Diabetic Retinopathy Detection

CSE 730 Deep Learning for Visual Recognition with Applications to Medical Imaging Analysis, Fall 2017

University at Buffalo
Seminar Project Report

Lakshmi Prasanna Ethiraj
lethiraj@buffalo.edu (5011-3455)

Nagadeesh Nagaraja
nagadees@buffalo.edu (5020-7355)

Sunil Kunjappan Vasu
sunilkun@buffalo.edu (5020-5673)

Abstract

Diabetic retinopathy is the leading cause of blindness in the working-age population of the developed world. It is estimated to affect over 93 million people. The current detection process is time consuming and requires manual interpretation where trained clinician need to identify each of the images. This process can also lead to more error. Hence the need for an automated, efficient and faster Diabetic retinopathy detection has been recognized long back. There are multiple efforts for making good progress using machine learning technique like image classification, pattern recognition and deep learning. With the advancement of deep learning the older solution can improved to produce better accuracy. In this project we use deep learning models coupled with transfer learning to create better accuracies. Data Augmentation technique was also used to understand how it affects performance for each of the model. This report illustrate the details of our approaches for Diabetic retinopathy detection.

1. INTRODUCTION

Diabetic retinopathy is the leading cause of blindness in the working-age population of the developed world. It is estimated to affect over 93 million people. The current detection process is time consuming and requires manual interpretation where trained clinician need to identify each of the images. This process can also lead to more error. Hence the need for an automated, efficient and faster Diabetic retinopathy detection has been recognized long back. There are multiple efforts for making good progress using machine learning technique like image classification, pattern recognition and deep learning. With the advancement of deep learning the older solution can improved to produce better accuracy. In this project we use deep learning models coupled with transfer learning to create better accuracies.

2. TYPE OF TASK AND TASK DESCRIPTION

The main task is to classify the color retinal image into 5 categories. The labels were provided by clinicians who rated the presence of diabetic retinopathy in each image by a scale of “0, 1, 2, 3, 4”, which represent “no DR”, “mild”, “moderate”, “severe”, “proliferative DR” respectively.

3. DATASET

The color retina images are downloaded from the Kaggle website. The dataset contains 35126 high resolution images under a variety of imaging conditions. These retina images were obtained from a group of subjects, and for each subject two images were obtained for left and right eyes, respectively. The labels were provided by clinicians who rated the presence of diabetic retinopathy in each image by a scale of “0, 1, 2, 3, 4”, which represent “no DR”, “mild”, “moderate”, “severe”, “proliferative DR” respectively. As mentioned in the description of the dataset, the images in the dataset come from different models and types of camera, which can affect the visual appearance of left vs. right. The samples images are shown in Fig 1. Also, the dataset doesn’t have the equal distributions among the 5 scales. As one can expect, normal data with label “0” is the biggest class in the whole dataset, while “proliferative DR” data is the smallest class. Fig 4 shows counts of images for different scales in the dataset.

For Data augmentation, we did two approaches. The first approach was to flip the right eye images so that all the images are aligned. This flipping example is shown in Fig 2. The next augmentation method we adopted was to do rotation and random stretches and flips. In this method multiple images are augmented so the same image. The main intuition to make the model learn as much feature of the image and not to overfit the model. A sample example is shown in Fig 3.

