TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Md Raisul Islam 194212IVEM

Classifying Medical Images on an Edge Device: A Deep Learning Approach Applied to Blood Cells

Master's thesis

Supervisor: Yannick Le Moullec
PhD

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Md Raisul Islam 194212IVEM

Meditsiiniliste kujutiste klassifikatsioon servaseadmel: vererakkudele rakendatood süvõappe lähenemisviis

Magistritöö

Juhendaja: Yannick Le Moullec
PhD

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Md Raisul Islam

03.01.2022

Abstract

In modern times, medical imaging is one of the most important components of clinical research and diagnosis. Compared to conventional radio image processing, Deep Learning (DL) becomes one of the most popular and well-known alternatives for medical image processing due to its robust automated processing capability and less human intervention. In this pape, the suitability of DL method for classifying microscopic medical images is investigated, especially four types of white blood cell images, namely: eosinophils, monocytes, lymphocytes and neutrophils, where the inference phase is performed on a single board computer (SBC).

Two different DL models are proposed to solve this classification problem. The first model was developed from scratch by careful selection of hyperparameters, while the second model uses a transfer learning approach where an already trained model (MobileNetV2) is used for training and validation. A Kaggle dataset of 12500 images is used to develop these models.

For the first model, the training and validation accuracies are 99% and 97% respectively, with an overall classification accuracy of 97.77%. The performance of the second model is slightly lower than the first, with training and validation accuracies of 92% and 87%, respectively. The overall classification accuracy of this model is 92%, but the individual accuracies are 86.15%, 97.71%, 92.5% and 92. 5% for eosinophils, monocytes, lymphocytes and neutrophils, respectively.

Finally, the two models were minimized and deployed on an RPI4 SBC and used for classification inference. On this SBC, the overall accuracy is 98.54% for the first model and 91.3% for the second model. The classification times are measured to be 48.2 ms and 142 ms, respectively. Thus, the first model outperforms the second model as expected.

This thesis is written in English language and is 90 pages long and includes 6 chapters, 42 figures and 11 tables.

Annotatsioon

Meditsiiniliste kujutiste klassifitseerimine servaseadmel: vererakkudele rakendatud sügavõpe lähenemisviis

Kaasaegsel ajastul kliiniliste uuringute ja diagnoosimise üheks olulisemaks osaks on meditsiiniline kujutise töötlemine.Võrreldes tavapärase raadiopilditehnoloogiaga on sügavõpe (DL) muutumas üheks silmapaistvamaks ja populaarsemaks alternatiiviks meditsiinilise pilditöötluse vahendiks tänu oma tugevale automatiseeritud töötlusvõimele ja väiksemale inimese sekkumise vajadusele.See lõputöö uurib sügavõpe metoodika sobivust mikroskoopiliste meditsiiniliste kujutiste klassifitseerimiseks nelja tüüpi valgete vereliblede kujutiste näitel, nimelt: eosinofiilid, monotsüüdid, lümfotsüüdid ja neutrofiilid, millest järeldusfaas teostatakse monoplaatarvutil (SBC).

Selle klassifitseerimisprobleemi lahendamiseks pakutakse välja kaks erinevat sügavõpe mudelit (or DL-mudelit). Esimene mudel töötatakse välja nullist, valides hoolikalt hüperparameetreid, aga teine mudel kasutab ülekande õppimise lähenemisviisi, mis kasutab õppimiseks ja kinnitamiseks eelõpetatud mudelit (MobileNetV2). Nende mudelite väljatöötamiseks kasutatakse 12500 kujutisega Kaggle'i andmekogumit.

Esimese mudeli puhul on õpe ja valideerimise täpsus vastavalt 99% ja 97%, üldise klassifitseerimise täpsusega 97,77%.Võrreldes esimesega on teise mudeli jõudlus veidi madalam, selle õpe ja valideerimistäpsus on vastavalt 92% ja 87% ning mudeli üldine klassifitseerimise täpsus on 92%. Kusjuures individuaalsed täpsused on eosinofiilide, monotsüütide, lümfotsüütide ja neutrofiilide puhul vastavalt 86,15%, 97,71%, 92,5% ja 92,5%.

Lõpuks on need kaks mudelit teisendatud ja kasutusele võetud RPI4 monoplaatarvutil (or SBC) klassifitseerimise järelduste tegemiseks. Tulemusena on esimese mudeli üldine täpsus 98,54% ja teise mudeli puhul 91,3%. Klassifitseerimisajad on mõõdetud vastavalt 48,2 ms ja 142 ms. Seega ületab esimene mudel ootuspäraselt teist mudelit.

See lõputöö on kirjutatud inglise keeles ja on 90 lehekülge pikk, sisaldab 6 peatükki, 42 joonist ja 11 tabelit.

List of abbreviations and terms

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CT	Computed Tomography
DL	Deep Learning
DSC	Depth-wise Separable Convolution
ELU	Exponential Linear Unit
GPU	Graphical Processing Unit
KLD	Kullback-Leibler Divergence
MRI	Magnetic Resonance Imaging
PSC	Point-wise Separable Convolution
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RPI	Raspberry Pi
SBC	Single Board Computer
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TL	Transfer Learning
TPU	Tensor Processing Unit
US	Ultra Sound
WBC	White Blood Cell

Table of contents

List of figures

List of tables

1 Introduction

In the modern history of clinical research and diagnosis, medical image analysis plays a very important role in detecting abnormalities in the human body [1]. Based on the different parts of the body, medical images can be divided into different segments. Figure 1 shows some of the areas where medical images are used as an important tool for diagnostic purposes.

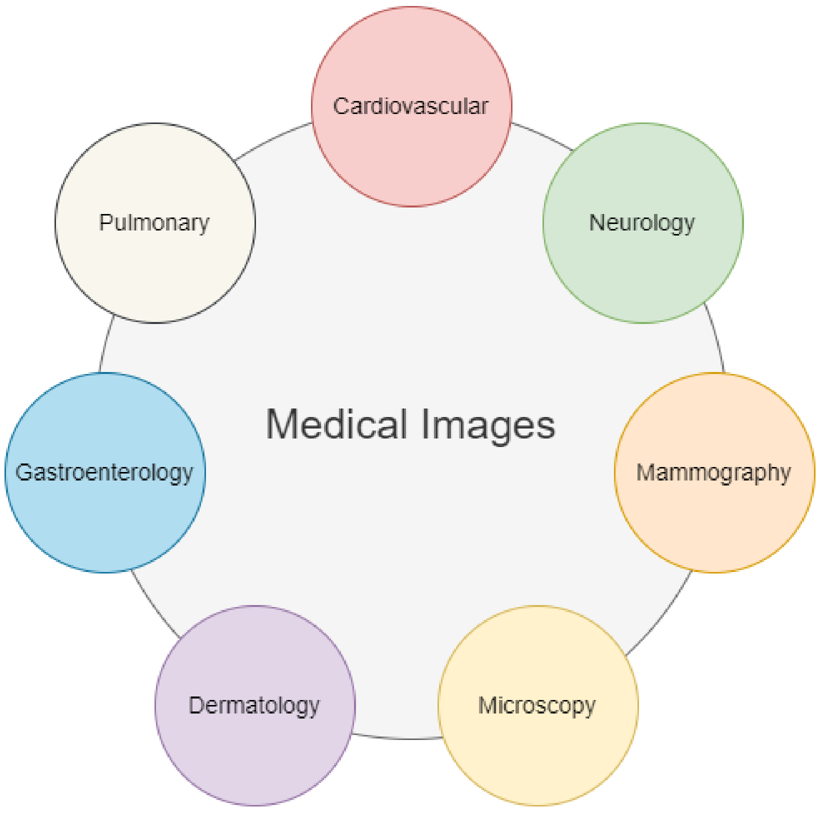


Figure 1: Different sectors of medical image for diagnostic purpose

Traditionally, medical imaging mostly uses various kinds of radio imaging technologies such as X-ray, ultrasound (US), computed tomography (CT), and magnetic resonance imaging (MRI), in which the scanned image of the abnormal area has been gathered and abnormality has been detected by a trained physician [2]. This usually requires pattern recognition and detection by human intervention. Thus, disease diagnosis using medical image analysis comes with a price of trained human resources to operate medical imaging equipment and doctor's experience, which are very limited compared to the world population. Specifically, the less-developed countries suffer from a limited number of both trained physicians who can operate medical imaging equipment and experienced doctors. This, among other reasons such as the need for improved efficiency in general, demands scientific exploration to introduce automation in medical image analysis for disease detection and diagnosis.

Developments in Deep Learning (DL), especially in the field of neural network (NN), have made them a popular choice for image processing for classification and object detection [3]. The history of using NN for image classification dates back to 1998 when Lecun et *al.* proposed a model called 'LeNet-5' [4] for "document recognition". However, the major breakthrough for using NN as an image classifier came after the development of VggNet [5] in 2014. In this architecture, the large kernel size was replaced by convolutional series and was eventually called convolutional neural network (CNN). This successfully reduced the number of parameters, resulting in shorter classification times. Later, AlexNet [6] adopted rectified linear unit (ReLU) as an activation function. The introduction of ReLU and dropout into the architecture successfully solved the overfitting problem, which ultimately improved the classification accuracy. In recent days, there are a number of CNN architectures such as ResNet [7], GoogleNet [8] and MobileNet [9] which solve various DL-based classification problems and improve the accuracy, decrease processing time, and enhance compatibility with resource-constrained devices, etc. This makes CNN the heart of image classification and object recognition techniques. This inspired scientists to apply the DL method for medical image analysis due to its tremendous effectiveness in image classification and object detection [10]. Over the last decade, there is an exceptional surge of scientific reporting of medical image analysis using ML/DL method. Figure 2 shows the number of reported articles in the scientific literature that applied Machine Learning (ML) / DL techniques to investigate medical image.

However, the review of the literature has shown that DL-based techniques for image classification are usually resource hungry, which require high computational power devices. So, again this state-of-the-art technology is stuck with the same old reachability problem for less-developed parts of the world. A possible solution to this problem is to use single board computer (SBC). SBCs have emerged as potent computational platforms for executing algorithms that insofar were restricted to desktop or large embedded platforms. These SBCs are thus attractive for the implementation of ML inference to develop portable and cost-effective image classifiers. While some works have recently used SBCs for DL-based analysis of medical images, there is room for further exploration of, and experimentation with, such relatively new technologies. Considering the fact, this thesis will evaluate the suitability of SBC boards to run DL for medical image classification at the microscopic scale; the specific use-case is the classification of white blood cells into four types (Eosinophil, Lymphocyte, Monocyte, and Neutrophil).

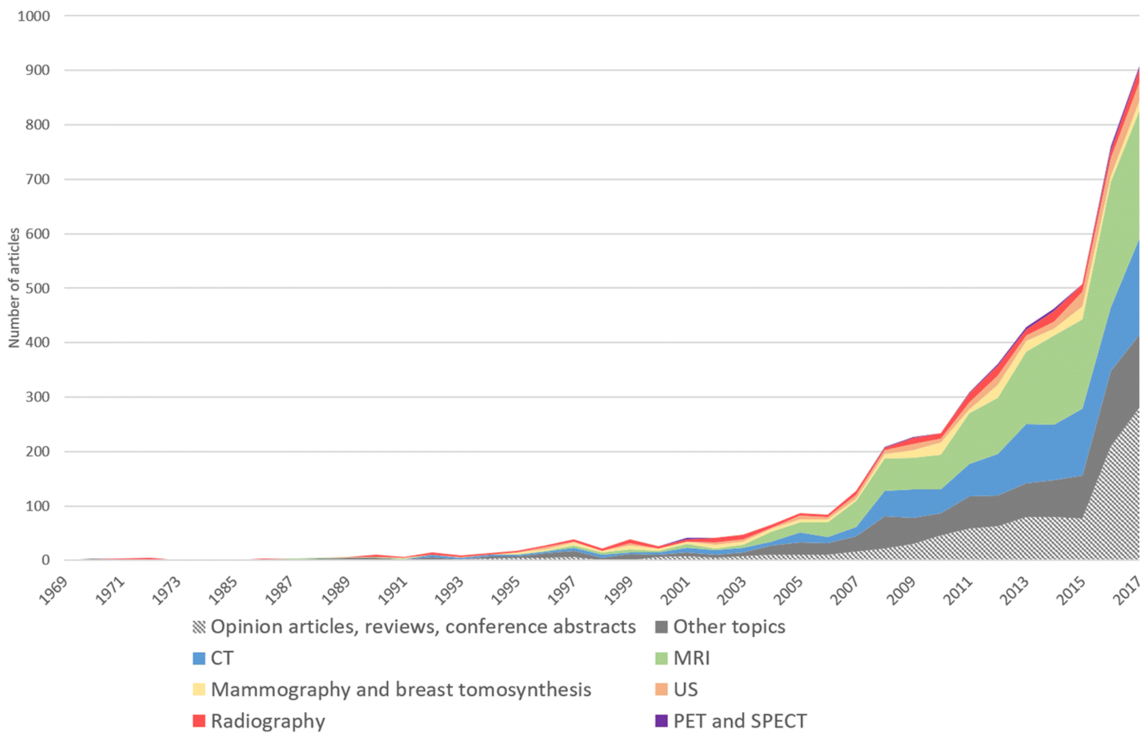


Figure 2: Number of reported articles in the scientific literature for various medical image analysis techniques using ML/DL techniques [11]

1.1 Research Statement

Since medical images are specific to the body parts and disease types, it is necessary that particular DL models are developed for each case. Therefore, one of the goal of this thesis is to develop a DL model from scratch that can successfully classify (white) blood cell images and evaluate its performance.

However, the fundamental challenge of applying DL techniques for classification of medical images is data scarcity. The DL is a layered architecture where each layer consists of a number of parameters to learn the pattern of the images. Thus, the higher the number of data that can be supplied, the more accurate the detection will be. Nevertheless, publicly available medical images are rare, which is one of a major obstacles to develop a DL model for image classification. But the development of transfer learning (see Section 3.4) allows to use one pre-trained DL model to solve a similar new problem. This eventually save time and effort to develop a new model from scratch. Thus, the second goal of this thesis work is to investigate the use of transfer learning for blood cell classification. A pre-trained model (MobileNetV2) is used for training the classification model and its performance will be characterized. Subsequently, it is implemented onto an SBC for portable image classification inference.

Finally, the performance of SBC is evaluated in terms of accuracy and classification time for both models (i.e. one developed from scratch and the other based on transfer learning). This comparison will help us to understand the effectiveness of transfer learning process over the conventional DL design flow for image classification.

Given the above, the research statement of this MSc thesis is expressed as:

- ❖ **Build a DL model from scratch to classify blood cell images and evaluate its performance**
- ❖ **Apply transfer learning approach for model training with a pre-trained model and evaluate its performance**
- ❖ **Deploy both models onto a selected SBC to classify blood cell images and compare their performance in terms of accuracy and classification time**

1.2 Main Steps Followed in this MSc Thesis

Below is the process followed to conduct this thesis to address the above problem statement:

- ➢ Study the fundamentals of DL strategies for image processing

- Explore a selected subset of suitable DL strategies and algorithms for image processing
- Explore and select a suitable SBC platform for implementation
- Train the models based on a predefined dataset
- Test the models, identify and tune parameters, and evaluate the performance of image classification
- Map the inference models onto the selected SBC and evaluate their performances

1.3 Thesis Organization

This thesis contains introductory knowledge and background on DL and particularly CNN. Among other things, a transfer leaning learning approach is adopted for model training by using a pre-trained model named MobileNetV2, the implementation and deployment details on an embedded device (i.e. SBC) are explored afterwards. This chapter (Chapter 1) provided an introduction to the application of DL for medical image analysis, research statement and the intended aim of the thesis.

Chapter 2 delves further into the state-of-the-art related to DL based image classification for microscopy images and explore the usage of SBC in medical image classification.

Chapter 3 explores a general background of the concept of DL, particularly CNN, also a comparatively new design approach named transfer learning is briefly discussed.

Chapter 4 contains the methodology and implementation process adopted in this thesis with a detailed description of the preparation of a simple CNN model from scratch and by using pre-trained model, hyper parameter selection process for model design, and selection of hardware, framework and software libraries. The results of these processes and their analyses are detailed in Chapter 5.

Finally, the last chapter gives a conclusive discussion about the work and suggest ideas for future work that could be carried out based on the author's recommendations.

2 State of the Art Review

In the last decade, the DL method has become very popular in the scientific community to process medical images as it has a robust image processing capability to identify the underlying patterns of the images [12]. Numerous works reported in the scientific literature have been published using DL methodology to extract clinical information from medical images for disease identification. In a broader sense, the application of the DL methodology in medical image processing can be divided into five distinct sections, namely: registration, localization, classification, recognition and segmentation (see Figure 2).

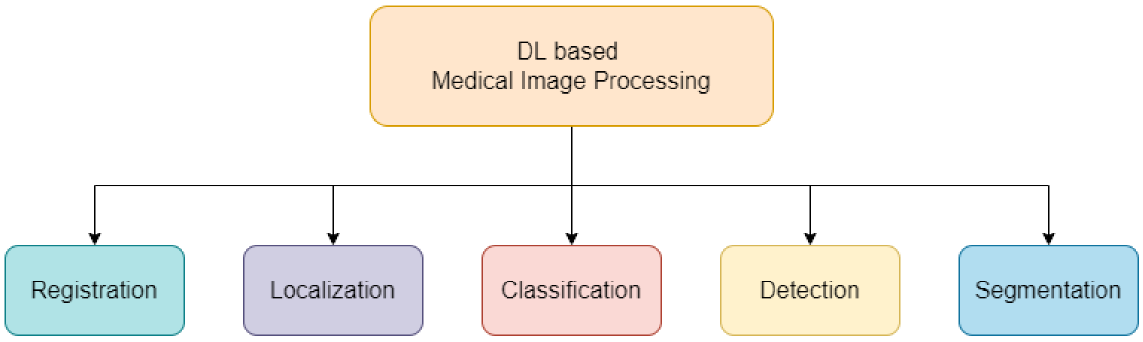


Figure 3: Application of DL methodology for medical image processing [12]

Different scientific groups apply the DL methodology in each of the medical image processing areas mentioned in Figure 1. For example, in the cardiovascular field, DL is mainly used to segment the heart chambers [13]. The goal is to obtain a clear view of the heart chambers. One of the earliest attempts was made by Shelhamer et al. where a fully connected convolutional network was used for segmentation of the left ventricle (LV) [14]. Later, 3D ultrasound images were used for segmentation of LV by Dong et al. to obtain 3D spatial information [15]. Tran et al. used CNN architecture to segmentation of the right ventricle (RV) using short-axis MRI images [16]. Cardiac motion tracking is another area where DL methods have been used to track cardiac muscle activity. Ferdian et al. applied a combination of RNN and CNN to identify the strain associated with myocardial motion [17]. In neuroimaging, DL based tissue classification and tumour detection is becoming popular. Several groups have reported various DL techniques for brain tumour segmentation, brain injury detection, and neural disease prediction. A recent work by Liu et al. proposed a novel DL approach for early detection of Alzheimer's disease. The accuracy of the model is 75.44%, 81.53% and 82.93% for MRI, positron emission tomography (PET) and MRI+PET image dataset respectively [18]. On the other hand, for brain tumour classification, Havaei et al. developed a deep neural model using the BraTS dataset, which effectively reduces the classification time with improved accuracy (85-88%) [19]. Intensive research is also being conducted in lung and mammography to develop DL methods for image processing in the respective fields. While the main focus in pulmonary field is to detect cancer cells [20] [21] [22], scientists in mammography apply DL technique to detect abnormal breast tissue for breast cancer diagnosis [23].

However, in this thesis, we are concerned with blood cell classification, which falls under the category of microscopic imaging. A brief overview of the application of DL for classification of microscopy images is given in the following section. Deep Learning in Microscopy Image Classification

2.1 Deep Learning in Microscopy Image Classification

Following the trend, DL is becoming a popular method for classifying microscopic images. The application mainly includes two areas: cellular & sub-cellular classification and disease diagnosis [24]. In disease diagnosis, the DL method is mostly used to classify different types of histology images to obtain information about various diseases such as leukemia, dengue, and malaria as well as cancer cell detection [24]. Several scientific groups have reported their findings. One of the earlier reports by Chen et al. presents various learning methods such as DNN, Support Vector Machine, Native Bias etc. and how they were successfully used for white blood cell (WBC) detection [25]. The model accuracy was over 85% for all learning methods, which was 17% higher than the conventional size-based method. Qin et al. proposed a deep residual network-based classifier that can recognise leukocytes with 98.15% accuracy [26]. Shahin, et al. developed their own WBCsNet architecture for WBC detection, which achieves 96.1%

accuracy [27]. In the context of anemia diagnosis, Alzubaidi, et al. used a transfer learning technique for red blood cell (RBC) prediction [28]. DL is also used to detect various parasites in histopathology images. Hung et al. applied faster region-based CNN (R-CNN) to identify erythrocytes infected with Plasmodium parasites causing malaria using the ImageNet dataset [29]. The prediction accuracy of this model is 72% for non-difficult infected cells. Recently, Deelder et al. used the DL technique for DNA sequencing of malaria parasites with 90% accuracy [30]. Moreover, there exist many scientific reports on mitosis detection using CNN-based architecture [31] [32] [33] [34] [35] for various cancers (lung, breast, prostate, etc.) in pathology. In the near past, DL based microscopy image processing is becoming popular in the field of imaging flow cytometry (IFC), also known as deep cytometry. The main focus in this field is to apply the DL technique to understand cellular changes [36], detect single cells [37] [38] and count T, B and NK types of white blood cells [39]. Table 1 lists some important works on classification of microscopy images with DL.

Table 1: State of the art of DL based microscopic image classification

Group	***Task***	***DL technique***	***Accuracy***	***Year***
Chien at *al* [40]	Immature WBC detection	CNN + faster R-CNN	90.1%	2021
Alzubaidi, et al. [28]	RBC classification	CNN	99.5%	2020
Lippeveld, et al. [41]	WBC classification	CNN + Traditional Machine learning	77.8%	2020
Khan, et al. [42]	Breast cancer detection	CNN	97.5%	2019
Jha et al. [43]	Acute Lymphocytic Leukaemia detection	CNN	98.7%	2019
Li et al. [44]	Mitosis detection in breast histopathology image	CNN	67.3%	2019

Wang et al. [45]	Lung cancer detection from whole slide image	FCN	97.3%	2019
Qin, et al. [26]	Leukocyte classification	Res-Net	76.8%	2018
Tellez et al. [46]	Mitosis detection in breast region	CNN	90.0%	2018
Niikoa et al. [47]	C2C12 cell classification	CNN	98.0%	2018
Durant et al. [48]	Erythrocyte Classification	CNN	92.7%	2017
Hung et *al* [29]	Malaria detection	Faster R-CNN	72.5%	2017
Zhang et al. [49]	Cervical cell classification	CNN	98.3%	2017

2.2 Classification of Medical Image using Edge Device

Image classification and recognition using DL usually requires devices with high computational power due to the complexity of mathematical computation and memory-hungry learning model with millions of parameters. However, with the development of various relatively lightweight CNN architectures like GoogleNet, CapsNet, AlexNet, MobileNet, YOLO along with different post-training quantization technique and frameworks like 'TensorFlow Lite', the number of parameters in / size of the model is effectively reduced without necessarily compromising the accuracy. Moreover, the introduction of GPUs (Graphical Processing Unit) and TPUs (Tensor Processing Unit) has improved the computational performance of SBCs. While the training phase is still computationally demanding, the above opens up the possibility of using various SBCs such as Raspberry Pi, Coral Dev, Jetson Nano and even smartphones in DL-based application for the inference phase of image classification and object recognition applications. This inspires researchers in the medical field to use SBC for medical image classification for early detection and diagnosis of diseases, taking the advantage of low-cost and portable solution. In recent years, much has been reported in the scientific literature on the use of edge devices for medical image classification for disease diagnosis. Table 2 enlist some of the recent work.

Table 2: DL based medical image analysis by edge device

Group	Task	Edge Device
Negh et al. [50]	Detection of skin cancer. MobileNetV2 is used for detection and U-Net is used for segmentation	Raspberry Pi 3B+
Lavanya et al. [51]	Classification and detection of Diabetic Retinopathy. CNN model has been developed. Kaggle data set is used	Raspberry Pi 3
Abid et al. [52]	Developed ML model to classify of chest X-Ray image for various lung disease.	NVIDA Jetson Nano, Raspberry Pi 3B+, Google Pixel, and Samsung Galaxy S10+
Krömer et al. [53]	Covid-19 detection using chest X-ray images.	NVIDA Jetson Nano
Javier et al. [54]	Eye fundus image segmentation for glaucoma detection	Raspberry Pi 4B, Coral Dev board

3 Deep Learning: An overview

ML, essentially described as a subfield of artificial intelligence (AI), is one of the most revolutionary areas of computer science in recent decades. As shown in Figure 4, DL belongs to a subclass of NN (itself a subclass of ML), the most important class of machine learning in AI taxonomy [55].

