good in extracting features. Figure 9 illustrates that proposed fast CNN architecture provided comparable performance and it's the smallest amount in terms of your time consumption among the deep learning architectures presented during this paper. Note that, this level of accuracy is achieved for the fast CNN with images of size 50 × 50 which helps in reducing memory consumption additionally. With reduced training time, algorithm are often easily re-trained with new sets of labeled images to reinforce the performance further. Combining the results of of these architectures provided a lift in performance both in terms of AUC and overall accuracy. A comprehensive study of these algorithms both in terms of computation (memory and time) and performance provides the topic matter experts to choose algorithms supported their choice. CAD of Plasmodium would be of great help for the microscopists for malaria screening and would help in providing a valuable second opinion.

1.4 Thesis Outline

The original whole slide image data contain significant amount of redundant information. so as to realize good classification accuracy, image segmentation and de-noising have to be done to extract only blood cells and take away those redundant image pixels simultaneously. First, each image tile was converted into a grayscale image from the colour space followed by the thresholding operation. However, noise as the byproduct of thresholding can degrade the standard of the acquired images and lower the classification performance. Therefore, the isolated noise pixels were eliminated by applying image morphological operations in Matlab. Another common issue is that segmented whole slide images almost inevitably have many overlapped red blood cells (RBCs), which can cause inaccurate classification. The Hough Circle transform [11] was applied to detect disk-like overlapped RBCs and so separate them. After the information preprocessing, we randomly selected an oversized number of cell images and provided them to pathologists at the University of Alabama at Birmingham. the complete whole slide image dataset are divided into four segments evenly. Each of 4 pathologists are assigned with two segments in order that each cell image are viewed and labeled by a minimum of two experienced pathologists. One cell image can only be considered as infected and included in our final dataset if all the reviewers mark it positively whereas it will be excluded otherwise. the identical selection rule also applies to the non-infected cells in our dataset. After this data curation, we collected 13,034 infected cells and 13,531 non-infected cells. Next we divide this dataset into two sets of approximately equal size: training set and testing set.

BACKGROUND AND PRELIMINARIES

Dealing with input variances is one among the most concerns in classifying images, regardless of its kind. It's difficult to account for the changes in size, background, angle and position of the objects inside images. within the process, state of-art image processing algorithms depend upon cleverly hand- engineered features for representing the underlying data. Extensive time is spend during this pre-processing step, demanding human expertise that severely limits the accuracy achievable by a training algorithm. Sufficient large training examples are needed to be told the suitable invariances with minimal processing, just by using the low-level data representations like raw pixels. Deep learning (DL), also known as deep machine learning (or hierarchical learning), is a class of ML algorithms that use a cascade of layers of non-linear processing units for end-to-end feature extraction and classification and are resilient to those variances [12]. DL using convolutional neural networks has gained research interest because it offers the promise of delivering high quality classification without the requirement for han selecting features. Unlike SVMs, the performance of DL models increases with the quantity of coaching examples, making them highly scalable [13]. Small datasets aren't adequate train a DL model which has a multitude of parameters that require tuning. Transfer Learning (TL) methods are commonly accustomed alleviate the problem where a pre-trained deep network is employed to extract the features that are subsequently utilized in a traditional classifier like SVM [14]. These pre-trained models have already learned features that are useful for many computer vision (CV) problems, and visualizing such features provides a more robust understanding of th educational process and allows reaching a comparable accuracy to it of a customized model. Krizhevsky et al. (2012) proposed the AlexNet model, trained on ImageNet data, containing over 15 million annotated images from a complete of over 22,000 categories [15]. The model used rectified linear units (ReLU) for imposing nonlinearity and data augmentation techniques that consisted of image translations and reflections. It also used dropout layers to combat the matter of overfitting to the training data and was trained using batch stochastic gradient descent (SGD) with specific values fo momentum and weight decay. Simonyan and Zisserman from the University of Oxford proposed a straightforward and deep model in 2014 dubbed VGGNet that used only 3 x 3 sized filters at some stage in the model [16]. Several variants of the VGG networks were proposed including VGG16 and VGG19, where “16” and “19” indicate the quantity of weight layers within the network. Thes models reinforced the notion that the mixture of two 3 x 3 convolution layers has an effective receptive field of 5 x 5 that simulates a bigger filter while

keeping the advantages of smaller filter sizes and parameters. The model performed equally well on image classification and localization tasks. TL models reduce the training time at the price of performance and will be suitable when larger training datasets don't seem to be available. They produce useful features s long because the domain under study doesn't deviate much from the data [17]. they have an inclination to perform poorly on data on which they are not trained on before. Further, we are constrained in terms of the spec. We can’t selectively modify the pre-trained network [18]. the opposite big difference lies within the formulation of the matter. TL models were designed for multiclass classification which suggests that they learn plenty of additional information that will not be needed in a very binary classification problem like ours. The issue is resolved by fine-tuning a pre-trained model augmented with some layers for binary classification with more number of epochs. This changes the intra-network information from multi-class to a binary-class problem. It is necessary to visualise the features extracted by the DL model and their activations so as to higher understand its learning strategy. The downside of such a practice is that the amount of weights stored internally will be huge, requiring additional regularization. Also, the performance of fine tuned models relies on the initial pre-trained model and any improvement in performance is tied to the representation learned by the first model [19]. These challenges may be overcome by employing a customized model, trained on the domain of interest. CNN based DL models gives promising results for perceptual applications like image classification. A survey of literature revealed few comparable articles for malaria cell classification using DL models. a way supported CNN for classifying the parasitemic and uninfected cells from thin blood smear images was attempted that used 27,578 single cell images leading to a mean accuracy of 97.3%. By comparison, a pre-trained model used for classifying the same data achieved 91.99% accuracy. In this article, we propose to use the benefits offered through visualizing extracted features and DL network activations in a very simple, customized DL model for malaria cell classification. Our goal is to not only improve the state of the art in malaria classification, but also to know the impact of varied learned parameters on DL models at different stages of the deep network.

2.1 Literature Review

2.1.1 Evaluations of Deep Convolutional Neural Networks for Automatic Identification of Malaria Infected Cells

Recently, machine learning algorithms have gained an increasing attention from researchers for its great capability in building automated diagnostic system for malaria [20], [21]. In [20], SVM and

Naive Bayes Classifier were utilized to achieve accuracies of 84% and 83.5% respectively. In contrast to the supervised learning, unsupervised learning, for instance, K-Nearest Neighbor (KNN) has also been proposed to recognize malaria infected cells in [21]. In [22], a three- layer Neural Network (NN) was designed as a classifier to detect malaria infected cells at an accuracy of 85%. Although these supervised learning algorithms had some success with good infection detection accuracies, their performances are very sensitive to the feature extraction utilized in their work. Thus building a discriminant feature vector with minimal redundancy is extremely crucial. lots of work has been done to extract features for the malaria infected cells [20], [21], [23]. In [24], a survey of feature extraction and optimization for malaria cells has been discussed intimately. Although good feature extraction method can improve the detection accuracy, this sort of infection detection cannot achieve fully automated diagnosis because it still requires trained experts to manually extract feature vectors per the particular datasets. During this work, we propose to use deep learning algorithms for malaria cell detection, with the last word goal of building a really automated diagnostic platform with none manual feature extraction. to the current end, deep machine learning methods can provide an honest solution. Deep learning methods can extract from the input a hierarchical representation of the data, with higher layers representing increasingly abstract concepts, which are increasingly invariant to transformations and scales. for instance, in [25], a deep convolutional neural network was applied to diagnose malaria in thick blood smear. However, to tell apart infected and non-infected samples in thick films is basically difficult for pathologists, because the difference isn't as clear as those individual red blood cells cropped from whole slide images based on thin films. In this work, three well-known deep convolutional neural networks, including LeNet-5 [26], [27], AlexNet [28] and GoogLeNet [29] were accustomed learn the inherent features of the malaria infected cells and also the non-infected cells. For comparison, a support vector machine (SVM) was trained on pre-selected features extracted from the identical dataset.

2.1.2 CNN-Based Image Analysis for Malaria Diagnosis

The standard way of diagnosing malaria is by visually examining blood smears for parasite-infected red blood cells under the microscope by qualified technicians. This method is inefficient and therefore the diagnosis depends on the experience and also the knowledge of the person doing the examination. Automatic image recognition technologies supported machine learning are applied to malaria blood smears for diagnosis before. However, the sensible performance has not been sufficient thus far. This study proposes a new and robust machine learning model supported a convolutional neural network to automatically classify single cells in thin blood smears on standard

microscope slides as either infected or uninfected. during a ten-fold cross validation supported 27,578 single cell images, the common accuracy of our new 16-layer CNN model is 97.37%. A transfer learning model only achieves 91.99% on the same images. The CNN model shows superiority over the transfer learning model altogether performance indicators like sensitivity (96.99% vs 89.00%), specificity (97.75% vs 94.98%), precision (97.73 vs 95.12%), F1 score (97.36% vs 90.24%), and Matthews correlation coefficient (94.75% vs 85.25%). Malaria may be prevented, controlled, and cured more effectively if a more accurate and efficient diagnostic method were available. the quality diagnostic method for malaria is the microscopic examination of blood smears for infected erythrocytes by qualified microscopists. However, this method is inefficient and therefore the quality of the diagnosis depends on the experience and knowledge of the microscopists. Rapid diagnostic tests also are widely used but they're costlier and provide less information than microscopy. Automatic image recognition technologies supported machine learning and large data are applied to both thick and thin malaria blood smears for microscopic diagnosis since 2005 [30]. In this work, we use a deep learning approach to detect parasite-infected red blood cells in thin smears on standard microscope slides prepared using routine methods. We apply a convolutional neural network model, which could be a deep learning model particularly designed for learning of two- dimensional data like images and videos. It's inspired by experiments on the underlying physiological mechanisms within the visual cortex of felines for recognizing objects [31]. The experiments motivate a pattern recognition model to mimic the visual IP of the brain [32]. The advantage of a CNN model is that its hierarchical data structure of learning layers can be trained in a very 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 perimeters in an image) to reduce the quantity of parameters that require to be learned. This improves the accuracy of the feed forward back propagation training procedure. Since deep learning can model very complex features, a CNN provides a general-purpose learning framework not requiring beforehand feature extraction and fine-tuning, which is a plus over traditional classifiers.

2.1.3 Performance Analysis of Machine Learning and Deep Learning Architectures for Malaria Detection on Cell Images

Malaria could be a life-threatening disease caused by the parasites transmitted through the bite of the feminine Anopheles mosquito. Thick and thin film microscopic examinations of blood smears are the foremost commonly used and reliable methods for diagnosis, however, its accuracy depends on the smear quality and human expertise in classifying the traditional

and parasitemic cells. Manual examination is burdensome for large-scale diagnoses in endemic regions resulting in poor quality, unnecessary medication, leading to severe economic impact to the individual health program. Automated malaria screening using machine learning techniques, like deep learning, offers the promise of serving as a good diagnostic aid. during this study, we propose the benefits offered through visualizing the features and activations in a very simple, customized deep learning model. We apply it to the challenge of malaria cell classification, and as a result the model achieves 98.61% classification accuracy with lower model complexity and computation time. it's found to considerably outperform the state of the art including other pre-trained deep learning models. Malaria could be a life-threatening disease caused by the parasites transmitted through the bite of the feminine Anopheles mosquito. Different varieties of plasmodium including P. falciparum, P. ovale, P. vivax and P. malariae can infect humans, of which P. falciparum is that the deadliest. According to the 2016 World Health Organization (WHO) report, there were 212 million instances of the disease worldwide and therefore the African region accounted for the bulk of the disease cases, followed by south-east Asian and eastern Mediterranean regions [33]. Microscopic examination of thick and thin blood smears for infected erythrocytes may be a commonly used method for malaria diagnosis. Depending on the local protocol, the examination includes: (I) classifying and counting the traditional and infected erythrocytes within the thin smear images; and/or (ii) counting parasites in thick smear images as per the WHO guidelines [34]. Thus, the diagnostic accuracy is heavily dependent on manual expertise and may be adversely impacted by the burden posed by large scale analyses that are common in malaria endemic regions. Alternative techniques like polymerase chain reaction (PCR) and rapid diagnostic tests (RDT) are widely used. However, PCR tests are limited in their performance [35] while RDTs are less cost-effective in zones with high disease prevalence [36]. Machine learning (ML) techniques are previously applied to detect the degree of parasitemia from Giemsa- stained blood smear images [37, 38]. The study focused on developing a software to automatically compute the degree of parasitemia using hand-selected features from the stained blood smears containing P. falciparum parasites. However, the study failed to explicitly classify normal and uninfected blood cells and also the manual selection of features demanded human intervention. Another study to perform expert-level malaria diagnosis supported automated microscopy was proposed where variety of classifiers including a linear support vector machine (SVM), radial basis function SVM, multi-layer perceptron (MLP) and Gradient-Boosted Decision Trees (GBM) were wont to classify the traditional and parasitemic cells [39]. However, the study involved only fewer samples and therefore the performance suffered with increase in data size. Diaz et al. proposed a SVM based classification method for detecting the infected red blood cells (RBC) in preprocessed blood smear

images [40]. The algorithm performed well on a smaller dataset of 450 images, however, the performance decreased when applied to the blood images carrying infected cells.

2.2 Convolutional Neural Network

A convolutional neural network could be a specific deep learning architecture suitable for image recognition. A CNN model processes computer file by its multiple layers and is characterized by four key features: local connections, shared weights, pooling, and also the use of the many layers [41]. The early applications of CNN will be traced back to the 1990s for speech recognition [42] and text recognition [43]. Its use is then extended to handwriting recognition [44] and later to natural image recognition [45]. The performance of CNN models for natural image classification received another boost by the introduction of ImageNet by Alex Krizhevsky, thus also known as AlexNet, in 2012. AlexNet is taken into account a breakthrough application of CNNs to multi-categorical classification. In the ILSVRC-2012 competition, ImageNet composed of seven convolutional layers successfully classified the ILSVRC-2012 sets with 10,184 categories and eight.9 million images with a top-5 error of 15.3% [46]. Following this primary success, the top-5 error has been reduced to 14.8% by ZFNet in 2013 [47], then to 6.7% by GoogLeNet in 2014 [48], and to 3.6% by ResNet in 2015 [49].

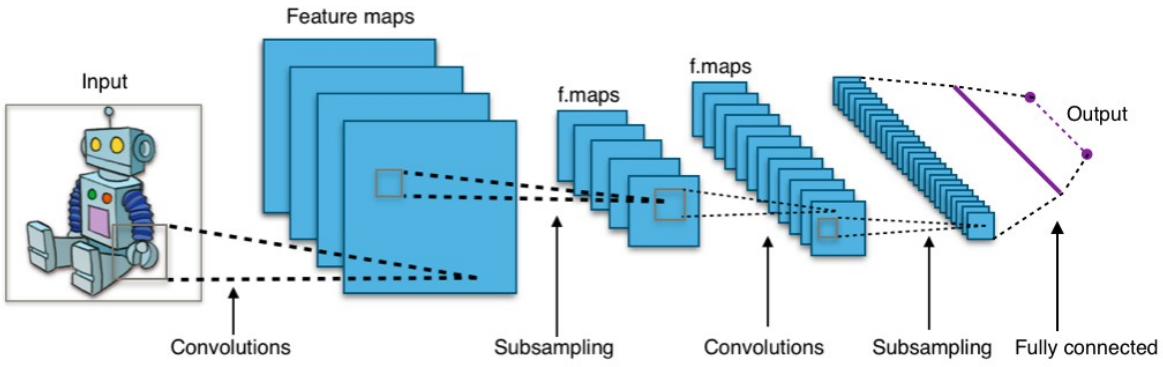

Figure 2.1 Typical CNN Architecture [50]

The architecture of a CNN will largely determine the ultimate net performance after training. the essential mechanism of deep learning is to use a multi-layer network to map the input space by transforming it at the hidden nodes. employing a series of transformations, the network tries to be told the optimal mapping of the computer file through a process called back-propagation. In this process, the partial or gradient of the input parameters is computed from the partial of the output, given an objective function, by applying the chain rule. Thus the changes of 1 layer will be computed recursively by measuring the changes of the subsequent layer connected thereto.