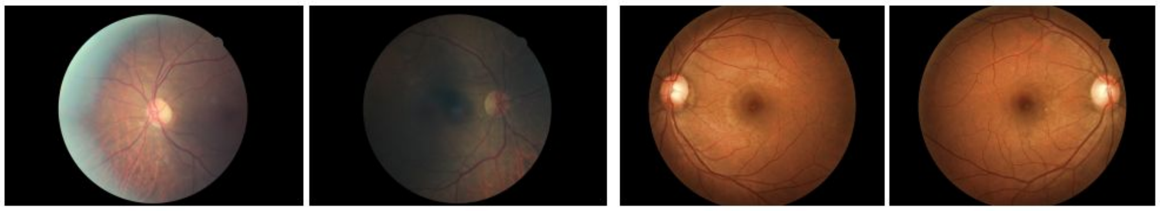


Fig1 :The samples images from Diabetic Retinopathy dataset

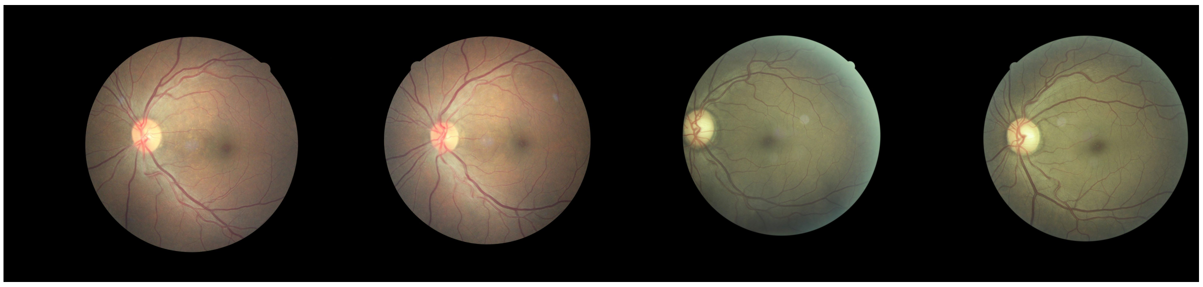


Fig2 : Sample Flipped Images of Diabetic Retinopathy dataset

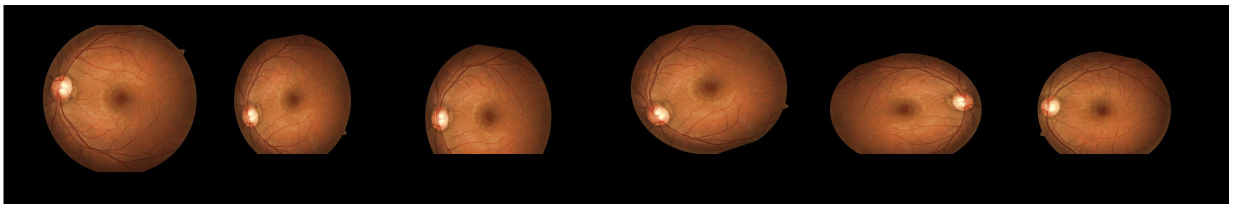


Fig3: Original Image followed by augmented images

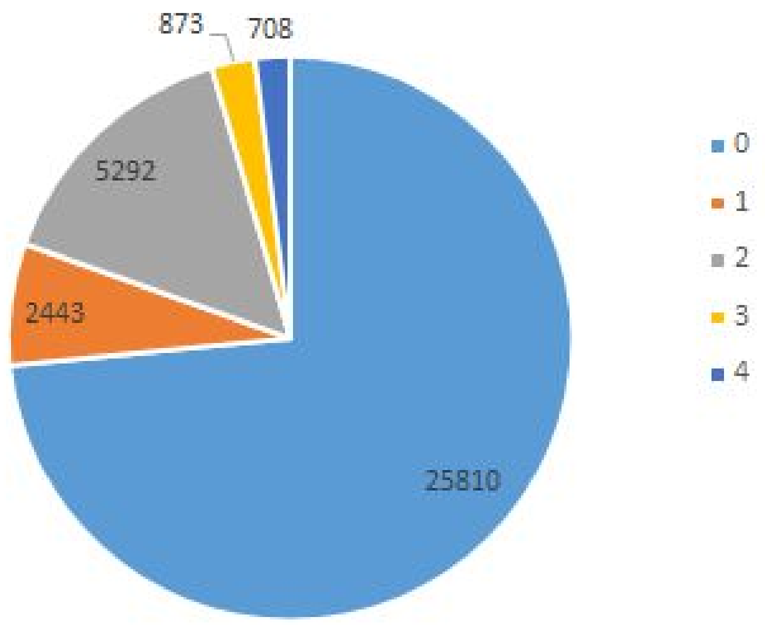


Fig4: Diabetic Retinopathy image distribution

4. MACHINE LEARNING MODELS & METHODS USED

It's well known that CNN training require significant amounts of data and resources to train. For example, the ImageNet ILSVRC model was trained on 1.2 million images over the period of 2–3 weeks across multiple GPUs. Hence creating a model from scratch is a very cumbersome and time consuming task. So in this situation we adopted transfer learning and hyperparameter tuning. These methods can be summarized as follows:

1. Transfer Learning: The last fully connected layer of a pre trained CNN is removed and treated as a feature extractor for the new dataset. Once we have successfully extracted all the features for all images we train a classifier on the new dataset.
2. Hyperparameter-tuning: These are the parameters that are not learnt by the network on its own, adjusting these parameters according to the specific needs of the network is essential in improving the performance.

So in this project we used 5 standard models and performed Transfer Learning and understand how well these models perform with the Diabetic Retinopathy image dataset.

We have used 4 models on the data set to compare results for retinopathy detection.

- AlexNet
- VGG16
- Inception V3
- MobileNet

I. VGG16 Network

A. Preprocessing the images for VGG16:

The data set was normalised to have almost same number of images as possible. Fig-6 shows counts of images for different scales in the training dataset. We were unable to perfectly normalise due to total number of images found in the class 3 and class 4, which is not sufficient for training purposes.

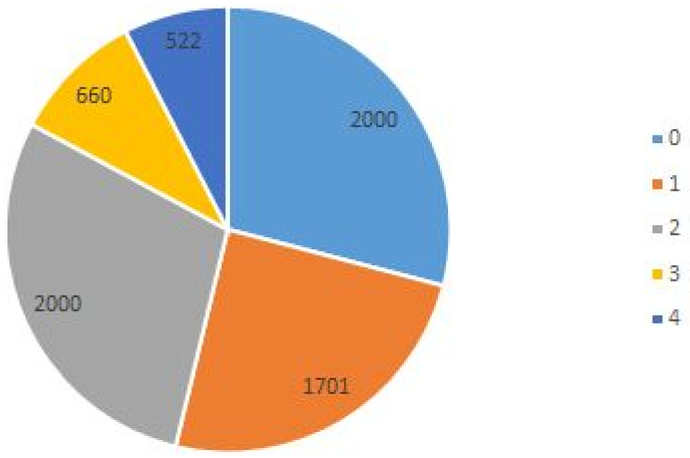


Fig6: Diabetic Retinopathy image distribution

Next step in our pre-processing was to flip all the right eye images horizontally so that there is uniformity between orientation of left and right eye.

The last step in this step was to resize the images to (150x150). Resize was mainly done due to constraint in the system memory which did not allow us to use higher resolution images for this network.

B. VGG16 Network:

The core layers of VGG16 is show in the Fig7a. And the modified model to suit our output/input dimensions are shown in Fig7b.

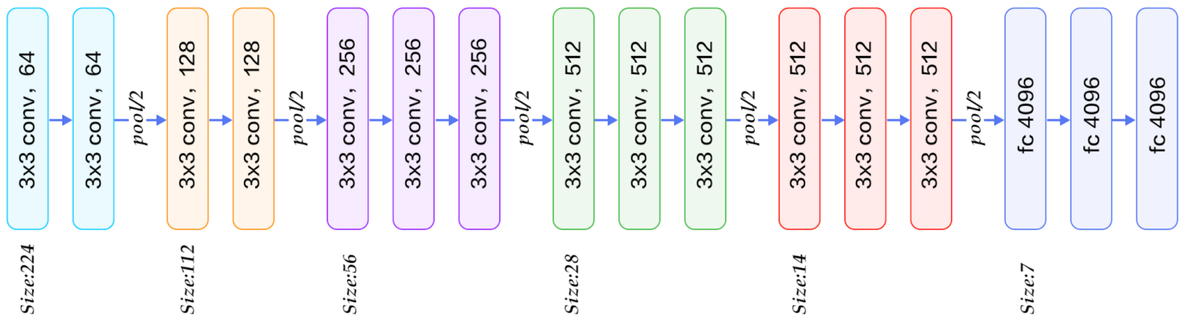


Fig7a: Diabetic Retinopathy image distribution

To accommodate the smaller size of our input, we added a input layer at the beginning of the network with input size 150x150, and output of size 224x224. And then to classify the images into 5 classes, we added 2 fully connected network at the end which produces the 5 outputs.

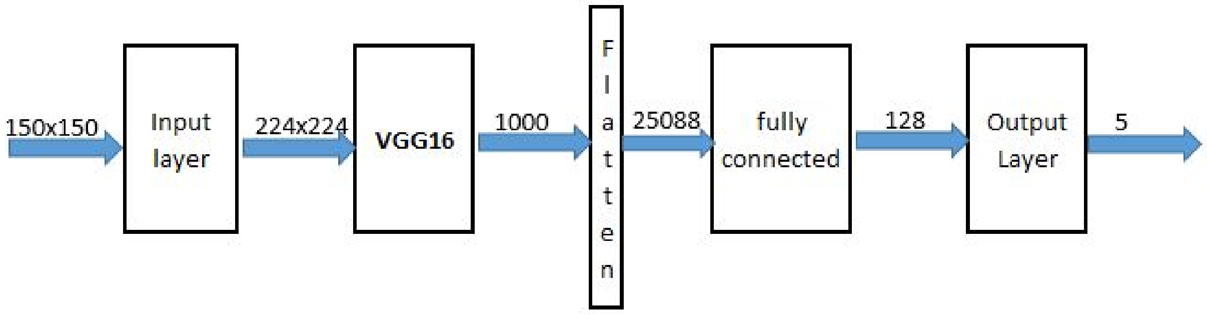
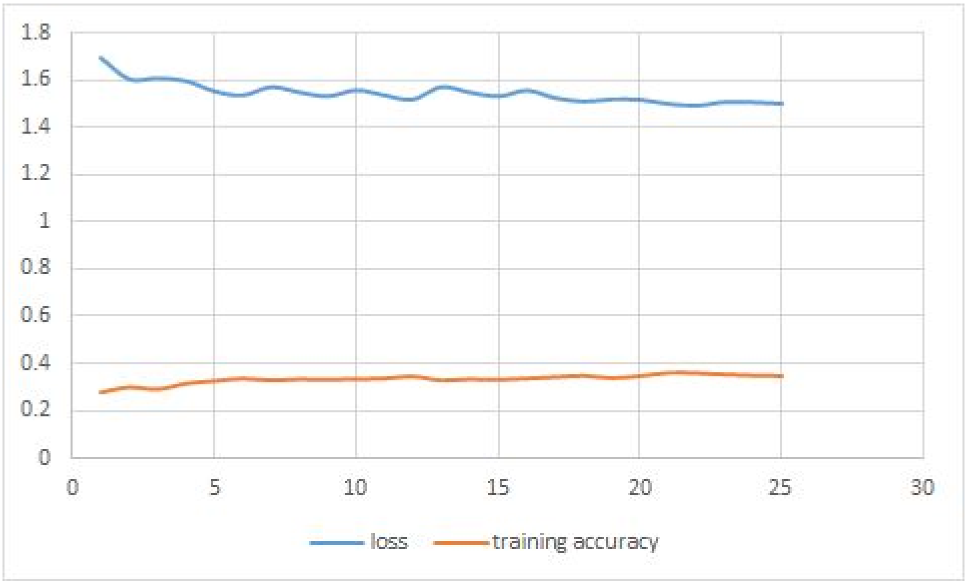