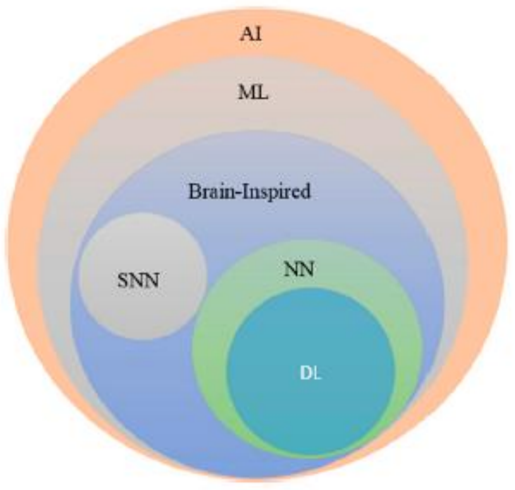


Figure 4: The taxonomy of AI. AI: Artificial Intelligence, ML: Machine Learning, NN: Neural Network, DL: Deep Learning, SNN: Spiking Neural Network [55]

Traditionally, ML provides a computational facility to solve a problem through learning. In general, various mathematical models have been developed to train a system to produce useful results based on input data. The result obtained during training is known as knowledge or experience. Based on the training data, the system predicts the output using various optimization algorithms.

Unlike the traditional ML, DL is not a specific method but a set of different techniques that usually involve multiple layers of nonlinear information processing units. These multilayer nonlinear information processing units are called artificial neurons. Different network architectures of these neurons are collectively called Deep Neural Network (DNN) which form the backbone of the DL method for learning and prediction.

3.1 Types of DL Approach

Based on the state of input data, the DL approaches are categorised into three basic categories which are shown in Figure 5 and discussed in what follows.

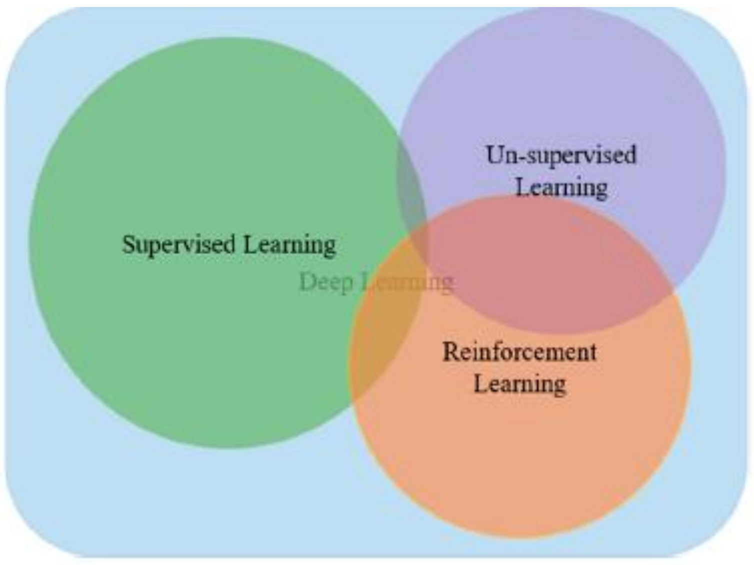


Figure 5: Types of DL Approach [55]

3.1.1 Deep Supervised Learning

In deep supervised learning, the input data is labelled in advance. In the training phase, a direct relationship between the input and the output is explicitly specified so that the model can learn over time. Later, when a new input is fed into the model, it predicts the accuracy of the output based on the loss function and adjusts it by minimising the error as long as it does not reach the minimum value. CNN is one of popular architecture of this class; CNN has been adopted to develop the model in this work.

3.1.2 Deep Unsupervised Learning

In deep unsupervised learning, the training datasets are not labelled. So, there is no direct relationship between input and output, but the system analyses the input, decomposes it and recognises the common function for all input data [56]. Some of the widely used techniques are Auto-Encoders (AE), Restricted Boltzmann Machines (RBM), and the recently developed GAN [57].

3.1.3 Deep Reinforcement Learning

Trial-and-error is the main idea behind Reinforcement Learning (RL). The training phase is designed based on a reward policy. The rewards are divided into positive and negative observations. When the model interacts with the target system, it receives these positive or negative feedbacks (rewards). While the positive reward 'reinforces' into the model and is considered as 'knowledge', the negative feedback is usually blocked [58]. This process is carried out until enough information about the target environment has been gathered to operate on it. Fig. 3 shows the principles of the RL strategy.

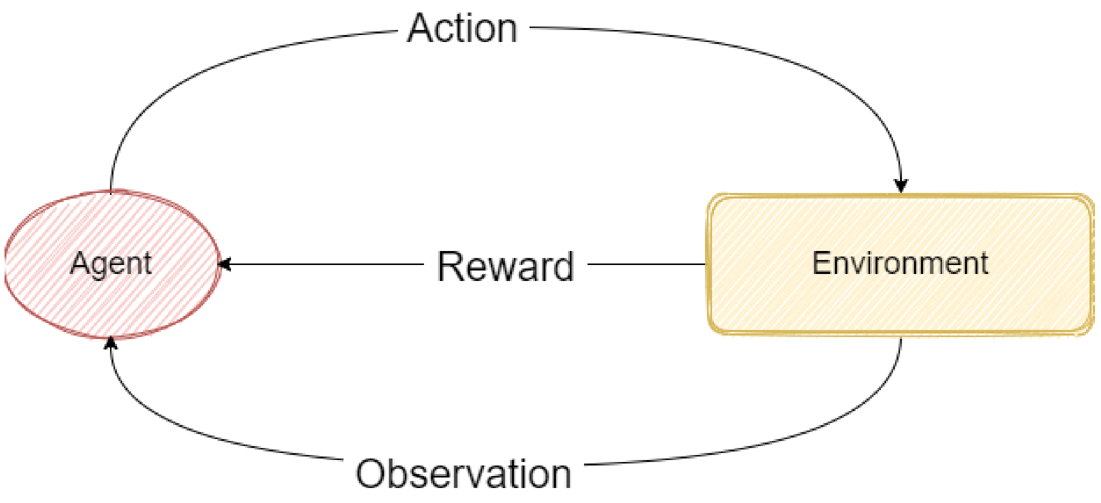


Figure 6: Reinforcement learning strategy [59]

Deep RL (DRL) combines RL and deep learning (for example NNs). DRL can work with unstructured input data and agents can perform decision-making about such data without manually specify the state space. As a result, DRL can work with very large inputs, making suitable e.g. for images.

3.2 Deep Neural Network (DNN)

A DNN is a subset of an ANN that consists of a series of interconnected computational units. These units are often referred to as nodes or neurons. As these neurons are organized into layers, a DNN is also known as multi-layered perceptron (MLP). In DNN, the input data passes through of a series of hidden layers where it transforms through various mathematical algorithms to recognize the underlying pattern of the input data. This process of pattern recognition is called training. After pattern recognition, the outputs are compared by an objective function. Throughout the training process, the parameters of the network are tweaked consistently to discover the underlying pattern. This network can be used to make predictions for new unseen data, once the pattern is identified. Figure 7 shows the general DNN architecture.

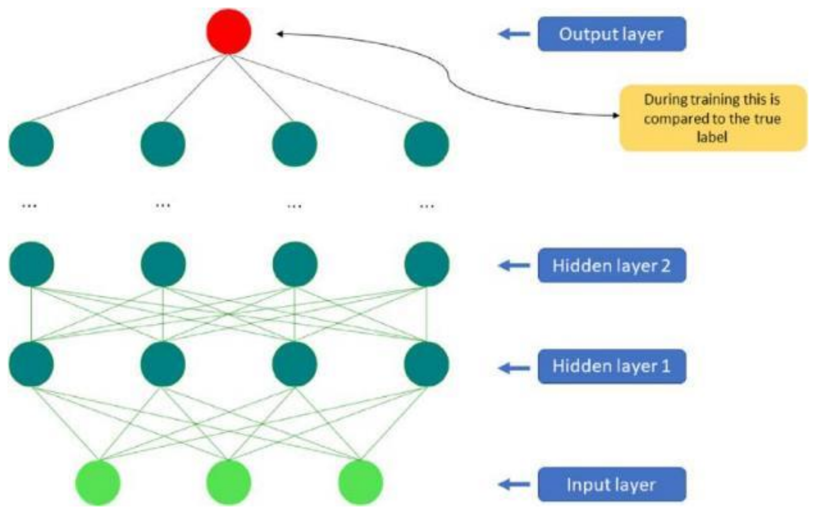


Figure 7: Illustration of a DNN architecture [60]

3.2.1 DNN Training: Approaches & Parameters

Some of the important procedures and parameters related to DNN-based training are briefly described in following subsections.

3.2.2 Gradient Descent (GD)

The gradient descent is the first-order optimization approach [61]. This process detects the local minima of the objective function for which reason it is widely used in DNN training. The detail of this algorithm is mentioned in reference [62].

3.2.3 Stochastic Gradient Descent (SGD)

Unlike GD, SGD determines the gradient of an objective function or cost function for a single batch for each iteration. This significantly reduces the training time. However, the gradient result is only an estimate for updating the weights, which are later fitted to the model by multiplying by a constant called the learning rate. In SGD, the weights are updated after each mini-batch is submitted to the algorithm [63].

3.2.4 Learning Rate (η)

The learning rate is an essential element for DNN training. It determines the step size during training. Therefore, the choice of the value for the learning rate is crucial as the training time is closely related to it. This is because a larger value for η may cause the network to diverge instead of converging, while a smaller value may cause the network to take longer to converge. Moreover, the network may be stuck in its local minima.

Generally, learning rate reduction policies are: constant, factored, and exponential decay. First, the learning rate is minimized with a defined step function. Then, the learning rate can be tuned with the following Equation (1) [55]:

$$\eta_t = \eta_0 \beta^{\left(\frac{t}{\varepsilon}\right)} \tag{1}$$

where η_t is the tth round learning rate,

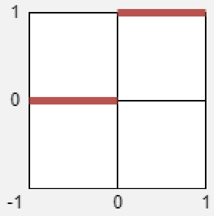
η_0 is the initial learning rate,

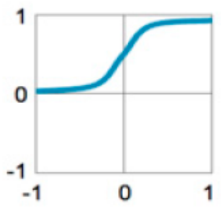
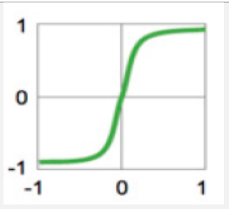
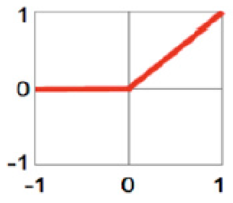
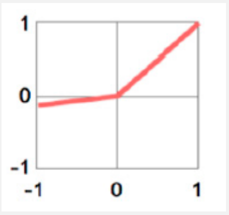
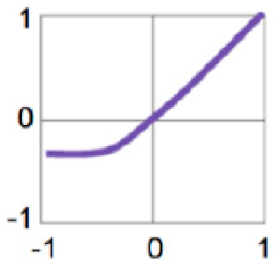
β is the decay factor with a value between the range of (0, 1)

3.2.5 Non-linear Neural Units (Activation Functions)

To overcome computational rigidity, there is an urge to introduce nonlinearity into the network. This is because linear transformation is not always suitable to deal with complex data patterns. Therefore, the linear weighted sum of inputs is paired by a nonlinear function to introduce nonlinearity into the neural unit. This nonlinear function is called the activation function. There are several types of activation function used in DNN training which is mentioned in Table 3.

Table 3: List of Non-linear Functions (activation functions)

Name	Equation	Graphical Characteristics
Binary threshold	$f(x) = \begin{cases} 1, & x > 0 \\ 0, & otherwise \end{cases}$	

Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	
Tanh	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	
Rectified Linear Unit (ReLu)	$f(x) = \begin{cases} x, & x > 0 \\ 0, & otherwise \text{ or } \max\{0,x\} \end{cases}$	
Leaky ReLU	$f(x) = \max(ax, x)$	
Exponential Linear Unit (ELU)	$f(x) = \begin{cases} x, & x \geq 0 \\ a(e^{x}-1), & x < 0\} \end{cases}$	

Softmax Activation Function: In classification problem of a DNN with k classes (k > 2), the conditional probability distribution $P(x|y)$ is needed to determine where in the output layer k neurons should be located whose sum of all weight values should be 1. To equip the network with the knowledge that the output of all k units should sum to 1, the activation function Softmax is used. This is a generalization of sigmoidal activation. The Softmax function compresses the output of each unit so that it is between 0 and 1, just as a sigmoidal function would. The Softmax function is defined by Equation (2) [21]:

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}} \quad (2)$$

where, z is a vector of the input to the output layer

j indexes the output units, *j* = 1, 2, ..., *K*

3.2.6 Loss Function

The loss function provides a comparison between the output of a neural network and the target values in the training. It produces a loss value/score to measure the accuracy of the network's predictions with the expected output. Below are a few popular loss functions [61] [62]:

- **Binary cross-entropy**: "*supply the log loss or cross-entropy loss for the two-class classification problem*".
- **Categorical cross-entropy**: use the generalized cross-entropy for class-size>2
- **Mean Squared Error**: calculate the mean squared sum error. This is often used for various regression problems.
- **Mean Absolute Error (MAE)**: Errors are calculated by squaring their values.
- **Mean Absolute Percentage Error**: Measures the magnitude of the error as a percentage.
- **Kullback-Leibler (KL) Divergence**: It is a probabilistic approach. The method calculates the deviation of probability distribution between each step.

3.2.7 Adaptive Learning Rate

Depending on the requirement, it is sometimes necessary to update the parameters at different rates instead of using a constant learning rate. Therefore, several types of adaptive gradient descent algorithms provide a resort to the classical SGD, by maintaining per-parameter learning rates. This often comes under the optimization process. Below are some of the popular optimization algorithms used in a DNN framework:

- **AdaGrad**: To adjust the learning rate, the algorithm first squares all the gradient values. Then it adds those square values and square roots to the sum. Finally, it inverts the value obtained from the square root operation. In this way, larger moves are made in the gently sloping direction of the error surface. However, using this trick at the very beginning of training can cause some of the learning rates to drop dramatically [61].

- **RMSprop**: It is a modified AdaGrad algorithm. It adopts the Exponentially Weighted Moving Average (EWMA) technique. It has a moving average parameter ρ that controls the length and scale of the moving average [62].

- **Adaptive Moments (Adam):** This algorithm is a combination of two different algorithms to detect the gradient. First, it extracts the best value results from momentum algorithm. Then it combines with the adaptive learning rate algorithm output [61].

3.2.8 Overfitting and Underfitting in DNN

Generally, in any DL model, the training dataset is divided into three subclasses: training, testing, and validation. Typically, the validation error is slightly higher than the training error. However, overfitting occurs if the difference between test and validation rises with iterations. In contrast, if the training error no longer decreases to a sufficiently low value, the issue is described as underfitting.

There are several techniques to overcome this over- and under-fitting problem which is known as regularization of the model. Below are the brief descriptions of such techniques:

- **Weight Sharing:** In this strategy, different layers in the network using the same set of weights results in fewer parameters to regulate. Using shared weights in a few layers helps the model generalize better by controlling model capacity [61].

- **Weight Decay:** L2 regularization approach used in this case. It is a good tool to generalize the network and it also helps to deal with the overfitting problem [61].

- **Dropout:** Dropout is an averaging technique that randomly masks the output of a fraction of nodes from a layer by setting their output to zero during the forward pass. It removes a fraction of nodes from a layer and creates a new neural network with fewer nodes. Typically, 20% nodes are dropped out at the input layers where in hidden layers, and up to 50% fractions of nodes can be dropped [55].

- **Batch Normalization:** In this strategy, the activation layers are normalized by subtracting the mean and dividing it by the standard deviation for each training batch. This process periodically changes the value of activation layer. For every batch it reduces the mean to zero mean and standard deviation to one which results in an increase in the training speed and reduces the dependency on parameter initialization [55].

3.3 Convolutional Neural Network (CNN)

The Convolutional Neural Network (CNN) is the most popular DNN architecture for pattern recognition with a high degree of invariance to translation, scaling, and rotation in two-dimensional image data. It is a multilayer hierarchical neural network first proposed by Fukushima in 1988 [64]. Compared to other ANN architectures, the CNN has a clear advantage due to its more human-like visual processing, optimized structure, and effective learning and extraction of 2D features. In addition, absorbing gradient-based learning algorithm results in minimal error introduction into the network and allows the CNN to produce highly optimized weights during the training phase. The input to a CNN is arranged in a lattice-like structure. This preserves the spatial relationships [65] between the layers. Moreover, it operates on a small region of the previous layer. In a CNN process, the input passes through a series of convolutional layers and activation layers where the weights of the data are determined. Later, pooling layers determine the weights required for feature extraction. Then backpropagation and gradient descent algorithm are used to train the system based on the weights extracted from the previous layer. Finally, a fully concatenated layer is used to determine the output.

3.3.1 Building blocks of CNN

Figure 8 shows the overall architecture of a CNN which consists of three major parts: convolutional layers, pooling/sub-sampling layer, and fully connected/classification layer.

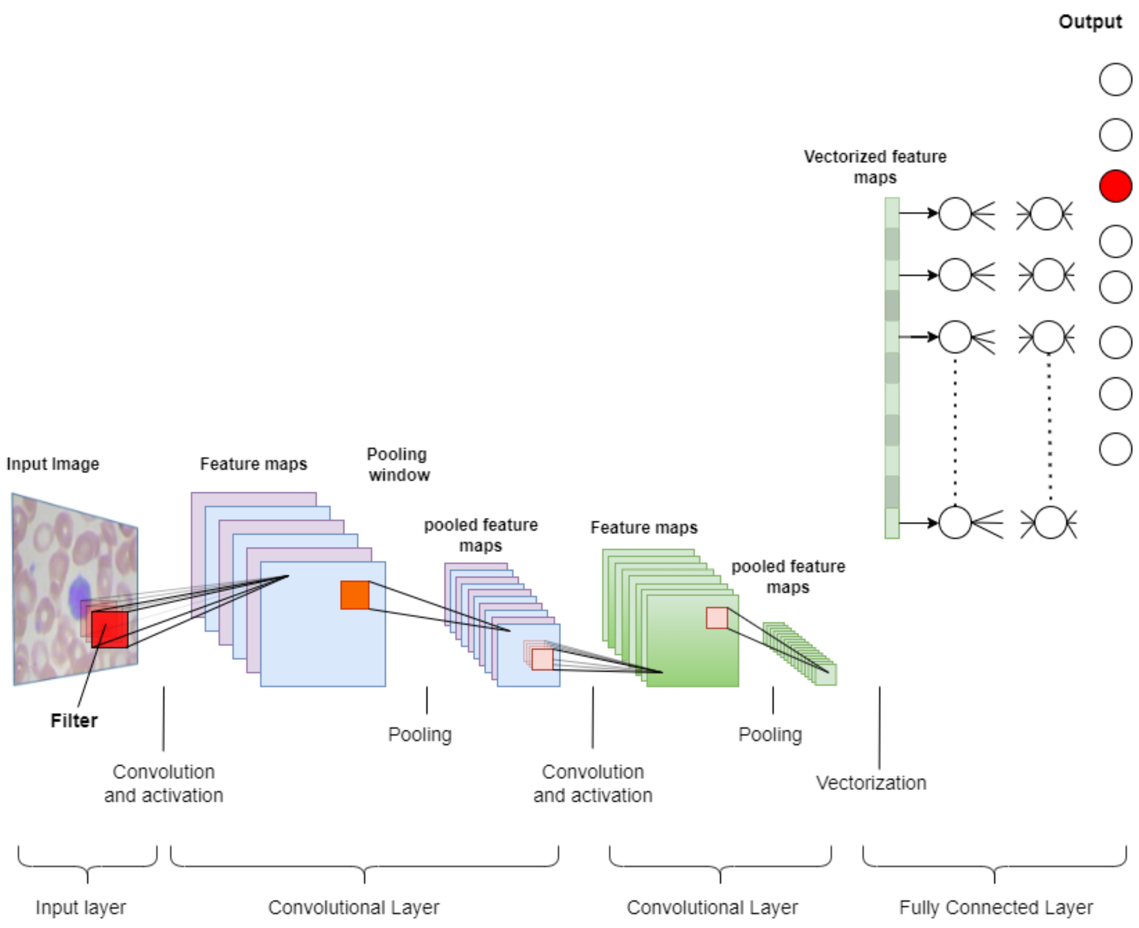


Figure 8: Building blocks of a typical CNN. A slight modification of a figure in [60], courtesy of the author.

(i) **Convolutional layers:** This is the first layer of a CNN architecture. In this layer, convolution between previous layers is performed with different learnable kernels [55]. The output thereof then passes through various activation functions mentioned in Section 5.2.1.4. That generates the output features. Later, each of these output features is compared with input features to discover the cohesion between the input and the output, which is known as weight-sharing [55]. Equation 3 describes the operation of the convolution layer:

$$x_j^l = f\left(\sum_{i\epsilon M_j} x_i^{l-1} * k_{ij}^l + b_j^l\right) \tag{3}$$

where x_j^l is the output of the current layer,

x_j^{l-1} is the previous layer output,

k_{ij}^l is the kernel for the present layer,

b_j^l is the biases for the current layer,

M_j is the selection of input maps.

A bias b has been introduced in this layer to direct the operation. Finally, the output go through a linear or non-linear activation function.

(ii) **Sub-sampling Layer**: The subsampling plane performs the downsampling operation on the input maps. This layer is commonly known as the pooling layer. In this layer, the number of input and output feature maps remains unchanged. The downsampling operation effectively reduces the dimension of the output maps based on the size of the mask [54] [55]. This process can be formulated as per Equation (4):

$$x_j^l = down\left(x_i^{l-1}\right) \tag{4}$$

where down(.) stands for subsampling function. Mostly, two types of pooling strategies are used for dimension reduction. The first one is max-pooling, where the highest value of the segmented grid is used. The second one is average pooling, where the average value from the segmented part is considered. Figure 9 shows the pooling operation in detail.

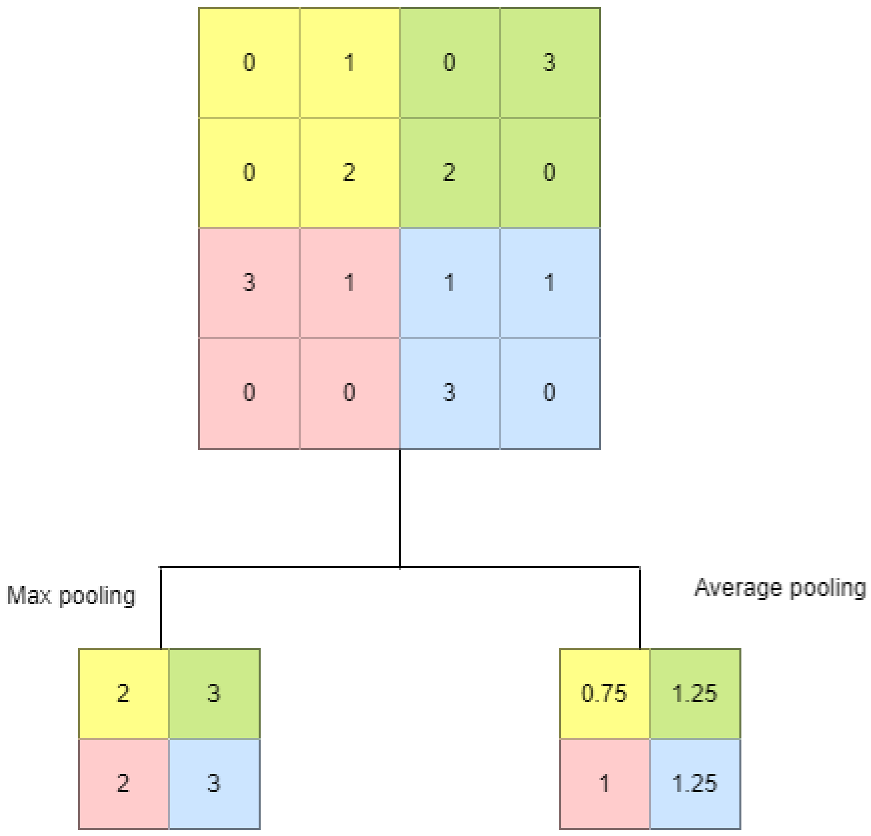


Figure 9: Process of Max pooling and Average pooling.