Learning of a CNN model consists of two inverse computations: the feed-forward and therefore the said back-propagation [51]. The feedforward propagation computes the output of all units in each layer, where for every unit a non-linear activation function f is applied to the weighted sum of all inputs from the preceding layer. The activation function are often a rectified linear measure (ReLU), hyperbolic tangent (tanh), logistic function etc. Back- propagation is applied to coach or fine-tune the deep network by optimizing the parameters/weights of every layer. By applying the above propagation mechanisms, a CNN can create a model of the input file when sufficient training data enters and propagates through the full network.

2.2.1 Criteria of Convolutional Neural Network

The rapid pace and successful application of machine learning research and development has seen widespread deployment of deep convolutional neural networks. Alongside thes algorithmic efforts, the compute- and memory-intensive nature of CNNs has stimulated an oversized amount of labor within the field of hardware acceleration for these networks. during this paper, we profile the memory requirements of CNNs in terms of both on-chip memory size and off-chip memory bandwidth, so as to know the impact of the memory system on accelerator design. We show that there are fundamental tradeoffs between performance, bandwidth, and on-chip memory. Further, this paper explores how the large choice of CNNs for various application domains each have fundamentally different characteristics. We show that bandwidth and memory requirements for various networks, and infrequently for various layers within a network, can each vary by multiple orders of magnitude. This makes designing fast and efficient hardware for all CNN applications difficult. To remedy this, we outline heuristic design points that try and optimize for select dataflow scenarios.

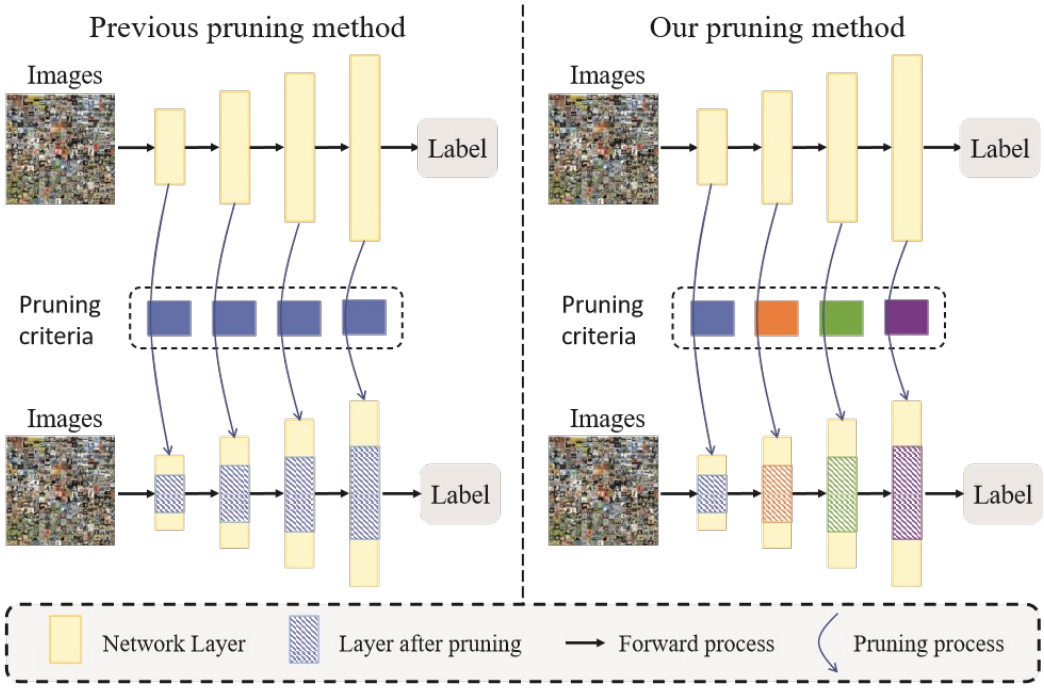

Figure 2.2 Filter Pruning Methods Manually Select Criterion And Apply to All Layers [52]

Filter pruning has been widely applied to neural network compression and acceleration. Existing methods usually utilize pre-defined pruning criteria, like Lp-norm, to prune unimportant filters. There are two major limitations to those methods. First, existing methods fail to contemplate the variability of filter distribution across layers. To extract features of the coarse level to the fine level, the filters of various layers have various distributions. Therefore, it's not suitable to utilize the identical pruning criteria to different functional layers. Second, prevailing layer-by-layer pruning methods process each layer independently and sequentially, failing to think about that each one the layers within the network collaboratively make the ultimate prediction. during this paper, we propose Learning Filter Pruning Criteria (LFPC) to resolve the above problems. Specifically, we develop a differentiable pruning criteria sampler. This sampler is learnable and optimized by the validation loss of the pruned network obtained from the sampled criteria. during this way, we could adaptively select the suitable pruning criteria for various functional layers. Besides, when evaluating the sampled criteria, LFPC comprehensively consider the contribution of all the layers at the identical time. Experiments validate our approach on three image classification benchmarks.

2.2.2 Importance of Convolutional Neural Network

Structural pruning of neural network parameters reduces computation, energy, and memory transfer costs during in- ference. We propose a completely unique method that estimates the contribution of

a neuron (filter) to the ultimate loss and iter- atively removes those with smaller scores. We describe two variations of our method using the primary and second- order Taylor expansions to approximate a filter's contribu- tion. Both methods scale consistently across any network layer without requiring per-layer sensitivity analysis and can be applied to any reasonably layer, including skip connections. for contemporary networks trained on ImageNet, we measured experimentally a high (>93%) correlation be- tween the contribution computed by our methods and a reliable estimate of actuality importance. Pruning with the proposed methods ends up in an improvement over state-of-the-art in terms of accuracy, FLOPs, and parameter reduction. On ResNet-101, we achieve a 40% FLOPS reduction by removing 30% of the parameters, with a loss of 0.02% in the top1 accuracy on ImageNet.

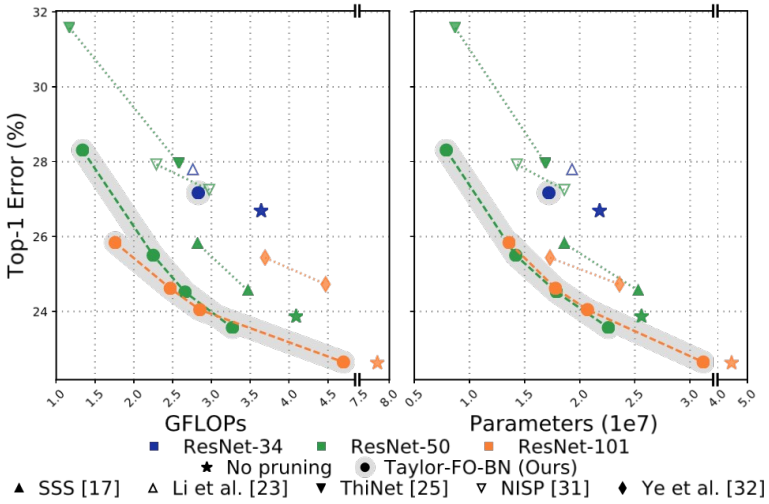

Figure 2.3 Pruning ResNets on the ImageNet dataset [53]

Convolutional neural networks (CNNs) are widely used in today's computer vision applications. Scaling up the dimensions of datasets similarly because the models trained on them has been responsible for the successes of deep learning. The dramatic increase in number of layers, from 8 in AlexNet [54], to over 100 in ResNet-152 [55], has enabled deep net- works to attain better-than-human performance on the Im- ageNet [56] classification task. Empirically, while larger networks have exhibited better performance, possibly thanks to the lottery ticket hypothesis [57], they need also been known to be heavily over-parameterized [58]. The growing size of CNNs could also be incompatible with their deployment on mobile or embedded devices, with limited computational resources. Even within the case of cloud services, prediction latency and energy consumption are important considerations. All of those use cases will benefit greatly from the supply of more compact networks. Pruning could be a common method to derive a compact network after training, some structural portion of the parameters is removed, together with its associated computations.

A variety of pruning methods are proposed, based on greedy algorithms [59, 60], sparse regularization [61, 62, 63], and reinforcement learning [64]. Many of them depend on the belief that the magnitude of a weight and its importance are strongly correlated. We question this belief and observe a significant gap in correlation between weight-based pruning decisions and empirically optimal one-step decisions a spot which our greedy criterion aims to fill. We focus our attention on extending previously proposed methods [65, 66, 67] with a brand new pruning criterion and a method that iteratively removes the smallest amount important set of neurons (typically filters) from the trained model. We define the importance because the squared change in loss induced by removing a particular filter from the network. Since computing the precise importance is extremely expensive for big networks, we approximate it with a Taylor expansion (akin to [67]), leading to a criterion computed from parameter gradients readily available during standard training. Our method is simple to implement in existing frameworks with minimal overhead.

2.2.3 Applications of Convolutional Neural Network

A tremendous interest in deep learning has emerged in recent years [68]. the foremost established algorithm among various deep learning models is convolutional neural network, a class of artificial neural networks that has been a dominant method in computer vision tasks since the astonishing results were shared on the article recognition competition referred to as the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012 [69, 70]. Medical research is not any exception, as CNN has achieved expert-level performances in various fields. Gulshan et al. [71], Esteva et al. [72], and Ehteshami Bejnordi et al. [73] demonstrated the potential of deep learning for diabetic retinopathy screening, skin lesion classification, and lymphatic tissue metastasis detection, respectively. Needless to say, there has been a surge of interest within the potential of CNN among radiology researchers, and several other studies have already been published in areas like lesion detection [74], classification [75], segmentation [76], image recons-truction [77, 78], and linguistic communication processing [79]. Familiarity with this state-of-the-art methodology would help not only re- searchers who apply CNN to their tasks in radiology and medical imaging, but also clinical radiologists, as deep learning may influence their practice within the near future. this text focuses on the essential concepts of CNN and their application to various radiology tasks, and discusses its challenges and future directions. Other deep learning models, like recurrent neural networks for sequence models, are beyond the scope of this text. The following terms are consistently employed throughout this article so on avoid confusion. A Bparameter^ during this article stands for a variable that's automatically learned during the training process. A Bhyperparameter^ refers to a variable that must be set before the training process starts. A Bkernel^ refers to the sets of learnable

parameters applied in convolution operations. A Bweight^ is usually used inter- changeably with Bparameter^; however, we tried to use this term when touching on a parameter outside of convolution layers, i.e., a kernel, as an example in fully connected layers. CNN may be a variety of deep learning model for processing data that has a grid pattern, like images, which is inspired by the organization of animal cortical area [80, 81] and designed to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns. CNN could be a mathematical construct that's typically composed of three kinds of layers (or building blocks): convolution, pooling, and fully connected layers. the primary two, convolution and pooling layers, perform feature extraction, whereas the third, a totally connected layer, maps the extracted features into final output, like classification. A convolution layer plays a key role in CNN, which is composed of a stack of mathematical operations, such as convolution, a specialized kind of linear operation. In digital images, pixel values are stored during a two-dimensional (2D) grid, i.e., an array of numbers, and atiny low grid of parameters called kernel, an optimizable feature ex- tractor, is applied at each image position, which makes CNNs highly efficient for image processing, since a feature may occur anywhere within the image. united layer feeds its output into the subsequent layer, extracted features can hierarchically and progressively become more complex. the method of optimizing parameters like kernels is named training, which is performed so on minimize the difference between outputs and ground truth labels through an optimization algorithm called backpropagation and gradient descent, among others. Most recent radiomics studies use hand-crafted feature extraction techniques, like texture analysis, followed by conventional machine learning classifiers, like random forests and support vector machines [82, 83]. There are several differences to notice between such methods and CNN. First, CNN does not require hand-crafted feature extraction. Second, CNN architectures don't necessarily require segmentation of tumors or organs by human experts. Third, CNN is much more data hungry thanks to its variant learnable parameters to estimate, and, thus, is more computationally expensive, result-ing in requiring graphical processing units (GPUs) for model training.

2.3 Challenges for CNN Applications

The idea of CNNs is to use smaller convolutional kernels (or filters) together with a deep specification to capture the discernable image features the maximum amount as possible. However, a more complex CNN architecture will inevitably increase the demand for more powerful computing resources. New technologies like GPU and cluster computing can effectively improve the training efficiency but they're unable to ensure the performance of classification. Other factors like data preprocessing and size of the training dataset strongly affec the classification

accuracy, however. Since the CNN classification accuracy depends on the number of coaching data, small data sets like those utilized in earlier approaches aren't large enough for training a deep model with its many parameters. As a compromise, the strategy of transfer learning has been introduced where a pre-trained network model is employed to extract features that a traditional classifier, like a SVM, can use for a fine-tuned classification [84]. Transfer learning will be used as a shortcut to deep learning where time for training is saved at the value of performance, which can be lower but still acceptable. It is used as a brief replacement when large training data isn't immediately accessible. during this study, we will implement deep learning by both training a newly configured CNN model and applying transfer learning so as to evaluate its use for malaria blood smear classification.

MATERIALS & CONFIGURATION OF CNN

CNNs emerged from the study of the brain's visual cortex, and that they are utilized in image recognition since the 1980s. within the previous couple of years, due to the rise in computational power, the number of obtainable training data, and therefore the tricks for training deep nets, CNNs have man- aged to attain superhuman performance on some complex visual tasks. They power image search services, self-driving cars, automatic video classification systems, and more. Moreover, CNNs aren't restricted to visual perception: they're also successful at many other tasks, like voice recognition or tongue processing (NLP); however, we'll specialize in visual applications for now. In this chapter we are going to present where CNNs came from, what their building blocks look like, and the way to implement them using TensorFlow and Keras. Then we'll discuss a number of the most effective CNN architectures, and discuss other visual tasks, including object detection (classifying multiple objects in a picture and placing bounding boxes around them) and semantic segmentation (classifying each pixel per the category of the item it belongs to).

3.1 The Architecture of Visual Cortex

David H. Hubel and Torsten Wiesel performed a series of experiments on cats in 1958 and 1959 (and some years afterward monkeys 3), giving crucial insights on the structure of the visual area (the authors received the laurels in Physiology or Medicine in 1981 for his or her work). specifically, they showed that a lot of neurons in the visual area have atiny low local receptive field, meaning they react only to visual stimuli located in an exceedingly limited region of the visual view (see Figure 100000000, during which the local receptive fields of 5 neurons are represented by dashed circles). The receptive fields of various neurons may overlap, and together they tile the entire visual view.