Fig7b: Diabetic Retinopathy image distribution

C. Training the VGG16 network:

The network was trained on 6883 images for about 25 epochs with batch size of 64 and learning rate of 0.01. However, due to time and resource constraint, we used the pre-trained weights of 'imagenet' dataset and finalised those weights (non trainable) and then continued with training the added network. Even with these restrictions, the training took 40 minutes per epoch.

Following Graph 1 shows the progress of the training.



Graph1: epoch vs loss/accuracy

D. Result for VGG16:

The VGG16 Network is trained on both original dataset and flipped dataset. The Network has a slightly better performance on flipped dataset i.e 33%, and a 27% of test accuracy for original dataset. The low accuracy could be associated to several attributes like, low image resolution, less number of epoch for training and all the parameters(weights) in the VGG16 was not trained for the given dataset. However this accuracies are low and there is a huge scope for improvement. The accuracies can be improved by adopting more data augmentation, training for more number of epoch and training the whole network.

II. Inception V3

A. Inception V3 Description

Inception V3[4] model is tuned and fitted to our dataset. The Inception architecture that has been shown to achieve very good performance at relatively low computational cost which is the deciding factor for selecting this model. Inception V3 is trained on 1000 everyday object categories (ImageNet ILSVRC). Kera and TensorFLow was chosen as the platform for creating our model. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. Keras helps us to focus on enabling fast experimentation.

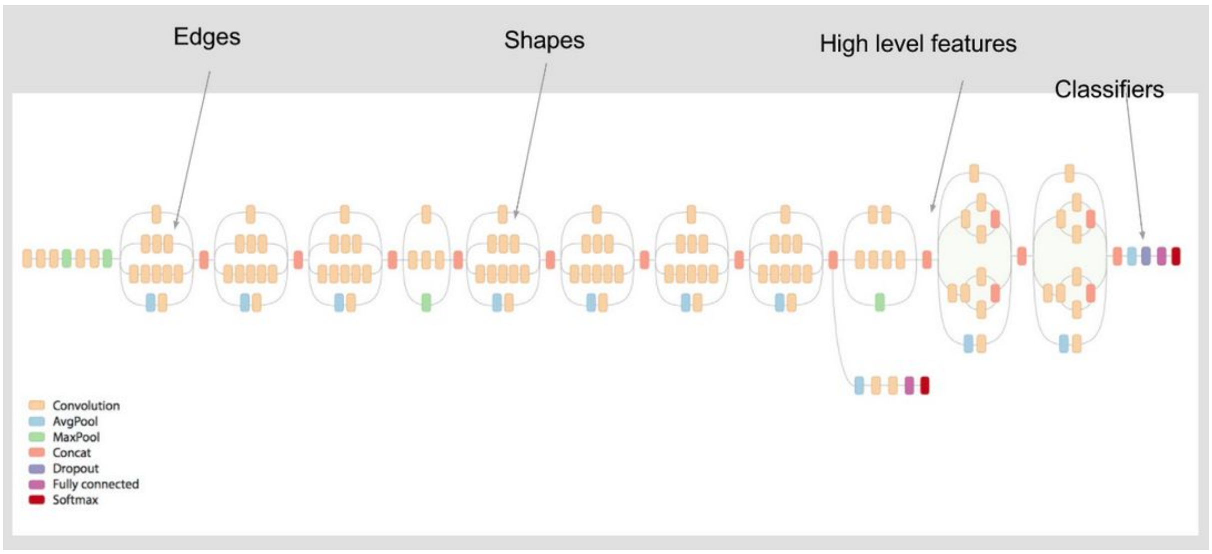


Fig: Inception Architecture

Using Transfer learning the Inception V3 model is trained on Diabetic Retinopathy Dataset[1]. The training are done on AWS g2.2xlarge[3] GPU for faster computation.

To better improve the model and to overcome overfitting due to the lack of images we tried to leverage Data Augmentation. We created multiple variation images of the same image with different image property. We hoped this would enhance the learnability of the model.

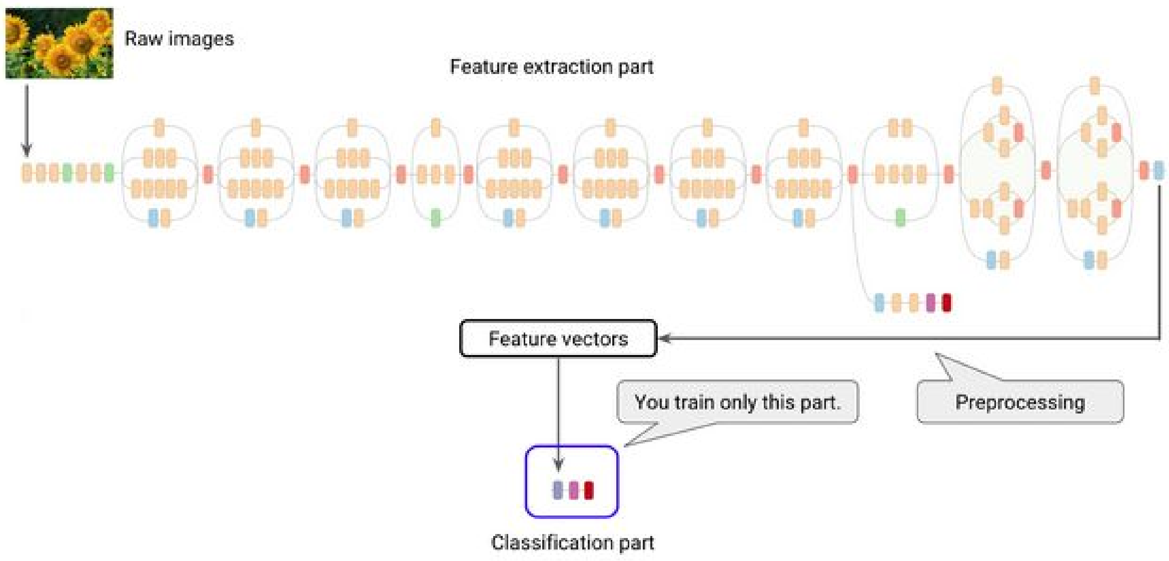
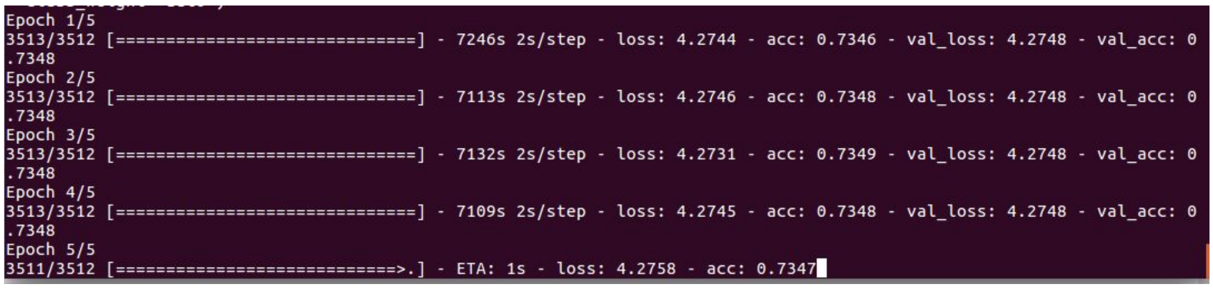


