Diabetic Retinopathy Detection Using Deep Learning

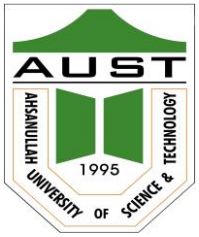
A Project for Soft Computing Lab,
Course no: ***CSE 4238***

Submitted by

Sabbir Ahmed	**170104011**
Lamia Anjum	**170104023**

Supervised by

Dr. -Ing. Nusrat Jahan Lisa



Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

Dhaka, Bangladesh

ABSTRACT

In most countries of the world, diabetes is blamed for various eye diseases. The most severe case of these is Diabetic Retinopathy(DR). On average, one out of every three people with diabetes has some form of Diabetic Retinopathy, and everyone with the disease is at risk. The most common effects of this disease are damage to the retina and the blood vessels of both eyes. The blood vessels may swell and rupture. Sometimes abnormal growth of blood vessels are also seen behind the eye. These blood vessels supply blood to the retina at any moment and if they get damaged the blood to retina may not be able to be supplied. This may have an adverse effect on vision. Excessive blood sugar levels damage these retinas behind the eyes and increases the risk of DR. If this disease is not treated at the right time, blindness can appear soon. The method of manual diagnosis of Diabetic Retinopathy is very anfractuous and time-consuming, but it is undeniable that DR must be diagnosed at an early stage to avoid irreversible vision loss.In this study, we use the **APTOS** dataset where many high resolution retinal images are publicly available. Here, we apply multiple deep learning models on this dataset and follow different approaches on the same dataset. We also create a binary dataset by modifying one dataset and also apply multiple models on it. Our goal is to create a comparison between different deep learning models, when they are applied into a same dataset. Our last approach is to apply clustering algorithm on a new **APTOS** dataset and use classification model on those clusters.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Diabetic retinopathy(DR) is the most weakening form of diabetic eye disease which occurs serious damage to retina and causes vision deterioration. It only affects people who have had diabetes (diagnosed or undiagnosed) for a significant number of years and can affect all diabetics and becomes particularly dangerous, increasing the risk of blindness, and results in irreversible damages if it is left untreated.

Progression of visual impairment may be prevented if detected early. But the early stages of DR may occur without symptoms and without any deterioration. An actual impact on the vision will not occur until the disease proceeds to the worst. As we know it is important to detect DR from early stage, but the early stage and normal stage of DR has no basic difference. So, it turns into a real challenge to admit it from starting stage.

Globally, the number of DR patients is expected to increase from 382 million to 592 million by 2025 [35]. A survey [35] conducted in the province of Khyber Pakhtunkhwa (KPK),Pakistan, report 30% of diabetes patients are affected by DR in which 5.6% succumbs to blindness. Over time, the mild NPDR develops into PDR if not controlled in the early stages. Another survey [36], conducted in Sindh, Pakistan, observed 130 patients with DR symptoms. It is reported that 23.85% of the total observed patients were DR in which 25.8% were diagnosed as PDR patients.

The five stages of Diabetic retinopathy are classified as ***No DR***, ***Mild DR***, ***Moderate DR***, ***Severe DR***, and ***Proliferative DR*** . Each stages has its own symptoms and specific properties. In the early stages of the DR the patients are asymptomatic but in advanced stages, it leads to floaters, blurred vision, distortions, and progressive visual acuity loss. Manual diagnosis is performed on retinal fundus images and requires experienced clinicians to detect and quantify the importance of several small details which makes this an exhaustive and time-consuming task. But no approach is able to detect the Mild stage. The detection of the mild stage is important for the early control of this fatal disease. This study focuses

to detect all the stages of DR (including the mild stage) using end-to-end deep ensemble networks.The below Table 1.1 gives the overall details about the DR stages:

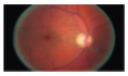
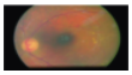
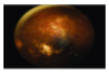
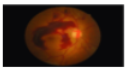
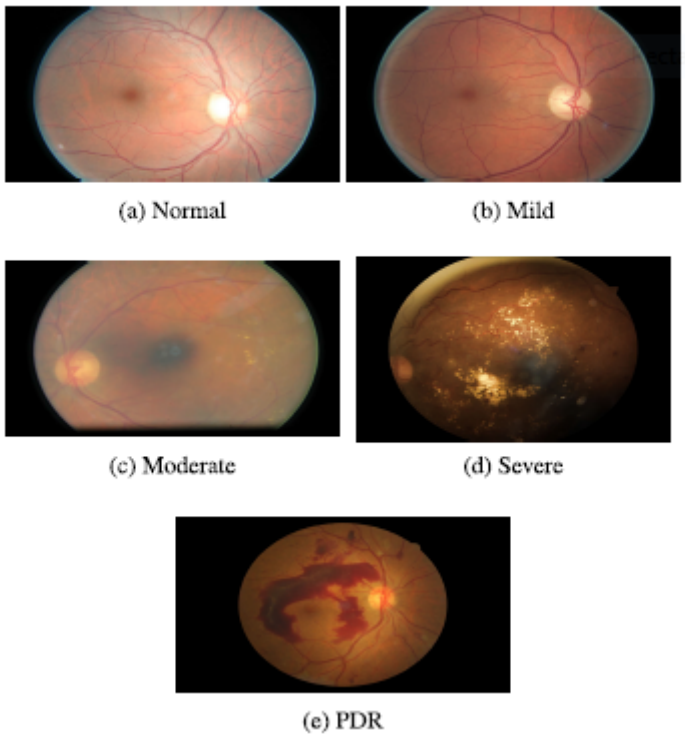
Levels	0	1	2	3	4
Diagnosis	No DR	Mild DR	Moderated DR	Severe DR	PDR
Description	No abnormalities	The most early stage,where only microaneurysms Can happen	Ability of blood transportation And swelling with the progress of the disease	Blood supply to the retina due to increased blockage of more blood vessels	The advanced stage, where the growth features secreted by the retina activate proliferation of new blood vessels
Sample images with there levels					

Table 1.1: Types of Diabetic Retinopathy.

In this work, we attempt to develop a computer-assisted tool to classify medical images of the retina in order to diagnose diabetic retinopathy quickly and accurately which identifies exudates, micro-aneurysms and hemorrhages at an early stage, by training with labeled samples which are publicly available on Kaggle Database.



(a) Normal (b) Mild

(c) Moderate (d) Severe

(e) PDR

Figure 1.1: The different stages of DR.

1.1 Objective

- To get familiar with deep learning models
- To gain knowledge about the symtoms of Diabetic Retinopathy
- To increase accuracy of Diabetic Retinopathy detection
- To create a comparison between different deep learning models

1.2 Motivation

- Increasing the working scope in Bangladesh about Diabetic Retinopathy detection
- By the early detection of Diabetic Retinopathy, we can save many diabetic people from blindness caused by time-consuming manual diagnosis

Chapter 2

Background Study

As we are implementing different architectures of deep learning, we need to know about some basics of these architectures. Topics related to these architectures are explained below.

2.1 Image Processing

Image Processing means extracting information from images. In this step of image processing the input will be set of images or video frame. The inputs will go through some specific procedures or methods which will help us to get the desired info from the images. Image processing can be used for object detection, measurement, identification and verification. Before image goes to the procedures or methods it may need some modification which can be called pre-processing steps. These pre-processing steps will make the image quality much better and that will lead us to achieve our output more precisely. There are some common pre-processing steps for image processing such as :

- Resizing and cropping of input images
- Proper data cleaning and remove unnecessary part from the images
- Data Augmentation

After pre-processing the images are perfectly ready for using in the main image processing architecture. Then the images are converted to multidimensional array. There are many pre-trained deep learning Convolutional Neural Network (CNN) models such as (VGG16, Densenet121, Alexnet, ResNet, Xception) are commonly used for classifying the pre-processed input images. As these models are pre-trained it will give much accurate output if we train the model with our own input datasets. Image processing mainly detects if there are any

similar pattern in the test data image and train data image. Lets see the following Figure 2.1 for an example :

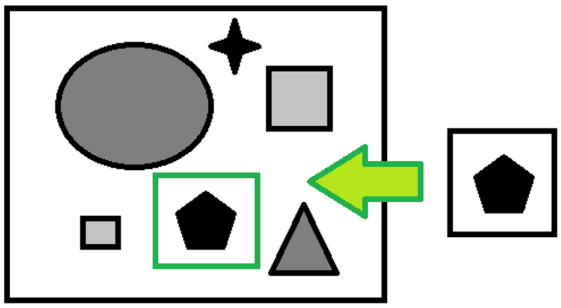


Figure 2.1: Pattern Matching [14]

Here we can see that a pentagon at the right side of the picture. Lets assume that it is an image from dataset. At the left side of the picture there are pictures of multiple shapes like circle, square, triangle, pentagon etc. Lets assume this picture as a test image. In image processing different CNN models classify the image by detecting some specific features of the test image by comparing with the dataset image. Here we can see that in the above Figure 2.1 by applying image processing on test image we can detect that it contains the shape pentagon like the dataset image. So by detecting this kind of similar patterns we can cluster any image based on its similarity with datasets.

2.2 Data Augmentation

Data augmentation is known as a technique to increase the amount of data by simply creating multiple copies of an image. The main advantage of data augmentation is, it can increase a huge number of dataset but all of them resulted into same output. It helps to reduce the overfitting property of a model. There are some common methods which can be used for data augmentation, such as :

- Flipping the image horizontally or vertically
- Rotating the image in different angles (+90 ,-90 ,+60 ,-60)
- Resizing and rescaling the image
- Cropping the image
- Increasing or decreasing the contrast and brightness of the image
- Adding noise to the image

We can understand the process of Data Augmentation more accurately from the given picture.

Figure 2.2: Common steps for Data Augmentation [15]

These steps make the CNN model stronger to detect or classify output accurately from the test images in more difficult conditions or scenarios. It can also be helpful to balance a dataset if number of images are comparatively low in one class than others.

2.3 ANN And CNN

2.3.1 Artificial Neutral Network (ANN)

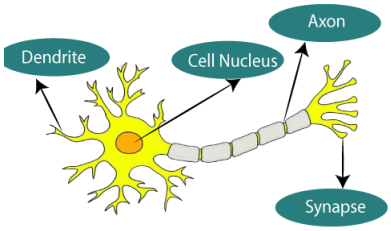


Figure 2.3: Human Brain Neural System [16]

Artificial Neutral Network can be described as Human brain. Working flow of ANN and a human brain is quite similar. We know that human brain is consisted of different types of cells like Dendrites, Cell nucleus, Synapse and Axon. In human brain Dendrites work as an input machine. It receives input from the body and sends those to cell nucleus. Cell nucleus processes the inputs and makes a decision based on the situation and sends it to axon. Axon works like an output for cell nucleus. If we compare Human brain system with Artificial Neural Network we can see that nodes in ANN are quite similar as cell nucleus which can take input and based on that it can do multiple operation and sends the output to another node via link. The input of a node in ANN can be compared with Dendrites of human brain. The output of each node is known as activation. In ANN the links are associated with different weights. The weight of the link may change based on how accurate the output of the total neural system comes for those inputs. In short, we can say that ANN can be

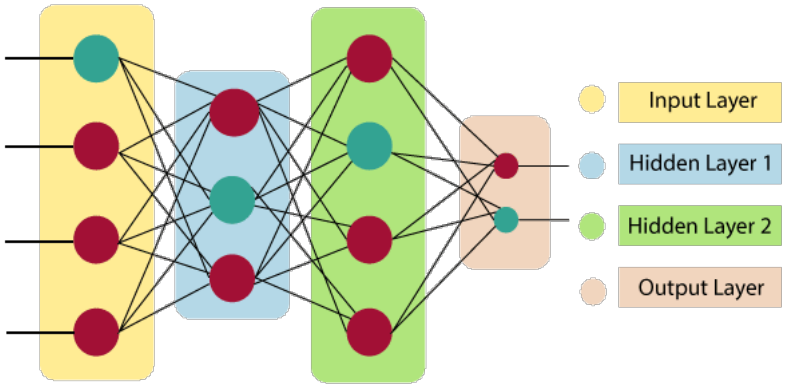


Figure 2.4: Artificial Neutral Network [17]

called as a artificial human brain system where it contains some input layer which takes input from surrounding then it can process the input data in hidden layers by performing some operation on them like neuron and try to get the best output from those input.

2.3.2 Convolutional Neural Network (CNN)

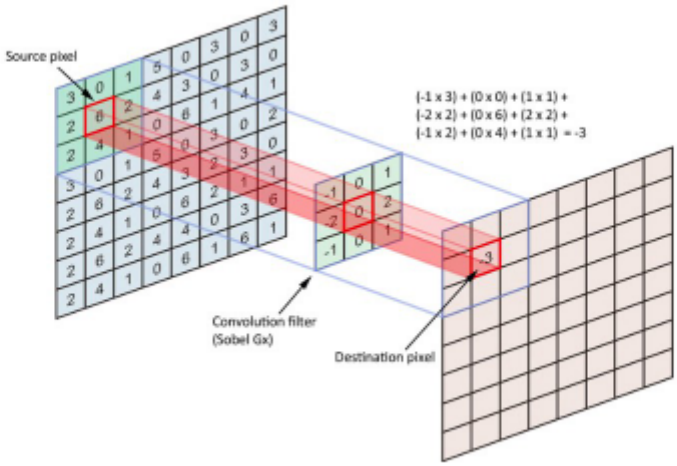


Figure 2.5: Calculation inside a convolutional layer in CNN [18]

Convolutional Neural Network can be defined as a special form of ANN. Here CNN have input block of hidden layer like ANN. But in CNN the hidden layer is Convolutional layer with filters which receives input information then transform it with some Convolutional filters and pooling then give output. Here filters mean a matrix and the dimensions(row*column) of that matrix are defined by us. At the initial stage filter matrix has random value in every cell. In CNN at first image data converted into multidimensional array format. Then a frame/matrix, which size is equal to the size of filter matrix size, convulate through the whole multidimensional array that we get from the image. Then a dot operation is performed between that filter matrix which convulates the frame and stores the value into a new matrix which dimension is same as the dimension of matrix that we get from image. Then the matrix goes through inside a pooling layer. This layer reduces the dimension of the image for detecting the feature inside a image more accurately. This procedure happens in a single convolutional layer block. This block gives a final output and it goes to another block of convolutional layers as a input. This is the core idea behind how the CNN works for feature selection of an image. After passing through all convolutional layer blocks the final output goes through more three layers named flatten, fully connected and softmax. Flatten

layer converts the final output into a 1D matrix. This three layers are used for classifying the input images.

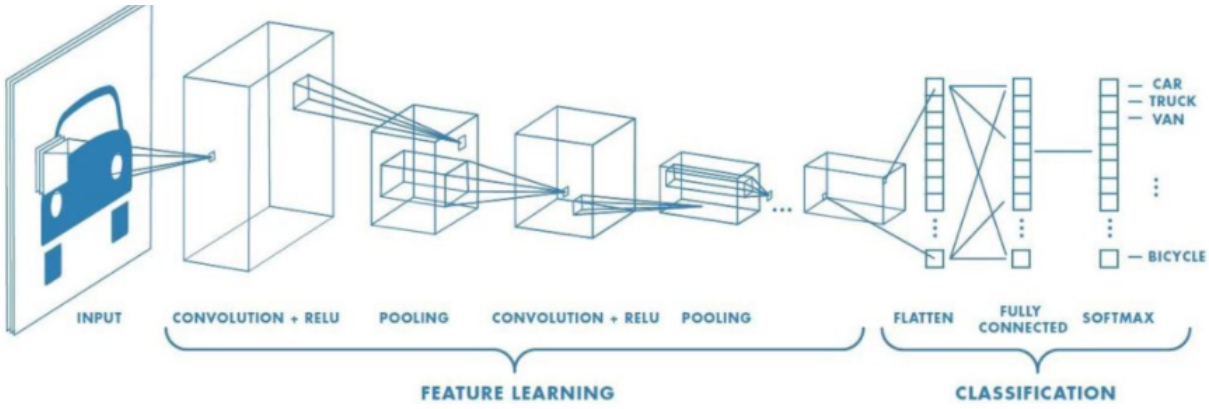


Figure 2.6: Convolutional Neural Network [19]

CNN models is good for pattern detection of images. Here pattern means shape, text, edges, corners of the images. In the initial convolutional layer blocks, the model can detect edges, square, different shapes etc. The deeper the convolutional layer blocks go, the better the model can precisely detects objects.

2.4 Activation Function [1]

Activation functions are mathematical equations that determine the output of a neural network model. Activation functions also have a major effect on the neural networks ability to converge and the speed of the convergence. In some cases activation functions may prevent neural networks from converging in the first place. Activation function also helps to normalize the output of any input in the range between 1 to -1 or 0 to 1. Activation function must be efficient and it should reduce the computation time because the neural network sometimes trained on millions of data points. Lets consider the simple neural network model(Figure 2.7) without any hidden layer.

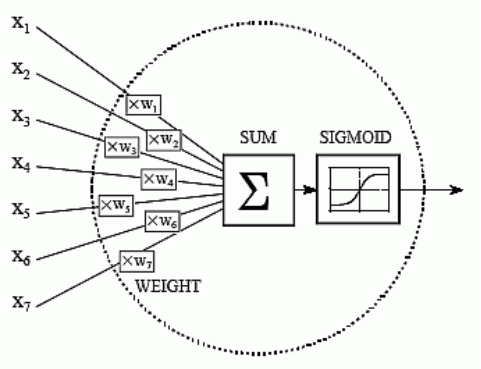


Figure 2.7: Simple Neural Network Model Without Any Hidden Layer [20]

Here is the output -

$$Y = \text{(weights * input + bias)}$$

and it can range from - infinity to + infinity. So it is necessary to bound the output to get the desired prediction or generalized results.

$$Y = \text{Activation function((weights * input + bias))}$$

So the activation function is an important part of an artificial neural network. They decide whether a neuron should be activated or not and it is a non-linear transformation that can be done on the input before sending it to the next layer of neurons or finalizing the output. Properties of an activation functions :

1. Non Linearity
2. Continuously differentiable
3. Range
4. Monotonic
5. Approximates identity near the origin

Some activation functions are explained below.