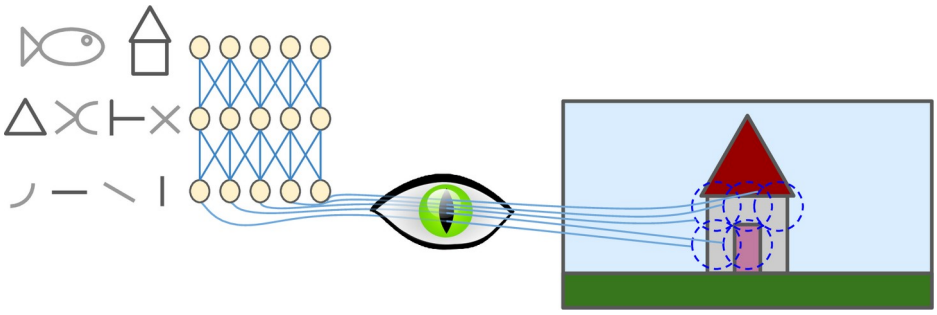

Figure 3.1 Typical CNN architecture [85]

Moreover, the authors showed that some neurons react only to pictures of horizontal lines, while others react only to lines with different orientations (two neurons may have the identical receptive field but react to different line orientations). They also noticed that some neurons have larger receptive fields, and that they react to more complex patterns that are combinations of the lower-level patterns. These observations led to the thought that the higher-level neurons are supported the outputs of neighboring lower-level neurons (in Figure 100000000, notice that every neuron is connected only to a few neurons from the previous layer). This powerful architecture is ready to detect all sorts of complex patterns in any area of the visual view. These studies of the cortical area inspired the neocognitron, introduced in 1980, 4 which gradually evolved into what we now call convolutional neural networks. An important milestone was a 1998 paper 5 by Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, which introduced the famous LeNet-5 architecture, widely used to recognize handwritten check numbers. This architecture has some building blocks that you already know, like fully connected layers and sigmoid activation functions, but it also introduces two new building blocks: convolutional layers and pooling layers.

3.2 Convolutional Layer

The most important building block of a CNN is that the convolutional layer: 6 neurons in the first convolutional layer don't seem to be connected to each single pixel within the input image (like they were in previous chapters), but only to pixels in their receptive fields (see Figure 3.2). In turn, each neuron within the second convolutional layer is connected only to neurons located within alittle rectangle within the first layer. This architecture allows the network to focus on small low-level features within the first hidden layer, then assemble them into larger higher-level features within the next hidden layer, and so on. This hierarchical data structure is common in real-world images, which is one among the reasons why CNNs work so well for image recognition.

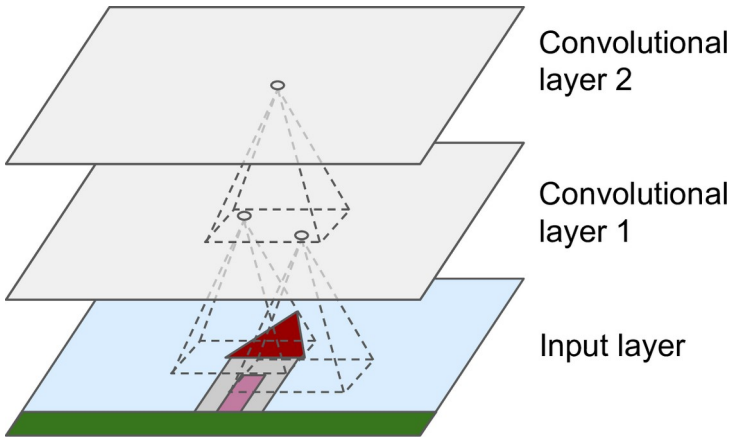

Figure 3.2 CNN layers with rectangular local receptive fields [86]

A neuron located in row i, column j of a given layer is connected to the outputs of the neurons within the previous layer located in rows i to i + f h – 1, columns j to j + f w – 1, where f h and f w are the peak and width of the receptive field (see Figure 3.3). In order for a layer to own the identical height and width because the previous layer, it's common to feature zeros round the inputs, as shown within the diagram. this is often called zero padding.

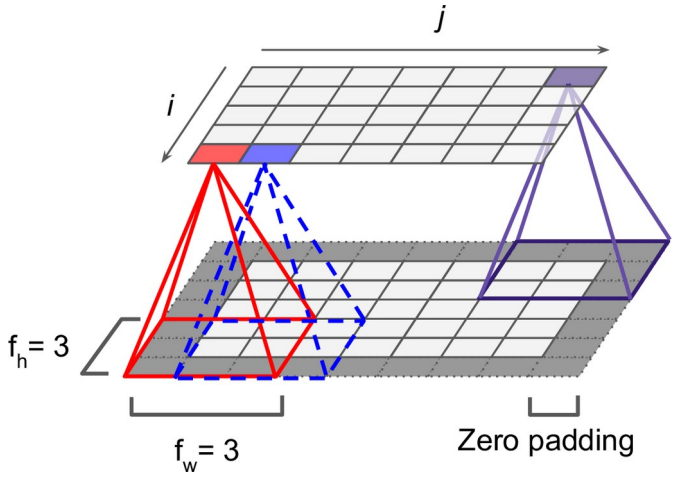

Figure 3.3 Connections between layers and zero padding [87]

It is also possible to attach an over-sized input layer to a way smaller layer by spacing out the receptive fields, as shown in Figure 3.4. The shift from one receptive field to the next is termed the stride. within the diagram, a 5 × 7 input layer (plus zero padding) is con- nected to a 3 × 4 layer, using 3 × 3 receptive fields and a stride of two (in this instance the stride is that the same in both directions, but it doesn't must be so). A neuron located in row i, column j within the upper layer is connected to the outputs of the neurons in the previous layer located in rows i × s

h to i × s h + f h – 1, columns j × s w to j × s w + f w – 1, where s h and s w are the vertical and horizontal strides.

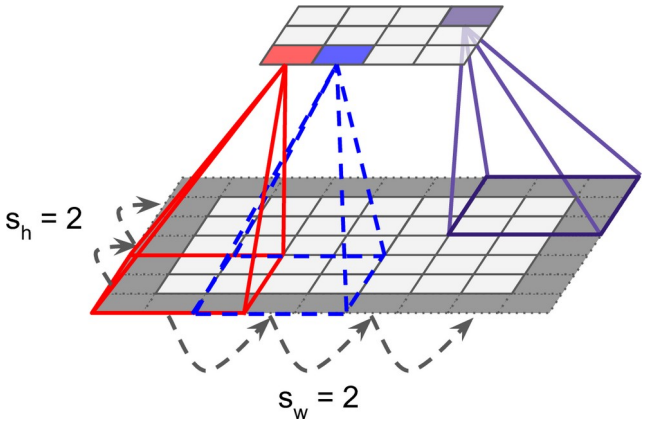

Figure 3.4 Reducing dimensionality using a stride of 2 [88]

3.2.1 Filters

A neuron's weights are often represented as atiny low image the dimensions of the receptive field. For example, Figure 3.5 shows two possible sets of weights, called filters (or convolution kernels). the primary one is represented as a black square with a vertical point of reference in the middle (it could be a 7 × 7 matrix filled with 0s aside from the central column, which is stuffed with 1s); neurons using these weights will ignore everything in their receptive field except for the central vertical line (since all inputs will get multiplied by 0, apart from the ones located within the central vertical line). The second filter could be a black square with a horizontal reference within the middle. Once again, neurons using these weights will ignore everything in their receptive field aside from the central horizontal line. Now if all neurons in an exceedingly layer use the identical vertical line filter (and the identical bias term), and you feed the network the input image shown in Figure 3.5 (bottom image), the layer will output the top-left image. Notice that the vertical white lines get enhanced while the remainder gets blurred. Similarly, the upper-right image is what you get if all neurons use the identical horizontal line filter; notice that the horizontal white lines get enhanced while the remainder is blurred out. Thus, a layer stuffed with neurons using the identical filter outputs a feature map, which highlights the areas in a picture that activate the filter the foremost. after all you are doing not need to define the filters manually: instead, during training the convolutional layer will automatically learn the foremost useful filters for its task, and therefore the layers above will learn to mix them into more complex patterns.

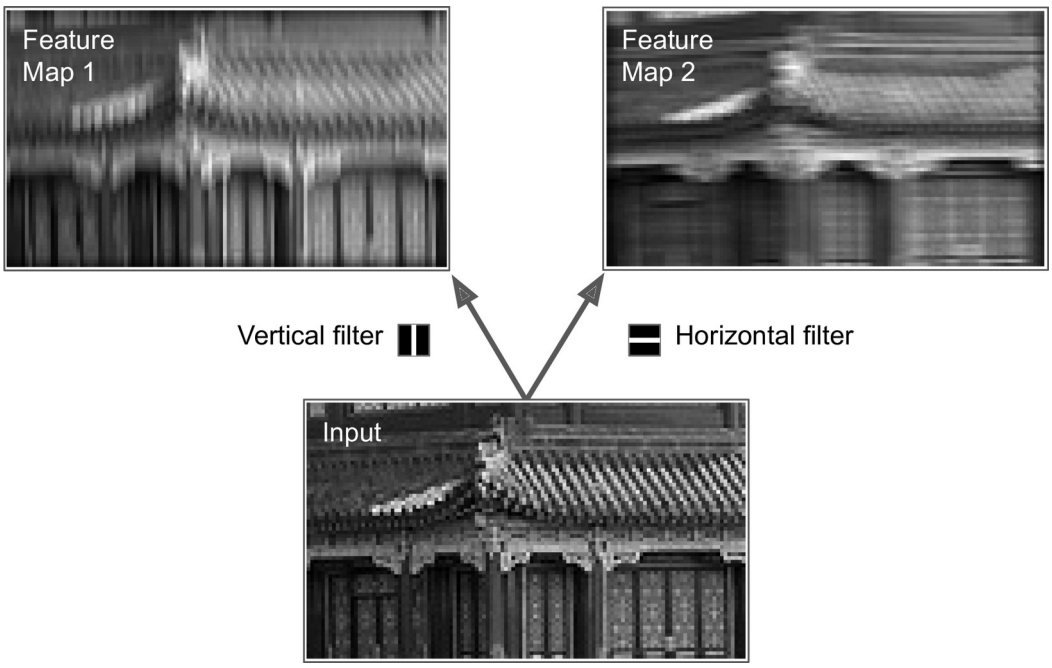

Figure 3.5 Applying two different filters to get two feature maps [89]

3.2.2 Stacking Multiple Feature Maps

Up to now, for simplicity, i've got represented the output of every convolutional layer as a thin 2D layer, but truly a convolutional layer has multiple filters (you decide how many), and it outputs one feature map per filter, so it's more accurately repre- sented in 3D (see Figure 3.6). To do so, it's one neuron per pixel in each feature map, and every one neurons within a given feature map share the identical parameters (i.e., the same weights and bias term). However, neurons in several feature maps use different parameters. A neuron’s receptive field is that the same as described earlier, but it extends across all the previous layers’ feature maps. In short, a convolutional layer simultaneously applies multiple trainable filters to its inputs, making it capable of detecting multiple features anywhere in its inputs. Moreover, input images are composed of multiple sublayers: one per color channel. There are typically three: red, green, and blue (RGB). Grayscale images have only one channel, but some images may have much more—for example, satellite images that capture extra light frequencies (such as infrared).

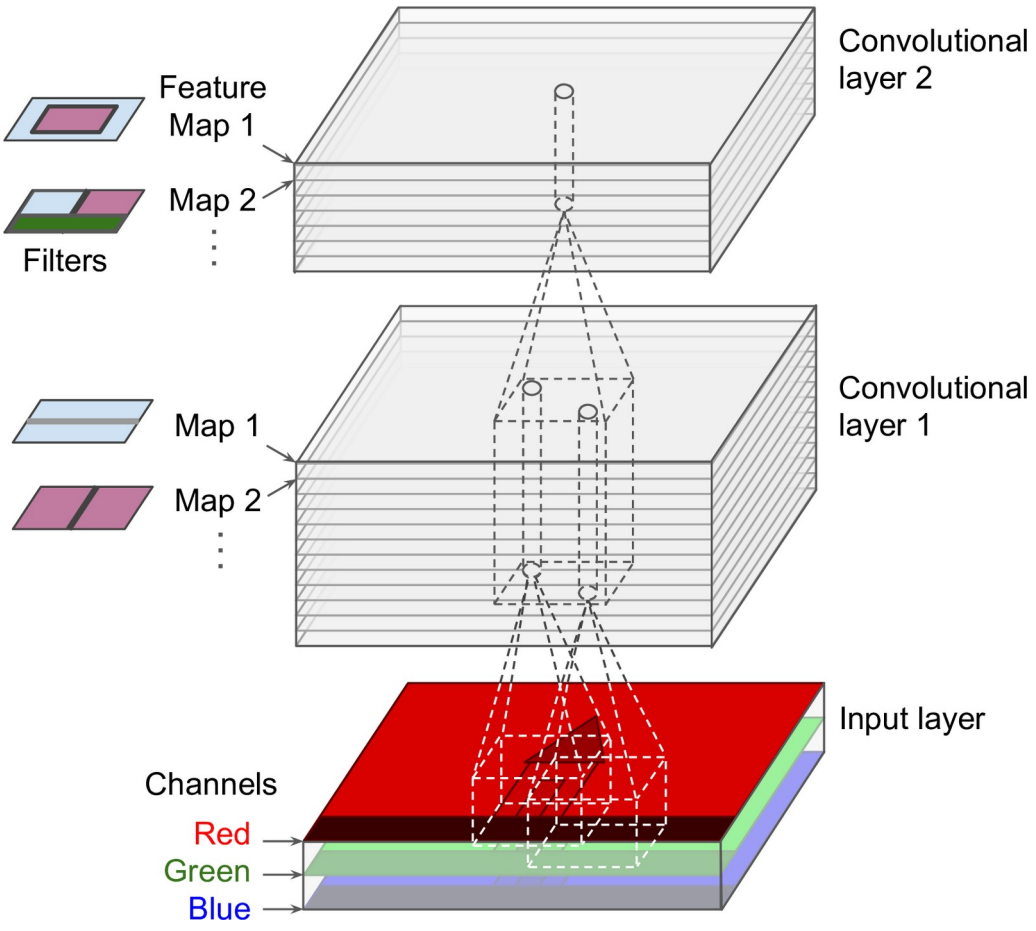

Figure 3.6 Convolution layers with multiple feature maps, and images with three color channels [90]

Specifically, a neuron located in row i, column j of the feature map k during a given convolutional layer l is connected to the outputs of the neurons within the previous layer l – 1, located in rows i × s h to i × s h + f h – 1 and columns j × s w to j × s w + f w – 1, across all feature maps (in layer l – 1). Note that each one neurons located within the same row i and column j but in numerous feature maps are connected to the outputs of the precise same neurons within the previous layer.