Fig: Transfer Learning on Inception V3

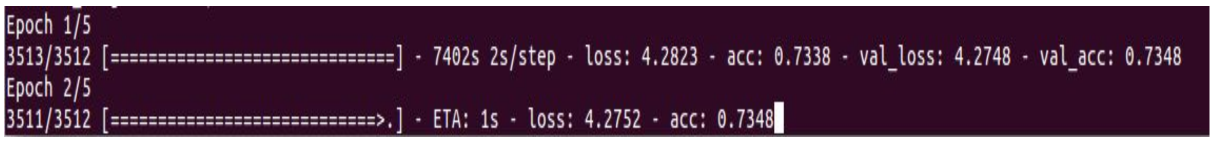
B. Training Inception Network

The inception network is trained on 2 dataset which consist of the whole images. The first dataset is the one which has all the images which are categories into respective classes. The second dataset is consist of the preprocessed images which has right eye images flipped horizontally. Due to the time constraint and limited GPU access we trained only for 5 epoch and hence we were not able to come to a perfect conclusion. The graphs are plotted for 5 epoch and if the training is done for more epoch then the accuracy and performance will increase.



```
Epoch 1/5
3513/3512 [==============================] - 7246s 2s/step - loss: 4.2744 - acc: 0.7346 - val_loss: 4.2748 - val_acc: 0
.7348
Epoch 2/5
3513/3512 [==============================] - 7113s 2s/step - loss: 4.2746 - acc: 0.7348 - val_loss: 4.2748 - val_acc: 0
.7348
Epoch 3/5
3513/3512 [==============================] - 7132s 2s/step - loss: 4.2731 - acc: 0.7349 - val_loss: 4.2748 - val_acc: 0
.7348
Epoch 4/5
3513/3512 [==============================] - 7109s 2s/step - loss: 4.2745 - acc: 0.7348 - val_loss: 4.2748 - val_acc: 0
.7348
Epoch 5/5
3511/3512 [============================>.] - ETA: 1s - loss: 4.2758 - acc: 0.7347
```

Fig: Training accuracy of Inception Network on Original dataset



```
Epoch 1/5
3513/3512 [==============================] - 7402s 2s/step - loss: 4.2823 - acc: 0.7338 - val_loss: 4.2748 - val_acc: 0.7348
Epoch 2/5
3511/3512 [============================>.] - ETA: 1s - loss: 4.2752 - acc: 0.7348
```

Fig: Training accuracy of Inception Network on flipped dataset.

B. Result of Inception V3

At the end of 5 epoch we obtained an training accuracy of 73.46% for inception v3 and validation accuracy of 73.48% when the original dataset is used. When the model was trained on the flipped dataset the test accuracy was 73.38% and the validation accuracy was at 73.48%.

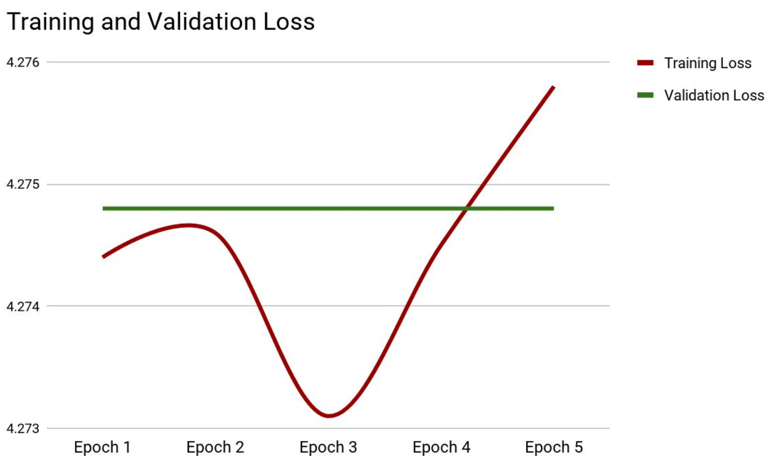


Fig: Training and Validation Loss

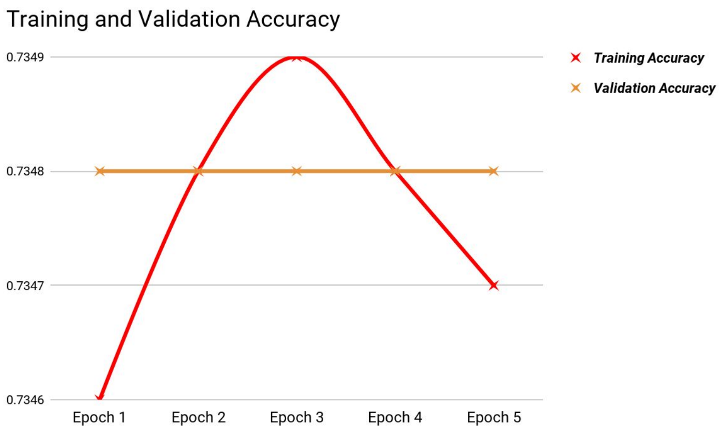


Fig: Training and Validation Accuracy

III. MobileNet

The next approach was to use the technique of transfer learning and hyperparameter tuning on MobileNet[3]. The principal difference between MobileNet and Inception model is that, Inception model is optimized for accuracy, while the MobileNet are optimized to be small and efficient, at the cost of some accuracy. MobileNet are smaller and efficient network which can run on mobile and smartphones.

