



**CLASSIFICATION OF DIABETIC RETINOPATHY USING PRE-TRAINED  
DEEP LEARNING MODELS**

**INAS AL-KAMACHY**

**OCTOBER 2019**

CLASSIFICATION OF DIABETIC RETINOPATHY USING PRE-TRAINED  
DEEP LEARNING MODELS

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF NATURAL AND APPLIED  
SCIENCES OF  
ÇANKAYA UNIVERSITY



INAS AL-KAMACHY

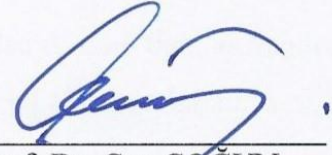
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE  
DEGREE OF  
MASTER SCIENCE  
IN  
COMPUTER ENGINEERING  
DEPARTMENT

OCTOBER 2019

Title of the Thesis: **Classification of Diabetic Retinopathy using Pre-trained Deep Learning Models.**

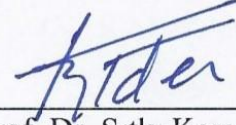
Submitted by **INAS AL-KAMACHY**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University



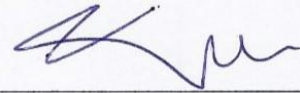
Prof. Dr. Can ÇOGUN  
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.



Prof. Dr. Sitki Kemal İDER  
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

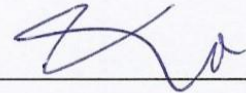


Assist. Prof Dr. Roya Choupani  
Supervisor

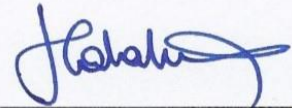
**Examination Date:** 9-Oct-2019

**Examining Committee Members**

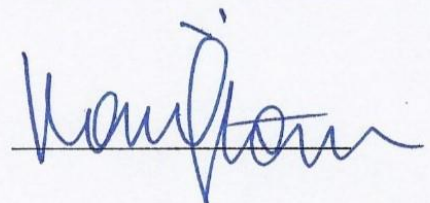
Assist. Prof. Dr. Roya CHOUPANI Çankaya Univ.



Assoc. Prof. Dr. Hadi Hakan MARAŞ Çankaya Univ.




Assoc. Prof. Dr. Kasim Oztoprak Karatay Univ.



## STATEMENT OF NON-PLAGIARISM PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : INAS AL-KAMACHY

Signature : 

Date : 21-10-2019

## **ABSTRACT**

### **CLASSIFICATION OF DIABETIC RETINOPATHY USING PRE-TRAINED DEEP LEARNING MODELS**

**AL-KAMACHY, INAS**

M.Sc., Department of Computer Engineering

Supervisor: Assist. Prof. Dr. Roya CHOUPANI

SEPTEMBER 2019, 93 pages

Diabetic Retinopathy (DR) is considered to be the first factor that leads to blindness. If it is not detected early, many people around the world would suffer from the diabetic disease that may lead to DR in their eyes. Any delay in regular monitoring and screening by ophthalmologists may cause rapid and dangerous progress of this disease which finally leads to human vision loss.

The imbalance between the numbers of doctors required to monitor this disease and the number of patients around the world increasing year by year shows a major problem leading to poor regular monitoring and loss vision in many cases which could have been detected had there been good treatment in the earlier stages of DR.

In order to solve this problem, serious aid was needed for a computer aid diagnosis (CAD).

Deep learning pre-trained models are state-of-art in image recognition and image detection with good performance.

In this research, we used image pre-processing and we built several convolution neural network models from scratch and fine-tuned five pre-trained deep learning models which used ImageNet as the dataset for medical images of diabetic retinopathy in order to classify diabetic retinopathy into five classes. After that, we selected the model that showed good performance to build a diabetic retinopathy web application using Flask as a framework web service.

We used the KAGGLE kernel website with Jupyter as a notebook as well as Flask to

build our web application. The final result of the AUC was 0.68 using InceptionResNetV2.

**Keywords:** Diabetic Retinopathy, Deep Learning, InceptionResNetV2, Flask, web application, AUC.



## ÖZ

# ÖN EĞİTİMLİ DERİN ÖĞRENME MODELLERİ KULLANARAK DİYABETİK RETİNOPATİSİNİN SINIFLANMASI

AL-KAMACHY, INAS

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Danışman: Dr. Öğr. Üyesi ROYA CHOUPANI

EYLÜL 2019, 93 sayfa

Diyabetik Retinopati (DR) körlüğe yol açan ilk faktör olarak kabul edilir. Erken tespit edilmezse, dünyadaki birçok insan gözlerinde DR'ye neden olabilecek diyabetik hastalıklardan muzdarip olur. Oftalmologlar tarafından düzenli izleme ve taramada meydana gelen herhangi bir gecikme, bu hastalığın hızlı ve tehlikeli bir şekilde ilerlemesine neden olabilir ve bu da insan görme kaybına neden olur.

Bu hastalığın izlenmesi için gerekli olan doktor sayısı ile her geçen yıl artan dünyadaki hasta sayısı arasındaki dengesizlik, birçok durumda iyi tedavi edilmiş olduğu tespit edilen birçok vakada kötü düzenli izleme ve kayıp görüşüne yol açan önemli bir problem olduğunu göstermektedir. DR.

Bu sorunu çözmek için, bir bilgisayar yardımı teşhisi (CAD) için ciddi yardıma ihtiyaç vardı.

Derin eğitim önceden eğitilmiş modeller, görüntü tanıma ve iyi performansla görüntü algılamada son teknolojidir.

Bu araştırmada, görüntü ön işlemeyi kullandık ve çizik ve ince ayarlı beş ön eğitilmiş derin öğrenme modelinden çeşitli evrişimli sinir ağı modelleri kurduk, ImageNet'i diyabetik retinopatinin tıbbi görüntüleri için veri seti olarak veri seti olarak kullandık. Beş sınıf Onan sonra, Flask'ı çerçeve web servisi olarak kullanarak diyabetik retinopati web uygulaması oluşturmak için iyi performans gösteren modeli seçtik.

Web uygulamamızı oluşturmak için JAGSER ile KAGGLE çekirdek web sitesini bir

dizüstü bilgisayar ve Flask olarak kullandık. AUC'nin nihai sonucu, InceptionResNetV2 kullanılarak 0.68 idi.

**Anahtar Kelimeler:** Diyabetik Retinopati, Derin Öğrenme, InceptionResNetV2, Flask, web uygulaması, AUC.





## ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Asst. Prof. Dr. Reza Hassanpour and Asst. Prof. Dr. Roya Choupani for their supervision, special guidance, suggestions, and encouragement through the development of this thesis.

Secondly, I would also like to thank my family, parents, husband, brothers, sister, and my lovely children Yusuf and Mariam, who helped me a lot to complete my projects within the limited time frame.

Finally, special thanks to my friend Dr. Nikhil Tomar for his patience with all my questions, and for his guidance.

## TABLE OF CONTENTS

<b>STATEMENT OF NON PLAGIARISM.....</b>	<b>iii</b>
<b>ABSTRACT.....</b>	<b>iv</b>
<b>ÖZ.....</b>	<b>vi</b>
<b>ACKNOWLEDGMENT.....</b>	<b>vii</b>
<b>TABLE OF CONTENTS.....</b>	<b>ix</b>
<b>LIST OF FIGURES.....</b>	<b>xiii</b>
<b>LIST OF TABLES.....</b>	<b>xv</b>
<b>LIST OF ABBREVIATIONS.....</b>	<b>xvi</b>
<b>CHAPTERS:</b>	
<b>1 INTRODUCTION</b>	
1.1 PROBLEMS DEFINITION.....	2
1.2 SCOPE AND OUTLINE OF THE THESIS.....	3
<b>2 MACHINE LEARNING ALGORITHMS</b>	
2.1 INTRODUCTION.....	4
2.2 UNSUPERVISED LEARNING.....	4
2.3 REINFORCEMENT LEARNING.....	4
2.4 SEMI-SUPERVISED LEARNING.....	5
<b>2.5 SUPERVISED LEARNING</b>	
2.5.1 SUPPORT VECTOR MACHINE.....	7
2.5.2 HISTORY.....	7
2.5.3 OVERVIEW .....	7
2.5.4 LINEAR SUPPORT VECTOR MACHINE .....	9
2.5.5 NON-LINEAR SVM CLASSIFIER.....	9
2.5.6 KERNEL FUNCETION.....	10
2.5.7 KERNEL TRICK.....	10
2.5.8 KERNEL TYPES.....	12

<b>2.6</b>	<b>DECISION TREE</b>	
2.6.1	INTRODUCTION.....	12
2.6.2	ENTROPY.....	13
2.6.3	COMPUTER INFORMATION GAIN.....	14
2.6.4	DECISION TREE ALGORITHM STEP.....	14
<b>2.7</b>	<b>FEED FORWARD NEURAL NETWORK</b>	
2.7.1	HISTORY.....	14
2.7.2	ACTIVATION FUNCTION.....	16
2.7.3	LINEAR ACTIVATION FUNCTION.....	17
2.7.4	NON- LINEAR ACTIVATION FUNCTION.....	17
2.7.5	MOST POPULAR NON-LINEAR ACTIVATION.....	18
2.7.6	FEED FORWARD NEURAL NETWORK.....	19
<b>2.8</b>	<b>BACKPROPAGATION NEURAL NETWORK</b>	
2.8.1	COST FUNCTION.....	19
2.8.2	QUADRATIC COST.....	20
2.8.3	GRADIENT DESCENT (GD).....	20
2.8.4	TYPE OF GRADIENT DESCENT	21
2.8.5	BACKPROPAGATION OVERVIEW.....	21
2.8.6	THE PHASES PERFORM IN BP	22
<b>2.9</b>	<b>DEEP LEARNING</b>	
2.9.1	HISTORY.....	24
2.9.2	DEEP LEARNING OVERVIEW.....	25
2.9.3	DEEP LEARNING ARCHITECTURE.....	26
2.9.4	CONVOLUTION NEURAL NETWORK.....	27
2.9.5	THE MAIN OPERATION OF CNN.....	28
2.9.6	TYPE OF LAYERS IN CNN.....	29
2.9.7	TYPES OF CNN ARCHITETURE.....	31
<b>3</b>	<b>PREVIOUS WORK</b>	
3.1	SVM FOR DR CLASSIFICATION.....	40
3.2	BP FOR DR CLASSIFICATION.....	42
3.3	CNN FOR DR CLASSIFICATION.....	46

3.4	INCEPTIONV3 FOR DR CLASSIFICATION.....	53
3.5	VGG FOR DR CLASSIFICATION.....	55
3.6	RESIDUAL NETWORK FOR DR CLASSIFICATION.....	57
3.7	ALEXNET FOR DR CLASSIFICATION.....	58
3.8	AUTOENCODER FOR DR CLASSIFICATION .....	59
<b>4</b>	<b>THE PROPOSED METHOD</b>	
4.1	DATASET AND PRE-PROCESSIN OPERATION.....	62
4.2	Hyper parameter of the network	63
4.3	MEASURE METRICS.....	64
<b>4.4</b>	<b>BUILD THE MODEL</b>	
4.4.1	CNN.....	65
4.4.2	MOBILENET.....	66
4.4.3	VGG16.....	67
4.4.4	INCEPTIONRESNETV2.....	68
4.4.5	INCPTIONV3.....	70
4.4.6	COMPARISION.....	71
4.4.7	FINE-TUNE PRE-TRAINED MODEL.....	72
4.4.8	Build ML web application using Flask.....	73
4.4.9	Main step to build HTTP accessible client of pre-trained DL model	73
<b>5</b>	<b>EXPERIMENTAL RESULTS</b>	
5.1	PLATFORM.....	75
5.2	PREREQUISITES.....	75
<b>5.3</b>	<b>LOAD DATA AND PRE-PROCESSING</b>	
5.3.1	RESIZING.....	76
5.3.2	SHUFFLE THE DAT.ASET.....	76
5.3.3	SPLIT THE DATASE.T.....	77
5.3.4	STANDARDIZE THE .DATASET.....	77
5.3.5	DATA-AUGMENTATION.....	77
5.4	EARLY STOPPING TECHNIQUE.....	78
5.5	COMPILE THE MODEL.....	79
5.6	FIT THE MODEL.....	79

<b>5.7 BUILD THE MODEL</b>	
5.7.1 Convolutional neural network.....	79
5.7.2 PRE-TRAINED MODEL.....	80
5.8 RESULT.....	81
<b>6 DISCUSSION AND FUTURE WORK.....</b>	<b>84</b>
<b>REFERENCES.....</b>	<b>88</b>



## LIST OF FIGURES

<b>FIGURE 1</b>	Human eyes.....	2
<b>FIGURE 2</b>	Reinforcement learning.....	5
<b>FIGURE 3</b>	The single neuron of the human brain.....	15
<b>FIGURE 4</b>	Perceptron Architecture.....	16
<b>FIGURE 5</b>	Forward-propagation.....	23
<b>FIGURE 6</b>	Second-phase propagation.....	24
<b>FIGURE 7</b>	Third-phase backpropagation.....	24
<b>FIGURE 8</b>	Edge detection.....	29
<b>FIGURE 9</b>	LetNet.....	33
<b>FIGURE 10</b>	AlexNet.....	34
<b>FIGURE 11</b>	VGG16 architecture.....	35
<b>FIGURE 12</b>	Single Inception Block.....	36
<b>FIGURE 13</b>	Inception module.....	37
<b>FIGURE 14</b>	Before using Conv.layer ( $1 \times 1$ ).....	37
<b>FIGURE 15</b>	After using Conv.layer ( $1 \times 1$ ).....	38
<b>FIGURE 16</b>	Skip connection.....	38
<b>FIGURE 17</b>	Generative Adversarial Network.....	40
<b>FIGURE 18</b>	Dataset distribution.....	63
<b>FIGURE 19</b>	Sample of Diabetic Retinopathy dataset.....	63
<b>FIGURE 20</b>	CNN model.....	66
<b>FIGURE 21</b>	InceptionResNetV2 blocks.....	70
<b>FIGURE 22</b>	Stage of inceptionv3.....	71
<b>FIGURE 23</b>	Stage four of original inception.....	72
<b>FIGURE 24</b>	Fine-tune layers.....	73
<b>FIGURE 25</b>	Build ML web application using Flask.....	74

<b>FIGURE 26</b>	Sample of predicted images using inceptionv3.....	83
<b>FIGURE 27</b>	Example of ImageNet dataset.....	86
<b>FIGURE 28</b>	Example of DR dataset.....	86



## LIST OF TABLES

<b>TABLE 1</b>	Difference between gradient types.....	22
<b>TABLE 2</b>	The computed output of the residual block.....	39
<b>TABLE 3</b>	Comparison between previous studies.....	61
<b>TABLE 4</b>	Dataset sample with levels.....	64
<b>TABLE 5</b>	Order of block of InceptionResNetV2.....	70
<b>TABLE 6</b>	Model's Achievement on ImageNet validation.....	72
<b>TABLE 7</b>	Result with color images.....	82
<b>TABLE 8</b>	Graphics Results of models.....	82
<b>TABLE 9</b>	Run-Time for each models with No. of Epoch.....	83



## LIST OF ABBREVIATIONS

DR	Diabetic Retinopathy
FFNN	Feed Forward Neural Network
ANN	Artificial Neural Network
DL	Deep Learning
ML	Machine Learning
CNN	Convolutional Neural Network
Conv.	Convolution
BV	Blood Vessels
SVM	Support Vector Machine
BP	Back Propagation
BRBP	Bayesian Regularization BP
RBBP	Resilient BP
MAS, uAns	Microaneurysms
EX	Exudate
HE	Hard Exudate
CE	Contrast Enhancement
CLAHE	Contrast Limited Adaptive Histogram Equalization
SERI	Singapore Eye Research Institute
CAM	Class Activation Map

## CHAPTER ONE

### INTRODUCTION

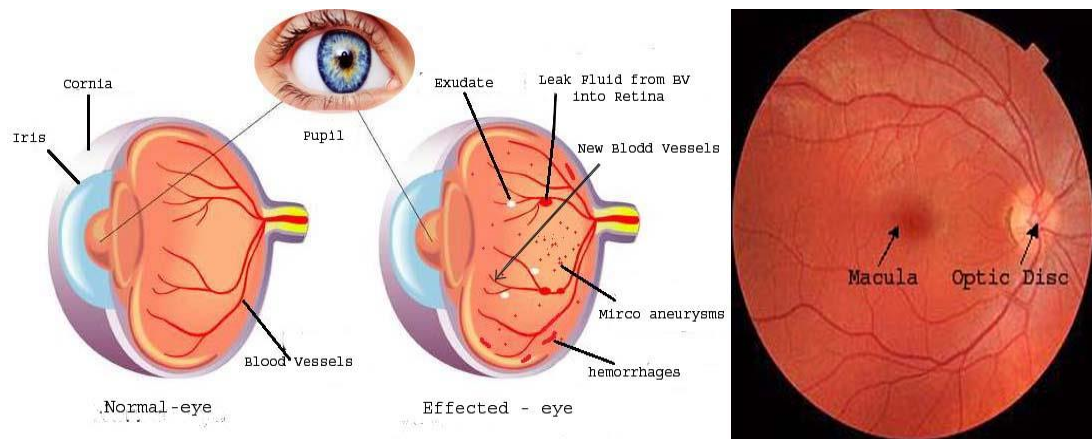
Diabetic Retinopathy is considered to be the world's first cause of blindness. Predictions of affected cases are estimated to exceed 370 million patients by 2030. The main effect of DR is the closing of the blood vessels in the eyes, followed by the eyes responding to this in two ways:

Firstly, new blood vessels are created and bleeding occurs above the main area (vitreous) which must be clear in order to allow light to be transmitted to the most sensitive part of eyes (retina) through the cornea, pupil and lens. The main role of the retina is to convert light into impulses which are transmitted through the optic nerve to the brain where processing occurs so as to be able to see the image and understand it.

Secondly, the blood vessels leak and this will influence and harm the retina, specifically in the macula (core part of the retina), whose role is detailed vision, as shown in **Figure 1**.

Diabetic retinopathy can be classified into five classes:

- Normal
- Mild non-proliferative diabetic retinopathy
- Moderate non-proliferative diabetic retinopathy
- Severe non-proliferative diabetic retinopathy
- Proliferative diabetic retinopathy



**Figure 1** Human eye

## 1.1 Problem Definition

Our goal in this research is to build a model with good performance and is capable of classifying DR images into five classes. We used color fundus images of Diabetic Retinopathy from the KAGGLE website, which was announced in 2014 for a competition named “**Diabetic Retinopathy Detection**,” after which we built a diabetic retinopathy web application using Flask as a web service.

The main challenges for us were:

### 1- Size of the Images

When the image had high resolution, it showed good results. However, it demanded high computational power when building the model. On the other hand, using low-resolution images showed lower model performance.

### 2- Sensitivity of the Images

Images that contained noise and varied in distribution between classes could affect the performance of the model.

### 3- The Algorithm

Different algorithms showed good results for image classification using specific datasets, but using the same algorithm with different domains and different datasets was a challenge in computer vision and image classification.

## 1.2 Scope and Outline of the Thesis

The aim of our thesis is to use different pre-trained deep learning convolutional neural network models varying in width (numbers of parameters) and depth (numbers of layers) to design a CAD (computer aid diagnosis) which is able to classify input images into diabetic retinopathy levels. This is challenging due to the difference between the domain that used ImageNet as a dataset in order to build a deep learning model and a medical image domain that used diabetic retinopathy images as a dataset, and whether fine-tuning a pre-trained deep learning convolutional neural network model could give similar results in DR images. Therefore, we would want to meet the challenge by selecting an appropriate algorithm to build a model for DR image classification.

This thesis contains six chapters. After the introduction and problem definitions of diabetic retinopathy classification in this chapter, we introduce in the following chapter different machine learning algorithms that can be used for image recognition and detection, and we present a brief history for each in addition to their types and main structure. In Chapter Three, we present thirty different previous studies pertaining to diabetic retinopathy classification using different machine learning algorithms with their results. In Chapter Four, we specifically introduce our proposed method with details using five types of deep learning algorithm and different pre-processing methods as well as a brief introduction to using Flask as a web service in order to build our DR web application. In Chapter Five, we introduce the experimental results with the graphical and numeric results of our models using AUC as a metric of the performance. Finally, in Chapter Six, we discuss the results and introduce future work ensuing from our research.

## CHAPTER TWO

### MACHINE LEARNING ALGORITHMS

#### 2.1 INTRODUCTION

Machine learning is a subset of the Artificial Intelligence (AI) field which used to in which we can build a model or algorithm for specific purposes based on given data using special techniques such as programs and statistical computation.

The main types of ML include:

- Supervised learning;
- Unsupervised learning;
- Reinforcement learning; and
- Semi-supervised learning.

#### 2.2 Unsupervised Learning

This type of ML contains data without labels.

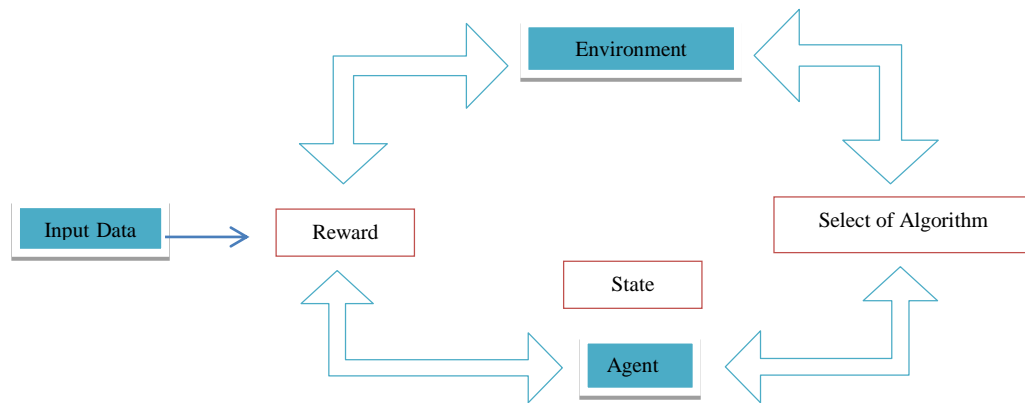
Training set =  $\{(X1), (X2), (X3), (Xn)\}$ .

The main aim is to understand the structure of the data given by algorithms.

Types of unsupervised learning include clustering and association.

#### 2.3 Reinforcement Learning

In this type of ML, the main goal is to maximize the reward action when an agent is performing the right path (increase the behavior); otherwise, a decrease in the behavior is considered to be a result of a punishment. The next step of the agent path occurs according to the kind of reward of the previous step if it is positive an continues on the same path; otherwise, it changes the path.



**Figure 2** Reinforcement Learning

Types of Reinforcement Learning include:

1. Positive Reinforcement; and
2. Negative Reinforcement.

The main factors of RL are:

1. Reward (R): denoting the feedback signal indicating how well the step that the agent makes in a particular time;
2. Action (A): the function of the reward (R) and state (S);
3. State (S): describing the environment;
4. Policy (P): the transfer from the environment to the action being (P);
5. Value Function (V): a measurement tool to indicate how good the step is; and
6. Model (M): the demonstration of the agent's environment.