3.2.3 Memory Requirements

Another problem with CNNs is that the convolutional layers require a large amount of RAM. this can be very true during training, because the reverse pass of backpro- pagation requires all the intermediate values computed during the pass. For example, consider a convolutional layer with 5 × 5 filters, outputting 200 feature maps of size 150 × 100, with stride 1 and SAME padding. If the input may be a 150 × 100RGB image (three channels), then the amount of parameters is (5 × 5 × 3 + 1) × 200 = 15,200 (the +1 corresponds to the bias terms), which is fairly small compared to a fully connected layer. 7 However, each of the 200 feature maps contains 150 × 100 neurons, and

every of those neurons has to compute a weighted sum of its 5 × 5 × 3 = 75 inputs: that's a complete of 225 million float multiplications. Not as bad as a totally con- nected layer, but still quite computationally intensive. Moreover, if the feature maps are represented using 32-bit floats, then the convolutional layer's output will occupy 200 × 150 × 100 × 32 = 96 million bits (12 MB) of RAM. 8 And that's only for one instance! If a training batch contains 100 instances, then this layer will burn up 1.2 GB of RAM! During inference (i.e., when making a prediction for a brand new instance) the RAM occupied by one layer will be released as soon because the next layer has been computed, so you only need the maximum amount RAM as needed by two consecutive layers. But during training everything computed during the aerial has to be preserved for the reverse pass, that the amount of RAM needed is (at least) the full amount of RAM required by all layers.

3.3 Pooling Layer

Once you understand how convolutional layers work, the pooling layers are quite easy to know. Their goal is to subsample (i.e., shrink) the input image so as to reduce the computational load, the memory usage, and also the number of parameters (thereby limiting the chance of overfitting). Just like in convolutional layers, each neuron in an exceedingly pooling layer is connected to the outputs of a limited number of neurons within the previous layer, located within atiny low rectangular receptive field. you need to define its size, the stride, and therefore the padding type, just like before. However, a pooling neuron has no weights; all it does is aggregate the inputs using an aggregation function like the max or mean. Figure 3.7 shows a max pooling layer, which is that the commonest sort of pooling layer. during this example, we use a 2 × 2 _pooling kernel_ 9 , with a stride of two, and no padding. Only the max input value in each receptive field makes it to the following layer, while the opposite inputs are dropped. as an example, within the lower left receptive field in Figure 3.7, the input values are 1, 5, 3, 2, so only the max value, 5, is propagated to the subsequent layer. Because of the stride of two, the output image has half the peak and half the width of the input image (rounded down since we use no padding).

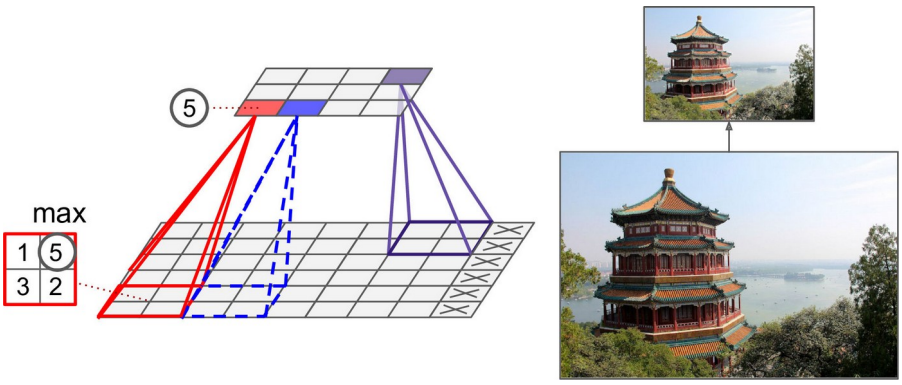

Figure 3.7 Max pooling layer (2 × 2 pooling kernel, stride 2, no padding) [91]

Other than reducing computations, memory usage and therefore the number of parameters, a max pooling layer also introduces some level of invariance to small translations, as shown in Figure 3.8. Here we assume that the brilliant pixels have a lower value than dark pixels, and that we consider 3 images (A, B, C) longing a max pooling layer with a 2 × 2 kernel and stride 2. Images B and C are the identical as image A, but shifted by one and two pixels to the proper. As you'll be able to see, the outputs of the max pooling layer for images A and B are identical. this is often what translation invariance means. However, for image C, the output is different: it's shifted by one pixel to the correct (but there's still 75% invariance). By inserting a max pooling layer every few layers in a CNN, it's possible to induce some level of translation invariance at a bigger scale.

Moreover, max pooling also offers alittle amount of rotational invariance and a slight scale invariance. Such invariance (even if it's limited) are often useful in cases where the prediction shouldn't depend upon these details, like in classification tasks.

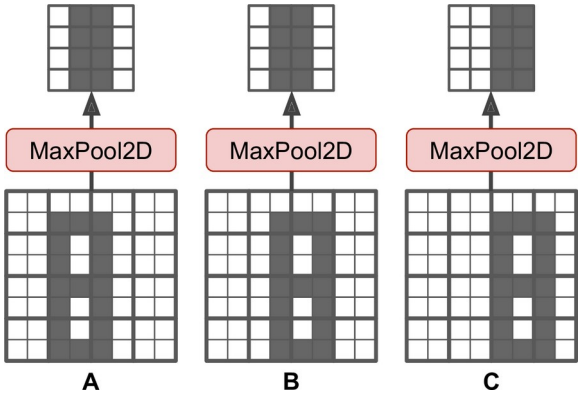

Figure 3.8 Invariance to small translations [92]

But max pooling has some downsides: firstly, it's obviously very destructive: even with a little 2 × 2 kernel and a stride of two, the output are going to be double smaller in both directions (so its area are going to be fourfold smaller), simply dropping 75% of the input values. And in some

applications, invariance isn't desirable, as an example for semantic segmentation: this is often the task of classifying each pixel in a picture looking on the object that pixel belongs to: obviously, if the input image is translated by 1 pixel to the right, the output should even be translated by 1 pixel to the correct. The goal during this case is equivariance, not invariance: alittle change to the inputs should cause a corresponding chickenfeed within the output.

3.4 CNN Architectures

Typical CNN architectures stack some convolutional layers (each one generally fol- lowed by a ReLU layer), then a pooling layer, then another few convolutional layers (+ReLU), then another pooling layer, and so on. The image gets smaller and smaller as it progresses through the network, but it also typically gets deeper and deeper (i.e., with more feature maps) because of the convolutional layers (see Figure 3.9). At the top of the stack, an everyday feedforward neural network is added, composed of some fully connected layers (+ReLUs), and therefore the final layer outputs the prediction (e.g., a softmax layer that outputs estimated class probabilities).

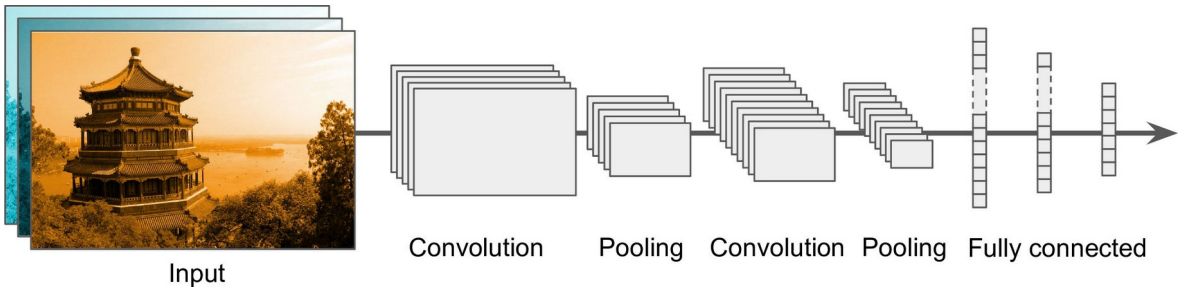

Figure 3.9 Typical CNN architecture [93]

Here is how we can implement a simple CNN to tackle the fashion MNIST dataset:

```
from functools import partial

DefaultConv2D = partial(keras.layers.Conv2D,
        kernel_size=3, activation='relu', padding="SAME")

model = keras.models.Sequential([
        DefaultConv2D(filters=64, kernel_size=7, input_shape=[28, 28, 1]),
        keras.layers.MaxPooling2D(pool_size=2),
```

```
    DefaultConv2D(filters=128),
    DefaultConv2D(filters=128),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=256),
    DefaultConv2D(filters=256),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(units=128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=10, activation='softmax'),
])
```

- In this code, we start by using the partial() function to define a thin wrapper around the Conv2D class, called DefaultConv2D : it simply avoids having to repeat the same hyperparameter values over and over again.
- The first layer uses a large kernel size, but no stride because the input images are not very large. It also sets input_shape=[28, 28, 1] , which means the images are 28 × 28 pixels, with a single color channel (i.e., grayscale).
- Next, we have a max pooling layer, which divides each spatial dimension by a factor of two (since pool_size=2).
- Then we repeat the same structure twice: two convolutional layers followed by a max pooling layer. For larger images, we could repeat this structure several times (the number of repetitions is a hyperparameter you can tune).
- Note that the number of filters grows as we climb up the CNN towards the output layer (it is initially 64, then 128, then 256): it makes sense for it to grow, since the number of low level features is often fairly low (e.g., small circles, horizontal lines, etc.), but there are many different ways to combine them into higher level features. It is a common practice to double the number of filters after each pooling layer: since a pooling layer divides each spatial dimension by a factor of 2, we can afford doubling the number of feature maps in the next layer, without fear of exploding the number of parameters, memory usage, or computational load.

- Next is the fully connected network, composed of 2 hidden dense layers and a dense output layer. Note that we must flatten its inputs, since a dense network expects a 1D array of features for each instance. We also add two dropout layers, with a dropout rate of 50% each, to reduce overfitting.

Over the years, variants of this fundamental architecture are developed, leading to amazing advances within the field. a decent measure of this progress is that the error rate in competitions like the ILSVRC ImageNet challenge. during this competition the top-5 error rate for image classification fell from over 26% to but 2.3% in only six years. The top-five error rate is that the number of test images that the system's top 5 predictions didn't include the right answer. the photographs are large (256 pixels high) and there are 1,000 classes, a number of which are really subtle (try distinguishing 120 dog breeds). observing the evolution of the winning entries may be a great way to under- stand how CNNs work. We will first study the classical LeNet-5 architecture (1998), then three of the winners of the ILSVRC challenge: AlexNet (2012), GoogLeNet (2014), and ResNet (2015).

3.4.1 LeNet-5

The LeNet-5 architecture 10 is perhaps the most widely known CNN architecture. As mentioned earlier, it was created by Yann LeCun in 1998 and widely used for hand- written digit recognition (MNIST).

There are a few extra details to be noted:

- MNIST images are 28 × 28 pixels, but they are zero-padded to 32 × 32 pixels and normalized before being fed to the network. The rest of the network does not use any padding, which is why the size keeps shrinking as the image progresses through the network.
- The average pooling layers are slightly more complex than usual: each neuron computes the mean of its inputs, then multiplies the result by a learnable coeffi- cient (one per map) and adds a learnable bias term (again, one per map), then finally applies the activation function.
- The output layer is a bit special: instead of computing the matrix multiplication of the inputs and the weight vector, each neuron outputs the square of the Euclidian distance between its input vector and its weight vector. Each output measures how much the image belongs to a particular digit class. The cross entropy cost function is now preferred, as it penalizes bad predictions much more, producing larger gradients and converging faster.

3.4.2 AlexNet

The AlexNet CNN architecture 11 won the 2012 ImageNet ILSVRC challenge by a large margin: it achieved 17% top-5 error rate while the runner-up achieved only 26%! it had been developed by Alex Krizhevsky (hence the name), Ilya Sutskever, and Geoffrey Hinton. it's quite just like LeNet-5, only much larger and deeper, and it was the primary to stack convolutional layers directly on top of every other, instead of stacking a pooling layer on top of every convolutional layer. To reduce overfitting, the authors used two regularization techniques: first they applied dropout with a 50% dropout rate during training to the outputs of layers F8 and F9. Second, they performed data augmentation by ran- domly shifting the training images by various offsets, flipping them horizontally, and changing the lighting conditions. AlexNet also uses a competitive normalization step immediately after the ReLU step of layers C1 and C3, called local response normalization. the foremost strongly activated neurons inhibit other neurons located at the identical position in neighboring feature maps (such competitive activation has been observed in biological neurons). This encourages different feature maps to specialize, pushing them apart and forcing them to explore a wider range of features, ultimately improving generalization. A variant of AlexNet called ZF Net was developed by Matthew Zeiler and Rob Fergus and won the 2013 ILSVRC challenge. it's essentially AlexNet with some tweaked hyperparameters (number of feature maps, kernel size, stride, etc.).

3.4.3 GoogLeNet

The GoogLeNet architecture was developed by Christian Szegedy et al. from Google Research, 12 and it won the ILSVRC 2014 challenge by pushing the top-5 error rate below 7%. This great performance came in large part from the very fact that the network was much deeper than previous CNNs (see Figure 3.10). This was made possible by sub-networks called inception modules, 13 which permit GoogLeNet to use parameters far more efficiently than previous architectures: GoogLeNet actually has 10 times fewer parameters than AlexNet (roughly 6 million rather than 60 million). Figure 3.10 shows the architecture of an inception module. The notation “3 × 3 + 1(S)” means the layer uses a 3 × 3 kernel, stride 1, and SAME padding. The input signal is first copied and fed to four different layers. All convolutional layers use the ReLU activation function. Note that the second set of convolutional layers uses different kernel sizes (1 × 1, 3 × 3, and 5 × 5), allowing them to capture patterns at different scales. Also note that each single layer uses a stride of 1 and SAME padding (even the max pooling layer), so their outputs all have the identical height and width as their inputs. This makes it possible to concatenate all the outputs along the depth dimen-

sion within the final depth concat layer (i.e., stack the feature maps from all four top con- volutional layers). This concatenation layer will be implemented in TensorFlow using the tf.concat() operation, with axis=3 (axis 3 is that the depth).

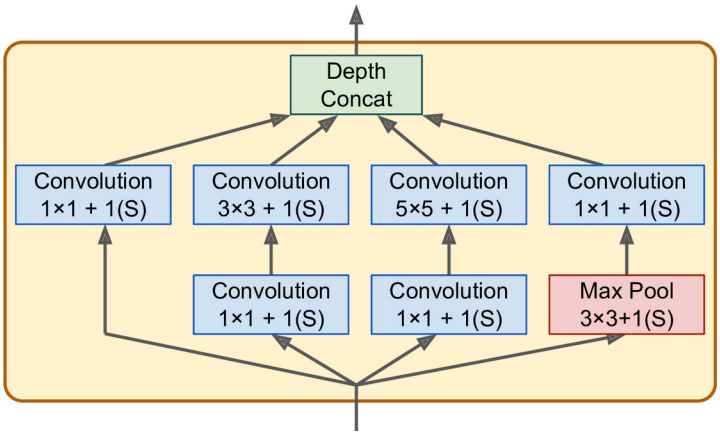

Figure 3.10 Inception module [94]

Now let’s have a look at the architecture of the GoogLeNet CNN (see Figure 3.11). The number of feature maps output by each convolutional layer and every pooling layer is shown before the kernel size. The architecture is so deep that it's to be represented in three columns, but GoogLeNet is really one tall stack, including nine inception modules (the boxes with the spinning tops). The six numbers within the inception modules represent the amount of feature maps output by each convolutional layer within the module (in the identical order as in Figure 3.11). Note that each one the convolutional layers use the ReLU activation function.