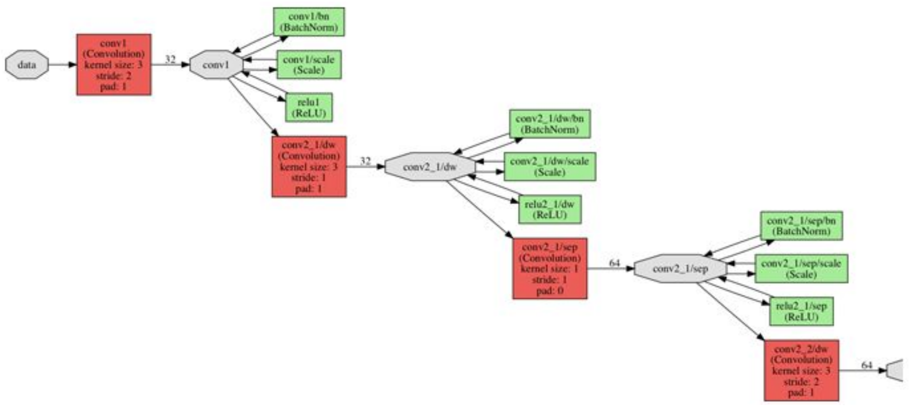
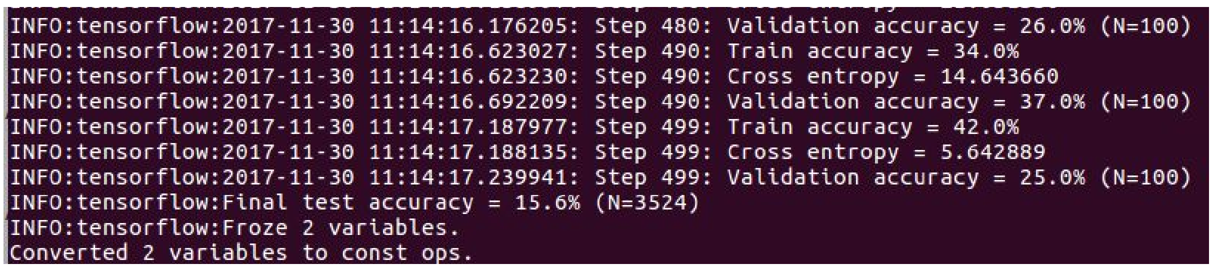

data
conv1 (Convolution) kernel size: 3 stride: 2 pad: 1
32
conv1
conv1/bn (BatchNorm)
conv1/scale (Scale)
relu1 (ReLU)
conv2_1/dw (Convolution) kernel size: 3 stride: 1 pad: 1
32
conv2_1/dw
conv2_1/dw/bn (BatchNorm)
conv2_1/dw/scale (Scale)
relu2_1/dw (ReLU)
conv2_1/sep (Convolution) kernel size: 1 stride: 1 pad: 0
64
conv2_1/sep
conv2_1/sep/bn (BatchNorm)
conv2_1/sep/scale (Scale)
relu2_1/sep (ReLU)
conv2_2/dw (Convolution) kernel size: 3 stride: 2 pad: 1
64

Fig: Sample MobileNet Architecture.

A full mobile network consist of 30 layers.

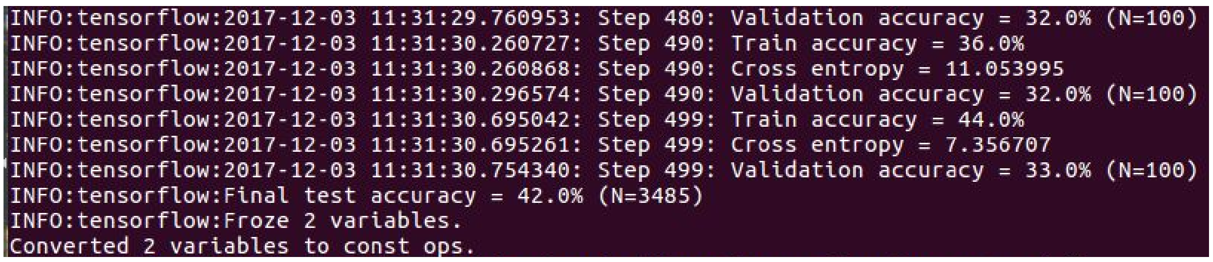
A. Training Mobile Network

MobileNet was trained for 500 epoch on GPU which took 1-2 hours. As the model was described it was faster and efficient but the accuracy was not as good as the other models we have tried for Diabetic Retinopathy.



```
INFO:tensorflow:2017-11-30 11:14:16.176205: Step 480: Validation accuracy = 26.0% (N=100)
INFO:tensorflow:2017-11-30 11:14:16.623027: Step 490: Train accuracy = 34.0%
INFO:tensorflow:2017-11-30 11:14:16.623230: Step 490: Cross entropy = 14.643660
INFO:tensorflow:2017-11-30 11:14:16.692209: Step 490: Validation accuracy = 37.0% (N=100)
INFO:tensorflow:2017-11-30 11:14:17.187977: Step 499: Train accuracy = 42.0%
INFO:tensorflow:2017-11-30 11:14:17.188135: Step 499: Cross entropy = 5.642889
INFO:tensorflow:2017-11-30 11:14:17.239941: Step 499: Validation accuracy = 25.0% (N=100)
INFO:tensorflow:Final test accuracy = 15.6% (N=3524)
INFO:tensorflow:Froze 2 variables.
Converted 2 variables to const ops.
```

Fig: Training accuracy for MobileNet on original dataset



```
INFO:tensorflow:2017-12-03 11:31:29.760953: Step 480: Validation accuracy = 32.0% (N=100)
INFO:tensorflow:2017-12-03 11:31:30.260727: Step 490: Train accuracy = 36.0%
INFO:tensorflow:2017-12-03 11:31:30.260868: Step 490: Cross entropy = 11.053995
INFO:tensorflow:2017-12-03 11:31:30.296574: Step 490: Validation accuracy = 32.0% (N=100)
INFO:tensorflow:2017-12-03 11:31:30.695042: Step 499: Train accuracy = 44.0%
INFO:tensorflow:2017-12-03 11:31:30.695261: Step 499: Cross entropy = 7.356707
INFO:tensorflow:2017-12-03 11:31:30.754340: Step 499: Validation accuracy = 33.0% (N=100)
INFO:tensorflow:Final test accuracy = 42.0% (N=3485)
INFO:tensorflow:Froze 2 variables.
Converted 2 variables to const ops.
```

Fig: Training accuracy for MobileNet on flipped dataset

B. Result of MobileNet

The Mobile Network is trained on both original dataset and flipped dataset. The Network has a slightly better performance on flipped dataset i.e 36%, and a 34% for original dataset. However this accuracies are low and there is a huge scope for improvement. The accuracies can be improved by adopting more data augmentation. Due to the very low training and validation accuracy the test accuracy are also low. We were able to obtain accuracies in the range 30% for flipped dataset and 15% for original dataset.

IV. Alex Net

A. About Alex Net

Alex Net is a convolutional neural network model proposed by Alex Krizhevsky. It is a standard Convolutional network model that has previously demonstrated good classification results with huge datasets like ImageNet.

The model consists of 5 convolutional layers each followed by max pooling and ReLu activation layers. The model also has 3 Fully connected Layers each followed by dropout layer for regularization. The softmax classifier is used to classify the image labels. The convolutional layers act as the "eyes" of the network by identifying and extracting useful feature maps and pass it to the fully connected layers for learning. The Alex Net architecture is shown in the figure below.