(iii) **Classification Layer:** The classification layer is the last layer of the CNN architecture. It receives the inputs from the last convolutional layer where all the features of the inputs are extracted and the feature is determined. The classification layer then assigns all these features to the individual neurons and evaluate the score of each class based on this extraction. Afterwards, the classification is done using the Softmax algorithm based on the probability points obtained from the operation [60]. Therefore, the output of a CNN is often given in terms of probabilities. For example, if a model is developed to recognise a car in an image, the probability of a successful prediction could be e.g. 95% or more. If the model contains multiple classes, the output will also contain a small probability value (e.g. 1-5%) for the remaining classes.

3.4 Transfer Learning

In recent years, tremendous research and development in the ML field has resulted in the ML model being more robust, lightweight, and efficient in pattern analysis. This has led to almost every industry using the ML method, from the medical industry to social media to finance, to identify the underlying pattern of the user class. This leads to a problem with the classic ML method, as the training is strictly task-specific and a separate model must be created for each problem. Also, there is a possibility that there is not enough data to train a new model. Transfer learning (TL) provides a solution to this problem, as the main idea behind it is to use the experience (trained parameters) of a previously trained model and apply this experience to solve a similar new task [66]. Figure 10 shows the concept of transfer learning in contrast to the conventional ML method

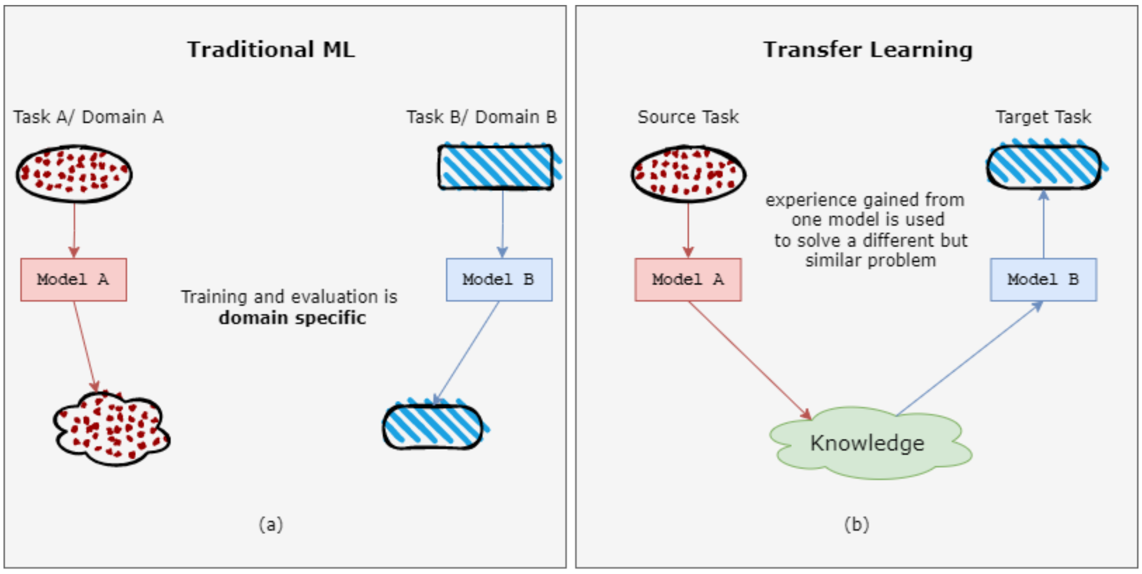


Figure 10: Comparison between traditional ML and Transfer learning methodology. This image has been regenerated using [67]

Normally, the transfer learning technique facilitates the learning mechanism by enhancing the baseline performance, effectively reducing the model creation time and achieving better final output. Figure 11 summarizes the key advantages of transfer learning.

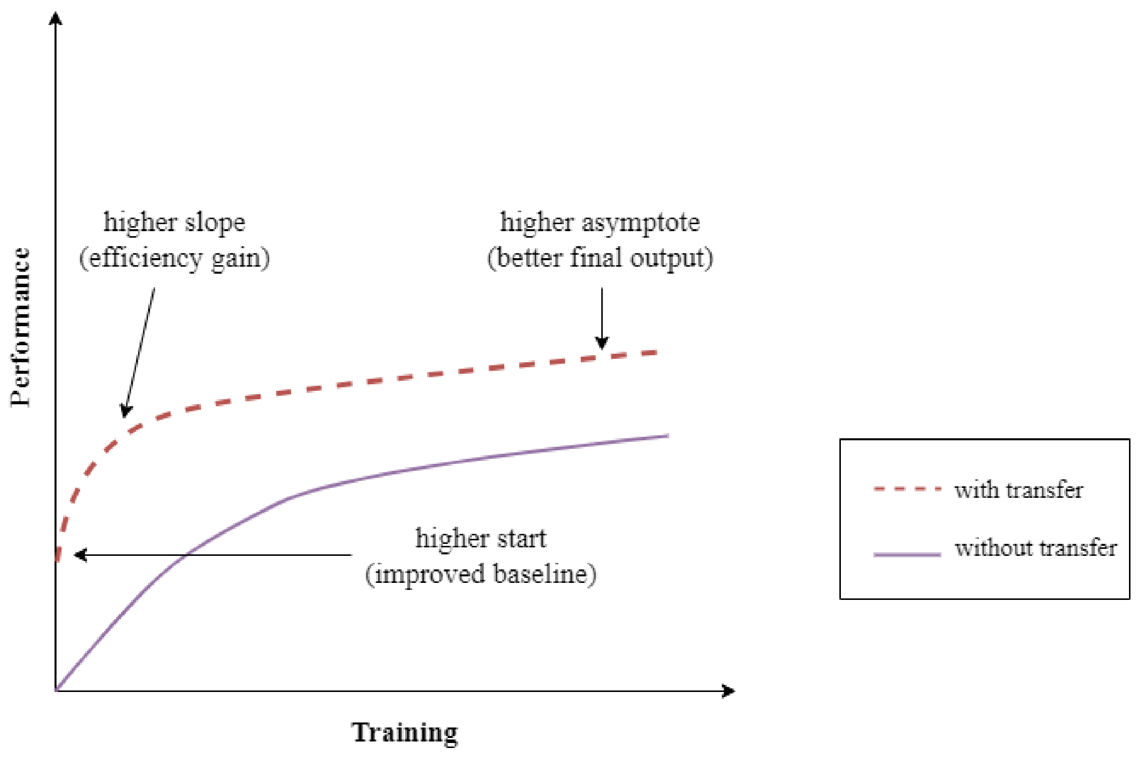


Figure 11: Probable advantages of transfer learning [68]

3.4.1 Transfer Learning Approaches

Transfer learning is often considered a design approach rather than a DL technique. These approaches can be broadly divided into two classes: domain-based and feature specs based. Pan and Yang (2010) [69] describe in detail the domain-based approach, dividing the TL method into four classes: Instance Transfer, Feature Representation Transfer, Parameter Transfer, and Relational Knowledge Transfer. In addition, Weiss et al. in their study provided a detailed description of the feature-spaces based methodology, dividing the methodology broadly divided into two main categories, homogeneous and heterogeneous [70]. In a more recent study, Nam et al. studied the feature-based approach extensively and classify the heterogeneous approach into two parts: symmetric and asymmetric; based on the dependencies between source and target domains [71]. Figure 12 shows the overall categorization of such classes.

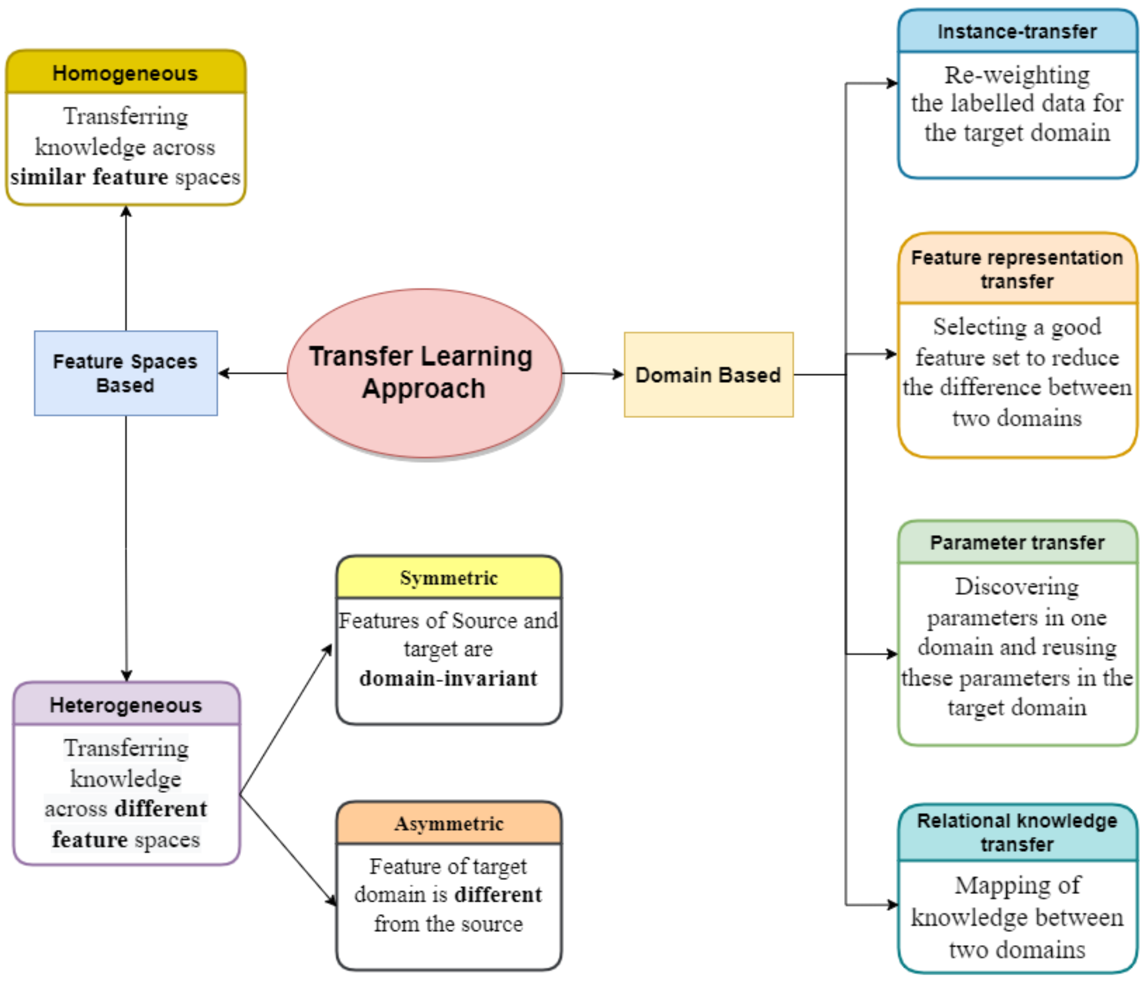

Homogeneous
Transferring knowledge across similar feature spaces
Feature Spaces Based
Transfer Learning Approach
Domain Based
Heterogeneous
Transferring knowledge across different feature spaces
Symmetric
Features of Source and target are domain-invariant
Asymmetric
Feature of target domain is different from the source
Instance-transfer
Re-weighting the labelled data for the target domain
Feature representation transfer
Selecting a good feature set to reduce the difference between two domains
Parameter transfer
Discovering parameters in one domain and reusing these parameters in the target domain
Relational knowledge transfer
Mapping of knowledge between two domains

Figure 12: Transfer learning approaches. This figure has been generated by author inspired from [69],[70],[71]

3.4.2 Transfer Learning using Pre-trained Models

Over the last decade, there has been a tremendous development in creating DL models with a high degree of accuracy. These models have been trained on millions of pieces of data using high performance computers. This opens up the possibility to use these models to solve similar tasks using the transfer learning approach. The idea behind this is to use the knowledge learned from the readily available model and apply it to solve an analogous task that suffers from data scarcity. Some of these popular CCN models are AlexNet, GoogleNet, VGG, MobileNet, and Tensorflow-model-maker which have been used extensively in object recognition and image classification problems.

In general, using a pre-trained model means using the weights (also known as features) that have been berthed in different activation layers. This process is called deep feature extraction. The other method is fine-tuning, where the input and convolution layers are left unchanged, and the final layer is optimized to represent the features of the new dataset. Then the old model is re-trained for the target task to produce updated weights. Fine-tuning is commonly used to solve problems in the same domain.

Using a pre-trained model often shortens the training time since all parameters do not need to be estimated from scratch. Compared to higher layers, the lower layer contains more generalized features that can be easily used for a newer task. Therefore, reusing an existing model is often very convenient to solve an analogous problem.

3.5 MobileNetV2

MobileNetV2 is one of the most popular CNN architectures designed specifically for resource-constrained devices for real-time detection. This architecture was first proposed by Google [72] in 2019. It is an updated version of MobileNetV1. This architecture is an improved version of RestNet, where a new layer called Inverted Residual Block has been introduced instead of Residual Block. It also uses depth-wise separable convolution which effectively minimizes the model parameter to 3.4 million, which is much lower than its competitors such as SuffleNet, NasNet, etc. [72]. This reduces the model size and makes MobileNetV2 a better candidate for lightweight device applications [72]. The backbone of this model is the inverted residual block along with the depth-wise separable convolution and linear bottleneck block. Below is a brief description of each of these elements.

3.5.1 Depth-wise Separable Convolution

Depth-wise separable convolution is the backbone of any EfficientNet architecture. The process consists of a 3×3 depth-wise convolution followed by a 1×1 pointwise convolution [72]. In this pointwise convolution, the previous steps are summed, and a new result is generated without changing the dimension. The comparison between the normal, depth-wise and depth-wise separable convolution is described in Figure 13.

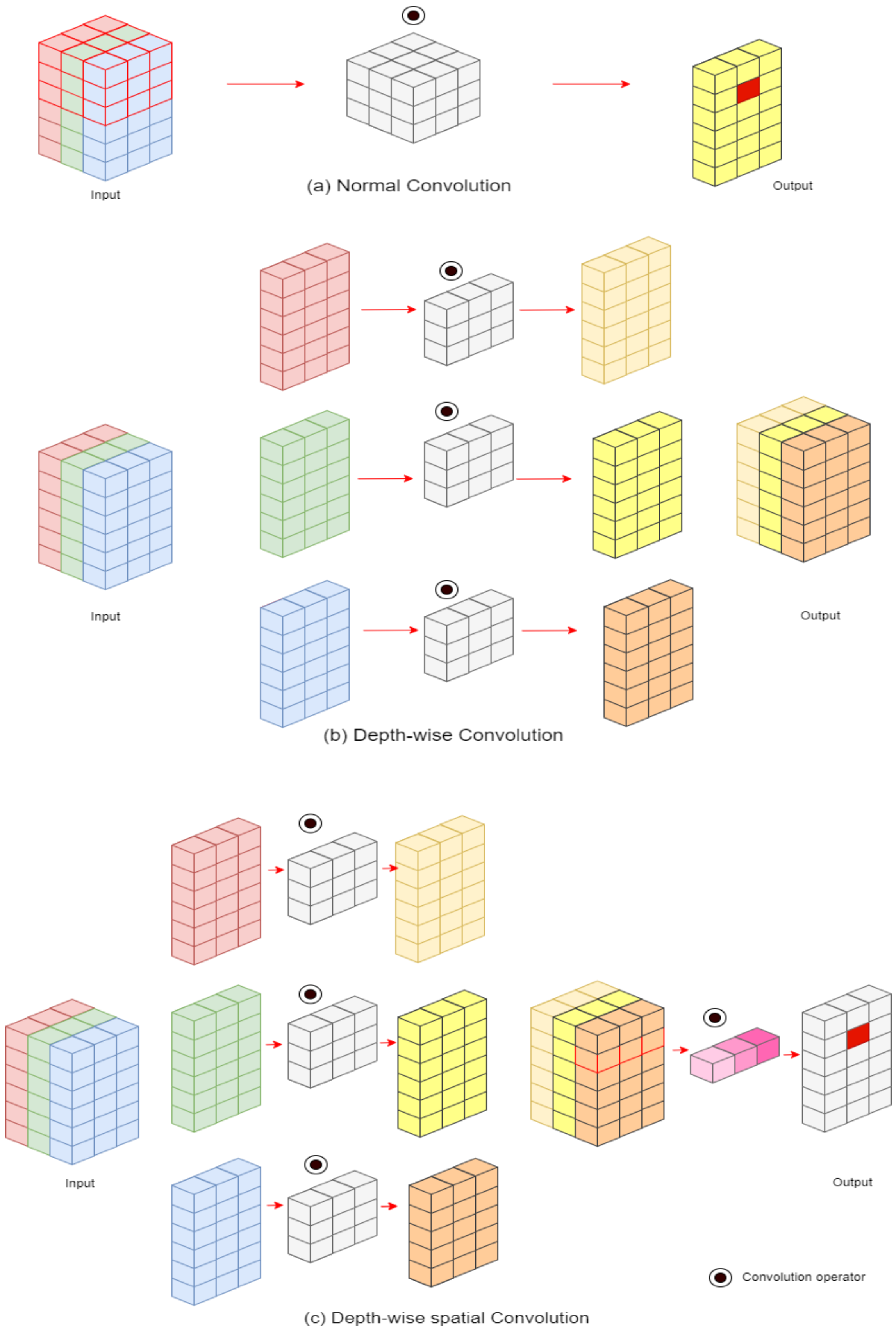


Figure 13: Different type of convolutional operations (a) normal; (b) depth-wise; (c) depth-wise spatial convolution. Image is recreated by the author, inspired from [72].

3.5.2 Linear Bottleneck

Bottleneck block was first proposed by He et al., which effectively reduces the training time by reducing the dimension of the input image and performing a 1×1 convolution at the beginning [72]. Then, the actual 3×3 convolution is performed, which then undergoes by a 1×1 convolution to obtain the same feature dimension in the output (see Figure 14). However, this non-linear ReLU operation just before the output layer affects the output performance as it suppresses all weights smaller than zero and causes a loss of information. In MobileNetV2, this nonlinear convolutional block has been eliminated, resulting in improved performance [72]. Since the non-linear block is no longer present, it is referred to as a linear bottleneck.

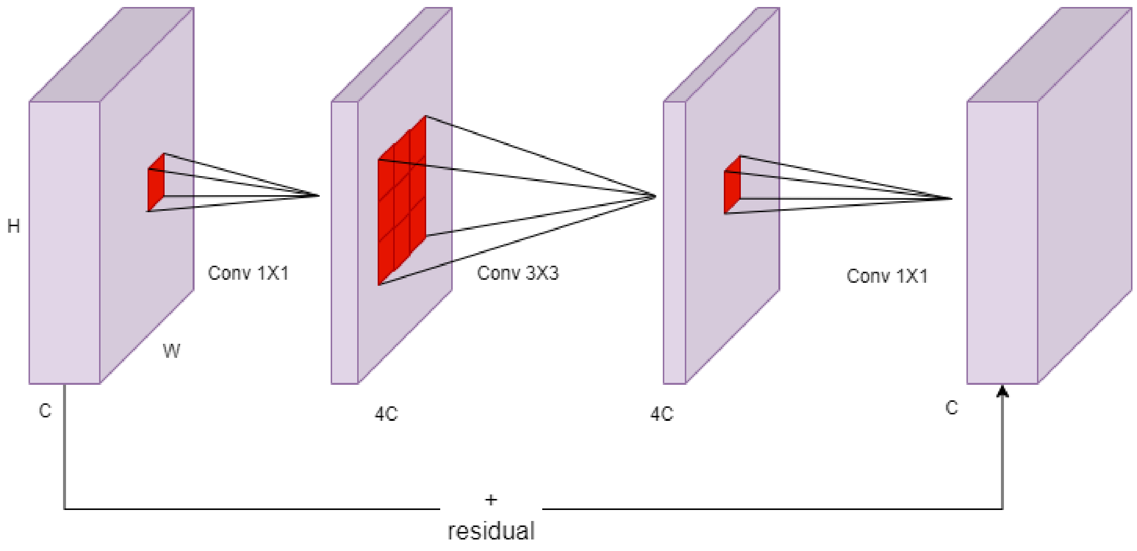


Figure 14: Linear bottleneck operation.

3.5.3 Inverted Residual Block

The inverted residual block performs the reverse operation of the bottleneck block. Instead of shrinking the input, it expands the input data by multiplying it by an expansion factor. The idea behind designing this block is to improve gradient propagation across the connected layer, which ultimately leads to higher memory efficiency. Figure 15 describes the inverted residual operation.

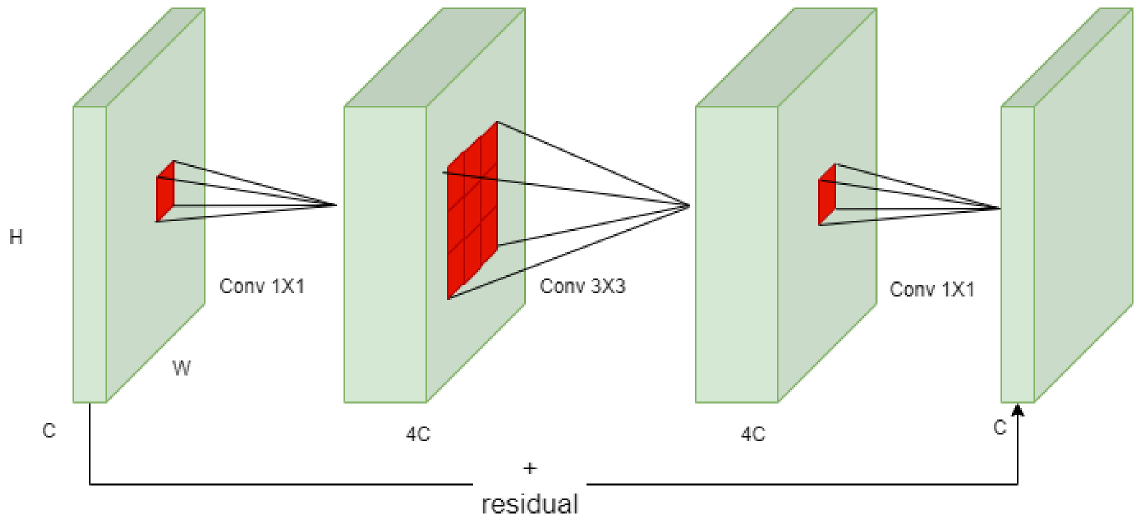


Figure 15: Operation method for Inverse residual block.

3.5.4 MobileNetV2 Model Architecture

As mentioned earlier, the inverted residual block, linear bottleneck, and depth-wise separable convolution are the main elements of MobileNetV2. Based on this, the architecture shown in Table 4 and Figure 16 shows the description of the convolutional layer. From this, it can be deduced that the input undergoes the 1×1 convolution with an expansion factor *t*. In the literature, the value of *t* = 6 has been chosen for ImageNet classification, which gives good inference [72]. Then, the dimension is reduced by depth-wise separable convolution operation, followed by a 1×1 linear convolution that results in a new channel *k*′.