## 2.4 Semi-Supervised Learning

There are techniques which combine supervised and unsupervised learning to build a model which is able to predict a large number of unlabeled data. Supervised learning uses a small number of label data initially to train the model with a known target to build the model. Unsupervised learning is used by unlabeled data for the same model which is trained prior to predicting this kind of data. This operation is named semi-supervised learning.

This technique is used in web mining, text mining and video mining in which there are huge numbers of unlabeled data and a small number of labeled data.

The main reasons to use semi-supervised learning is that the number of label data is lower than the number of unlabeled data because obtaining label data is very expensive

and difficult. Therefore, by using this method, it will make use of a large number of unlabeled data as well as increase the model accuracy.

## 2.5 Supervised Learning

Also called “learning with a master,” this type of ML contains data and labels.

Training set =  $\{(X_1, Y_1), (X_2, Y_2), (X_3, Y_3), (X_n, Y_n)\}$ .

The main target is to build a model  $\theta$  that maps  $X$  to the  $Y$  label.

Types of supervised learning include classification and regression.

Classification is supervised learning that predicts categories using specific algorithms and label data. Classification problem is the idea of categorizing data points or instances into a set of labels.

In binary classification, each input image will be classified into one of two classes (such as predicting whether an animal is a dog or a cat or whether mail is spam or non-spam).

On the other hand, in multi-classification, an input image is classified into one of the number of classes (such as classifying a MNIST handwritten digit or DR levels).

Some samples of classification include speech recognition, handwriting recognition, biometric identification, image recognition, and so on.

The most commonly used algorithms for image classification are:

- Support Vector Machine;
- Decision Tree;
- Feed-Forward Neural Network;
- Back-propagation network; and
- Deep Learning.

In this research, we pre-train a deep learning convolutional neural network that is widely used and considered as state-of-art for image classification. The algorithm shows high accuracy and consumes less time than other machine learning algorithms. It also uses a great quantity of data and performs well, requiring less image pre-processing despite noisy data.

## 2.5.1 Support Vector Machine

### ▪ History

SVM was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis (Institute of Control Sciences of the Russian Academy of Sciences, Moscow, Russia) in the framework of the “Generalized Portrait Method” for computer learning and pattern recognition for linearly separated data. The development of these ideas started in 1962 and they were first published in 1964. In 1992, Vapnik et al. presented the concept of applying what is called the ‘kernel trick,’ which allows the use of the SVM to classify linearly non-separable data hard margins. The soft margin was given by Vapnik et al. (1995), which extended the version of hard margin SVM.

Hard margin SVM can work only when data are completely linearly separable without any errors (noise or outliers). In cases of errors, either the margin is smaller or hard margin SVM fails. Soft margin SVM was proposed by Vapnik to solve this problem by introducing slack variables. SVM has become popular because of its success in both regression and classification tasks:

- Support Vector Machine (SVM): used for classification; and
- Support Vector Regression (SVR): used for regression.

However, it is widely used in classification objectives.

### ▪ Overview of Support Vector Machine

A support vector machine (SVM) performs classification by finding an optimal separating hyperplane by leaving the largest possible fraction of points of the same class on the same side and maximizing the distance of either class from the hyperplane and minimizing the risk of misclassifying the training samples and the unseen test samples. The main goal of a support vector machine is to find the optimal separating hyperplane which maximizes the margin of the training data which is the distance between the hyperplane and the closest data point.

If a hyperplane is very close to a data point, its margin will be small. The further a hyperplane is from a data point, the larger its margin will be. This means that the optimal hyperplane will be the one with the largest margin.

The two classes labels are +1 (a positive example) and -1 (negative example). The



main use of it is to perform the binary classification of the data which are linearly separable. It attempts to find the best plane or hyperplane that separates the data between two classes.

In 2 dimensions, a line must be found.

In 3 dimensions, a plane (surface) must be found.

In 4 or more dimensions, a hyperplane must be found.

The equation of the linear classifier is as follows:

$$g(x) = \text{sign}(w^t x + b) \quad \text{Eq. 1}$$

where  $w$  = weight vector (orientation of the hyperplane),  $b$  = bias, and  $x$  = the input feature vector.

### **Classification Rule**

If  $W_i X_i + b > 0$  then  $X_i \in \text{Class}_1$ , where  $X_i$  lies on the positive side

If  $W_i X_i + b < 0$  then  $X_i \in \text{Class}_2$ , where  $X_i$  lies on the negative side.

SVM attempts to maximize the distance of the separating boundary between the two classes by maximizing the distance of the separating plane of the feature vectors whether the feature vector belongs to Class1 or to Class2.

If ( $X_i \in \text{Class}_1$  and  $Y_i = +1$ ) then  $Y_i (W_0 X_i + b) > 0$ .

If ( $X_i \in \text{Class}_2$  and  $Y_i = -1$ ) then  $Y_i (W_0 X_i + b) < 0$ .

Unknown feature vector (P) has to be classified to either Class1 or Class2 using W and b that were obtained after designing the classifier.

If  $WP + b > 0$  then  $P \in \text{Class}_1$ .

If  $WP + b < 0$  then  $P \in \text{Class}_2$ .

Types of SVM classifier include:

- Linear SVM Classifiers; and
- Non-Linear SVM Classifiers.

### **2.5.2 Linear Support Vector Machine (LSVM)**

This is a type of SVM classifier where the data used in this form of SVM are linearly separable and with low variance. It has become popular for solving classification tasks due to its fast and simple online application to large-scale datasets. The distance from any point to the separator can be illustrated in the Equation **Eq. 2**.

$$r = \frac{w^t x + b}{\|w\|} \quad \text{Eq. 2}$$

Moreover, the **length of the optimal margin** can be computed using **Eq. 3**:

$$\frac{w}{\|w\|} \cdot (x_2, x_1) = \text{width} = \frac{2}{\|w\|} \quad \text{Eq. 3}$$

To design an SVM,  $W$  must be minimized simultaneously while maximizing bias  $b$ . By training the classifier, we determine what  $W$  vector and  $b$  bias are by using the initial value of  $W$  and  $b$  and for every training sample belonging to Class1, we have attempted to see whether  $W^t X + b > 0$ . If not, then we have modified  $W$  and  $b$  such that the position and/or orientation of the hyperplane is so modified that a particular  $X$ , which is taken from Class1, is moved to the positive side of this hyperplane. Similarly, if we take a vector from Class2, we check whether this  $W^t X < 0$ , where  $X$  is taken from Class2. If it is not negative, then again we have modified  $W$  and  $b$  such that a particular  $X$  is moved to the negative side of the hyperplane.

- **SVM as a Minimizing Problem**

Maximizing  $2/\|w\|$  is the same as minimizing  $\|w\|/2$ . Hence SVM becomes a minimization problem:

$$\min \frac{1}{2} \|w\|^2 \text{ s.t. } y_i(w \cdot x_i + b) \geq 1, \forall x_i \quad \text{Eq. 4}$$

$Y_i = \{+, -\}$  classes

We are now optimizing a quadratic function subject to linear constraints. Moreover, quadratic optimization problems are a standard, well-known class of mathematical optimization problems and many algorithms exist to solve them.

Therefore, the SVM is transformed into a minimization problem where we endeavor to maximize the margin between the two classes. It is used for binary classification where data is linearly separable.

### 2.5.3 Non-Linear SVM Classifier

Used with non-linear separated data and the situations where a non-linear region can separate the groups more efficiently, SVM handles this by using a kernel function (non-linear) to map the data into a different space where a hyperplane (linear) cannot

be used to perform the separation.

#### 2.5.4 Kernel Function

The kernel function is used to convert non-linear space into linear space so we can use a linear model to separate non-linear separated data and reduce the computation cost. This uses the inner product of the new vectors where we have a binary classification with  $x$  input and single features (non-linear separated data).

By mapping the input example to a new representation or too high a dimensionality, we can use a linear model to classify the non-linear dataset.

$$\mathbf{g}(x) = \mathbf{w}^t \phi(x) + \mathbf{b} \quad \text{Eq. 5}$$

$$\mathbf{g}(x) = \sum_{i \in sv} \alpha_i \phi(x_i)^t \phi(x) + \mathbf{b} \quad \text{Eq. 6}$$

$$K(x_a, x_b) = \phi(x_a) \phi(x_b) \quad \text{Eq. 7}$$

where  $K$  is kernel function and  $\phi$  is the mapping from  $X$  to an (inner product) feature space.

#### 2.5.5 Kernel Trick

This is the kernel function that transforms data into a higher dimensional feature space to make it possible to perform a linear separation, and compute the inner product of the definition space without visiting it and making the number of dimensions depend on the number of examples, not on the dimension of the space that is defined.

Let  $y = \{y_1, y_2\}$  with two feature spaces that are non-linear separable.

The inner product is a function between the transformation of  $x$  and  $x'$ :

$$\text{Let } z^t z' = K(x, x') \text{ *the kernel*} \quad \text{Eq. 8}$$

$$y = \phi(x) \quad \text{Eq. 9}$$

$$l(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^m z_n z_m \alpha_n \alpha_m y_n^t y_m \quad \text{Eq. 10}$$

Conditions:  $\alpha_n \geq 0$  for  $n = 1, 2, \dots, N$  and  $\sum_{n=1}^N \alpha_n y_n = 0$

$$g(x) = \sin(w^t \cdot z + b) \quad \text{Eq. 11}$$

We need  $y_n^t y$ .

$$b = z_m (w^t y_m + b) = 1 \quad \text{Eq. 12}$$

Using the original method for classification without using kernel function, the transformation function will convert from non-linear separable to linear separable as follows:

$$\Phi(y) \rightarrow y_1^2, y_2^2, \sqrt{2y_1 y_2}$$

**Where;**

$\Phi(y)$  is a transformation function that converts 2 dimensions into 3 dimensions.

By using the decision boundary in 3-dimensional space:

$$\beta_0 + \beta_1 y_1^2 + \beta_3 \sqrt{2y_1 y_2} = 0 \quad \text{Eq. 13}$$

For  $i$  point here, we use 4 operations for  $i$  point.

$$\Phi(y_i) = (y_{i1}, y_{i2}) \quad \text{Eq. 14}$$

$$\Phi(y_j) = (y_{j1}, y_{j2}) \quad \text{Eq. 16}$$

$$= (y_{j1}^2, y_{j2}^2, \sqrt{2y_{j1} y_{j2}}) \quad \text{Eq. 17}$$

$$= (y_{i1}^2, y_{i2}^2, \sqrt{2y_{i1} y_{i2}}) \quad \text{Eq. 15}$$

On the other hand, we have  $j$  point:

Again, we use 4 operations for  $j$  point, and finally, we compute the dot product between the vector (similarity measure):

$$(\Phi(y_i), \Phi(y_j)) = y_{i1}^2 y_{j1}^2 + y_{i2}^2 y_{j2}^2 + 2y_{i1} y_{i2} 2y_{j1} y_{j2} \quad \text{Eq. 18}$$

Here, we use 3 operations of the product and 2 additional operations.

The total number of operations using the original method is:  $4 + 4 + 3 + 2 = 13$  operations.

Using the kernel trick method:

$$(y_i, y_j)^2 = (\{y_{i1}, y_{i2}\}, \{y_{j1}, y_{j2}\})^2 \quad \text{Eq. 19}$$

$$= (y_{i1}y_{i2} + y_{j1}y_{j2})^2 \quad \text{Eq. 20}$$

$$= y_{i1}^2 y_{j1}^2 + y_{i2}^2 y_{j2}^2 + 2y_{i1}y_{i2}2y_{j1}y_{j2} \quad \text{Eq. 21}$$

The number of operations used by the kernel is as follows:

- Using three multiplication operations (two for the dot product and one for the square operation); and
- Using one additional operation.

The total number is  $3 + 1 = 4$  operations used by the kernel, which is less than the original method than the mapping data from 2-dimensional space to 3-dimensional space followed by applying the required operations while using the kernel trick which is about to stay in the same 2-dimensional space and computing the same result as in the 3-dimensional space.

### 2.5.6 Kernel Types

- Polynomial
- Gaussian
- Gaussian Radial Basis Function (RBF)
- Laplace RBF
- Hyperbolic tangent
- Sigmoid
- Bessel function of the first kernel
- ANOVA radial basis
- Linear splines kernel in one dimension

## 2.6 Decision Tree

### 2.6.1 Introduction

A decision tree builds classification or regression models using a tree structure. It divides a dataset into smaller and smaller subsets while simultaneously an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

- A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and

Rainy)

- A leaf node (e.g., Play) represents a classification or decision.

The topmost decision node in a tree which corresponds to the best predictor called the root node and decision trees can handle both categorical and numerical data. The core algorithm for building decision trees called ID3 by J.R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking and ID3 uses Entropy and Information Gain to construct a decision tree.

## 2.6.2 Entropy

These are the measures of impurity, disorder or uncertainty in a number of examples. Its work controls how a Decision Tree decides to split data. In fact, it affects how a Decision Tree draws its boundaries.

A decision tree is built top-down from a root node and involves partitioning data into subsets containing instances with similar values (homogeneous). The ID3 algorithm uses entropy to calculate the homogeneity of a sample such that if the sample is completely homogeneous, the entropy is zero, and if the sample is equally divided, it has entropy equal to one. The value of the entropy is computed using two or perhaps three types of class or category by multiplying the probability of each category of each class by  $\log_2 p$  of the value of that probability and summing over all the values of classes. To build a decision tree, we need to calculate two types of entropy using frequency tables as follows:

Entropy using the frequency table of one attribute (Entropy of the Target) for splitting:

$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i \quad \text{Eq.22}$$

S = our Target

$p_i$  = the probability of each of these classes

C = the number of classes; we use minus here because the probability is between 0 and 1, and the logarithm of any value between 0 and 1 is negative, and by canceling each other, it become positive. After splitting, we compute the edge and the difference between two entropies and determine the highest information gain.

$$\text{Gain}(T, X) = \text{Entropy}(T) - \text{Entropy}(T, X) \quad \text{Eq.23}$$

b) Entropy using the frequency table of two attributes:

$$E(T, X) = \sum_{c \in X} P(c) - E(c) \quad \text{Eq.24}$$

where  $P(c)$  is a probability of the category and  $E(c)$  the entropy of that category.

### 2.6.3 Computer Information Gain

Information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree pertains to finding an attribute that returns the highest information gain (i.e., the most homogeneous branches). Therefore, we want the one with the lowest entropy so that the difference here is as high as possible.

### 2.6.4 Decision Tree Algorithm step for classification

Step 1: Calculate the entropy of the target.

Step 2: The dataset is then split into different attributes. The entropy for each branch is calculated. Then it is added proportionally to obtain the total entropy for the split, and the resulting entropy is subtracted from the entropy before the split. After that, the result is the Information Gain or a decrease in entropy.

Step3: Select the attribute with the largest information gain as the decision node.

Step4a: A branch with the entropy becomes a leaf node.

Step4b: A branch with entropy greater than 0 needs further splitting.

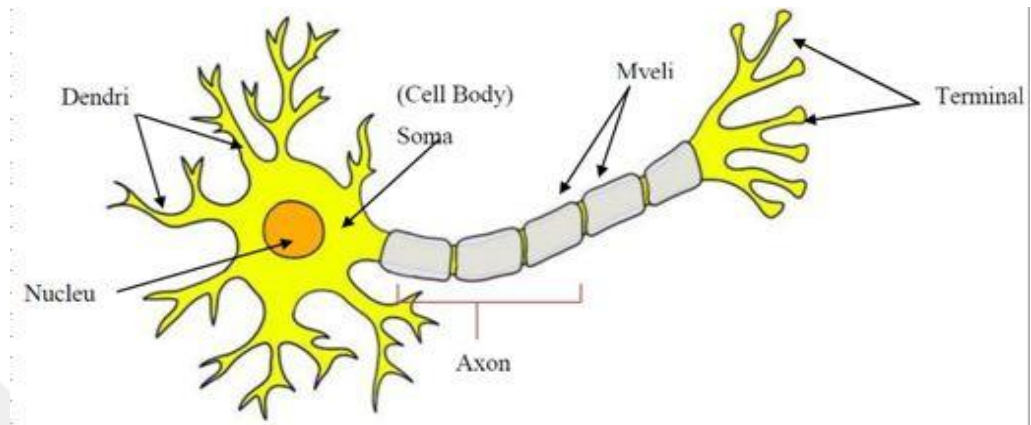
Step5: The ID3 algorithm is run recursively on the non-leaf branches until all data are classified.

## 2.7 Feed Forward Neural Network

### 2.7.1 History

FFNN has existed for a long time. The idea of a model neural network of the human brain was begun in 1943 by WARREN S. McCULLOCH and WALTER PITTS by defining threshold logic and introducing a neural network model which attempted to simulate the human brain. This led to an understanding of the structure of the brain, especially the external cortex of the brain which consists of a huge number of neurons

connecting between each other in a parallel distributed process so as to speed up and make robust the learning process using weights that need to be tuned by human interaction manually.



**Figure 3** Single neuron of the human brain

As shown in **Figure 3**, every single neuron in the human brain has three operations[1]:

- An input operation using a synapse (to receive signals from another neuron);
- Excitation or damping collected signals using the cell body (soma), which depends on the chemistry of the cell body.
- An output operation using an axon to send the final signal to another neuron.

Therefore, each of the neurons is fired (on or off) with a non-linear function, where the output of a neuron will be the input to another neuron. In the human brain, approximately 100 billion neurons that connect to each other via 100 trillion connections. The transmission rate of the synapse is approximately 100 bits per second.

In 1949, Donald Hebb defined the first learning methods to be formulated named “Hebbian Learning” and introduce the concept of “correlation learning.” This is the idea that the weight of connection is adjusted based on the values of the neurons to which it connects, thus:

$$\Delta w_{ij} = \alpha a_i a_j \quad \text{Eq. 25}$$

where

$\alpha$  = learning rate

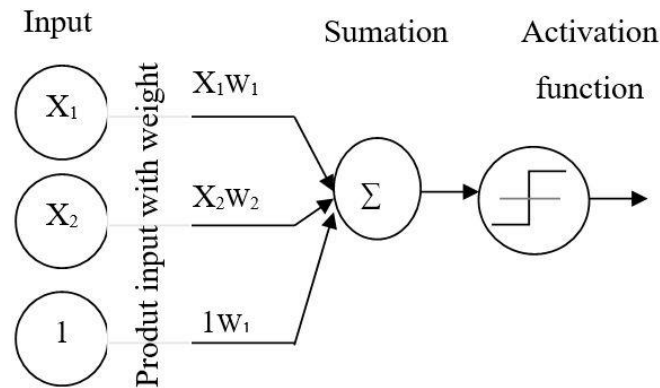
$a_i$  = the activation of the  $i^{\text{th}}$  neurons in one neuron layer



$a_j$  = the activation of the  $j^{\text{th}}$  neurons in another layer

$w_{ij}$  = the connection strength between the two neurons

In 1959, Rosenblat introduced the perceptron algorithm with one layer that solves linear classification problems (binary image classification), as shown in **Figure 4**.



**Figure 4** Perceptron architecture

The limitation of the perceptron algorithm is that it could not solve the non-linear classification (XOR) problem. Therefore, people such as Marvin Minsky were skeptical of neuron nets for a long time when he declared the limitation of the perceptron algorithm for solving the non-linear problem in 1969. This coincided with an AI winter. The neural network had a revival in 1974 by Paul J. Werbos, who defined the Backpropagation algorithm with a three-layered perceptron network. In the 1980s, interest in the backpropagation algorithm increased and developed into a Backpropagation algorithm with the MLP (multi-layer perceptron) by Rumelhart and McClelland such that it can deal with the non-linearity problem consisting of three main components (input layer, hidden layers and output layers), and opening a new window to more research and development in the neural network field and an ability to solve many problems which were previously difficult to solve.

### 2.7.2 Activation Function

The activation function translates input signals into output signals, after which it computes the weight summation and adds the bias and passes it to the activation function, which controls the output and manages the fire operation (on and off) of the nodes in the specific layer (hidden layer and output layer), thus:

$$\text{Output} = \text{ActivationFunction} \sum [(weight * input) + bias] \quad \text{Eq.26}$$

If the output exceeds the threshold, then the output will pass and be an input into another neuron, otherwise it will not. The choice of activation function in a neural network model is one of the essential tasks of deep learning model. These activation functions are categorized in two main types:

- **Linear activation (identity) function**

This has a simple structure:

$$Z = cx \quad \text{Eq. 27}$$

It receives inputs and then multiplies them by the weights for each neuron and generates an output signal comparable to the input.

The main drawback of the linear activation function:

- With constant derivatives, it cannot be use in BP (GD).
- Whatever the numbers of hidden layers, with the linear activation function, it will be one layer and the final output of a network is still a linear function of the input in spite of the number of hidden layers, which was the limitation of using the linear function on a hidden layer as it does not allow us to have many hidden layers as we use in deep learning.

Moreover, linear activation function performs dreadfully when the network deals with high-dimensions of data and various data types (images, audio, speech, etc.), or in some case dealing with Big data. Therefore, this type of activation function is mainly use in the output layer for the classification task to separate the data into classes while using a non-linear activation function in the hidden layer.

- **Non-linear activation function**

The factor that makes ANNs and BP develop and be widely used in many studies that solve large numbers of complicated problems is the non-linearity activation function.

The main features of this type of activation function:

- It is the derivative that will allow to the network to be learned from the

weights.

- It allows the use of many hidden layers and it can be used with different types of data. It can perform with big data and allow us to learn non-linear problems.

The most popular and widely use non-linear activation functions are:

- **Sigmoid (logistic) function**

$$f(x) = \frac{1}{1 + e^x} \quad \text{Eq. 28}$$

With a curve shaped like the letter S, the output of this function changes continuously. However, it does not change linearly and we can observe that the value of the output lies on (0, 1) as the input gradually changes from negative infinity to positive infinity. Therefore, it is used when the output is expected to be a positive number (MLP and Backpropagation algorithms). Moreover, it is considered a reasonable approximation of real neurons.

Using the sigmoid function is a bad choice because of the vanishing gradient issue, which occurs when the function directs the input to a small range, namely [0, 1], of the output. Therefore, for any changes to the input parameter, even when large, the result is a very small change to the value of the output and the gradient will be very small, thereby causing the vanishing gradient obstacle [2].

- **Hyperbolic Tangent (tanh) function**

This function is a commonly used activation function as it works with both negative and positive numbers. Its output ranges over [-1, +1]. Its equation is:

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \quad \text{Eq. 29}$$

Moreover, it has a vanishing gradient slope.

- **Rectified Linear Unit (ReLU)**

ReLU is a familiar activation function because of its simplicity. It has a good effect such that it removes vanishing gradients and is used in hidden layers. However, the weak point here is dead neurons.

In 2017, Shumin Kong et al. stated that “ReLU thresholds all negative values to zero

$f(x) = \max(0, x)$ ; its positive part has a fixed gradient of 1. Hence, ReLU will not saturate in the positive part. However, in the negative part, the gradient of ReLU, with respect to the input, is zero, which means once a ReLU neuron produces a negative output, the gradient flow into the neurons will always be zero. The weight of the neuron would therefore never be optimized, due to zero gradient” [2, p. 2563].