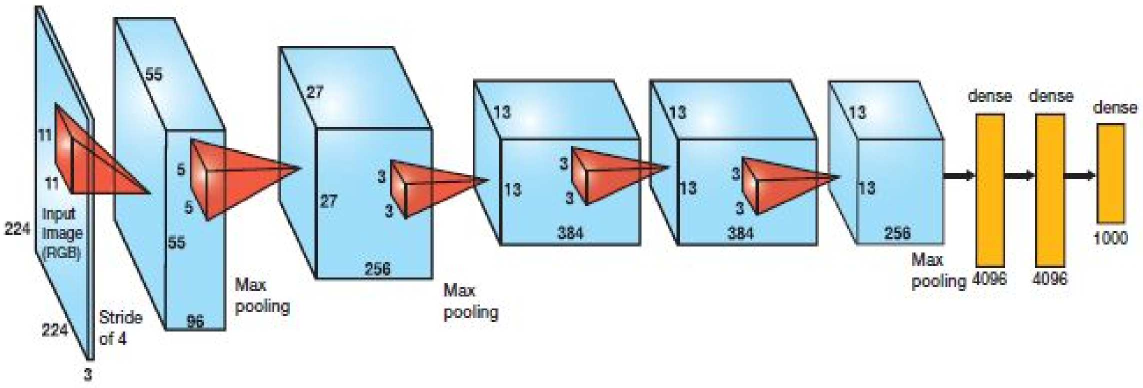


Fig: Alex Net Architecture

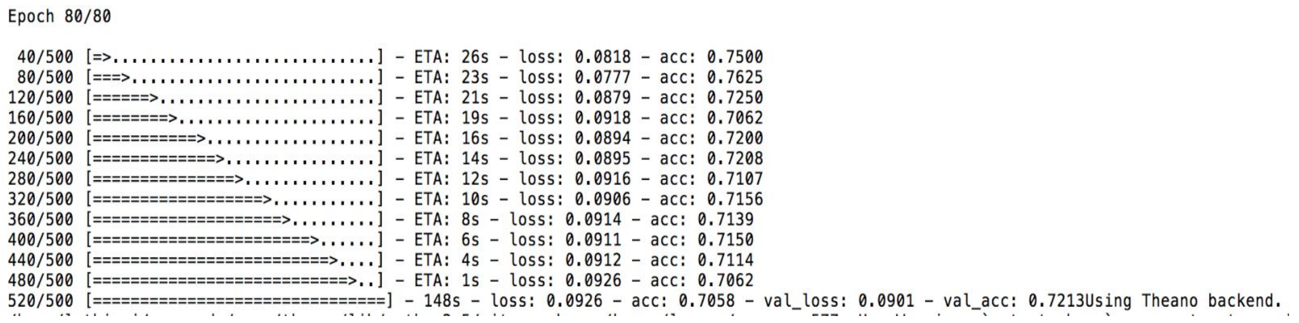
B. **Training Alex Net on Diabetic Retinopathy Dataset**

The Diabetics Retinopathy dataset has about 50,000 images and was downsized to 35,000 images. Alex Net was implemented using Keras framework with Theano backend. To get better performance, it was trained on the flipped image and real time data augmentation was performed using Keras Image Generator.

For the training purposes, SGD was used with 80 epochs and batch size 40 with learning rate 0.01.The model was trained from scratch without using any pre trained weights. There could be a slight increase in performance if the model was fine tuned on pre trained weights.

C. Alex Net Results

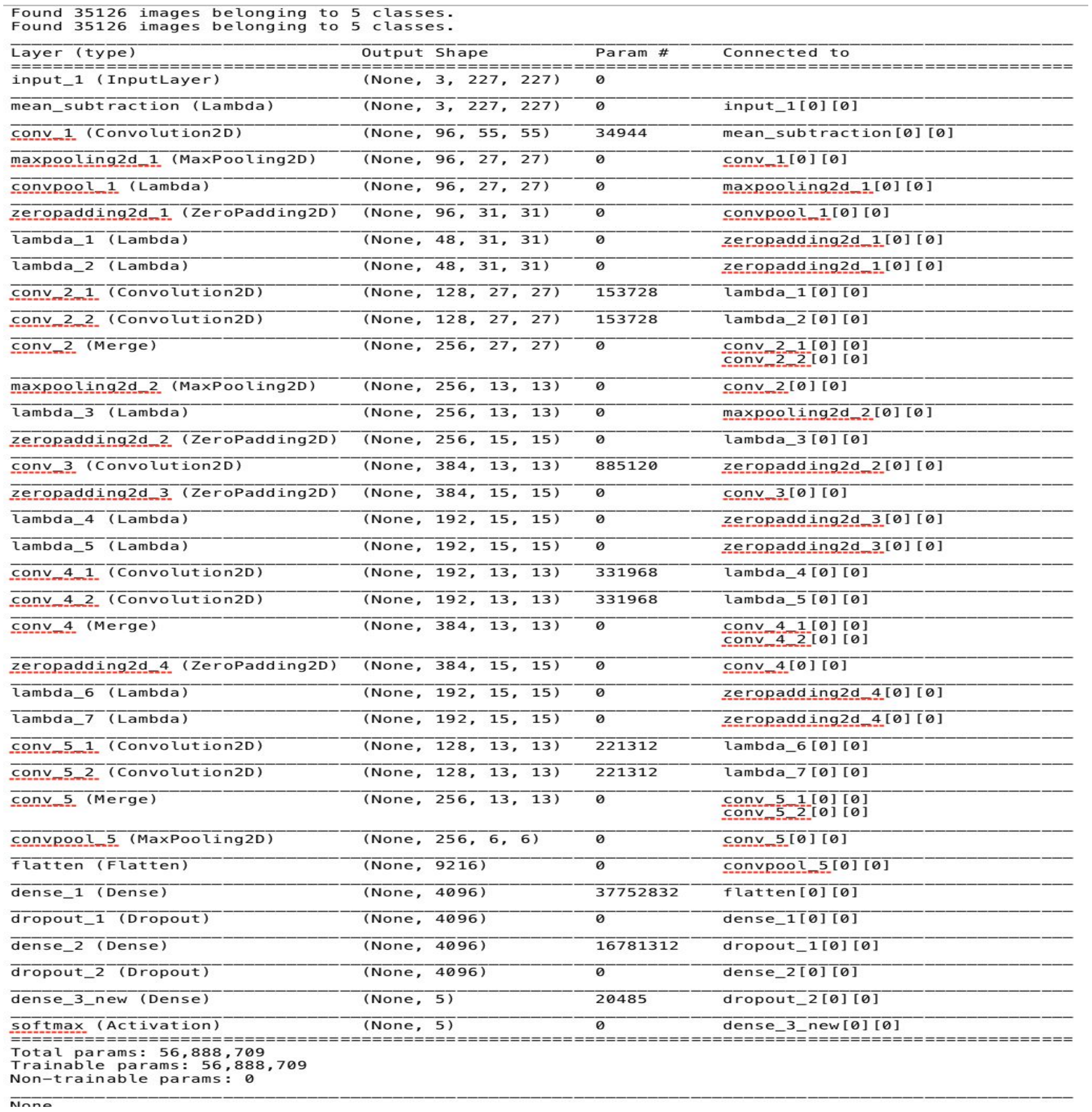
The training accuracy was around ~70% and validation accuracy was around ~72%.



```
Epoch 80/80

 40/500 [=>............................] - ETA: 26s - loss: 0.0818 - acc: 0.7500
 80/500 [===>..........................] - ETA: 23s - loss: 0.0777 - acc: 0.7625
120/500 [======>.......................] - ETA: 21s - loss: 0.0879 - acc: 0.7250
160/500 [========>.....................] - ETA: 19s - loss: 0.0918 - acc: 0.7062
200/500 [===========>..................] - ETA: 16s - loss: 0.0894 - acc: 0.7200
240/500 [=============>................] - ETA: 14s - loss: 0.0895 - acc: 0.7208
280/500 [===============>..............] - ETA: 12s - loss: 0.0916 - acc: 0.7107
320/500 [==================>...........] - ETA: 10s - loss: 0.0906 - acc: 0.7156
360/500 [====================>.........] - ETA: 8s - loss: 0.0914 - acc: 0.7139
400/500 [=======================>......] - ETA: 6s - loss: 0.0911 - acc: 0.7150
440/500 [=========================>....] - ETA: 4s - loss: 0.0912 - acc: 0.7114
480/500 [===========================>..] - ETA: 1s - loss: 0.0926 - acc: 0.7062
520/500 [==============================] - 148s - loss: 0.0926 - acc: 0.7058 - val_loss: 0.0901 - val_acc: 0.7213Using Theano backend.
```