2.4.1 Binary Step Function

A binary step function is generally used in the Perceptron linear classifier. It thresholds the input values to 1 and 0, if they are greater or less than zero, respectively.

$$a_j^i = f(z_j^i) = \begin{cases} 0 & \text{if } z_j^i < 0 \\ 1 & \text{if } z_j^i > 0 \end{cases}$$

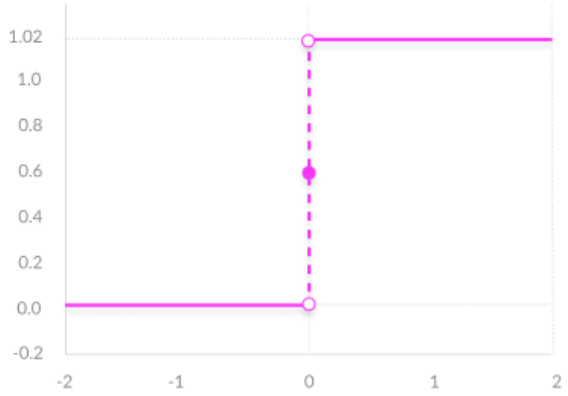


Figure 2.8: Binary Step Function Graph [21]

The step function is mainly used in binary classification problems and works well for linearly severable problems. It can not classify the multi-class problems.

2.4.2 Linear Activation Function

The equation for Linear activation function is:

$$\mathbf{f(x) = a.x}$$

When a = 1 then f(x) = x and this is a special case known as identity.

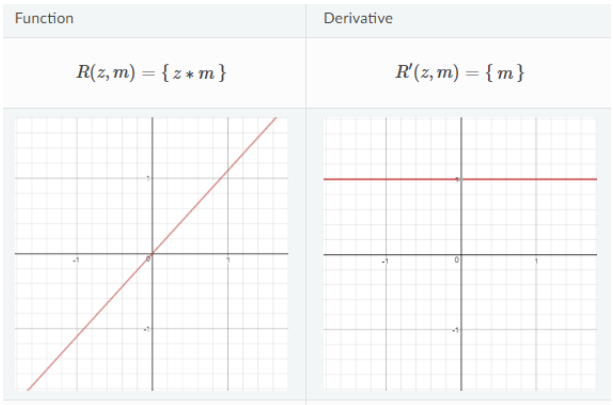


Figure 2.9: Linear Activation Function Graph [22]

The properties of linear activation function are:

- Range is - infinity to + infinity
- Provides a convex error surface so optimisation can be achieved faster
- df(x)/dx = a which is constant. So cannot be optimised with gradient descent

But this linear activation function has some limitations. As the derivative of the function is constant, it can be concluded that the gradient has no relation with the given input. Also the back propagation is constant as the change is delta x.

2.4.3 Non-Linear Activation Function

Modern neural network models use non-linear activation functions. They allow the model to create complex mappings between the network's inputs and outputs, such as images, video, audio, and data sets that are non-linear or have high dimensionality.
Some of the Non-Linear Activation functions are explained below.

Sigmoid Activation Function

The sigmoid function is a logistic function and the output is ranging between 0 and 1.

$$\phi(s_k) = \frac{1}{1+e^{-s_k}} = \frac{e^{s_k}}{1+e^{s_k}}$$

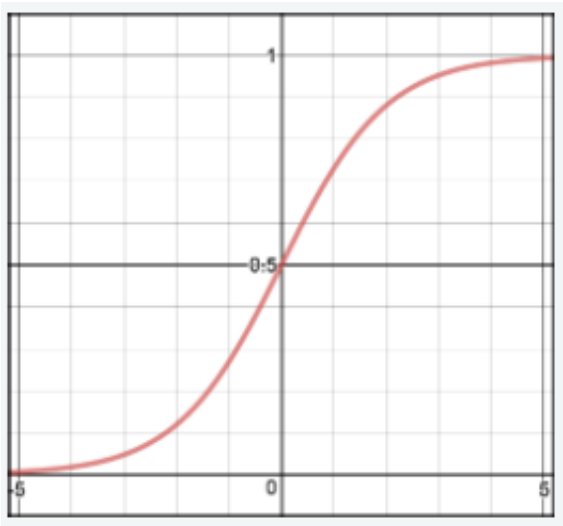


Figure 2.10: Sigmoid Activation Function Graph [23]

The output of the activation function is always going to be in range (0,1) compared to (-inf, inf) of linear function. It is non-linear, continuously differentiable, monotonic, and has a fixed output range. But it is not zero centred.

Problems with Sigmoid Activation Functions

1. **Vanishing Gradients Problem**
   The main problem with deep neural networks is that the gradient diminishes dramatically as it is propagated backward through the network. The error may be so small by the time it reaches layers close to the input of the model that it may have very little effect. As such, this problem is referred to as the "vanishing gradients" problem.

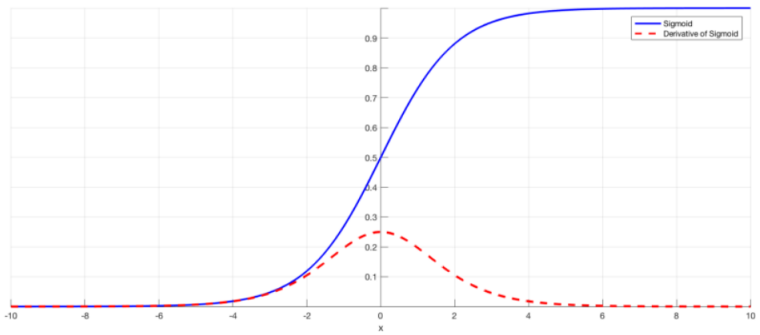


Figure 2.11: Vanishing Gradients [24]

A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network.

2. **Exploding Gradients**
   Exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training. These large updates in turn results in an unstable network. At an extreme, the values of weights can become so large as to overflow and result in NaN values.

Softmax

The softmax function is sometimes called the soft argmax function, or multi-class logistic regression. This is because the softmax is a generalization of logistic regression that can be used for multi-class classification, and its formula is very similar to the sigmoid function which is used for logistic regression. The softmax function can be used in a classifier only when the classes are mutually exclusive.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Rectified linear Units or ReLU

The rectified linear activation is the default activation when developing multilayer Perceptron and convolutional neural networks. It also overcomes the vanishing gradient problem, allowing models to learn faster and perform better.ReLU is the most commonly used activation function in neural networks. The mathematical equation for ReLU is:

$$\mathbf{ReLU(x) = max(0,X)}$$

So if the input is negative, the output of ReLU is 0 and for positive values, it is x.
Though it looks like a linear function, it's not. ReLU has a derivative function and allows for backpropagation. There is one problem with ReLU. Let's suppose most of the input values are negative or 0, the ReLU produces the output as 0 and the neural network can't perform the back propagation. This is called the Dying ReLU problem. Also, ReLU is an unbounded function which means there is no maximum value.

- Pros:

  1. Less time and space complexity.

2. Avoids the vanishing gradient problem.

- Cons:

1. Introduces the dead relu problem.
2. Does not avoid the exploding gradient problem.

2.5 Forward and Backward Propagation [2]

Forward Propagation is the process by means of which a neural network takes input data and keeps on producing another value, which is fed into the subsequent layer of neural network. This implies that the network is 'Connected'. Finally, the output is put into a loss function and used to find the loss.
Backward Propagation is the process where gradients which is the derivative of loss computed in weights/parameters of a neural network are computed. First, the gradient of parameters of last layer is computed. Now, this computed gradient is used to compute the gradients of penultimate layer and so on. Basically in back prop, information flows from right to left (output layer to input layer) as against left to right (input to output layer) in forward propagation. And remember, as and when gradients of a particular layer are found, its weights are also simultaneously updated by the formula :

parameter = parameter - learning rate * parameter gradient

Again, as the information of backward propagation reaches the first layer, weights of all layers had been updated by now, so again starts the process of forward propagation. This way the neural network keeps updating parameters and makes better and better predictions. Forward Propagation and Backward Propagation also known as Forward pass and Backward pass.

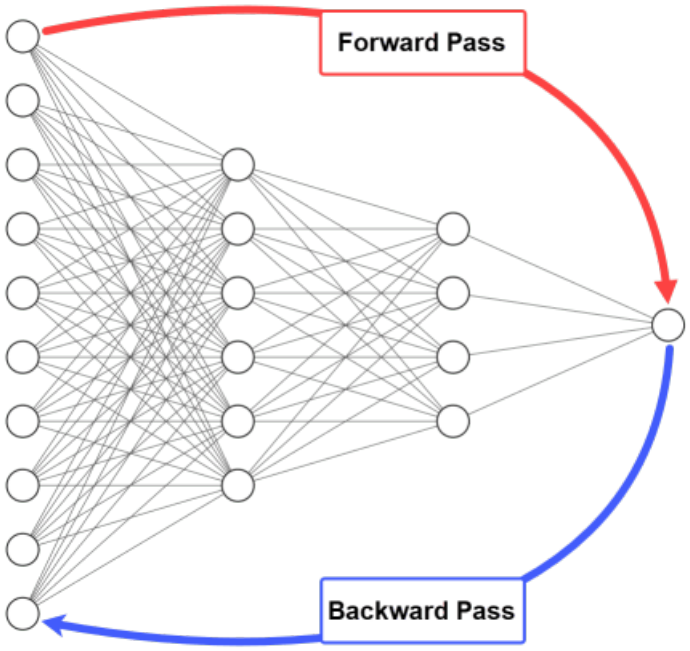


Figure 2.12: Forward and Backward Propagation [25]

In simple terms we randomly initializing the weights and the biases for the neurons in the hidden layers just because we do not know how important a feature is. During forward propagation the network computes the loss based on the initialized weights and only during back propagation it tries to update the weights and biases based on the gradients it has computed against the loss (This is where the actual learning happens). By repeating these steps for multiple iterations help to increase models accuracy.

2.6 Optimizer

Optimizers are algorithms or methods that used to minimize an error function (loss function) or to maximize the efficiency of production. Optimizers are mathematical functions which are dependent on models learnable parameters i.e weights and biases. Optimizers help to know how to change weights and learning rates of neural network to reduce the losses.

2.6.1 Gradient Descent [3]

Gradient descent is an optimization algorithm based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum. Gradient Descent iteratively reduces a loss function by moving in the direction opposite to that of steepest ascent. It depends on the derivatives of the loss function for finding minima. It uses the

data of the entire training set to calculate the gradient of the cost function to the parameters which requires large amount of memory and slows down the process.

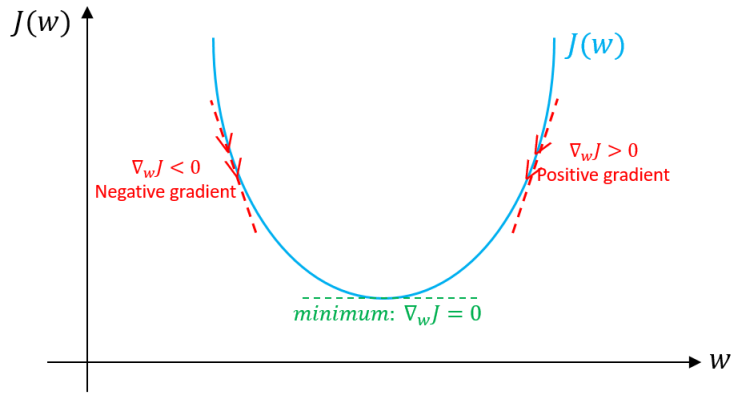


Figure 2.13: Gradient Descent [26]

$$W_{new} = W_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$

Advantages

1. Easy to understand.
2. Easy to implement.

Disadvantages

1. As the method calculates the gradient for the entire data set in one update, the calculation is very slow.
2. It requires large memory and it is computationally expensive.

2.6.2 Learning Rate

How big the steps of gradient descent takes into the direction of the local minimum are determined by the learning rate, which figures out how fast or slow we will move towards the optimal weights. For gradient descent to reach the local minimum we must set the learning rate to an appropriate value, which is neither too low nor too high. This is important

because if the steps it takes are too big, it may not reach the local minimum because it bounces back and forth between the convex function of gradient descent. If we set the learning rate to a very small value, gradient descent will eventually reach the local minimum but that may take a while.

Figure 2.14: Learning Rate [27]

2.6.3 Stochastic Gradient Descent [4]

In this method one training sample (example) is passed through the neural network at a time and the parameters (weights) of each layer are updated with the computed gradient. So, at a time a single training sample is passed through the network and its corresponding loss is computed. The parameters of all the layers of the network are updated after every training sample. For example, if the training set contains 100 samples then the parameters are updated 100 times that is one time after every individual example is passed through the network. For stochastic gradient descent the equation of gradient descent is iterated over 'n' times for 'n' training samples in the training set.

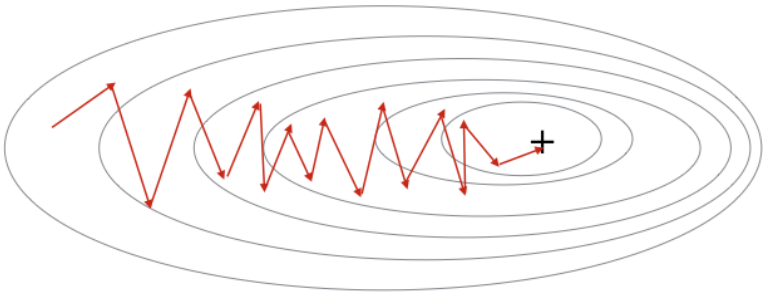


Figure 2.15: Stochastic Gradient Descent [28]

Advantages

1. It is easier to fit into memory due to a single training sample being processed by the network.

2. It is computationally fast as only one sample is processed at a time.
3. For larger datasets it can converge faster as it causes updates to the parameters more frequently.
4. Due to frequent updates the steps taken towards the minima of the loss function have oscillations which can help getting out of local minimums of the loss function (in case the computed position turns out to be the local minimum).

Disadvantages

1. Due to frequent updates the steps taken towards the minima are very noisy. This can often lead the gradient descent into other directions.
2. Also, due to noisy steps it may take longer to achieve convergence to the minima of the loss function.
3. Frequent updates are computationally expensive due to using all resources for processing one training sample at a time.
4. It loses the advantage of vectorized operations as it deals with only a single example at a time.

2.6.4 Adaptive Moment Extimation(Adam) [5]

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface. We compute the decaying averages of past and past squared gradients m_t and v_t respectively as follows :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e.

β_1 and β_{12} are close to 1). They counteract these biases by computing bias-corrected first and second moment estimates :

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

They then use these to update the parameters just as we have seen in Adadelta and RMSprop, which yields the Adam update rule :

$$\theta_{t-1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$$

Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

2.7 Confusion Matrix [6]

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Figure 2.16: Confusion Matrix

Confusion Matrix is a multidimensional matrix which can describe the performance of a classification model on a test dataset. For that dataset, actual or true values for those test data must be known. Column's in confusion matrix holds the actual result and rows in confusion matrix holds the predicted result for that test dataset. There are four types of values can be found in the confusion matrix. They are :

- True Positive (TP) : The model predicted yes and the actual output is yes for that test data.
- True Negative (TN) : The model predicted no and the actual output is no for that test data.
- False Positive (FP) : The model predicted yes and the actual output is no for that test data. This can be called type-1 error.

- False Negative (FN) : The model predicted no and the actual output is yes for that test data. This can be called type-2 error.

Lets assume 12 patients who are suspected that are infected with Covid-19. Among them 8 people who are not infected with covid-19 (0) and 4 of them are infected (1). Lets run a model on their collected symptoms dataset. Lets assume the actual dataset has = 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1 And the model predict = 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0 So we can see that the model gives 9 accurate predictions and 3 wrong predictions. Among of total 3 wrong predictions 2 of them has not infected by covid-19 in real life but the model detect them as positive. And the remaining one was detected negative by the model but it was positive. So for upper case confusion matrix will be :

Predicted Class / Actual class	Not Affected	Affected
Not Affected	6	2
Affected	1	3

Table 2.1: Confusion Matrix For Given Case

2.8 Tools

2.8.1 Pytorch [7]

PyTorch is a Python-based scientific computing package that uses the power of graphics processing units. It is also one of the preferred deep learning research platforms built to provide maximum flexibility and speed. It is known for providing two of the most high-level features; namely, tensor computations with strong GPU acceleration support and building deep neural networks on a tape-based autograd systems. PyTorch is also great for deep learning research and provides maximum flexibility and speed. PyTorch Tensors are very similar to NumPy arrays with the addition that they can run on the GPU. This is important because it helps accelerate numerical computations, which can increase the speed of neural networks by 50 times or greater.
There is some reason for using Pytorch in research :

- **Dynamic Computational graphs [37] :** PyTorch uses different backends for CPU, GPU and for various functional features rather than using a single back-end. It uses

tensor backend TH for CPU and THC for GPU. While neural network backends such as THNN and THCUNN for CPU and GPU respectively. Using separate backends makes it very easy to deploy PyTorch on constrained systems.

- **Imperative style [37] :** PyTorch library is specially designed to be intuitive and easy to use. When you execute a line of code, it gets executed thus allowing you to perform real-time tracking of how your neural network models are built. Because of its excellent imperative architecture and fast and lean approach it has increased overall PyTorch adoption in the community.
- **Highly extensible [37] :** PyTorch is deeply integrated with the C++ code, and it shares some C++ backend with the deep learning framework, Torch. Thus allowing users to program in C/C++ by using an extension API based on cFFI for Python and compiled for CPU for GPU operation. This feature has extended the PyTorch usage for new and experimental use cases thus making them a preferable choice for research use.
- **Python-Approach [37] :** PyTorch is a native Python package by design. Its functionalities are built as Python classes, hence all its code can seamlessly integrate with Python packages and modules. Similar to NumPy, this Python-based library enables GPU-accelerated tensor computations plus provides rich options of APIs for neural network applications. PyTorch provides a complete end-to-end research framework which comes with the most common building blocks for carrying out everyday deep learning research. It allows chaining of high-level neural network modules because it supports Keras-like API in its torch.nn package.

2.8.2 Python [8]

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is an open-source language, anyone can use Python to code. What's more, there is a community that supports and develops the ecosystem, adding their own contributions and libraries. It supports functional and structured programming methods as well as OOP. Python also provides very high-level dynamic data types and supports dynamic type checking. Here are the reasons why Python is so popular :

- **Simple Syntax [38] :** Python is relatively easy to read and understand, as its syntax is more like English. Its straightforward layout means that you can work out what each line of code is doing.
- **Versatility [38] :** There are many uses for Python. Whether you're interested in data visualisation, artificial intelligence or web development, you can find a use for the

language.

- **Interpreted Language [38] :** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Interactive Language [38] :** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portability [38] :** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **A broad standard library [38] :** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **GUI Programming [38] :** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

In recent time python used in every sector like Artificial Intelligence and Machine Learning, Data analytics, Data visualisation, Programming applications, Web development, Game development, search engine optimisation (SEO) etc.

