MINOR PROJECT

Final Evaluation Report

Detection Of Blindness Due To Diabetic Retinopathy Detection Using Deep Learning

Submitted By:

Arijit Bhaduri[17103268][B7]
Rahul Malhotra[17104046][B12]
Sarthak Jain[17104064][B12]

Name of Supervisor: Dr. Ankit Vidhyarthi

Submitted To: Dr. Shikha Jain, Dr. Mohit

Table of Contents

Declaration

We hereby declare that this submission is our own work and that to the best of my knowledge and belief it contains more material previously published or written by another person no material which has been accepted for the reward of any degree or diploma of the university or any other institute of higher learning, except where due acknowledgment has been made in the text.

Date:- 30/11/2019

Arijit Bhaduri (17103268)

Rahul Malhotra(17104046)

Sarthak Jain (170104064)

Certificate

This is to certify that the work entitled "**Diabetic Retinopathy Detection"** submitted by **Arijit Bhaduri, Sathak Jain and Rahul Malhotra** of Jaypee Institute of Information Technology Noida has been carried out under my supervision this work has not been submitted partially or wholly to any other university or institute for the award of any other degree or diploma.

Signature of the Supervisor

Name of the Supervisor: Dr. Ankit Vidyarthi

Date: 30/11/2019

Acknowledgment

First and foremost we would like to thank our guide Dr. Ankit Vidyarthi of Jaypee Institute of Information Technology, Noida for guiding us thoughtfully and efficiently throughout this project, giving us an opportunity to work at our own pace along our own lines, while providing us with very useful directions whenever necessary. We would also like to thank our friends and classmates for being great sources of motivation and for providing us encouragement throughout the length of this project. We offer our sincere thanks to other persons who knowingly or unknowingly helped us in this project.

Signature of Students

Arijit Bhaduri (17103268)

Rahul Malhotra (17104046)

Sarthak Jain (17104064)

1.Introduction

Worldwide, diabetic retinopathy is the leading cause of blindness among working-age adults. Hundreds and sometimes thousands of dollars are spent on detection and constant monitoring for the spread of disease in the eye. Many non-privileged people do not even get the check-up done and it results in irritation, redness and can even lead to permanent blindness for the person. This is when our “Diabetic Retinopathy Detector ” can prove to be very useful and efficient in preventing and monitoring the problem.

Our project is a tool that can be used to classify the different types of diabetic eye. It classifies the eye into one of the five classes on the basis of the location of the spot, nature of the veins. The five classes are as follows:-

Class 0: No abnormalities, Normal eye.

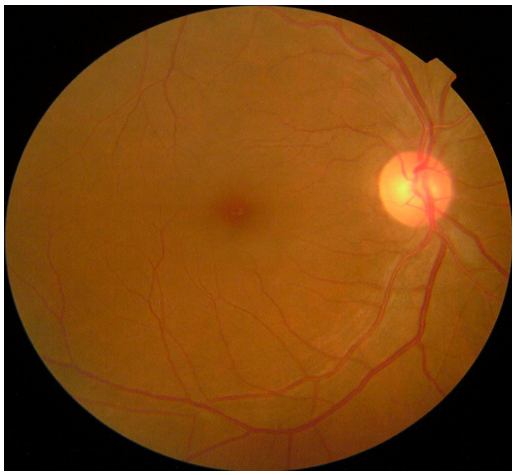

Figure 1.0.1 Normal eye

Class 1: Mild NPDR(nonproliferative diabetic retinopathy), this is the beginning of diabetic retinopathy, small blood clots start forming in the eye.

Figure 1.0.2 Mild NPDR

Class 2: Moderate NPDR, is when the condition starts deteriorating blood spot accumulates in the center between the lens and the retina which starts preventing the light rays from reaching the retina and hence the vision starts getting blur.

Figure 1.0.3 Moderate NPDR

Class 3: Severe NPDR, Definite venous beading is seen as well as the number of blood spots also increase and spread randomly across the eye.

Figure 1.0.4 Severe NPDR

Class 4: Proliferative diabetic, this is the most severe stage of diabetic retinopathy in this stage the veins starts to crumble as well as bleeding is observed and hence no light is able to hit the retina and the vision gets completely blocked.

Figure 1.0.5 Proliferative diabetic eye

Just like early detection of cancer is very important to save a person's life, early detection, and proper monitoring can save a person's eye and since the number of patients that suffer from this critical condition every year is increasing exponentially, having a tool that is handy and accurate can be considered as need of the hour. Since diabetic eye retinopathy is a very expensive process this tool can also prove to be of great help to reduce the cost of monitoring regularly for better treatment.

1.1 Problem Statement

This project helps in the detection of blindness due to diabetic eye retinopathy and classify the severeness of the disease.

2. Literature Survey

2.1 Paper 1

Title:	Deep Neural Network for Diabetic Retinopathy Detection
Publishing detail:	International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (Com-IT-Con), India,2019
Learning outcome:	It proposes to solve the problem of detecting diabetic retinopathy by using a deep learning model which has the ability to identify the pattern and classify retina images among one of the five classes, instead of using the existing applications of machine learning. It highlights the benefits of using deep learning over machine learning. Diabetic retinopathy can be broadly classified into 4 classes based on the severity of the disease. Also, people suffering from diabetes for more than 20 years are prone to this disease.

2.2 Paper 2

Title:	Balanced Mini-batch Training for Imbalanced Image DataClassification with Neural Network
Publishing detail:	First International Conference on Artificial Intelligence for Industries,2018
Learning outcome:	The difference in the number of samples causes an imbalance in training data. A neural network trained with imbalanced data often has varying levels of precision in determining each class depending on the difference in the number of class samples in the training data, which is a significant problem. Therefore, we need to look for a feasible solution to reduce imbalanced data to avoid biasing. The learning outcome of this research is:- It proposes a new algorithm known as “mini-batch” that is implemented for imbalanced classes of a dataset. The results are better than the existing algorithms like over-sampling and under-sampling.

2.3 Paper 3

Title:	Classification of Diabetic Retinopathy Images Based on Customised CNN Architecture
Publishing detail:	Amity International Conference on Artificial Intelligence (AICAI),2019
Learning outcome:	It contrasts the results of pre-trained CNN models AlexNet, VGG-16 and SqueezeNet, and a customized CNN model. The customized CNN model consists of 5 layers, 2 convolution layers to extract features and three fully connected neural networks for classification. The customized model shows better results than pre-trained models for Diabetic retinopathy images. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another.

2.4 Paper 4

Title:	Very Deep Convolutional Networks for Large-Scale Image Recognition
Publishing detail:	ImageNet Large Scale Visual Recognition Challenge,2014
Learning outcome:	This paper talks about the image recognition model known as VGG-16. It outperformed the existing models and achieved an accuracy of 92.7% in ImageNet. ImageNet is a dataset of over 14 million images belonging to 1000 different classes. Therefore, for datasets with a large number of classes using VGG-16 is preferred. 16 in VGG-16 depicts that there are 16 layers in the model that have weights.