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad \text{Eq. 30}$$

- **Exponential Linear Unit (ELUs)**

This function overcomes the dead neuron problem by using an exponential process, so it is better than the ReLU function. This function is used in hidden layers.

$$f(x) = \begin{cases} \alpha (e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases} \quad \text{Eq. 31}$$

- **Softmax function**

This function is a type of sigmoid function over  $[0, 1]$ . The main difference is that the sigmoid function is used to classify two classes while Softmax is used to classify more than two classes (multiclass). It is placed on the final layer (O/P layer) that turns logits, the vector of numbers, into a probability for each class. A higher probability could indicate the predicted class where the sum of these probabilities is equal to one.

$$(\sigma)(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}} \quad \text{Eq. 32}$$

### 2.7.3 Feedforward neural network overview

Feedforward neural network is a non-recurrent network which contains inputs, outputs and hidden layers. The signals can only travel in one direction. The input is passed onto a layer of processing elements where it performs the computations performing a weighted summation of its inputs and adding the bias value. This is followed by applying the activation function where the new calculated output value becomes the new input value that feeds the next layer. This process continues until it has passed through all the layers and has determined the output. A threshold transfer function is sometimes used to quantify the output of a neuron in the output layer.

## 2.8 Backpropagation Neural Network

### 2.8.1 Cost function

The cost function is the measurement of accuracy or performance of the model by defining the value between the actual output and the predicted output. If the value is small, it means the performance of the model is good because the actual output and predicted output are covered.

The parameters of the cost function (C) are:

W = weight of the neural network

B = bias of the neural network

$S^i$  = input of the training data

$Y^i$  = desire or actual output of the training data

$P^i$  = predicted output of training data

### 2.8.2 Quadratic cost

This is a type of cost function known as the MSE (mean square error) cost function or sum square error [32].

When training set  $T = \{(X^{(1)}, Y^{(1)}) \dots (X^{(n)}, Y^{(n)})\}$ , then:

$$C(MSE) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \quad \text{Eq. 33}$$

**Where;**

- $h_{\theta}$  is the predictive value,
- $y^{(i)}$  the real-world value,
- $m$  the number of the training example.

In order to minimize the cost function, the difference between the actual output and the predicted output must be minimized. To do so, the change must occur in the predicted output because the actual output is real data and we cannot change it.

In the predicted output, the main parameters we can change are the weight of the

network and to do that, we need to use the gradient descent technique.

### 2.8.3 Gradient Descent (GD)

The Gradient Descent is an optimization algorithm used to minimize cost function so as to speed up learning by calculating the derivative of a function (updating the value of weight and bias) during each iteration until convergence.

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \quad \text{Eq. 34}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \quad \text{Eq. 35}$$

where  $\alpha$  is the learning rate that defines the speed of GD reaching a local minimum, and  $\theta$  refers to the weight.

### 2.8.4 Type of Gradient Descent depending on data size [3]

#### - Batch GD

This algorithm computes the gradient of the entire dataset in order to make one update; therefore, it slows with a large dataset and does not allow updating the model online.

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta) \quad \text{Eq. 36}$$

#### - Stochastic GD

The Stochastic GD can be used for big data because it is faster than Batch GD. However, it is less accurate because it computes the gradient of a single parameter for every training example  $x^{(i)}$  and label  $y^{(i)}$  and it can update the model online.

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)}) \quad \text{Eq. 37}$$

#### ▪ Mini-Bach GD

This algorithm differs from GD algorithms that divide the training dataset into  $n$  batches which are used to compute the gradient of the cost function instead of computing the gradient of the entire training dataset. The operation may take the average or summation of all the Mini-Bach GD to minimize the variation of the gradient. Typical sizes of mini-batch ( $n$ ) occur over the ranges [50-256]. This type of GD is a family used in DL.

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \quad \text{Eq. 38}$$

**Table 1** Differences between Gradient types

GD type	Bach GD	Stochastic GD	Mini-Bach GD
Efficiency	High	Low	Balance
Time-period	Long	Short	Balance

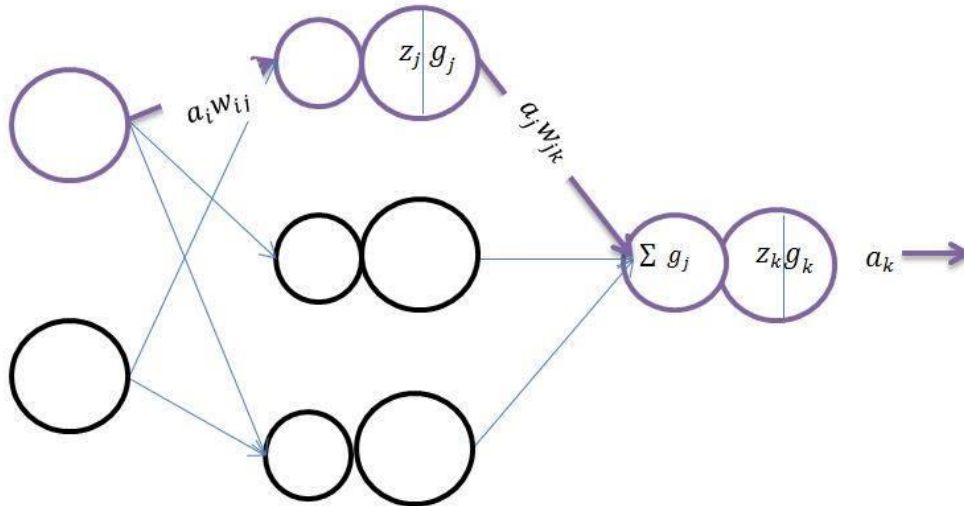
### 2.8.5 Backpropagation overview

Backpropagation (BP) considers the light that enables the development toward NN. Its supervised learning requires knowledge of the desired output for each neuron and it is used by the gradient descent to compute the gradient in order to minimize the cost function using derivative calculations. It consists of three types of layers (an input layer, a hidden layer and an output layer).

- **Phases performed in BP**

- **First Phase:** Forward- propagation:

In a typical backpropagation network, the first phase of forward-propagation will multiply the weight ( $w_{ij}$ ) with the input ( $a_i$ ) is to become the product value ( $w_{ij}a_i$ ) as shown in **Figure 4**. After the bias are added, the middle layer will apply activation function on the weighted summation of the result ( $z_j g_j$ ) which will be the output of the first layer and the input to the next layer ( $a_j w_{jk}$ ). Finally, the cost function is used to compare between the actual output and the result (predicted output).



**Figure 5.** Forward propagation

$$a_k = g_k(b_k + \underbrace{\sum g_j (b_j + \underbrace{\sum a_i w_{ij}}_{z_j}) w_{jk}}_{z_k}) \quad \text{Eq. 39}$$

- **Second Phase:** Back-propagation of error:

Back-propagation is the error from the output layer to the input layer, which will be a factor for updating weights and bias in two steps.

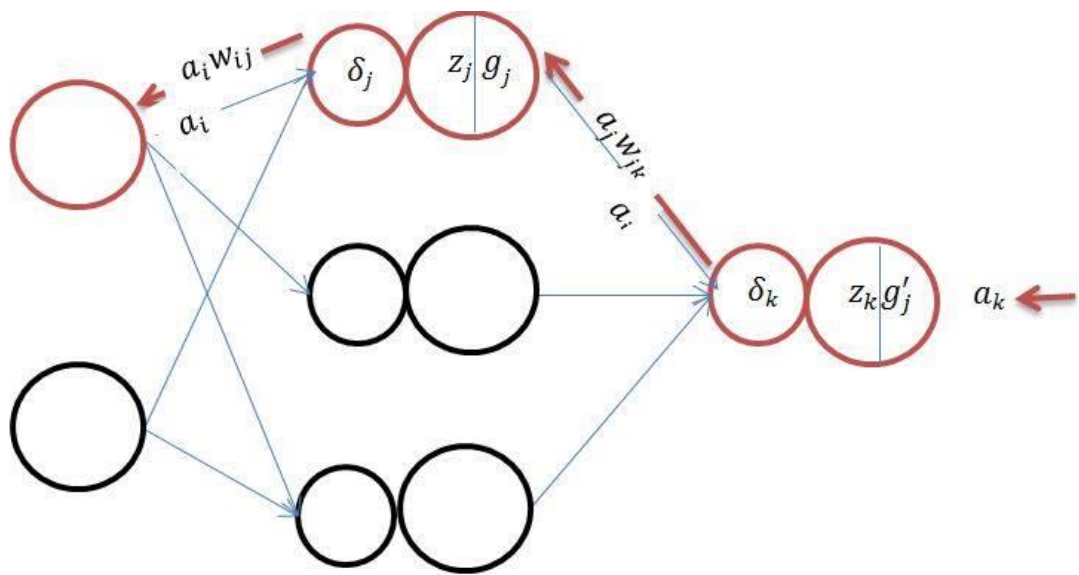
From the output layer to the hidden layer, the network output error is computed.

$$\delta_k = g'_k(z_k) E'(a_k, t_k) \quad \text{Eq. 40}$$

From the hidden layer to the input layer:

$$\delta_j = g'_j(z_j) \sum_k \delta_k w_{jk} \quad \text{Eq. 41}$$





**Figure 6** Second phase Backpropagation

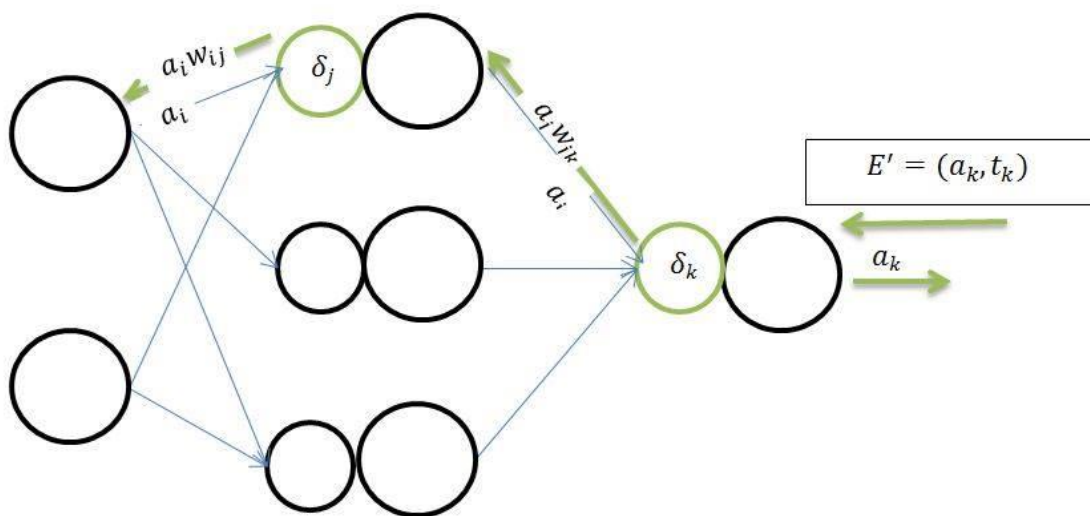
- **Third Phase:** Computing the gradient into steps:

From the output layer to the hidden layer:

$$\frac{\partial E}{\partial w_{ij}} = a_i \delta_j \quad \text{Eq. 42}$$

From the hidden layer to the input layer:

$$\frac{\partial E}{\partial w_{jk}} = a_j \delta_k \quad \text{Eq. 43}$$



**Figure 7** Third phase of Backpropagation

- **Fourth Phase:** Updating the weights of the network

Updating the weights connecting the input layer to the hidden layer:

$$w_{ij} = w_{ij} - \eta a_i \delta_j \quad \text{Eq. 44}$$

Updating the weights connecting the hidden layer to the output layer:

$$w_{jk} = w_{jk} - \eta a_j \delta_k \quad \text{Eq. 45}$$

The repetition of four phases is stopped until the value of the cost function is minimized to the smallest value.

## 2.9 Deep Learning

### 2.9.1 History

Deep Learning history can be traced back to 1943 when a computer model was developed by Walter Pitts and Warren McCulloch based on neural networks, with imitation of human brain. They used a combination of algorithms and mathematics they called “threshold logic” to mimic the thought process. Furthermore, in the 90s Yann LeCun & Yoshua Bengio (1995) take a further step from natural Multilayer Perceptron (MLP) with the sole purpose to reduce both high computational tasks and availability of high number of dimensions. Subsequent to that, a paper was published [4] that presented a better pattern recognition system by canceling unrelated variables.

Then E Hinton publish a paper that introduced Restricted Boltzmann Machines (RBM) [5] which are used for filtering, the classification of labeled and non-labeled data, and the reduction of data dimensionality, which helps to simplify computations. In 2012, Alex Krizhevsky and his team used a convolutional neural network named AlexNet which won ImageNet LSVRC-2012, 2010 (Large Scale Visual Recognition Competition) using a training set of images of more than one million high-resolution RGB label images to classify them into 1000 categories.

DL algorithms had improved and be used to solve many tasks, including image segmentation, detection, style transfer, analogies and image classification. This improvement is due to the existence of the great quantity of data (big data) as well as high level hardware which is capable of dealing with high-level computational tasks. Finally, in 2017 research was made into the deep neural net with error rates of 3%.

## 2.9.2 Deep Learning Overview

The main definition of DL is that it is a Neural Network with many hidden layers, so “deep” here refers to the depth of layers consisting more than 2 hidden layers. It provides automatic feature extraction by determining the properties of the input data which can be used as a pointer to label the input data accurately. Each layer extracts features from the output of previous layer. This was a revolution in computer vision tasks. In contrast, shallow networks required manual designs for feature extraction and a great amount of experience in the image processing field. On the other hand, transformations from input images to vectors led to losses of much interesting information.

There are 7 main applications that are mainly practiced in DL [6]:

- Automatic Speech Recognition (ASR)
- Image Recognition
- Natural Language Processing
- Drug Discovery and Toxicology
- Customer Relationship Management
- Recommendation Systems
- Bioinformatics
- Deep Learning Frameworks
- Tensor Flow
- Theano
- Keras
- Torch
- Caffe

## 2.9.3 Deep Learning Architectures [7]

- **Restricted Boltzman Machines (RBMs)**

RBMs are stochastic NNs that consist of two types of layer (a visible layer and a hidden layer) such that each neuron in a visible layer is fully connected to each neuron in the hidden layer, and vice versa with neurons of in the hidden layer. Moreover, each

neuron has to make a stochastic determination whether to transmit the signal between the two layers. There are no connections between the neurons on the same layer, so they are restricted in terms of connection such that each input will be part of the network which has no output layer; thus, it is a generative model.

RBMS is an unsupervised ML and an energy-based model used to determine the probability of distribution with consideration for input data or the minimization of the energy.

The visible layer (input layer) passes the input to the hidden layer by using the weights ( $w_{ij}$ ). And the result of the product will be apply to the activation function and the weight value will be updated as illustrated in the **Equation Eq. 53**.

$$\frac{\partial \log p(v^o)}{\partial w_{ij}} = (v i^o h j^o) - (v i^\infty h j^\infty) \quad \text{Eq. 46}$$

where  $i$  is the visible unit,  $j$  the hidden unit,  $w_{ij}$  the weight between the layers, and  $(v i^o h j^o)$  and  $(v i^\infty h j^\infty)$  the correlations when  $(i,j)$  are in the minimum and maximum layers.

RBM is used to reduce the dimension, filtering, classification, and feature extraction. However, drawbacks include the fact that it cannot obtain the partition function at the same resolution and it is difficult to train [8].

- **Deep Belief Networks (DBNs)**

A DBN is a stack of RBMs and each layer in a DBN represents two performance visible layers considered to be the layer after it. The hidden layer considers the layer before it. DBN Hinton applies two phases to solve the inaccuracy of the network due to the depths of layers [9].

**Phase1: Layer-Wise**

By training the input data on an RBM and obtaining the parameters, the output will be as input to the next RBM. This form will be a DBN which contains several RBMs and the parameter used for feature extractions. The training in phase 1 is unsupervised learning.

## Phase2: Fine-Turning

A convenient classifier will be placed at the end of the DBN using the Contrastive Divergence CD algorithm and the feature extraction to build the FFNN, which can be a BP or the use of the DBN Softmax classifier.

The main uses of DBNs include feature extraction, image generation, clustering, recognition.

- **Autoencoder (AE)**

Also named the Autoassociator, the Autoencoder is unsupervised learning and an FFNN which consists of three main components, namely an encoder, code, and a decoder. After the data are reduced in size in the encoder, also called **latent-space representation**, it presents the code to transmit to the decoder which rebuilds the input data using this code to transmit the code.

Autoencoder (AE) can be used for:

- Reducing data dimensionally;
- Data compression;
- Reforming data corruption; and
- Avoiding local minimums using a pre-training deep network.

### 2.9.4 Convolutional Neural Network (CNN or ConvNet)

The main drawback of earlier neural networks were as follows:

- Large numbers of trainable parameters which were difficult to handle even by a GPU, especially when dealing with high-dimensional inputs (RGB);
- Insufficient or no-invariance shifting or scaling or other types of distortion; and
- Ignoring the topology of any input data such that there were no relations between the pixels.

With CNN, considered to be a revolution in computer vision with mathematical operations, the main negative aspects that have been resolved using CNN are:

- Invariance using scaling, shifting, etc.
- Solving the overfitting issue by limiting the numbers of weights using parameter sharing; and

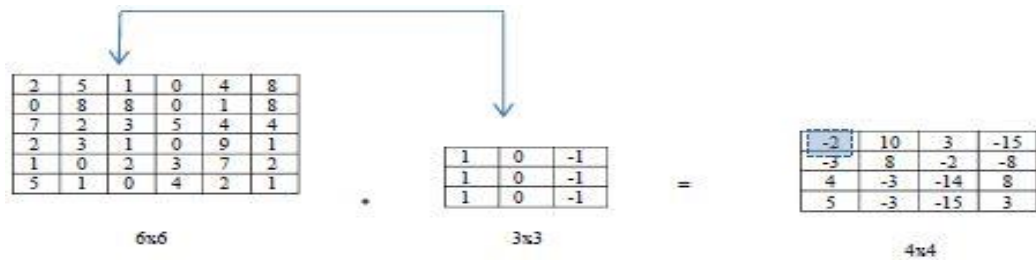
- Making use of the spatial proximity of the pixels by making connections between neurons that are similar to neighboring pixels.

The CNN classifies directly from picture elements with minimum treatment and best performance using small parameters due to parameter sharing and lower connections. In many studies, CNN has shown high accuracy and achievement in image classification due to automatic feature extraction with lower image pre-processing as well as parameter sharing and sparsity of connection.

## 2.9.5 Main Operations of Convolution

### 1. Edge Detection

Edge detection is the operation of specifying a method of finding the vertical or horizontal edges in images using a filter (kernel).



**Figure 8.** Edge detection using a convolution operator

According to the **Figure 8.**, a 3\*3 kernel is constructed (also known as a filter or vertical detector) and a 6\*6 image is taken. The convolution operation (\*) occurs by applying the element product followed by summation. The first element on the top right corner will be:

$$2 * 1 + 5 * 0 + 1 * -1 + 0 * 1 + 8 * 0 + 8 * -1 + 7 * 1 + 2 * 0 + 3 * -1 = -2$$

By shifting the blue box one step to the right and performing the same convolution operation, we obtain the second element on the 4 \* 4 matrix. This is performed on every element.

## 2. Padding

Padding is used to avoid the following factors:

- Images shrinking when applying the convolution operator every time by using edge detection; and
- Throwing away a large quantity of information near the edges of the images.

To specify the dimension of output matrix we used the following equation:

$$(n + 2p - f + 1) \times (n + 2p - f + 1) \quad \text{Eq. 47}$$

where  $n$  is the input image dimension,  $p$  the padding and  $f$  the filter.

In order to verify whether padding is to be used, we use:

- **Valid convolution:** there is no padding and the output dimension will be:

$$(n - f + 1) \times (n - f + 1) \quad \text{Eq. 48}$$

- **Same convolution:** the output size is equal to the input size:

$$p = \frac{f - 1}{2} \quad \text{Eq. 49}$$

## 3. Stride

Stride refers to the number of steps that shifts the kernel. The output dimension is:

$$\left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right) \quad \text{Eq. 50}$$

where  $n$  is the input image dimension,  $p$  the padding and  $f$  the filter.

The main advantages of the stride include the fact that a large stride would mean less extension in the output dimension that leads to a smaller output size to the next layer with lower computation and memory allocation.

### 2.9.6 Types of layers in a CNN

- **Convolutional Layer**

The core of CNN is the Convolution layer which represents a mathematical part of the network by applying element-wise and addition operations using height, width and depth of the input image (respective fields).

The main hyper-parameters of Convolution layer are:

- Size and number of filters;

- Padding;
- Stride; and
- Activation function (linear and non-linear).

The purpose of earlier Convolution layer is to detect low level feature such as edge, line, etc., where the deeper Convolution layer used for high level feature detection such as shape.

The depth refers to RGB or number of channel and Filter(s) (kernel) is a square matrix consist of weights that slide on each region and the output is the feature(s) map which will be as input to next layer.

We take the image matrix and kernel matrix and applying the element-wise operation, as shown in **Figure 12**, where we have a simple image with 2 dimensions (height and width) with a  $6 \times 6$  dimension, and filters (height and width) with a  $3 \times 3$  dimension. We then apply the convolution operation (\*) to both and we have one features map with an output dimension being a  $4 \times 4$  volume:

$$(n - f + 1) \times (n - f + 1) = (6 - 3 + 1) \times (6 - 3 + 1) = 4 \times 4$$

This output will be the input into the next layer.

Therefore, in the conv layer, the size of the input image decreases from  $6 \times 6 \times 1$  to  $4 \times 4 \times 1$ . Applying this operation many times may lead to losses in much image information, so we used padding.

#### **Filter Types:**

- Sobel filter which can apply edge-detection
- Filters to blur images
- Filter to sharpen images
- Schoss filter for v – edge detection

#### **▪ Non-linearity layer (activation layer)**

This layer receives the stack of the feature map of the Conv layer and applies the non-linearity operation by adding bias and applying the activation function used for CNN (such as Softmax, ReLU, ELU or sigmoid).

$$y^{[1]} = w^{[1]}a^{[0]} + b^{[1]} \tag{Eq. 51}$$



- **Pooling Layer**

The pooling layer is layer added after convolution layer so that to order the layers within a convolutional neural network that can be replicated in a given model one or more times. To create a new set of the same number of pooled feature maps, the pooling layer operates separately on each feature map.

There are two types of pooling layer (avg-pooling and max-pooling). Max-pooling is widely used and by applying the max pool, we slide the window of  $2 \times 2$  regions and take the max value on each part and place them into the new matrix. Average pooling takes the average of the window of  $2 \times 2$  regions.

- **Fully Connected**

The final layer of the CNN is responsible for the classification task by taking the output matrix (feature map)  $S \times S \times n$  from the previous layer and converting it into a vector ( $S \times 1$ ) so that each pixel in the matrix represents the neuron value of the vector that is fully connected to the next hidden layer and with the previous layer. This is followed by applying an activation function such as Softmax for multiclass classification to indicate the probability for each class, or such as sigmoid for binary classification to predict to which class the images belong.