2.9 Clustering Algorithm

Clustering is one of the most common exploratory data analysis technique used to get an intuition about the structure of the data. It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different. [9]

2.9.1 K Means Clustering Algorithm [9]

Unlike supervised learning, clustering is considered an unsupervised learning method. K Means algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. K-means which is considered as one of the most used clustering algorithms due to its simplicity.
The k-means clustering algorithm mainly performs two tasks:

1. Determines the best value for K center points or centroids by an iterative process.
2. Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

The Work flow of K-Means [39]

1. Select the number K to decide the number of clusters.
2. Select random K points or centroids.
3. Assign each data point to their closest centroid, which will form the predefined K clusters.
4. Calculate the variance and place a new centroid of each cluster.
5. Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
6. If any reassignment occurs, then go to step-4 else go to FINISH.
7. the model is ready.

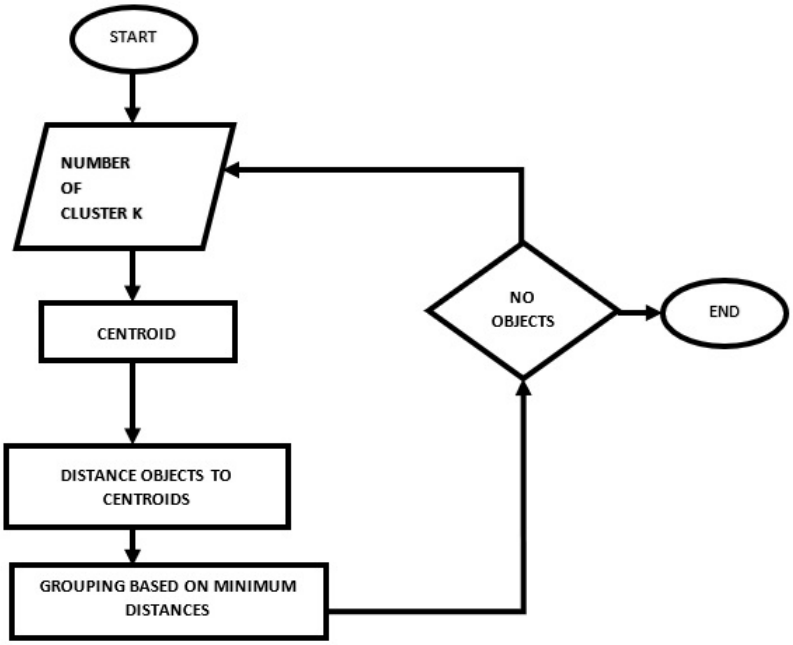


Table 2.2: Workflow of K-means

The approach k means follows to solve the problem is called Expectation-Maximization. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster. The objective function is:

$$J = \sum_{i=1}^{m}\sum_{k=1}^{K} \omega_{\mathrm{ik}} \parallel x^{\mathrm{i}} - \mu_{\mathrm{k}} \parallel^{2}$$

where $\omega_{\mathrm{ik}} = 1$ for data point x^{i} if it belongs to cluster k; otherwise, $\omega_{\mathrm{ik}} = 0$. Also, k is the centroid of x^{i}'s cluster.It's a minimization problem of two parts. We first minimize J with respect to ω_{ik} and treat μ_{k} fixed. Then we minimize J with respect to μ_{k} and treat ω_{ik} fixed. Technically speaking, we differentiate J with respect to ω_{ik} first and update cluster assignments (E-step). Then we differentiate J with respect to μ_{k} and recompute the centroids after the cluster assignments from previous step (M-step). Therefore, E-step is:

$$\frac{\delta J}{\delta \omega_i k} = \sum_{i=1}^{m}\sum_{k=1}^{K} \omega_{\mathrm{ik}} \parallel x^{\mathrm{i}} - \mu_{\mathrm{k}} \parallel^{2}$$

$$\Rightarrow \omega_{ik} = \begin{cases} 1 & \text{if } k = argmin_j \parallel x^i - \mu_j \parallel^2 \\ 0 & \text{otherwise} \end{cases}$$

In other words, assign the data point x^{i} to the closest cluster judged by its sum of squared distance from cluster's centroid. And M-step is:

$$\frac{\delta J}{\delta \mu_k} = 2\sum_{i=1}^{m} \omega_{ik}(x^i - \mu_k) = 0$$

$$\Rightarrow \mu_k = \frac{\sum_{i=1}^{m} \omega_{ik} x^i}{\sum_{i=1}^{m} \omega_{ik}}$$

Which translates to recomputing the centroid of each cluster to reflect the new assignments. K Means clustering is one of the most popular clustering algorithms and usually the first thing practitioners apply when solving clustering tasks to get an idea of the structure of the dataset. The goal of *K-means* is to group data points into distinct non-overlapping subgroups. It does a very good job when the clusters have a kind of spherical shape. However, it suffers as the geometric shapes of clusters deviate from spherical shapes. Moreover, it also doesn't learn the number of clusters from the data and requires it to be pre-defined. [9]

Chapter 3

Implemented Models

3.1 Workflow of different Models

We implemented four approach for detecting diabetic retinopathy. Approach 1 is the binary classification for the Diabetic Retinopathy. Where Approach 2 and Approach 3 can detect multiple class of Diabetic Retinopathy. In every approach we use different models or architectures of Deep Learning. The Approach 4 is different than others. In this we first apply clustering algorithm on a new dataset then we classify the classes in those clusters.

3.1.1 Workflow of Approach 1 [10]

Preprocessing

Retinal images were provided by EyePACS [40], a free platform for retinopathy detection. A clinician has rated the presence of diabetic retinopathy in each image on a scale of 0 to 4. In this process batch size 16 has been used. 7600 images from level 0 as healthy cases and 7600 images from level 1 to 4 as diseased ones have been randomly chosen. The images are re-scaled to 299x299 as they are provided by a public database which are not standardized. A data augmentation method has been used to increase the amount of data by rotating, flipping and taking parts of images.

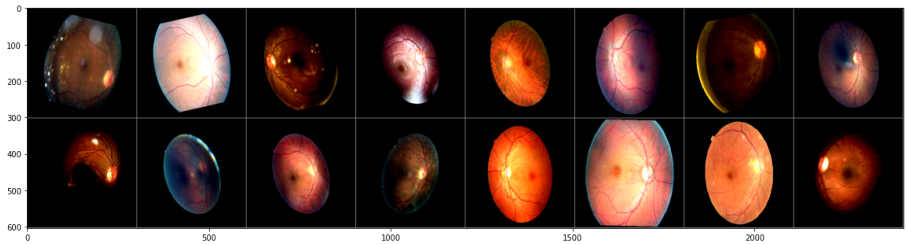


Figure 3.1: Retinal Fundus Images after Preprocessing

Here, we perform a binary classification, where we divide the dataset into two classes - one for healthy cases and one for diseased cases. The ratio between two classes is 1 to 1.

Class	Name	Number of images	Percentage
0	Healthy	7600	50%
1	Diseased	7600	50%

Table 3.1: Binary Classification

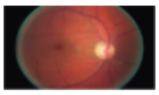
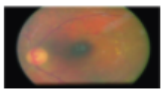
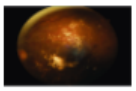
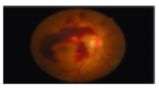
Levels	0	1			
Diagnosis	No DR	Mild DR	Moderated DR	Severe DR	PDR
Description	No abnormalities	The most early stage,where only microaneurysms Can happen	Ability of blood transportation And swelling with the progress of the disease	Blood supply to the retina due to increased blockage of more blood vessels	The advanced stage, where the growth features secreted by the retina activate proliferation of new blood vessels
Sample images with there levels					

Table 3.2: Stages of Binary Classification

Methodology

Pre-trained architectures of different neural networks have been tested. A pre-trained network means a network which already contains trained filters such as edge detection or corners in their first layers. As the network models InceptionV3, Densenet121 and VGG16 are Pre-trained, they already have such filters and are trained in the ImageNet database [41]. The output layer of each network uses Softmax as an activation function which generates the probabilities of how much the input belongs to the set of different classes(Healthy, Diseased). Here Adam optimizer has been used and 20 epochs have been performed with

0.0001 learning rate. We use the pre-processed trained image into different models for training them. During training we used batch size 16 for every model. A short procedure details for every model are given below. In this method we applied five models on a single dataset. Those models are :

- Densenet121
- InceptionV3
- VGG16

Densenet121

This DenseNet121 model is consist of some dense blocks. In a dense block there are multiple dense layers. We can see in the bottom portion of the Figure 3.2 every layer is adding to the previous volume these 32 new feature maps. This is why we go from 64 to 256 after 6 layers. In addition, Transition Block performs as 1x1 convolution with 128 filters followed by a 2x2 pooling with a stride of 2, resulting on dividing the size of the volume and the number of feature maps on half. That is the procedure inside one dense block. The overview of the total process are given in the upper portion of the Figure 3.2.

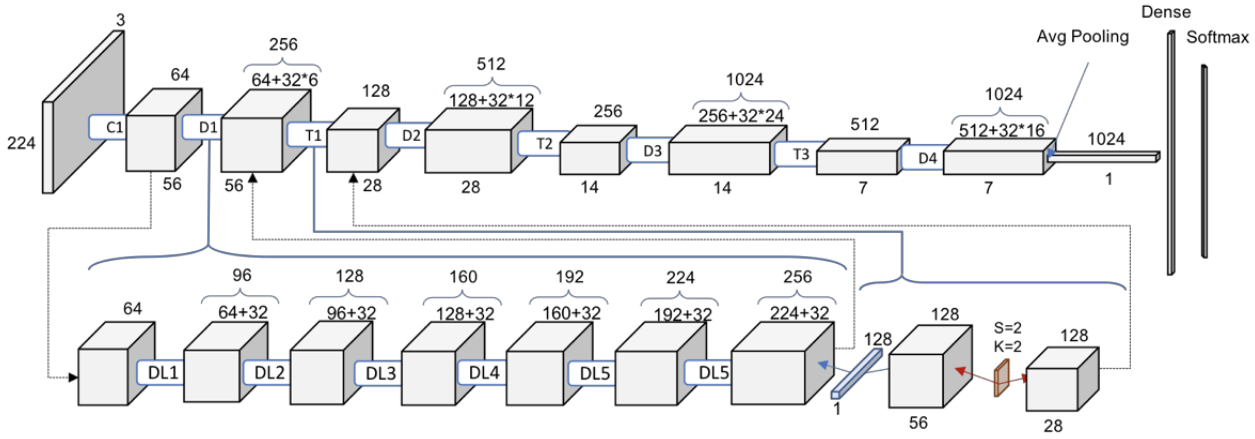


Figure 3.2: Procedure of Densenet121 [29]

VGG16 [42]

This model is consist of multiple convulational blocks.
The input to the model is image of dimensions (224, 224, 3). The first two layers have 64 channels of 3*3 filter size and all have same padding. Then after a max pool layer of stride (2, 2), two layers which have convolution layers of 256 filter size and filter size (3, 3). This followed by a max pooling layer of stride (2, 2) which is same as previous layer. Then there

are 2 convolution layers of filter size (3, 3) and 256 filter. After that there are 2 sets of 3 convolution layer and a max pool layer. Each have 512 filters of (3, 3) size with same padding. This image is then passed to the stack of two convolution layers.
After the stack of convolution and max-pooling layer, we got a (7, 7, 512) feature map. We flatten this output to make it a (1, 25088) feature vector by flatten layer. After this there are 3 fully connected layer, the first layer takes input from the last feature vector and outputs a (1, 4096) vector, second layer also outputs a vector of size (1, 4096) but the third layer output a 1000 channels for 1000 classes of ILSVRC challenge, then after the output of 3rd fully connected layer is passed to softmax layer in order to normalize the classification vector. After the output of classification vector top-5 categories for evaluation. All the hidden layers use ReLU as its activation function.

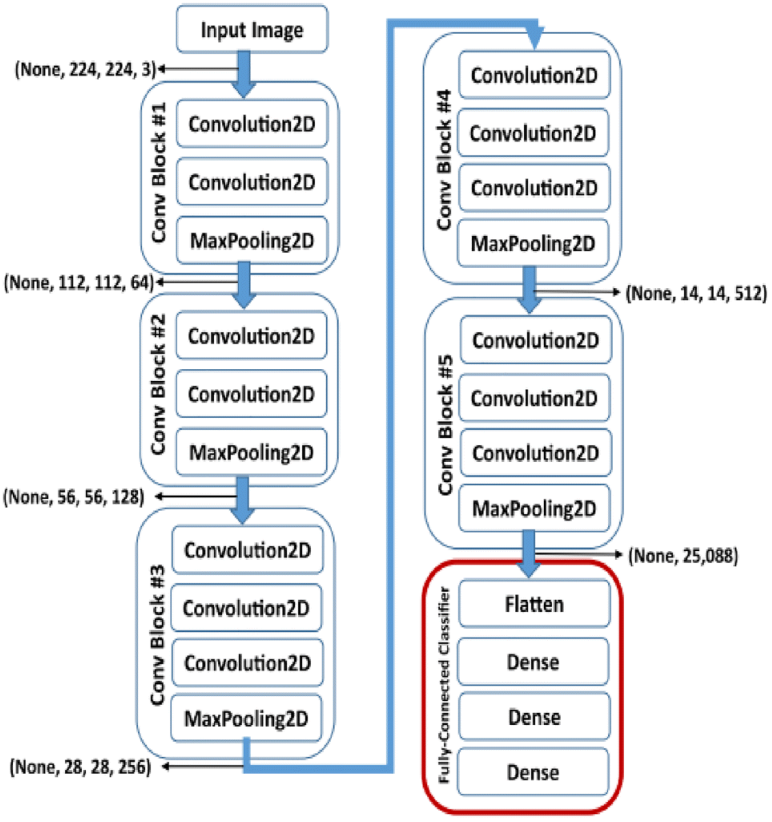


Figure 3.3: Procedure of VGG 16 [30]

InceptionV3 [43]

InceptionV3 is similar to InceptionV2. But it contains some changes. That are :

- Use of 7×7 factorized Convolution
- Batch Normalization in the fully connected layer of Auxiliary classifier

- Label Smoothing Regularization which is a method to regularize the classifier by estimating the effect of label-dropout during training. It prevents the classifier to predict too confidently a class. The addition of label smoothing gives 0.2 percent improvement from the error rate.

Inception Layer is a combination of multiple Convolutional layers like (1×1 Convolutional layer, 3×3 Convolutional layer, 5×5 Convolutional layer) with their layers output filtered and summed in a single output vector. Which is the input for the next stage. Parallel Max Pooling layer provides another option to add or create more inception layer. Another 1×1 Convolutional layer used before applying another layer for reduce the dimension.

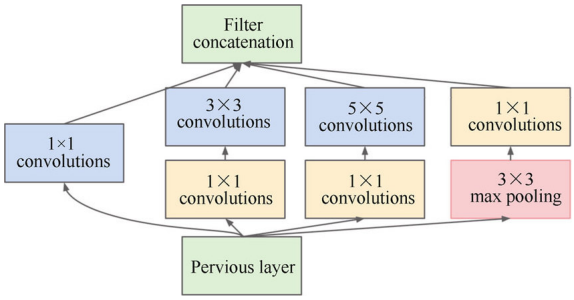


Figure 3.4: Architecture of Inception Layer [31]

Working Procedure of Approach 1

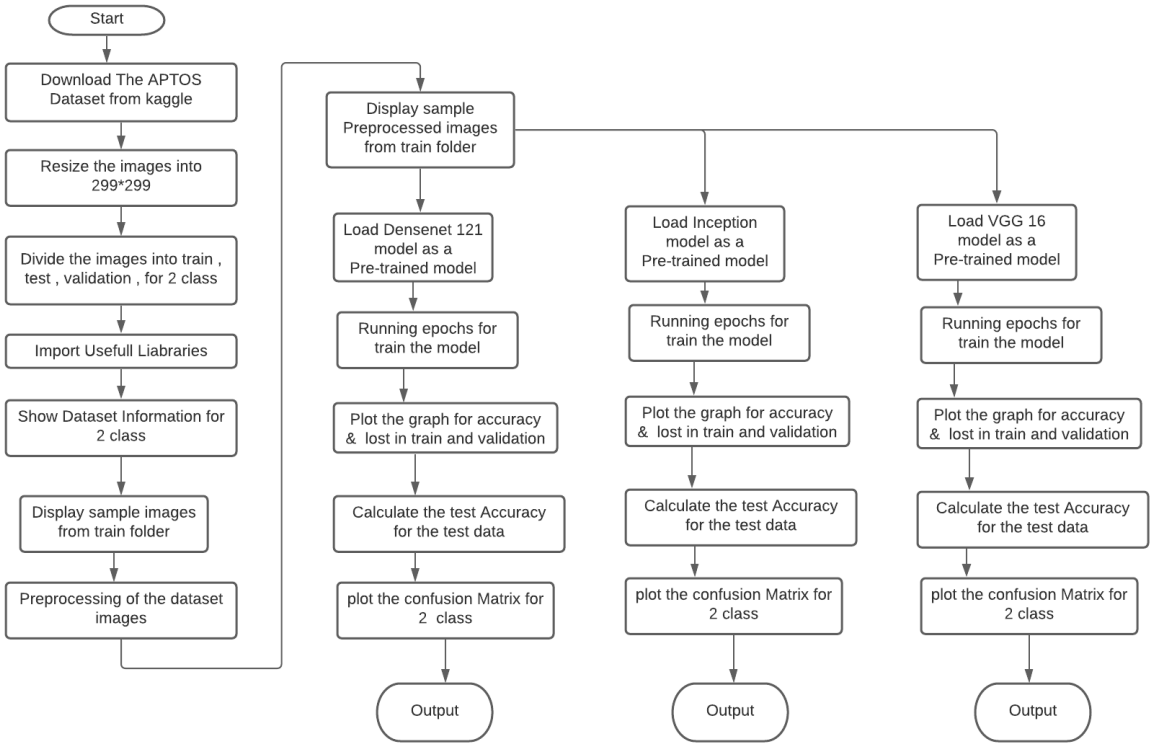


Figure 3.5: Workflow of Approach 1

3.1.2 Workflow of Approach 2 [11]

Preprocessing

In this approach we start the workflow by downloading the APTOS dataset which are used by the researchers in that paper. Then we found 4k resolutions image there. So we resize the images into 299*299 pixels. This help us to reduce our dataset size. Then we divide the dataset images into three folders named train, test, validation for every class. The distribution amount of images for every class are given below.