Table 4: Convolutional operation in MobileNetV2 [72]

Input	**Operator**	**Output**
$\boldsymbol{h \times w \times k}$	1×1 conv2d, ReLU6	$\boldsymbol{h \times w \times (tk)}$
$\boldsymbol{h \times w \times tk}$	3×3 dwise s=s, ReLU6	$\boldsymbol{h/_s \times w/_s \times (tk)}$
$\boldsymbol{h/_s \times w/_s \times tk}$	linear 1×1 conv2d	$\boldsymbol{h/_s \times w/_s \times k'}$

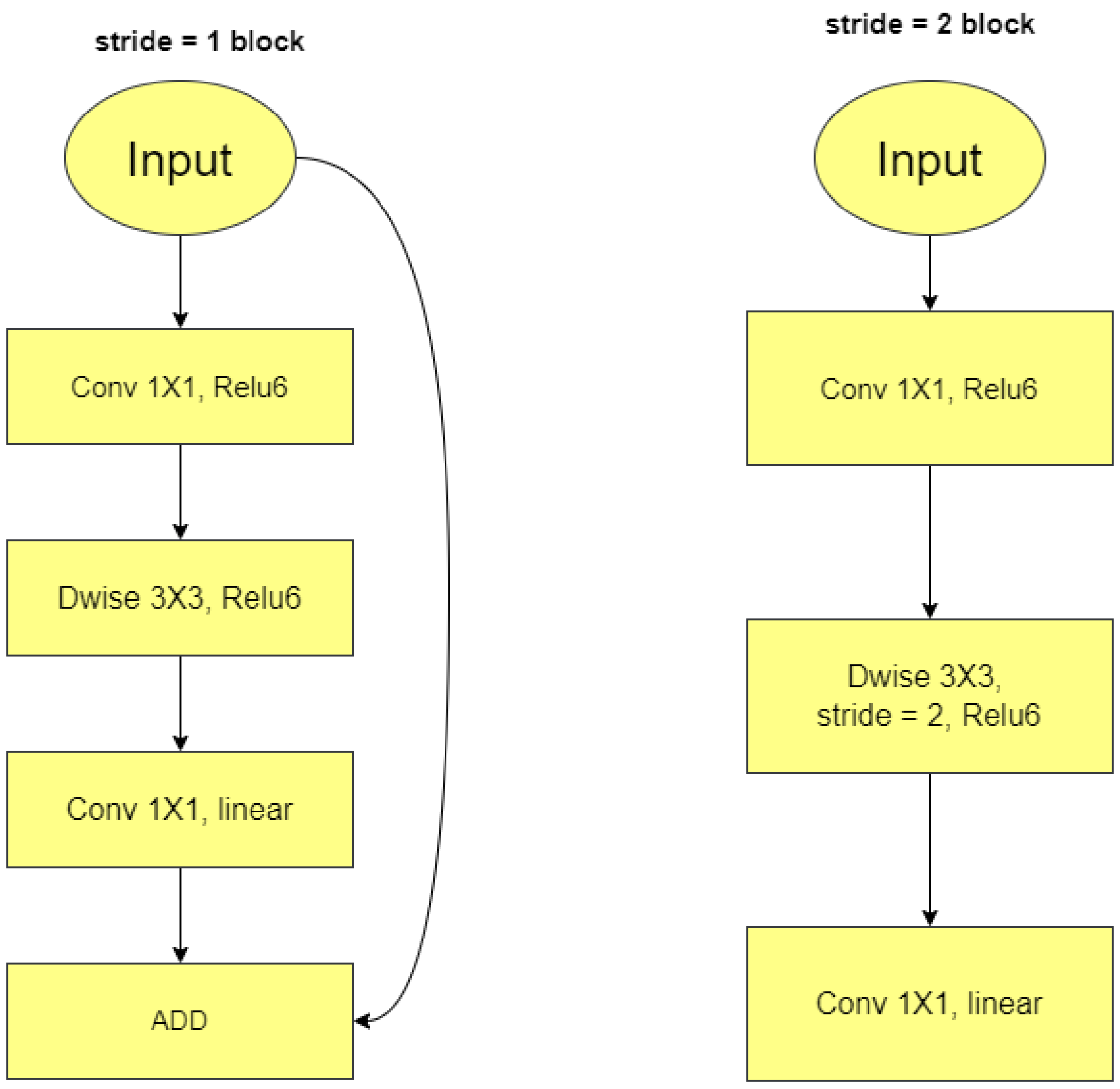


Figure 16: Convolution algorithm of MobileNetV2 [72]

In this MSc thesis, the MobileNetV2 architecture is used for investigating transfer learning technique.

4 Implementation Workflow

This chapter is focused on the design, implementation and experimentation process performed throughout this thesis work. Figure 17 exhibits the full process that has been followed in this project. For successful implementation, the whole work is divided into four major parts: data and platform preparation, CNN architecture development, DL model creation (including model training) and finally detection with SBC and benchmarking (including model size reduction to use it in SBC to determine classification performance). The detailed explanation for each of these steps are described in following subsections.

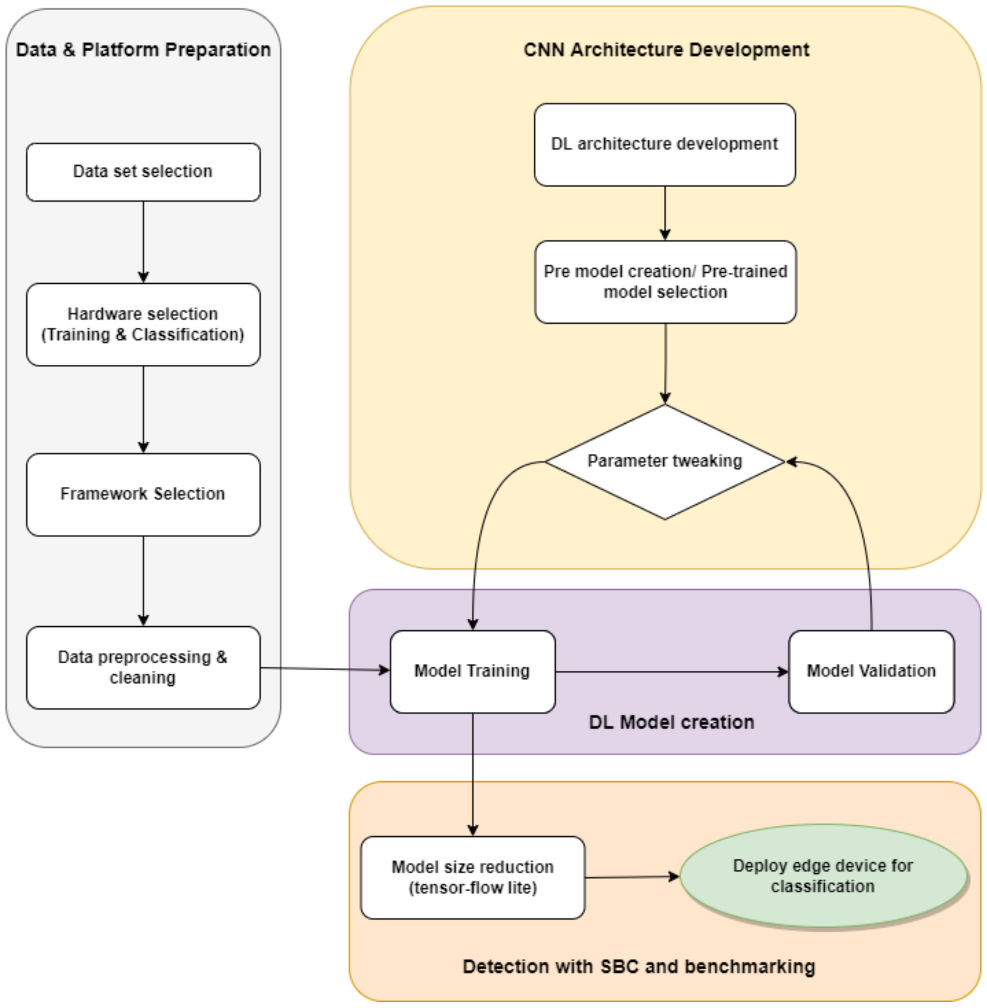

Data & Platform Preparation
Data set selection
Hardware selection (Training & Classification)
Framework Selection
Data preprocessing & cleaning
CNN Architecture Development
DL architecture development
Pre model creation/ Pre-trained model selection
Parameter tweaking
Model Training
Model Validation
DL Model creation
Model size reduction (tensor-flow lite)
Deploy edge device for classification
Detection with SBC and benchmarking

Figure 17: Implementation process followed for this MSc thesis project.

4.1 Dataset Selection

For training the system model, sample data has been chosen which will meet the criteria of interest for this MSc thesis in line with an ongoing research project (PRG620) at Thomas Johann Seebeck Department of Electronics at TalTech. In particular, the size of the objects to be detected and classify should be in the order of a few micrometres. After consulting with researchers involved in this research project, it has been decided to use a dataset of blood cells, specifically the “Blood Cell Images” dataset [73] available on the Kaggle repository.

The “Blood Cell Images” dataset contains 12 500 so-called augmented images of blood cells in the JPEG format. They are provided with corresponding cell type labels in a CSV format. The images are approximately equally divided as 3000 images corresponding to four different types of cells, namely Eosinophil, Lymphocyte, Monocyte, and Neutrophil. More details about the contents and organization of this dataset can be found in [73]. Figure 18 shows four sample images from the dataset.

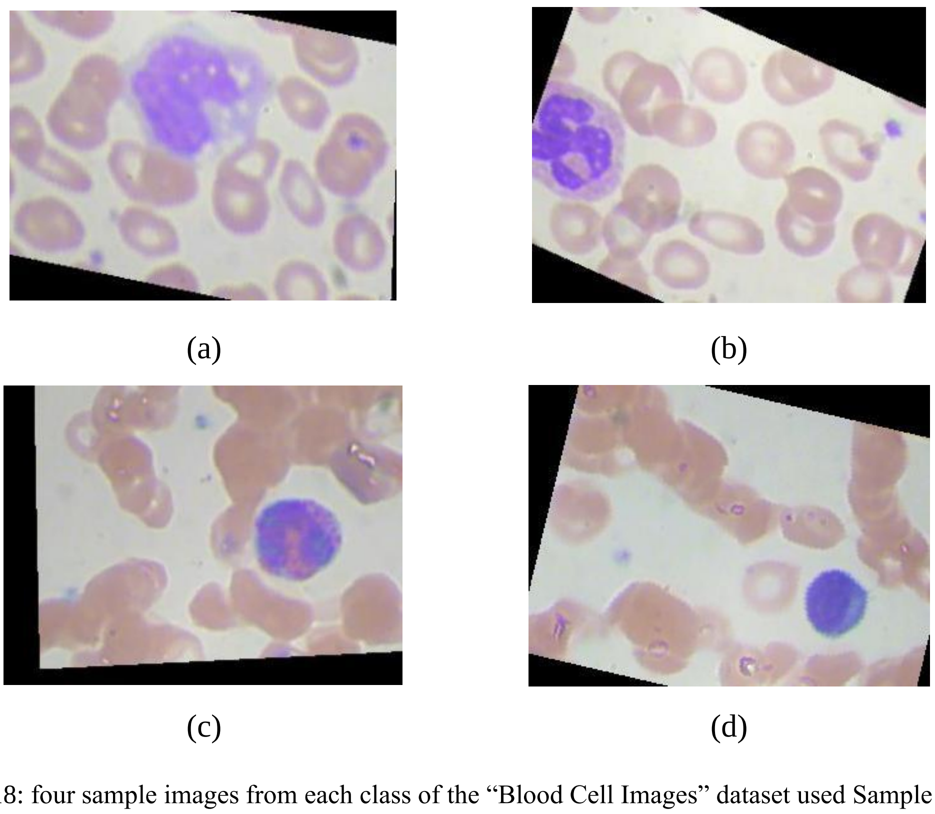


Figure 18: four sample images from each class of the “Blood Cell Images” dataset used Sample Dataset for Training purpose: (a) Eosinophil blood cell type (b) Lymphocyte blood cell type (c) Monocyte blood cell type and (d) Neutrophil blood cell type

4.2 Hardware Selection

Hardware selection is divided into two parts. The first one is for training which requires access to hardware with high computational power. Thus, the online Google-Colaboratory platform is used for this scenario. It is a free platform which provides access to GPUs such as Nvidia K80, T4, P4, P100, 12 GB of RAM and 107 GB of memory for each session. These sessions' allocation are time-limited, usually for 12 hours; there is no guaranty to access Colab resources; the usage limits can change over time. Thus, regular monitoring is required during training phase.

The second one is the selection of SBC for image classification. In recent year, the introduction of TPU in SBCs enhanced its computational ability into next level for DL application. Thus, *Coral Dev* Board manufactured by *Google* is often considered for object detection for its GPU and tensor TPU processor. *NVIDIA Jetson Nano* gives a neck-to-neck challenge to the *Coral Dev* board with its "*Cuda Core*" AI accelerator for portable object detection and computer vision industry. Apart from that, *Raspberry Pi 4, ODROID-C4, Orange Pi 4* etc. are some good candidates for image classification backed up by good amount of scientific literature. For some on-going PhD work at Thomas Johann Seebeck Department of Electronics at TalTech, a detailed list of SBCs has been prepared. From there, the characteristics of some SBCs are mentioned in Table 5.

Table 5 Comparison of different SBCs (partly based on onoing PhD works at Thomas Johann Seebeck Department of Electronics)

Board Name	SoC	Processor (Number of Cores, chip architecture, clock speed...)	Memory
ODROID-N2+	Amlogic S922X Processor	Quad-core Cortex-A73(up to 2.4 GHz) and Dual-core Cortex-A53 (up to 2 GHz), Mali-G52 GPU	DDR4 4 GiB or 2 GiB with 32-bit bus width, 1 x eMMC connector (8G, 16G, 32G, 64G and 128G are available) 1 x microSD slot (DS/HS modes up to UHS-I SDR104)
ODROID-N2	Amlogic S922X Processor	Quad-core Cortex-A73 (1.8 GHz) and Dual-core Cortex-A53 (1.9 GHz) ARMv8-A architecture with Neon and Crypto extensions, Mali-G52 GPU	DDR4 4 GiB or 2 GiB with 32-bit bus width, 1 x eMMC connector (8G, 16G, 32G, 64G and 128G are available) 1 x microSD slot (DS/HS modes up to UHS-I SDR104)

ODROID-C4	Amlogic S905X3	Quad-Core Cortex-A55 (2.016 GHz), Mali-G31 GPU	DDR4 4 GiB with 32-bit bus width, 1x eMMC connector (8/16/32/64 GiB are available) 1x Micro SD slot (DS/HS mode up to UHS-I SDR104)
Raspberry Pi 4	Broadcom BCM2711	Quad core Cortex-A72 (ARM v8), 64-bit SoC @ 1.5GHz	RAM, 2 GiB, 4 GiB, or 8 GiB
Orange Pi 4	Rockchip RK3399 (28nm HKMG process)	6-core ARM® 64-bit processor ,main frequency speeds up to 2.0 GHz Based on the large and small size core architecture of big.LITTLE : Dual-core Cortex-A72 (large core) + Quad-core Cortex-A53 (small core), Mali-T864 GPU Supports OpenGL ES1.1/2.0/3.0/3.1, OpenVG1.1,OpenCL, DX11, support for AFBC	Dual 4 GiB LPDDR4 + 16 GiB EMMC Flash Dual 4 GiB LPDDR4 +EMMC Flash(Default Empty)
ASUS Tinker Board	Rockchip RK3288	QuadCore ARM SOC 1.8 GHz, Mali™-T764 GPU	2 GiB of LPDDR3 dual-channel
NVIDIA Jetson Nano	Nvidia	Quad-core ARM A57 @ 1.43 GHz	4 GiB 64-bit LPDDR4 25.6 GB/s
Google Coral dev. board	NXP i.MX 8M SoC	Quad-core Cortex-A53, plus Cortex-M4F, Google Edge TPU ML , 0 - 66 MHz accelerator coprocessor, Cryptographic coprocessor	1 GiB LPDDR4 , flash: 8 GiB eMMC, MicroSD slot

While there are several promising platforms to choose from, the selection also needs to be done based on the availably of SBCs in the department and in coherence with the related ongoing PhD research works, as well as sufficient references and scientific reports. After careful consideration, the Raspberry Pi 4 has been chosen for this MSc project. Table 6 shows its configuration and Figure 19 shows a photograph of the board.

Table 6: RPI 4 configuration used for this MSc project.

Board Name	SoC	Processor (Number of Cores, chip architecture, clock speed...)	Memory	Price ($)	Input Power	Network Connectivity
Raspberry Pi 4	Broadcom BCM2711	Quad core Cortex-A72 (ARM v8), 64-bit SoC @ 1.5GHz	RAM: 4 GB	$104	5V DC via USB-C connector (minimum 3A*)	2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE

Figure 19: Photograph of the Raspberry Pi 4 (Model B) used in this MSc thesis.

4.3 Framework Selection

Python deep learning ecosystem is the key reason for widespread adoption of DL [55]. Starting with '*Theano*' in 2007, the first deep learning framework [55] this industry undergoes with tremendous research and development and different frameworks have been introduces since then. Among them, *Tensorflow* developed by '*Google brain*' is a very popular DL framework. Apart from this, *Facebook*-developed *Pytorch* and *Keras* are also some of the widely used frameworks for DL application creation. Figure 20 shows available DL framework.

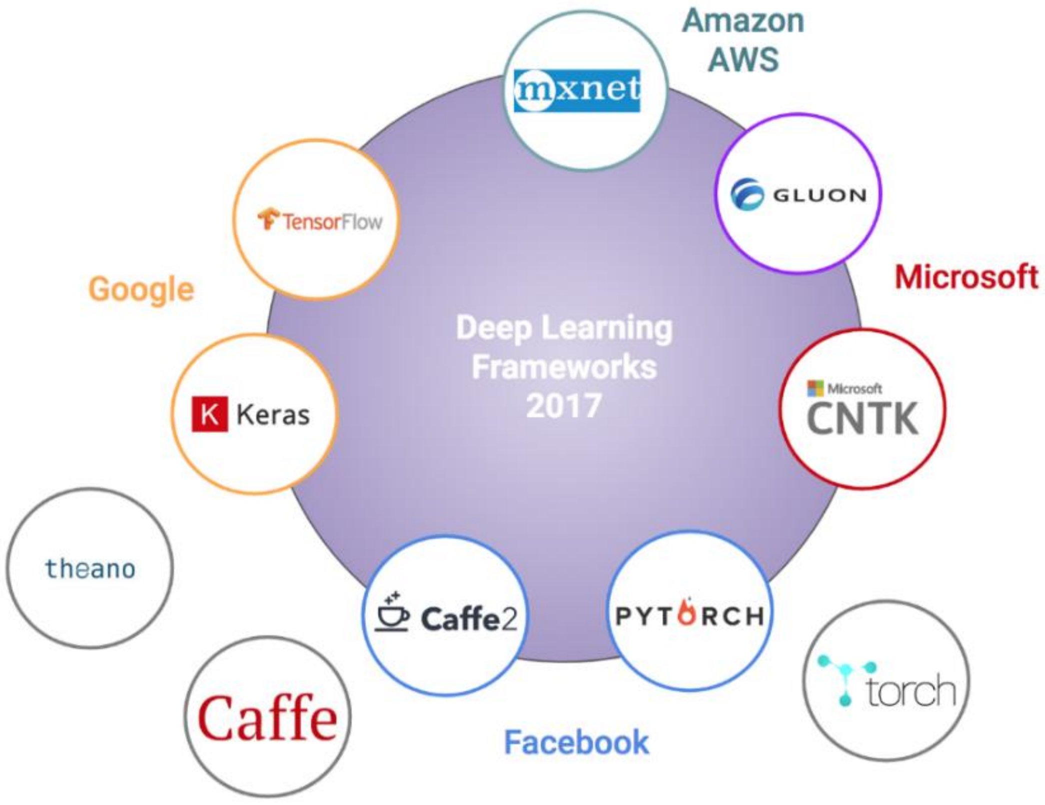


Figure 20: State of DL frameworks [74].

For this MSc thesis work we adopted the *Tensorflow* framework for flexibility of using high-level APIs. It also facilitates porting to SBCs with a relatively easy model size reduction technique in '*Tensorflow lite'*, often written as '*tflite*' which is a light version of *Tensorflow* for resource limited devices. Additionally, some *Keras* libraries are used for CNN architecture development. For image pre-processing, *Open-CV* and *scikit-learn* libraries are used. An overview of the selected framework and libraries is shown in Figure 21.

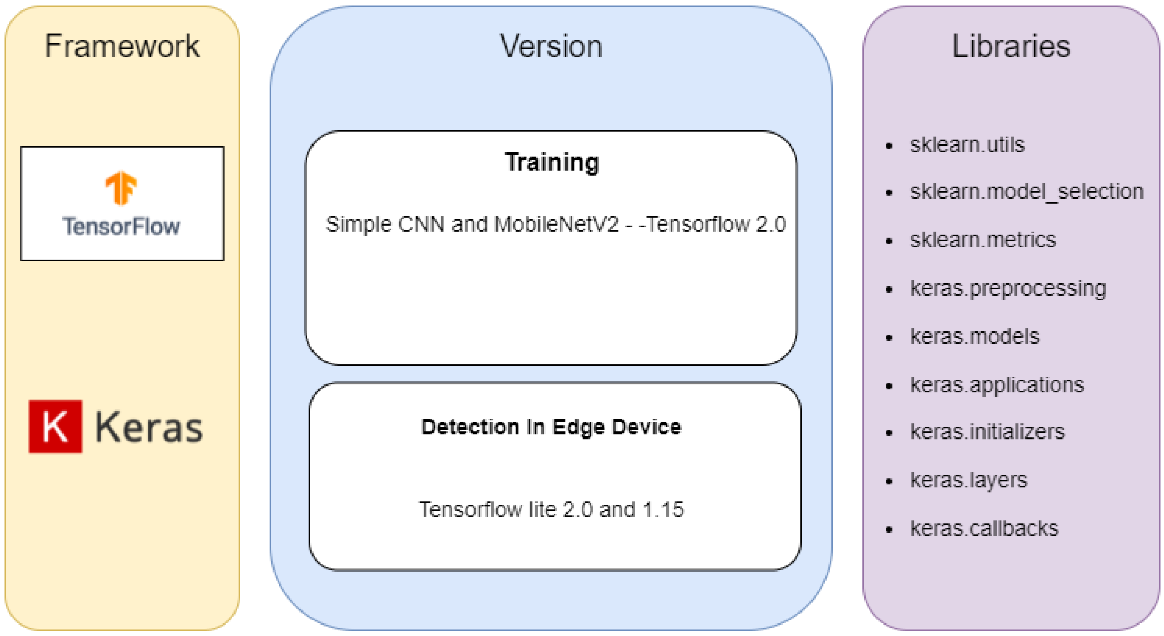
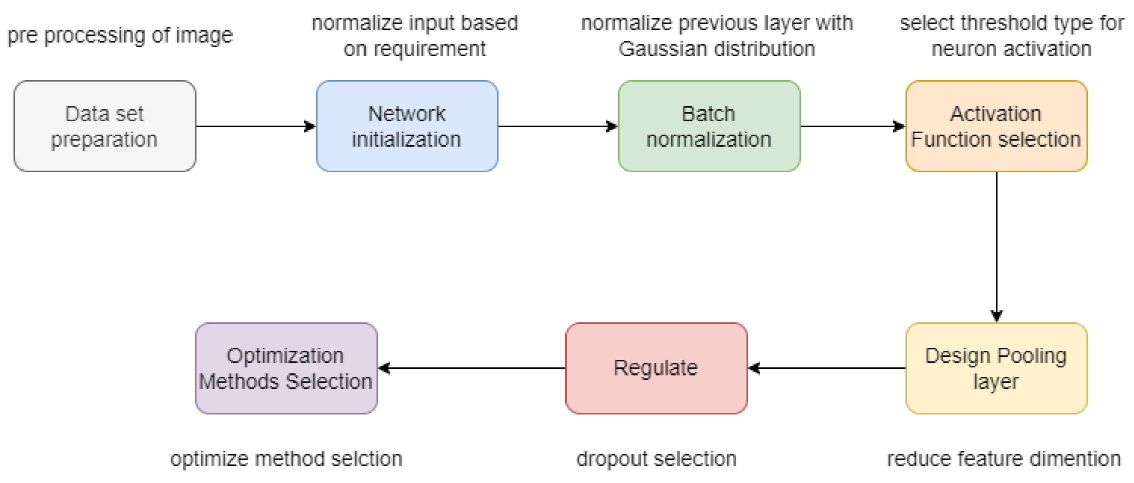


Figure 21: Framework and library information for this MSc thesis work

4.4 Model Preparation

Two different CNN models have been developed for blood cell classification. The first one is a conventional CNN architecture where each layer has been developed from scratch. On the other hand, the second model have been trained using a transfer learning approach. For the second approach (Section 6.5) MobileNetV2 has been used as a pre-trained model. For all models, a conventional ML flow for CNN model development is used, which is shown in Figure 22.


pre processing of image
Data set preparation
normalize input based on requirement
Network initialization
normalize previous layer with Gaussian distribution
Batch normalization
select threshold type for neuron activation
Activation Function selection
Design Pooling layer
reduce feature dimention
Regulate
dropout selection
Optimization Methods Selection
optimize method selction
Figure 22: Flow diagram for CNN model development used in this MSc thesis

4.4.1 Image Pre-possessing

The idea behind image pre-processing is to clean the input image data. Often, different types of objects may be situated in an image. For example, in our case there exists different types of blood-cells, such as red blood cell, eosinophil, neutrophil, etc., in a single image. Our dataset contains two types of blood cells in a single image. For example, for monocyte images, apart from a monocyte cell there also exits some red blood cells. The goal is to extract only the image of monocyte and suppress the rest of the red blood cells. This processing has been done through various mathematical procedures to extract the desired blood cell for which we want train the model and detect later during classification.