Fig: Training accuracy for Alex Net on flipped dataset



```
Found 35126 images belonging to 5 classes.
Found 35126 images belonging to 5 classes.
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 3, 227, 227)   0
____________________________________________________________________________________________________
mean_subtraction (Lambda)        (None, 3, 227, 227)   0           input_1[0][0]
____________________________________________________________________________________________________
conv_1 (Convolution2D)           (None, 96, 55, 55)    34944       mean_subtraction[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 96, 27, 27)    0           conv_1[0][0]
____________________________________________________________________________________________________
convpool_1 (Lambda)              (None, 96, 27, 27)    0           maxpooling2d_1[0][0]
____________________________________________________________________________________________________
zeropadding2d_1 (ZeroPadding2D)  (None, 96, 31, 31)    0           convpool_1[0][0]
____________________________________________________________________________________________________
lambda_1 (Lambda)                (None, 48, 31, 31)    0           zeropadding2d_1[0][0]
____________________________________________________________________________________________________
lambda_2 (Lambda)                (None, 48, 31, 31)    0           zeropadding2d_1[0][0]
____________________________________________________________________________________________________
conv_2_1 (Convolution2D)         (None, 128, 27, 27)   153728      lambda_1[0][0]
____________________________________________________________________________________________________
conv_2_2 (Convolution2D)         (None, 128, 27, 27)   153728      lambda_2[0][0]
____________________________________________________________________________________________________
conv_2 (Merge)                   (None, 256, 27, 27)   0           conv_2_1[0][0]
                                                                   conv_2_2[0][0]
____________________________________________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 256, 13, 13)   0           conv_2[0][0]
____________________________________________________________________________________________________
lambda_3 (Lambda)                (None, 256, 13, 13)   0           maxpooling2d_2[0][0]
____________________________________________________________________________________________________
zeropadding2d_2 (ZeroPadding2D)  (None, 256, 15, 15)   0           lambda_3[0][0]
____________________________________________________________________________________________________
conv_3 (Convolution2D)           (None, 384, 13, 13)   885120      zeropadding2d_2[0][0]
____________________________________________________________________________________________________
zeropadding2d_3 (ZeroPadding2D)  (None, 384, 15, 15)   0           conv_3[0][0]
____________________________________________________________________________________________________
lambda_4 (Lambda)                (None, 192, 15, 15)   0           zeropadding2d_3[0][0]
____________________________________________________________________________________________________
lambda_5 (Lambda)                (None, 192, 15, 15)   0           zeropadding2d_3[0][0]
____________________________________________________________________________________________________
conv_4_1 (Convolution2D)         (None, 192, 13, 13)   331968      lambda_4[0][0]
____________________________________________________________________________________________________
conv_4_2 (Convolution2D)         (None, 192, 13, 13)   331968      lambda_5[0][0]
____________________________________________________________________________________________________
conv_4 (Merge)                   (None, 384, 13, 13)   0           conv_4_1[0][0]
                                                                   conv_4_2[0][0]
____________________________________________________________________________________________________
zeropadding2d_4 (ZeroPadding2D)  (None, 384, 15, 15)   0           conv_4[0][0]
____________________________________________________________________________________________________
lambda_6 (Lambda)                (None, 192, 15, 15)   0           zeropadding2d_4[0][0]
____________________________________________________________________________________________________
lambda_7 (Lambda)                (None, 192, 15, 15)   0           zeropadding2d_4[0][0]
____________________________________________________________________________________________________
conv_5_1 (Convolution2D)         (None, 128, 13, 13)   221312      lambda_6[0][0]
____________________________________________________________________________________________________
conv_5_2 (Convolution2D)         (None, 128, 13, 13)   221312      lambda_7[0][0]
____________________________________________________________________________________________________
conv_5 (Merge)                   (None, 256, 13, 13)   0           conv_5_1[0][0]
                                                                   conv_5_2[0][0]
____________________________________________________________________________________________________
convpool_5 (MaxPooling2D)        (None, 256, 6, 6)     0           conv_5[0][0]
____________________________________________________________________________________________________
flatten (Flatten)                (None, 9216)          0           convpool_5[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 4096)          37752832    flatten[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 4096)          0           dense_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 4096)          16781312    dropout_1[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 4096)          0           dense_2[0][0]
____________________________________________________________________________________________________
dense_3_new (Dense)              (None, 5)             20485       dropout_2[0][0]
____________________________________________________________________________________________________
softmax (Activation)             (None, 5)             0           dense_3_new[0][0]
====================================================================================================
Total params: 56,888,709
Trainable params: 56,888,709
Non-trainable params: 0
____________________________________________________________________________________________________
```