	Test	Train	Validation
Class 0	400	1100	400
Class 1	400	1100	400
Class 2	400	1100	400
Class 3	400	1100	400
Class 4	400	1100	400
	2000	5500	2000

Table 3.3: Table for distributing the data

There we can see that we take total amount of 2000 image for test , 5500 image for train and 2000 image for validation . Then all of the images goes through some pre-processing steps. They are -

- Rotation
- Horizontal , Vertical flipping
- Resizing and cropping

Methodology

Pre-trained architectures of different neural networks have been tested. A pre-trained network means a network which already contains trained filters such as edge detection or corners in their first layers. As the network models InceptionV3, Densenet121 and Xception, Resnet50 are Pre-trained, they already have such filters and are trained in the ImageNet database [10]. The output layer of each network uses Softmax as an activation function which generates the probabilities of how much the input belongs to the set of 5 classes 0,1,2,3,4 .This 1,2,3,4 is the different stage of diabetic retinopathy. Here Adam optimizer has been used and 20 epochs have been performed with 0.0001 learning rate. Here we use VGG16 as a non pre-trained model. Then we use the pre-processed image into different models to train

them. A short procedure details for every model are given below.
In this method we applied five models on a single dataset. Those models are -

- Densenet121
- InceptionV3
- VGG16
- Xception
- ResNet50

Xception [44]

Xception module has three main parts. The Entry flow, the Middle flow (which is repeated eight times), and the Exit flow. The entry flow has two blocks of convolutional layer followed by a ReLU activation. The Figure 3.6 also mentions in detail the number of filters, the filter size (kernel size), and the strides .There are also various Separable convolutional layers. There are also Max Pooling layers. When the strides are different than one, the strides are also mentioned. For the Middle flow and the Exit flow, this diagram clearly explains the image size, the various layers, the number of filters, the shape of filters, the type of pooling, the number of repetitions, and the option of adding a fully connected layer in the end.

Separable convolutions consist of first performing a depthwise spatial convolution (which acts on each input channel separately) followed by a pointwise convolution which mixes the resulting output channels.

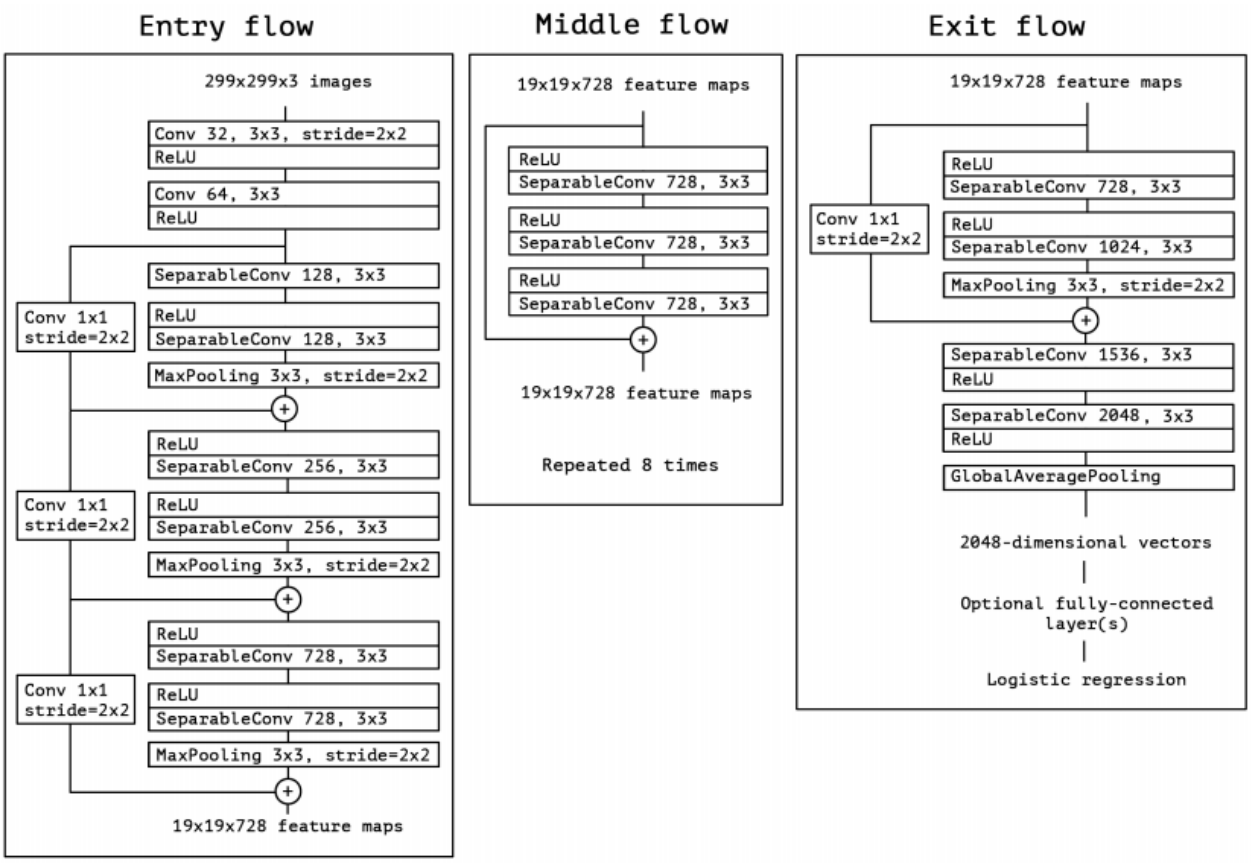


Figure 3.6: Procedure of Xception [32]

ResNet50 [45]

ResNet50 has a convoultion with a kernel size of 7 * 7 and 64 different kernels all with a stride of size 2 giving output as 1 layer. Next it has max pooling with also a stride size of 2. In the next convolution there is a 1 * 1, 64 kernel following this a 3 * 3, 64 kernel and at last a 1 * 1, 256 kernel. These three layers are repeated in total 3 time that produce 9 layers. Next there are kernel of 1 * 1, 128 after that a kernel of 3 * 3, 128 and at last a kernel of 1 * 1, 512 this step was repeated 4 time that produce 12 layers. After that there is a kernal of 1 * 1, 256 and two more kernels with 3 * 3, 256 and 1 * 1, 1024 and this is repeated 6 time that produce 18 layers. And then again a 1 * 1, 512 kernel with two more of 3 * 3, 512 and 1 * 1, 2048 and this was repeated 3 times that produce 9 layers. After that we do a average pool and end it with a fully connected layer containing 1000 nodes and at the end a softmax function so this required 1 layer.
So there is $1 + 9 + 12 + 18 + 9 + 1 = 50$ layers Deep Convolutional network .

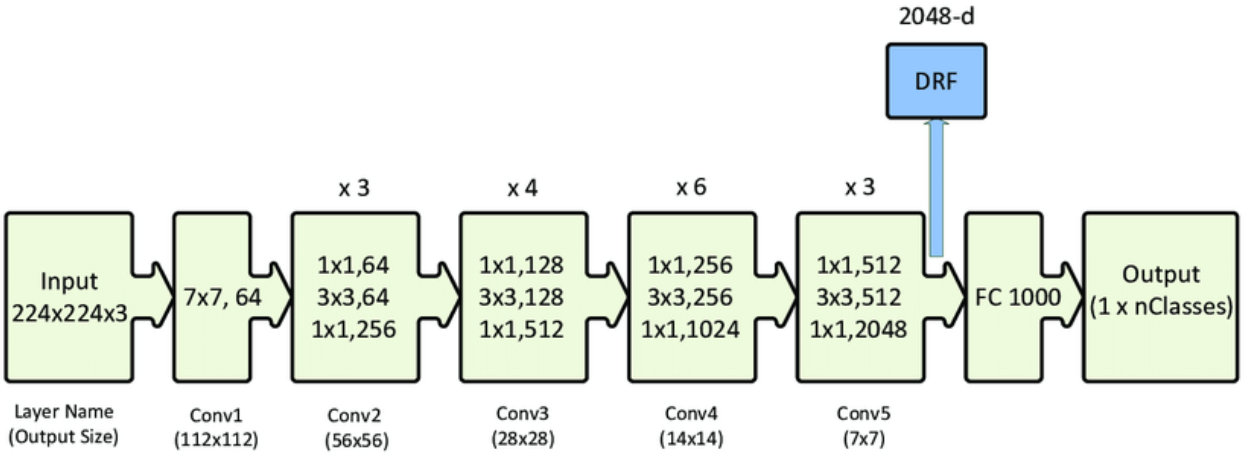


Figure 3.7: Procedure of ResNet50 [33]

Working Procedure of Approach 2

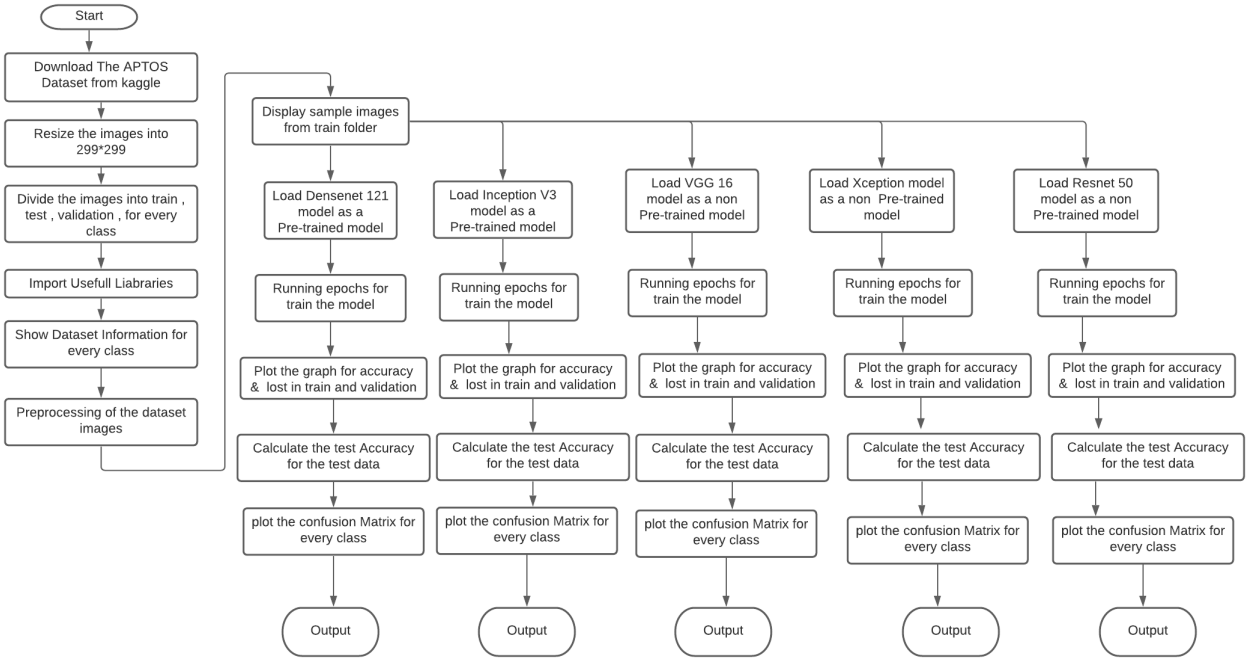


Figure 3.8: Workflow of Approach 2

3.1.3 Workflow of Approach 3 [12]

Preprocessing

In this approach the preprocessing steps are same as the preprocessing step for approach 2 [Section 3.1.2].

Methodology

Here we use the ensemble method with pre-trained network. A pre-trained network means a network which already contains trained filters such as edge detection or corners in their first layers. As the network models Inception V3, Densenet121 and Xception, ResNet50 are Pre-trained, they already have such filters and are trained in the ImageNet database[10]. The output of these network for a specific test data goes through a special step named ensemble which decides the probabilities of how much the input belongs to the set of 5 classes 0,1,2,3,4. This 1,2,3,4 is the different stage of diabetic retinopathy. Here 20 epochs have been performed with 0.0001 learning rate.

Ensemble Methods [46]

Ensemble methods are techniques that create multiple models and then combine them to produce improved results. Ensemble methods usually produces more accurate solutions than a single model would. It simply take the output from different model for a single test data and perform these operations :

- **Majority Voting :** Every model makes a prediction (votes) for each test instance and the final output prediction is the one that receives more than half of the votes. If none of the predictions get more than half of the votes, that concludes the ensemble method could not make a stable prediction for this instance.
- **Voting and Averaging Based Ensemble Methods :** Voting and averaging are two of the easiest ensemble methods. They are both easy to understand and implement. Voting is used for classification and averaging is used for regression. In both methods, the first step is to create multiple classification/regression models using some training dataset. Each base model can be created using different splits of the same training dataset and same algorithm, or using the same dataset with different algorithms, or any other method.
- **Weighted Voting :** Unlike majority voting, where each model has the same rights, we can increase the importance of one or more models. In weighted voting you count the

prediction of the better models multiple times. Finding a reasonable set of weights is dependent on human.

- **Simple Averaging :** In simple averaging method, for every instance of test dataset, the average predictions are calculated. This method often reduces overfit and creates a smoother regression model.

In this approach we take four models to create a ensemble method and then apply it on a single dataset. Those models are :

- Densenet121
- InceptionV3
- Xception
- ResNet50

```
import a set of pretrained models
H={'Inception_v3','Xception','Densenet121','Resnet50'}
lr=0.0001
foreach h∈H do
    for epochs=1 to 20
        foreach mini batch(X_i,Y_i)∈(X_train,Y_train) do
            Update the parameters of the model h
        end
    end
end

foreach x∈X_test do
    Ensemble the output of all models
end
```

Figure 3.9: Ensemble Algorithm

Working Procedure of Approach 3

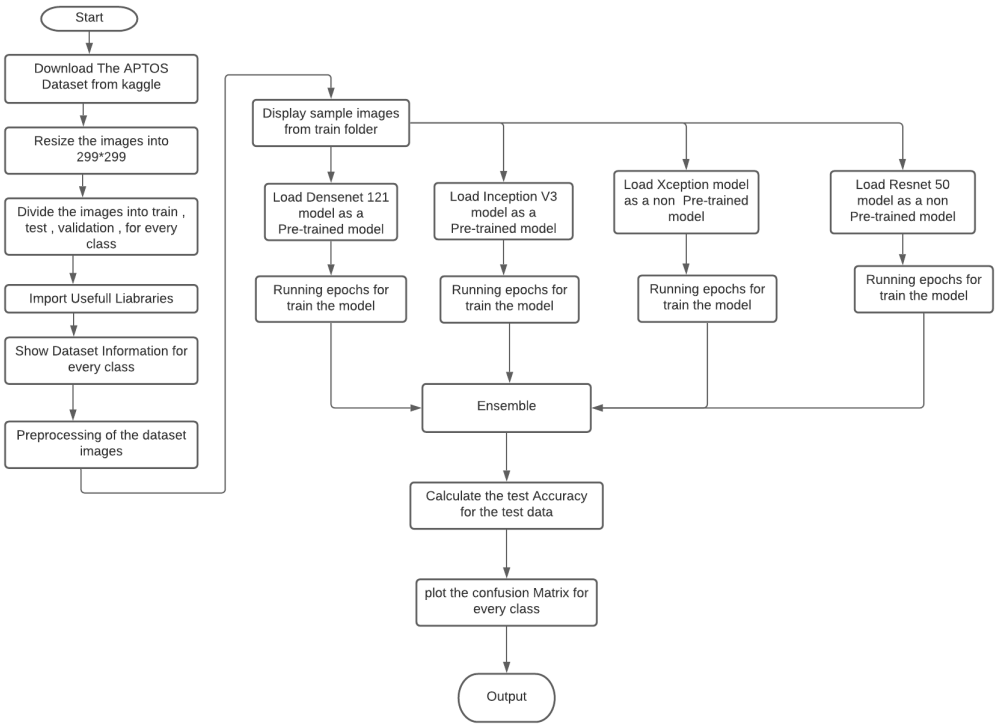


Figure 3.10: Workflow of Approach 3

3.1.4 Accuracy

From the confusion matrix we can calculate the accuracy in terms of positive and negative classes :

$$\text{Accuracy} = \text{TP} + \text{TN} \,/\, \text{TP} + \text{TN} + \text{FP} + \text{FN}$$

Here TP, TN, FP, FN means :

- True Positive (TP)
- True Negative (TN)
- False Positive (FP)
- False Negative (FN)

3.1.5 Recall/Sensitivity [13]

It can be called as True Positive rate. It measures the proportion of accurately diagnosed positive(i.e. the proportion of those who have some condition (affected) who are correctly identified as having the condition).

$$\text{Sensitivity} = \text{TP} \ / \ \text{TP} + \text{FN}$$

3.1.6 Specificity [13]

It can be called as True Negative rate. It measures the proportion of accurately diagnosed negative (i.e. the proportion of those who do not have the condition (unaffected) who are correctly identified as not having the condition).

$$\text{Specificity} = \text{TN} \ / \ \text{TN} + \text{FP}$$

3.1.7 Workflow of Approach 4

Preprocessing

We used APTOS 2019 Blindness Detection [34] dataset for this approach. First we converted the images to grayscale images and then the images were resized to 28X28 images. After that we converted the whole data into ***csv***(Comma Separated Values) file. The number of images in every class is given in table 3.4.

Class	Number of Images
0	1805
1	370
2	999
3	193
4	295

Table 3.4: Number of Images in APTOS [34] Dataset

Methodology

After these pre-processings, we divided the whole dataset into 5 segments using **K-Means** algorithm. The APTOS [34] dataset has 5 classes of retinal images. But it could be seen the differences between several classes are quiet negligible and it is very difficult to separate them. So, we used K-Means to cluster based on similarities of the images. But it could be seen that, in a cluster, images of other classes also exist.
After getting these 5 clusters, we take all the images every cluster separately into classification model of our own. For example, The cluster 0 has 953 images, which consist the images from all 5 classes. So we used those images to train and test in a classification model and get the results.