3. Flowchart

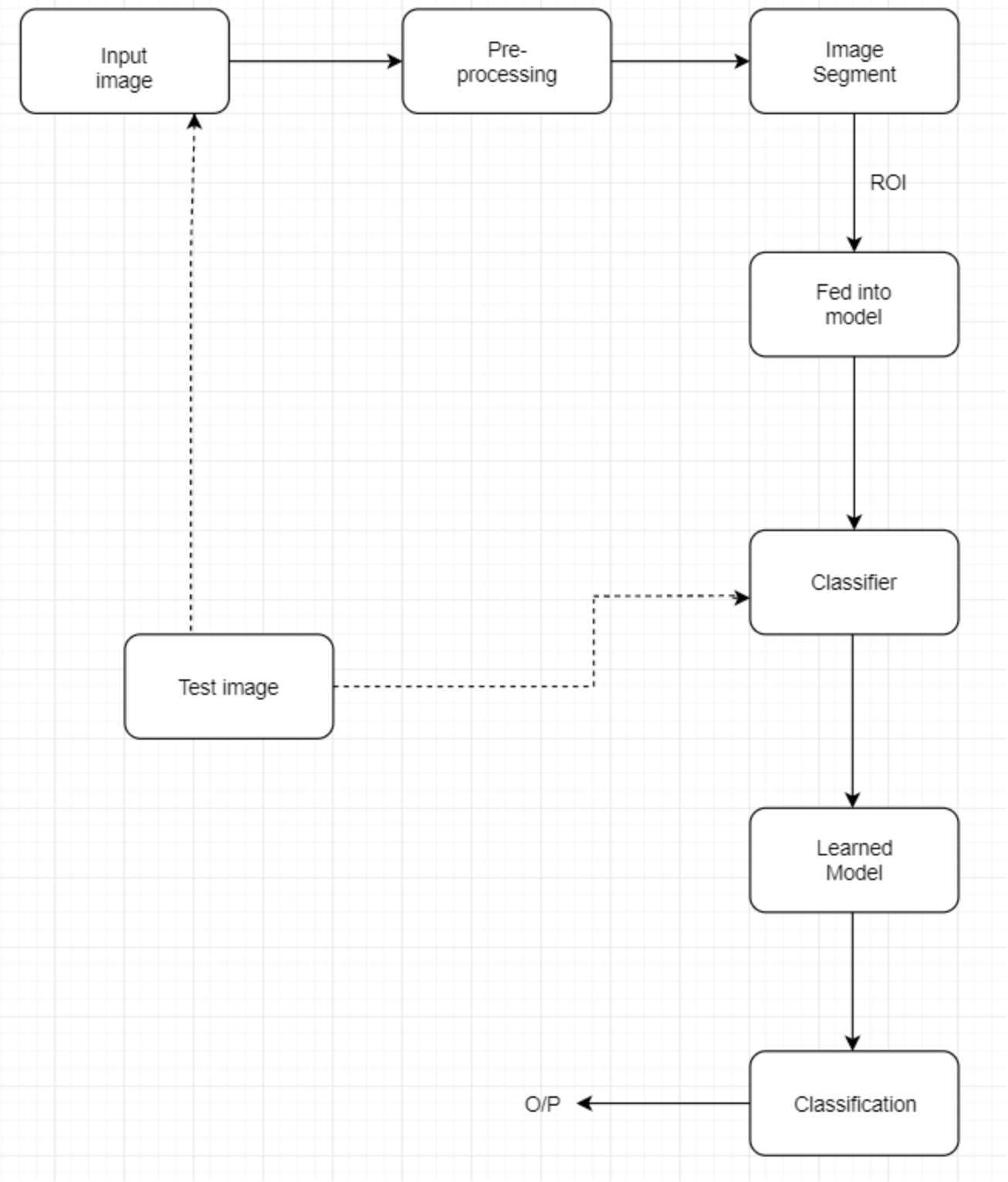

Figure 3.1 Flow Diagram 1- classification without augmentation

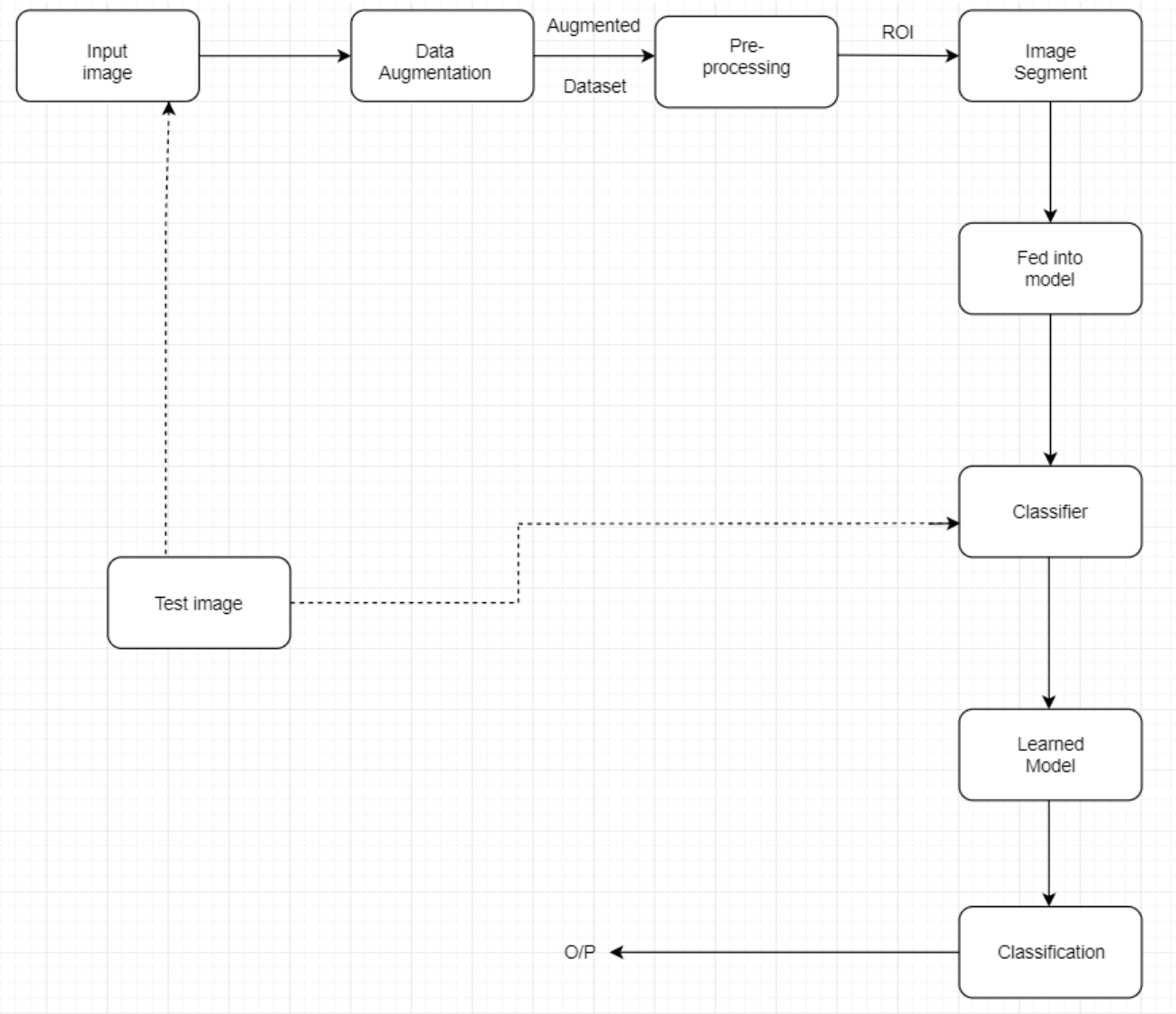

Figure 3.2 Flow Diagram 2-classification with augmentation

4.0 Dataset

We have used out data from a Kaggle challenge called “Aptos 2019 Blindness Detection”. It is a dataset of retina images taken using fundus photography (photographing the rear of an eye) under a variety of imaging conditions. Each image has been rated on the severity of diabetic retinopathy on a scale of 0 to 4.
0 - No Diabetic retinopathy, 1 - Mild, 2 - Moderate, 3 - Severe and **4 - Proliferative Diabetic Retinopathy**.

Class	Size
0	**1805**
1	**370**
2	**999**
3	**193**
4	**295**
Test Images	**1928**

Table 4.1 Image distribution per class

During pre-processing, we found some of the images to be out of focus, underexposed, or overexposed. We also found the dataset to be largely imbalanced, where most of the results were biased towards classes 0 and 2 which contained around 1800 and 1000 images respectively compared to the small number of the other images which were around 400 at most.