- **Batch normalization layer**

BN (internal covariate shift) is used to reduce the variety of distributions in the input layer during the training process using two parameters:

- Shifting, by subtracting the mean; and
- Scaling, by dividing the batch standard deviation.

The output will lie on  $[0, 1]$  or  $[-1, 1]$ . This accelerates the training process, reduces overfitting and increases the learning rate. This makes the training more stable.

- **Dropout Layer**

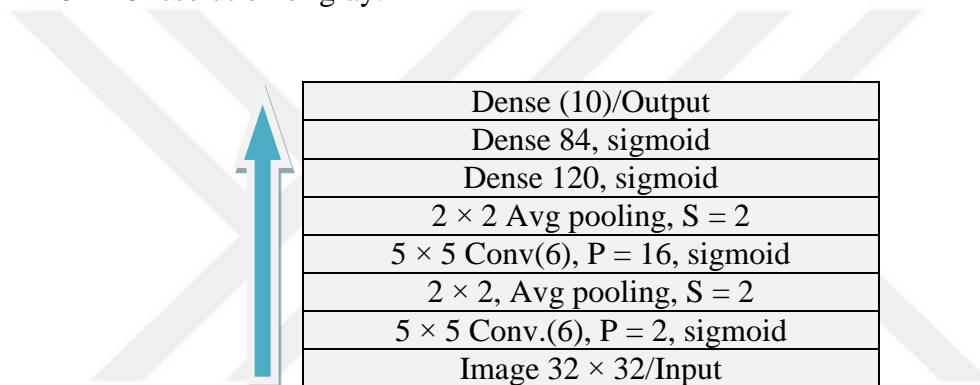
The dropout layer is used to ignore or drop neurons and their connection out from hidden layers with probabilities of  $p$  so as to reduce overfitting and avoid co-adaptation feature detection whenever some neurons are detected with the same feature.

Therefore, it decreases the connections between the previous and following layer.

### 2.9.7 Types of convolutional neural network architectures

- **LetNet-5**

The first CNN architecture was introduced in 1995 by Yann LeCun et. al [10]. The main goal was to use Gradient-Based learning algorithms to classify pattern recognition of low dimensions and with less computational expenditure. The focus was to build an automatic handwritten binary classification using the MNIST dataset, which consisted of 50,000 training images and 10,000 validation and test images in a  $28 \times 28$  resolution of gray.



**Figure 9** LetNet

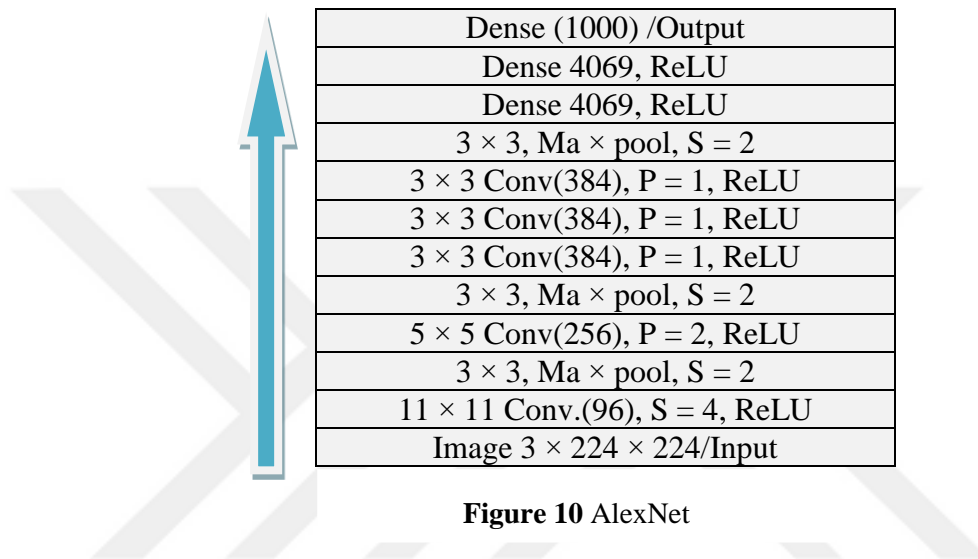
By building a network with eight layers, as shown in **Figure 9**, the main idea of this network is to minimize the height and width when we go deeper, while the number of the channel (feature map) is maximized. Additionally, following every pooling layer, there is a sigmoid non-linearity layer. Applying the weight sharing technique allows the minimizing of the number of trainable parameters. The total connections in LetNet-5 was 340,908 and 60,000 trainable parameters.

- **AlexNet**

A paper was published in 2012 by Alex Krizhevsky et. al [11] presenting a classification of 1.2 million high-resolution images into one-thousand different classes using in ImageNet dataset. The result was a 37.5% error rate in Top-1 and 17.0% in Top-5. The total was 60 million trainable parameters and 650,000 neurons. The main

aspect was using the following parameter:

- “Dropout-Regularization” so as to minimize the overfitting;
- Maxpooling instead of the Avergpooling layer; and
- Increasing the number of the convolutional layers with ReLU as the activation function so that it will increase the non-linearity through the depth of the network and solve the vanishing gradients.
- Using a small learning rate (0.01) and large image size ( $224 \times 224 \times 3$ ).



**Figure 10** AlexNet

Every dataset was collected using crowdsourcing on Amazon Mechanical Turk by building a network containing five convolution layers, three Max pool layers and three fully connected layers with 1000 neurons using Softmax as an activation function consisting of 12 layers.

This network was able to solve the overfitting problem using two methods:

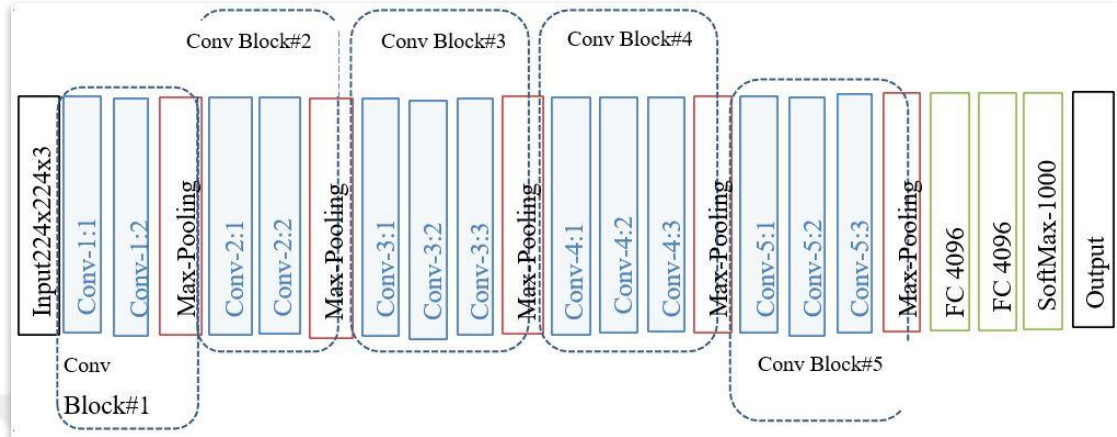
- **Data augmentation** with a label-preserving transformation; and
- **Dropout layer after each dense layer**, which assigns zero to the output of hidden layers with 0.5 probabilities.

#### ▪ VGG-16

Karen Simonyan and Andrew published a paper [12] which introduced a stack of convolution layers considered to be a new architecture of the convolution neural network which attempted to simplified the network using small receptive fields in Conv layers ( $3 \times 3$ ) with strides equal to one, where all Max-pooling sizes were  $2 \times 2$

and the stride equal to 2. This concept of the block is now used in many DL architectures where each block contains a stack of Conv.layers and Max pool layers followed by three fully connected layers, as shown in **Figure 11**.

**Figure 11** VGG-16 Architecture

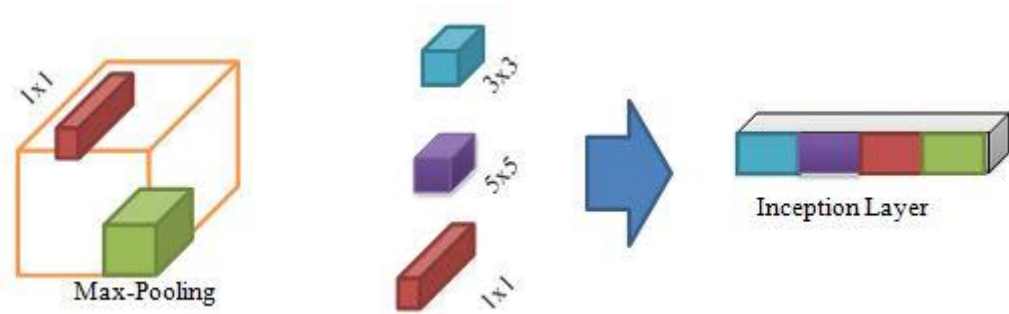


The aim of this network was to contribute to the competition of ImageNet 2014. The result was Top-1 val.error 23.7% and Top-5 val.error 6.8%. They indicate that using deep layers will increase the performance and accuracy of the network.

The name VGG-16 refers to the number of layers, here 16 (13 Conv-Layers and 3 FC-Layers). It has 128 million parameters to train and the main strength of this network is that it doubles the value of the channel in every block while the numbers of the height and width are decreased.

- **Inception/GoogleNet**

A paper was introduced to contribute with the competition of ImageNet on ILSVRC 2014 which achieved 93.3% top accuracy. The main idea was that instead of selecting a specific convolution layer with a specific filter size, the inception network could be concatenated more than the convolution layer and max-pooling layer with different filter sizes, followed by stacking them in one block known as the inception block, as shown in **Figure 12**. This reduces the number of parameters and computational tasks.



**Figure 12** Single Inception Block

The inception network was built using different numbers of convolution layers ( $1 \times 1$ ) and nine inception blocks, followed by centering and cropping the input image to  $224 \times 224 \times 3$ .

ReLU was used in the hidden layer, while Softmax was used in the output layer to specify the probability of the output for each class.

The optimizer that was used in the inception network was the stochastic gradient descent (SGD) equal to 0.9 momentums, and minimizing the learning rate to 0.04 every eight epochs. Another aspect used in inception network was the Average Pooling layer after the last convolution layer instead of a fully connected layer that reduces the number of parameters.

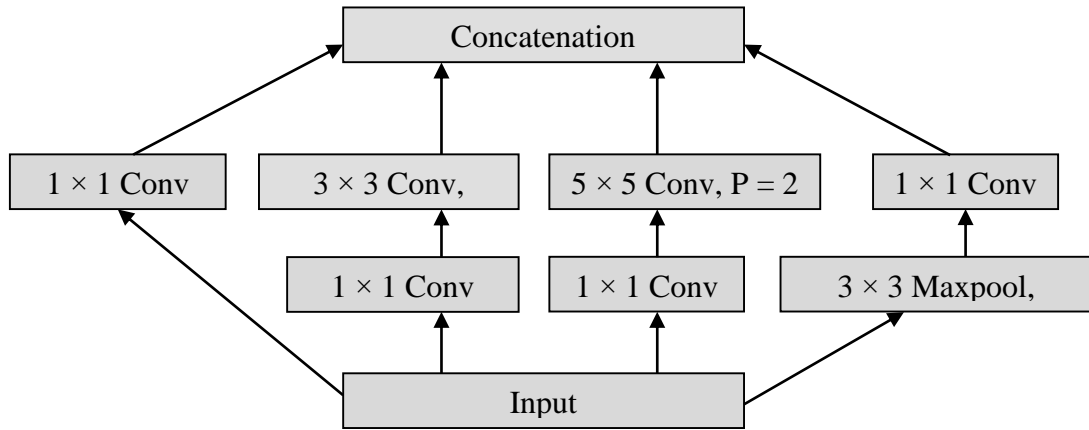
By increasing both the depth (number of layers) and width (number of units), it showed better performance in the accuracy of the network.

This method was inspired by two main concepts:

- Network in Network introduced by Lin et al. [14]; and
- The familiar sentence which was taken from the film “We Need To Go Deeper.”

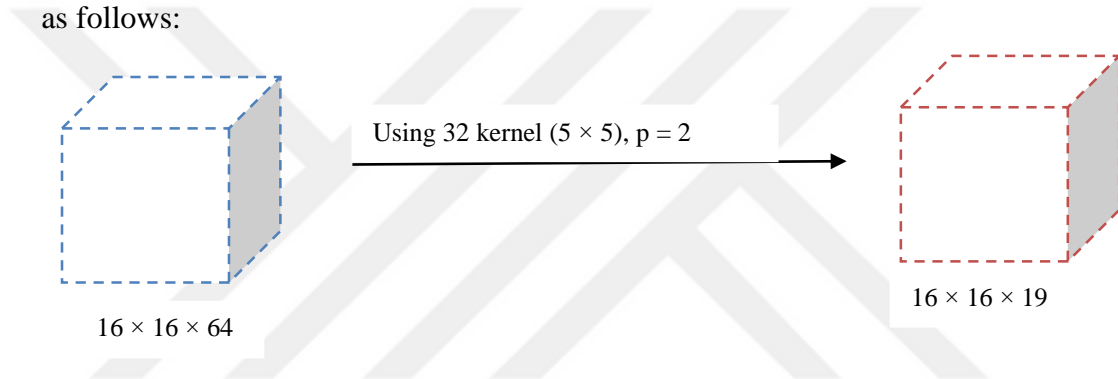
The main challenge in the increase of depth was the increase in the computational cost problem known as “bottle neck.”

In order to solve the bottle neck, conv.net ( $1 \times 1$ ) was applied by placing it between the layers, as shown in **Figure 13**.



**Figure 13** Inception module

We make a comparison between computational cost with and without Conv.net ( $1 \times 1$ ) as follows:



**Figure 14** Before using Conv.layer ( $1 \times 1$ )

$$\begin{aligned}
 \text{Computational cost in Figure 14} &= (N_h \times N_w \times N_{ch}) \times (F_h \times F_w \times F_{ch}) \\
 &= 16 \times 16 \times 64 \times 5 \times 5 \times 32 \\
 &= 13,107,200
 \end{aligned}$$

Where;

$N_h$  = height of input

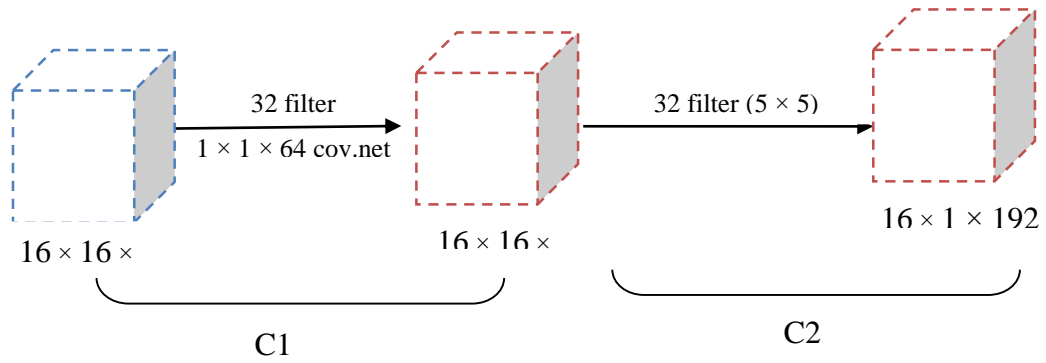
$N_w$  = width of input

$N_{ch}$  = number of input channel

$F_{ch}$  = numbers of filters

$F_h$  = height of filter

$F_w$  = width of filter



**Figure 15** After using Conv.layer (1 × 1)

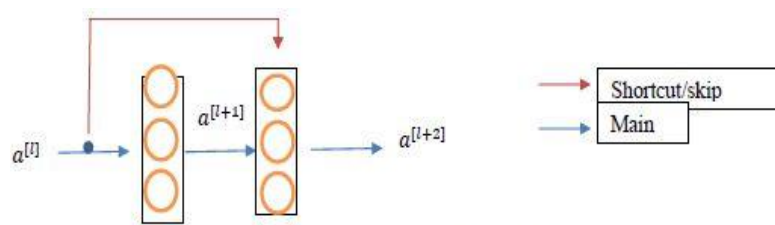
No. of parameters in **Figure 15** = C1 + C2

$$= (16 \times 16 \times 64 \times 1 \times 1 \times 64) + (16 \times 16 \times 32 \times 32 \times 5 \times 5)$$

= 7,602,176, which is much smaller than computational cost in **Figure 19**.

- **Residual Network**

In 2015, new research [15] described a new method which allows the building of a very deep neural network with thousands of layers with the ability to avoid vanishing and exploding gradient problems that lead to the degradation of the accuracy of the network using numbers of residual blocks. By stacking them, it will produce a residual network, each block consisting of a “skip connection/shortcut,” as shown in **Figure 19**, which takes an output of the layer and makes it an input to another deeper layer and skips a number of hidden layers (almost two hidden layers). ResNet on the ImageNet test set obtains 3.57% of the Top-5 error with 152 layers, where the size of the residual net (8× greater than VGG net). It achieved first place in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2015 classification competition.



**Figure 16.** skip connection

**Table 2** Computing the output of the residual block

Step No.	Input	Activation Function	Output
1	$a^{[l]}$	Linear	$z^{[l+1]} = w^{[l+1]} \cdot a^{[l]} + b^{[l+1]}$
2	$z^{[l]}$	Non-linear (ReLU)	$a^{[l+1]} = g(z^{[l+1]})$
3	$a^{[l+1]}$	Linear	$z^{[l+2]} = w^{[l+2]} \cdot a^{[l+1]}$
4	$z^{[l+2]}$	Non-linear (ReLU)	$a^{[l+2]} = g(z^{[l+2]})$

By using the skip connection, which is between linearity and non-linearity (before non-linearity), we skip steps 2 and 3 and compute the output as follows:

$$a^{[l+2]} = g(z^{[l+2]}) + a^{[l]} \quad \text{Eq. 52}$$

Using the Resnet network, the accuracy is increased while the depth is increased such that it contrasts with the normal network when increasing the depth of the network, which leads to decreasing the network accuracy.

- **Generative Adversarial Networks (GAN)**

In [52], Ian Goodfellow et al. (2014) introduce (GAN) as a new method to generate fabricated data from given actual input data.

GAN consists of two models that work as a competitive (Min (g)/Max (d) V (D, G)) game:

- **Discriminative Model:** This acts as a classifier that determines whether an image is generated from the input dataset or from the artificial generator (real or fake). Using supervised learning algorithms, the main objective of this model is to minimize the error rate.

We update the discriminator ( $d$ ) by updating the weights using a stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D(X^{(i)}) + \log(1 - D(G(Z^{(i)})))] \quad \text{Eq. 53}$$

- **Generative Model:** This is used to build a model for each class to make a classification for the new image by comparing it to the models and computing the

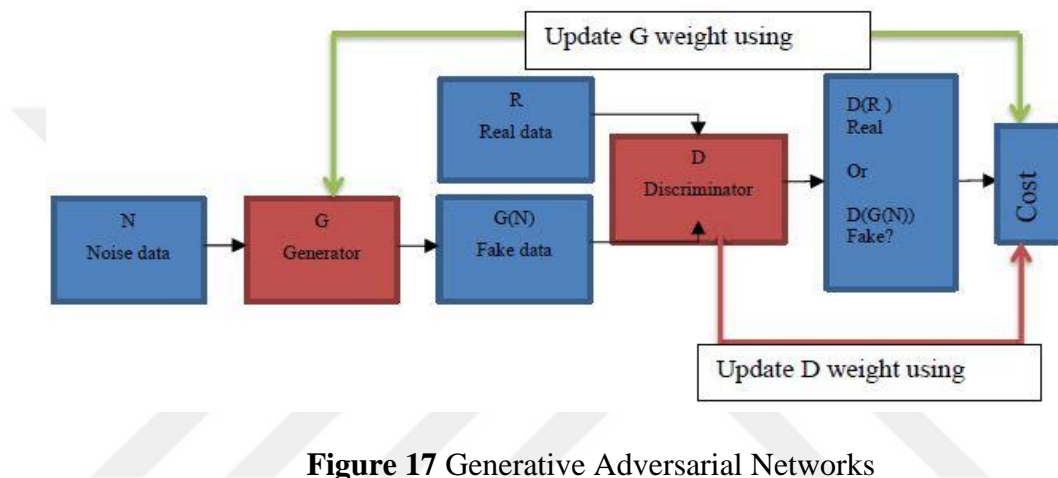


distribution for each by receiving noise data and generating an image with the objective of the high error rate of the discriminatory model.

We update the generator model ( $g$ ) by updating the weights using a stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(Z^{(i)}))) \quad \text{Eq. 54}$$

- Both models still improve each other until reaching the end of the result when the generator model creates images that are closest to real images; therefore, the discriminator cannot distinguish whether the image is real or fake.



**Figure 17** Generative Adversarial Networks

### Types of Generative Adversarial Network:

- 1- Deep Convolutional GANs (DCGANs)
- 2- Conditional GANs (cGANs)
- 3- StackGAN
- 4- InfoGANs
- 5- Wasserstein GANs (WGAN)
- 6- Discover Cross-Domain Relations with Generative Adversarial Networks (Disco GANS)

### GAN Applications:

- Text-to-image synthesis
- Image-to-image translation
- Face aging

## CHAPTER THREE

### PREVIOUS WORK

Different methods and algorithms were used to classify diabetic retinopathy into two, three, four or five classes. Below are the main methods using different ML algorithms used for diabetic retinopathy classification.

#### 3.1 SVM for DR Classification:

**A. Biran et al.** [16] used the Star database with 33 RGB retina images with diabetic retinopathy, divided them into a group (23 training images and 10 testing images). They classified the images as Normal, Non-proliferative diabetic retinopathy and PDR using two main phases:

##### **Image Pre-Processing Phase:**

The Gabor Filter was used to detect Exs, while the Circular Hough Transform (CHT) was used to detect Hemorrhage regions, followed by 31 features being extracted and selecting only six features that would be used as inputs to the next phase (Green Channel, Contrast-Limited Adaptive Histogram Equalization (CLAHE), Gabor Filter, HE, Hemorrhages and the pre-processing step).

**Classification Phase:** Support Vector Machine (SVM) algorithms were used with a non-linear kernel function; the final accuracy was 91.4%.

**Arisha Roy et al.** [17] used fundus images from DIARET, DRIVE, and MESSIDOR. Two phases were applied in the research:

**Image pre-processing phase:** used in order to extract the main feature of DR and making them the inputs to build the model classifier using the support vector machine. The first feature was the Exudates (Exs), which is the main characteristic of DR. In order to detect exudates in the images, they extracted the green channel from the images and applied a histogram followed by specifying the optical nerve using

Convex-Hull and counting the optical discs.

The second feature was Neovascularization that specifies the phase of DR, by detected Retinal vessels using cascaded Gaussian and median filter, finally used the top-hat filter

**Classification phase:** both features (Exs, Neovascularization) were used as input to the classification network.

- First SVM classify the 100 fundus images into Normal and abnormal DR.
- Second SVM classify the input abnormal DR (40 images) into PDR and NPDR.

By using a kernel with type polynomial 2 the final accuracy was 96.23%.