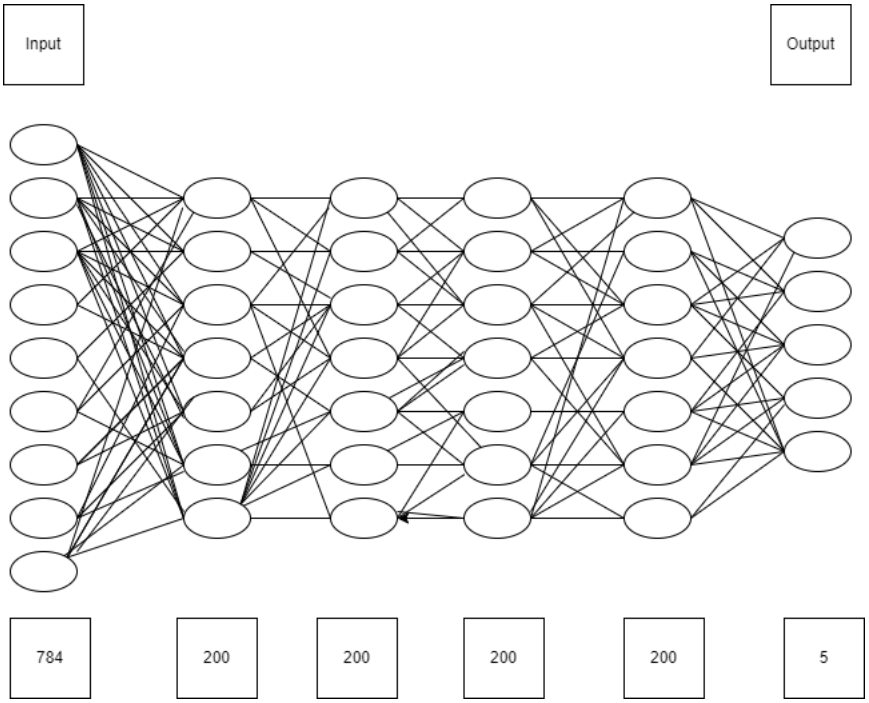


Figure 3.11: Our Model's Inner Layers

In the classification model we used 6 linear layers(fig. 3.11). The model runs 20 epochs. We used ReLU(Rectified Linear Unit) as the activation function and Softmax activation function for the output layer. The input layer take 784 inputs and final layer give us the output of 5.

Working Flow of Approach 4

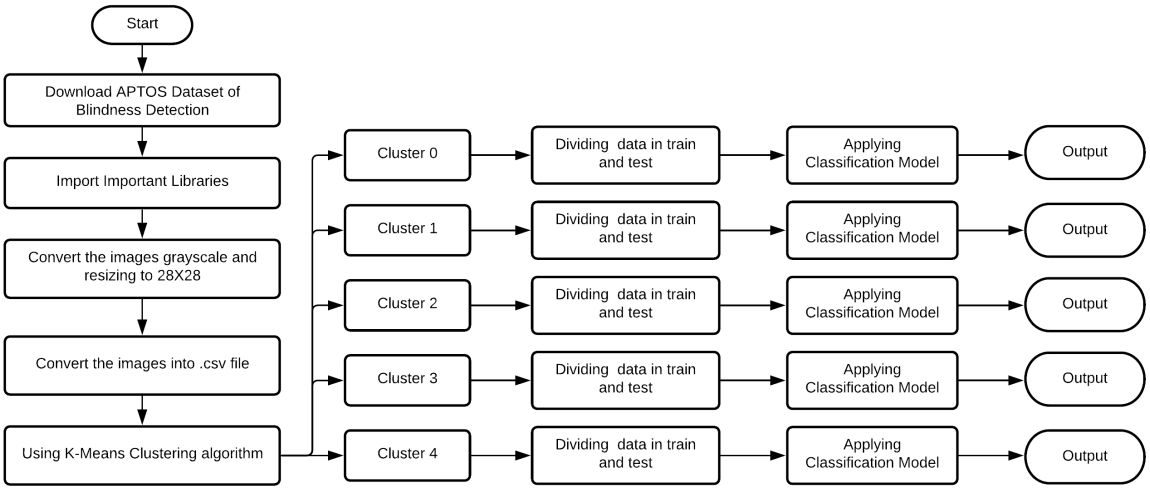


Figure 3.12: Workflow of Approach 4

Chapter 4

Simulation

4.1 Approach 1

Healthy and Diseased Retinal Eye image Detection using ***InceptionV3*** , ***DenseNet121*** and ***VGG16*** pre-trained models

4.1.1 Preprocessing

Here is the images of before preproessing(Figure 4.1) and after preprocessing(Figure 4.2) of Approach 1

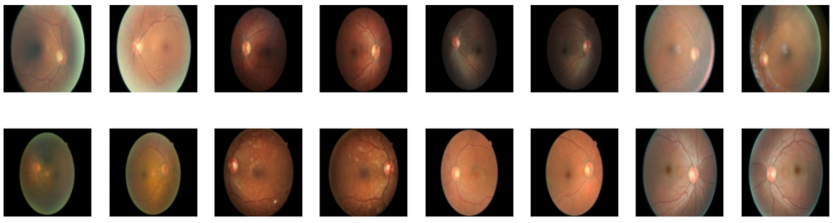


Figure 4.1: Before Preprocessing of Approach 1

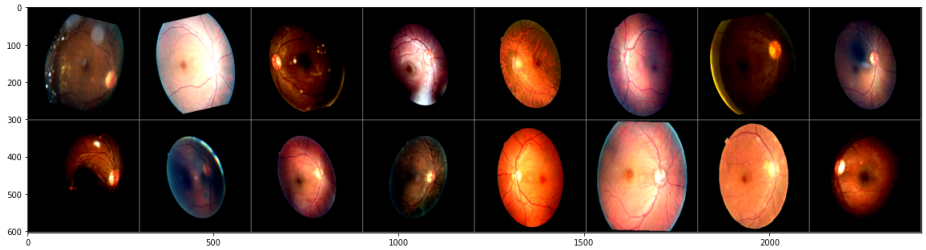


Figure 4.2: After Preprocessing of Approach 1

4.1.2 Output

Here are some images(Figure 4.3 and Figure 4.4) of the model predicting if the eye in the images are healthy or diseased.

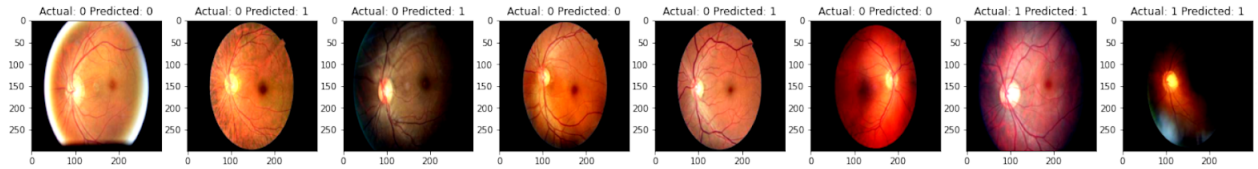


Figure 4.3: Output 1

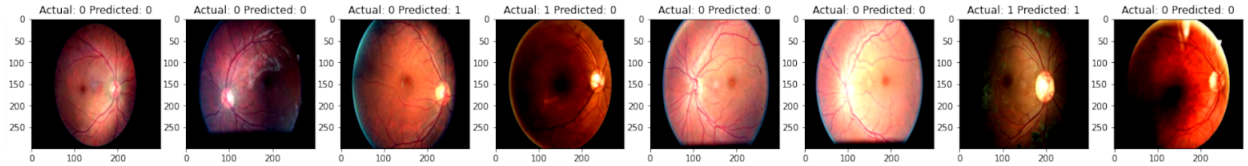


Figure 4.4: Output 2

4.1.3 Prediction Table

No of Data	Actual	Predicted	Result of Prediction	Total Prediction result
1	0	0	true	Here, from 16 images the model predicted 11 images perfectly, and missed 5 images
2	0	0	true	
3	0	1	miss	
4	1	0	miss	
5	0	0	true	
6	0	0	true	
7	1	1	true	
8	0	0	true	
9	0	0	true	
10	0	1	miss	
11	0	1	miss	
12	0	0	true	
13	0	1	miss	
14	0	0	true	
15	1	1	true	
16	1	1	true	

Table 4.1: Output Table of Approach 1

The Table 4.1 shows the how the accuracy is measured.

4.1.4 Result

Network Model	Epochs	Accuracy	Sensitivity	Specificity	Training Mode	Learning Rate
InceptionV3	20	74%	75%	76%	Pre-Train	0.0001
DenseNet121	20	63%	60%	78%	Pre-Train	0.0001
VGG16	20	76%	73%	80%	Pre-Train	0.0001

Table 4.2: Result of Approach 1

After performing the test images through each model of this Approach 1, it turns out as VGG16 is giving the most accuracy. InceptionV3 has higher sensitivity than other models means it has a higher true positive rate. The accuracy of the model InceptionV3, DenseNet121 and VGG16 are 74%, 63%, and 76% respectively. All the models are pre-trained and the learning rate is 0.0001. The sensitivity and specificity of the models are 75% and 76%, 60% and 78%, 73% and 80% respectively. Here in Table 4.2, we can see the model InceptionV3 is more sensible but its specificity is lower than others. VGG16 has higher specificity than the other existing models.

4.1.5 InceptionV3

Graph

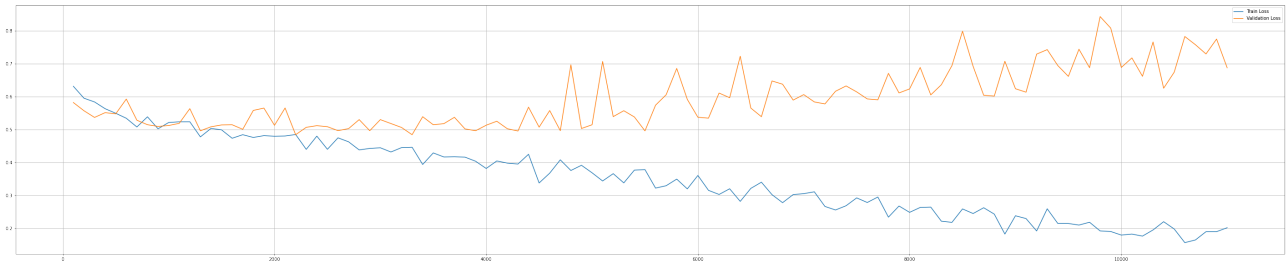


Figure 4.5: Epochs vs loss function for InceptionV3.

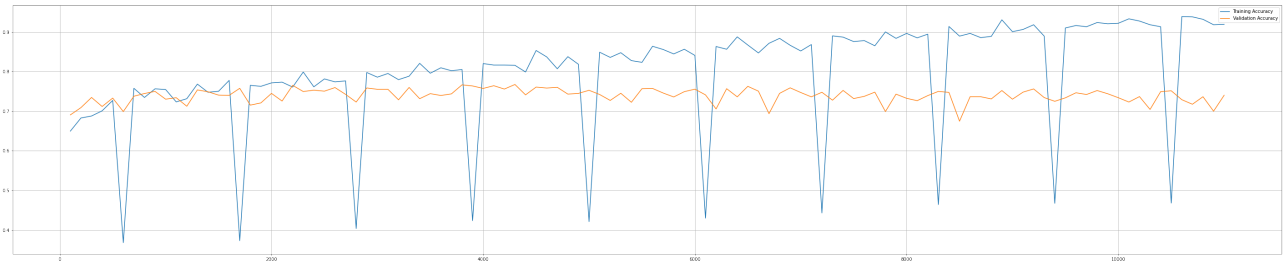


Figure 4.6: Epochs vs accuracy function for InceptionV3.

From Figure 4.5 and Figure 4.6, we can see that the network InceptionV3 is highly noisy and not uniform in the training and test set, although it has a continuous and non-noise graphics. So it can be deduced that it has a low value of sensitivity, and specificity, which can be verified in Table 4.2.

Confusion Matrix

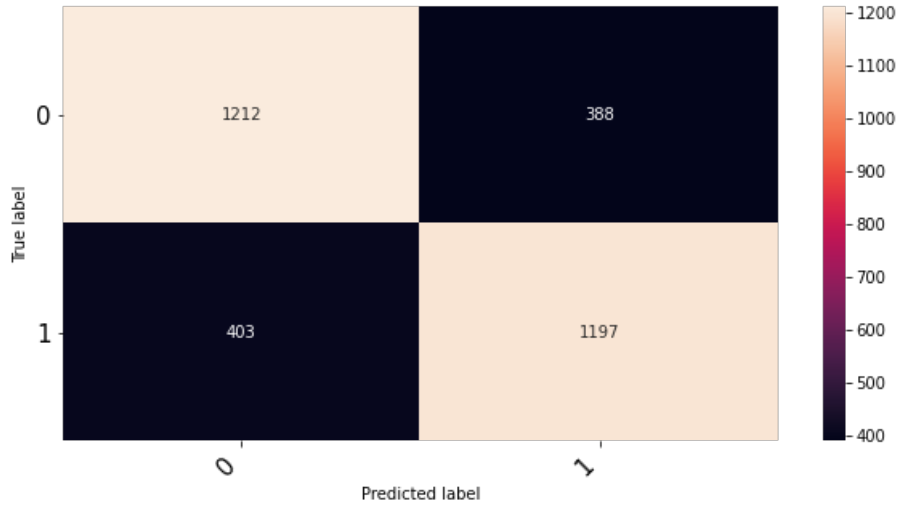


Figure 4.7: Confusion matrix for InceptionV3.

For Class 0: From 1600 test images, this model predicted 1212 images perfectly and missed 388 images.
For Class 1: From 1600 test images, this model predicted 1197 images perfectly and missed 403 images.

4.1.6 DenseNet121

Graph

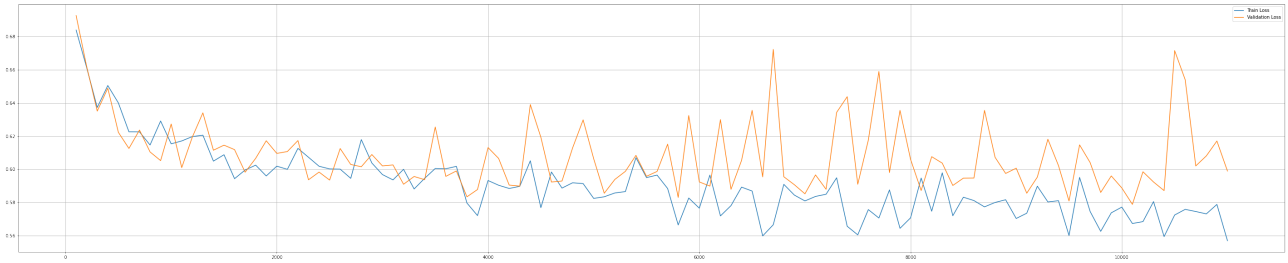


Figure 4.8: Epochs vs loss function for DenseNet121.

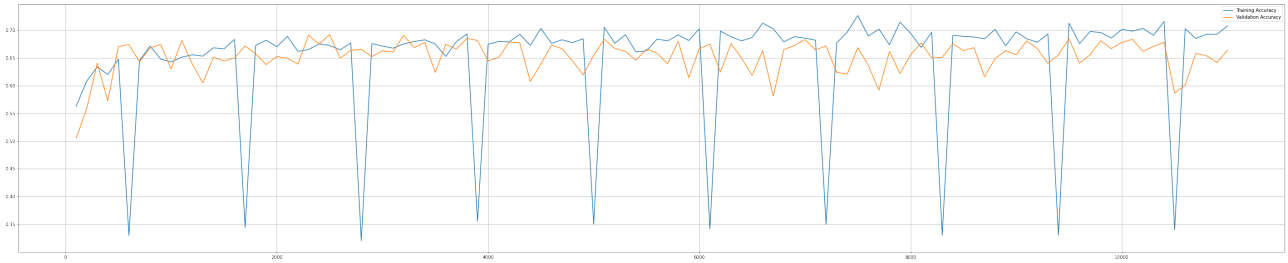


Figure 4.9: Epochs vs accuracy function for DenseNet121.

In Figure 4.8 and Figure 4.9, we can see that the network DenseNet121 has also high noise and is not uniform in the training and test set, it has a continuous and non-noise graphics. So it can be deduced that it has a lower value of sensitivity, and specificity than VGG16, which can be verified in Table 4.2.

Confusion Matrix

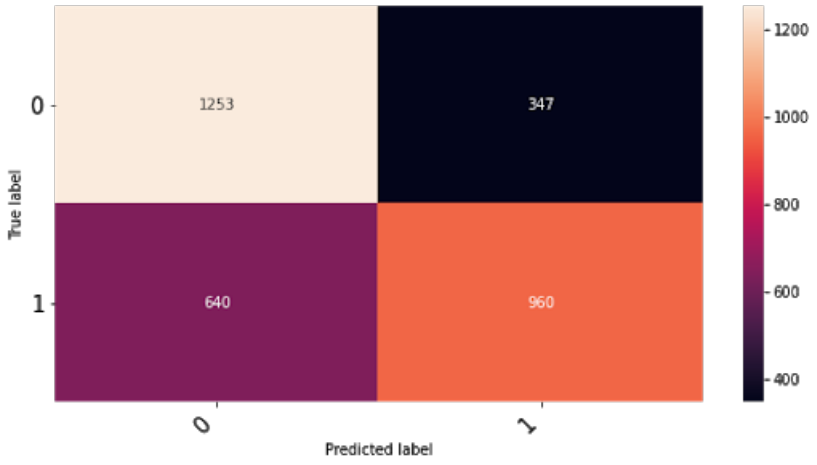


Figure 4.10: Confusion matrix for DenseNet121.

For Class 0: From 1600 test images, this model predicted 1253 images perfectly and missed 347 images.
For Class 1: From 1600 images, this model predicted 960 images perfectly and missed 640 images.

4.1.7 VGG16

Graph

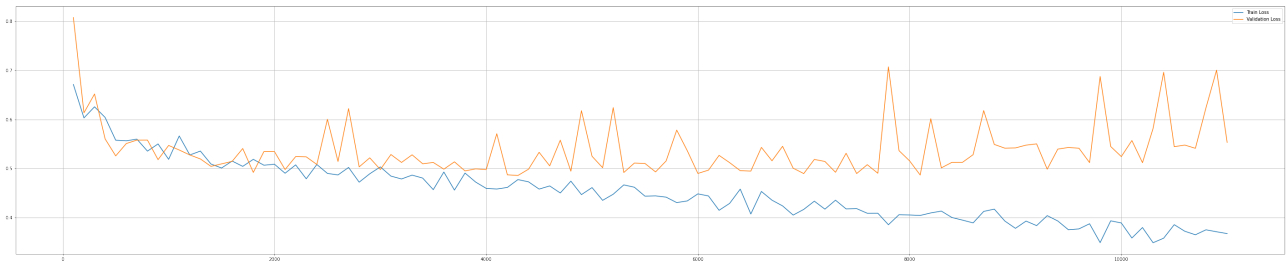


Figure 4.11: Epochs vs loss function for VGG16.

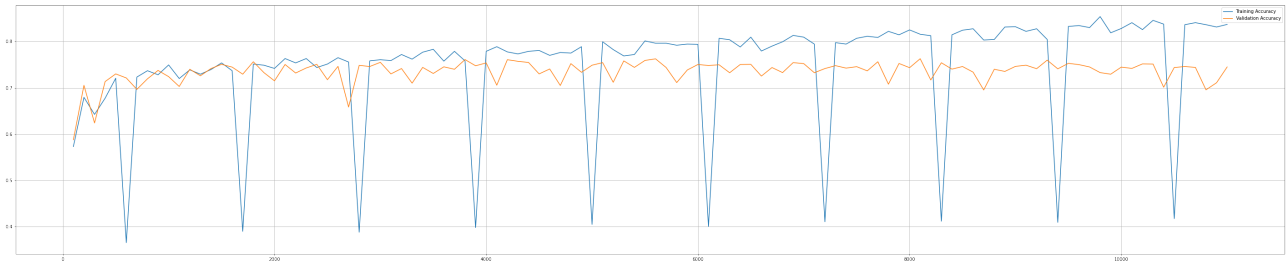


Figure 4.12: Epochs vs accuracy function for VGG16.