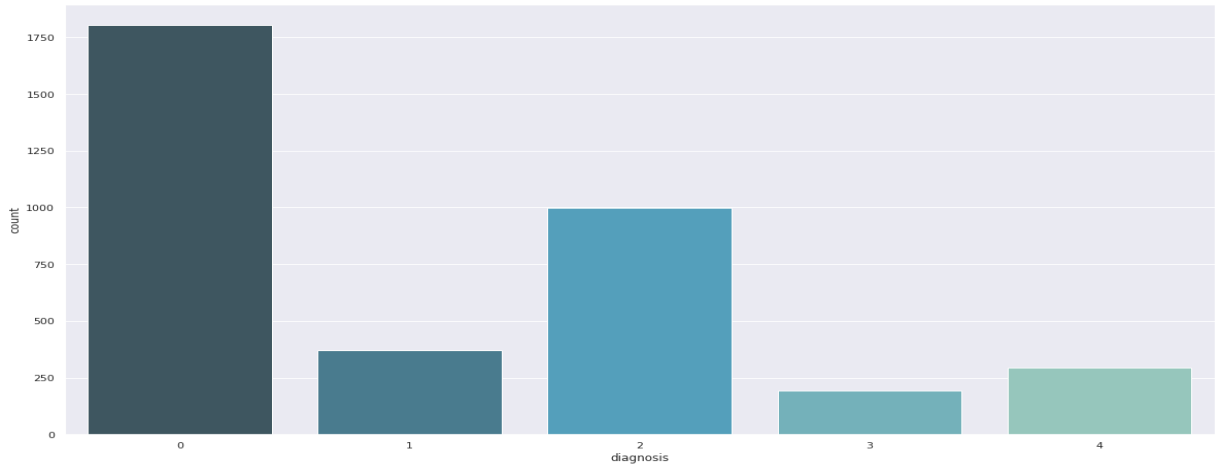

Figure 4.1 Graphical representation of the imbalanced dataset

5.0 Solution Approach

5.1 Augmentation - It is mostly used to remove problems related to imbalanced or small data. It is basically a way of creating new data with different orientations. Its benefits are
1) the ability to generate more data from limited data and
2) it prevents overfitting.

5.1.1 Methods Of Augmentation Used -

Over-Sampling - It is one of the most common ways used to tackle the issue of imbalanced data. Its aim is to increase the number of instances from the underrepresented class in the data set.

Under-Sampling - Its aim is to decrease the number of instances
from the over-represented class in the data set.

Rotation - Its aim is to randomly rotate the underrepresented class of images in the data set and create new images with different orientations.

5.2. Segmentation - It is the process of partitioning an image into multiple segments. Its goal is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. It is used in assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

5.2.1 Algorithm used for Segmentation

Gaussian Blur - It is typically used for filtering images by reducing the image noise (random variation of brightness) and reducing the details of the image. Mathematically, it is basically convolving an image with a Gaussian function. It is normally used for images with high noise and with a Gaussian function of small variance.

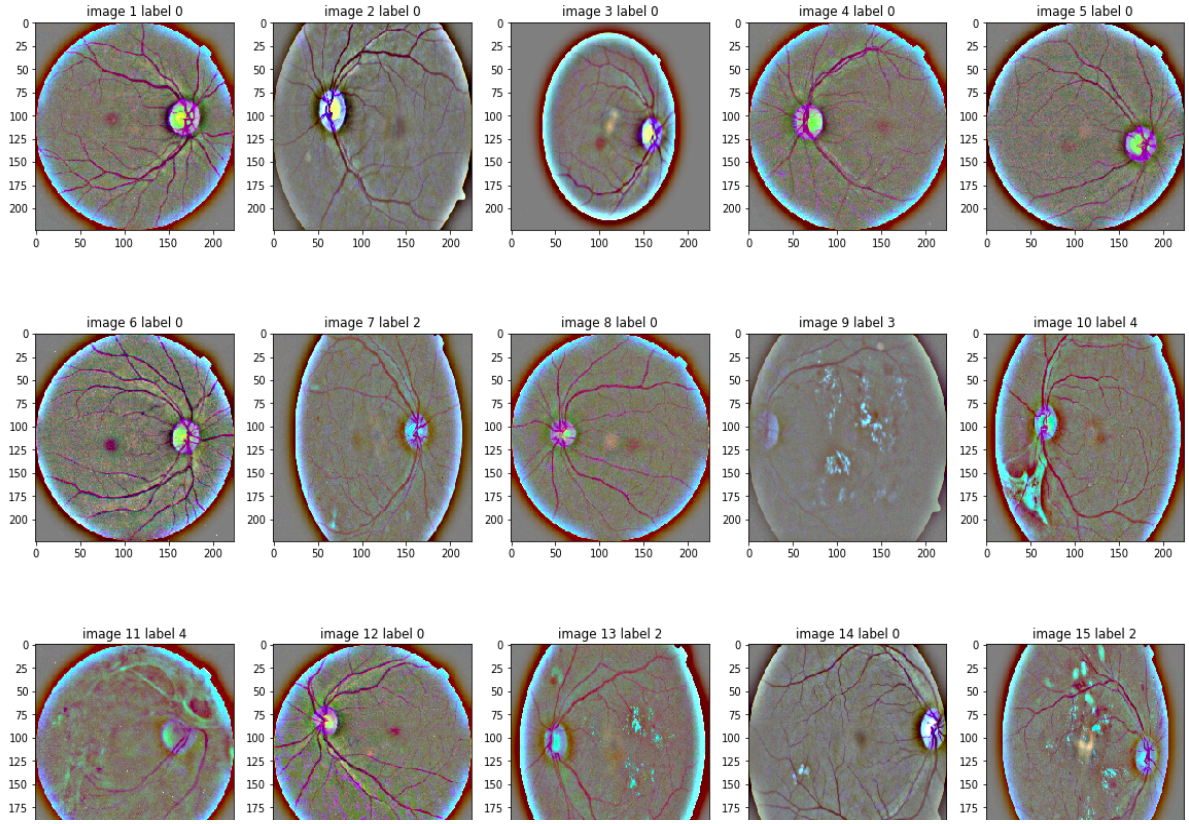

Figure 5.2.1.1 Images after Gaussian blur function

6. MODELS USED

6.1 CNN - CNN's are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. However, CNN's take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extremity.

ARCHITECTURE

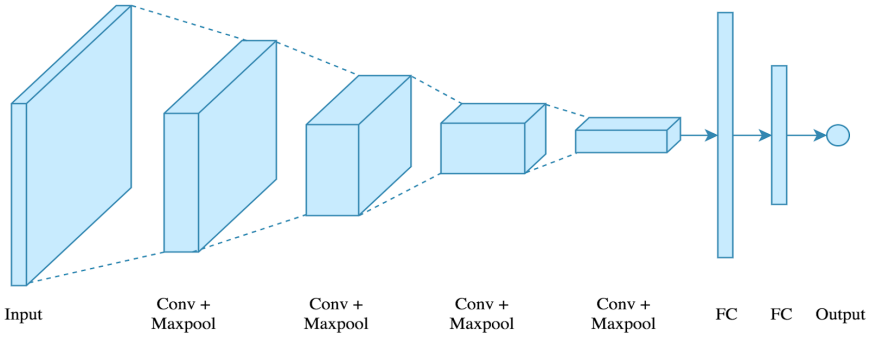

Figure 6.1.1 CNN Model architecture

The input to the conv1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). The convolution stride is fixed to 1 pixel; the spatial padding of Conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 Conv. layers. Spatial pooling is carried out by four max-pooling layers, which follow all of the Conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 512 channels each, the second performs 5-way classification and thus contains 5 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (eLU) linearity.

Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 52, 52, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 26, 26, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 18432)             0
_________________________________________________________________
dense (Dense)                (None, 512)               9437696
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 2565
=================================================================
Total params: 9,681,093
Trainable params: 9,681,093
Non-trainable params: 0
_________________________________________________________________
```

6.2 VGG16 - VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. It was one of the famous models submitted to ILSVRC-2014.

ARCHITECTURE

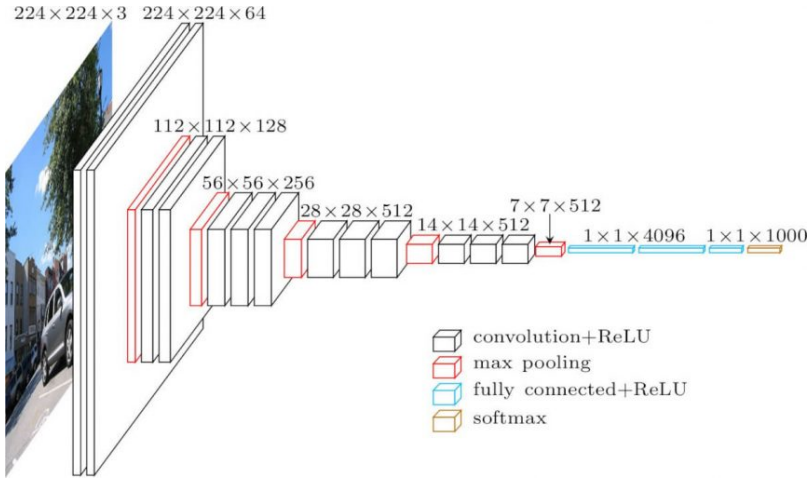

Figure 6.2 VGG16 Model architecture

The input to the Conv1 layer is a fixed 224 x 224 RBG image with 3 channels. The image is passed by a stack of two Conv. layers, where the filters were used with a very small receptive field of size 3 x 3. The convolution stride is fixed to 1 pixel. The spatial padding of Conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 Conv. layers. The spatial pooling is carried out by six max-pooling layers and one global average pooling layer. Max pooling is performed with a window size of 2 x 2 with a stride of 2. The first two pooling layers follow stacks of two Conv. layers, the next two follow a stack of three Conv. layers. And the last two max-pooling layers together follow a stack of six Conv. layers. The final global average pooling is followed by the concatenated layer of the last two max-pooling layers. The final layer is the soft-max layer. It performs 5-way classification and thus contains 5 channels (one for each class). All hidden layers are equipped with the rectification (ReLU) non-linearity.

Model Summary

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Conv2D)	(None, 112, 112, 128	73856	block1_pool[0][0]
block2_conv2 (Conv2D)	(None, 112, 112, 128	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
dil1_conv1 (Conv2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808	block5_conv1[0][0]
dil1_conv2 (Conv2D)	(None, 14, 14, 512)	2359808	dil1_conv1[0][0]
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808	block5_conv2[0][0]
dil1_conv3 (Conv2D)	(None, 14, 14, 512)	2359808	dil1_conv2[0][0]
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	block5_conv3[0][0]
dil1_pool (MaxPooling2D)	(None, 7, 7, 512)	0	dil1_conv3[0][0]
concatenate_1 (Concatenate)	(None, 7, 7, 1024)	0	block5_pool[0][0] dil1_pool[0][0]
global_average_pooling2d_1 (Glo	(None, 1024)	0	concatenate_1[0][0]
dense_1 (Dense)	(None, 5)	5125	global_average_pooling2d_1[0][0]

```
Total params: 21,799,237
Trainable params: 14,163,973
Non-trainable params: 7,635,264

None
```

6.3 AlexNet - AlexNet is the name of a convolutional neural network that has had a large impact on the field of machine learning, specifically in the application of deep learning to machine vision. It famously won the 2012 ImageNet LSVRC-2012 competition by a large margin (15.3% VS 26.2% (second place) error rates). It consisted of 11×11, 5×5,3×3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum. It attached ReLU activations after every convolutional and fully-connected layer.

ARCHITECTURE

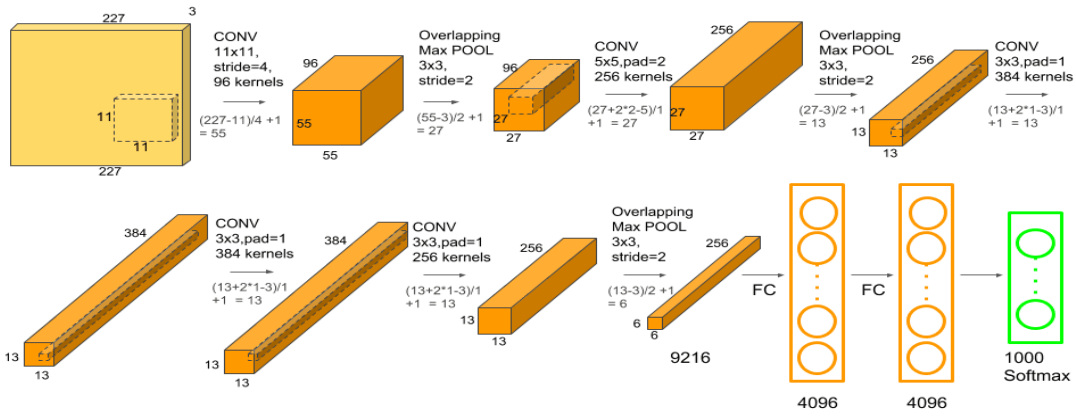

Figure 6.3 AlexNet Model architecture

The AlexNet was created with some modifications. The input to the Conv1 layer is a fixed 224 x 224 RBG image with 3 channels. The first two layers following the input layer are a stack of Conv. and max-pooling layers. The image is passed by a stack of two Conv. layers, where the filters were used with a very small receptive field of size 11 x 11. The convolution stride is fixed to 1 pixel. The Conv. layer is regularised using L2 regularizer which is used to remove over-fitting of images. The layer is then normalized using Batch Normalization(batch size of 32) to stabilize the learning process by applying normalization techniques to the activation units of the prior layer or the inputs directly. The spatial padding of Conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 11 x 11 Conv. layers. This Conv. layer is followed by a max-pooling layer which is performed with a window size of 2 x 2 with a stride of 2. The third and fifth layer starts with a zero padding of 1-pixel followed by Conv. layer with Batch Normalization and Max Pooling layer. The fourth layer is a Conv layer without any pooling layer. Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first one has 3072 channels each and

the second one has 4096 channels each, the third performs 5-way classification and thus contains 5 channels (one for each class). The final layer is the soft-max layer. The fully connected layers are regularized using Dropout regularizer with a drop ratio of 0.5. All hidden layers are equipped with the rectification (ReLU) non-linearity.

Model Summary

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 96, 96, 96)        34944
_________________________________________________________________
batch_normalization_1 (Batch (None, 96, 96, 96)        384
_________________________________________________________________
activation_1 (Activation)    (None, 96, 96, 96)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 48, 48, 96)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 48, 48, 256)       614656
_________________________________________________________________
batch_normalization_2 (Batch (None, 48, 48, 256)       1024
_________________________________________________________________
activation_2 (Activation)    (None, 48, 48, 256)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 24, 24, 256)       0
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 26, 26, 256)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 26, 26, 512)       1180160
_________________________________________________________________
batch_normalization_3 (Batch (None, 26, 26, 512)       2048
_________________________________________________________________
activation_3 (Activation)    (None, 26, 26, 512)       0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 512)       0
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 15, 15, 512)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 1024)      4719616
_________________________________________________________________
batch_normalization_4 (Batch (None, 15, 15, 1024)      4096
_________________________________________________________________
activation_4 (Activation)    (None, 15, 15, 1024)      0
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 17, 17, 1024)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 17, 17, 1024)      9438208
_________________________________________________________________
batch_normalization_5 (Batch (None, 17, 17, 1024)      4096
_________________________________________________________________
activation_5 (Activation)    (None, 17, 17, 1024)      0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 1024)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_1 (Dense)              (None, 3072)              201329664
_________________________________________________________________
batch_normalization_6 (Batch (None, 3072)              12288
_________________________________________________________________
activation_6 (Activation)    (None, 3072)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 3072)              0
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              12587008
_________________________________________________________________
batch_normalization_7 (Batch (None, 4096)              16384
_________________________________________________________________
activation_7 (Activation)    (None, 4096)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 20485
_________________________________________________________________
batch_normalization_8 (Batch (None, 5)                 20
_________________________________________________________________
activation_8 (Activation)    (None, 5)                 0
=================================================================
Total params: 229,965,081
Trainable params: 229,944,911
Non-trainable params: 20,170
_________________________________________________________________
```