**Wen Cao et al.** [18] used the DIARETDB dataset with 89 fundus images of people who were suffering from DR. The aim was to classify the images according to the detection of microaneurysms (MAs), which is the first feature to appear when people suffering from diabetes are affected by DR. Therefore, with 84 fundus images with mild NPDR, and 5 fundus images as normal. Three ML algorithms (Randal forest, Neural Network and Support Vector Machine) were applied:

- using image-patch with  $25 \times 25$  windows and defining label H1 as images with MA and H0 as an image without MA;
- applying the operation to the green layer of the RGB images as MA has a high variation in the green layer;
- applying the technique to reduce the dimensionality of the images using PCA followed by using data whitening in order to produce the final two dimensions.
- using two concepts for model evaluation (receiver operating characteristic curve ROC, Area under the ROC curve AUC) and F-feature using the weight average and threshold.

The SVM model showed a high accuracy of 98.5% for ROC and 92.6% of AUC using the radial basis function kernel.

## Summary

[16] used only 23 images and 6 features to train the SVM classifier. It showed less accuracy at 91.4% than [17], which used 100 images with two features in the first SVM classifier followed by using 40 images and in the second SVM classifier using the

kernel type polynomial-2 at 96.23%. [18] showed high accuracy at 98.5% by using 89 images with three sets of features using the radial basis as the kernel function.

The main drawback of SVM algorithms include:

- Overfitting problems due to noisy datasets;
- Weakness with large numbers of features (high-dimensionality);
- The requirement for large memory, expensive computation costs, and time-consumption in training operations; and
- The reduction of transparency due to the extraction rule in the SVM during the beginning stages.

### 3.2 BP for DR Classification

**Abu Abraham** [19] used the Diabetic Retinopathy Debrecen Data Set from the ML repository with 1151 images each including twenty attributes.

The goal of the model is to classify the images into two classes (DR and not DR) using BP algorithms with several techniques to improve the accuracy of the network:

- the genetic algorithm to select the appropriate hyper-parameter;
- the distinctiveness, pruning technique to minimize the number of hidden layers using the angle of the cosine value and to remove the similar hidden layers;
- mini-batch, with which to escape from local minimization;
- normalization and standardization to deal with the distribution of the data set; and
- the application of early-stopping to indicate the number of the period with the minimum loss function.

The model was inspired by the natural recreation process that makes the best selection in every new creation and drops the weak chromosome (hyper-parameter).

Additionally, the factor of the performance of the hyper-parameter is the accuracy (fitness function). The outcome was an A model with an accuracy of 74.8% using 467 hidden layers, 375 epochs, a 0.00158 learning rate, three batches and the use of Adam as the optimizer.

**Mingli et al.** [20] used the DIARETEDB0 dataset standard diabetic retinopathy database calibration level-0. The first operation was applied to image processing and the second operation was a feature extraction finally applying the classification

method.

In the image processing, the main task is to obtain the value of the channels (red, green, blue), the result of which was the value of the green channel being higher than the others so that the later operation would be on it;

- Histogram Equalization;
- Gaussian filter;
- Wiener filter; and
- Gray scale invert.

The feature was prepared using feature extraction using the canny algorithm and gradient magnitude to separate the BV.

They used the threshold algorithm on the separated optic disc and applied the BP algorithms using nine features and using two techniques to avoid the local minimum thus:

- Using momentum to improve the accuracy by enhancing the weight update; and
- Using the adaptive learning rate to accelerate the training operation with 5,000 epochs with four neurons in the input layer, 12 neurons in the hidden layer, and one neuron in the output layer.

The final accuracy was 96% using 70 images for training and 50 images for the test.

**Mohammed A. Al-Jarrah et al.** [21] used the DIARETDB dataset with 89 RGBs of  $1500 \times 1152$  resolution to apply different classification methods, namely the Bayesian Regularization of Neural Network, Backpropagation and Resilient Backpropagation. The main aim of their studies was to classify those input datasets into four classes (normal, mild, moderate and severe). 70% were considered the training set and 30% were consider the testing and validation sets.

The entire operation was divided into two phases:

#### **Image Processing Phase**

- Using a median filter and histogram equalization in order to clear away any noise
- Removing optical discs using special algorithm introduced in [53]
- Using morphological operations to detect Exudates (EX)
- Removing blood vessels using a modified algorithm of Maheswari & Punnolil

- An edge detection operation using Maheswari & Punnolil's modified algorithm
- Using the size of the object as a factor for split hemorrhages (HA) and microaneurysms (MA).

### **Classification phase:**

During this phase, the dataset is classified into 4 classes (according to the EURODIAB grading system):

- Two algorithms used Resilient Backpropagation, Bayesian Regularization Backpropagation with one hidden layer which had 8 neurons;
- Using 8 features that distinguish the DR disease (No. of EXs, Area of EXs, Avg intensity of EX, No. HA, Area of HA, the extent of HA, No. of MA) as input features to the input layer and one hidden layer with 8 neurons.

The accuracy of the BRBP algorithm was 96.6% and the accuracy of the RBBP algorithm was 89.9%.

**Fikirte Girma Woldemichael et al.** [22] used the PIMA dataset with 768 images, using data-mining methods and an artificial neural network algorithm for classification between with or with-out retinopathy diabetics.

The first step was to prepare the dataset by handling any missing data and inconsistent data; no noise was located in the dataset.

Then the form of the dataset was converted in order to apply a data mining operation using min-max normalization, after which a feature extraction technique (Chi-Square) was applied and which obtained eight features that were the input neurons to the Backpropagation neural network.

Finally, the BP network was built with three layers:

- an input layer with 8 neurons;
- one hidden layer with 6 neurons; and
- an output layer with one neuron.

The main technique that was used to improve the performance of the BP algorithm was the use of the GD algorithm to compute the error of weights and 5-fold cross validation for training and evaluating the model and increasing the learning rate. The results were 83.11% accuracy using three iterations. These methods yielded the best

results compared with another algorithms (SVM, Naive Bayes, J48) that were used in the same research.

**Jesús Salvador Velázquez-González et al.** [23] used the Techno-vision dataset containing 216 RGB fundus images in TIFF format of  $3504 \times 2336$  resolution. They applied three main operations (image pre-processing, fuzzy logic, and the artificial neural network algorithm) to classify the dataset into four classes (normal, light non-NPDR, moderate NPDR, and severe NPDR), using two phases:

**Image pre-processing phase:** This is the application of the normalization method to adjust the size of each image to  $720 \times 720$ , and to split each channel (red, green, blue), color-decomposition was used, after which the color images were converted to gray images via image-enhancement and making every minute detail clear to human vision with the intensity technique. The last operation in the image pre-processing was the use of histogram-equalization in order to obtain two contrasting gray scales.

▪ Eye-retinopathy is homogenous; therefore, segmentation techniques were proposed by (Canny, Otsu) for this recognition phase. With the advice of an ophthalmologist, the main categorization of eye diabetic retinopathy disease is:

- Blood Vessels (BV)
- Microaneurysms( $\mu$ Ans)
- Hard Exudates (HE)
- Homogeneity
- Entropy.

Therefore, extracting BV, HE and  $\mu$ Ans features occurs using feature extraction methods and the fuzzy logic operation (AND operation) between two sets of images after image processing to detect each of the Blood Vessel, hard exudate and microaneurysms.

Texture analysis is used to measure entropy and homogeneity on the Level Co-occurrence Matrix (GLCM).

**Classification phase:** The Backpropagation neural network was built in four layers as follows:

- an input layer with 5 neurons (number of input features).
- two hidden layers with each consist of 10 neurons (obtained using Kolmogorov theory).
- an output layer with four neurons to classify the input feature into four classes (Normal, Light NPDR, Moderate NPDR, and Sever NPDR).

With 290 iterations and 0.001 MES, 143 training datasets and 73 testing datasets, the final accuracy was 92%.

With a small number of neurons in the hidden layer, a large number of features and by the increasing the number of iterations or epochs, it showed good performance in terms of accuracy.

### Summary

[19] used one hidden layer with 467neurons and applied 375 using 20 features as inputs. It obtained a low accuracy of 74.8%. [20] used one hidden layer with 12 neurons, 9 features and 5,000 epochs, which had better accuracy at 96%. On the other hand, [21] used two algorithms (BRBP and RBBP) with one hidden layer which had 8 neurons, 64 training images and 8 features being classified into 4 classes and obtaining accuracies of 96.6% for BRBP and 89.9% for RBBP, while [22] used 8 features as inputs, one hidden layer with 6 neurons, an output with one neuron and applying only 3 iterations; it showed a low accuracy of 83.11%.

Finally, [23] used fuzzy logic combined with an ANN. The combination of these two methods require a fine-tuning technique to solve the main limitation of their studies.

The main problems of this algorithm were that:

- It was delicate with noisy data;
- It had slow operations and could fall into local-minima; and
- The achievement of the algorithm relied on input data.

### 3.3 CNN for DR Classification

**Pavle Prentašić et al.** [25] using 50 GB of DRIDB fundus images built a model which was capable of classifying each pixel from these images as “with Exs” or “without Exs,” considering the exudate as being the primary characteristic of DR disease by taking coordinate achievements for both optical disc detection and exudate detection.



The CNN architecture had 10 layers except for the input layer which used the raw pixel severity of source images as the input to the network with  $65 \times 65$  input maps using the green layer of the color images:

- Four Conv.layers, each followed by one Max-pooling layer using ReLU as the activation function.
- Two FCs, using Softmax for classification with 2 neurons on the output layer.

The time consumption was ten hours for the training process. The F-score was 0.77.

**Shuang Yu et al.** [26] used the E-Opha data set with 82 images and applied a pixel-wise operation for exudate detection with the Ex being the main feature of the DR.

Two main phases were used, as follows:

#### **Image pre-processing phase**

- Delete Optic Disc by using the algorithm of the local phase symmetry;
- Delete Retinal Vessels using segmentation followed by inpainting operations; and
- Using Ultimate Opening (U.O.) operator to specify pixel level competitors of exudate images using the  $64 \times 64$  patch.

#### **Classification phase using CNN**

With 14 layers (7 Conv-layers, 4 Max-pool layers, 2 FCs and 1 Dropout layer), the primary Conv-layer is used to extract elementary features (edges detection) using ReLU as the activation function, which was in contrast to the deeper Conv-layer which was used to extract high-level features. The Max-pool layer was used to shrink the output dimension of the Conv-layer and reduce the computational time. The final Max-pool layer was followed by the FC layer with 64 neurons and to solve the over-fitting problems, they used Dropout between the FC layers. Here, Softmax was used with two neurons as the activation function.

The experiment of the study was implemented using Theano and Lasagne library. After 150 iterations with 47 images with Exs, and 55 images without, the final accuracy of the experiment recorded 91.92%. The limitation in these studies is that the dataset is too small and unfamiliar.

**Avula Benzamin et al.** [27] presented an important feature of DR in the early stages as the appearance of a Hard Exudate. They used CNN on an IDRiD dataset collect from three different location (Nanded, Maharashtra, and India) for Hard Exudate (HE)

recognition. They apply image segmentation with the use of 32-by32 patch of fundus images consisting of approximately 200,000 patches which was divided into two classes. These two classes consist of:

- 100,000 patches with HE.
- 100,000 patches without HE.

The operation was focus on the center of the fundus images, which was labeled “1” when it indicated the sensitivity of HE, otherwise was labeled “0.”

Using eight conv.layers, except for the last convolution layer, each one was followed by two pooling layers and three fully connected layers.

The Batch Normalization technique was used for quick learning operations and the problem of over-fitting was solved using the Dropout layer. For activation, the ReLU function was used and cross-entropy was used as the loss function and Adam as the optimizer at a 0.0001 learning rate.

In this paper, the researchers used “streak” words to refer to all images that were trained in the network. Therefore, by dividing the batch into five, the first 3 batches were trained one by one, each batch with 40,000 images and 500 iterations were applied. The accuracy was 99.4%.

By training all 54 images (40 fundus images as the training dataset, 14 fundus images as test dataset), each image was resized to a resolution of  $256 \times 256$ . The final accuracy obtained in this paper was 98.6%.

The main weak points of this paper were that it focused on the center of the patch and ignored the edges and that it used a low number of images.

**Manaswini Jena et al.** [28] used high-resolution fundus images to build a model capable of image classification into three classes (DR patients, Glaucoma patients and normal patients) with 15 images for each class. The HRF images were divided into 50% test images and 50% training images with 17 layers building CNN, as follows:

- 6 Convolution Layers each followed by a ReLU layer, except for the last layer which used the Softmax activation function.
- Five convolution layers
- Five max pooling Layers
- One fully connected layer using Softmax as the activation function.
- The fully connected layer was not used in this research because it slow the

speed process of the network model.

There was no need to perform the feature extraction operation because with CNN, there was an automatic feature extraction and that was the model used in this research as well as the value of the weight being able to be fine-tuned.

The accuracy was increasing with the increase of the iteration with the final accuracy being 91.66%.1

**Ratul Ghosh et al.** [29] applied CNN using the KAGGLE dataset with 30,000 fundus color images, a high-end GPU and the theano platform to classify the input images into two classes (DR or no DR) as well as into five classes (normal, mild, moderate, sever and PDR).

The main steps that were taken in this research can be explained as follows:-

- using image pre-processing to change the sizes of the images to a smaller size from  $3000 \times 2000$  pixels to  $512 \times 512$  pixels using image resizing.

#### **Image pre-processing**

- Image augmentation (float, scale, rotate, etc.) used to increase the numbers in the training dataset
- Image-Normalization to render all images to a similar scale
- Applying Non-Local Means Denoising (NLMD) for noise removal

#### **Building the model**

- Using CNN with 13 convolution Layers, 5 Maxpool layers, 3 dropout layers, 2 max-out layers and 1 Softmax layer, each convolution layer and fully connected layer was followed by PReLU as the activation function, except for the last dropout layer which used Softmax as activation function.
- Using Xavier initialization to predict the proper value of the weight automatically
- Using Dropout as the normalization technique to save the effective neurons
- Using the Nesterov accelerated gradient technique

The accuracy was 95% between two classes and 85% between five classes.

**Shorav Suriyal et al.** [30] used a dataset of 16,798 color fundus images from KAGGLE in order to classify them into two classes (with DR and without DR) by building a model which was able to be used on an Android platform on a mobile device.

After the dataset was gathered and set up, two main phases were applied:

**Image pre-processing:** removing noise using “Box Blur” and fitting all images into  $256 \times 256$  pixels.

**Building the model:** which is inspired by Mobile-Net and used the transfer learning with 25 convolution layers.

The main aspect that was used in this network had split the network to “Depth-wise,” which used one filter, and “Point-wise,” which used the linear activation function.

The Linux OS was used for the training and testing operations, while the Windows OS was used with the Android platform. The final accuracy was 73.3%.

**Darshit Doshi** [31] used the dataset that had been used in this paper taken from Eye-Pacs with 5,000 RGB fundus images being used as testing images and 72,126 images being used as training images.

The main goal was to classify the input images into 5 classes corresponding to the stages of DR disease. Two phases were applied:

**Image pre-processing**

- 1- Reducing the scale of all images to a fixed resolution of  $512 \times 512$  pixels
- 2- Changing the three RGB layers into a single layer (green) according to the contrast of the green layer on which all operations focused
- 3- Using histograms, image-normalization and image augmentation

**Building the model:**

With 12 Convolution layers, 5 Max-pool layers, 6 Dropout layers, 2 hidden layers, 2 fully connected layers and 1 non-linearity layer, the sigmoid was used as the activation function and the Convolution layer and hidden layer used the leaky-Rectifier as the activation function.

Nesterov-Momentum was used to reduce the loss function, where Glory-style was used to initialize both the weights and biases.

Three models were built with small improvements for each using kappa scores of 0.301, 0.35 and 0.38. The final score with the three models altogether was 0.39.

**Rami Safarjalani et al.** [32] used three datasets:

- 1- The HK data set with 189 images;
- 2- The SERI dataset with 32 SD-optical coherence tomography in  $512 \times 1024$  pixels;

and

3- The Srinivasan dataset.

Two main phases were applied:

**Image pre-processing operation:** to label the images followed by applying image normalization, removing noise from the images, resizing operation to  $256 \times 256$  pixels, and cropping any black pixels from the images.

**Building the CNN model:** using 5 convolution layers, 5 max-pooling layers, two fully connected layers, 1 dropout layer and Softmax as a classifier. The last FC layer had 4 neurons to classify into 4 classes (DME Diabetic Macular Edema, DR, DR/DME, and Normal). Using HK as input data and the grid-search-process to select the best parameter and weight initialization, Adam as the optimizer function with 50 iterations, and ReLU as the activation function, the final accuracy was found to be 90.5%.

Next, the model as a pre-trained model with the SERI dataset to binary was classified into two classes (normal and DME) using the same parameters. The accuracy was 99.02%.

The Srinivasan dataset was used twice as the test dataset with the model using HK as training data with accuracy of 80%. For the model using SERI as a training data, accuracy was 86.7%.

**P. Khojasteh et al.** [33] used a dataset from DIARETDB1 with 89 fundus color images in order to classify them as Exudate, hemorrhage, microaneurysm and background). The image patch used in this study is  $25 \times 25$  windows divided into two phases of process. These processes include:

- **Image-enhancement operation:** including both contrast-enhancement (CE) and contrast-limited-adaptive-histogram-equalization (CLAHE) and using them as a layer which was added to the CNN architecture known as “PPL” (pre-processing layer).
- **Building the CNN model:** With 4 convolution layers, 3 max-pooling layers, 3 normalization layers and 2 fully connected layers, the last fully connected layer contained 4 neurons corresponding to a number of the classes using SGD as an optimization algorithm. After 100 iterations, the accuracy was 87% and 90% using CE, CLAHE.

**Oscar Perdomo et al.** [34] used the Singapore Eye Research Institute dataset (SERI), which contained 32 SD-OCT (Spectral-Domain-Optical-Coherence-Tomography)  $1024 \times 512$ -pixel resolution images to classify into two classes (Normal OCT and DME OCT) where DME refers to Diabetic Macular Edema, building a model named “OCT-NET” using CNN architecture. Two phases were used:

**Image pre-processing:** included cropping and resizing to  $224 \times 224$  pixels.

**Building the model:** 4 blocks were used as follows:

- Ten convolution layers;
- Three max-pooling layers;
- Two fully connected layers, the last being Softmax as the activation function; and
- One dropout layer.

Using ReLU as the activation function in hidden layers, and grid search as the weight and parameter initialization methods and SGD as the optimization algorithm, the final accuracy was 93.75%, which is better than the 90% accuracy using VGG as the pre-trained model.

**Waleed M. Gondal et al.** [35], with small sized labeled images, used a weakly-supervised learning and CAM (class activation maps) technique to determine the region of the affected area in retina images that produce the main features and to enable the model to perform more accurately than traditional CNN architecture. They used two datasets: the KAGGLE dataset, which contained 88,702 color fundus (training and validating), and the DIARETDB1 dataset, which contained 89 color fundus images for testing.

Two main phases were used:

**Image-pre-processing:** including crop, resize ( $512 \times 512$  pixels), CH, and data-augmentation.

**Building the model:** the researchers used the CNN architecture of the o-O solution that was introduced by Antony and Brüggemann with several modifications such as changing the fully connected layer with the GAP (Global Average Pooling) layer where the final fully connected layer used two neurons (non-referable diabetic retinopathy and referable diabetic retinopathy). With 150 iterations, momentum as an optimizer and L2-regularization as the weight initialization technique, the final accuracy was 94.54%.

#### ▪ **Summary**

As we see in the previous method using CNN architecture to detect DR, [26][27][25] attempted to detect images and classify them into two classes (with Hard Exudate, and without) using binary classification. In [25], the final f-score was 0.77 using 50 color images and 10 layers using raw-pixels. There were more pre-processing images required in order to improve the performance. In [26], 14 layers and 150 iteration were used to obtain a 91.92% accuracy. Using an unfamiliar dataset again required more image pre-processing operations while in [27], 98.6% was obtained using 30 layers and Adam as an optimizer. This increased the performance of the network.

In [30], [35] and [42], CNN network was used with the binary classification using two classes (DR and NOT DR). [30] obtained 73.3% accuracy. The main weak point in this paper was that the pre-processing operation was too short. Many operations, such as histogram, data-augmentation, etc., needed to occur to enhance the accuracy. [34] used the binary classification in two classes, namely Normal OCT and DME OCT, obtaining accuracy of 93.75% using OCT-Net and four blocks of layers with each block containing 17 layers. [35] gave the best performance with 94.54% using a large number of data and two sets of images and the CAM technique.

[28] used CNN architecture to classify images into three classes at an accuracy of 91.66% without using the FC layer nor any image processing operations.

Classifying DR into four classes was carried out by [32] and [34]. [33] showed low performance at 87%, but 90% accuracy using CH, CIAHE, and using the SGD as an optimizer with 100 iterations. SGD is sensitive toward feature scaling that requires a large number of iterations. [32] showed good performance by obtaining 90.5% accuracy using 14 layers and 50 iterations and Adam as the optimizer.

[29] and [32] classified images into 5 classes, [29] with 24 layers and Xavier as the initializer technique. The accuracy obtained was 85%. [30] had several weaknesses:

- It ignored two layers (Red and Blue) that would lose some of the features; and
- It used three models that would cause computations to become more complex.

#### **3.4 InceptionV3 to classify DR:**

In [36], **Sarfaraz Masood et al.** used pre-processing images and the transfer learning

technique (Inception V3) to classify DR into five classes (No DR, Mild, Moderate, Severe and PDR) using the dataset that had been arranged by Eye-Paces on KAGGLE. The total number of images was around 800 for each class except for PDR, which numbered 708 images. The main image pre-processing operations are:

- Reducing the image size to a radius of 200 pixels;
- Highlighting the important features using the blue function; and
- Deleting the boundaries of the images.

In this study, they used transfer learning technique with qual weights and biases values for the training data. They also used same amount of data of DR with equal images size for each class and for the training of the model. The model in this study is of same architecture like that of ImageNet (Iception-V3), and used a Cross-Entropy as the loss function of the network. Furthermore, they observe that as the image size is increase to say 800 or more per each class, the radius size of the accuracy increases to 500 pixels. The recorded test accuracy in this experiment was 48.2%.

**Shikhar Srivastava** [37] used the dataset from KAGGLE to classify the RGB fundus images into three classes (Normal, Moderate DR and PDR).

Two phases were applied:

#### **Image pre-processing operations**

By using data augmentation, local-contrast-normalization, they reduced the scale and deleted the border of each image.

#### **Classification phase**

Transfer learning was used to make use of the Google-Inception-v3 model by changing the last layer with one FC layer which contained 3 neurons corresponding to the number of classes.

This was followed by using the inceptionism technique, which made use of the features that were learned by the network with visualization.

Finally, with 100 training images, 4000 iterations, a  $10^{-2}$  learning rate and using 200 testing images, the accuracy was 80%.

**Xiaoliang Wang et al.** [38] used 166 high-quality color fundus images from the KAGGLE dataset to classify images into DR stages (5 classes). The images dataset consists of noise. Therefore, in this research they resize the images into three resolutions in order to fit into their CNN model architecture. These include:



- AlexNet: resizing the images to  $227 \times 227$  pixels
- VGG-16: resizing the images to  $224 \times 224$  pixels
- Inception-V3: resizing the images to  $299 \times 299$  pixels

They also adopt a different transfer learning technique on large number of images in order to make use of the weight and bias parameters in their experiment. This can also be used again for a similar task.