Fig: Alex Net Model on Keras

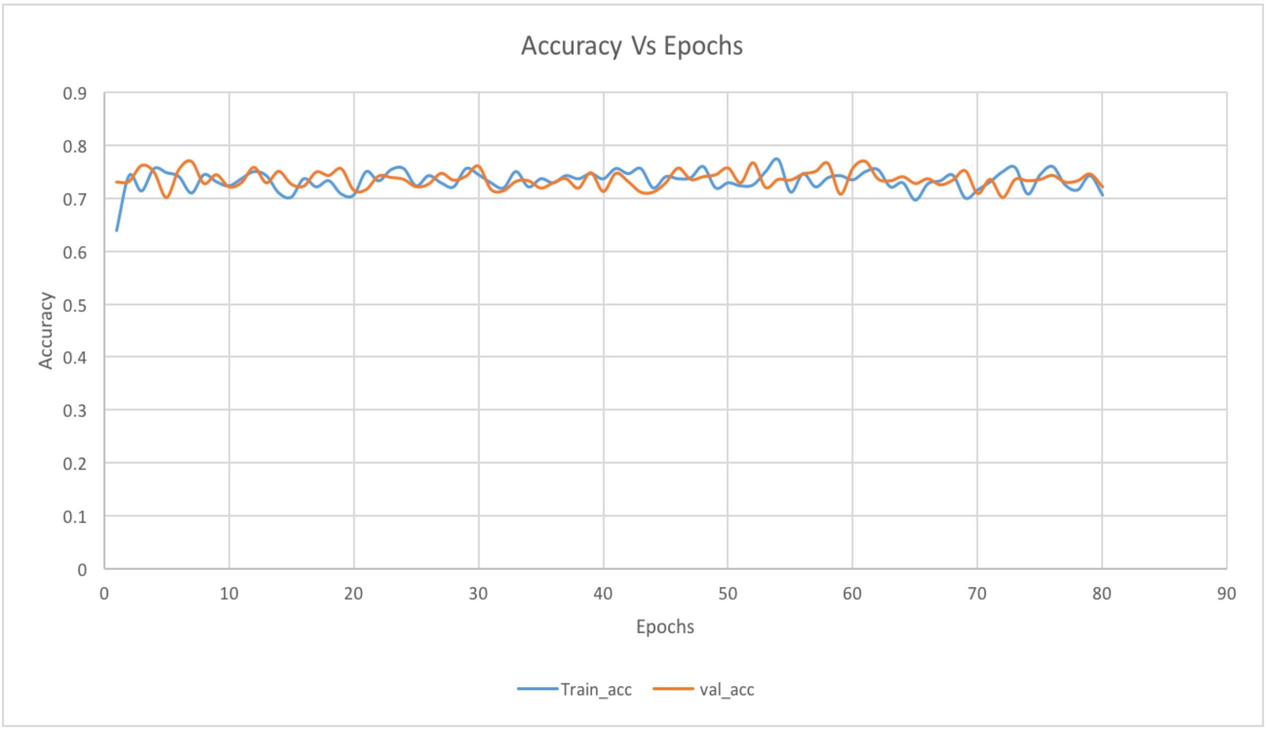


Fig: Training and Validation Accuracy

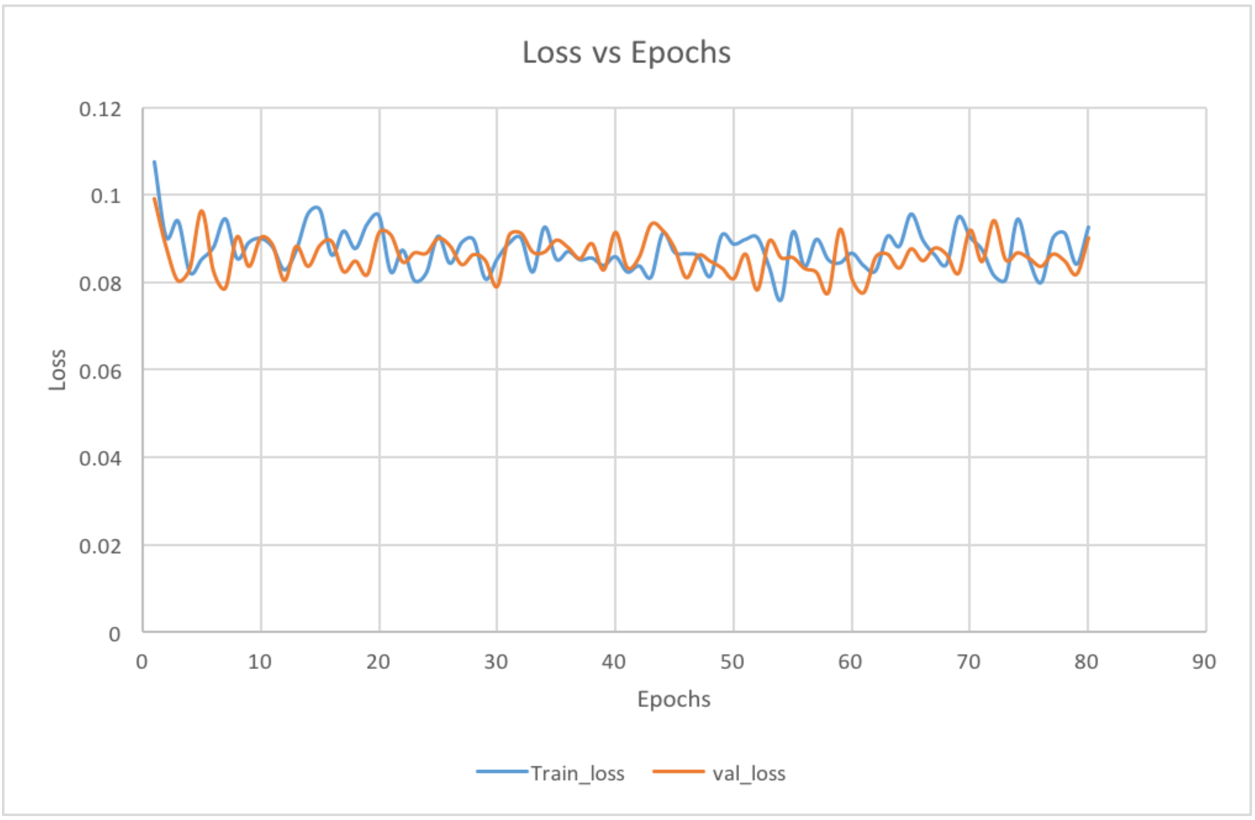


Fig: Training and Validation Loss

5. PERFORMANCE MEASURE AND RESULTS

Model	**Accuracy**
AlexNet	**For Flipped Dataset** Training Accuracy: 70.58% Validation Accuracy : 72.13%
VGG Net	**For Original Dataset** Training Accuracy: 32% Validation Accuracy: 27% **For Flipped Dataset** Training Accuracy: 35.61% Validation Accuracy: 33%
Inception V4	**For Original Dataset** Training Accuracy: 73% Validation Accuracy: 72% **For Flipped Dataset** Training Accuracy: 72% Validation Accuracy: 73%
MobileNet	**For Original Dataset** Training Accuracy: 34% Validation Accuracy: 25% **For Flipped Dataset** Training Accuracy: 36% Validation Accuracy: 33%

6. CONCLUSION AND SCOPE FOR IMPROVEMENT

We see a lot of scope for improvement, like data augmentation will help in increasing the accuracy, and dataset normalization could be done in more reformed process to get the best dataset to train with.

Training the whole network instead of training only few layers will help us achieve more accuracy.

With better resources, we could train the image with higher resolution images which will help in identifying the minute differences between the classes.

Due to time and resource constraint, we were unable to train three other network we had planned to do (googleNet, AlexNet, and ResNet). This would be taken up in the future and compare the result of all the network.

7. REFERENCE

[1] https://www.kaggle.com/c/diabetic-retinopathy-detection#description
[2] https://arxiv.org/pdf/1703.10757.pdf
[3] https://arxiv.org/abs/1704.04861
[4] https://arxiv.org/abs/1512.00567
[5] https://arxiv.org/abs/1704.04861
[6] https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#3
[7] https://deeplearningsandbox.com/
[8] https://sushscience.wordpress.com/2016/12/04/understanding-alexnet/
[9] https://github.com/duggalrahul/AlexNet-Experiments-Keras
[10]https://rahulduggal2608.wordpress.com/2017/04/02/alexnet-in-keras/
[11]A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In NIPS, 2012.
[12] http://book.paddlepaddle.org/03.image_classification/