7.Testing and Results

1) We first trained our dataset on a basic CNN model with five layers. We trained it for 10 epochs. We had used Adam optimizer for a better learning rate and to achieve desired results faster.

```
(3662, 224, 224, 3)
(3662, 5)
Train on 3295 samples, validate on 367 samples
Epoch 1/10
3295/3295 [==============================] - 299s 91ms/sample - loss: 18.0810 - mae: 0.1990 - accuracy: 0.5347 - val_loss: 1.4807 - val_mae: 0.2509 - val_accuracy: 0.5014
Epoch 2/10
3295/3295 [==============================] - 487s 148ms/sample - loss: 1.1075 - mae: 0.1654 - accuracy: 0.6355 - val_loss: 0.8849 - val_mae: 0.1411 - val_accuracy: 0.6894
Epoch 3/10
3295/3295 [==============================] - 236s 72ms/sample - loss: 0.9496 - mae: 0.1535 - accuracy: 0.6753 - val_loss: 1.2237 - val_mae: 0.2097 - val_accuracy: 0.6213
Epoch 4/10
3295/3295 [==============================] - 233s 71ms/sample - loss: 1.0208 - mae: 0.1521 - accuracy: 0.6765 - val_loss: 0.8701 - val_mae: 0.1304 - val_accuracy: 0.7166
Epoch 5/10
3295/3295 [==============================] - 257s 78ms/sample - loss: 0.9642 - mae: 0.1473 - accuracy: 0.6926 - val_loss: 0.8188 - val_mae: 0.1430 - val_accuracy: 0.7003
Epoch 6/10
3295/3295 [==============================] - 267s 81ms/sample - loss: 0.7938 - mae: 0.1425 - accuracy: 0.7199 - val_loss: 0.8457 - val_mae: 0.1445 - val_accuracy: 0.7030
Epoch 7/10
3295/3295 [==============================] - 265s 80ms/sample - loss: 0.7924 - mae: 0.1372 - accuracy: 0.7235 - val_loss: 0.7188 - val_mae: 0.1245 - val_accuracy: 0.7384
Epoch 8/10
3295/3295 [==============================] - 248s 75ms/sample - loss: 0.6633 - mae: 0.1292 - accuracy: 0.7554 - val_loss: 0.7327 - val_mae: 0.1366 - val_accuracy: 0.6921
Epoch 9/10
3295/3295 [==============================] - 241s 73ms/sample - loss: 0.6404 - mae: 0.1229 - accuracy: 0.7675 - val_loss: 0.6680 - val_mae: 0.1325 - val_accuracy: 0.7411
Epoch 10/10
3295/3295 [==============================] - 242s 73ms/sample - loss: 0.6183 - mae: 0.1163 - accuracy: 0.7772 - val_loss: 0.8977 - val_mae: 0.1326 - val_accuracy: 0.7302
```

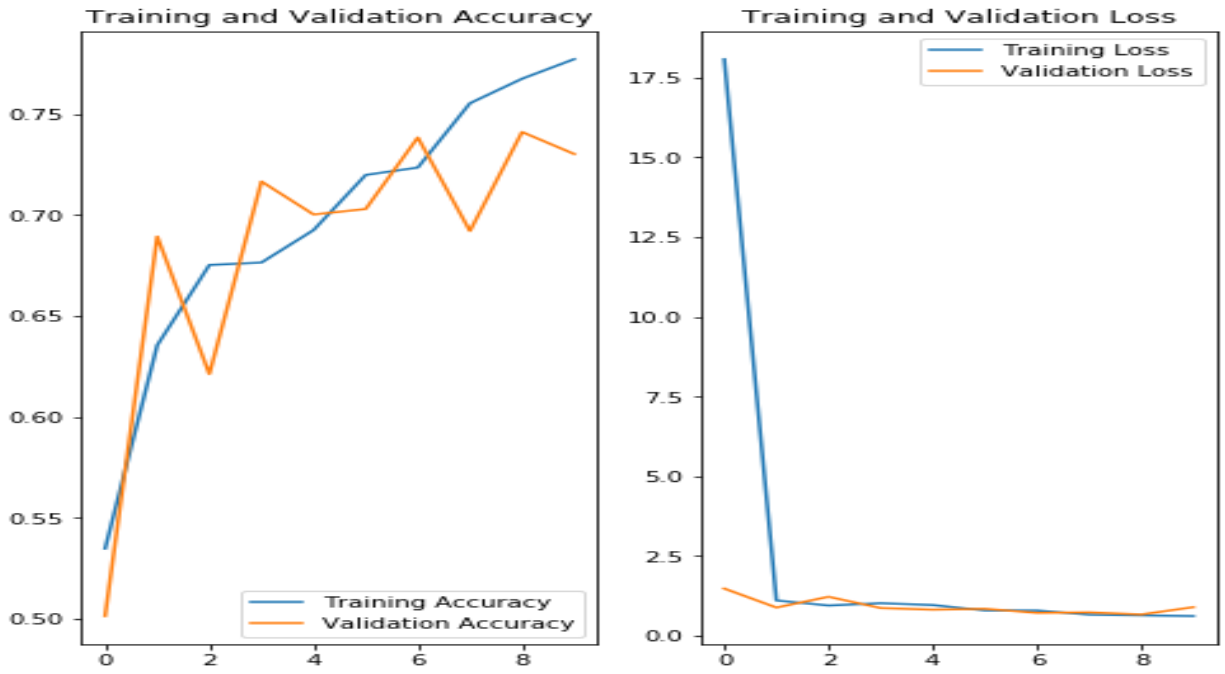

Figure 7.1 Graphs of training and validation accuracy and loss

2) Next, we trained our dataset on our customized version of AlexNet which has even more depth consisting of 7 layers. Here also we had used Adam optimizer and we trained in for 5 epochs.

```
Epoch 1/5
92/91 [==============================] - 1428s 16s/step - loss: 1.1588 - accuracy: 0.5910 - val_loss: 1.1978 - val_accuracy: 0.5293
val_kappa: 0.5878
Epoch 2/5
92/91 [==============================] - 1434s 16s/step - loss: 1.0728 - accuracy: 0.6333 - val_loss: 1.0464 - val_accuracy: 0.6903
val_kappa: 0.5967
Epoch 3/5
92/91 [==============================] - 1433s 16s/step - loss: 1.0512 - accuracy: 0.6473 - val_loss: 1.0295 - val_accuracy: 0.6971
val_kappa: 0.5558
Epoch 4/5
92/91 [==============================] - 1476s 16s/step - loss: 1.0213 - accuracy: 0.6705 - val_loss: 1.0795 - val_accuracy: 0.6044
val_kappa: 0.6780
Epoch 5/5
92/91 [==============================] - 1472s 16s/step - loss: 1.0195 - accuracy: 0.6664 - val_loss: 1.0629 - val_accuracy: 0.6398
val_kappa: 0.3261
```