5-fold cross-validation was used to split the source images into training images, test images and validation images.

Finally, the accuracy was as follows:

- AlexNet: 37.43%
- VGG-16: 50.03%
- InceptionNet-V3:63.23%, which was the highest accuracy using the fine-tuned and SGDM optimizers to reduce the loss function.

The main weak point in this paper was that the resizing operation to the original image would cause losses of some features of the images, and the number of images used in the study was small compared to the source numbers.

**Saboora Mohammadian et al.** [39] used a dataset from KAGGLE with 35,126 color fundus images to classify them into two classes (with DR and without DR) using binary classification. Two phases were applied:

- A pre-processing operation which included all images to be of the same size and changing the average of the images to 0.5 gray and cutting out the boundary as well as using data-augmentation.
- a classification phase which used two pre-trained models.

They used Inception-v3 and xception deep learning models for their sudy. In the experiment, three distinct operations were performed:

- Unfreezing the last 4 blocks from each model (Inception-v3 and xception ) and using fine-tune, showing a decrease in the accuracy
- Freezing all the blocks and retraining them using the FC layer with two neurons, which showed less accuracy
- Finally unfreezing the last two blocks and applying the fine-tune operation and using ReLU as the activation function and Adams as the optimizer function as

well as using 20% of all the images as testing images which increased the final accuracy to 87% using the Inception-v3 pre-trained model.

### **Summary**

In [36] [38] images were classified into 5 classes corresponding to DR stages. In [36], a lower accuracy of 48.2% was obtained, while [38] showed performance with an accuracy of 63.23% using several techniques, including fine-tune and the SGDM optimizer.

### **3.5 VGG for DR Classification**

**Arkadiusz Kwasigroch et al.** [40] in their research paper used the dataset from eyepaces.com containing about 88,000  $1500 \times 1500$ -pixel color fundus images to classify them into five stages of DR.

To have all the training images equal in number, they used under-sampling and over-sampling and split the dataset into three groups (3,500 training images, 1,000 validation images and 1,000 testing images), after which two main phases were applied:

#### **Image Pre-processing**

- 1- Resizing every image to a fixed size of  $224 \times 224$  pixels and fixing radius of every eye;
- 2- Modifying the image scale between 0 and 1;
- 3- Applying image-normalization with means equal to zero, and the variance equal to one.
- 4- Applying the local-average-subtracted operation.
- 5- Finally using the data-augmentation technique.

#### **Classification Phase**

This phase uses the transfer-learning technique and VGG-D model that was built using a vast dataset and changed the last layer of the network and using the same parameter and weight value and bias values and making the model used in a similar task.

The neural network in this study consisted of 5 Convolution layers followed by ReLU activation function for each layer, and subsequently Max or Avg Pooling layer. At the last layer, they add special FC layer and Dropout layer for the output neurons. The

final layer was a non-linearity layer using the sigmoid as the activation function with 4 neurons corresponding to the number of classes. The main aspect in this paper was that it used a special technique to label the classes as follows:  $[0\ 0\ 0]^t$ ,  $[1\ 0\ 0]^t$ ,  $[1\ 1\ 0]^t$ ,  $[1\ 1\ 1]^t$ . 200 images per class used the kappa-score as a measurement of the accuracy, which was 81.7%.

**Xiaogang Li et al.** [41] used two sources of color fundus images, namely DR1 containing 687 normal fundus color images and 327 abnormal fundus color images in  $640 \times 640$ -pixel resolution and MESSIDOR, which contained 1200 fundus images in different resolutions in order to build a binary model classifier (normal and abnormal). The two phases in their paper were as follows:

- Resizing every image to  $224 \times 224$  pixels and using image augmentation techniques; and
- Applying transfer learning using four models all of which were trained previously with a large number of images and used again in a similar task with a small number of images in order to solve the problem of preparing a special GPU device and for the long time required for the training operation.

The pre-training models were AlexNet, VGG-m (2048, 1024 and 128), and GoogleNet by changing the last layer of these models to binary classification and applying two main techniques (fine-tuning to the full layer and fine-tuning in a layer wise manner) with Gaussian-distribution as the weight initialization.

The output of the layers becomes an input feature to the SVM classifier with 1000 dimensions as input features using the Gaussian kernel and 30 iterations, where the learning rate decreased from 0.1 to 0.0001 using the MATLAB program.

Moreover, 5-fold cross-validation was used to measure the performance of this model. The results showed that VGG produced better results than AlexNet and Inception-v3 at 92.01% accuracy using the Messidor dataset, and VGG Net-19 at 94.12% accuracy using the DR1 dataset.

The main weak point in this paper was the small number of images used and the combination of the convolution layer and FC layer features, which could also improve the accuracy.

### 3.6 RESIDUAL.NETWORK for DR Classification

**Igi Ardiyanto et al.** [42] built a network named “Deep-DR-Net” in order to classify input fundus images into three stages (Normal, Mild NPDR and severe NPDR), and reduce sizes to  $320 \times 240$  and 237 images as a training dataset using three phases, thus: **First, they** constructed a feature map at the beginning of the network by using both the max-pooling layer and convolution layer to obtain 16 feature maps as an input to the next stage of the network.

**Second, they** built the coder using a set of convolution blocks in positions which would be influenced by CNN architectures (Res.net and inception net), using five cascaded Convolution Layers where ReLU and batch normalization were between the blocks at the end of the coder using Dropout layer and max-pooling layer at the beginning of the coder.

**Third, they** used softmax for classification after the second phase.

By using the FINdERS dataset with 315 fundus images and gathering “Deep-DR-Net” and SGD (stochastic gradient descent), the final accuracy was 60.82%.

**Syahidah Izza Rufaida et al.** [43] used 80,000 color fundus images that were included in Eye-Pacs. In their research, they used 35,126 training images and 39,424 testing images to classify DR into 5 classes (normal, mild, moderate, severe and PDR). Two main phases were used:

#### **Image pre-processing:**

Resizing the images to  $128 \times 128$  pixels, removing the noise, and finally using image-augmentation techniques.

#### **Building the model classifier**

The model was built using Res-net with 8 convolution layers, 2 max-pool layers, 2 dropout layers, 2 dense layers, 1 RMS pool and 1 feature pool.

A leaky rectifier was used as the activation function, Adagrad as the optimizer function and to reduce the loss function, the L2-Ridge (Linear Regression) was used. The kappa score was 0.5104 after eight to ten hours of the training operation.

The main weak points in this paper were their disregarding of the channels of the BV and EXs and using too few pre-processing operations.

### 3.7 AlexNet for DR Classification

**FengLi Yu et al.** [44] in their paper suggested a new method in order to detect images of high quality labeled as 1 and images of low quality labeled as 0. The main idea of this paper was to use two sources for feature extractions.

**Saliency-maps**, which define the important regions in images that fascinate human vision. This operation is performed after resizing the images to  $256 \times 256$  pixels.

**Pre-trained model** (AlexNet), which is a model with fine-tuning and several changes as a feature extraction tool.

After merging the results of the two sources of the feature extractions and normalizing these features, they are input into the SVM classifier by replacing the final layer of pre-trained AlexNet model with an SVM as a classifier which uses the Radial basis function as a kernel.

By using the KAGGLE dataset with 3,000 training color fundus images and 2,200 testing color fundus images, the final accuracy was 95.45%.

**Henok E. et al.** [45] in their paper used the AlexNet architecture with several changes in order to build RetiNet to classify the input image with Diabetic Retinopathy (DR) and Age-related Macular Degeneration (AMD), using three datasets, thus:

- the KAGGLE dataset with 35,126 color fundus images;
- the Messidor dataset with 1,200 color fundus images; and
- the UCH-AMD dataset with 197 color fundus images.

Two phases were applied:

**Images pre-processing:**

This is the use of a noise removal operation and a resizing to  $512 \times 512$  pixels and the CH process as well as other image-augmentation techniques.

**Classification phase:**

Here they build the RetiNet model which was AlexNet with several changes by creating two blocks, namely:

- “Net-A,” which contains two Convolution Layers each of which is followed by a Max-pooling layer; and
- “Net-B,” which contains three Convolution Layers, the last of which is followed by a Max-pooling layer followed by fully connected layer.

By using 62,578 training images and 1,060 testing images, 60 iterations, momentum as optimizer function and ReLU as the activation function, the final accuracy was 99.875%.

### 3.8 AUTOENCODER for DR Classification

**Juan Shan et al.** [46] downloaded the DIARETDB dataset with 89 color fundus images in order to classify the images with MA (positive instance) or without MA (negative instance). They split these images into 80 training images and 9 testing images.

Two small image processes were applied using a single channel and the images were scaled over [0, 1]. For high level automatic feature extraction, they used a stacked-sparse-autoencoder, which is the approach of DL. Using two hidden layers afterwards, they used Softmax as a classifier and classified them into two classes. The input was a patch of the images with raw pixels and 100 iterations using (SSAE and SMC) and fine-tuning the process to increase the accuracy to 91.38%.

The weak point in this paper was their use of a small numbers of images and the lack of variety in the autoencoder structure.

**Table 3** Comparison between previous studies

Reference	Source of Dataset	Method	Size of Image	Number of Images	Accuracy
<b>Two Classes</b>					
[17]	DIARET & DRIVE & Messidor	SVM			96.23%
[18]	DIARET	SVM	Image patch 25 × 25	89	ROC = 98.5% AUC = 92.6%
[19]	Debrecen	BP	720 × 720	1151	74.8%
[20]	DIARET	SVM	Image patch 25 × 25	120	96%
[22]	PIMA	BP		768	83.11%
[25]	DIARET DB	CNN	1 map of 65 × 65 neurons		f-score0.77
[26]	E-Ophtha Ex	CNN	64 × 64 patch image	82	91.92%
[27]	IDRID	CNN	256 × 256	54	98.61%
[29]	KAGGLE	CNN	512 × 512	30,000	95% ; 85% for five class
[30]	KAGGLE	CNN	256 × 256	16,798	73.3%

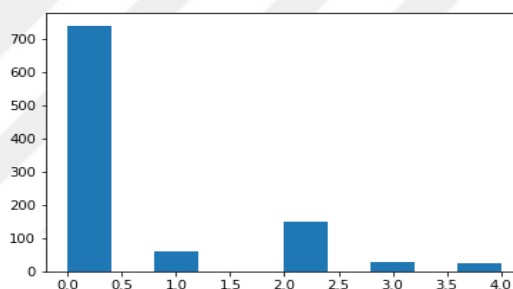
[34]	SERI	CNN	224 × 224		93.75%
[35]	KAGGLE & DIARETDB	CNN	512 × 512	88,702 89	94.54%
[39]	KAGGLE	InceptionV3		35,126	87%
[41]	Messidor & DR1	VGG	224 × 224	1014 1200	92.01% & 94.12%
[44]	KAGGLE	AlexNet	256 × 256	5200	95.45%
[45]	KAGGLE & Messidor & UCH-AMD	Alex-Net	512 × 512	62,578	99.875%
[46]	DIARETDB	Autoencoder		89	91.38%
<b>Three Classes</b>					
[16]	Star DB	SVM		33	91.4%
[28]	HRF	CNN	584 × 876		91.66%
[37]	KAGGLE	InceptionV3		100	80%
[42]	Finders	ResidualNet	320 × 240	237	60.82%
<b>Four Classes</b>					
[21]	DIARETDB	BP	11500 × 1152	89	96% BRBP 89.9% RBBP
[23]	Techno-vision	BP	720 × 720	216	92%
[32]	HK & SERI & Srinivasan	CNN	256 × 512	221	99.05%
[33]	DIARETDB1	CNN		82	90% CLAHE 87% CE
<b>Five Classes</b>					
[31]	KAGGLE	CNN	512 × 512	77,126	0.39 kappa score
[36]	KAGGLE	InceptionV3	500 pi × el	3908	48.2%
[38]	KAGGLE	InceptionV3	229 × 229	166	63.23%
[40]	KAGGLE	VGG	224 × 224	5500	81.7%
[43]	KAGGLE	Res-Net	128 × 128	35,126 39,424	0.5104 kappa score

## CHAPTER FOUR

### THE PROPOSED METHOD

#### 4.1 Dataset and pre-processing operation

We used a dataset from the KAGGLE competition containing 1000 color images of  $350 \times 350$  resolution. Each image showed specific classes of DR disease, namely Normal, Mild, Moderate, Severe and PDR. These levels were in an unbalanced distribution in the 5 classes, as shown in **Figure 18**. The main operation below was used in order to prepare the images for the model.



**Figure 18** Dataset distribution

- Resizing the images to  $350 \times 350 \times 3$ , except for MobileNet where the input was  $224 \times 224 \times 3$ . We used color images, such as is shown in **Figure 19**.



**Figure 19** Sample from the KAGGLE dataset

- The use 2000 fundus images and splitting them into three parts (80% training set, 10% testing set, 10% validation set)
- Standardizing the training set ( $x_{train}$ ) and ( $y_{train}$ ) by dividing each with the




standard deviation (255 in the RGB image, which denotes to the maximizing value of the pixel channel) so that each image value lies on [0, 1]




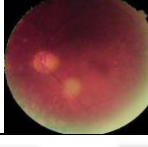
- Specifying five classes which denote the level of DR using one-hot encoded vector for (y\_train) and (y\_test), thus:
  - [1,0,0,0,0] refers to the normal class, [0,1,0,0,0] refers to the moderate class, etc.
  - In order to reduce overfitting and increase the accuracy of the model, we used image-augmentation as an important operation to be applied for both the training and testing datasets using the following characteristics:
    - Shift (width and height)
    - Flip (vertical and horizontal)
    - Zooming
    - Channel\_shift
    - Rotation (0°-360°)
    - Rescale
    - Normalization

#### 4.2 Hyper parameter of the network

- Loss function: We used categorical\_crossentropy.
- Activation function: We used 'ReLU' in hidden layers and in the last layer of the model we used 'softmax' as the activation function.
- Optimizer: We used Adam.
- Learning Rate: Set at 0.0001 and decreasing the value when the loss function increases during the training operation.
- Batch size: Set to 16.

**Table 4** Dataset samples with levels

	Name	Image
Level_0	Normal	

Level_1	Mild- Non-Proliferative Diabetic Retinopathy (NPDR)	
Level_2	Moderate NPDR	
Level_3	Severe NPDR	
Level_4	PDR (Proliferative Diabetic Retinopathy)	

### 4.3 Measure metrics

There are two main measure metrics used to measure the performance of the model:

- Threshold metrics; and
- Rank metrics.

As our dataset had unbalanced class distributions, as shown in **Figure 21**, we will use AUC (Area under Curve ROC) which is a rank metric and a good measure of performance for unbalance datasets.

The ROC curve is denoted to the receiver operating characteristic curve and it has two main aspects:

- True Positive Rate (TPR)
- False Positive Rate (FPR)

$$\text{Sensitivity/TPR} = \frac{TP}{TP + FN} \quad \text{Eq. 63}$$

$$\text{Specificity/FPR} = \frac{FP}{FP + TN} \quad \text{Eq. 63}$$

$$\text{Accuracy} = (TP + TN) / (TP + FP + FN + TN) \quad \text{Eq. 64}$$

where

TPR = True Positive Rate

FPR = False Positive Rate

TP = True Positive value, the label reside to specific class, and classify correctly

TN = True Negative value, the label reside to specific class, and classify wrongly

FP = False Positive value, the label does not reside to specific class, and classify correctly

FN = False Negative value, the label does not reside to specific class, and classify wrongly

#### 4.4 Building the model

##### 4.4.1 Convolutional Neural Network

We built a CNN which is considered state-of-the-art for various image classification problems and we achieved good results. Our CNN architecture was built from scratch for DR image classification using the following layers:

- Convolutional layer: used for the extract feature from an input image which consists of pixels, by using 16 kernels of  $3 \times 3$  size
- Batch-normalization layer: to normalize the output of the previous layer over the batch size.
- Maxpooling layer: to reduce the output image dimension with a  $2 \times 2$  window
- Dropout layer: to reduce overfitting by randomly dropping some neurons during the training
- Flatten layer: to prepare the input to be one dimensional (linear array) in the next layer
- Dense layer: a fully-connected layer which we use for final classification with 5 classes followed by the non-linear activation function ‘softmax’

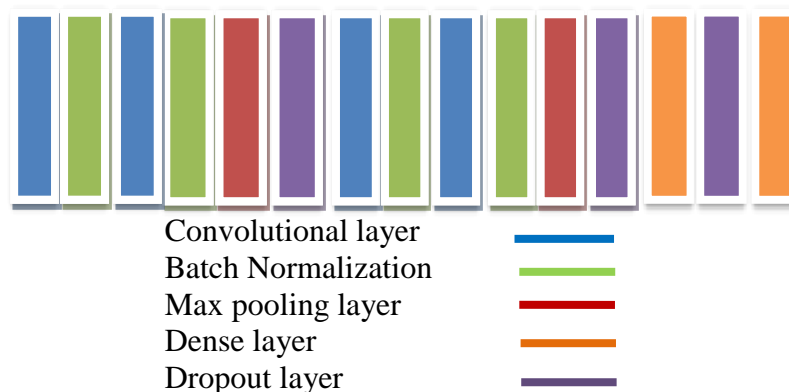


Figure 20 CNN model

#### 4.4.2 MobileNet

With 28 layers and by using the same image pre-processing, we apply Mobile\_Net as a pre-trained model using the same hyper-parameter and converting the last layer with a dense layer with 5 classes.

This model is used for mobile vision applications and embedded systems because of these properties:

- The size of the model being small compared with another models; and
- The computational operations being faster due to depthwise separable convolution.

MobileNet does not have fully-connected layers and it uses a  $1 \times 1 \times 3$  kernel. The main aspect of MobileNet uses depthwise separable convolutions on two levels:

Depthwise Convolution considered at the filter level; and

Pointwise Convolution, which is considered to be a combination level.

The first level is used in order to reduce the computational operation by using a signal channel of the input color image instead of three channels.

The number of computational operation depends on:

the size of the output image;

the number of output channels; and

the size of the kernel.

Number of Computational operations [50] =  $D_k \times D_k \times M \times N \times D_f \times D_f$ .

where

$D_k$  = width/height of the kernel.

$D_f$  = width/height of feature maps

$M$  = number of input channels

$N$  = number of output channels

If we have a  $16 \times 16 \times 3$  input color image and use a kernel size of  $7 \times 7 \times 3$  and want to use 192 channels as an output without padding and stride equal to one, the result of the number of computational operations in a **typical convolution layer** will be as follows:

$$\begin{aligned} \text{Size of the output image} &= (16 - 7 + 1) \times (16 - 7 + 1) \\ &= 10 \times 10 \end{aligned}$$

$$\text{Number of computational operations} = 10 \times 10 \times 7 \times 7 \times 3 \times 192$$

$$= 2,822,400$$

We compute the number of computational operations using the depthwise separable convolution, thus:  $D_k \times D_k \times M \times D_f \times D_f + M \times N \times D_f \times D_f$

Making summations of the computational operation results for both the number of computational operations in the depthwise convolution level and pointwise convolution levels, thus:

$$\text{Number of computational operations} = D_k \times D_k \times M \times D_f \times D_f + M \times N \times D_f \times D_f$$

The number of computational operations in the depthwise convolution level

$$= 3(7 \times 7 \times 1) \times 10 \times 10 \times 1$$

$$= 14,700$$

The number of computational operations in the pointwise convolution level

$$= 192 \times 3 \times 1 \times 1 \times 10 \times 10$$

$$= 57,600$$

The total number of computational operations

$$= 14,700 + 57,600$$

$$= 72,300, \text{ which is less than } 2,822,400$$

In typical convolution layer, an image size of  $224 \times 224 \times 3$  was the input to the MobileNet model and the number of outputs were changed to 5, which represents the levels of Diabetic Retinopathy Disease.

#### 4.4.3 VGG\_16 Net

The first introduction of the stack used in different deep learning architectures was in the VGG Net (Visual Geometry Group). The only pre-processing image subtracted the mean of the RGB. VGG Net contains four blocks and each block consists of a stack of the following layers:

A convolution layer which has a small receptive field of filters,  $3 \times 3$  and  $1 \times 1$ , and a stride equal to one.

A Max pooling layer which has a  $2 \times 2$  window and a stride equal to two.

With 3 fully-connected layers, two contained 4029 neurons and the third contained 1000 neurons such that the number of Convolution Layers and Maxpooling layers were respectively 13 and 5.

The main characteristic of VGG is:

- its use of small respective files which decrease the computation task and increase the operation of the training;
- the use of blocks;
- Ignoring the normalization layer which leads to an increase in the computational task and cost memory size; and
- the use ReLU as the activation function in the hidden layers and Softmax' in the output layer with 1000 neurons.

The input image was  $224 \times 224 \times 3$ .

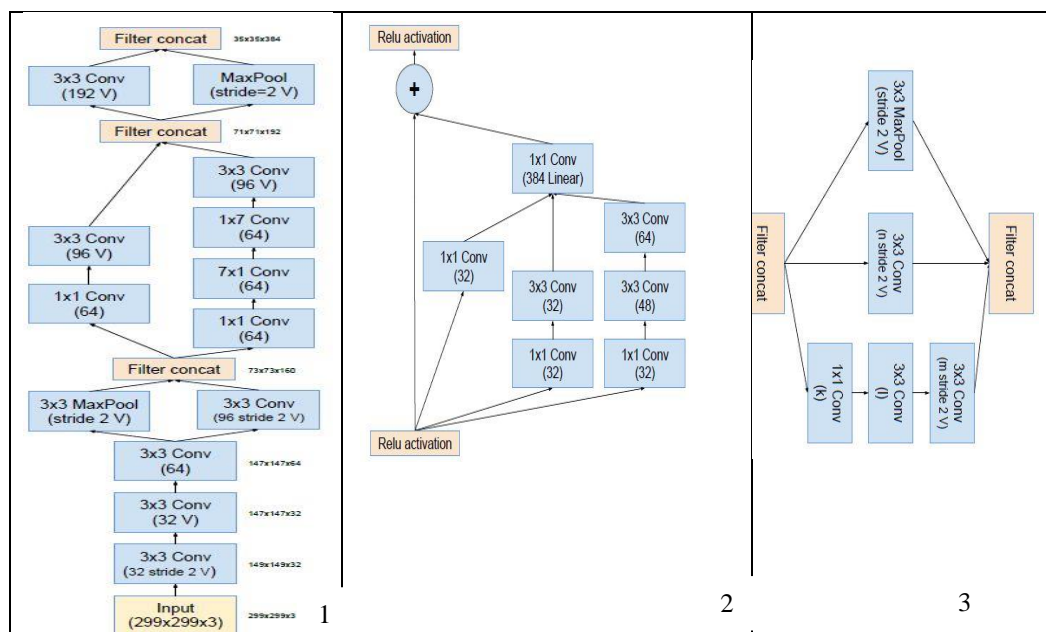
Finally, the VGG Net indicates that the architecture with deeper layers shows good performance in terms of accuracy.