From Figure 4.11 and Figure 4.12 we can see that the network VGG16, has a little non-noised and uniform graphics in the training and test set, though it is not fully continuous and non-noise but better than the other models. So it can be deduced that it has a higher value of specificity, but its sensitivity is 2% lower than InceptionV3, which can be verified in Table 4.6.

Confusion Matrix

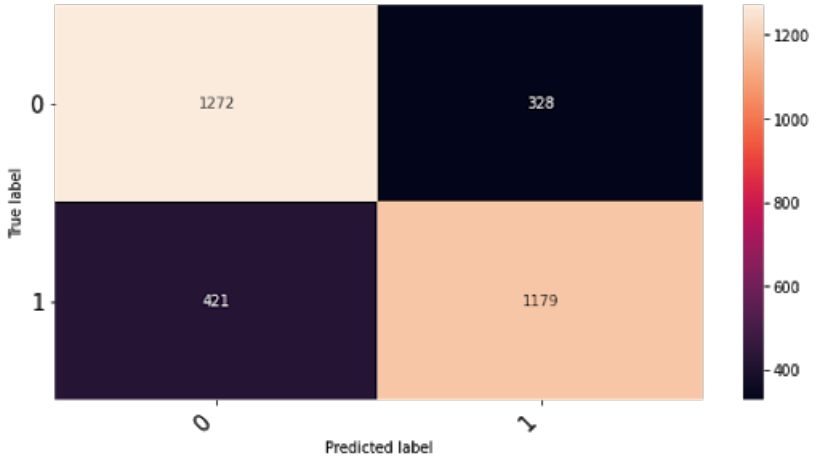


Figure 4.13: Confusion matrix for VGG16.

For Class 0: From 1600 test images, this model predicted 1272 images perfectly and missed 328 images.
For Class 1: From 1600 test images, this model predicted 1179 images perfectly and missed 421 images.

4.1.8 Approach 1 with Different Hyper Parameters

Here we change the hyper parameters such as epoch, batch size and learning rate. The results get improved. The difference in the previous and current model shown in the table 4.3.

Parameters	Previous	Current
Epoch	20	40
Batch Size	16	64
Learning Rate	0.0001	0.001

Table 4.3: Hyper Parameters of Previous and Current

Improved Results

Models	Previous Accuracy	Improved Accuracy
InceptionV3	74%	76%
DenseNet121	63%	65%
VGG16	76%	77%

Table 4.4: Accuracy of Previous and Current Model

After performing the test images through each model of this approach, it turns out all three models get improved as shown in the table 4.4. The accuracy got improved 2%, 2% and 1% in respectively ***InceptionV3***, ***DenseNet121*** and ***Xception***.

4.2 Approach 2

Diabetic Retinopathy Level image Detection using ***InceptionV3***, ***DenseNet121***, ***Xception***, ***ResNet50*** and ***VGG16*** model.

4.2.1 Preprocessing

Here is the images of before preproessing Figure 4.14 and after preprocessing Figure 4.15 of Approach 2.

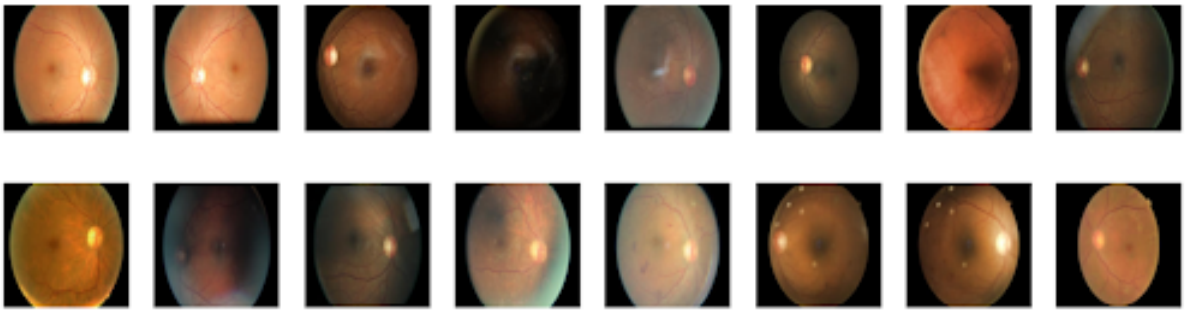


Figure 4.14: Before Preprocessing of Approach 2

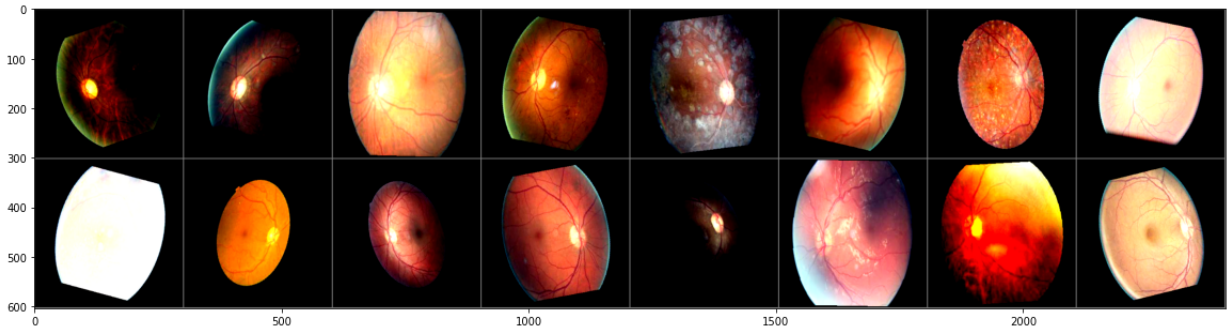


Figure 4.15: After Preprocessing of Approach 2

4.2.2 Output

Here the Figure 4.16 and the Figure 4.17 shows the actual label and the predicted label on the different fundus images.

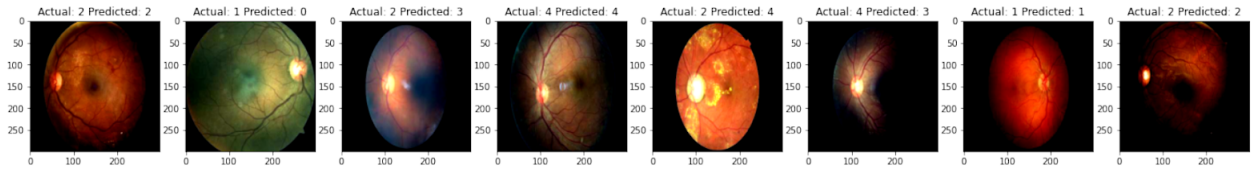


Figure 4.16: Output 1

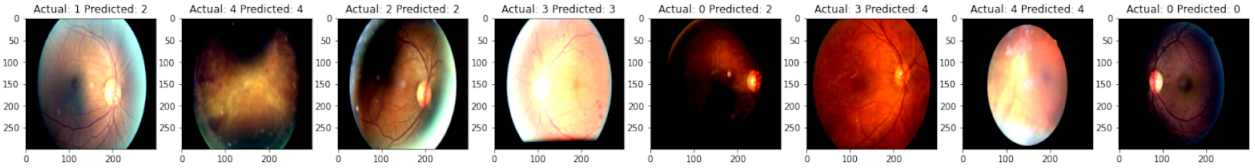


Figure 4.17: Output 2

4.2.3 Prediction Table

No of Data	Actual	Predicted	Result of prediction	Total Prediction Result
1	2	2	true	Here, from 16 images the model predicted 9 images perfectly, and missed 7 images
2	1	0	miss	
3	2	3	miss	
4	4	4	true	
5	2	4	miss	
6	4	3	miss	
7	1	1	true	
8	2	2	true	
9	1	2	miss	
10	4	4	true	
11	2	2	true	
12	3	3	true	
13	0	2	miss	
14	3	4	miss	
15	4	4	true	
16	0	0	true	

Table 4.5: Output Table of Approach 2

The Table 4.5 shows the how the accuracy is measured.

4.2.4 Result

Network Models	Epochs	Accuracy	Training Mode	Learning Rate
Inception V3	20	37%	Pre-Train	0.0001
DenseNet121	20	37%	Pre-Train	0.0001
VGG16	20	20%	Scratch	0.0001
Xception	20	48%	Pre-Train	0.0001
RenseNet50	20	47%	Pre-Train	0.0001

Table 4.6: Result of Approach 2

Here, we can see the result of Approach 2 in the Table 4.6. The accuracy of Xception model is higher than others and VGG16 has a low value of accuracy. All the models are pre-trained except VGG16 which has extremely lower accuracy (20%).

4.2.5 InceptionV3

Graph

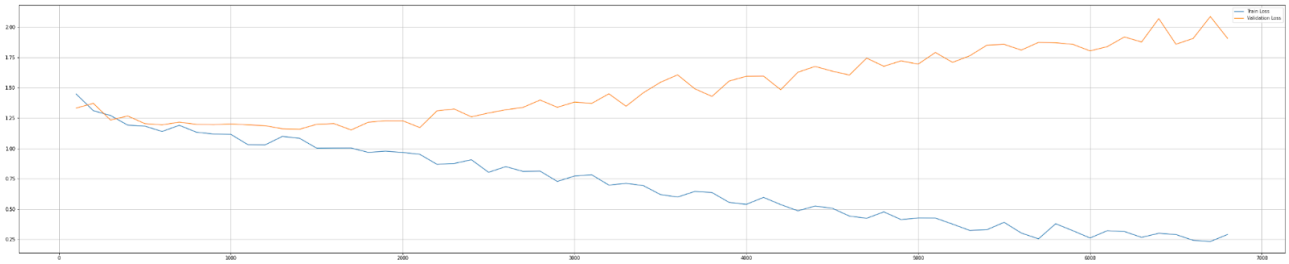


Figure 4.18: Epochs vs loss function for InceptionV3.

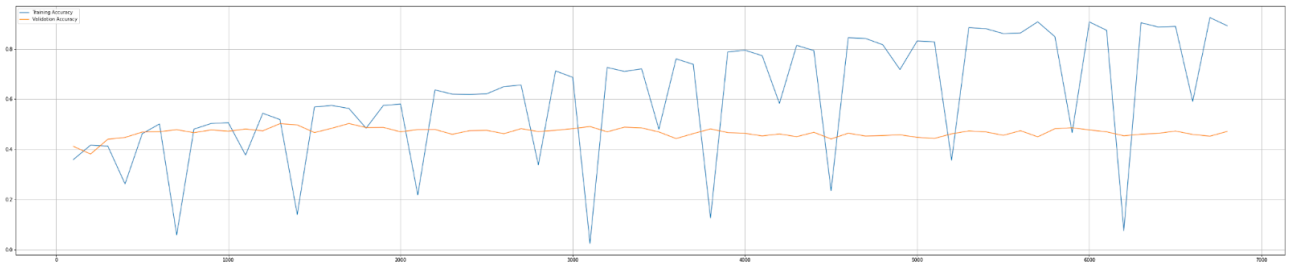


Figure 4.19: Epochs vs accuracy function for InceptionV3.

Here in Figure 4.18, after 2000 steps training loss is decreasing but validation loss is increasing. In Figure 4.19, after 2000 steps training accuracy is increasing, but validation accuracy is decreasing. The model tried to learn more from training data but getting poorer models for unknown data.

Confusion Matrix

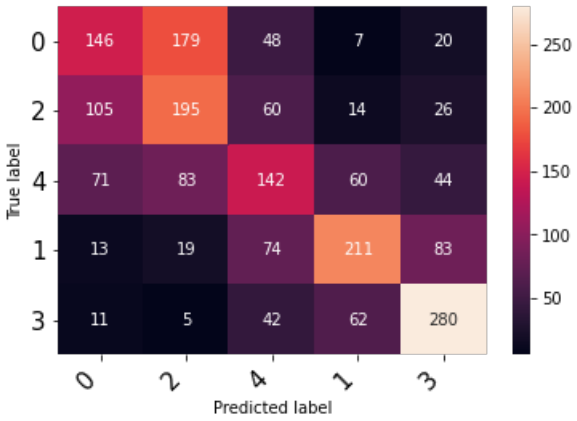


Figure 4.20: Confusion Matrix of InceptionV3.

This model predicted the label 0 images as 0 - 146 times and missed 254 times, the label 1 images as 1 - 211 times and missed 189 times, the label 2 images as 2 - 195 times and missed 205 times, the label 3 images as 3 - 280 times and missed 120 times, the label 4 images as 4 - 142 times and missed 258 times.

4.2.6 DenseNet121

Graph

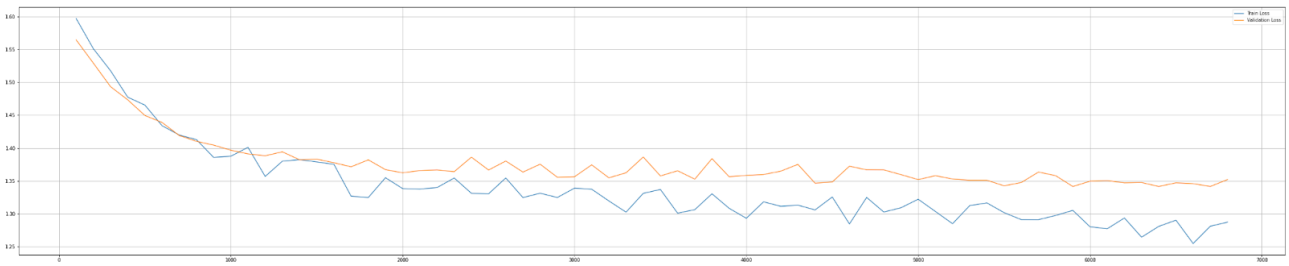


Figure 4.21: Epochs vs loss function for DenseNet121.

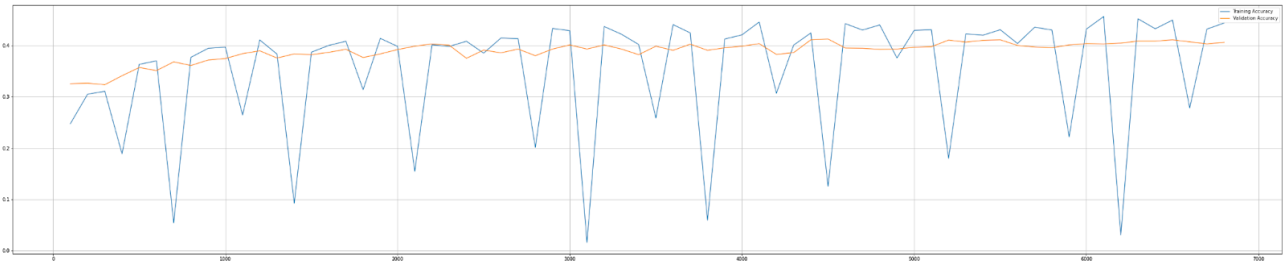


Figure 4.22: Epochs vs accuracy function for DenseNet121.

Here in Figure 4.21, both Training and Validation loss are decreasing. In Figure 4.22, both Training and Validation accuracy are increasing.

Confusion Matrix

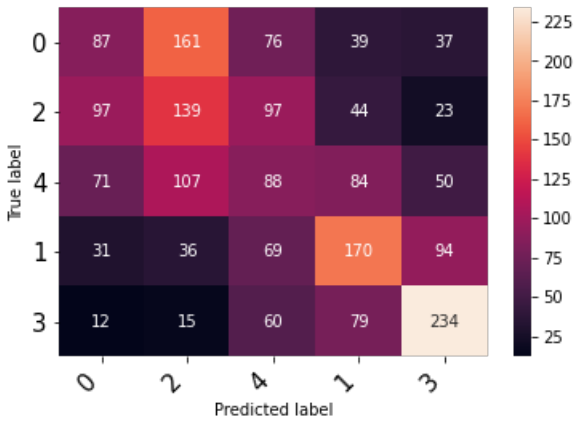


Figure 4.23: Confusion Matrix of DenseNet121.

This model predicted the label 0 images as 0 - 87 times and missed 313 times, the label 1 images as 1 - 170 times and missed 230 times, the label 2 images as 2 - 139 times and missed 261 times, the label 3 images as 3 - 234 times and missed 166 times, the label 4 images as 4 - 88 times and missed 312 times.

4.2.7 Xception

Graph

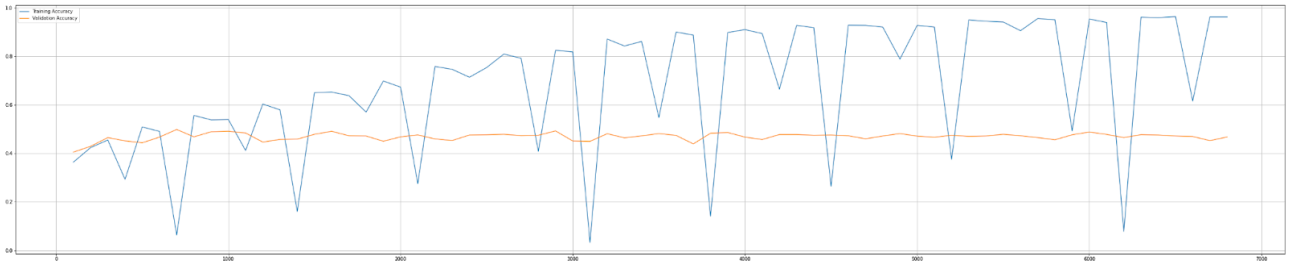


Figure 4.24: Epochs vs accuracy function for Xception.

Here in Figure 4.24, after 3000 steps training accuracy is increasing but validation accuracy is decreasing. The model tried to learn more from training data but getting poorer models for unknown data.