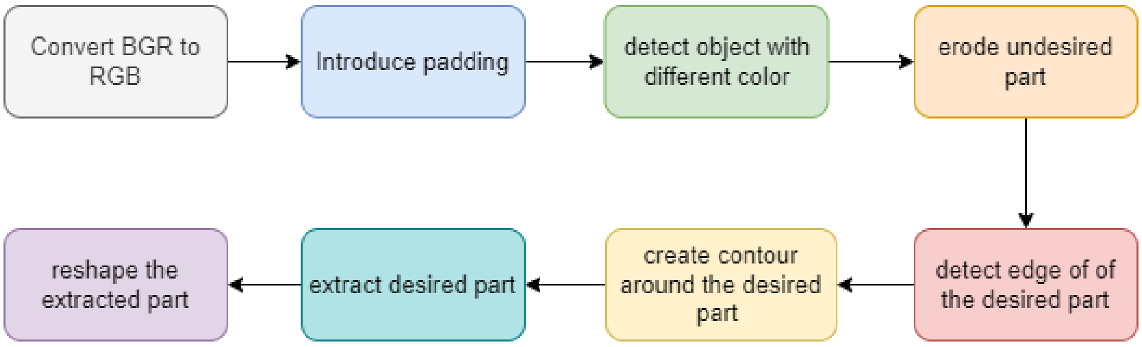

Convert BGR to RGB
Introduce padding
detect object with different color
erode undesired part
detect edge of of the desired part
create contour around the desired part
extract desired part
reshape the extracted part

Figure 23: Image pre-processing process followed in this MSc thesis work.

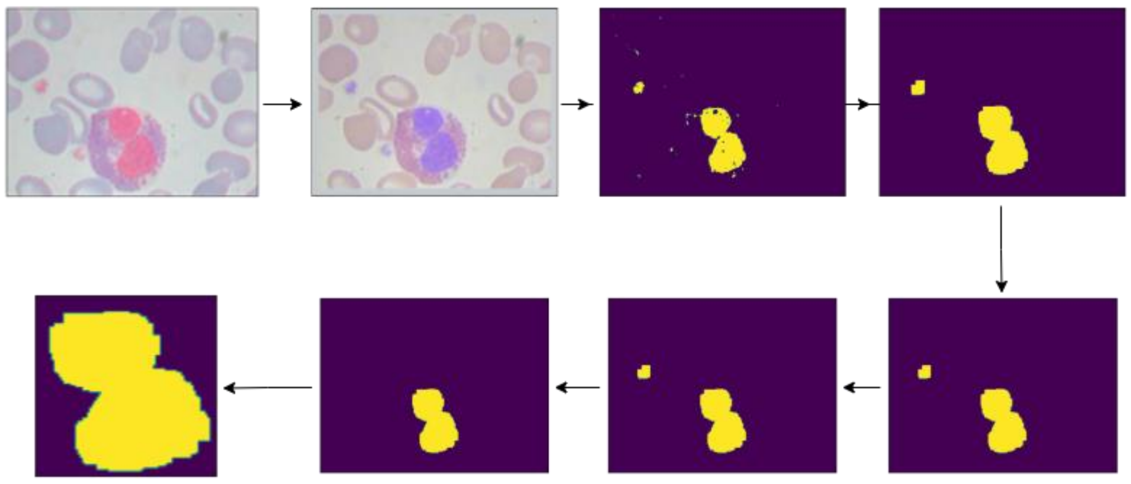


Figure 24: Evolution of an EOSINOPHIL image in pre-processing stage (corresponding to Figure 23).

As shown in Figure 23, to extract a desired blood cell, first we convert the image from Blue Green Red (BGR) to Red Green Blue (RGB) which provides the colour combination of each pixel. Then padding is introduced around the border of the image to overcome information loss. Next, we determine colour differences in different objects in the image. Then the edge of different object is detected, and we draw the contour around the desired portion. Then, the rest of the pixels are suppressed, and the desired portion is extracted out. Finally, the extracted image is resized to a common dimension. The evolution of the image throughout these operations is shown in Figure 24.

Some of the sample data are given after pre-processing stage in Figure 25.

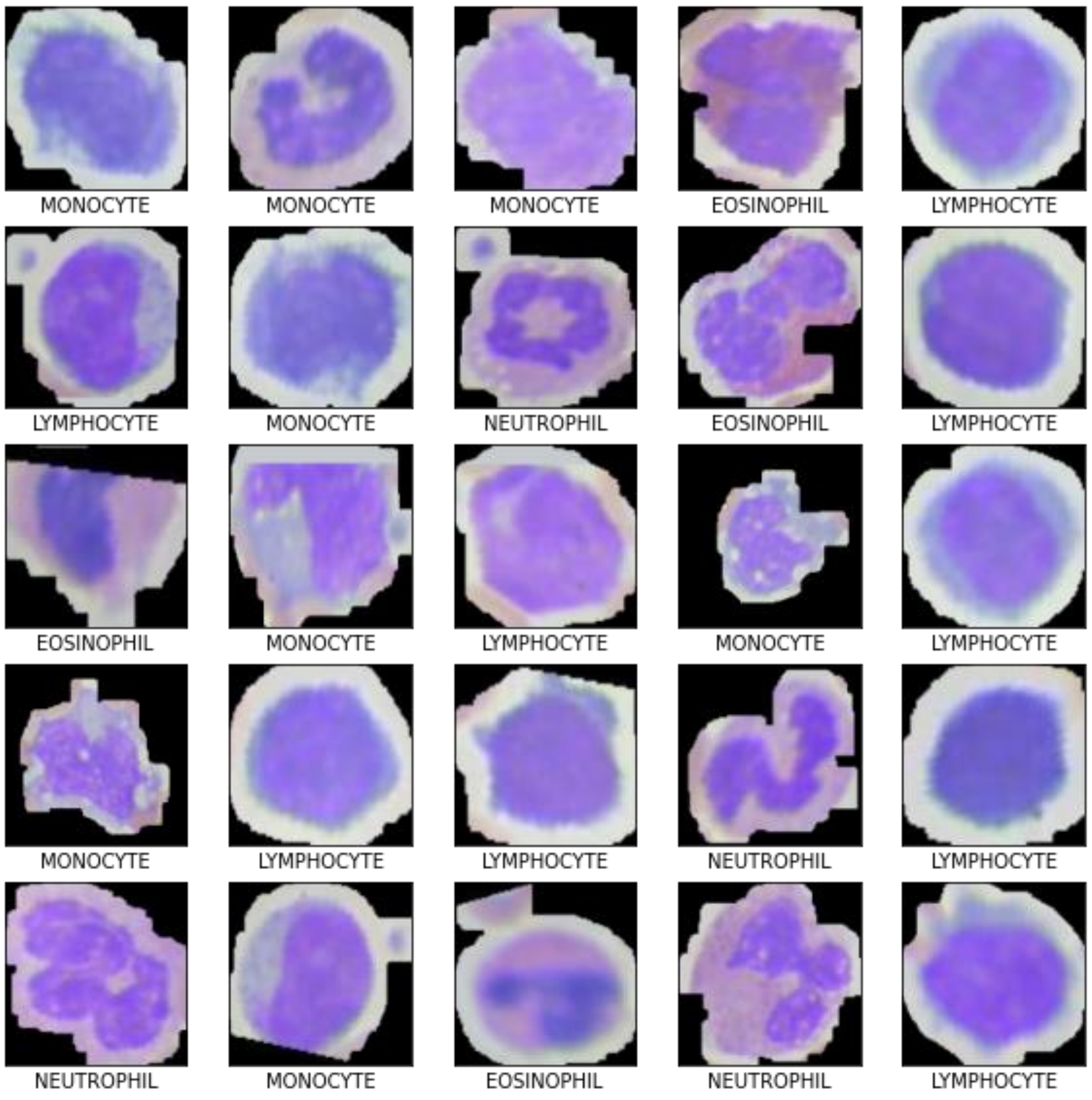


Figure 25: Example of image dataset samples after pre-processing.

4.4.2 Dataset Split for Training and Testing

For CNN model development, the dataset used for training, testing, and validation need to be clearly mentioned. The conventional rule is to use 20% of the total image for validation and testing. Our dataset contains 12000 images with 2500 for each class. Thus, we use around 250 images for testing and 250 images for validation. Figure 26 shows the population of each observed category and Figure 27 shows the train, testing, and validation split used to train this model.

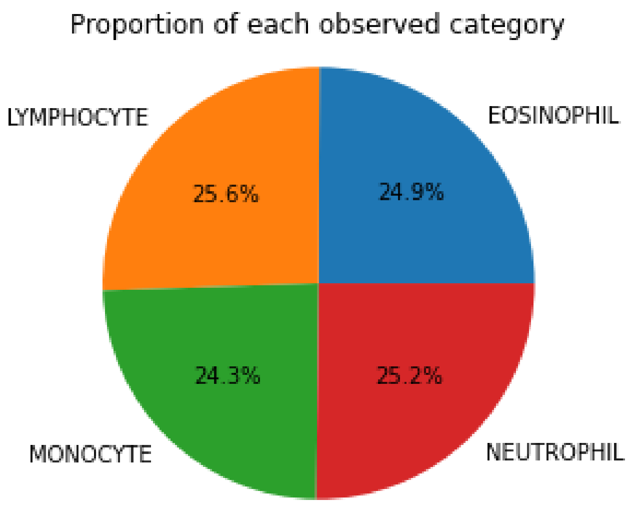


Figure 26: Percentage-wise population for each observed category in the selected dataset

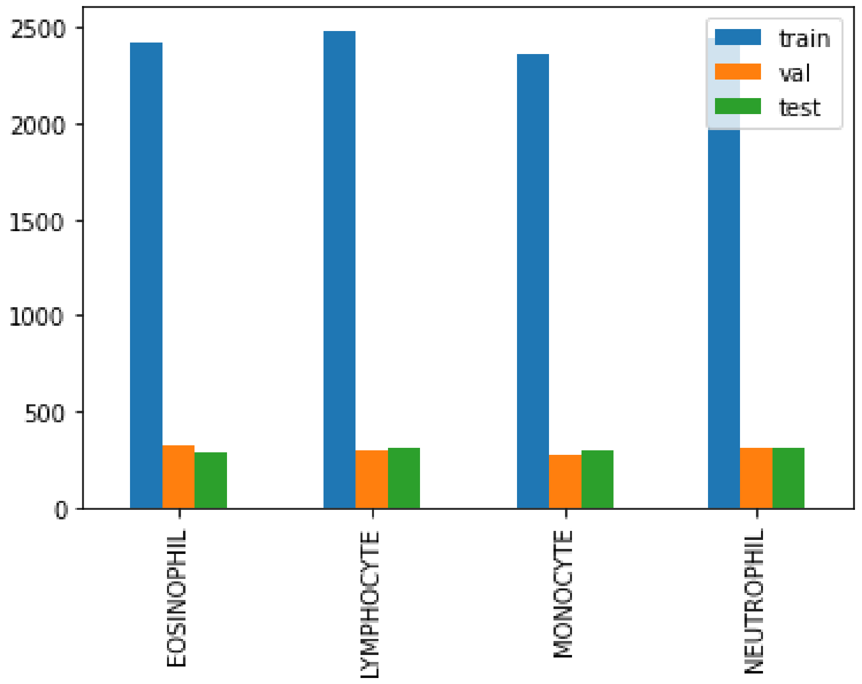


Figure 27: Dataset split for training, testing and validation

4.4.3 CNN Layer Design

For the CNN layer, we consider five convolutional blocks followed by a fully connected (FC) layer and output layer. The first convolution block (input layer) uses 2D convolution whereas the remaining convolutional blocks use separable 2D convolution. For the input layer, the number of neurons (output shape) is kept at 16 and is doubled in each of the hidden layers. Moreover, each of the convolution blocks use a *ReLU* activation function, maxpooling technique, and 3×3 kernel for the RGB image. We also use padding to reduce information loss.

The FC layer uses deeply connected neural network as per convention. Here the '*tanh'* activation function is used with different dropout rate at it moves towards the output. Finally, in the output layer we use a '*softmax*' activation function as per practice in the industry [55]. Table 7 describes each of the layers' configuration. This whole architecture is inspired by [74].

Table 7 CNN model layers' configuration

Layer type	Output Shape	Number of parameters
conv2d_4 (Conv2D)	(None, 120, 120, 16)	448
conv2d_5 (Conv2D)	(None, 120, 120, 16)	2320
max_pooling2d_10 (MaxPooling2D)	(None, 60, 60, 16)	0
separable_conv2d_16 (SeparableConv2D)	(None, 60, 60, 32)	688
separable_conv2d_17 (SeparableConv2D)	(None, 60, 60, 32)	1344
batch_normalization_8 (BatchNormalization)	(None, 60, 60, 32)	128
max_pooling2d_11 (MaxPooling2D)	(None, 30, 30, 32)	0
separable_conv2d_18 (SeparableConv2D)	(None, 30, 30, 64)	2400
separable_conv2d_19 (SeparableConv2D)	(None, 30, 30, 64)	4736
batch_normalization_9 (BatchNormalization)	(None, 30, 30, 64)	256
max_pooling2d_12 (MaxPooling2D)	(None, 15, 15, 64)	0
separable_conv2d_20 (SeparableConv2D)	(None, 15, 15, 128)	8896
separable_conv2d_21 (SeparableConv2D)	(None, 15, 15, 128)	17664
batch_normalization_10 (BatchNormalization)	(None, 15, 15, 128)	512
max_pooling2d_13 (MaxPooling2D)	(None, 7, 7, 128)	0
dropout_10 (Dropout)	(None, 7, 7, 128)	0
separable_conv2d_22 (SeparableConv2D)	(None, 7, 7, 256)	34176
separable_conv2d_23 (SeparableConv2D)	(None, 7, 7, 256)	68096
batch_normalization_11 (BatchNormalization)	(None, 7, 7, 256)	1024
max_pooling2d_14 (MaxPooling2D)	(None, 3, 3, 256)	0

dropout_11 (Dropout)	(None, 3, 3, 256)	0
flatten_2 (Flatten)	(None, 2304)	0
dense_8 (Dense)	(None, 512)	1180160
dropout_12 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 128)	65664
dropout_13 (Dropout)	(None, 128)	0
dense_10 (Dense)	(None, 64)	8256
dropout_14 (Dropout)	(None, 64)	0
dense_11 (Dense)	(None, 4)	260
Total parameters: 1397028 Trainable parameters: 1396068 Non-trainable parameters: 960		

4.5 Evaluation of Trained Model

For model evaluation, the conventional process is to check the training and validation accuracy and loss. Additionally, a confusion matrix has been generated to get a visual understanding of the models' performance. This confusion matrix identifies the following elements:

- **True Positive**: prediction that the object belongs to a class and the object actually belongs to that class;
- **True Negative**: prediction that the object does not belong to a class but the object actually does not belong to that specific class;
- **False Positive**: prediction that the object belongs to a class, and the object actually does not belong to that class;
- **False Negative**: prediction that the object does not belongs to a class but the object actually belongs to that class.

Figure 28 shows the principle of the confusion matrix.

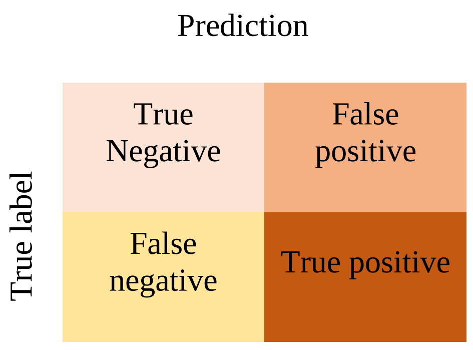


Figure 28: Principle of confusion matrix

To evaluate our model we also follow the same procedure. The following sub-section describes the performance of our model in detail. Table 8 introduces the parameters used to evaluate these models.

Table 8: Evaluation parameter of DL model

Evaluation Parameter	**Description of the parameter**	**Value range**
Training Accuracy	Measurement of accurate classification between training data and testing data.	0 to 100%, i.e. 0.0 to 1.0
Validation Accuracy	Measurement of accurate prediction from the validation dataset	0 to 100%, i.e. 0.0 to 1.0
Training loss	Measurement of inaccurate prediction between training and testing data. It is measured after each batch	0 to ∞
Validation loss	Measurement of inaccurate prediction between training and testing data. Generally measured after each epoch	0 to ∞
Precision	$Precision = \frac{true\ positive}{true\ positive + false\ positive}$	0 to 100%, i.e. 0.0 to 1.0

4.5.1 Optimizer Selection

As discussed in Subsection 3.2.7, the optimizer selection is one of the important tasks for DL model training. There are several optimizers available, but among them Adam and

RMSprop are the most common and widely used in DL architecture. Usually, optimizers are model oriented and thus always selected by trial-and-error method. In this work, we compare these two optimizers to check their impact on the training and figure out the suitable one from the model.

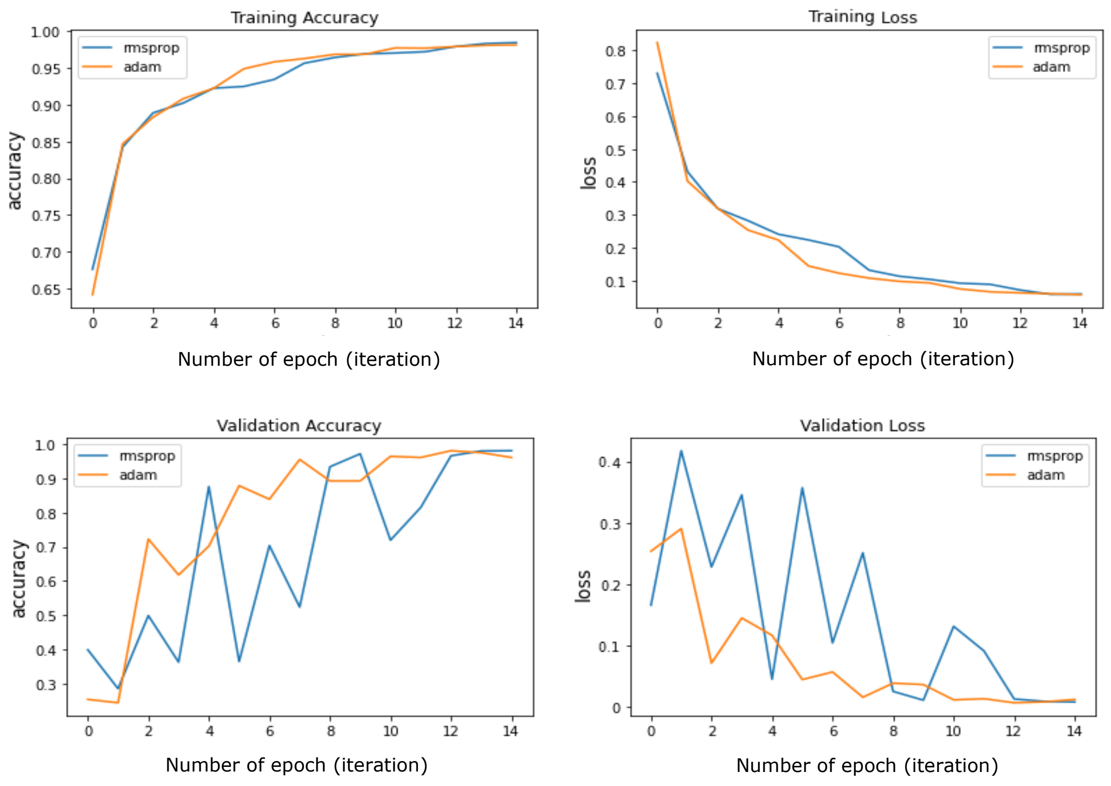


Figure 29: Impact of Adam and RMSprop on model training in terms of training accuracy and training loss, as well as validation accuracy and validation loss.

Figure 29 shows the performance of Adam and RMSprop for model training. Here a pre-model has been developed for only 15 epochs (i.e. “training iterations”) to determine the performance. It is clear that the Adam optimizer performs slightly better (higher accuracy and lower loss, on average) than the other one for our model. Additionally, the training time for Adam was less than the RMSprop. Thus, we opted for Adam.

4.5.2 Learning Rate Selection

Another important parameter which is also figured out by trial-and-error is the learning rate (η). The impact of the learning rate on the CNN model is described in Subsection 5.2.1.3. Learning rate values like .000001, 0.00001, 0.00005, 0.0001 are common practice in DL. In this MSc work, we compared the impact for 15 epochs (i.e. “training iterations”) and found η = 0.00001 performs slightly better than the others. The comparison is shown in Figure 28.

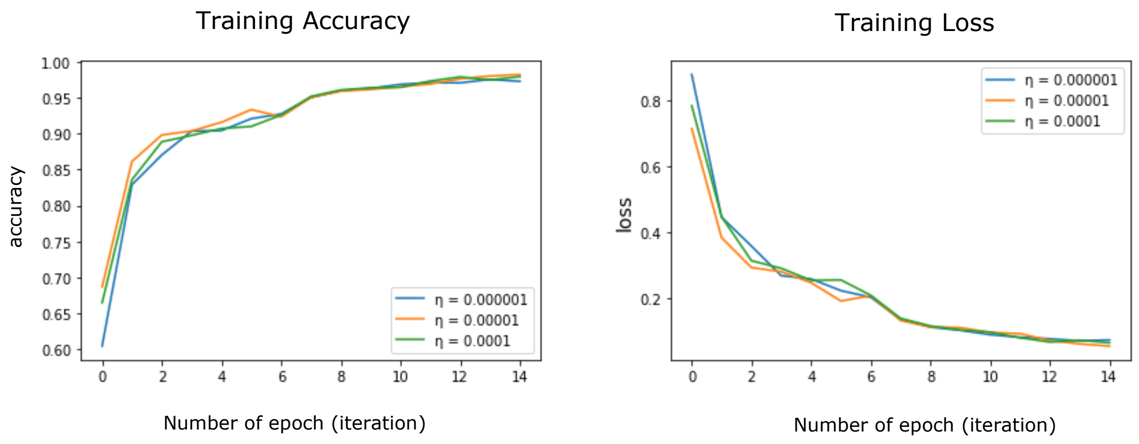


Figure 30: Effect of different learning rate (η) in model.

4.6 Model Preparation using pertained model

As discussed in Section 3.4, there are different CNN models readily available which can serve our purpose of blood cell classification. Among them, ResNet, DenseNet, MobileNet and EfficientNet are some of the popular architectures used for object detection [72]. However, the performance of a model is highly dependent on the data and its preprocessing procedure. Therefore, we characterize the performance of each of the CNN architectures mentioned above that suits our data. For characterization, we use the same preprocessing method mentioned in Subsection 4.4.1 for 15 epochs. The result is shown in Figure 31.