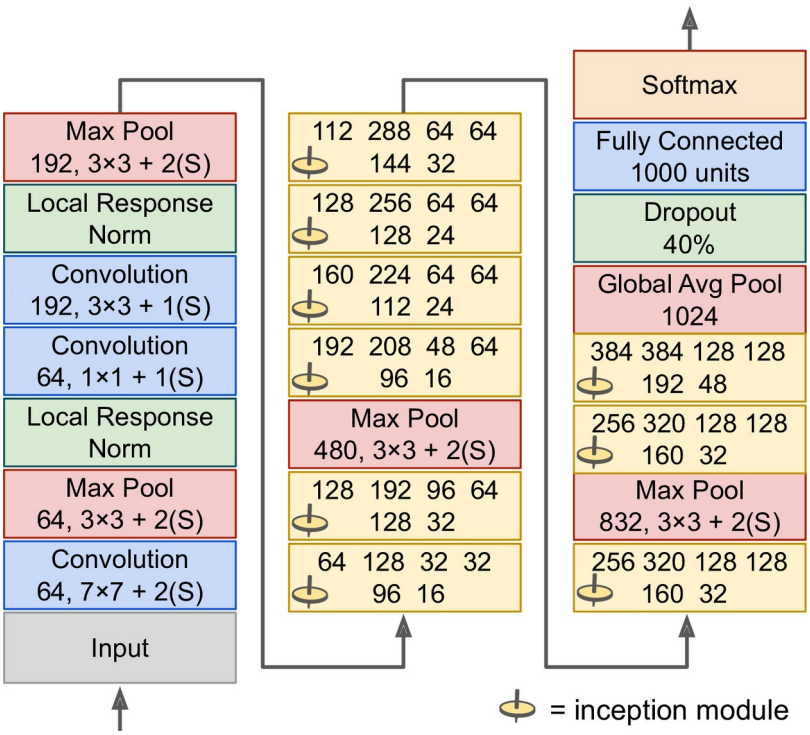

Figure 3.11 GoogLeNet architecture [95]

This diagram is slightly simplified: the first GoogLeNet architecture also included two auxiliary classifiers plugged on top of the third and sixth inception modules. They were both composed of 1 average pooling layer, one convolutional layer, two fully connected layers, and a softmax activation layer. During training, their loss (scaled down by 70%) was added to the loss. The goal was to fight the vanishing gradients problem and regularize the network. However, it absolutely was later shown that their effect was relatively minor.

Several variants of the GoogLeNet architecture were later proposed by Google researchers, including Inception-v3 and Inception-v4, using slightly different inception modules, and reaching even better performance.

3.4.4 VGGNet

The runner up within the ILSVRC 2014 challenge was VGGNet 14 , developed by K. Simonyan and A. Zisserman. It had a really simple and Greco-Roman architecture, with 2 or 3 convolutional layers, a pooling layer, except 2 or 3 convolutional layers, a pooling layer, and then on (with a complete of just 16 convolutional layers), plus a final dense net- work with 2 hidden layers and therefore the output layer. It used only 3 × 3 filters, but many filters.

3.4.5 ResNet

The ILSVRC 2015 challenge was won employing a Residual Network (or ResNet), devel- oped by Kaiming He et al., 15 which delivered an astounding top-5 error rate under 3.6%, using a particularly deep CNN composed of 152 layers. It confirmed the final trend: models have gotten deeper and deeper, with fewer and fewer parameters. The key to having the ability to coach such a deep network is to use skip connections (also called shortcut connections): the signal feeding into a layer is additionally added to the output of a layer located a touch in a higher place the stack. Let's see why this can be useful. When training a neural network, the goal is to create it model a target function h(x). If you add the input x to the output of the network, then the network are going to be forced to model f(x) = h(x) – x instead of h(x).

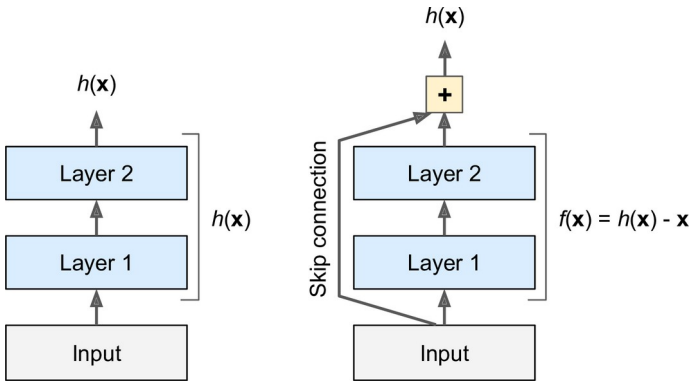

Figure 3.12 Residual learning [96]

When you initialize an everyday neural network, its weights are near zero, therefore the network just outputs values near zero. If you add a skip connection, the resulting network just outputs a replica of its inputs; in other words, it initially models the identity function. If the target function is fairly near the identity function (which is commonly the case), this may speed up training considerably.

Moreover, if you add many skip connections, the network can start making progress even if several layers haven't started learning yet (see Figure 3.13). because of skip connections, the signal can easily make its way across the entire network. The deep residual network will be seen as a stack of residual units, where each residual unit may be a small neural network with a skip connection.

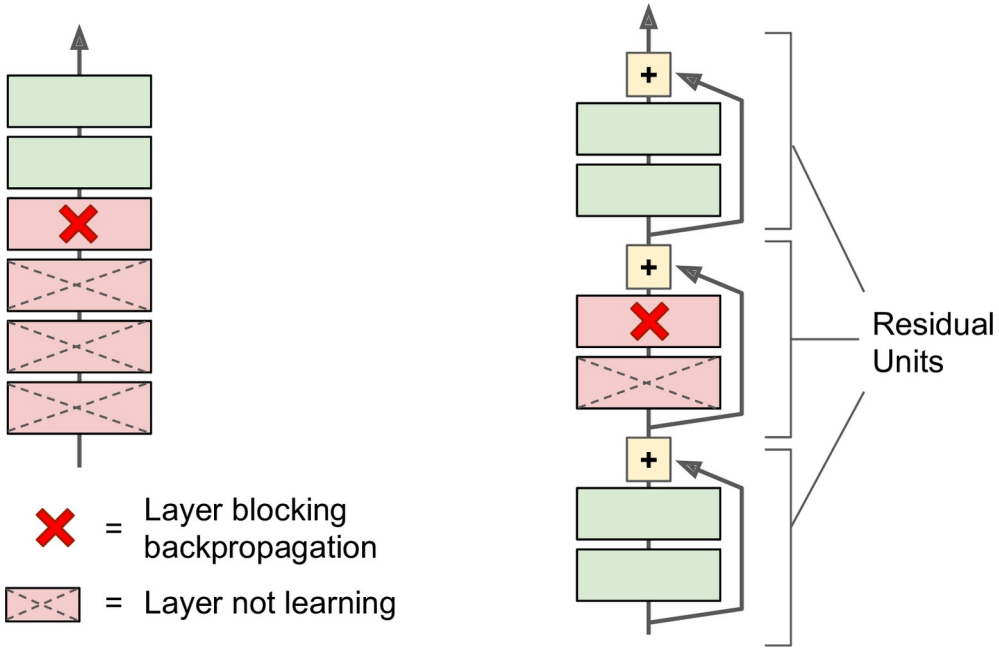

Figure 3.13 Regular deep neural network (left) and deep residual network (right) [97]

Now let's observe ResNet's architecture (see Figure 3.14). it's actually surprisingly simple. It starts and ends exactly like GoogLeNet (except without a dropout layer), and in between is simply a really deep stack of easy residual units. Each residual unit is composed of two convolutional layers (and no pooling layer!), with Batch Normalization (BN) and ReLU activation, using 3 × 3 kernels and preserving spatial dimensions (stride 1, SAME padding).

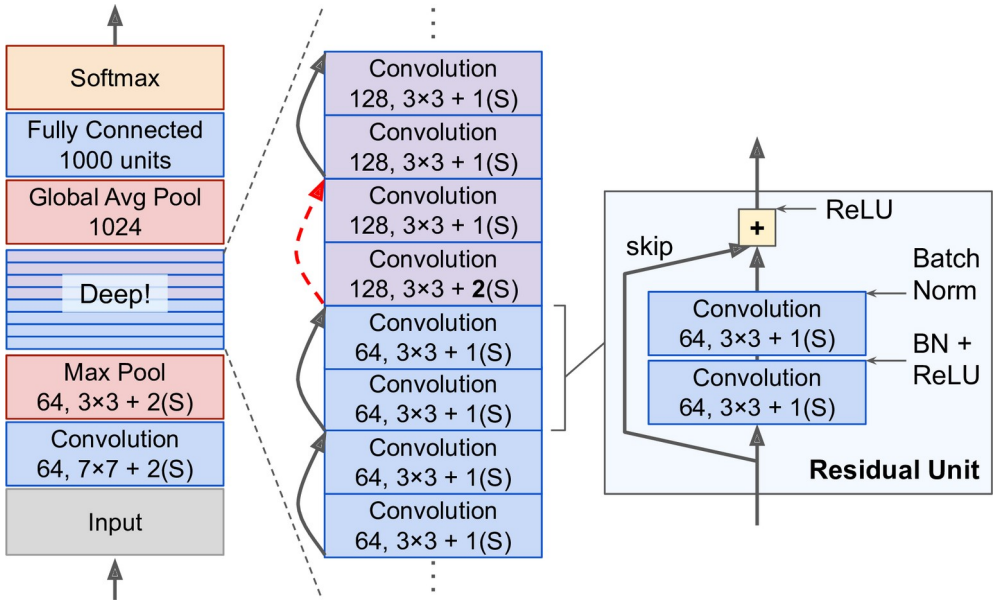

Figure 3.14 ResNet architecture [98]

Note that the amount of feature maps is doubled every few residual units, at the identical time as their height and width are halved (using a convolutional layer with stride 2). When this happens the inputs can not be added on to the outputs of the residual unit since they don't have the identical shape (for example, this problem affects the skip connection represented by the dashed arrow in Figure 3.15). to unravel this problem, the inputs are older a 1 × 1 convolutional layer with stride 2 and therefore the right number of output feature maps (see Figure 3.15).

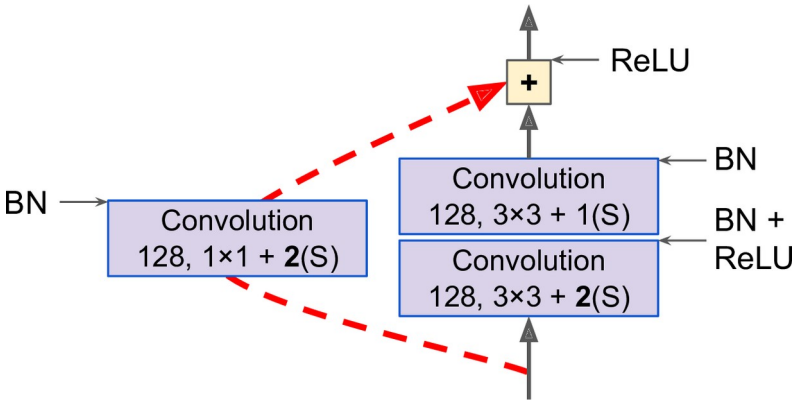

Figure 3.15 Skip connection when changing feature map size and depth [99]

ResNet-34 is that the ResNet with 34 layers (only counting the convolutional layers and the fully connected layer) containing three residual units that output 64 feature maps, 4 RUs with 128 maps, 6 RUs with 256 maps, and three RUs with 512 maps. we'll imple- ment this architecture later during this chapter.

3.4.6 Xception

Another variant of the GoogLeNet architecture is additionally worth noting: Xception (which stands for Extreme Inception) was proposed in 2016 by François Chollet (the author of Keras), and it significantly outperformed Inception-v3 on an enormous vision task (350 million images and 17,000 classes). similar to Inception-v4, it also merges the ideas of GoogLeNet and ResNet, but it replaces the inception modules with a special type of layer called a depthwise separable convolution (or separable convolution for short 18). These layers had been used before in some CNN architectures, but they were not as central as within the Xception architecture. While an everyday convolutional layer uses filters that try and simultaneously capture spatial patterns (e.g.,

an oval) and cross-channel patterns (e.g., mouth + nose + eyes = face), a separable convolutional layer makes the strong assumption that spatial patterns and cross-channel patterns are often modeled separately (see Figure 3.16). Thus, it's composed of two parts: the primary part applies one spatial filter for every input feature map, then the second part looks exclusively for cross-channel patterns—it is simply a daily convolutional layer with 1 × 1 filters.

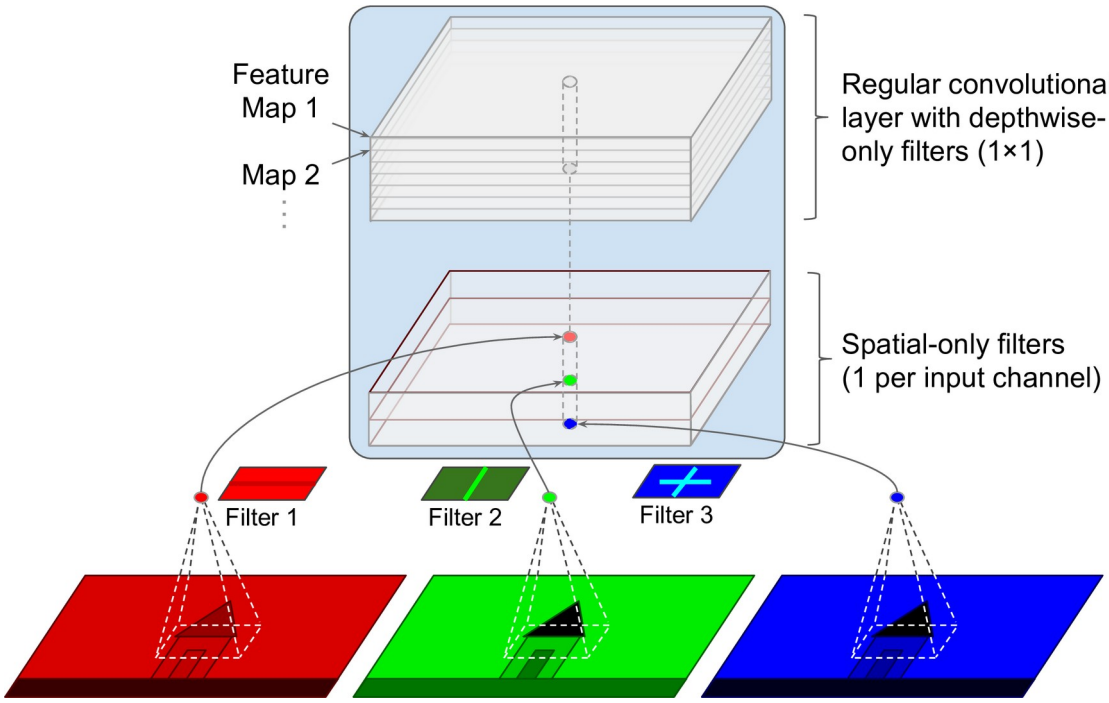

Figure 3.16 Depthwise Separable Convolutional Layer [100]

Since separable convolutional layers only have one spatial filter per input channel, you should avoid using them after layers that have too few channels, like the input layer (granted, that’s what Figure 3.16 represents, but it's only for illustration purposes). For this reason, the Xception architecture starts with 2 regular convolutional layers, on the other hand the remainder of the architecture uses only separable convolutions (34 in all), plus some max pooling layers and therefore the usual final layers (a global average pooling layer, and a dense output layer). We might wonder why Xception is taken into account a variant of GoogLeNet, since it con- tains no inception module at all? Well, as we discussed earlier, an Inception module contains convolutional layers with 1 × 1 filters: these look exclusively for cross-channel patterns. However, the convolution layers that sit on top of them are regular convolutional layers that look both for spatial and cross-channel patterns. So you can think of an Inception module as an intermediate between an everyday convolutional layer (which considers spatial patterns and cross-channel patterns jointly) and a separable convolutional layer

(which considers them separately). In practice, it seems that separable convolutions generally perform better.