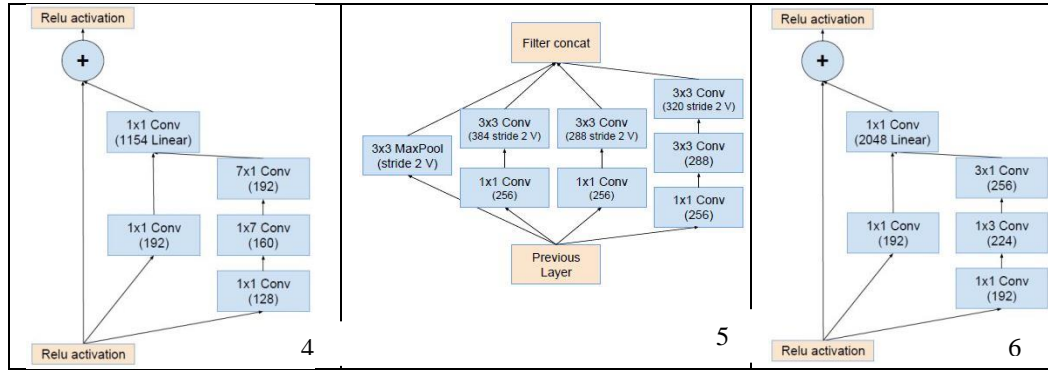
#### 4.4.4 InceptionResNetV2

The combination between two powerful deep neural networks (Inception and Residual networks) produces new a network named InceptionresNetV2.

The main strong side of the **inception network** is the reduction of computational costs, model sizes and dimensionality.

The **residual network** is able to use deeper layers without any impact on the accuracy of the network and speed up the training process of the model by using residual blocks that contain residual connection depths equal to 572.





**Figure 21** InceptionResNetV2 blocks

**Table 5** Order of Block of InceptionResNetV2

Ordered of Blocks and layers	Outputs
Input ( $299 \times 299 \times 3$ )	$299 \times 299 \times 3$
Block_1	$35 \times 35 \times 256$
Block_2	$35 \times 35 \times 256$
Block_3	$17 \times 17 \times 896$
Block_4	$17 \times 17 \times 896$
Block_5	$8 \times 8 \times 1792$
Block_6	$8 \times 8 \times 1792$
Average Pooling	1792
Dropout (0.8)	1792
Softmax	1000

The optimizer that was used was RMSProp (Root Mean Square), which is considered to be a good and fast optimizer. We specify the learning rate as being equal to 0.045 and a decay every two epochs of 0.94.

#### 4.4.5 Inception-v3

The inception network has been developed in many ways:

Inception v1 combines the filter, convolution layer and max-pooling layer using an inception block and it uses an average pooling instead of a fully connected layer.

Inception v2 introduces batch normalization.

Inception v3 [48] won the competition in 2015 of the ImageNet dataset and achieved part 1. It introduces two aspects, namely the factorization method in the third stage of the inception network architecture. This reduces the parameters without any impact on the performance of the network.

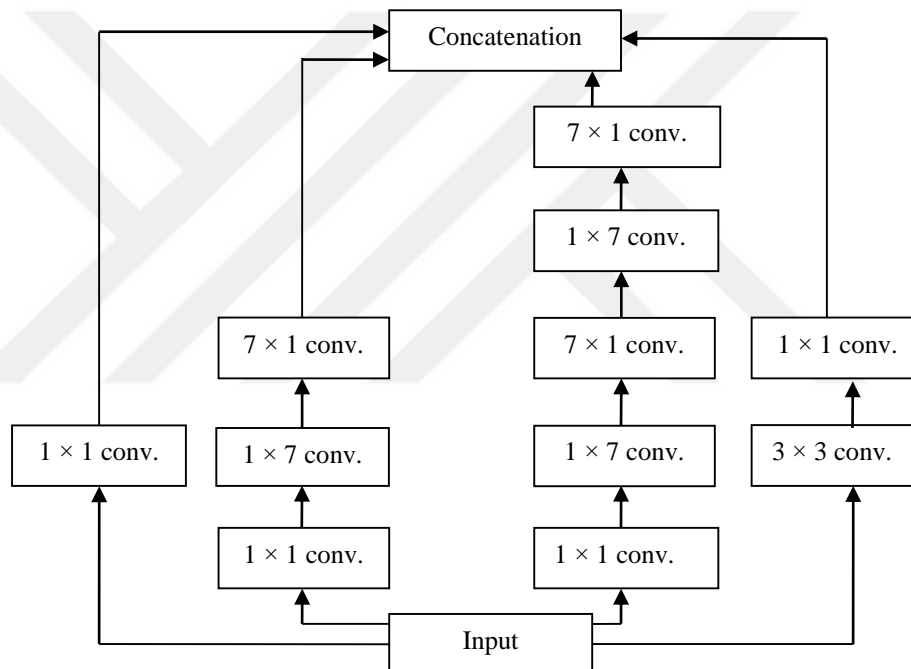
### Regularization method

This applies the idea of shrinking the resolution and increasing the number of channels using a variety of shapes of the convolutional layer, as shown in **Figure 22**, which allows the network to be deeper without any impact on the performance of the network such that deeper and narrower are considered the best architecture for the network. To do so, several changes have been made in the nine-stage inception architecture (inception blocks), as follows:

Using  $3 \times 3$  convolution instead of  $5 \times 5$

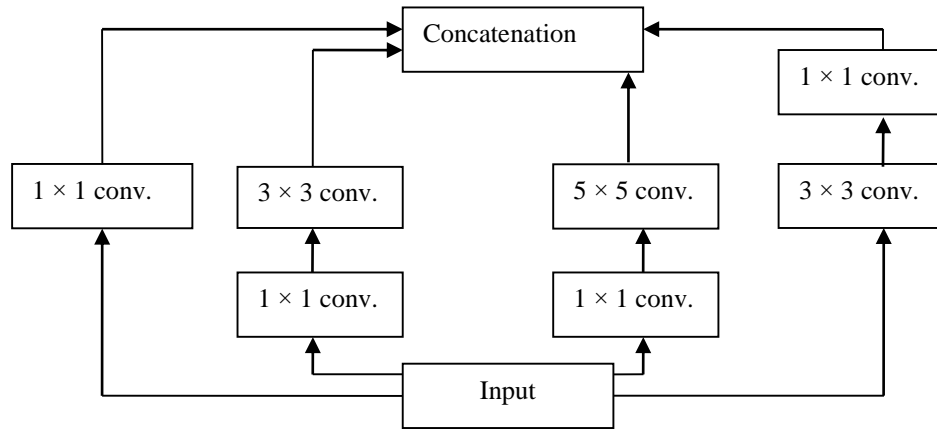
Using  $1 \times 7$  and  $7 \times 1$  convolutions instead of  $5 \times 5$

Using  $1 \times 3$  and  $3 \times 1$  convolutions instead of  $3 \times 3$



**Figure 22** Stage Four of Inception-V3





**Figure 23** Stage Four of Original Inception

#### 4.4.6 Comparison

Below is a comparison between the original network of MobileNet, VGG-16, InceptionV3 and InceptionResNetv2, showing that the third network had the best accuracy.

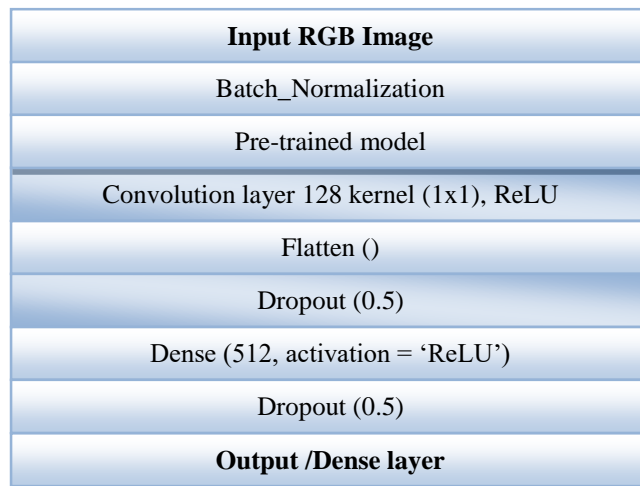
**Table 6** Achievement of the model on the ImageNet validation [49]

Model	Size of Input	Weight_size	Depth	Parameters	Top-5 Accuracy
MobileNet	$224 \times 224 \times 3$	16 MB	88	4,253 M	0.895
VGG-16	$224 \times 224 \times 3$	528 MB	23	138 M	0.901
Inception-V3	$299 \times 299 \times 3$	92 MB	159	23,851,784 M	0.937
Inception_Res_Net_v2	$299 \times 299 \times 3$	215 MB	572	55,873 M	0.953

#### 4.4.7 Fine-Tuned pre-trained model

In order to avoid time consumption due to computational tasks and to make use of the weight parameter of the model that was trained in ImageNet of approximately around 1.2 M, we use a small dataset of different domain compared to ImageNet. In our study, our model act as an automatic extractor, and add new layer using a fine tune operation. This enable weights of the base neural network model to be feed to the main model as pretrained model. Moreover, we add different layers at the bottom in place of the freeze layers in order to handle overfitting problem occurrence.

Basically, earlier layers extract uncomplicated features, while deeper layers extract complicated features and learn the high-level features of an entire image.



**Figure 24** Fine-tune layers

The input image was resized to be  $350 \times 350 \times 3$  for VGG-16, InceptionV3 and InceptionResNetV2, while MobileNet used  $224 \times 224 \times 3$  as the size of the input image.

The distribution of our classes is imbalanced, so we used a batch normalization layer to convert the distribution of the input images over the interval  $[-1, 1]$  and we made the mean equal to 0 while setting the standard deviation to 1.

#### 4.4.8 Building the ML web application using Flask

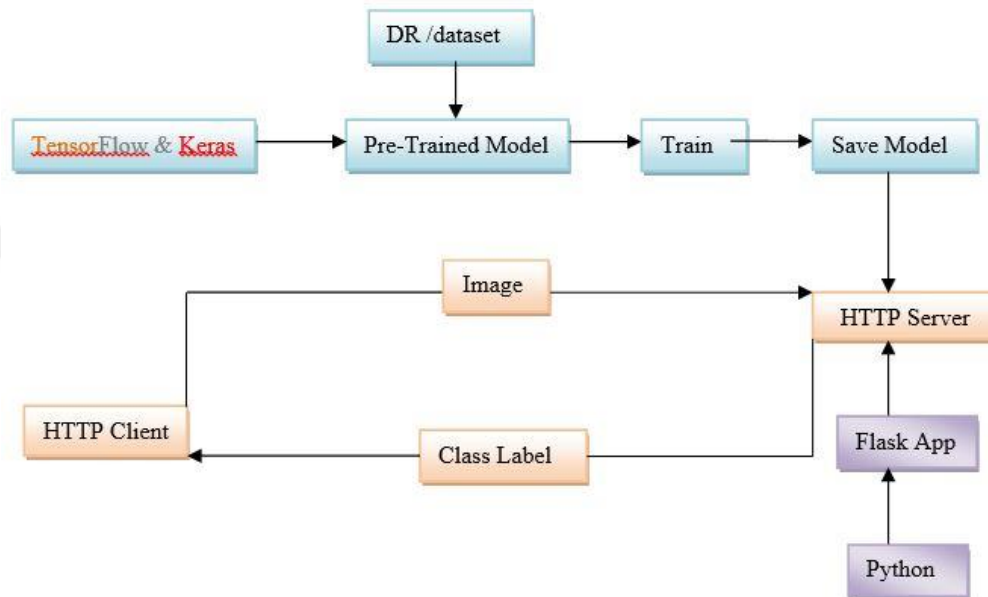
Flask is a micro-framework of Python. It is light and the shortest way to build small web applications using a great amount of code behind it.

After we select the model that has more accuracy, we build our web application using Python and Flask. This can access our model using the http client by uploading DR images to the http client and then sending the request to the http server which responds to the class label of the image by indicating the probability of the distribution. Therefore, we can specify class to which it belongs, as shown in **Figure 25**.

#### 4.4.9 Main steps to building an HTTP accessible client of pre-trained DL models

- Collect the dataset and prepare it to be fed into our model

- Build and train the model
- Save the model so we can use it for our prediction
- Build our Flask web app which gives us the ability to access the pre-trained DL model to classify our DR dataset by using the http protocol.
- Build an html home page which give us the ability to upload our images
- Use a css file to style our web app
- Use our pre-trained DL for prediction



**Figure 25** Building an ML web application using Flask

## CHAPTER FIVE

### EXPERIMENTAL RESULTS

#### 5.1 Platform

In our project, we used the KAGGLE kernel as a platform for our implementation which provides 4 CPUs, 17 GB RAM, a 1 GB disk and 60 minutes for editing the code. We can add the GPU which has 2 CPUs and 14 GB RAM), while the execution time was six hours, using a jupyter type notebook with Python. Similarly, to ML, the cloud system allowed us to build our model easily.

#### 5.2 Prerequisites

Below are the main libraries we used in our implementation.

- Keras, which is an open source neural network in the Python language that has a library that helps the implementation of DL projects and which can import optimizer types and different layers (convolutional layers, max-pool layers, fully-connected layers, etc.) as well as a loss function type.
- A tensorflow library
- A numpy library (Numerical Python) for the computational task
- A matplotlib library for plotting
- A pandas library, which takes data from the input file

#### 5.3 Loading data and pre-processing images

We load the trainLabels.csv file which contains image names and corresponding labels:

```
import pandas as pd
```

```

random.seed(10)
trainLabels = pd.read_csv("../input/diabetic-retinopathy-detection/trainLabels.csv")
trainLabels.head()

```

### 5.3.1 Resizing

The input images are resized to  $350 \times 350 \times 3$  for (VGG16, InceptionV3 and InceptionResNetV2), while for MobileNet we used  $224 \times 224 \times 3$  as the input image.

```

from PIL import Image
from keras.preprocessing import image
import os
import numpy as np

img_rows, img_cols = 350,350
listing = os.listdir("../input/diabetic-retinopathy-detection")
listing.remove("trainLabels.csv")
immatrix = []
imlabel = []
for file in listing:
    base = os.path.basename("../input/diabetic-retinopathy-detection/" + file)
    fileName = os.path.splitext(base)[0]
    imlabel.append(trainLabels.loc[trainLabels.image==fileName, 'level'].values[0])
    im = Image.open("../input/diabetic-retinopathy-detection/" + file)
    img = np.array(im.resize((img_rows,img_cols)))
    immatrix.append(np.array(img))

```

### 5.3.2 Shuffling the Data

To create our train\_data and visualize our label:

```

from sklearn.utils import shuffle
data,label = shuffle(immatrix, imlabel, random_state=42)

```

```
train_data = [data,label]
```

### 5.3.3 Splitting the Data

We divided our data into three parts (80% as the training dataset, and 10% as the testing dataset and 10% as the validation dataset.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(train_data[0], train_data[1], test_size
= 0.2, random_state = 42)
```

### 5.3.4 Standardizing the Dataset

We standardize the dataset (x\_train, x\_test) after converting it into a numpy array and dividing each row by 255, which is the maximum number of pixels.

```
from keras.utils import np_utils
x_train = np.array(x_train).astype("float32")/255.
x_test = np.array(x_test).astype("float32")/255.
```

5 - One-hot encoded

With five classes, we use the one-hot encoded vector as follows:

Class0 as [1,0,0,0,0] and class1 as [0,1,0,0,0] , and so on.

```
from keras.utils import np_utils
y_train = np_utils.to_categorical(np.array(y_train), 5)
y_test = np_utils.to_categorical(np.array(y_test), 5)
```

### 5.3.5 Data-Augmentation

To increase the number of training datasets from 800 to 1620 color datasets and to avoid overfitting, we used different options of data-augmentation on the training and testing images.

```

def brightness_adjustment(img):
    hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    ratio = .5 + np.random.uniform()
    hsv[:, :, 2] = np.clip(hsv[:, :, 2].astype(np.int32) * ratio, 0, 255).astype(np.uint8)
    return cv2.cvtColor(hsv, cv2.COLOR_HSV)

```

```

from keras.preprocessing.image import ImageDataGenerator
import numpy
X_train = numpy.array(x_train, copy=True)
Y_train = numpy.array(y_train, copy=True)
shift = 0.2
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=360,
    preprocessing_function=brightness_adjustment,
    width_shift_range=shift,
    height_shift_range=shift, shear_range=0.2,
    zoom_range=0.2, channel_shift_range=4.,
    horizontal_flip=True, vertical_flip=True,
    rescale=1. /255,
    fill_mode='nearest')
datagen.fit(x_train)
# concatenating the old data with the augmented data
train_x = numpy.concatenate((x_train, X_train), axis=0)
train_y = numpy.concatenate((y_train, Y_train), axis=0)

```

After augmentation our dataset were as follow:

N.Training dataset = 1620

N.Validation dataset = 180

N.testing dataset = 200

## 5.4 Early Stopping Technique

In order to reduce the overfitting and reach the minimum loss function, we used the early stop, thus:

```
from keras.callbacks import ModelCheckpoint, LearningRateScheduler,
EarlyStopping, ReduceLROnPlateau
weight_path="{ }_weights.best.hdf5".format('boat_detector')

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,
                             save_best_only=True, mode='min', save_weights_only = True)

reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.8, patience=10,
verbose=1, mode='auto', epsilon=0.0001, cooldown=5, min_lr=0.0001)
early = EarlyStopping(monitor="val_loss",
                      mode="min",
                      patience=15) # probably needs to be more patient, but kaggle time is
limited
callbacks_list = [checkpoint, early, reduceLROnPlat]
```

## 5.5 Compiling the Model

```
Model_Name = model.compile(optimizer=Adam(1e-3),
loss='categorical_crossentropy', metrics=['acc',f1])
```

## 5.6 Fitting the Model

```
Model_Name =model.fit(train_x, train_y, batch_size = 16,
epochs =150,validation_split=0.1, verbose=1,callbacks=callbacks_list, shuffle=True)
```



## 5.7 Building the Model

### 5.7.1 Convolutional Neural Network

```
model = Sequential()
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
                 input_shape=x_train[0].shape))
model.add(BatchNormalization())

model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu'))
model.add(BatchNormalization())

activation='relu'))
model.add(MaxPooling2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(BatchNormalization())

model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(BatchNormalization())

activation='relu'))
model.add(MaxPooling2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(5, activation='softmax'))
```

### 5.7.2 Pre-training the DL Model

```
def create_model(input_shape, n_out):
    pretrain_model = X (include_top=False, weights='imagenet',
input_shape=input_shape)