Confusion Matrix

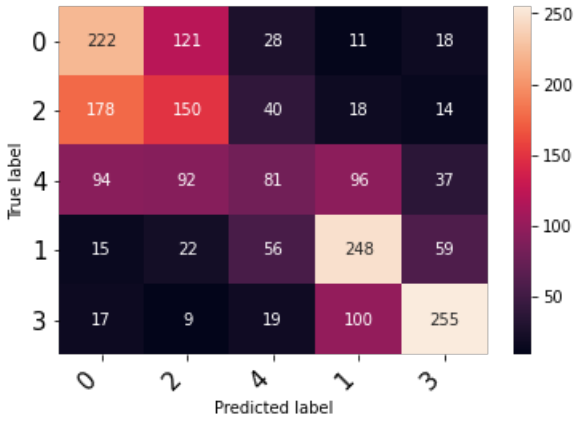


Figure 4.25: Confusion Matrix of Xception

This model predicted the label 0 images as 0 - 222 times and missed 178 times, the label 1 images as 1 - 248 times and missed 152 times, the label 2 images as 2 - 150 times and missed 250 times, the label 3 images as 3 - 255 times and missed 145 times, the label 4 images as 4 - 81 times and missed 319 times.

4.2.8 ResNet50

Graph

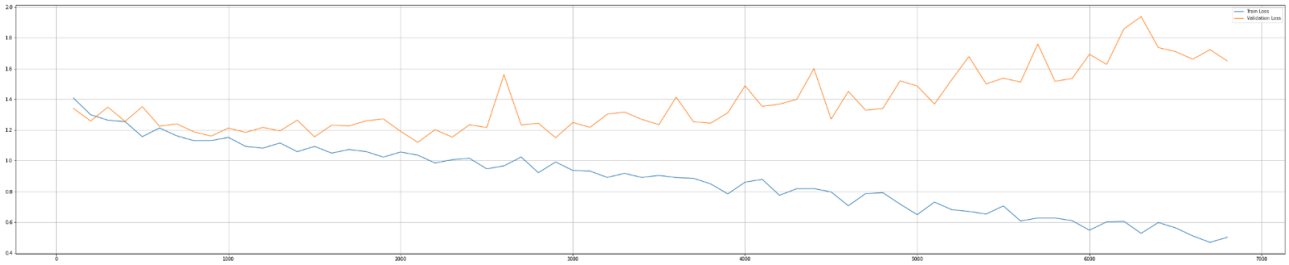


Figure 4.26: Epochs vs loss function for ResNet50.

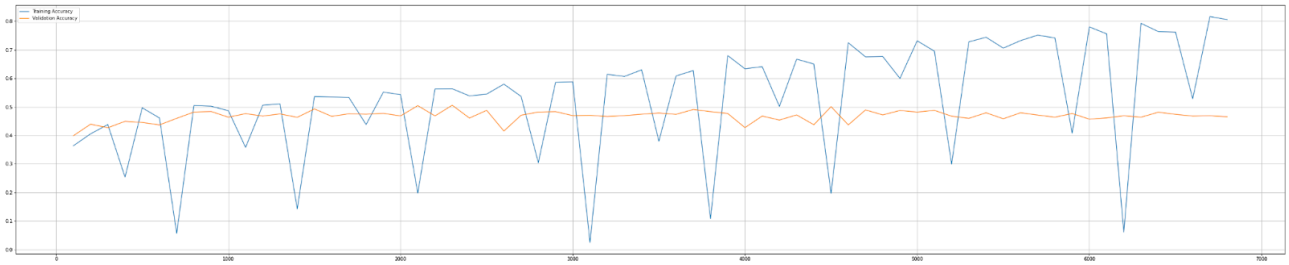


Figure 4.27: Epochs vs accuracy function for ResNet50.

Here in Figure 4.27, after 3000 steps training accuracy is increasing but validation accuracy is decreasing. The model tried to learn more from training data but getting poorer models for unknown data.

Confusion Matrix

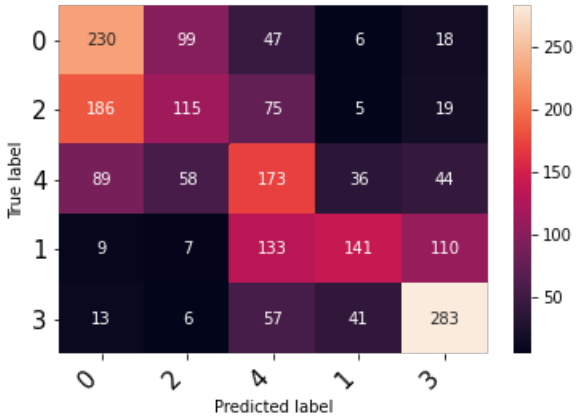


Figure 4.28: Confusion Matrix of ResNet50.

This model predicted the label 0 images as 0 - 230 times and missed 170 times, the label 1 images as 1 - 141 times and missed 259 times, the label 2 images as 2 - 115 times and missed 285 times, the label 3 images as 3 - 283 times and missed 117 times, the label 4 images as 4 - 173 times and missed 227 times.

4.2.9 VGG16

Graph

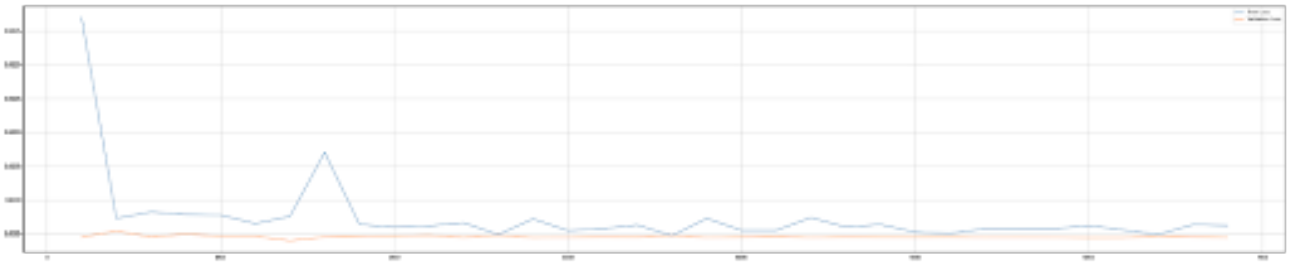


Figure 4.29: Epochs vs loss function for VGG16.

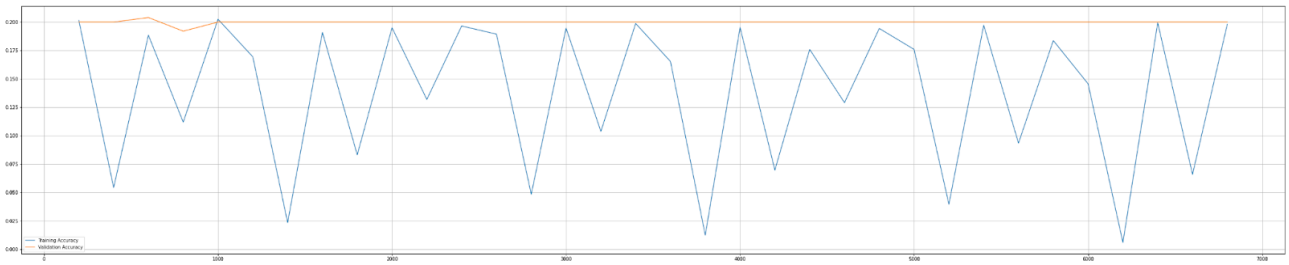


Figure 4.30: Epochs vs accuracy function for VGG16.

From Figure 4.29 and Figure 4.30, we can see in the graph, the test set is non-noised but the training set is highly noised. VGG16 model has extremely lower accuracy (20%) than others which is verified in Table 4.6. As this model isn't learning much, it's behaving like a poor model for all known and unknown data.

4.2.10 Approach 2 with APTOS Dataset

In this approach we used a dataset from Kaggle Competition APTOS 2019 Blindness Detection [34]. This dataset consists of total 3662 retinal images of 5 classes. As the dataset is imbalanced, the model will not have data to learn the pattern of minority classes. So, we calculated weights for each classes.

$$w_j = (n\_samples)/(n\_classes * n\_samples_j)$$

Class	Number of Images	Class Weight
0	1805	0.4058
1	370	1.9795
2	999	0.7331
3	193	3.7948
4	295	2.4827

Table 4.7: Number of Images and Class Weight For Each Class

We applied the four models **InceptionV3**, **Resnet50**, **VGG16** and **DenseNet121** on the new dataset. The results on this dataset is better than the previous dataset. The models in this approach was all pre-trained models.

Models	Accuracy on Previous Dataset	Accuracy on APTOS Dataset
InceptionV3	37%	73%
Resnet50	47%	84%
VGG16	20%	71%
DenseNet121	37%	81%

Table 4.8: Accuracy on the Previous and APTOS Dataset

Models	Performance Matrices	Class 0	Class 1	Class 2	Class 3	Class 4
InceptionV3	Precision	0.97	0.38	0.71	0.32	0.44
	Recall	0.85	0.49	0.71	0.53	0.46
	f1-Score	0.91	0.43	0.71	0.40	0.45
Resnet50	Precision	0.97	0.60	0.77	0.80	0.52
	Recall	0.99	0.63	0.83	0.24	0.50
	f1-Score	0.98	0.62	0.80	0.36	0.51
VGG16	Precision	0.96	0.37	0.77	0.38	0.24
	Recall	0.88	0.71	0.57	0.29	0.33
	f1-Score	0.92	0.49	0.65	0.33	0.28
DenseNet121	Precision	0.96	0.54	0.72	0.60	0.53
	Recall	0.98	0.54	0.81	0.35	0.33
	f1-Score	0.97	0.54	0.76	0.44	0.41

Table 4.9: Performance Matrices of the Models on APTOS Dataset

In the table 4.9, we can see the performance matrices of 5 classes in 4 models. We can see the *Class 0* is has the best precision, recall and f1-score in all the models. The reason is the number of images of Class 0 in the dataset is more than the others. So, the model had much more data to train and detecting it more precisely.

4.3 Approach 3

Diabetic Retinopathy Level image Detection using ***Ensemble*** Learning.

4.3.1 Preprocessing

Here is the images of before preproessing (Figure 4.31) and after preprocessing (Figure 4.32) of Approach 3 (same as Approach 2).

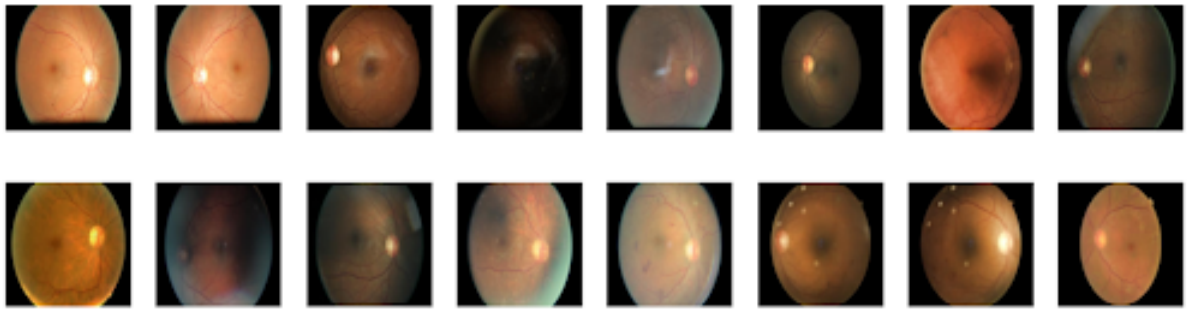


Figure 4.31: Before Preprocessing of Approach 3

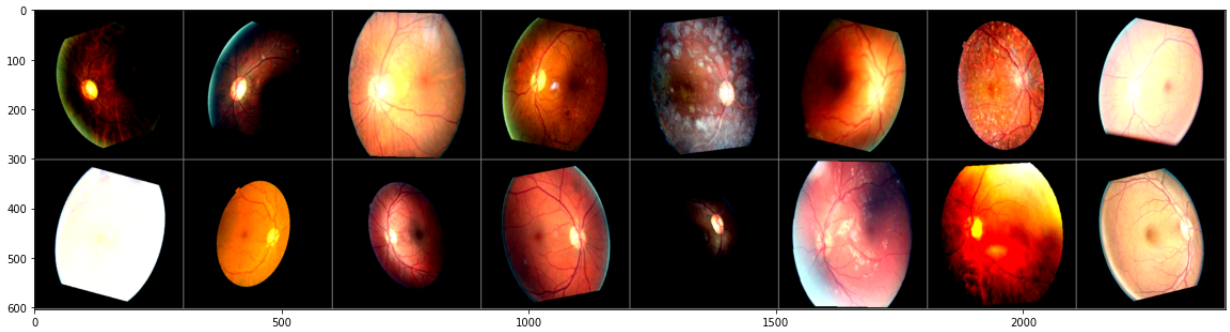


Figure 4.32: After Preprocessing of Approach 3

4.3.2 Output

Here the Figure 4.33 and Figure 4.34 shows the actual label and the predicted label on the different fundus images.

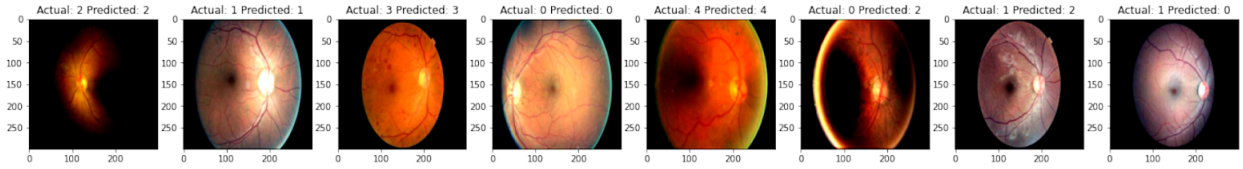


Figure 4.33: Output 1

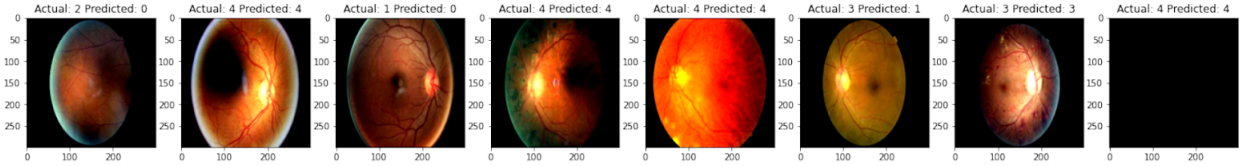


Figure 4.34: Output 2

4.3.3 Prediction Table

No of Data	Actual	Predicted	Result of prediction	Total Prediction Result
1	2	2	true	Here, from 16 images the model predicted 10 images perfectly, and missed 6 images
2	1	1	true	
3	3	3	true	
4	0	0	true	
5	4	4	true	
6	0	2	miss	
7	1	2	miss	
8	1	0	miss	
9	2	0	miss	
10	4	4	true	
11	1	0	miss	
12	4	4	true	
13	4	4	true	
14	3	1	miss	
15	3	3	true	
16	4	4	true	

Table 4.10: Output Table of Approach 3

The Table 4.10 shows the how the accuracy is measured.

4.3.4 Result

Network Model	Epochs		Accuracy	Training Mode	Learning Rate
InceptionV3	20	Ensemble	52%	Pre-Train	0.0001
Xception	20			Pre-Train	0.0001
DenseNet121	20			Pre-Train	0.0001
ResNet50	20			Pre-Train	0.0001

Table 4.11: Result of Approach 3

Here we can see in the Table 4.11, using the four pre-trained model *InceptionV3*, *Xception*, *DenseNet121* and *ResNet50* in ensemble method the accuracy is 52% in detecting different levels of images.

4.3.5 Ensemble Model

Confusion Matrix

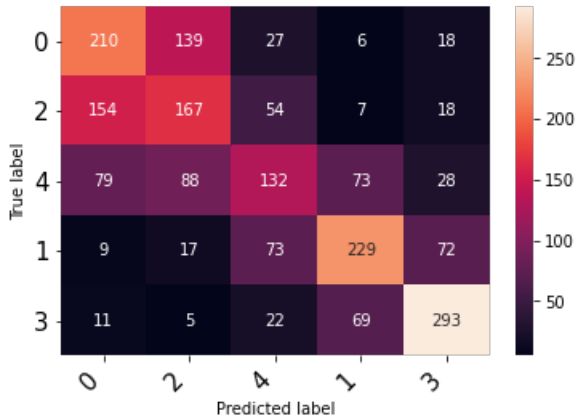
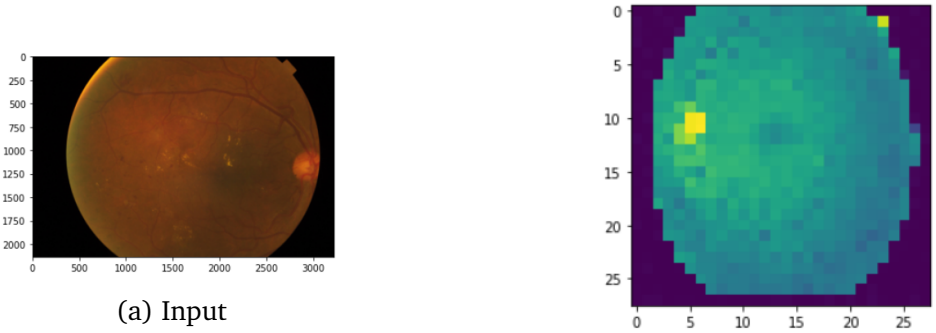


Figure 4.35: Confusion Matrix of Ensemble Model.

This model predicted the label 0 images as 0 - 210 times and missed 190 times, the label 1 images as 1 - 229 times and missed 171 times, the label 2 images as 2 - 167 times and missed 233 times, the label 3 images as 3 - 293 times and missed 107 times, the label 4 images as 4 - 132 times and missed 268 times.

4.4 Approach 4

First we change the image data into *csv* file as it will help us to process the data faster in the models. After converting the images , it will look like fig. 4.36a.



(a) Input

(b) Output

Figure 4.36: Converting Input Images to 28X28 and CSV

After applying the clustering algorithm on the dataset based on the similarities. The distribution of images are kind of like the table 4.12.

Cluster	Total Number of Images	Class 0	Class 1	Class 2	Class 3	Class 4
0	953	330	111	336	70	106
1	531	498	6	3	20	4
2	699	95	203	247	66	88
3	943	877	19	37	1	9
4	536	5	31	376	36	88

Table 4.12: The Number of Images in 5 Clusters

4.4.1 Performance Matrix

After that we applied our classification model on each cluster of images based on dissimilarities. The performance matrices of each models are given in table 4.13, table 4.14, table 4.15, table 4.16 and table 4.17.