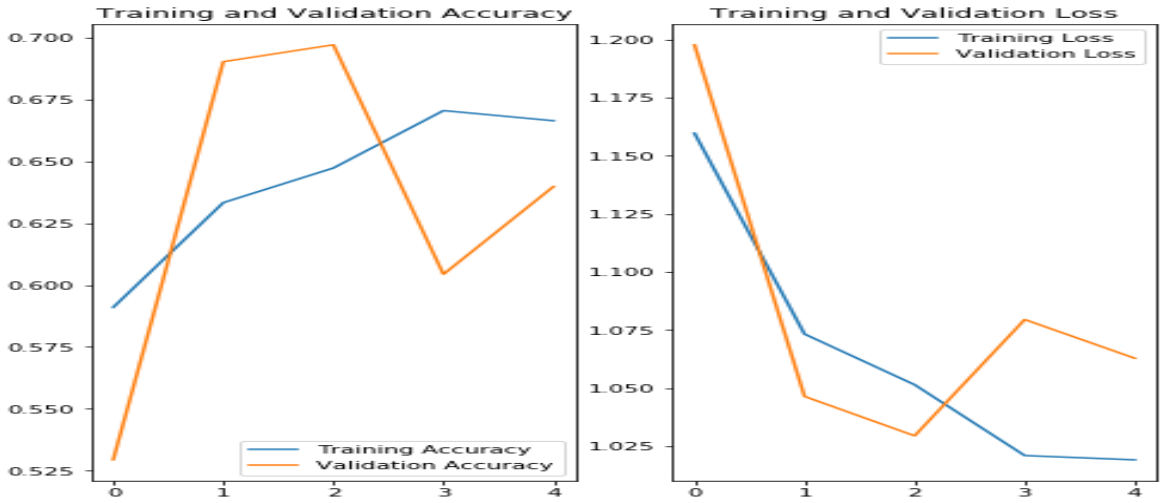

Figure 7.2 Graphs of training and validation accuracy and loss

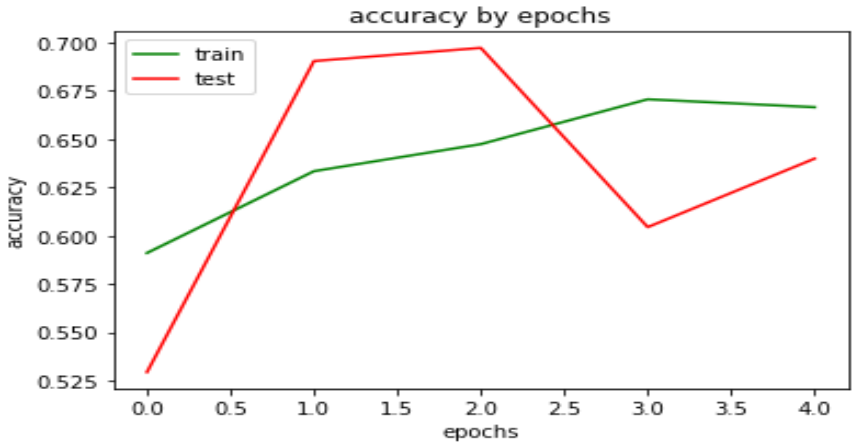

Figure 7.3 Graphs of train and test accuracy and loss

3) Next, we trained our dataset on our customized version of VGG 16 which had even more deep layers consisting of 16 layers. We trained it for 10 epochs.

```
Epoch 1/10
111/111 [==============================] - 2822s 25s/step - loss: 1.6712 - acc: 0.5453 - val_loss: 1.0696 -
Epoch 2/10
111/111 [==============================] - 2806s 25s/step - loss: 1.0467 - acc: 0.6480 - val_loss: 0.9740 -
Epoch 3/10
111/111 [==============================] - 2784s 25s/step - loss: 0.9635 - acc: 0.6975 - val_loss: 0.9662 -
Epoch 4/10
111/111 [==============================] - 2785s 25s/step - loss: 0.9264 - acc: 0.7031 - val_loss: 0.8891 -
Epoch 5/10
111/111 [==============================] - 2777s 25s/step - loss: 0.8637 - acc: 0.7352 - val_loss: 0.9092 -
Epoch 6/10
111/111 [==============================] - 2773s 25s/step - loss: 0.8520 - acc: 0.7406 - val_loss: 0.9049 -
Epoch 7/10
111/111 [==============================] - 2773s 25s/step - loss: 0.8033 - acc: 0.7572 - val_loss: 0.8454 -
Epoch 8/10
111/111 [==============================] - 2774s 25s/step - loss: 0.8095 - acc: 0.7457 - val_loss: 0.8818 -
Epoch 9/10
111/111 [==============================] - 2775s 25s/step - loss: 0.7724 - acc: 0.7676 - val_loss: 0.9246 -
Epoch 10/10
111/111 [==============================] - 2775s 25s/step - loss: 0.7483 - acc: 0.7749 - val_loss: 0.8315 -
<keras.callbacks.History at 0x7faabc59dfd0>
```

4)Next, before training our dataset, we used augmentation to remove imbalanced data and fitted it into our first basic CNN model. We had used Early Stopping regularizer to stop the epoch calculations the moment the accuracy went low from its previous epoch. Due to this technique, it stopped at 112 epochs while it was to be trained to 120 epochs.

```
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.0586 - mae: 0.0096 - accuracy: 0.9815 - val_loss: 2.2792 - val_mae: 0.1063 - val_accuracy: 0.7357
Epoch 103/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.0960 - mae: 0.0146 - accuracy: 0.9700 - val_loss: 2.7899 - val_mae: 0.1076 - val_accuracy: 0.7357
Epoch 104/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.1303 - mae: 0.0182 - accuracy: 0.9627 - val_loss: 2.9733 - val_mae: 0.1138 - val_accuracy: 0.7248
Epoch 105/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.1359 - mae: 0.0182 - accuracy: 0.9654 - val_loss: 3.1738 - val_mae: 0.1023 - val_accuracy: 0.7411
Epoch 106/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.1872 - mae: 0.0227 - accuracy: 0.9484 - val_loss: 3.4673 - val_mae: 0.1132 - val_accuracy: 0.7275
Epoch 107/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.1447 - mae: 0.0189 - accuracy: 0.9602 - val_loss: 2.6375 - val_mae: 0.1064 - val_accuracy: 0.7411
Epoch 108/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.0887 - mae: 0.0126 - accuracy: 0.9763 - val_loss: 2.8048 - val_mae: 0.1059 - val_accuracy: 0.7357
Epoch 109/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.0648 - mae: 0.0115 - accuracy: 0.9778 - val_loss: 3.2987 - val_mae: 0.1176 - val_accuracy: 0.7030
Epoch 110/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.1016 - mae: 0.0142 - accuracy: 0.9700 - val_loss: 3.5832 - val_mae: 0.1195 - val_accuracy: 0.7084
Epoch 111/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.1093 - mae: 0.0154 - accuracy: 0.9690 - val_loss: 3.2322 - val_mae: 0.1195 - val_accuracy: 0.6975
Epoch 112/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.0897 - mae: 0.0138 - accuracy: 0.9727 - val_loss: 3.2899 - val_mae: 0.1098 - val_accuracy: 0.7275
Epoch 113/200
3295/3295 [==============================] - 4s 1ms/sample - loss: 0.0774 - mae: 0.0099 - accuracy: 0.9812 - val_loss: 2.6238 - val_mae: 0.1017 - val_accuracy: 0.7493
Epoch 114/200
1056/3295 [========>.....................] - ETA: 2s - loss: 0.0804 - mae: 0.0113 - accuracy: 0.9763
```