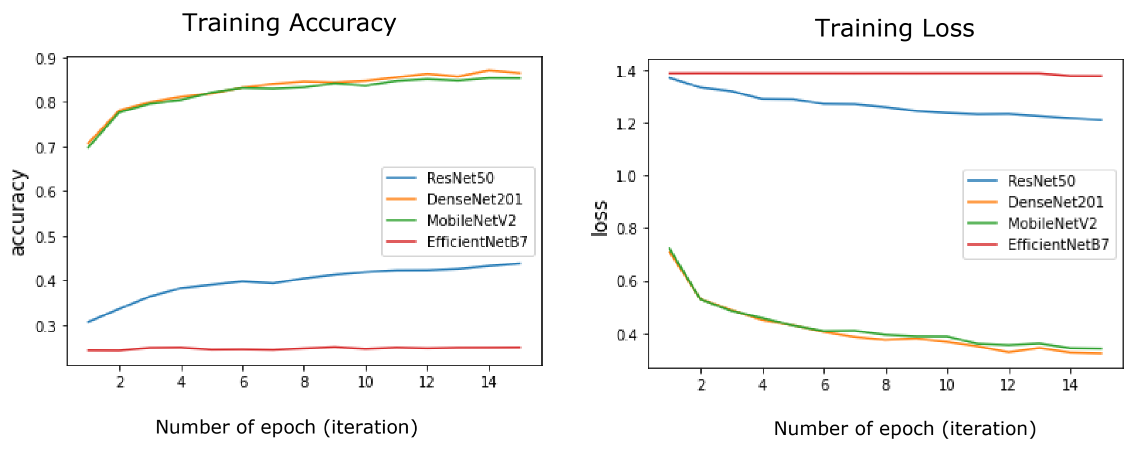
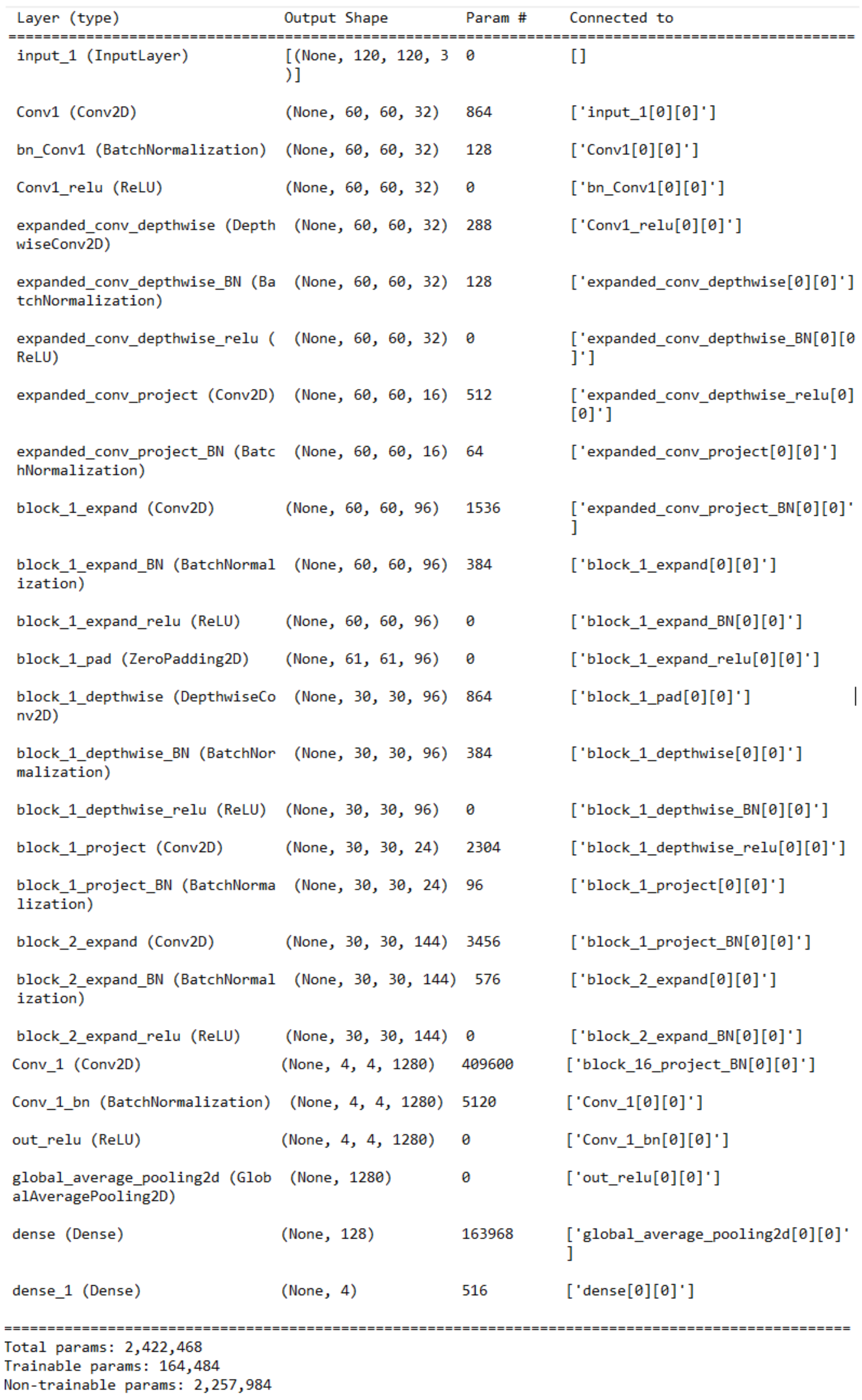


Figure 31: Performance comparison between different CNN pre-trained model.

From Figure 31 it can be seen that DenseNet201 and MobileNetV2 outperform the other two CNN architecture for our dataset. However, MobileNetV2 is more memory efficient [72] with a lower number of parameters compared to DenseNet. Thus, we choose MobileNetV2.

4.6.1 Data Pre-processing & Model Preparation

To train the model using MobileNetV2, the dataset needs to be pre-processed. For pre-processing, the same methodology mentioned in Subsection 4.4.1 is followed. Then, we load the model from the source and modify the input output parameters based on our dataset. Similarly to the previous model, the *softmax* activation function is used for the output while *ReLU* is used for input. The same feature map weight developed for the original MobileNet model are used for training. The summary of the model is shown in Figure 32.



```
 Layer (type)                Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 120, 120, 3  0           []
                             )]

 Conv1 (Conv2D)              (None, 60, 60, 32)   864         ['input_1[0][0]']

 bn_Conv1 (BatchNormalization)  (None, 60, 60, 32)  128       ['Conv1[0][0]']

 Conv1_relu (ReLU)           (None, 60, 60, 32)   0           ['bn_Conv1[0][0]']

 expanded_conv_depthwise (Depth  (None, 60, 60, 32)  288      ['Conv1_relu[0][0]']
 wiseConv2D)

 expanded_conv_depthwise_BN (Ba  (None, 60, 60, 32)  128      ['expanded_conv_depthwise[0][0]']
 tchNormalization)

 expanded_conv_depthwise_relu (  (None, 60, 60, 32)  0        ['expanded_conv_depthwise_BN[0][0
 ReLU)                                                        ]']

 expanded_conv_project (Conv2D)  (None, 60, 60, 16)  512      ['expanded_conv_depthwise_relu[0]
                                                              [0]']

 expanded_conv_project_BN (Batc  (None, 60, 60, 16)  64       ['expanded_conv_project[0][0]']
 hNormalization)

 block_1_expand (Conv2D)     (None, 60, 60, 96)   1536        ['expanded_conv_project_BN[0][0]'
                                                              ]

 block_1_expand_BN (BatchNormal  (None, 60, 60, 96)  384      ['block_1_expand[0][0]']
 ization)

 block_1_expand_relu (ReLU)  (None, 60, 60, 96)   0           ['block_1_expand_BN[0][0]']

 block_1_pad (ZeroPadding2D)  (None, 61, 61, 96)  0           ['block_1_expand_relu[0][0]']

 block_1_depthwise (DepthwiseCo  (None, 30, 30, 96)  864      ['block_1_pad[0][0]']
 nv2D)

 block_1_depthwise_BN (BatchNor  (None, 30, 30, 96)  384      ['block_1_depthwise[0][0]']
 malization)

 block_1_depthwise_relu (ReLU)  (None, 30, 30, 96)  0         ['block_1_depthwise_BN[0][0]']

 block_1_project (Conv2D)    (None, 30, 30, 24)   2304        ['block_1_depthwise_relu[0][0]']

 block_1_project_BN (BatchNorma  (None, 30, 30, 24)  96       ['block_1_project[0][0]']
 lization)

 block_2_expand (Conv2D)     (None, 30, 30, 144)  3456        ['block_1_project_BN[0][0]']

 block_2_expand_BN (BatchNormal  (None, 30, 30, 144)  576     ['block_2_expand[0][0]']
 ization)

 block_2_expand_relu (ReLU)  (None, 30, 30, 144)  0           ['block_2_expand_BN[0][0]']
 Conv_1 (Conv2D)             (None, 4, 4, 1280)   409600      ['block_16_project_BN[0][0]']

 Conv_1_bn (BatchNormalization)  (None, 4, 4, 1280)  5120     ['Conv_1[0][0]']

 out_relu (ReLU)             (None, 4, 4, 1280)   0           ['Conv_1_bn[0][0]']

 global_average_pooling2d (Glob  (None, 1280)     0           ['out_relu[0][0]']
 alAveragePooling2D)

 dense (Dense)               (None, 128)          163968      ['global_average_pooling2d[0][0]'
                                                              ]

 dense_1 (Dense)             (None, 4)            516         ['dense[0][0]']

==================================================================================================
Total params: 2,422,468
Trainable params: 164,484
Non-trainable params: 2,257,984
```

Figure 32: MobileNetV2 model summary

4.7 Model Minimization

After developing the model, its size needs to be reduced to make it fit and run on the edge device. One of the processes is to convert the model into Tensorflow-Lite (*tflite*) which is a light version of Tensorflow that compresses the model to make it suitable to run on edge devices. In this MSc thesis, we converted the model into .*tflite* using the high-level API supplied by Tensorflow. Table 7 shows a screenshot that provides a comparison between the sizes of the original Keras models (.*h5*) and the .*tflite* models after conversion. For both Model 1 the size is reduced from 16.5 MB to 5.4 MB (divided by approximately 3). For Model 2 the size reduction is more modest, from 11.2 MB to 9.1 MB (divided by approximately 1.23).

Table 9: Model size comparison after conversion

Model Name	Keras model size (MB)	Converted tflite model size (MB)
Model 1 (simple CNN developed from scratch)	16.5	5.4
Model 2 (pre-trained model with learning transfer)	11.2	9.1

5 Result and Analysis

As explained previously, there are two different models developed for this MSc thesis work. The first one is a simple CNN model built from scratch that has been described in Section 4.4 (Model 1). The other one is a pre-trained model which is used to train our dataset for image classification using transfer learning method (Model 2). Both models were converted into *.tflite* to reduce their sizes and make them suitable for edge devices (Raspberry Pi 4 in our case). This chapter is focused on the evaluation of the trained models.

5.1.1 Evaluation of Model 1

Figure 33 shows the learning curve of our Model 1 for 30 epochs. This figure is a classic example of a DL learning curve. From the figure, it is clear that initially the training accuracy increases exponentially up to fifth epoch and then gradually reaches to an accuracy of 99% by the 15th epoch and maintains this accuracy throughout the remaining iterations. After an initial fluctuation, the validation accuracy also follows a similar trend, although not as smooth. Here the training accuracy for the first epoch is 60%, which clearly indicates that the image pre-processing was very well suited for this model. The loss curve for training was < 1 in the entire training time which is also an indication of a successful model preparation. Moreover, the validation accuracy is slightly lower than the training accuracy while the validation loss is slightly higher than the training loss; this infers that this model is free from overfitting or underfitting problems (see Subsection 3.2.8). This means that the dropout selection for this model is quite ideal.

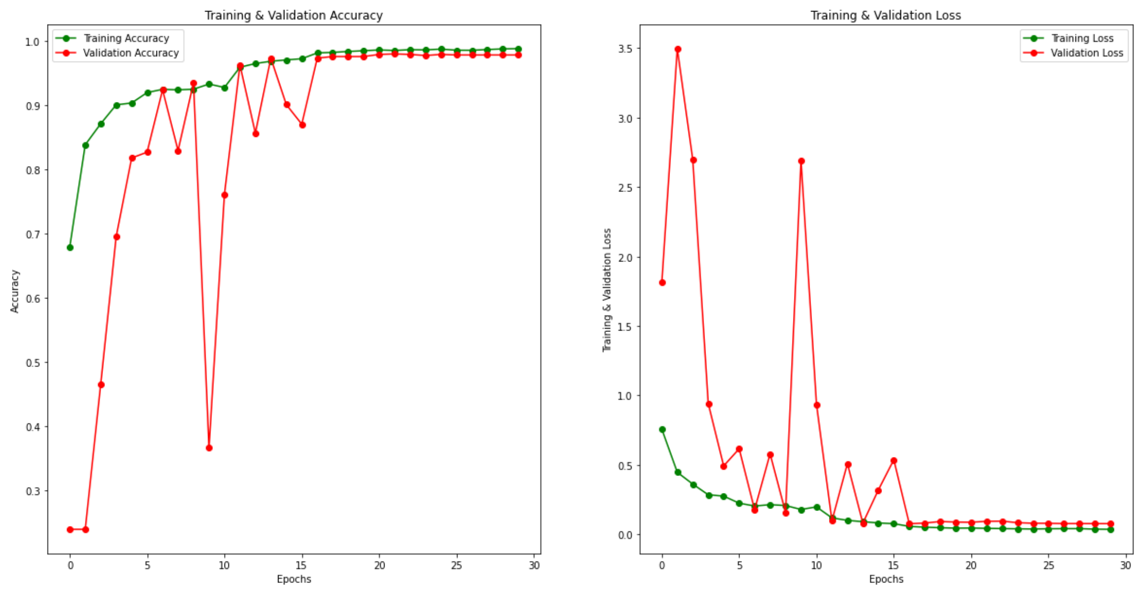


Figure 33: Learning curve of Model 1 for 30 epochs. The X axes represent the epochs ("training iterations") and the Y axes represent the accuracy and loss.

In order to investigate the possibility of overfitting or underfitting issues, we train the model again up to 60 epochs (see Figure 34) and observe similar trends between training and validation accuracies, as well as between training & validation losses. Thus, we can conclude that this model will not suffer from the fitting problem.

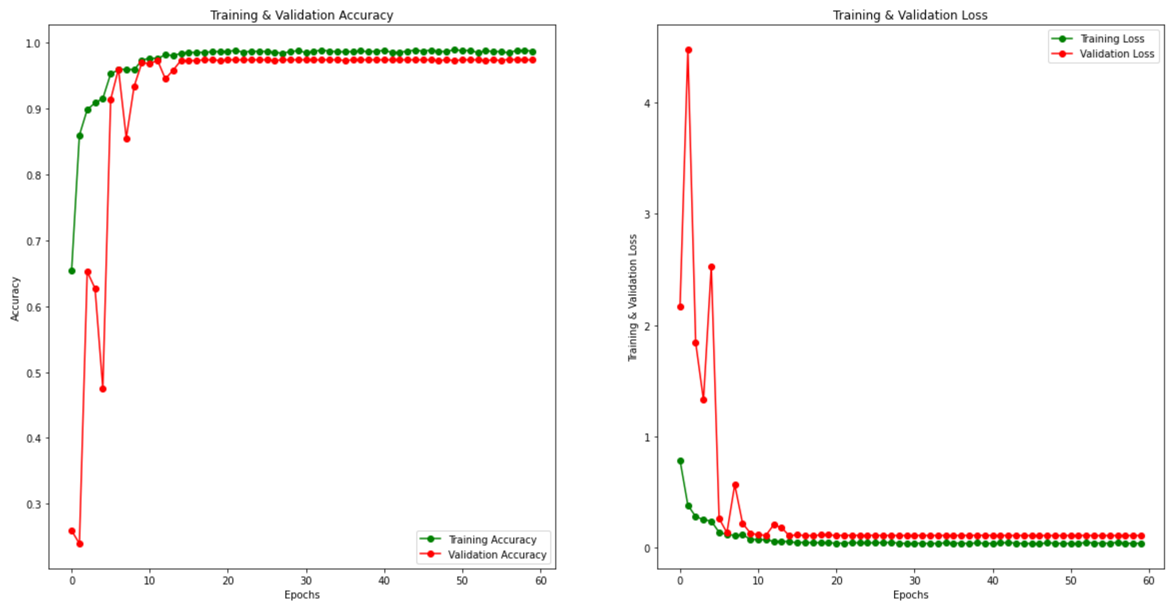


Figure 34: Learning curve of Model 1 for 60 epochs. The X axes represent the epochs ("training iterations") and the Y axes represent the accuracy and loss.

Figure 35 shows the confusion matrix for Model 1, which is also an evidence of the model precision. The percentage of false positive and false negatives (indicated by the light blue color) are almost negligible. Below are the precision calculations for each class and for the overall model:

Precision for Eosinophil $= \frac{272}{272+7} \times 100 = 97.49\%$

Precision for Lymphocyte $= \frac{297}{297+2} \times 100 = 99.33\%$

Precision for Monocyte $= \frac{318}{318+7} \times 100 = 97.84\%$

Precision for Neutrophil $= \frac{300}{300+11} \times 100 = 96.46\%$

Overall, model precision $= \frac{1187}{1187+27} \times 100 = 97.77\%$

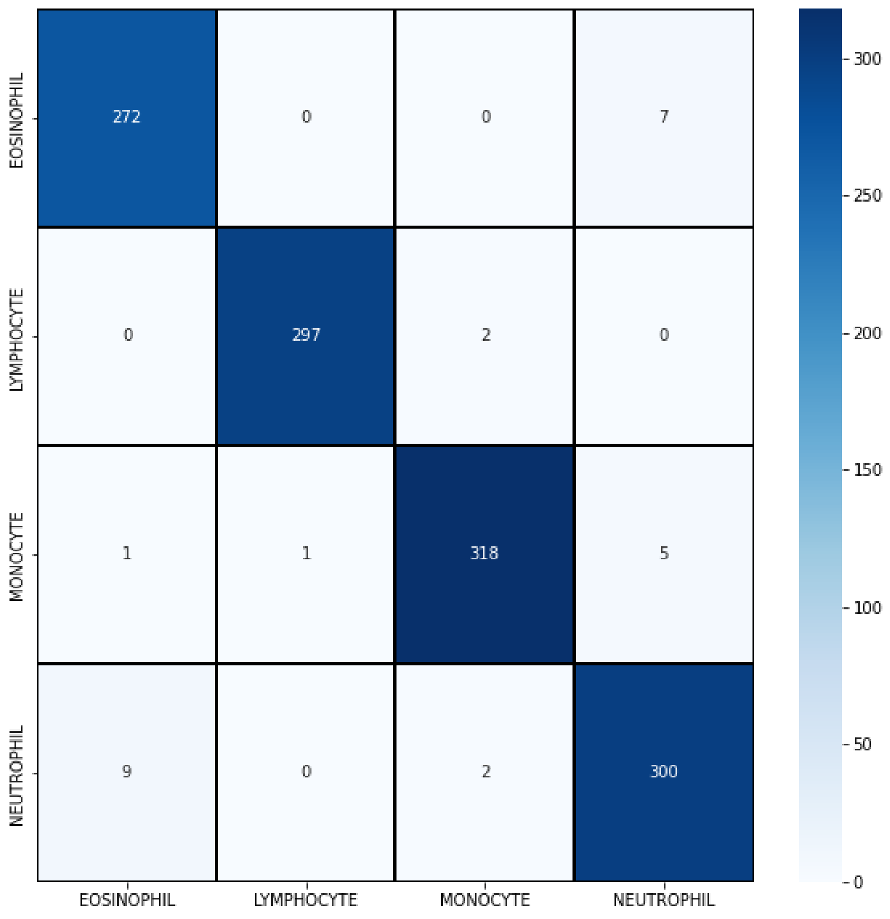


Figure 35: Confusion Matrix for Model 1

5.1.2 Evaluation of Model 2

The performance of Model 2 is not as good as that of Model 1. The learning curve of Model 2 is shown in Figure 36. From the figure, it is clear that the model suffers from the under-fitting problem as the validation loss is lower than the training loss. Moreover, both validation accuracy and loss fluctuate across the corresponding training accuracy and training loss. A probable reason for this problem is the architecture of the MobileNetV2. Originally, the MobileNotV2 was developed for an image resolution of 240 × 240 pixels. But our model is trained with 120 × 120 pixel images due to insufficient RAM of *Google Colaboratory*. 240 × 240 pixels resolution of input data occupies 75% of the RAM, for which training cannot be conducted. Some other platforms were also investigated but nothing suitable was found. However, the accuracy (89%) and the loss (0.3) of the model is quite attractive which encourages to keep the model.

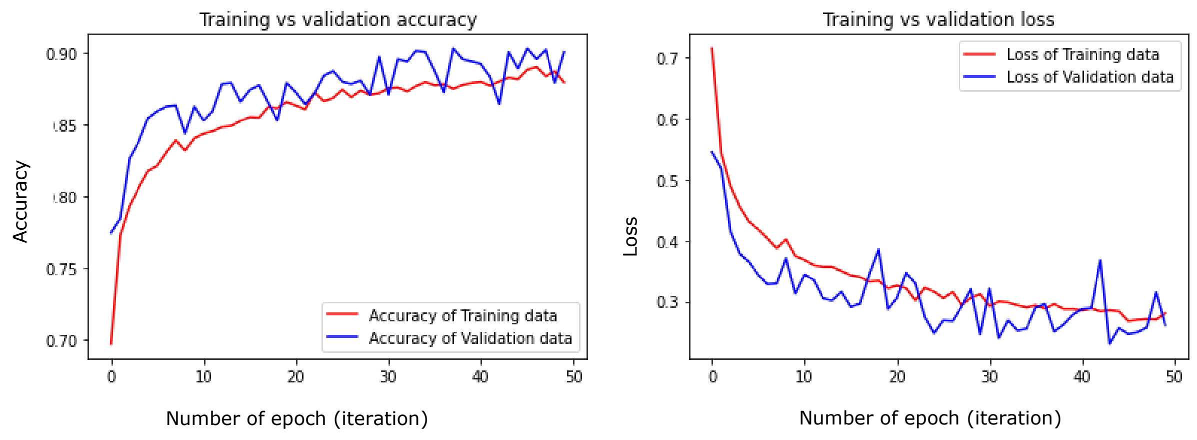


Figure 36: Learning curve of Model 2. The X axes represent the epochs ("training iterations") and the Y axes represent the accuracy (max value is 10and loss.

The confusion matrix of Model 2 is shown in Figure 37 which is quite acceptable. Below are the precision calculations.

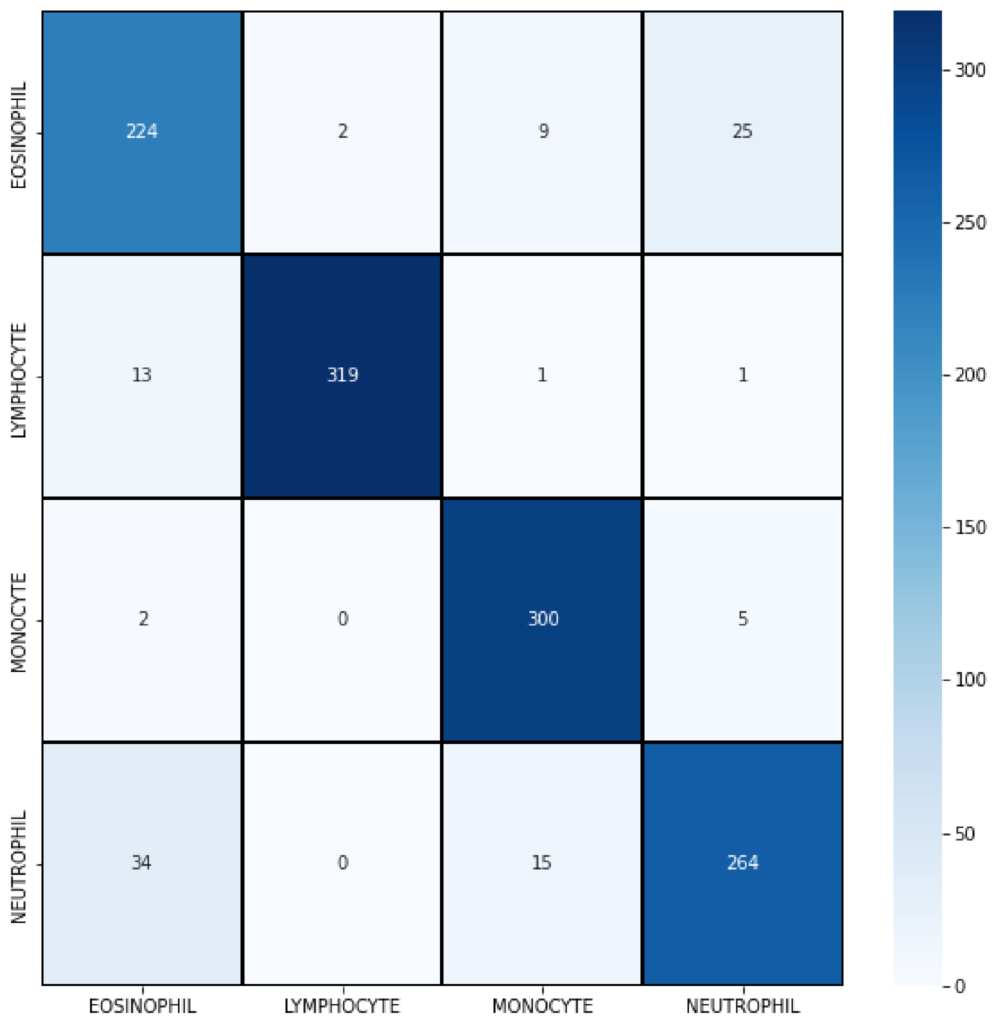


Figure 37: Confusion matrix of Model 2

Precision for Eosinophil $= \frac{224}{224+36} \times 100 = 86.15\%$

Precision for Lymphocyte $= \frac{319}{319+15} \times 100 = 95.50\%$

Precision for Monocyte $= \frac{300}{300+7} \times 100 = 97.71\%$

Precision for Neutrophil $= \frac{264}{264+49} \times 100 = 84.34\%$

Overall, model precision $= \frac{1107}{1107+107} \times 100 = 92.86\%$

The precision results confirm that Model 2 can be considered as acceptable, and thus we keep both Model 1 and Model 2 for implementation on the edge device (RPI4) and compared their performances.

5.2 Evaluation on Edge device

The edge device (RPI4) only performs the classification from the two trained models compared in Sections 5.1.1 and 5.1.2. For the RPI4 implementation, the model minimization presented in Section 4.6 was applied. Our actual goal for this MSc thesis work is to investigate the classification accuracy, classification time, the highest number of images detected by the RPI 4. For investigation, two different methodologies have been considered. In the first case, the image folder contains images of the same class. Three different datasets have been created with 50, 200, and 400 images. Then the average classification time and accuracy has been measured using Table 8. For the second case two types of dataset has been created with 400 (100 of each class) and 1600 (400 of each class) images with all four class and the classification accuracy of RPI is measured using Table 8. The result analysis for both of the methodology is discussed in the following subsections.