The ILSVRC 2016 challenge was won by the CUImage team from the Chinese Uni- versity of port. They used an ensemble of the many different techniques, including a complicated object-detection system called GBD-Net 19 , to realize a top-5 error rate below 3%. Although this result's unquestionably impressive, the complexity of the solution contrasted with the simplicity of ResNets. Moreover, one year later another fairly simple architecture performed even better, as we are going to see now.

3.4.7 SENet

The winning architecture within the ILSVRC 2017 challenge was the Squeeze-and- Excitation Network (SENet) 20 . This architecture extends existing architectures like inception networks or ResNets, and boosts their performance. This allowed SENet to win the competition with an astonishing 2.25% top-5 error rate! The extended versions of inception networks and ResNet are called SE-Inception and SE-ResNet respectively. The boost comes from the actual fact that a SENet adds alittle neural network, called a SE Block, to each unit within the original architecture (i.e., every inception module or every residual unit), as shown in Figure 3.17.

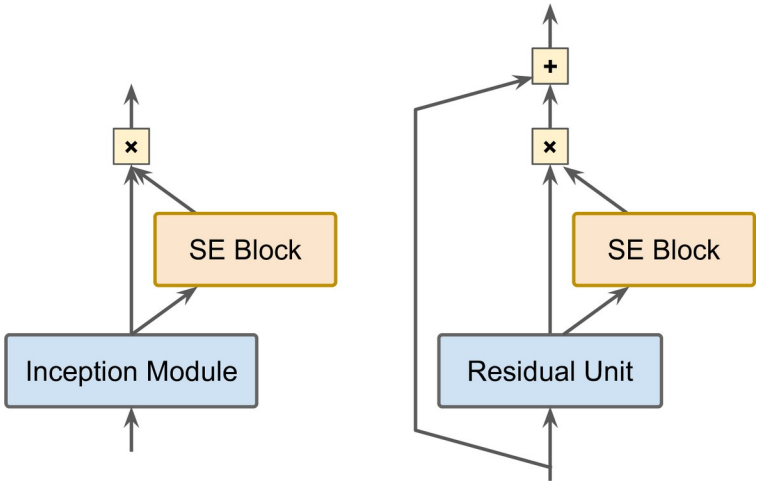

Figure 3.17 SE-Inception Module (left) and SE-ResNet Unit (right) [101]

A SE Block analyzes the output of the unit it's attached to, focusing exclusively on the depth dimension (it doesn't rummage around for any spatial pattern), and it learns which features are usually most active together. It then uses this information to recalibrate the feature maps, as shown

in Figure 3.18. as an example, a SE Block may learn that mouths, noses and eyes usually appear together in pictures: if you see a mouth and a nose, you must expect to work out eyes further. So if a SE Block sees a powerful activation in the mouth and nose feature maps, but only mild activation within the eye feature map, it will boost the attention feature map (more accurately, it'll reduce irrelevant feature maps). If the eyes were somewhat confused with something else, this feature map recalibration will help resolve the paradox.

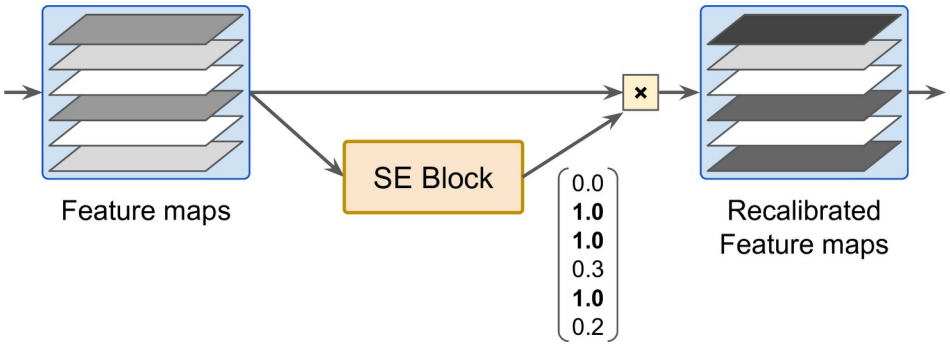

Figure 3.18 An SE Block Performs Feature Map Recalibration [102]

A SE Block consists of just 3 layers: a worldwide average pooling layer, a hidden dense layer using the ReLU activation function, and a dense output layer using the sigmoid activation function (see Figure 3.19):

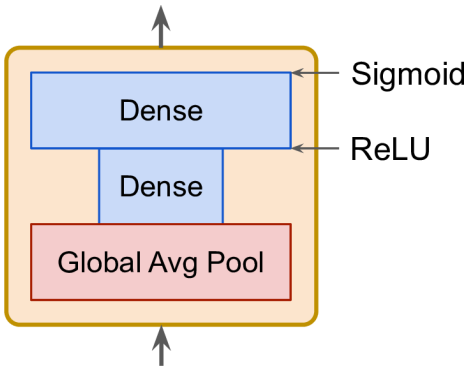

Figure 3.19 SE Block Architecture [103]

As earlier, the world average pooling layer computes the mean activation for every feature map: as an example, if its input contains 256 feature maps, it'll output 256 numbers representing the general level of response for every filter. the subsequent layer is where the “squeeze” happens: this

layer has much but 256 neurons, typically 16 times less than the amount of feature maps (e.g., 16 neurons), therefore the 256 numbers get com- pressed into atiny low vector (e.g., 16 dimen- sional).this is often a low-dimensional vector representation (i.e., an embedding) of the distribution of feature responses. This bot- tleneck step forces the SE Block to find out a general representation of the feature combinations (we will see this principle in action again once we discuss autoencoders in ???). Finally, the output layer takes the embedding and outputs a recalibration vector containing one number per feature map (e.g., 256), each between 0 and 1. The feature maps are then multiplied by this recalibration vector, so irrelevant features (with a coffee recalibration score) get scaled down while relevant features (with a recalibration score near 1) are left alone.

3.5 Semantic Segmentation

In semantic segmentation, each pixel is assessed in step with the category of the article it belongs to (e.g., road, car, pedestrian, building, etc.), as shown in Figure 3.20. Note that different objects of the identical class don't seem to be distinguished. as an example, all the bicycles on the correct side of the segmented image find yourself in concert big lump of pixels. The main difficulty during this task is that when images bear an everyday CNN, they gradually lose their spatial resolution (due to the layers with strides greater than 1): so a regular CNN may find yourself knowing that there’s an individual within the image, somewhere in the bottom left of the image, but it'll not be way more precise than that.

Figure 3.20 Semantic segmentation [104]

Just like for object detection, there are many alternative approaches to tackle this problem, some

quite complex. However, a reasonably simple solution was proposed within the 2015 paper by Jonathan Long et al. we discussed earlier. they begin by taking a pretrained CNN and turning into an FCN, as discussed earlier. The CNN applies a stride of 32 to the input image overall (i.e., if we add up all the strides greater than 1), meaning the last layer outputs feature maps that are 32 times smaller than the input image. This is clearly too coarse, so that they add one upsampling layer that multiplies the resolution by 32. There are several solutions available for upsampling (increasing the scale of an image), like bilinear interpolation, but it only works reasonably arise to ×4 or ×8. Instead, they used a transposed convolutional layer: 31 it's reminiscent of first stretching the image by inserting empty rows and columns (full of zeros), then per- forming a daily convolution (see Figure 3.21). Alternatively, some people favor to think of it as an everyday convolutional layer that uses fractional strides (e.g., 1/2 in Figure 3.21). The transposed convolutional layer is initialized to perform something near linear interpolation, but since it's a trainable layer, it'll learn to try to to better during training.

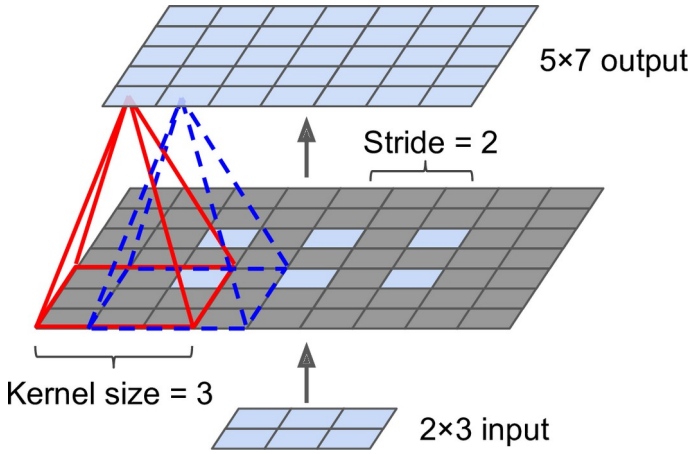

Figure 3.21 Upsampling Using a Transpose Convolutional Layer [105]

This solution is okay, but still too imprecise. to try and do better, the authors added skip connections from lower layers: as an example, they upsampled the output image by an element of 2 (instead of 32), and that they added the output of a lower layer that had this double resolution. Then they upsampled the result by an element of 16, resulting in a complete upsampling factor of 32 (see Figure 3.22). This recovered a number of the spatial resolution that was lost in earlier pooling layers. In their best architecture, they used a second similar skip connection to recover even finer details from a fair lower layer: in short, the output of the initial CNN goes through the subsequent extra steps: upscale ×2, add the output of a lower layer (of the suitable scale), upscale ×2, add the output of a good lower layer, and at last upscale ×8. it's even possible to proportion beyond the

dimensions of the first image: this will be wont to increase the resolution of an image, which may be a technique called super-resolution.

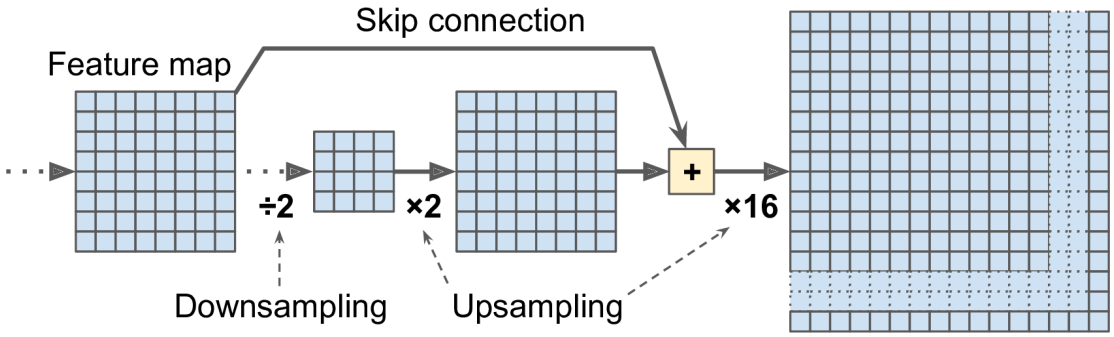

Figure 3.22 Skip layers recover some spatial resolution from lower layers [106]

Once again, many github repositories provide TensorFlow implementations of semantic segmentation (TensorFlow 1 for now), and you'll even find a pretrained instance segmentation model within the TensorFlow Models project. Instance segmentation is analogous to semantic segmentation, but rather than merging all objects of the same class into one big lump, each object is distinguished from the others (e.g., it identifies each individual bicycle). At this, they supply multiple implementations of the Mask R-CNN architecture, which was proposed in an exceedingly 2017 paper: it extends the Faster R-CNN model by additionally producing a pixel-mask for every bounding box. So not only does one get a bounding box around each object, with a set of estimated class probabilities, you furthermore may get a pixel mask that locates pixels within the bounding box that belong to the thing. A we'll be able to see, the sector of Deep Computer Vision is vast and moving fast, with al sorts of architectures commencing per annum, all supported Convolutional Neural Net- works. The progress made in exactly some years has been astounding,
and researchers are now specializing in harder and harder problems, like adversarial learning (which attempts to create the network more proof against images designed to fool it), explainability (understanding why the network makes a particular classification), realistic image generation (which we'll come to in ???), single-shot learning (a system that may recognize an object after it's seen it just once), and far more. Some even explore completely novel architectures, like Geoffrey Hinton’s capsule networks 32 (I pre- sented them in an exceedingly couple videos, with the corresponding code during a notebook). Now on to the subsequent chapter, where we are going to study the way to process sequential data like time series using Recurrent Neural Networks and Convolutional Neural Networks.

IMPLEMENTATION & RESULT EVALUATION

Recognition of infected sporozoan begins with image preprocessing techniques. To improve local brightness and contrast each pixel intensity is subtracted from mean intensity of the image. to acknowledge infected malaria CNN based techniques are ideal, because it can mimic human intelligence. Training a brand new CNN from scratch on the image dataset yields a reasonable result without performing any custom image enhancement, segmentation or feature extraction. But, it requires an oversized amount of knowledge to coach the model. This section elaborates the selection process of pre-trained CNN model and its unification with another classifier for better recognition of infected malaria.

4.1 Visualizing The Features

Analyzing the extracted features helps in understanding the impact of the learned parameters at various stages of the network. Study of literature reveals insufficient discussion on the impact of problem-specific features learned or the activations at different layers of the network. This study aims to analyze the features and activations learned by the proposed model that are specific to identifying the traditional and parasitemic cells to help in malaria screening. The features learned at different stages (layers) of the model are investigated to visualise the parameters learned from the training examples. this is often done by generating images that strongly activate a specific channel of the network layers. The first convolution layer learns 20 features which are visualized. the photographs mostly contain colors and edges, indicating that the channels are color filters and edge detectors. this permits the proposed model to construct useful complex features within the deeper layers. The features on the second convolutional layer are crafted using the features from the primary convolutional layer. the primary 30 features learned by this layer are visualized, where we observe that the model begins to find out high-level features including the form and placement of the parasites along with the colour and texture information. The third convolutional layer learns the features by combining the low-level features from the primary and second convolutional layers. it's observed that this layer, deeper into the network, yields detailed information on the form, color and texture of the features. An instance of the uninfected and parasitemic cells. The fully connected layer towards the tip of the network learns high-level abstractions of the features learned by the sooner layers and outputs two channels such as the traditional and parasitemic cell classes respectively and are visualized. a better scrutinize the

features learned by this layer shows that they resemble the uninfected and parasitemic classes, respectively.

4.2 Visualizing The Activations

The activations of the various layers of the network are visualized to know the model learning strategy. The features learned by the network are evaluated by examining the activations and comparing them with the first image. A parasitemic cell image is read into the model for visualizing the activations at different layers of the network. The first convolutional layer performs convolutions with the input and therefore the features are investigated by observing the areas where the layer activates on the input image and comparing them with the corresponding areas within the original image.