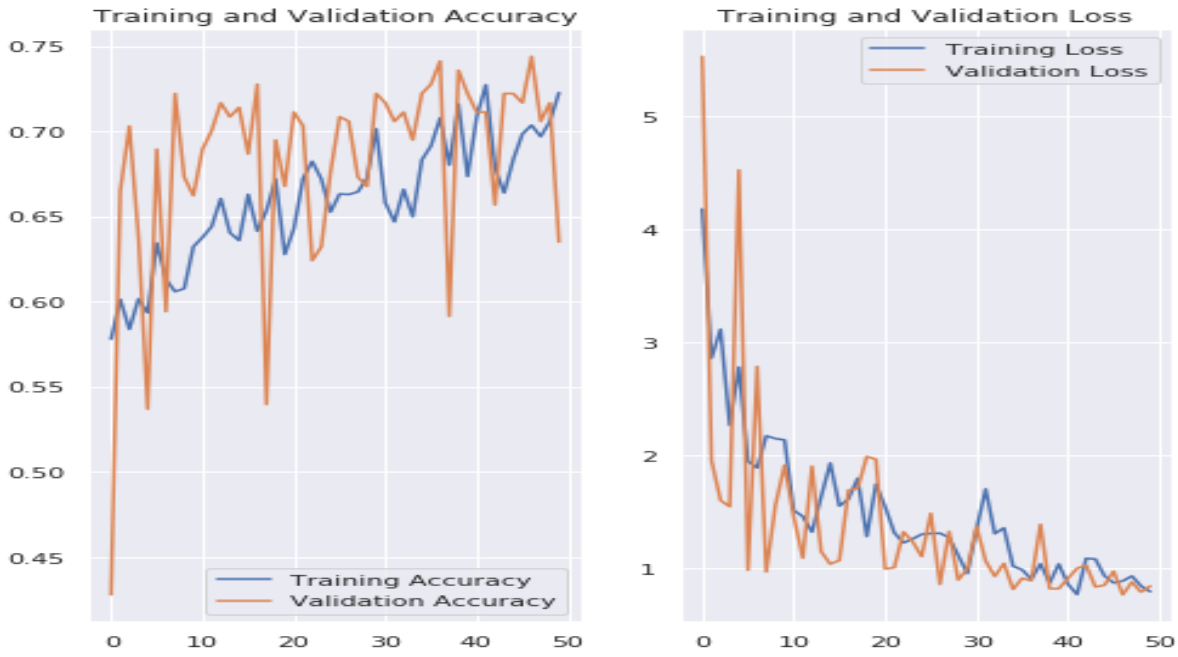

Figure 7.4 Graphs of train and validation accuracy and loss

5) Next, we trained our dataset on our customized version of AlexNet before training our dataset, we used augmentation to remove imbalanced data. We had used the Early Stopping regularizer. We trained it for 10 epochs.

```
[ ] Epoch 1/10
92/91 [==============================] - 18s 201ms/step - loss: 0.7077 - accuracy: 0.8245 - val_loss: 1.1062 - val_accuracy: 0.5853
val_kappa: 0.5340
Epoch 2/10
92/91 [==============================] - 18s 200ms/step - loss: 0.7023 - accuracy: 0.8276 - val_loss: 1.6955 - val_accuracy: 0.4843
val_kappa: 0.3970
Epoch 3/10
92/91 [==============================] - 18s 200ms/step - loss: 0.6762 - accuracy: 0.8389 - val_loss: 1.4883 - val_accuracy: 0.5116
val_kappa: 0.6356
Epoch 4/10
92/91 [==============================] - 18s 200ms/step - loss: 0.6688 - accuracy: 0.8498 - val_loss: 1.1110 - val_accuracy: 0.6357
val_kappa: 0.3588
Epoch 5/10
92/91 [==============================] - 18s 200ms/step - loss: 0.6478 - accuracy: 0.8559 - val_loss: 2.3399 - val_accuracy: 0.3888
val_kappa: 0.4099
Epoch 6/10
92/91 [==============================] - 18s 200ms/step - loss: 0.6335 - accuracy: 0.8662 - val_loss: 1.0981 - val_accuracy: 0.5962
val_kappa: 0.4701
Epoch 7/10
92/91 [==============================] - 18s 201ms/step - loss: 0.6294 - accuracy: 0.8651 - val_loss: 1.0239 - val_accuracy: 0.7135
val_kappa: 0.6934
Epoch 8/10
92/91 [==============================] - 18s 200ms/step - loss: 0.6141 - accuracy: 0.8638 - val_loss: 1.5913 - val_accuracy: 0.5402
val_kappa: 0.4060
Epoch 9/10
92/91 [==============================] - 18s 199ms/step - loss: 0.5873 - accuracy: 0.8849 - val_loss: 1.3315 - val_accuracy: 0.4829
val_kappa: 0.0027
Epoch 10/10
92/91 [==============================] - 18s 199ms/step - loss: 0.5907 - accuracy: 0.8860 - val_loss: 1.5327 - val_accuracy: 0.5935
val_kappa: 0.6767
```

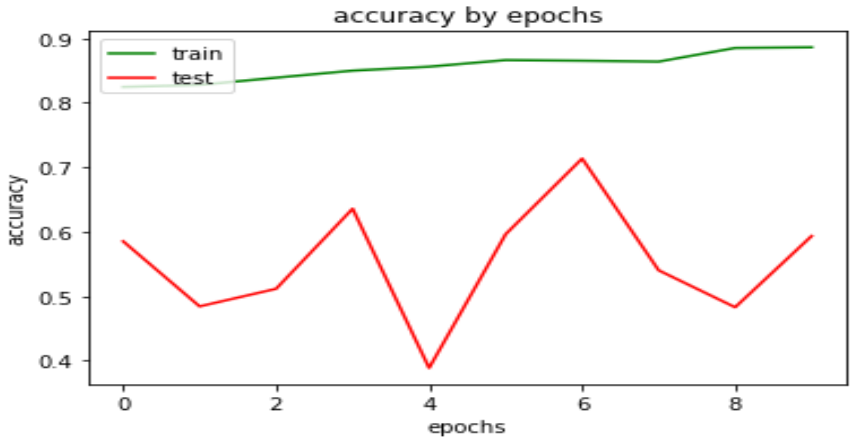

Figure 7.5 Graphs of train and test accuracy

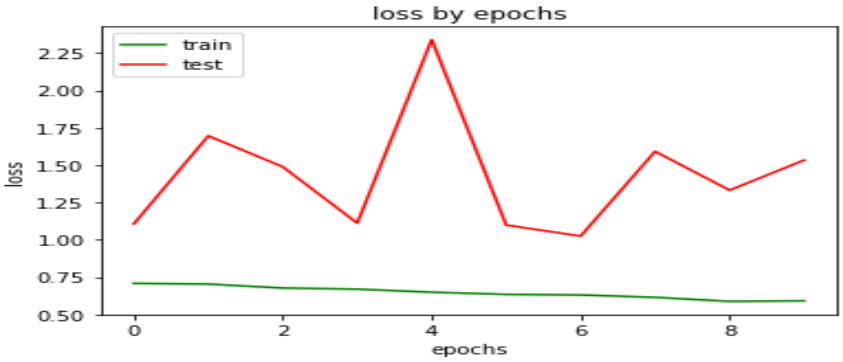

Figure 7.6 Graphs of train and test loss

5) Finally, we trained our dataset on our customized VGG model before training our dataset, we used augmentation to remove imbalanced data. We had used the Early Stopping regularizer. We trained it for 30 epochs, but it stopped on 12 epochs due to overfitting condition

```
1/30
4 [==============================] - 45s 392ms/step - loss: 0.6220 - mae: 0.1079 - accuracy: 0.8159 - val_loss: 0.7443 - val_mae: 0.1359 - val_accuracy: 0.6
2/30
4 [==============================] - 43s 380ms/step - loss: 0.6067 - mae: 0.1065 - accuracy: 0.8184 - val_loss: 0.5828 - val_mae: 0.1089 - val_accuracy: 0.7
3/30
4 [==============================] - 44s 382ms/step - loss: 0.6019 - mae: 0.1042 - accuracy: 0.8222 - val_loss: 0.7703 - val_mae: 0.1328 - val_accuracy: 0.7
4/30
4 [==============================] - 43s 380ms/step - loss: 0.6222 - mae: 0.1069 - accuracy: 0.8183 - val_loss: 0.7700 - val_mae: 0.1334 - val_accuracy: 0.7
5/30
4 [==============================] - 44s 382ms/step - loss: 0.5623 - mae: 0.1010 - accuracy: 0.8249 - val_loss: 0.6422 - val_mae: 0.1142 - val_accuracy: 0.7
6/30
4 [==============================] - 43s 380ms/step - loss: 0.5818 - mae: 0.1014 - accuracy: 0.8275 - val_loss: 0.6548 - val_mae: 0.1224 - val_accuracy: 0.7
7/30
4 [==============================] - 44s 385ms/step - loss: 0.5593 - mae: 0.0995 - accuracy: 0.8384 - val_loss: 0.8386 - val_mae: 0.1238 - val_accuracy: 0.7
8/30
4 [==============================] - 43s 376ms/step - loss: 0.5586 - mae: 0.1006 - accuracy: 0.8197 - val_loss: 0.6718 - val_mae: 0.1142 - val_accuracy: 0.7
9/30
4 [==============================] - 43s 379ms/step - loss: 0.5446 - mae: 0.0955 - accuracy: 0.8382 - val_loss: 0.6577 - val_mae: 0.1124 - val_accuracy: 0.7
10/30
4 [==============================] - 43s 374ms/step - loss: 0.5446 - mae: 0.0982 - accuracy: 0.8302 - val_loss: 0.7691 - val_mae: 0.1235 - val_accuracy: 0.7
11/30
4 [==============================] - 42s 372ms/step - loss: 0.4914 - mae: 0.0923 - accuracy: 0.8453 - val_loss: 0.6561 - val_mae: 0.1088 - val_accuracy: 0.7
12/30
4 [==============================] - 42s 371ms/step - loss: 0.5510 - mae: 0.0961 - accuracy: 0.8353 - val_loss: 0.7791 - val_mae: 0.1273 - val_accuracy: 0.6
00012: early stopping
```