5.2.1 Performance evaluation of RPI in Methodology 1

Table 10 shows the performance matrix of RPI for Methodology 1.

Table 10: Performance of Model 1 and Model 2 for the same class of images in the folder (Methodology 1)

Class name	Number of Image	Model 1			Model 2		
		Average Classification Time (ms)	Detected cell types	Accuracy (%)	Average Classification Time (ms)	Detected cell types	Accuracy (%)
Eosinophil	50	51.4	Eosinophil: 48 Lymphocyte: 0 Monocyte: 1 Neutrophil: 1	96	137.1	Eosinophil: 44 Lymphocyte: 0 Monocyte: 2 Neutrophil: 4	88
	200	47.7	Eosinophil: 195 Lymphocyte: 0 Monocyte: 2 Neutrophil: 3	97.5	137.5	Eosinophil: 172 Lymphocyte: 0 Monocyte: 3 Neutrophil: 25	86
	400	48.8	Eosinophil: 393 Lymphocyte: 0 Monocyte: 2 Neutrophil: 5	98.25	137.3	Eosinophil: 333 Lymphocyte: 0 Monocyte: 10 Neutrophil: 57	83.25

Monocyte	50	46.5	Eosinophil: 0 Lymphocyte: 0 Monocyte: 50 Neutrophil: 0	100	126.0	Eosinophil: 1 Lymphocyte: 0 Monocyte: 48 Neutrophil: 1	96
	200	47.8	Eosinophil: 0 Lymphocyte: 0 Monocyte: 200 Neutrophil: 0	100	133.0	Eosinophil: 1 Lymphocyte: 0 Monocyte: 198 Neutrophil: 1	99
	400	48.1	Eosinophil: 0 Lymphocyte: 0 Monocyte: 400 Neutrophil: 0	100	187.8	Eosinophil: 2 Lymphocyte: 0 Monocyte: 395 Neutrophil: 3	98.75
Lymphocyte	50	48.2	Eosinophil: 1 Lymphocyte:49 Monocyte: 0 Neutrophil: 0	98	133.1	Eosinophil: 1 Lymphocyte: 49 Monocyte: 0 Neutrophil: 0	98
	200	48.3	Eosinophil: 1 Lymphocyte: 199 Monocyte: 0 Neutrophil: 0	99.5	141.5	Eosinophil: 2 Lymphocyte:198 Monocyte: 0 Neutrophil: 0	99
	400	47.6	Eosinophil: 4 Lymphocyte:396 Monocyte: 0 Neutrophil: 0	99	163.3	Eosinophil: 3 Lymphocyte: 393 Monocyte: 1 Neutrophil: 3	98.25
Neutrophil	50	46.9	Eosinophil: 1 Lymphocyte: 0 Monocyte: 0 Neutrophil: 49	98	130.4	Eosinophil: 9 Lymphocyte: 0 Monocyte: 2 Neutrophil: 39	78
	200	48.6	Eosinophil: 4 Lymphocyte: 0 Monocyte: 0 Neutrophil: 196	98	144.1	Eosinophil: 26 Lymphocyte: 0 Monocyte: 4 Neutrophil: 170	85
	400	48.2	Eosinophil: 7 Lymphocyte: 0 Monocyte: 0 Neutrophil: 393	98.25	133.4	Eosinophil: 40 Lymphocyte:1 Monocyte: 9 Neutrophil: 350	87.5
Average		48.2		98.54	142.04		91.3

From the measured data it is clear, like in Section 5.1, that Model 1 outperforms Model 2 in terms of accuracy and classification times. Model 2 struggles to differentiate between Eosinophil and Neutrophil for which the accuracy rate is below 90%. On average, Model 2 is 7.8% less accurate than Model 1.

Model 1 is almost three times faster than Model 2. A graphical comparison between Model 2 and Model 1 performances is shown in Figure 38 and Figure 39. The average classification time for monocyte is comparatively higher for the 3rd dataset.

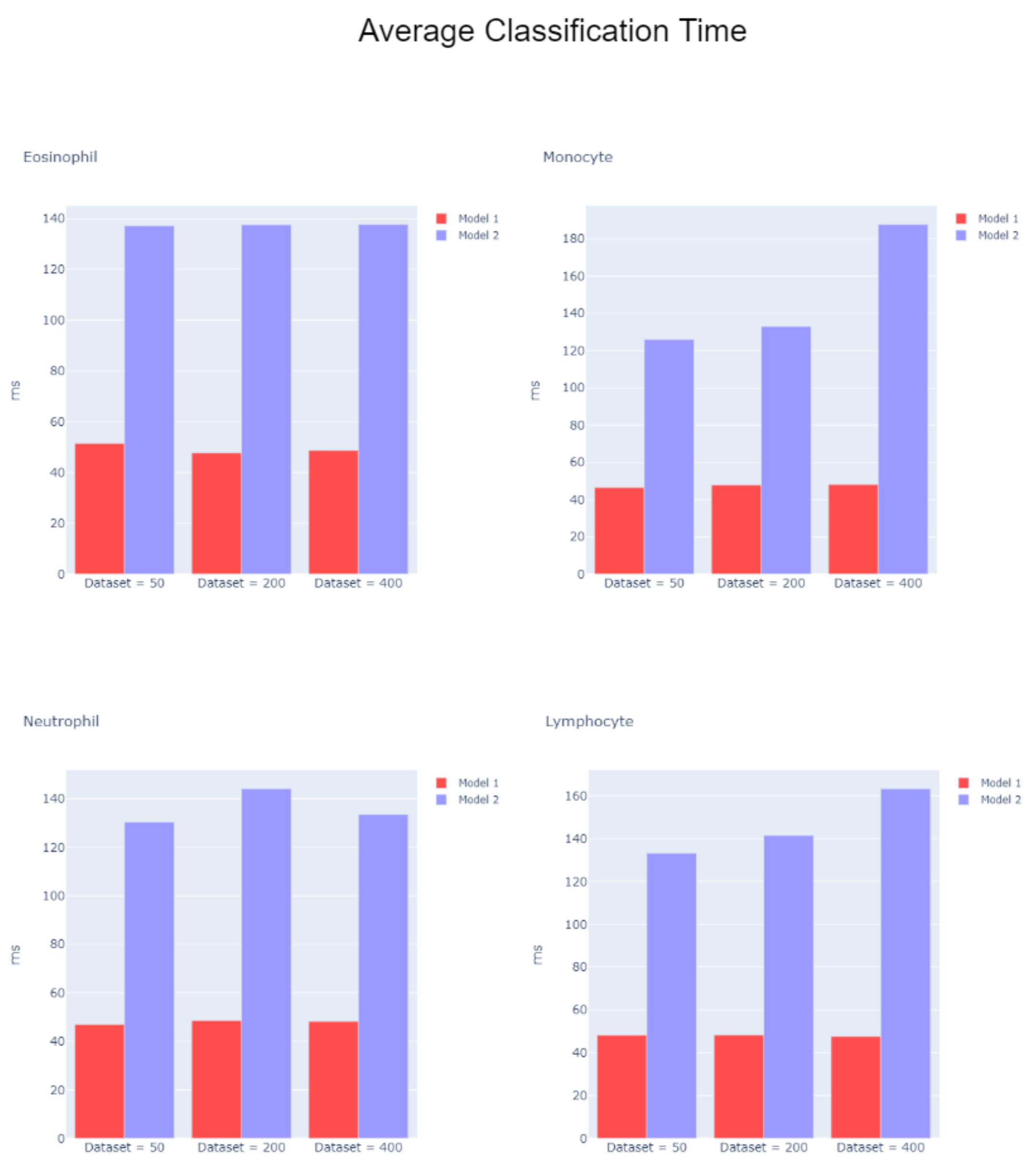


Figure 38: Average classification times for Model 1 and Model 2.

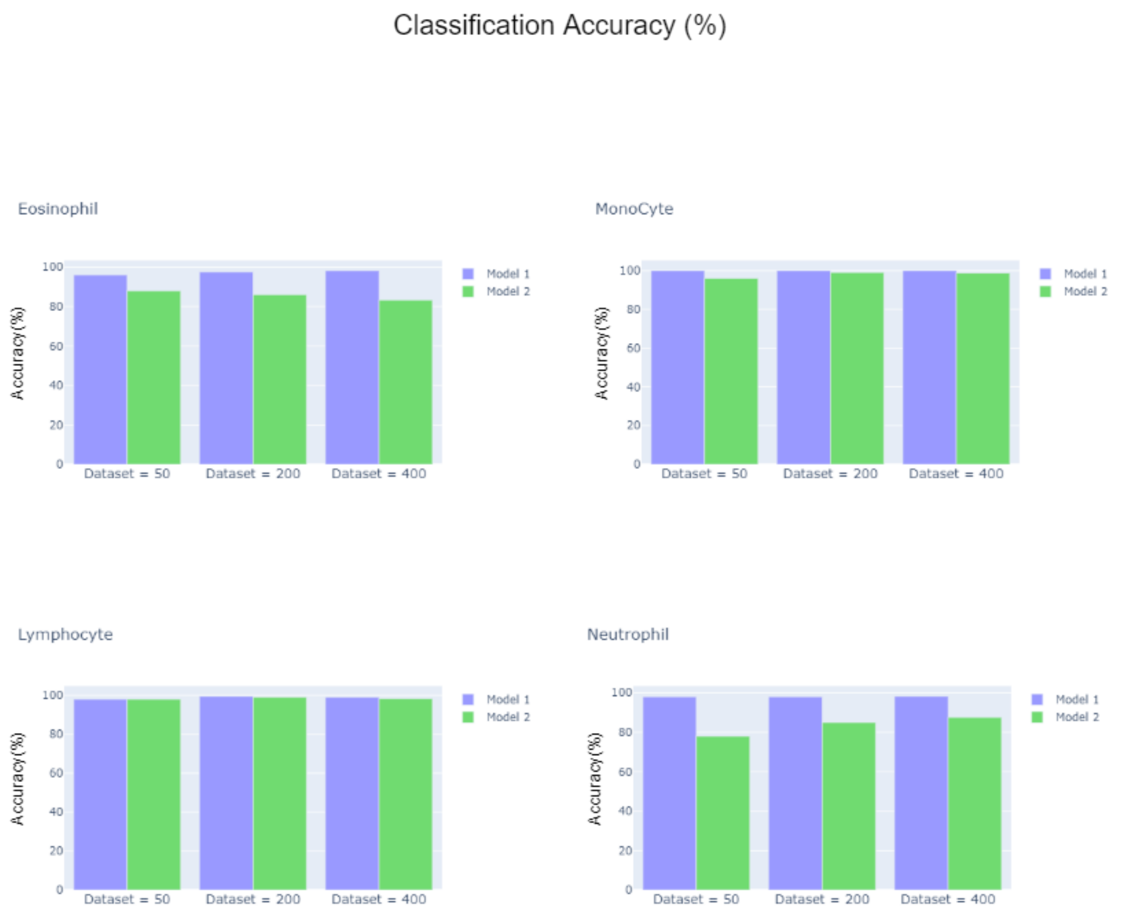


Figure 39: Accuracy comparison for Model 1 and Model 2 for same class.

All in all, for the first evaluation approach on the edge device, it can be concluded that Model 1, which was developed from scratch, performs better than Model 2 which was pre-trained and used transfer learning.

5.2.2 Performance evaluation of RPI in Methodology 2

In the second approach ("Methodology 2"), the image folder contains images of all four classes. The first dataset (Dataset 1) is prepared with 400 hundred images with 100 images for each class. The second dataset (Dataset 2) contains a total of 1600 images with 400 of each of the four classes. Table 11 shows the average classification time for two types of dataset. Similar to the previous case, classification with Model 1 is almost three times faster than with Model 2. However, compared to Methodology 1, the average classification time for Model 1 is slightly higher than for Methodology 2 which is due to higher number of images. In contrast, classification time for Model 2 is almost 4% lower for the second approach as compared to the first one. That is because classification time for lymphocyte is much higher for Model 2. In the first approach, as the model only has to classify one type of image, that increases the average classification time.

Table 11: Performance of Model 1 and Model 2 for different class image in folder (Methodology 2)

Total Number of Image	Number of image of each class	Model 1			Model 2		
		Total Classification time	Average Classification Time (ms)	Detection	Total Classification time	Average Classification Time (ms)	Detection
400	100	20.22	50.55	Eosinophil : 98 Lymphocyte:99 Monocyte: 100 Neutrophil: 103	55.227	138.068	Eosinophil : 103 Lymphocyte:100 Monocyte: 101 Neutrophil: 96
1600	400	76.9	48.0625	Eosinophil : 403 Lymphocyte:394 Monocyte: 402 Neutrophil: 398	214400	134	Eosinophil : 376 Lymphocyte:393 Monocyte: 420 Neutrophil: 408
Average			49.30625			136.034	

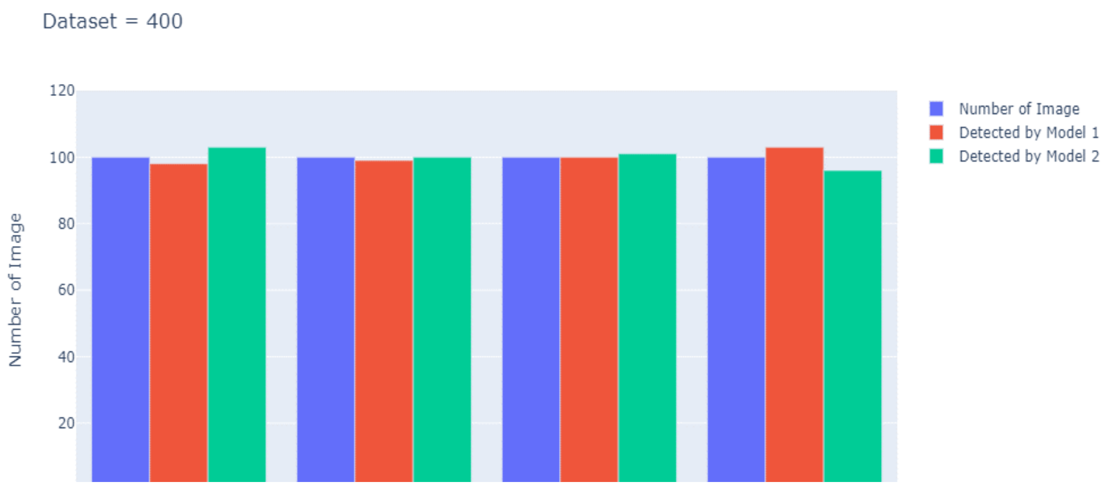


Figure 40: Number of detections of each class for the Dataset = 400 images.

Figure 40 and Figure 41 showing the number of detections for each class for 400 and 1600 dataset with mixed class image. The detection of Model 1 is satisfactory while Model 2 suffers to differentiate between eosinophil and monocyte for dataset 2. Out of 400 images of both monocyte and eosinophil, Model 2 detects 376 eosinophil and 420 monocyte which is approximately 6% error. One of the major reasons is both of these blood cells are similar in shape. Moreover, they have been marked with same blue colour (see Figure 25) which makes them difficult to distinguish. However, the error rate is in the acceptable range.

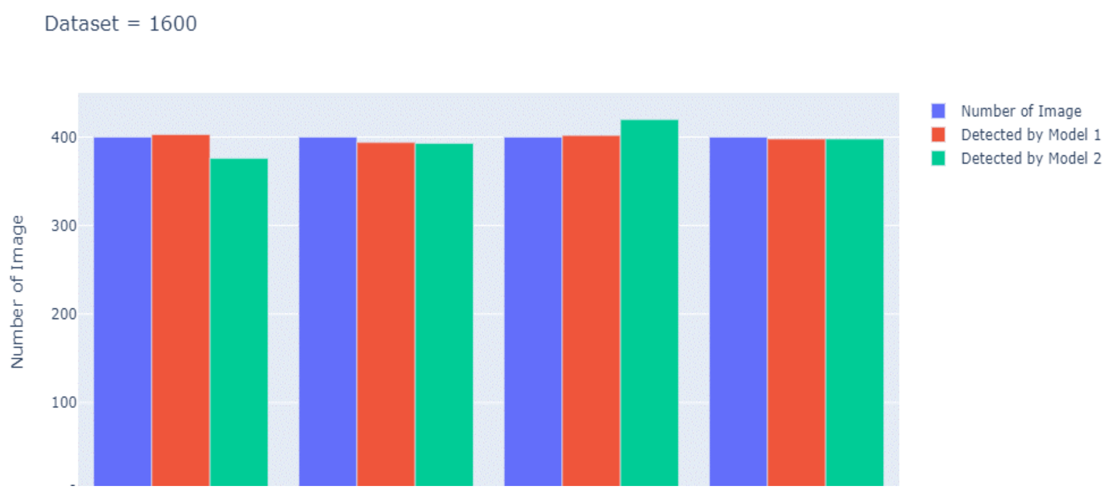


Figure 41: Number of detections for each class for the Dataset = 1600 images.

Average classification time for a single image is presented in Figure 42. Here the Model 1 classification time is almost three time faster than Model 2.

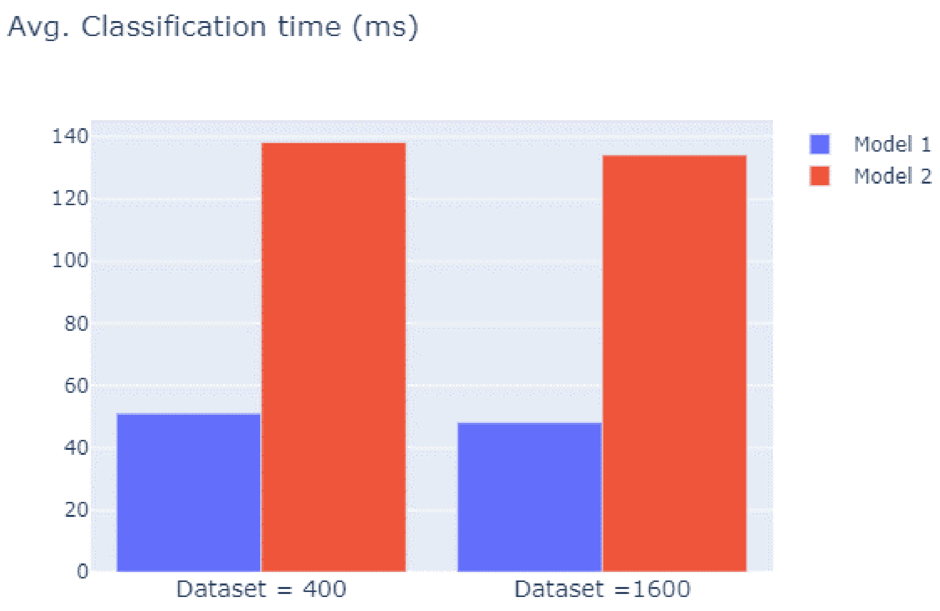


Figure 42: Comparison of average classification time of a single image between model 1 and model 2

Overall, the accuracy of Model 1 for both of the methodology is highly attractive with almost 99% of accuracy and 48 ms of classification time while Model 2 accuracy rate (91%) is not as good as Model 1 but still in acceptable range. Also Model 1 is three times faster than Model 1. Model 1 outperforms Model 2, as it is built from the scratch with careful parameters selection by trial-and-error which requires time and effort. On the other hand, transfer learning methodology is implemented in Model 2 in which a pre-trained model is used for training with minimum effort. Thus, there is a trade-off between model development time/effort and classification performance.

The next chapter concludes this MSc thesis by summarizing the work and presenting a few suggestions for future work.

6 Conclusion

6.1 Summary

The focus of this thesis work was to explore the application of DL methodology for microscopic medical images, in particular, four types of white blood cell classification using an SBC. The initial goal was to train a DL model and deploy it into SBC to evaluate its suitability for image classification. Moreover, the thesis work also investigated alternative training techniques to model training to deal with data scarcity of medical images. The obtained results throughout this work are promising.

The performance of the model (Model 1) developed from scratch exhibits competitive performance both in the cloud (Google Colaboratory, training and inference) and on the edge (RPI4, inference only) platforms. In the cloud, the overall classification precision is almost 98%. The lowest precision is 96.46% for neutrophils and the highest is 99.33% for Lymphocytes, which is very encouraging considering the current state-of-the-art. The developed model performs well in the SBC too. The average classification accuracy on the RPI4is 98.5%. The lowest accuracy is 96% for 50 images of a a single class (Eosinophil) which is due to the smaller dataset size. The average classification time is 48.2 ms which is also good because a simple ARM quad-core processor is used without the usage of a dedicated accelerator (GPU or TPU).

The performance of Model 2 is not as impressive as that of Model 1. In the cloud, the overall precision of the model is 92.86%. However, the precision for Eosinophil and Neutrophil is 86% and 84%, respectively. This indicates that this model struggles to differentiate between these two blood cell types. One of the reasons can be the similar-look of these two blood cells types. Moreover, their marker is also of the same color. For the different colors of the annotation marker, the performance could have been better. but it is a matter of investigation. On the RPI4, Model 2's performance is also worse as compared to Model 1. Although the average accuracy is 91%, it decreases to 78% for Neutrophil classification. The classification time is also three times higher than for Model

1 with an average of 142 ms to classify a single image. However, considering the state-of-the-art and SBC configuration the result is still considered as up to the mark.

Comparing the performance of these two models, we can conclude that Model 1 (which was developed from scratch) is superior to Model 2 (based on transfer learning). This indicates that the DL model must be specifically designed for the medical image type. However, transfer learning is also a suitable approach in case of data scarcity or time-constrained situations. In the case of RPI4 performance, the result clearly show that competitive accuracy can be achieved although the models are converted to be smaller so that they can fit in the limited memory space. At the same time, real-time image classification cannot be achieved. However, a plain RPI4 can be a good alternative when instantaneous classification is not required.

6.2 Future work

A plain RPI4 was selected due to its simplicity and good number of application examples. If (near) real-time performance is required, then an SBC featuring a SoC with a dedicated hardware accelerator (e.g. GPU, TPU) should be considered. Using such a device is expected to result in better performance for classification time. Thus, one of the primary investigations in future work would be to evaluate the two models developed in this thesis onto GPU/TPU based SBCs to evaluate their suitability for real-time detection and classification.

Another issue to consider for the transfer learning approach is that different pre-trained models can used to check the performance. Finally, these models could be optimized to develop a portable blood cell classification system and integrate it into a medical device.

References

[1] J. Wang, H. Zhu, H. Wang, Y. D. Zhang "A Review of Deep Learning on Medical Image Analysis" *Mobile Netw Appl*, vol. 26, pp. 351–380, Nov 2020.

[2] M. Biswas, V. Kuppili, L. Saba, D. R. Edla, et al., "State-of-the-art review on deep learning in medical imaging", *Frontiers in Bioscience-Landmark*, vol. 24(3), pp. 392-426, Jan. 2019.

[3] Z. Liu, L. Jin, J. Chen, Q. Fang, S. Ablameyko, et al., "A survey on applications of deep learning in microscopy image analysis", *Computers in Biology and Medicine*, vol. 134, pp. 104523, Jul 2021.

[4] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *in Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

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

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks*," Commun. ACM,* vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[7] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[8] C. Szegedy et al., "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[9] A. G. Howard et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” *arXiv:1704.04861 [cs],* Apr. 2017, Accessed: Dec. 27, 2020. [Online], doi: http://arxiv.org/abs/1704.04861.

[10] L. Cai, et al. “A review of the application of deep learning in medical image classification and segmentation.” *Annals of translational medicine,* vol. 8(11), pp.713, Jun 2020, doi:10.21037/atm.2020.02.44

[11] F. Pesapane, M. Codari, and F. Sardanelli, “Artificial intelligence in medical imaging: threat or opportunity? Radiologists again at the forefront of innovation in medicine” *Eur Radiol Exp 2,* 35, Oct. 2018. https://doi.org/10.1186/s41747-018-0061-6

[12] S. Suganyadevi, V. Seethalakshmi & K. Balasamy, "A review on deep learning in medical image analysis," *Int J Multimed Info Retr,* September 2021.

[13] S. K. Zhou, H. Greenspan, C. Davatzikos, J. S. Duncan, et al. "A Review of Deep Learning in Medical Imaging: Imaging Traits, Technology Trends, Case Studies With Progress Highlights, and Future Promises," *in Proceedings of the IEEE*, vol. 109, no. 5, pp. 820-838, May 2021.