The activations are returned as a three-dimensional array where the dimension represents the quantity of channels in a given layer. A montage for activations in each layer, one for every channel within the layer. The output channel within the first convolutional layer is displayed as squares within the montage of activations. Strong positive activations are represented by white pixels and negative activations, by black pixels. A gray channel doesn't activate as strongly on the initial input. The position of a pixel in a very given activation corresponds to the identical position within the original image, e.g., a white pixel at a given location in an exceedingly channel activation indicates that the channel is strongly activated at that position. this can be correlated with the first image to verify location of the parasites. The original images and also the activations for the 11 th channel for an example image. The channel activations are resized to have the identical size because the original image. the very best channel activation, identified as white pixels, corresponds t the location of the parasites within the original image. This cumbersome manual process of identifying interesting channels is alleviated by programmatically investigating the channels for activations with large values. This automation ends up in a greater throughput within the analysis leading to a far better understanding of the learned parameters. The proposed model learns to detect features like colors and edges in its first convolution layer and more complex (and abstract) concepts in deeper layers. These deep layers extract this information by developing their features by combining the features from the early-stage layers. We demonstrate this by investigating the third convolutional layer in an exceedingly manner the same as the primary convolutional layer. The activations are displayed as a montage. Within the montage of all channels, the 40 th channel activates strongly on the situation of the parasites. The channel contains both positive and negative activations but only positive activations are used due to the presence of ReLU non-linearity, following the convolutional layers.

4.3 Model Evaluation

4.3.1 Data Source

We used archived blood smear images acquired from Chittagong Medical College Hospital, Bangladesh, and segmented the visual region of the erythrocytes from the first images. Our data set contains 22K erythrocyte images where the ratio of infected cells to uninfected cells is 1:1. All images are normalized to the median width and height for the training and classification experiments, at 128*128 pixels, with three color channels.

4.3.2 Data Preprocessing

All images are read into Google Colab, resized if needed, and serialized to get the input to the MatConvnet toolbox. Before we pass the information to the CNN network for training, we apply a normalization to enhance local brightness and contrast, and whiten the complete dataset using an eigenvalue decompositio (EVD) operation on the covariance matrix.

4.3.3 Dataset Complilation

After the info preprocessing, we randomly selected an outsized number of cell images and provided them to pathologists at the University of Alabama at Birmingham. the whole whole slide image dataset are divided into four segments evenly. Each of 4 pathologists are assigned with two segments so each cell image are viewed and labeled by a minimum of two experienced pathologists. One cell image can only be considered as infected and included in our final dataset if all the reviewers mark it positively whereas it will be excluded otherwise. the identical selection rule also applies to the non-infected cells in our dataset. After this data curation, we collected 13,034 infected cells and 13,531 non-infected cells. Next we divide this dataset into two sets of approximately equal size: training set and testing set. Table II shows the quantity of infected and non-infected cells for training set and testing set.

4.3.4 CNN Model Training

In order to coach and evaluate our CNN model, we implement a ten-fold cross-validation on the entire data set, where 90% of the images are used for training, and 10% are used for testing. In model training, 90% of the photographs are separated from the training set for the particular training and therefore the remaining 10% are used for back-propagation validation. The performance evaluation criteria are the common accuracy, sensitivity, specificity, precision, F1

score, and Matthews parametric statistic over the ten-fold cross- validation. A pre-trained AlexNet supported the CIFAR-100 data set is applied because the feature extractor for transfer learning. it's linked to a standard SVM classifier to implement transfer learning as a comparison to our CNN model.

4.4 Working Process

4.4.1 Importing Necessary Libraries

```
import tensorflow as tf
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras import Sequential
from tensorflow.keras.layers import
Conv2D,MaxPool2D,Dropout,Flatten,Dense,BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
```

4.4.2 Displaying Uninfected And Infected Cell Tissues

```
upic='../input/cell-images-for-detecting-malaria/cell_images/Uninfected/
C100P61ThinF_IMG_20150918_144104_cell_131.png'
apic='../input/cell-images-for-detecting-malaria/cell_images/Parasitized/
C100P61ThinF_IMG_20150918_144104_cell_164.png'
plt.figure(1, figsize = (15 , 7))
plt.subplot(1 , 2 , 1)
plt.imshow(cv2.imread(upic))
plt.title('Uninfected Cell')
plt.xticks([]) , plt.yticks([])
plt.subplot(1 , 2 , 2)
plt.imshow(cv2.imread(apic))
plt.title('Infected Cell')
plt.xticks([]) , plt.yticks([])
```

plt.show()

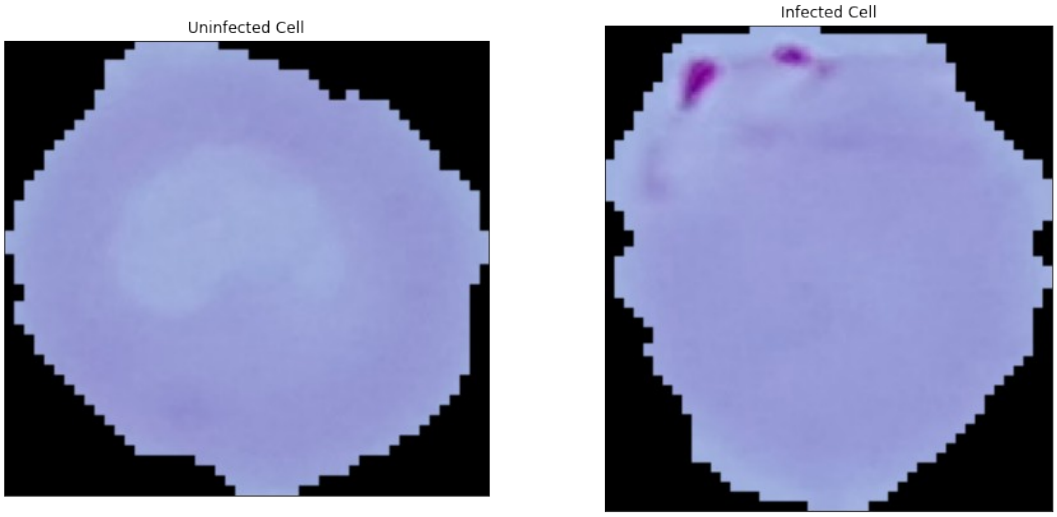

Figure 4.1 Displaying Uninfected and Infected Cell tissues

4.4.3 Set Height Width in PX

width = 128

height = 128

4.4.4 Dividing Dataset into Two Folders Train And Test

datagen = ImageDataGenerator(rescale=1/255.0, validation_split=0.2)

4.4.5 Preparing Train And Test Image Generator

trainDatagen = datagen.flow_from_directory(directory='../input/cell-images-for-detecting-malaria/cell_images/cell_images/',

target_size=(width,height),

class_mode = 'binary',

batch_size = 16,

subset='training')

valDatagen = datagen.flow_from_directory(directory='../input/cell-images-for-detecting-malaria/cell_images/cell_images/',

target_size=(width,height),

```
class_mode = 'binary',
batch_size = 16,
subset='validation')
```

4.4.6 Preparing The Model

```
model = Sequential()
model.add(Conv2D(16,(3,3),activation='relu',input_shape=(128,128,3)))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.2))
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.3))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))

model.summary()
```

4.4.7 Summary of Model

Table 4.1 Summary of Sequential Model

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 16)	448
max_pooling2d (MaxPooling2D)	(None, 63, 63, 16)	0
dropout (Dropout)	(None, 63, 63, 16)	0
conv2d_1 (Conv2D)	(None, 61, 61, 32)	4640

```
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 64)                802880
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
                      Total params: 826,529
                    Trainable params: 826,529
                     Non-trainable params: 0
_________________________________________________________________
```

4.4.8 Model Compilation

```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
early_stop = EarlyStopping(monitor='val_loss',patience=2)
history = model.fit_generator(generator = trainDatagen,
                    steps_per_epoch = len(trainDatagen),
                     epochs =20,
                     validation_data = valDatagen,
                     validation_steps=len(valDatagen),
                    callbacks=[early_stop])
```

4.4.9 Displaying Epoch for Best Result

Table 4.2 First Seven Epoch for Best Result

```
Epoch 1/20
1378/1378 [==============================] - 87s 63ms/step - loss: 0.3928 -
accuracy: 0.8332 - val_loss: 0.1934 - val_accuracy: 0.9307
Epoch 2/20
1378/1378 [==============================] - 47s 34ms/step - loss: 0.1788 -
accuracy: 0.9465 - val_loss: 0.1971 - val_accuracy: 0.9405
Epoch 3/20
1378/1378 [==============================] - 47s 34ms/step - loss: 0.1571 -
accuracy: 0.9535 - val_loss: 0.1701 - val_accuracy: 0.9410
```

```
Epoch 4/20
1378/1378 [==============================] - 48s 35ms/step - loss: 0.1495 -
accuracy: 0.9557 - val_loss: 0.1789 - val_accuracy: 0.9416
Epoch 5/20
1378/1378 [==============================] - 48s 35ms/step - loss: 0.1361 -
accuracy: 0.9568 - val_loss: 0.1660 - val_accuracy: 0.9448
Epoch 6/20
1378/1378 [==============================] - 47s 34ms/step - loss: 0.1320 -
accuracy: 0.9591 - val_loss: 0.1665 - val_accuracy: 0.9403
Epoch 7/20
1378/1378 [==============================] - 47s 34ms/step - loss: 0.1274 -
accuracy: 0.9591 - val_loss: 0.1741 - val_accuracy: 0.9452
```

4.4.10 Plot The Result

```
def plotLearningCurve(history,epochs):
    epochRange = range(1,epochs+1)
    plt.plot(epochRange,history.history['accuracy'])
    plt.plot(epochRange,history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train','Validation'],loc='upper left')
    plt.show()
    plt.plot(epochRange,history.history['loss'])
    plt.plot(epochRange,history.history['val_loss'])
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train','Validation'],loc='upper left')
    plt.show()

plotLearningCurve(history,7)
```

4.5 Simulation Result And Analysis

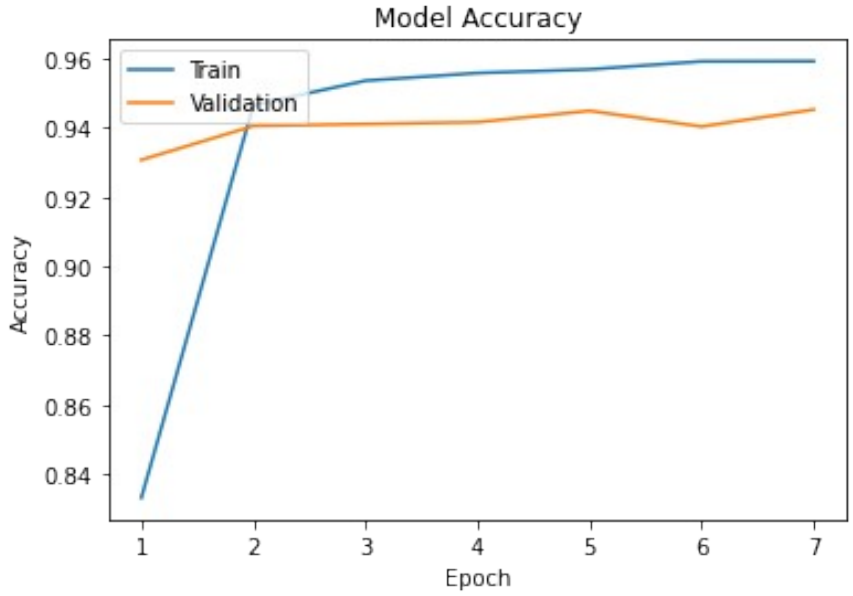

Figure 4.2 Simulation Result for Accuracy of Training and Validation

Here, we can see the simulation result and analysis for malaria detecting cell. The training accuracy of the model is 96% and also the validation accuracy is 94% after seven epoch done which is better then previous related work.

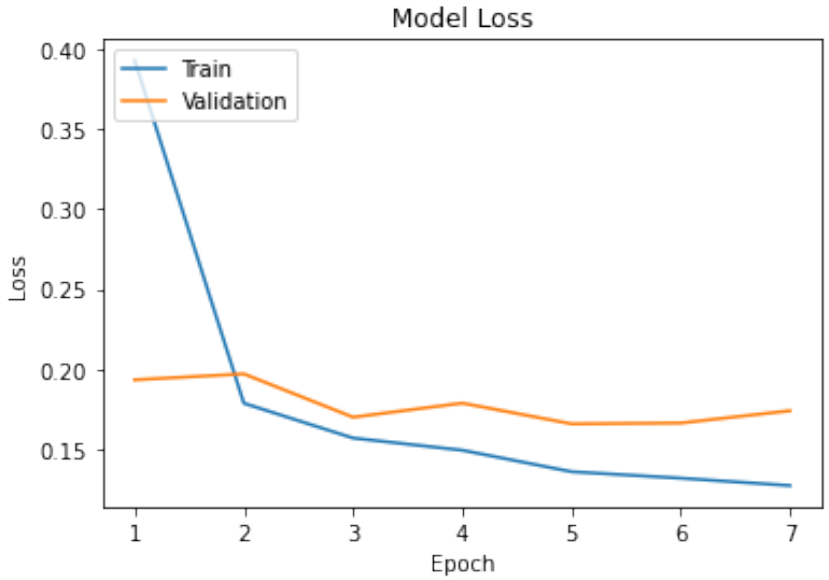

Figure 4.3 Simulation Result for Loss of Training and Validation

Here, we also can see the simulation result and analysis for malaria detecting cell. The training loss of the model is 4% and also the validation accuracy is 14% after seven epoch done which is better then previous related work.

CONCLUSION & FUTURE WORK

The theme of this thesis is to teach the machine about malaria cell. After that machine can analyze the malaria cell and can detect it. This thesis is based on machine learning where convolutional neural networks applied for the visualizing of result. In the following, the main contributions and insights of this thesis are summarized and some possible lines for future works are discussed.

5.1 Conclusion

This thesis conclude here, where the accuracy of training is 96% and the validation accuracy is 94%. That means machine can understand all most all infected cell in malaria and can detect that successfully. CNN in machine learning is used to train the machine to decide this malaria infected cell and also to decide which cell is not infected.

5.2 Future Work

We expect that deep learning will significantly improve the working efficiency and accuracy of malaria diagnosis and other health-related applications, following our previous studies on deep learning for genomics.