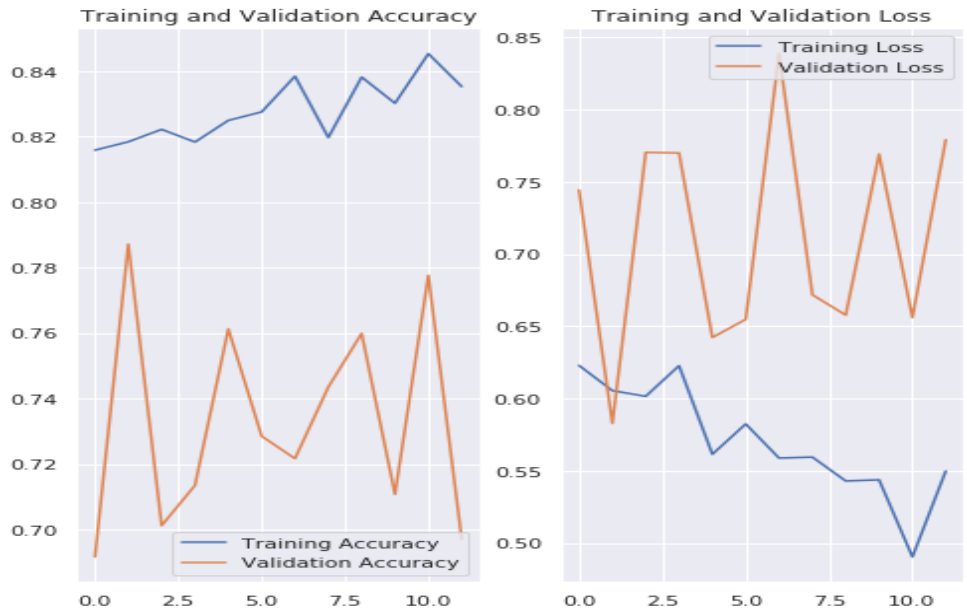

Figure 7.7 Graph of training and Validation Accuracy

8.Findings and Conclusion

We applied our models directly on the raw dataset (extracted from kaggle challenge) ie. without data augmentation or image segmentation. After the findings of the results, it was observed that the accuracies were not up to the mark(<80%).

Raw Dataset	CNN	AlexNet	VGG-16
Training	77.72%	66.64%	77.49%
Validation	73.02%	63.98%	--

We found out that data imbalance is a major issue and had to be resolved to improve the accuracy

Hence, it was clear to improve the accuracy we need to optimize the models by experimenting with different approaches.

8.1 Conclusion

Dataset augmentation was tested which increased the number of images in the classes which had fewer images hence this reduced the data imbalance problem is reduced to some extent. After augmentation was implemented the following results were observed.

Augmented Dataset	CNN	AlexNet	VGG-16
Training	98.12%	88.6%	84.54%
Validation	74.93%	71.35%	77.76%

Hence augmentation of the dataset can be very helpful to increase the accuracy since it helps the model to learn some new features from the images as well as reduce data imbalance.

9. Future Scope

1) Data augmentation may increase the accuracy of the models but it is not the best approach to resolve class imbalance problem since after some point it leads to overfitting and hence will certainly affect the accuracy of the model in a negative manner. Hence an **Optimized Approach** for reducing the class imbalance problem needs to be implemented.

2)It can be observed as the depth of the model increases it requires more computation power to train the model. Since this is a real-life medical problem hence it is very important that the result is very accurate and hence we will be implementing our models in a more powerful GPU unit so as to get more accurate results.

10.References

1.Ryota Shimizu et al., “Balanced Mini-batch Training for Imbalanced Image Data Classification with Neural Network”, First International Conference on Artificial Intelligence for Industries,2018, 27-30. 10.1109/AI4I.2018.8665709.

2. M. Arora and M. Pandey, "Deep Neural Network for Diabetic Retinopathy Detection," *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, Faridabad, India, 2019, pp. 189-193.doi: 10.1109/COMITCon.2019.8862217

3. Mobeen-ur-Rehman, S. H. Khan, Z. Abbas and S. M. Danish Rizvi, "Classification of Diabetic Retinopathy Images Based on Customised CNN Architecture," *2019 Amity International Conference on Artificial Intelligence (AICAI)*, Dubai, United Arab Emirates, 2019, pp. 244-248. DOI: 10.1109/AICAI.2019.8701231

4. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

5. M. Yang and K. P. Sinaga, "A Feature-Reduction Multi-View k-Means Clustering Algorithm," in *IEEE Access*, vol. 7, pp. 114472-114486, 2019.

DOI: 10.1109/ACCESS.2019.2934179

6. Ö. Deperlıoğlu and U. Köse, "Diagnosis of Diabetic Retinopathy by Using Image Processing and Convolutional Neural Network," *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Ankara, 2018, pp. 1-5. DOI: 10.1109/ISMSIT.2018.8567055

7. A. Fawzi, H. Samulowitz, D. Turaga and P. Frossard, "Adaptive data augmentation for image classification," *2016 IEEE International Conference on Image Processing (ICIP)*, Phoenix, AZ, 2016, pp. 3688-3692. DOI: 10.1109/ICIP.2016.7533048

8. T. E. Tallo and A. Musdholifah, "The Implementation of Genetic Algorithm in Smote (Synthetic Minority Oversampling Technique) for Handling Imbalanced Dataset Problem," *2018 4th International Conference on Science and Technology (ICST)*, Yogyakarta, 2018, pp. 1-4. doi: 10.1109/ICSTC.2018.8528591

9. T. Deepa and M. Punithavalli, "A new sampling technique and SVM classification for feature selection in high-dimensional Imbalanced dataset," *2011 3rd International*

Conference on Electronics Computer Technology, Kanyakumari, 2011, pp. 395-398. DOI: 10.1109/ICECTECH.2011.5942028

10. Zhang Sheng, Shang Xiuyu, Wang Wei and Huang Xiuli, "Optimizing the classification accuracy of an imbalanced dataset based on SVM," *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, Taiyuan, 2010, pp. V4-338-V4-341. DOI: 10.1109/ICCASM.2010.5620370

11. J. Novosel, K. A. Vermeer, L. Pierrache, C. C. W. Klaver, L. I. van den Born and L. J. van Vliet, "Method for segmentation of the layers in the outer retina," *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Milan, 2015, pp. 5646-5649. DOI: 10.1109/EMBC.2015.7319673

12. S. Muniyappan, A. Allirani, and S. Saraswathi, "A novel approach for image enhancement by using contrast limited adaptive histogram equalization method," *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, Tiruchengode, 2013, pp. 1-6. DOI: 10.1109/ICCCNT.2013.6726470