[14] E. Shelhamer, J. Long and T. Darrell, "Fully convolutional networks for semantic segmentation", *IEEE Trans. Pattern Anal Mach. Intell.*, vol. 39, no. 4, pp. 640-651, Apr. 2017/

[15] S. Dong et al., "3D left ventricle segmentation on echocardiography with atlas guided generation and voxel-to-voxel discrimination", *Proc. Int. Conf. Med. Image Comput. Comput. Assist. Intervent. (MICCAI)*, 2018, pp. 622-629.

[16] P. Vu Tran, "A fully convolutional neural network for cardiac segmentation in short-axis MRI", *arXiv:1604.00494,* 2016, [online] Available: http://arxiv.org/abs/1604.00494.

[17] E. Ferdian et al., "Fully automated myocardial strain estimation from cardiovascular MRI–tagged images using a deep learning framework in the UK biobank", *Radiol. Cardiothoracic Imag.*, vol. 2, no. 1, Feb. 2020.

[18] S. Liu, S. Liu, W. Cai, S. Pujol, R. Kikinis and D. Feng, "Early diagnosis of Alzheimer’s disease with deep learning", *Proc. Int. Symp. Biomed. Imag. (ISBI)*, pp. 1015-1018, 2014.

[19] M. Havaei et al., "Brain tumor segmentation with deep neural networks", *Med. Image Anal*, vol. 35, pp. 18-31, Jan. 2017.

[20] F. Ciompi et al., "Towards automatic pulmonary nodule management in lung cancer screening with deep learning*", Sci. Rep*., vol. 7, no. 1, pp. 46479, Apr. 2017.

[21] D. Ardila et al., "End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography", *Nature Med.,* vol. 25, no. 6, pp. 954-961, Jun. 2019.

[22] K. Murphy et al., "Computer aided detection of tuberculosis on chest radiographs: An evaluation of the CAD4TB v6 system", *Sci. Rep.*, vol. 10, no. 1, pp. 1-11, 2020, [online] Available: https://arxiv.org/abs/1903.03349.

[23] B. Q. Huynh; H. Li; M. L. Giger, "Digital mammographic tumor classification using transfer learning from deep convolutional neural networks", *Journal of medical imaging (Bellingham, Wash.)*, vol. 3, pp. 34501, Aug. 2016.

[24] M. Salvi, U. R. Acharya, F. Molinari, et al., "The impact of pre- and post-image processing techniques on deep learning frameworks: A comprehensive review for digital pathology image analysis", *Computers in Biology and Medicine*, vol. 128, Jan. 2021. Available: https://doi.org/10.1016/j.compbiomed.2020.104129

[25] C. Chen, A. Mahjoubfar, LC. Tai, et al. "Deep Learning in Label-free Cell Classification", *Sci Rep*, vol. 6, pp. 21471 Jan. 2016. Available: https://doi.org/10.1038/srep21471

[26] F. Qin, N. Gao, Y. Peng, Z. Wu, S. Shen, A. Grudtsin, "Fine-grained leukocyte classification with deep residual learning for microscopic images", *Computer Methods and Programs in Biomedicine*, vol. 162, pp. 243-252, Aug. 2018.

[27] A.I. Shahin, Y. Guo, K.M. Amin, A. A. Sharawi, "White blood cells identification system based on convolutional deep neural learning networks", *Computer Methods and Programs in Biomedicine*, vol. 168, pp. 68-80, Jan. 2019. Available: https://doi.org/10.1016/j.cmpb.2017.11.015

[28] L. Alzubaidi, M. A. Fadhel, O. Al-Shamma, J. Zhang, and Y. Duan, "Deep Learning Models for Classification of Red Blood Cells in Microscopy Images to Aid in Sickle Cell Anemia Diagnosis" *Electronics 9*, vol. 3, pp. 427, March 2020. Available: https://doi.org/10.3390/electronics9030427

[29] J. Hung and A. Carpenter, "Applying Faster R-CNN for Object Detection on Malaria Images," *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 808-813, doi: 10.1109/CVPRW.2017.112

[30] W. Deelder, E.D. Benavente, J. Phelan, E. Manko, S. Campino, L. Palla, T.G. Clark, "Using deep learning to identify recent positive selection in malaria parasite sequence data", *Malaria Journal*, vol. 20, pp. 270, Jun. 2021. doi: 10.1186/s12936-021-03788-x.

[31] H. S. Ryu, M.-S. Jin, J. H. Park, S. Lee, J. Cho, S. Oh, T.-Y. Kwak, J. I. Woo, Y. Mun, S. W. Kim, S. Hwang, S.-J. Shin, and H. Chang, "Automated Gleason Scoring and Tumor Quantification in Prostate Core Needle Biopsy Images Using Deep Neural Networks and Its Comparison with Pathologist-Based Assessment," *Cancers*, vol. 11, no. 12, p. 1860, Nov. 2019.

[32] H. Le, R. Gupta, L. Hou, S. Abousamra, D. Fassler, et al., "Utilizing Automated Breast Cancer Detection to Identify Spatial Distributions of Tumor-Infiltrating Lymphocytes in Invasive Breast Cancer", *The American Journal of Pathology*, vol. 190, pp. 1491-1504, Jul. 2020. DOI: https://doi.org/10.1016/j.ajpath.2020.03.012.

[33] J. W. Wei, L. J. Tafe, Y. A. Linnik, et al. "Pathologist-level classification of histologic patterns on resected lung adenocarcinoma slides with deep neural networks", *Scintific Report*, vol. 9, article no. 3358, 2019.

[34] B. Korbar, et al. “Deep Learning for Classification of Colorectal Polyps on Whole-slide Images.” *Journal of pathology informatics*, vol. 8 30. 25 Jul. 2017, doi:10.4103/jpi.jpi_34_17

[35] T. Mahmood, M. Arsalan, M. Owais, M. B. Lee, and K. R. Park, “Artificial Intelligence-Based Mitosis Detection in Breast Cancer Histopathology Images Using Faster R-CNN and Deep CNNs,” *Journal of Clinical Medicine*, vol. 9, no. 3, p. 749, Mar. 2020.

[36] H. Li, F. Pang, Y. Shi and Z. Liu, "Cell dynamic morphology classification using deep convolutional neural networks", *Cytometry*, vol. 93, pp. 628-638, May, 2018 https://doi.org/10.1002/cyto.a.23490

[37] A. Gupta, P. J. Harrison, H. Wieslander, N. Pielawski, K. Kartasalo, G. Partel, L. Solorzano, A. Suveer, A.H. Klemm, O. Spjuth, I. M. Sintorn, and C. Wählby, "Deep Learning in Image Cytometry: A Review", *Cytometry*, vol. 95, pp. 366-380. Jan. 2019. DOI: https://doi.org/10.1002/cyto.a.23701

[38] Y. Gu, A. Chen, X. Zhang, C. Fan, K. Li, and J. Shen, “Deep Learning based Cell Classification in Imaging Flow Cytometer”, *ASP Trans. Pattern Recognit. Intell. Syst.*, vol. 1, no. 2, pp. 18–27, Jun. 2021.

[39] M. Kräter, S/ Abuhattum, D. Soteriou, A. Jacobi, T. Krüger, J. Guck, M. Herbig, “AIDeveloper: Deep Learning Image Classification in Life Science and Beyond”, *Adv. Sci.*, vol. 8, pp. 2003743, Mar. 2021 https://doi.org/10.1002/advs.202003743

[40] J. -H. Chien, S. Chan, S. Cheng and Y. -C. Ouyang, "Identification and Detection of Immature White Blood Cells through Deep Learning," *IEEE 3rd Global Conference on Life Sciences and Technologies (LifeTech)*, 2021, pp. 1-3, doi: 10.1109/LifeTech52111.2021.9391955.

[41] M. Lippeveld, C. Knill, E. Ladlow, et al. "Classification of Human White Blood Cells Using Machine Learning for Stain-Free Imaging Flow Cytometry", *Cytometry A.*, vol. 97, pp. 308-319, MAR. 2020.

[42] S. Khan, N. Islam, Z. Jan, I. U. Din, J. J. P. C Rodrigues, "A novel deep learning based framework for the detection and classification of breast cancer using transfer learning", *Pattern Recognition Letters*, vol. 125, pp. 1-6, Jul. 2019.

[43] K. K. Jha, H. S. Dutta, "Mutual Information based hybrid model and deep learning for Acute Lymphocytic Leukemia detection in single cell blood smear images", *Computer Methods and Programs in Biomedicine*, vol. 179, pp. 104987, Oct. 2019.

[44] H. Lei, S. Liu, A. Elazab, X. Gong and B. Lei, "Attention-Guided Multi-Branch Convolutional Neural Network for Mitosis Detection From Histopathological Images," in IEEE Journal of Biomedical and Health Informatics, vol. 25, no. 2, pp. 358-370, Feb. 2021, doi: 10.1109/JBHI.2020.3027566.

[45] X. Wang, H. Chen, C. Gan, et al., "Weakly Supervised Deep Learning for Whole Slide Lung Cancer Image Analysis," *in IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3950-3962, Sept. 2020, doi: 10.1109/TCYB.2019.2935141.

[46] D. Tellez, M. Balkenhol, I. Otte-Holler, et al., "Whole-Slide Mitosis Detection in H&E Breast Histology Using PHH3 as a Reference to Train Distilled Stain-Invariant Convolutional Networks," *in IEEE Transactions on Medical Imaging*, vol. 37, no. 9, pp. 2126-2136, Sept. 2018, doi: 10.1109/TMI.2018.2820199.

[47] H. Niioka, S. Asatani, A. Yoshimura, et al. "Classification of C2C12 cells at differentiation by convolutional neural network of deep learning using phase contrast images," *Human Cell*, vol. 31, pp. 87–93, Oct 2018. https://doi.org/10.1007/s13577-017-0191-9

[48] T. J. S. Durant, E. M. Olson, W. L. Schulz, R. Torres, "Very Deep Convolutional Neural Networks for Morphologic Classification of Erythrocytes," *Clinical Chemistry*, vol. 63, pp. 1847–1855, Dec. 2017. https://doi.org/10.1373/clinchem.2017.276345

[49] L. Zhang, Le Lu, I. Nogues, R. M. Summers, S. Liu and J. Yao, "DeepPap: Deep Convolutional Networks for Cervical Cell Classification," *in IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 6, pp. 1633-1643, Nov. 2017, doi: 10.1109/JBHI.2017.2705583.

[50] C. Jen Ngeh, C. Ma, T. Kuan-Wei Ho, Y. Wang and J. Raiti, "Deep Learning on Edge Device for Early Prescreening of Skin Cancers in Rural Communities," *IEEE Global Humanitarian Technology Conference (GHTC)*, 2020, pp. 1-4, doi: 10.1109/GHTC46280.2020.9342911.

[51] R. Vidhya Lavanya, S. EP, C. Jayakumari and R. Isaac, "Detection and Classification of Diabetic Retinopathy using Raspberry PI," *4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2020, pp. 1688-1691, doi: 10.1109/ICECA49313.2020.9297408.

[52] A. Abid, P. Sinha, A. Harpale, J. Gichoya and S. Purkayastha, "Optimizing Medical Image Classification Models for Edge Devices" *in Distributed Computing and Artificial Intelligence, Volume 1: 18th International Conference, Lecture Notes in Networks and Systems*, vol 327, K. Matsui, S. Omatu, T. Yigitcanlar, S.R. González, Eds. Springer, 2021, pp. 77-87.

[53] P. Krömer, J. Nowaková, "Medical Image Analysis with NVIDIA Jetson GPU Modules", *in: Advances in Intelligent Networking and Collaborative Systems. INCoS 2021. Lecture Notes in Networks and Systems*, vol. 312, L. Barolli, H.C. Chen, H. Miwa, Eds. Springer, 2022, pp. 233-242

[54] J. Civit-Masot, F. Luna-Perejón, J. M. Rodríguez Corral, M. Domínguez-Morales, A. Morgado-Estévez, A. Civit, "A study on the use of Edge TPUs for eye fundus image segmentation", *Engineering Applications of Artificial Intelligence*, vol. 104, pp. 104384, Sep. 2021.

[55] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. S. Awwal, and V. K. Asari, “A State-of-the-Art Survey on Deep Learning Theory and Architectures,” *Electronics*, vol. 8, no. 3, p. 292, Mar. 2019.

[56] S. Ota, I. Sato and R. Horisaki, “Implementing machine learning methods for imaging flow cytometry”, *Microscopy*, Volume 69, Issue 2, Pages 61–68, Apr. 2020.

[57] Y. LeCun, Y. Bengio, & G. Hinton, “Deep learning” *Nature,* vol. 521, pp. 436–444, May 2015. https://doi.org/10.1038/nature14539

[58] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, et al. “Human-level control through deep reinforcement learning”, *Nature,* vol. 518, pp. 529–533, Feb. 2015.

[59] M. Lapan, “What Is Reinforcement Learning?” *Deep Reinforcement Learning Hands-On*. Birmingham, UK: Packt Publishing, 2018, pp. 6.

[60] A. S. Lundervold, A. Lundervold, "An overview of deep learning in medical imaging focusing on MRI", *Zeitschrift für Medizinische Physik*, vol. 29, Issue 2, pp. 102-127, May 2019.

[61] D. Sarkar, R. Bali, & T. Ghosh, “Neural network basics”, *Hands-On Transfer Learning with Python: Implement Advanced Deep Learning and Neural Network Models Using TensorFlow and Keras*. Birmingham: Packt Publishing Ltd., 2018, pp. 138 – 201.

[62] E. Alpaydin,” Linear Discrimination”, *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004, pp. 187-188.

[63] Bottou L. (2012) Stochastic Gradient Descent Tricks. In: Montavon G., Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35289-8_25

[64] K. Fukushima, “Neocognitron: A hierarchical neural network capable of visual pattern recognition”. *Neural Netw*. Vol 1. pp. 119–130, Mar. 1988.

[65] T. Watanabe and H. Iima, “Nonlinear Optimization Method Based on Stochastic Gradient Descent for Fast Convergence,” *in 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC),* Oct. 2018, pp. 4198–4203, doi:10.1109/SMC.2018.00711

[66] L. Alzubaidi, et al. "Novel Transfer Learning Approach for Medical Imaging with Limited Labeled Data." *Cancers,* vol. 13(7), pp. 1590, Mar. 2021.

[67] AISmartz, Nov 25, 2019, “https://www.aismartz.com/blog/an-introduction-to-transfer-learning/”

[68] L. Torrey, & J. Shavlik, “Transfer Learning”, in *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques,* Eds. IGI Global, 2010, pp 242-264. http://doi:10.4018/978-1-60566-766-9.ch011

[69] S.J. Pan, and Y. Qiang, "A survey on transfer learning." *IEEE Transactions on knowledge and data engineering,* vol. 22(10), pp. 1345-1359, Oct. 2009.

[70] K. R. Weiss, and T. M. Khoshgoftaar. "An investigation of transfer learning and traditional machine learning algorithms." *IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI).* IEEE, 2016.

[71] J. Nam, S.J. Pan, S. Kim, “Transfer defect learning”, *Proceedings of the 2013 International Conference on Software Engineering, IEEE Press (2013),* pp. 382-391

[72] M. Sandler, et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” *IEEE/CVF Conference on Computer Vision and Pattern Recognition,* 2018, pp. 4510-4520.

[73] P. Mooney,” Blood Cell Images: 12,500 images: 4 different cell types”,*Kaggle*, 2017. Available: https://www.kaggle.com/paultimothymooney/blood-cells

[74] M. Kamal, “CNN Train(0.99)- Val(0.98) - Test(0.986)”, *Kaggle*, 2020. Available: https://www.kaggle.com/mohamedkamal77/cnn-train-0-99-val-0-98-test-0-986

Appendix 1 – Python source code for Libraries and Framework Selection

This appendix and the following one provide the essential Python source code that has been used in this MSc thesis work.

```python
import numpy as np
import pandas as pd
from scipy.spatial import distance as dist
import matplotlib.pyplot as plt
import os
import cv2
import seaborn as sns
from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn import decomposition
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import tensorflow as tf
import keras
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.models import Sequential, Model
from keras.initializers import he_normal
from keras.layers import Lambda, SeparableConv2D,
BatchNormalization, Dropout, MaxPooling2D, Input, Dense,
Conv2D, Activation, Flatten
from keras.callbacks import EarlyStopping, ReduceLROnPlateau,
ModelCheckpoint
import imutils
```

Appendix 2 – Python Source Code for Image Pre-Processing

1. Function for locating desired object

```python
def findEdges(image):
    # find edges in image
    gray = cv2.GaussianBlur(image, (1, 1), 0)
    edged = cv2.Canny(gray, 100, 400)
    edged = cv2.dilate(edged, None, iterations=1)
    edged = cv2.erode(edged, None, iterations=1)
    return edged

def getImgContours(edged):
    # find contours in the edge map
    contours = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)
    contours = sorted(contours, key=lambda x: cv2.contourArea(x))
    return contours

def getBoxes(contours, orig):
    # get the boxes
    boxes = []
    centers = []
    for contour in contours:
        box = cv2.minAreaRect(contour)
        box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box)
        box = np.array(box, dtype="int")
        (tl, tr, br, bl) = box
        if (dist.euclidean(tl, bl)) > 0 and (dist.euclidean(tl, tr)) > 0:
            boxes.append(box)
    return boxes
```

2. Image Pre-processing to get Image data

```
# Open
image = cv2.imread(img_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# add padding to the image to better detect cell at the edge
image =
cv2.copyMakeBorder(image,10,10,10,10,cv2.BORDER_CONSTANT,value=[198, 203,
208])

#thresholding the image to get the target cell
image1 = cv2.inRange(image,(80, 80, 180),(180, 170, 245))

# openning errosion then dilation
kernel = np.ones((3, 3), np.uint8)
kernel1 = np.ones((5, 5), np.uint8)
img_erosion = cv2.erode(image1, kernel, iterations=2)
image1 = cv2.dilate(img_erosion, kernel1, iterations=5)


#detecting the blood cell
edgedImage = findEdges(image1)
edgedContours = getImgContours(edgedImage)
edgedBoxes =  getBoxes(edgedContours, image.copy())

      if len(edgedBoxes)==0:
        count +=1
        continue

# draw the contour and fill it
mask = np.zeros_like(image)
cv2.drawContours(mask, edgedContours, len(edgedContours)-1, (255,255,255),
-1)


# all pixel outside inside the contour is zero
image[mask==0] = 0

# extract blood cell
image = image[min_y:max_y, min_x:max_x]

if (np.size(image)==0):
   count +=1
   continue
# resize image
image = cv2.resize(image, image_size)
```

Appendix 3 – Python Source Code for Convolutional Block Design

```
# First Conv block
model1.add(Conv2D(16 , (3,3) , padding = 'same' , activation = 'relu' ,
input_shape = (120,120,3)))
model1.add(Conv2D(16 , (3,3), padding = 'same' , activation = 'relu'))
model1.add(MaxPooling2D(pool_size = (2,2)))

# Second Conv block
model1.add(SeparableConv2D(32, (3,3), activation = 'relu', padding = 'same'))
model1.add(SeparableConv2D(32, (3,3), activation = 'relu', padding = 'same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size = (2,2)))

# Third Conv block
model1.add(SeparableConv2D(64, (3,3), activation = 'relu', padding = 'same'))
model1.add(SeparableConv2D(64, (3,3), activation = 'relu', padding = 'same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size = (2,2)))

# Forth Conv block
model1.add(SeparableConv2D(128, (3,3), activation = 'relu', padding = 'same'))
model1.add(SeparableConv2D(128, (3,3), activation = 'relu', padding = 'same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size = (2,2)))
model1.add(Dropout(0.2))

# Fifth Conv block
model1.add(SeparableConv2D(256, (3,3), activation = 'relu', padding = 'same'))
model1.add(SeparableConv2D(256, (3,3), activation = 'relu', padding = 'same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size = (2,2)))
model1.add(Dropout(0.2))


# FC layer
model1.add(Flatten())
model1.add(Dense(units = 512 , activation = 'tanh'))
model1.add(Dropout(0.7))
model1.add(Dense(units = 128 , activation = 'tanh'))
model1.add(Dropout(0.5))
model1.add(Dense(units = 64 , activation = 'tanh'))
model1.add(Dropout(0.3))

# Output layer
model1.add(Dense(units = 4 , activation = 'softmax'))

# Compile
model1.compile(optimizer = "adam" , loss = 'sparse_categorical_crossentropy' ,
metrics = ['accuracy'])
```

Appendix 4 – Python Source Code for Transfer Learning Model Design

```
#Download pre-trained model

datagen =
ImageDataGenerator(horizontal_flip=True,vertical_flip=True,rotation_
range=20,zoom_range=0.2,

width_shift_range=0.2,height_shift_range=0.2,shear_range=0.1,fill_mo
de="nearest")

pretrained_model =
tf.keras.applications.MobileNetV2(input_shape=(120,120,3),include_to
p=False,weights='imagenet',pooling='avg')
pretrained_model.trainable = False

#defining input and output of the model

inputs = pretrained_model.input
x = tf.keras.layers.Dense(128,
activation='relu')(pretrained_model.output)
outputs = tf.keras.layers.Dense(4, activation='softmax')(x)
model = tf.keras.Model(inputs=inputs, outputs=outputs)


#Model compile

model.compile(
    optimizer = 'adam' ,
    loss = 'sparse_categorical_crossentropy' ,
    metrics = ['accuracy']
)
print(model.summary())


#Train the model
history =
model.fit(datagen.flow(train_images,train_labels,batch_size=32),
              validation_data=(val_images,val_labels), epochs=50)

get_acc = history.history['accuracy']
value_acc = history.history['val_accuracy']
get_loss = history.history['loss']
validation_loss = history.history['val_loss']
```

Appendix 5 – Python SourceCode for Inference on SBC

```python
for img in imageList:
  class_labels = ['EOSINOPHIL', 'LYMPHOCYTE', 'MONOCYTE', 'NEUTROPHIL']
  nb_classes = len(class_labels)
  image_size = (120,120)

  #Preprocessing of input image

  image = img
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  image = cv2.copyMakeBorder(image,10,10,10,10,cv2.BORDER_CONSTANT,value=[198, 203, 
208])
  image1 = cv2.inRange(image,(80, 80, 180),(180, 170, 245))

  kernel = np.ones((3, 3), np.uint8)
  kernel1 = np.ones((5, 5), np.uint8)
  img_erosion = cv2.erode(image1, kernel, iterations=2)
  image1 = cv2.dilate(img_erosion, kernel1, iterations=5)

  #detecting the blood cell
  edgedImage = findEdges(image1)
  edgedContours = getImgContours(edgedImage)
  edgedBoxes =  getBoxes(edgedContours, image.copy())

  # get the large box and get its cordinate
  last = edgedBoxes[-1]
  max_x = int(max(last[:,0]))
  min_x = int( min(last[:,0]))
  max_y = int(max(last[:,1]))
  min_y = int(min(last[:,1]))

  # draw the contour and fill it
  mask = np.zeros_like(image)
  cv2.drawContours(mask, edgedContours, len(edgedContours)-1, (255,255,255), -1)

  # any pixel but the pixels inside the contour is zero
  image[mask==0] = 0

  # extract th blood cell
  image = image[min_y:max_y, min_x:max_x]
  # print(image)
  if len(image) == 0:
    break
  image = cv2.resize(image, image_size)

  image = np.array(image, dtype = 'float32')

  img = image / 255.0
  img = np.expand_dims(img, axis=0)

# load the trained model
  tflite_model_path = "/content/MyDrive/model.tflite"

  Bloodcell_interpreter = tf.lite.Interpreter(model_path=tflite_model_path)
  Bloodcell_interpreter.allocate_tensors()
```

```
#check the input and output formate
  input_details = Bloodcell_interpreter.get_input_details()
  output_details = Bloodcell_interpreter.get_output_details()
  print(input_details)

  input_shape = input_details[0]['shape']
  print(input_shape)

  Bloodcell_shape = input_details[0]['shape']

  input_data = img

#Alocate tensor


Bloodcell_interpreter.set_tensor(input_details[0]['index'], input_data)

  time1=time()
  Bloodcell_interpreter.invoke()
  time2=time()
  classification_time = np.round(time2-time1, 3)
  totalTime = totalTime + classification_time
  print("Classificaiton Time =", classification_time, "seconds.")

#predeiction
  Bloodcell_preds = Bloodcell_interpreter.get_tensor(output_details[0]['index'])

  print("%%% " , str( np.round(Bloodcell_preds[0][Bloodcell_preds.argmax()] * 100,
3) ) + "%")


  Bloodcell_label = class_labels[Bloodcell_preds.argmax()]  #Find the label
  dic[Bloodcell_label] = dic[Bloodcell_label] + 1
```

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.