REFERENCES

[1] May Z, Mohd Aziz SSA, Salamat R (2013) Automated quantification and classification of malaria parasites in thin blood smears. IEEE International Conference on Signal and Image Processing Applications, Melaka, pp 369–373

[2] World Health Organization World Malaria Report – 2017. http://www.who.int/malaria/publications/world-malaria-report-2017/en. Accessed Jan 2018

[3] Chavan SN, Sutkar AM (2014) Malaria disease identification and analysis using image processing. Int J Latest Trends Eng Technol 3:218–223

[4] Poostchi M, Silamut K, Maude RJ, Jaeger S, Thoma G (2018) Image analysis and machine learning for detecting malaria. Trans Res: J Lab Clin Med 194:36–55

[5] Rajaraman S et al. (2018) Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. Peer J

[6] Savkare SS, Narote SP (2015) Automated system for malaria parasite identification. International Conference on Communication, Information & Computing Technology (ICCICT), Mumbai, pp 1–4

[7] Das DK, Maiti AK, Chakraborty C (2015) Automated system for characterization and classification of malaria-infected stages using light microscopic images of thin blood smears. J Microsc 257 (3):238–252

[8] Dong Y et al (2017) Evaluations of deep convolutional neural networks for automatic identification of malaria infected cells. IEEE EMBS International Conference on Biomedical & Health Informatics (BHI), Orlando, pp 101–104

[9] Makkapati VV, Rao RM (2011) Ontology-based malaria parasite stage and species identification from peripheral blood smear images. Ann Int Conf IEEE Eng Med Biol Soc:6138–6141

[10] K. Jarrett, K. Kavukcuoglu, M.A. Ranzato and Y. LeCun, “What is the best multi-stage architecture for object recognition?” In International Conference on Computer Vision, IEEE, 2009, pp. 2146-2153.

[11] R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” Communications of the ACM, vol. 15, no. 1, pp. 11–15, 1972.

[12] Schmidhuber, J. Deep Learning in Neural Networks: An Overview. Neural Networks, 61(February 2015), 85–117.

[13] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15 (June 2014), 1929–1958.
[14] Mathworks.http://www.mathworks.com/discovery/dee-p-learning.html.
[15] Krizhevsky, A., Sutskever, I. and Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of NIPS'12 (Lake Tahoe NA, December 2012), 1097–1105.
[16] Simonyan, K. and Zisserman, A. Very Deep Convolutional Networks for large scale image recognition. arXiv preprint arXiv: 1409,1556, 2015.
[17] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In Proceedings of CVPR'09 (Miami FL, June 2009), IEEE Press, 248–255.
[18] Convolutional Neural Networks for Visual Recognition.http://cs231n.github.io/transferlearning
[19] Schwenk, H. and Bengio, Y. Boosting neural networks. Neural Computation, 2 (August 2000), 1869–1887.
[20] D. K. Das, et al. , "Machine learning approach for automated screening of malaria parasite using light microscopic images," Journal of Micron, vol. 45, pp. 97–106, Feb 2013.
[21] F. B. Tek, A. G. Dempster and I. Kale, "Parasite detection and identification for automated thin blood film malaria diagnosis," Journal of Computer Vision and Image Understanding, vol. 114, no. 1, pp. 21–32, January 2010.
[22] N. E. Ross, et al., "Automated image processing method for the diagnosis and classification of malaria on thin blood smears," Medical and Biological Engineering and Computing, vol. 44, no. 5, pp. 427–436, May 2005.
[23] F. B. Tek, "Computerised diagnosis of malaria," Ph.D thesis, University of Westminster, 2007.
[24] V. Muralidharan, Y. Dong, and W. D. Pan, "A comparison of feature selection methods for machine learning based automatic malarial cell recognition in wholeslide images," in 2016 IEEE EMBS International Conference on Biomedical and Health Informatics (BHI), Feb 2016, pp. 216–219.
[25] J. A. Quinn, R. Nakasi, P. K. Mugagga, P. Byanyima, W. Lubega, and A. Andama, "Deep convolutional neural networks for microscopy-based point of care diagnostics," arXiv preprint arXiv:1608.02989, 2016.
[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.
[27] Y. LeCun, K. Kavukvuoglu, and C. Farabet, "Convolutional networks and applications in vision," in Proc. International Symposium on Circuits and Systems (ISCAS'10). IEEE, 2010.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutinal neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.
[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.
[30] F.Tokumasu, R.M. Fairhurst and G.R. Ostera, “Band 3 modifications in Plasmodium falciparum-infected AA and CC erythrocytes assayed by autocorrelation analysis using quantum dots.” Journal of Cell Science, 2005, 118(5), pp. 1091-1098.
[31] D.H. Hubel and T.N. Wiesel, “Receptive fields of single neurones in the cat's striate cortex.” The Journal of physiology. 1959 Oct 1;148(3), pp. 574-91.
[32] I. Arel, D.C. Rose and T.P. Karnowski, “Deep machine learning-a new frontier in artificial intelligence research [research frontier].” IEEE Computational Intelligence Magazine. 2010 Nov;5(4), pp.13-8.
[33] World Malaria Report 2016. http://www.who.int/malaria/publications/world_malaria_repo/en/
[34] Centers for disease control and prevention (CDC). https://www.cdc.gov/malaria/diagnosis-treatment/clini cians1.html
[35] Hommelsheim, C.M., Frantzeskakis, L., Huang, M., Ulker. B. PCR amplification of repetitive DNA: a limitation to genome editing technologies and many other applications. Scientific Reports, 4 (May 2014), 5052.
[36] Hawkes, M., Katsuva, J.P., Masumbuko, C.K. Use and limitations of malaria rapid diagnostic testing by community health workers in war-torn Democratic Republic of Congo. Malaria Journal, 8 (December 2009), 308.
[37] Sio, S.W., Sun, W., Kumar, S., Bin, W.Z., Tan, S.S., Ong, S.H., Kikuchi, H., Oshima, Y. and Tan, K.S. MalariaCount: an image analysis-based program for the accurate determination of parasitemia. Journal of microbiological methods, 68 (February 2007), 11–18.
[38] Poostchi, M., Ersoy, I., Bansal, A., Palaniappan, K., Antani, S., Jaeger, S. and Thoma, G. Image analysis of blood slides for automatic malaria diagnosis. In proceedings of HI-POCT’15 (Bethesda MD, November 2015), MoPoster04.22.
[39] Delahunt, C.B., Mehanian, C., Hu, L., McGuire, S.K., Champlin, C.R., Horning, M.P., Wilson, B.K. and Thompson, C.M. Automated Microscopy and Machine Learning for Expert-Level Malaria Field Diagnosis. In Proceedings of GHTC’15 (Seattle WA, October 2015), IEEE Press, 393–399.
[40] Díaz, G., González, F.A. and Romero, E. A semi-automatic method for quantification and classification of erythrocytes infected with malaria parasites in microscopic images. Journal of Biomedical Informatics, 42 (April 2009), 296–307.

[41] Y. LeCun, Y. Bengio, G. Hinton, “Deep learning.” Nature. 2015 May 28;521(7553), pp. 436-44.

[42] A. Waibel, T. Hanazawa, G.E. Hinton, K. Shikano and K. Lang, “Phoneme recognition using time-delay neural networks.” IEEE Trans. Speech Signal Process,1989 37, pp. 328-339.

[43] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition.” Proceedings of the IEEE. 1998 Nov;86(11), pp. 2278-324.

[44] P.Y. Simard, D. Steinkraus and J.C. Platt, “Best practices for convolutional neural networks applied to visual document analysis.” InICDAR 2003 Aug 3, Vol. 3, pp. 958-962.

[45] R. Vaillant, C. Monrocq and Y. LeCun, “Original approach for the localisation of objects in images.” IEEE Proceedings-Vision, Image and Signal Processing. 1994 Aug, 141(4), pp. 245-50.

[46] A. Krizhevsky, I. Sutskever and G.E. Hinton. “Imagenet classification with deep convolutional neural networks.” Advances in neural information processing systems. 2012, pp. 1097-1105.

[47] M.D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks.” European Conference on Computer Vision. 2014, Sep 6, pp. 818-833.

[48] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going deeper with convolutions.” IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1-9.

[49] K. He, X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition.” arXiv preprint, arXiv: 1512.03385, 2015 Dec.

[50] Chayadevi M, Raju G (2014) Usage of art for automatic malaria parasite identification based on fractal features. Int J Video Image Proc Netw Sec 14(4):7–15

[51] K. Jarrett, K. Kavukcuoglu, M.A. Ranzato and Y. LeCun, “What is the best multi-stage architecture for object recognition?” In International Conference on Computer Vision, IEEE, 2009, pp. 2146-2153.

[52] Christodoulidis S, Anthimopoulos M, Ebner L, Christe A, Mougiakakou S (2017) Multisource transfer learning with convolutional neural networks for lung pattern analysis. IEEE J Biomed Health Inform 21(1): 76–84

[53] Damahe LB, Krishna R, Janwe N (2011) Segmentation based approach to detect parasites and RBCs in blood cell images. Int J Comput Sci Appl 4(2):71–81

[54] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, pages 1097–1105, 2012.

[55] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In ECCV, 2016.

[56] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015.

[57] J. Frankle and M. Carbin. The lottery ticket hypothesis: Training pruned neural networks. ArXiv preprint arXiv:1803.03635, 2018.

[58] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Un- derstanding deep learning requires rethinking generalization. ICLR, 2016.

[59] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. ICCV, 2017.

[60] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. NISP: Pruning networks using neuron importance score propagation. CVPR, 2017.

[61] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In CVPR, pages 2554–2564, 2016.

[62] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. ICLR, 2017.

[63] J. Ye, X. Lu, Z. Lin, and J. Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. ICLR, 2018.

[64] Y. He and S. Han. Adc: Automated deep compression and acceleration with reinforcement learning. arXiv preprint arXiv:1802.03494, 2018.

[65] Y. Chauvin. A back-propagation algorithm with optimal use of hidden units. In NIPS, 1989.

[66] Y. LeCun, J. S. Denker, S. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In NIPS, 1990.

[67] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. ICLR, 2017.

[68] LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521: 436–444

[69] Russakovsky O, Deng J, Su H et al (2015) ImageNet Large Scale Visual Recognition Challenge. Int J Comput Vis 115:211–252

[70] Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classifica- tion with deep convolutional neural networks. Adv Neural Inf Process Syst 25. Available online at: https://papers.nips.cc/pape 4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. Accessed 22 Jan 2018.

[71] Gulshan V, Peng L, Coram M et al (2016) Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. JAMA 316:2402 2410

[72] Esteva A, Kuprel B, Novoa RA et al (2017) Dermatologist-level classification of skin cancer with deep neural networks. Nature 542: 115–118

[73] Ehteshami Bejnordi B, Veta M, Johannes van Diest P et al (2017) Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer. JAMA 318: 2199–2210

[74] Lakhani P, Sundaram B (2017) Deep learning at chest radiography: automated classification of pulmonary tuberculosis by using convolutional neural networks. Radiology 284:574–582

[75] Yasaka K, Akai H, Abe O, Kiryu S (2018) Deep learning with convolutional neural network for differentiation of liver masses at dynamic contrast-enhanced CT: a preliminary study. Radiology 286:887–896

[76] Christ PF, Elshaer MEA, Ettlinger F et al (2016) Automatic liver and lesion segmentation in CT using cascaded fully convolutional neural networks and 3D conditional random fields. In: Ourselin S, Joskowicz L, Sabuncu M, Unal G, Wells W (eds) Proceedings of Medical image computing and computer-assisted intervention –MICCAI 2016. https://doi.org/10.1007/978-3-319-46723-8_48

[77] Kim KH, Choi SH, Park SH (2018) Improving arterial spin labeling by using deep learning. Radiology 287:658–666. https://doi.org/10.1148/radiol.2017171154

[78] Liu F, Jang H, Kijowski R, Bradshaw T, McMillan AB (2018) Deep learning MR imaging-based attenuation correction for PET/MR imaging. Radiology 286:676–684

[79] Chen MC, Ball RL, Yang L et al (2018) Deep learning to classify radiology free-text reports. Radiology 286:845–852

[80] Hubel DH, Wiesel TN (1968) Receptive fields and functional architecture of monkey striate cortex. J Physiol 195:215–243

[81] Fukushima K (1980) Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol Cybern 36:193–202

[82] Aerts HJ, Velazquez ER, Leijenaar RT et al (2014) Decoding tumour phenotype by noninvasive imaging using a quantitative radiomics approach. Nat Commun 5:4006

[83] Lambin P, Rios-Velazquez E, Leijenaar R et al (2012) Radiomics: extracting more information from medical images using advanced feature analysis. Eur J Cancer 48:441–446

[84] Mathworks, Deep learning, 2016, Access on August24,URL:http://www.mathworks.com/discovery/deep-learning.html

[85] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, 2012, pp. 1097–1105.

[86] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” CoRR, vol. abs/1409.1556v6, 2014.

[87] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” CoRR, vol. abs/1512.03385v1, 2015.
[88] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” CoRR, vol. abs/1409.4842, 2014.
[89] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” CoRR, vol. Abs/1704.04861, 2017.
[90] A. Delmas, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, and A. Moshovos, “Bit-tactical: Exploiting ineffectual computations in convolutional neural networks: Which, why, and how,” CoRR, vol. Abs/1803.03688, 2018.
[91] G. Huang, Z. Liu, and K. Q. Weinberger, “Densely connected convolutional networks,” CoRR, vol. abs/1608.06993, 2016.
[92] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, “Dual path networks,” CoRR, vol. abs/1707.01629, 2017.
[93] X. Yang, J. Pu, B. B. Rister, N. Bhagdikar, S. Richardson, S. Kvatinsky, J. Ragan-Kelley, A. Pedram, and M. Horowitz, “A systematic approach to blocking convolutional neural networks,” CoRR, vol. Abs/1606.04209, 2016.
[94] C. Szegedy, A. Toshev, and D. Erhan, “Deep neural networks for object detection,” in Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, 2013, pp. 2553–2561.
[95] Lecun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11):2278 - 2324.
[96] Dan C. Ciresan, Ueli Meier, Jonathan Masci, et al. Flexible, High Performance Convolutional Neural Networks for Image Classification[J]. PROCEEDINGS OF THE TWENTY-SECOND IN TERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2011.
[97] Hinton A K I S G. ImageNet classification with deep convolutional neural networks,” NIPS[J]. Advances in Neural Information Processing Systems, 2012:2012.
[98] Yadan O, Adams K, Taigman Y, et al. Multi-GPU Training of ConvNets[J]. Eprint Arxiv, 2013.
[99] Lei Y, Ferrer L, Lawson A, et al. Application of convolutional neural networks to language identification in noisy conditions[C]//Proc. Speaker Odyssey Workshop (submitted). 2014.
[100] Sun Y, Wang X, Tang X. Deep Convolutional Network Cascade for Facial Point Detection[C]// Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. IEEE, 2013:3476-3483.

[101] Prasad K, Winter J, Bhat UM, Acharya RV, Prabhu GK (2012) Image analysis approach for development of a decision support system for detection of malaria parasites in thin blood smear images. J Digit Imaging 25(4):542–549
[102] Goodfellow I J, Bulatov Y, Ibarz J, et al. Multi-digit number recognition from street view imagery using deep convolutional neural networks[J]. arXiv preprint arXiv:1312.6082, 2013.
[103] Abdel-Hamid O, Mohamed A R, Jiang H, et al. Applying Convolutional Neural Networks concepts to hybrid NN-HMM model for speech recognition[C]// Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on. IEEE, 2012:4277 - 4280.
[104] Boureau Y L, Ponce J, Lecun Y. A Theoretical Analysis of Feature Pooling in Visual Recognition[J]. International Conference on Machine Learning Haifa Israel, 2010:111-118.
[105] Lecun Y, Kavukcuoglu K, Farabet C. Convolutional networks and applications in vision[C]/ Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on. IEEE, 20 10:253 - 256.
[106] Lecun P S. Convolutional Neural Networks Applied to HouseNumbers Digit Classification[C] // Pattern Recognition (ICPR), 2012 21st International Conference on. IEEE, 2012:3288 – 3291.