Cluster 0				
Class	Precision	Recall	f1-Score	Accuracy
0	0.84	0.41	0.55	26%
1	0.22	0.56	0.32	
2	0.00	0.00	0.00	
3	0.07	0.33	0.11	
4	0.14	0.44	0.22	

Table 4.13: Performance Matrices of Cluster 0

Cluster 1				
Class	Precision	Recall	f1-Score	Accuracy
0	1.00	1.00	1.00	96%
1	0.00	0.00	0.00	
2	0.00	0.00	0.00	
3	0.57	1.00	0.73	
4	0.00	0.00	0.00	

Table 4.14: Performance Matrices of Cluster 1

Cluster 2				
Class	Precision	Recall	f1-Score	Accuracy
0	0.52	0.75	0.62	42%
1	0.37	0.93	0.53	
2	0.75	0.12	0.21	
3	0.00	0.00	0.00	
4	0.00	0.00	0.00	

Table 4.15: Performance Matrices of Cluster 2

Cluster 3				
Class	Precision	Recall	f1-Score	Accuracy
0	0.93	1.00	0.96	92%
1	0.00	0.00	0.00	
2	0.00	0.00	0.00	
3	0.00	0.00	0.00	
4	0.00	0.00	0.00	

Table 4.16: Performance Matrices of Cluster 3

Cluster 4				
Class	Precision	Recall	f1-Score	Accuracy
0	1.00	1.00	1.00	74%
1	0.00	0.00	0.00	
2	0.74	1.00	0.85	
3	0.00	0.00	0.00	
4	0.00	0.00	0.00	

Table 4.17: Performance Matrices of Cluster 4

4.4.2 Confusion Matrix

The confusion matrices of each models are given in fig. 4.37, fig. 4.38, fig. 4.39, fig. 4.40 and fig. 4.41.

True Positive (TP): Predicted positive and the result is true.We predicted that the patient has diabetic retinopathy(DR), and he has DR.
True Negative (TN): Predicted negative and the result is true. We predicted that patient has no DR, and he has no DR.
False Positive (FP): Predicted positive but the result is not true. We predicted that patient has DR but, he has no DR.
False Negative (FN): Predicted negative but the result is true. We predicted that the patient has no DR but, he has DR.

In medical science, false negative is important that means if patient has disease and model can not predict it, then it will be harmful for them. So, the rate of false negative should be lower.
On the contrary, false positive may not be harmful for patient as he has no disease but diagnose as diseased and further diagnosis can solve this condition.

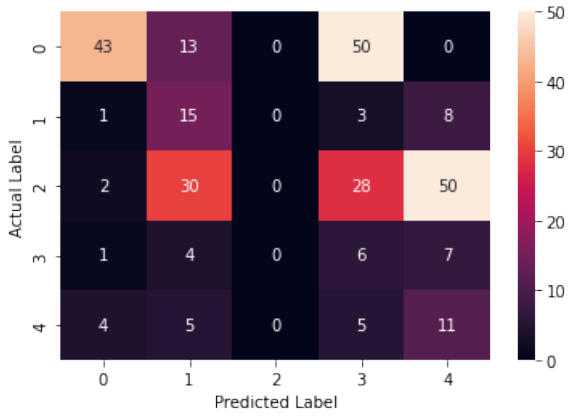


Figure 4.37: Confusion Matrix for Cluster 0

For Class 0, the model predicted 43 images as class 0 out of 106 images. Here, 13 images class 1 and 50 images class 3 has been predicted wrongly.

For Class 1, the model predicted 15 images as class 1 out of 27 images. Here, 1 images as class 0, 3 images as class 3 and 8 images as class 4 has been predicted wrongly.

For Class 2, the model predicted 0 images as class 2 out of 110 images. Here, 2 images as class 0, 30 images as class 1, 28 images as class 3 and 50 images as class 4 has been predicted wrongly.

For Class 3, the model predicted 6 images as class 3 out of 18 images. Here, 1 images as class 0, 4 images as class 1 and 7 images as class 4 has been predicted wrongly.

For Class 4, the model predicted 11 images as class 4 out of 25 images. Here, 4 images as class 0, 5 images as class 1 and 5 images as class 3 has been predicted wrongly.

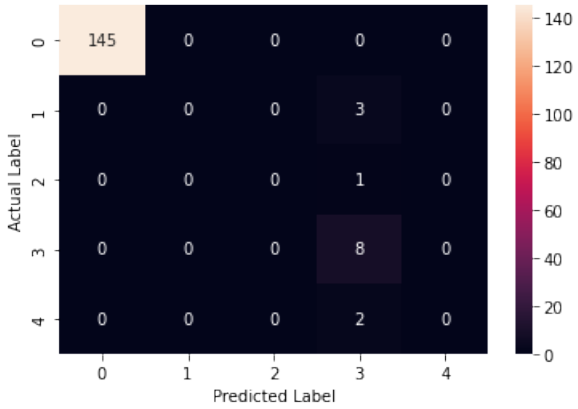


Figure 4.38: Confusion Matrix for Cluster 1

For Class 0, the model predicted 145 images as class 0 out of 145 images.

For Class 1, the model predicted 0 images as class 1 out of 3 images. Here, 3 images as class 3 has been predicted wrongly.

For Class 2, the model predicted 0 images as class 2 out of 1 images. Here, 1 images as class 3 has been predicted wrongly.

For Class 3, the model predicted 8 images as class 3 out of 8 images.

For Class 4, the model predicted 0 images as class 4 out of 2 images. Here, 2 images as class 3 has been predicted wrongly.

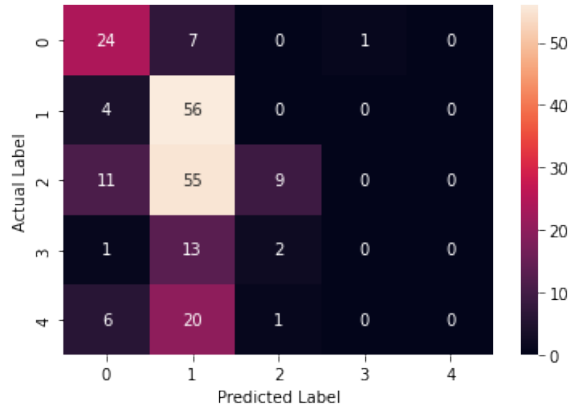


Figure 4.39: Confusion Matrix for Cluster 2

For Class 0, the model predicted 24 images as class 0 out of 32 images. Here, 7 images class 1 and 1 images class 3 has been predicted wrongly.

For Class 1, the model predicted 56 images as class 1 out of 60 images. Here, 4 images as class 0 has been predicted wrongly.

For Class 2, the model predicted 9 images as class 2 out of 75 images. Here, 55 images as class 1 and 11 images as class 0 has been predicted wrongly.

For Class 3, the model predicted 0 images as class 3 out of 16 images. Here, 1 images as class 0, 13 images as class 1 and 2 images as class 2 has been predicted wrongly.

For Class 4, the model predicted 0 images as class 4 out of 27 images. Here, 6 images as class 0, 20 images as class 1 and 1 images as class 2 has been predicted wrongly.

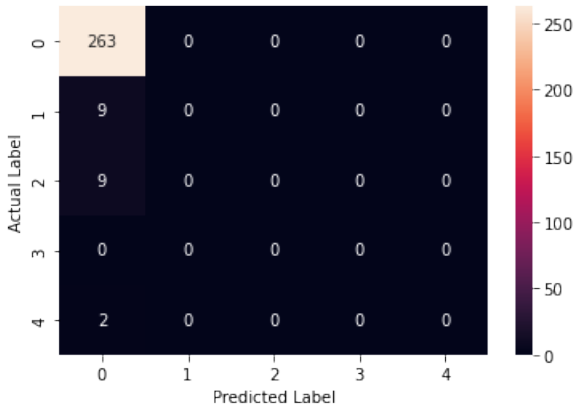


Figure 4.40: Confusion Matrix for Cluster 3

For Class 0, the model predicted 263 images as class 0 out of 263 images.

For Class 1, the model predicted 0 images as class 1 out of 9 images. Here, 9 images as class 0 has been predicted wrongly.

For Class 2, the model predicted 0 images as class 2 out of 9 images. Here, 9 images as class 0 has been predicted wrongly.

For Class 4, the model predicted 0 images as class 4 out of 2 images. Here, 2 images as class

0 has been predicted wrongly.

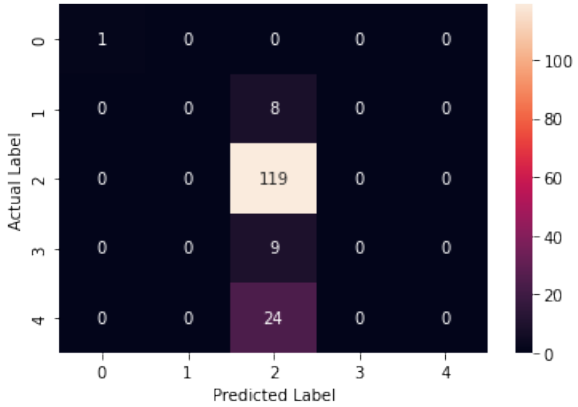


Figure 4.41: Confusion Matrix for Cluster 4

For Class 0, the model predicted 1 images as class 0 out of 1 images.

For Class 1, the model predicted 0 images as class 1 out of 8 images. Here, 8 images as class 2 has been predicted wrongly.

For Class 2, the model predicted 119 images as class 2 out of 119 images.

For Class 3, the model predicted 0 images as class 3 out of 9 images. Here, 9 images as class 2 has been predicted wrongly.

For Class 4, the model predicted 0 images as class 4 out of 24 images. Here, 24 images as class 2 has been predicted wrongly.

4.5 Comparison Among the Results of the Approaches

Approaches	Accuracy
Approach 1 (Old)	76%
Approach 1 (New)	77%
Approach 2 (Old Dataset)	48%
Approach 2 (New Dataset)	84%
Approach 3	52%
Approach 4	65%

Table 4.18: Comparison Among the Approaches

Here from Table 4.18, we can see that, in Approach 1 Old version the model *VGG16* gave the best accuracy (76%) among all the approach models. Again in Approach 1 New version the model *VGG16* gave the best accuracy (77%) among all the approach models. Then *Resnet50* model has given the best accuracy (48%) among the models of Approach 2 on EyePacs Dataset. Also *Resnet50* model has given the best accuracy (84%) among the models of Approach 2 on Aptos Dataset. In Approach 3 *Ensemble* model, which is a combination of *InceptionV3*, *DenseNet121*, *ResNet50*, and *Xception* has given 52% accuracy. In Approach 4 our custom made Linear model gives average of (65%) accuracy for multiclass classification.

Chapter 5

Conclusion

In this research we apply four approaches to detect diabetic retinopathy . In the first approach we detect binary classification of Diabetic Retinopathy. We only detect if the image is healthy or diseased. In that approach we use *InceptionV3*, *DensenetNet121* and *VGG16* architectures. In this approach among the models, *VGG16* perform better than any other models.
In the second approach we detect Multi classification of Diabetic Retinopathy. In this approach we use *InceptionV3*, *DensenetNet121* and *VGG16*, *Xception*, *ResNet50* architectures. In this approach among the models *Xception* performs better than any other models.
In the third approach we use ensemble method in which we use *InceptionV3, DensenetNet121*, *Xception* and *ResNet50* architectures. Ensemble method is the technique that creates multiple models and then combines them to produce improved results. In Approach 3, the accuracy increases comparing to Approach 2, where we use single model to detect Diabetic Retinopathy.
Our fourth approach is bit different. We applied clustering algorithm K-means, on a new dataset to make clusters based on similarities. Then we use our classification model to classify the classes based on dissimilarities in those clusters. As we use clusters, some clusters had high accuracy when others had low accuracy.

We had to face some challenges during this research. Some of them are given below :

- We are going through a pandemic situation of Covid-19.In this situation it was a big challenge to continue this research.
- Among the new environments we had to get acquainted with the *Pytorch*, which has limited resources.
- We had to balance the datasets based on healthy and diseased classes, which was imbalanced. In the new dataset number of images was low. So our model did not get enough data to train.

- For training our deep learning models, we need faster GPU, but none of our group teammates have GPU. So, we have to use colab GPU, which is cloud based. But colab allow to use GPU only for 12 hours which is not quite sufficient for training a huge dataset.

Our future work is to improve the models or approaches we mentioned above. We will try to get better accuracy. This would make it easier to diagnose *Diabetic Retinopathy* more accurately.

References

[1] "Activation Function description." https://www.mygreatlearning.com/blog/activation-functions/. Accessed: 2021-06-25.

[2] "Forward and Backward Propagation." https://www.kaggle.com/questions-and-answers/96566. Accessed: 2021-06-26.

[3] "Gradient Descent." https://machinelearningmastery.com/gradient-descent-for-machine-learning/. Accessed: 2021-06-26.

[4] "Stochastic Descent Description." https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1. Accessed: 2021-06-26.

[5] "Adam Description." https://ruder.io/optimizing-gradient-descent/index.html#fn14. Accessed: 2021-06-26.

[6] "Details of Confution Matrix." https://glassboxmedicine.com/2019/02/17/measuring-performance-the-confusion-matrix/. Accessed: 2021-06-26.

[7] " Pytorch ." https://hub.packtpub.com/what-is-pytorch-and-how-does-it
Accessed: 2021-06-26.

[8] "Python." https://www.futurelearn.com/info/blog/what-is-python-used-for. Accessed: 2021-06-26.

[9] "Clustering Algorithm." https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and
Accessed: 2021-12-16.

[10] G. García, J. Gallardo, A. Mauricio, J. López, and C. Del Carpio, "Detection of diabetic retinopathy based on a convolutional neural network using retinal fundus images," in *International Conference on Artificial Neural Networks*, pp. 635–642, Springer, 2017.

[11] S. Mishra, S. Hanchate, and Z. Saquib, "Diabetic retinopathy detection using deep learning," in *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, pp. 515–520, IEEE, 2020.

[12] S. Qummar, F. G. Khan, S. Shah, A. Khan, S. Shamshirband, Z. U. Rehman, I. A. Khan, and W. Jadoon, "A deep learning ensemble approach for diabetic retinopathy detection," *IEEE Access*, vol. 7, pp. 150530–150539, 2019.

[13] "Details about Sensitivity and specifity ." https://en.wikipedia.org/wiki/Sensitivity_and_specificity. Accessed: 2021-06-26.

[14] " Pattern Matching Figure." https://www.opto-e.com/basics/template-matching. Accessed: 2021-06-26.

[15] "Common steps for Data Augmentation Figure." https://www.mygreatlearning.com/blog/understanding-data-augmentation/. Accessed: 2021-06-26.

[16] " Figure Human Brain Neural System ." https://www.javatpoint.com/artificial-neural-network. Accessed: 2021-06-26.

[17] " Figure Artificial Neural Network ." https://www.javatpoint.com/artificial-neural-network. Accessed: 2021-06-26.

[18] " Figure Calculation inside a convolutional layer in CNN ." https://heartbeat.fritz.ai/exploring-convolutional-neural-networks-cnns-from-an-ios-develop Accessed: 2021-06-26.

[19] " Figure Convolutional Neural Network ." https://technoelearn.com/convolutional-neural-network-tutorial/. Accessed: 2021-06-26.

[20] "Simple Neural Networks Model Without Any Hidden Layer." https://www.mygreatlearning.com/blog/activation-functions/. Accessed: 2021-06-25.

[21] "Binary Step Function Graph." https://www.mygreatlearning.com/blog/activation-functions/. Accessed: 2021-06-25.

[22] "Linear Activation Function." https://www.mygreatlearning.com/blog/activation-functions/. Accessed: 2021-06-25.

[23] "Sigmoid Activation Function Graph." https://www.mygreatlearning.com/blog/activation-functions/. Accessed: 2021-06-25.

[24] "Vanishing Gradients." https://www.mygreatlearning.com/blog/activation-functions/. Accessed: 2021-06-25.

[25] " Figure of Forward and Backward Propagation ." https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide. Accessed: 2021-06-26.

[26] "Gradient Descent Figure." https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3. Accessed: 2021-06-26.

[27] "Learning Rate Figure." https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0. Accessed: 2021-06-26.

[28] "Stochastic Descent Figure." https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0. Accessed: 2021-06-26.

[29] "Procedure of Densenet 121." https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a. Accessed: 2021-06-26.

[30] "Procedure of VGG 16 ." https://link.springer.com/article/10.1007/s42979-020-0114-9/figures/4. Accessed: 2021-06-26.

[31] "Architecture of Inception Layer." https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/. Accessed: 2021-06-26.

[32] "Procedure of Xception Architecture ." https://towardsdatascience.com/xception-from-scratch-using-tensorflow-even-better-than-inc
Accessed: 2021-06-26.

[33] " Procedure of Resnet50 architecture ." https://www.researchgate.net/figure/ResNet-50-architecture-26-shown-with-the-residual-units-
fig1_338603223. Accessed: 2021-06-26.

[34] "Aptos 2019 Blindness Detection Dataset." https://www.kaggle.com/c/aptos2019-blindness-detection/data. Accessed: 2021-12-16.

[35] S. Jan, I. Ahmad, S. Karim, Z. Hussain, M. Rehman, and M. A. Shah, "Status of diabetic retinopathy and its presentation patterns in diabetics at ophthalomogy clinics," *Journal of Postgraduate Medical Institute (Peshawar-Pakistan)*, vol. 32, no. 1, 2018.

[36] W. R. Memon, B. Lal, and A. A. Sahto, "Diabetic retinopathy," *The Professional Medical Journal*, vol. 24, no. 02, pp. 234–238, 2017.

[37] "Pytorch Reasons." https://www.kdnuggets.com/2018/11/introduction-pytorch-deep-learning.html. Accessed: 2021-06-26.

[38] "Reason to use Python." https://www.tutorialspoint.com/python/index.htm. Accessed: 2021-06-26.

[39] "Work Flow of K-Means Algorithm." https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning. Accessed: 2021-12-16.

[40] "Aptos Dataset ." https://www.kaggle.com/c/diabetic-retinopathy-detection/data. Accessed: 2021-06-26.

[41] "Details about Imagenet." https://www.image-net.org/. Accessed: 2021-06-26.

[42] "VGG 16 Architecture." https://www.geeksforgeeks.org/vgg-16-cnn-model/a. Accessed: 2021-06-26.

[43] "Inception architecture ." https://www.geeksforgeeks.org/inception-v2-and-v3-inception-network-versions/?fbclid=IwAR16S3ROLLcp0we_FHq8lztpjMeiHxAgPl5EHRLahxGTXp0_jKzvwU0cZqA. Accessed: 2021-06-26.

[44] "Xception architecture." https://towardsdatascience.com/xception-from-scratch-using-tensorflow-even-better-than-inceptio Accessed: 2021-06-26.

[45] "Resnet50 architecture ." https://iq.opengenus.org/resnet50-architecture/. Accessed: 2021-06-26.

[46] "Ensemble method." https://www.toptal.com/machine-learning/ensemble-methods-machine-learning. Accessed: 2021-06-26.