    input_tensor = Input(shape=input_shape)
    bn = BatchNormalization()(input_tensor)
    x = pretrain_model(bn)
    x = Conv2D(128, kernel_size=(1,1), activation='relu')(x)
    x = Flatten()(x)
    x = Dropout(0.5)(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(n_out, activation='softmax')(x)
    model = Model(input_tensor, output)
    return model
```

By changing the character **X** to the name of the pre-trained model (MobileNet, VGG16, InceptionV3, InceptionResNetV2), we can import the pre-trained model with its weight and use it.

## 5.8 Results

Our Convolution Neural Network which was built from scratch using different layers shows lower performance and a lower AUC value, which was 0.50 at 16 epochs and a highest loss value of 5.318, which is considered the largest value compared to other pre-trained deep learning models. Moreover, as shown in **Table 8**, the CNN model shows overfitting between the training and validation datasets.

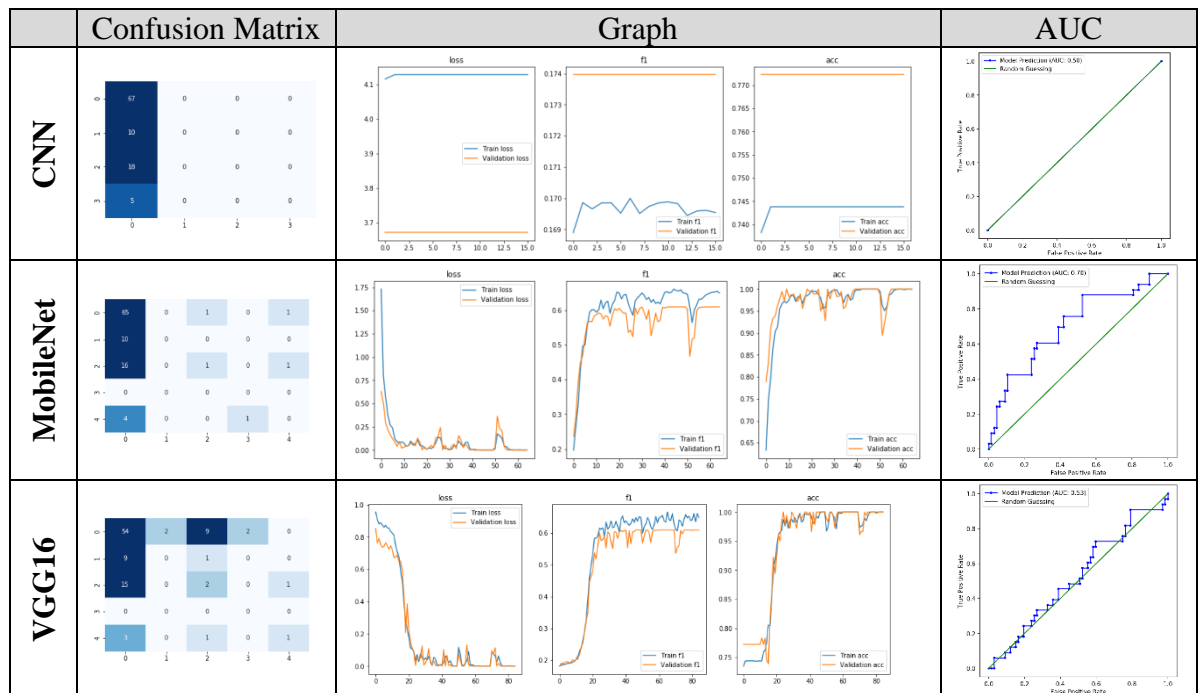
On the other hand, when using four types of the pre-trained model, each was fine-tuned by added several layers and replacing the final layer with a fully connected layer containing five neurons corresponding to the number of DR levels showing a better result.

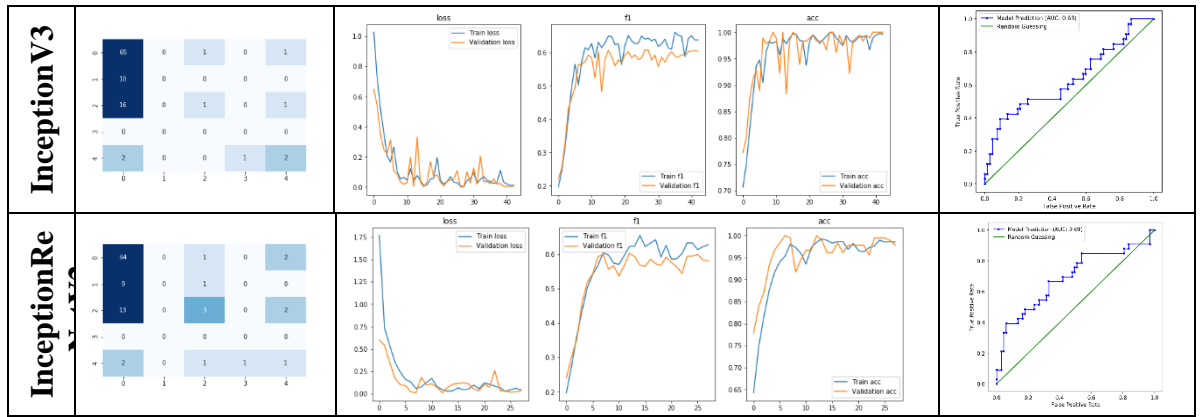
As seen in **Table 7**, VGG-16 showed a lower value of the AUC as MobileNet, while InceptionResNetV2 showed the largest value of the AUC at 0.69 and a lower loss equal to 2.191 in spite of the small numbers of epochs, which numbered 28 compared to the number of epochs in InceptionV3, numbering 43 with a loss equal to 3.262.

**Table 7** Result with color images

	ACC Test_data	AUC	Loss	Early Stop	Total parameters
CNN	67%	0.50	5.318	16 Epoch	116,153,493
Mobile-Net with 224 × 224	66%	.070	3.113	65 Epoch	31,782,929
VGG_16	57%	.053	4.727	85 Epoch	21,337,041
InceptionV3	68%	0.63	3.262	43 Epoch	27,376,561
InceptionResNetV2	68%	0.69	2.191	28 Epoch	59,844,977

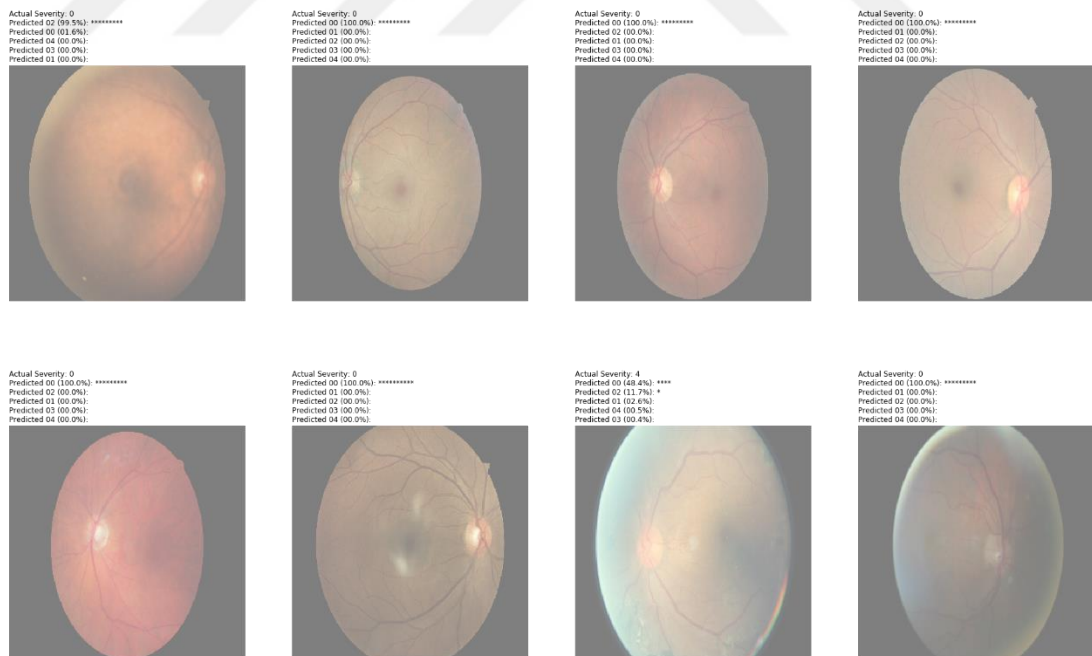
**Table 8** Graphics results of models





**Table 9** Run-Time for each models with No. of Epoch

	CNN	<u>MobileNet</u>	VGG16	InceptionResNetV2	InceptionV3
<u>Time/Second</u>	208	1969	2811	829	1161
<u>Epoches</u>	16	65	85	28	43



**Figure 26** Sample of Predicted Images using Inception-v3

## CHAPTER SIX

### DISCUSSION AND FUTURE WORK

In this study, we propose a model of different depths in terms of architecture. The proposed model achieves higher performance compared to the traditional CNN built from scratch due to the availability of limited depth in its architecture. This is also due to the size of the train data use for the experiment resulting the model to be inefficient and time-consuming.

We use an ImageNet dataset to train our fine-tuned pretrained model. In the experiment, we compare it terms of different domain and tune the parameters as well as freeze some layers in order to observe the behavior of the propose methods. Moreover, the lower level of the network layer extract low features likewise the high-level features extract the high and complex features. Therefore, we freeze low-level layer and some part of the high-level and add a different layer and softmax layer to output the probability of each given class.

Furthermore, the Fine-tuned pre-trained deep learning weights used for medical image classification using the KAGGLE dataset for diabetic retinopathy disease showed good results due to the depth and width of the network as well as the huge numbers of the training dataset (ImageNet) which was used for training.

The main aspect in MobileNet was the use of the Depthwise separable Convolution, which leads to shrinking the weight size and decreasing the model size with 88 layers as the depth size. This produced better results using a  $224 \times 224 \times 3$  image size, which was better than VGG16, which used a  $350 \times 350 \times 3$  image size producing a lower AUC value of 0.53 despite the large numbers of epochs (85 epochs). The model is of 23 layers, with a differ 29 epochs which is considered quite small compared to 65 epochs used in MobileNet as shown in **Table 7**.

One the other hand, both InceptionV3 and InceptionResNetV2, which is state-of-the-art in computer vision and image classification tasks, showed a convergence of their results in spite of variations between the epochs. They have the ability to shrink the

resolution and go deeper and increase the numbers of layers with 159 as a depth of InceptionV3 and 572 as a depth of InceptionResNetV2, of which achieve high performance using the KAGGLE diabetic retinopathy disease image dataset.

By training the KAGGLE images using a pre-trained InceptionV3 model, we made more improvements to the AUC value at 0.63 instead of 0.59 in [50], which used the same model (InceptionV3) after 9 epochs at a  $512 \times 512 \times 3$  image resolution and with the early stop technique. The accuracy of the testing dataset was equal to 68%, which showed higher performance than [36] and [38].

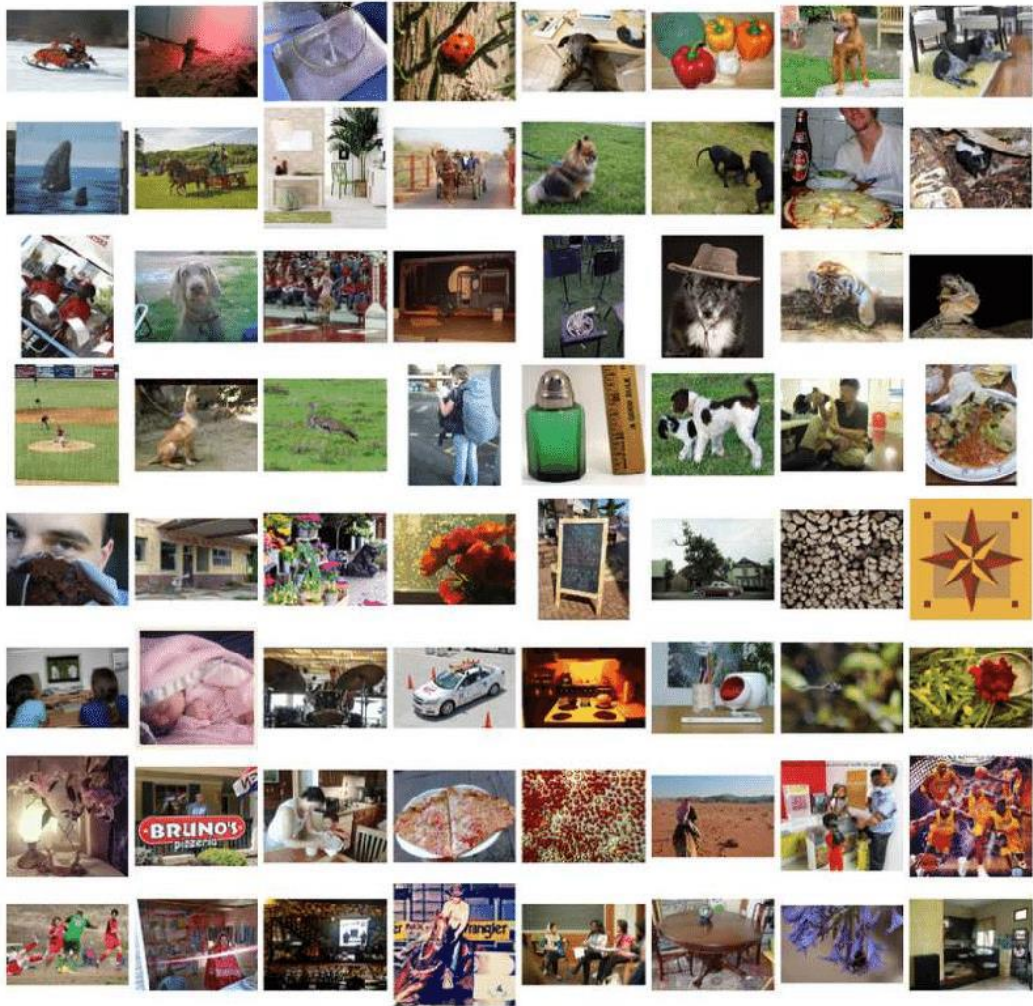
Our method used the pre-trained model (InceptionV3) with pre-processing and fine-tuning as follows:

- An image augmentation technique which increases the number of images to 2000;
- layer-wise fine-tuning pre-trained models by freezing the lower layer level and part of the higher layer level and adding several layers, as shown in **Figure 24**, using a  $350 \times 350 \times 3$  image resolution;
- using Adam as an optimizer; and
- After 43 epochs, using the early stopping method, the AUC was 0.63%.

The improvement in our research was due to increasing the number of images to 2000 and dividing them into 1620 training images, 200 testing images and 180 validation images as well as ordering of layers which were added at the end of the model. It was quite helpful to use a powerful layer convolutional for feature extraction, normalize the output of the pre-trained model and use it as an input to the next layer (convolutional with  $1 \times 1$  kernel size, 128 neurons and two blocks (dropout 0.5, dense) to reduce the overfitting of our model.

Using image-augmentation as well as increasing the numbers of training images could solve the overfitting problems.

Diabetic retinopathy images are considered medical images which contain more details with noise and variation, and their different domains form the ImageNet dataset, as shown in **Figure 27**, which contained 10 million labeled images and, 1000 categories with a  $469 \times 387$  resolution.



**Figure 27** Examples from the ImageNet dataset [52]

Finally, InceptionResNetV2 shows the best performance for medical images using the KAGGLE dataset of diabetic retinopathy disease.



**Figure 28** Examples from the Diabetic Retinopathy dataset



Our future works will be the following:

- Use of Cloud notebook Google Collaboratory, which considers a jupyter notebook environment that enables the writing and execution of our code in our browser using thousands of images and using the GPU in addition to setting the resolution of images to  $512 \times 512$  or higher, which increases the performance of the model and makes specific details in the DR clear.
- The use of the Flask python framework to build a diabetic retinopathy application for use on mobile devices.
- Making connections between a model and a mobile camera by taking pictures. The function of the model would be to perform a few image processes followed by detecting it at a diabetic retinopathy level.

With these steps, it would be possible for anyone suffering from diabetic diseases to check their eyes and determine whether or not they are infected in addition to checking the factors in the progress of the diabetic retinopathy disease using sugar measurement or blood pressure devices.



## REFERENCES

- [1] S. Kiranyaz, T. Ince, A. Iosifidis, and M. Gabbouj, “Progressive Operational Perceptrons,” *Neurocomputing*, vol. 224, no. November 2016, pp. 142–154, 2017.
- [2] S. Kong, “Hexpo : A Vanishing-Proof Activation Function,” pp. 2562–2567, 2017.
- [3] S. Ruder, “An overview of gradient descent optimization algorithms,” pp. 1–14, 2016.
- [4] Y. LeCun and Y. Bengio, “Convolutional Networks for Images, Speech, and Time-Series,” *Handb. Brain Theory Neural Networks*, no. May 2013, 1995.
- [5] Hinton, “A Practical Guide to Training Restricted Boltzmann Machines A Practical Guide to Training Restricted Boltzmann Machines,” *Computer (Long Beach, Calif.)*, vol. 9, no. 3, p. 1, 2010.
- [6] N. F. Hordri, S. S. Yuhaniz, and S. M. Shamsuddin, “Deep Learning and Its Applications: A Review,” no. May 2017, 2016.
- [7] W. Liu, Z. Wang, X. Liu, Y. Liu, and F. E. Alsaadi, “A survey of deep neural network architectures and their applications,” *Neurocomputing*, vol. 234, pp. 11–26, 2016.
- [8] Indraneel Pole Matrikel, “Restricted Boltzmann Machine - A comprehensive study with a focus on ...,” 2015. [Online]. Available: <https://www.slideshare.net/poleindraneel/413688151-restricted-boltzmann-machine-a-comprehensive-study-with-a-focus-on-deep-belief-networks>. [Accessed: 11-Dec-2018].
- [9] Y. Hua, J. Guo, and H. Zhao, “Deep Belief Networks and deep learning,” *Proc. 2015 Int. Conf. Intell. Comput. Internet Things, ICIT 2015*, pp. 1–4, 2015.
- [10] “LeCun\_et\_al\_Gradient-Based\_Learning\_Applied\_to\_Document\_Recognition\_IEEE1998.”
- [11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks.”

- [12] K. Simonyan and A. Zisserman, "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION," 2015.
- [13] C. Szegedy *et al.*, "Going Deeper with Convolutions."
- [14] M. Lin, Q. Chen, and S. Yan, "Network In Network," pp. 1–10, 2013.
- [15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016.
- [16] P. S. Bidari and V. L. Almazroo, "Automatic Detection and Classification of Diabetic Retinopathy Using Retinal Fundus Images," vol. 10, no. 7, pp. 1308–1311, 2016.
- [17] A. Roy, D. Dutta, P. Bhattacharya, and S. Choudhury, "Filter and Fuzzy C means based feature extraction and classification of diabetic retinopathy using support vector machines," *Proc. 2017 IEEE Int. Conf. Commun. Signal Process. ICCSP 2017*, vol. 2018-Janua, no. c, pp. 1844–1848, 2018.
- [18] W. Cao, J. Shan, N. Czarnek, and L. Li, "Microaneurysm detection in fundus images using small image patches and machine learning methods," *Proc. - 2017 IEEE Int. Conf. Bioinforma. Biomed. BIBM 2017*, vol. 2017-Janua, pp. 325–331, 2017.
- [19] A. Abraham, "Analysis of Genetic Algorithms and Distinctiveness in Back Propagation Neural Network on Diabetic Retinopathy classification," 2018.
- [20] M. You and Y. Li, "Automatic classification of the diabetes retina image based on improved BP neural network," *Proc. 33rd Chinese Control Conf. CCC 2014*, pp. 5021–5025, 2014.
- [21] M. A. Al-Jarrah and H. Shatnawi, "Non-proliferative diabetic retinopathy symptoms detection and classification using neural network," *J. Med. Eng. Technol.*, vol. 41, no. 6, pp. 498–505, 2017.
- [22] V. Mareeswari, R. Saranya, R. Mahalakshmi, and E. Preethi, "Prediction of diabetes using data mining techniques," *Res. J. Pharm. Technol.*, vol. 10, no. 4, pp. 1098–1104, 2017.
- [23] A. J. Rosales-silva and F. J. Gallegos-funes, "Detection and classification of Non-Proliferative Diabetic Retinopathy using a Back-Propagation neural network Detección y clasificación de Retinopatía Diabética no Proliferativa usando una Red Neuronal de Retropropagación," no. July, pp.

70–85, 2015.

- [24] M. Computing, “Comparison of ANNs , Fuzzy Logic and Neuro- Fuzzy Integrated Approach for Diagnosis of Coronary Heart Disease : A Survey,” vol. 2, no. June, pp. 216–224, 2013.
- [25] P. Prentasic and S. Loncaric, “Detection of exudates in fundus photographs using convolutional neural networks,” *9th Int. Symp. Image Signal Process. Anal. ISPA 2015*, no. Ispa, pp. 188–192, 2015.
- [26] S. Yu, D. Xiao, and Y. Kanagasingham, “Exudate detection for diabetic retinopathy with convolutional neural networks,” *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, pp. 1744–1747, 2017.
- [27] A. Benzamin and C. Chakraborty, “Detection of Hard Exudates in Retinal Fundus Images using Deep Learning,” *2018 IEEE Int. Conf. Syst. Comput. Autom. Netw.*, pp. 1–5, 2018.
- [28] M. Jena, S. P. Mishra, and D. Mishra, “Detection of Diabetic Retinopathy Images Using a Fully Convolutional Neural Network,” *2018 2nd Int. Conf. Data Sci. Bus. Anal.*, pp. 523–527, 2018.
- [29] R. Ghosh, K. Ghosh, and S. Maitra, “Automatic detection and classification of diabetic retinopathy stages using CNN,” *2017 4th Int. Conf. Signal Process. Integr. Networks, SPIN 2017*, pp. 550–554, 2017.
- [30] S. Suriyal, C. Druzgalski, and K. Gautam, “Mobile assisted diabetic retinopathy detection using deep neural network,” *2018 Glob. Med. Eng. Phys. Exch. Am. Heal. Care Exch. GMEPE/PAHCE 2018*, no. 562, pp. 1–4, 2018.
- [31] D. Doshi, A. Shenoy, D. Sidhpura, and P. Gharpure, “Diabetic retinopathy detection using deep convolutional neural networks,” *Int. Conf. Comput. Anal. Secur. Trends, CAST 2016*, pp. 261–266, 2017.
- [32] R. Safarjalani, S. Ainouz, A. Shahin, M. Fabrice, D. Fonderie, and L. Creusot, “Diabetic Retinal Tomographical Image Classification using Convolutional Neural Network,” *2018 Int. Conf. Intell. Adv. Syst.*, pp. 1–6, 2018.
- [33] P. Khojasteh, B. Aliahmad, S. P. Arjunan, and D. K. Kumar, “Introducing a Novel Layer in Convolutional Neural Network for Automatic Identification of Diabetic Retinopathy,” *Conf. Proc. ... Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. IEEE Eng. Med. Biol. Soc. Annu. Conf.*, vol. 2018, no. c, pp. 5938–5941, 2018.

- [34] O. Perdomo, S. Otalora, F. A. Gonzalez, F. Meriaudeau, and H. Muller, "OCT-NET: A convolutional network for automatic classification of normal and diabetic macular edema using sd-oct volumes," *Proc. - Int. Symp. Biomed. Imaging*, vol. 2018-April, no. Isbi, pp. 1423–1426, 2018.
- [35] W. M. Gondal and M. K. Jan, "WEAKLY-SUPERVISED LOCALIZATION OF DIABETIC RETINOPATHY LESIONS IN RETINAL FUNDUS IMAGES Bosch Center for Artificial Intelligence , Robert Bosch GmbH , Stuttgart , Germany Department of Computer Science , TU Dortmund University , Germany Max Planck Institu," pp. 2069–2073, 2017.
- [36] S. Masood, T. Luthra, H. Sundriyal, and M. Ahmed, "Identification of diabetic retinopathy in eye images using transfer learning," *Proceeding - IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2017*, vol. 2017-Janua, no. ii, pp. 1183–1187, 2017.
- [37] S. Srivastava, S. Prabhu, S. Ramesh, and S. Pratapneni, "Visualizing the Indicators of Diabetic Retinopathy Learnt by Convolutional Neural Networks," *2017 IEEE Int. Conf. Comput. Intell. Comput. Res.*, pp. 1–3, 2017.
- [38] X. Wang, Y. Lu, Y. Wang, and W. B. Chen, "Diabetic retinopathy stage classification using convolutional neural networks," *Proc. - 2018 IEEE 19th Int. Conf. Inf. Reuse Integr. Data Sci. IRI 2018*, pp. 465–471, 2018.
- [39] S. Mohammadian, A. Karsaz, and Y. M. Roshan, "Comparative Study of Fine-Tuning of Pre-Trained Convolutional Neural Networks for Diabetic Retinopathy Screening," *2017 24th Iran. Conf. Biomed. Eng. 2017 2nd Int. Iran. Conf. Biomed. Eng. ICBME 2017*, no. December, pp. 1–6, 2018.
- [40] A. Kwasigroch, B. Jarzembinski, and M. Grochowski, "Deep CNN based decision support system for detection and assessing the stage of diabetic retinopathy," *2018 Int. Interdiscip. PhD Work. IIPhDW 2018*, pp. 111–116, 2018.
- [41] X. Li, T. Pang, B. Xiong, W. Liu, P. Liang, and T. Wang, "Convolutional neural networks based transfer learning for diabetic retinopathy fundus image classification," *Proc. - 2017 10th Int. Congr. Image Signal Process. Biomed. Eng. Informatics, CISP-BMEI 2017*, vol. 2018-Janua, no. 978, pp. 1–11, 2018.
- [42] I. Ardiyanto, H. A. Nugroho, and R. L. B. Buana, "Deep learning-based Diabetic Retinopathy assessment on embedded system," *Proc. Annu. Int. Conf.*

- IEEE Eng. Med. Biol. Soc. EMBS*, pp. 1760–1763, 2017.
- [43] S. Rufaida and M. I. Fanany, “Residual Convolutional Neural Network,” 2017.
- [44] F. Yu, J. Sun, A. Li, J. Cheng, C. Wan, and J. Liu, “Image quality classification for DR screening using deep learning,” *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, pp. 664–667, 2017.
- [45] H. Ghebrechristos, G. Alaghband, and R. Y. Hwang, “RetiNet — Feature Extractor for Learning Patterns of Diabetic Retinopathy and Age-Related Macular Degeneration from Publicly Available Datasets,” *2017 Int. Conf. Comput. Sci. Comput. Intell.*, pp. 1643–1648, 2017.
- [46] J. Shan and L. Li, “A Deep Learning Method for Microaneurysm Detection in Fundus Images,” *Proc. - 2016 IEEE 1st Int. Conf. Connect. Heal. Appl. Syst. Eng. Technol. CHASE 2016*, pp. 357–358, 2016.
- [47] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017.
- [48] P. Goodman, A. Namdeo, F. Galatioto, M. C. Bell, E. Foster, and C. Shield, “Investigation of the emission and air quality impacts of low emission zone scenarios in Newcastle and Gateshead, UK,” *HARMO 2014 - 16th Int. Conf. Harmon. within Atmos. Dispers. Model. Regul. Purp. Proc.*, 2014.
- [49] Keras, “No Title.” [Online]. Available: <https://keras.io/applications/>.
- [50] kevin mader, “InceptionV3,” 2017, 2018. [Online]. Available: <https://www.kaggle.com/kmader/inceptionv3-for-retinopathy-gpu-hr>.
- [51] Y. Ren, “Examples-in-the-ImageNet-dataset.” p. 2016.
- [52] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).
- [53] Deshmukh, A. V., Patil, T. G., Patankar, S. S., & Kulkarni, J. V. (2015, August). Features based classification of hard exudates in retinal images. In *2015 international conference on advances in computing, communications and informatics (ICACCI)* (pp. 1652-1655). IEEE.