Practical Aspects of Data Science

Report on Diabetic Retinopathy Detection using Deep Learning Convolution Networks

Project I

Members:

Swapnil Bansal (SXB180020)
Paresh Dashore (PXD190004)
Meghna Kurup (MXK180015)

Preface:

As estimated by the World Health Organization around 347 million people worldwide and around 29.1 million people in the US have diabetes. Around 40% to 45% of Americans with diabetes have some stage of Diabetic Retinopathy which is an eye disease. If detected on time, vision impairment can be avoided and effective treatment can be provided. Effective treatments for DR require early diagnosis and continuous monitoring of diabetic patients, but this is a demanding task as the disease shows few symptoms until it is too late to provide treatment.

The detection process is manual and time-consuming and requires clinicians to evaluate the photographs of the retina. This time-consuming process can lead to delayed results, miscommunication and delayed treatments. Lesions associated with vascular abnormalities in how clinicians identify the presence of DR and lack of resources, growth in the number of diabetic patients have increased demands in the infrastructure needed. A requirement of an automated process fan automated system for DR detection of color fundus images could have a huge impact on making timely treatment accessible to more patients. Towards this end, we have explored Convolution Neural Networks to automatically detect the severity of DR using information from retina images in this work.

Classes of Diabetic Retinopathy:

1. Class 1 - Mild Diabetic Retinopathy
2. Class 2 - Moderate DR
3. Class 3 - Severe DR
4. Class 4 - Proliferative Diabetic Retinopathy
5. Class 0 - Healthy eye

Dataset :

We have used the dataset provided by Kaggle, which are high-resolution images of the eye and we have tried to identify defects in the eye using each individual pixel of the image in order to classify the image. Left and right field is provided for every subject. The images are labeled with a subject id followed by _left or _right. (For example, if the subject id is 23 then there are two images for the subject 23_left and 23_right.)

As mentioned above, a clinician has rated the presence of diabetic retinopathy in each image on a scale of 0 to 4, according to the following scale:

0 - No DR

1 - Mild

2 - Moderate

3 - Severe

4 - Proliferative DR

The images are taken from different types of cameras and models. Because of this, the visual appearance of left versus right may be different. Some images of the retina are the anatomical appearance that is, macula on the left, optic nerve on the right for the right eye. Other images are the images as one would see through a microscope condensing lens which is inverted, like that in a typical live eye exam.

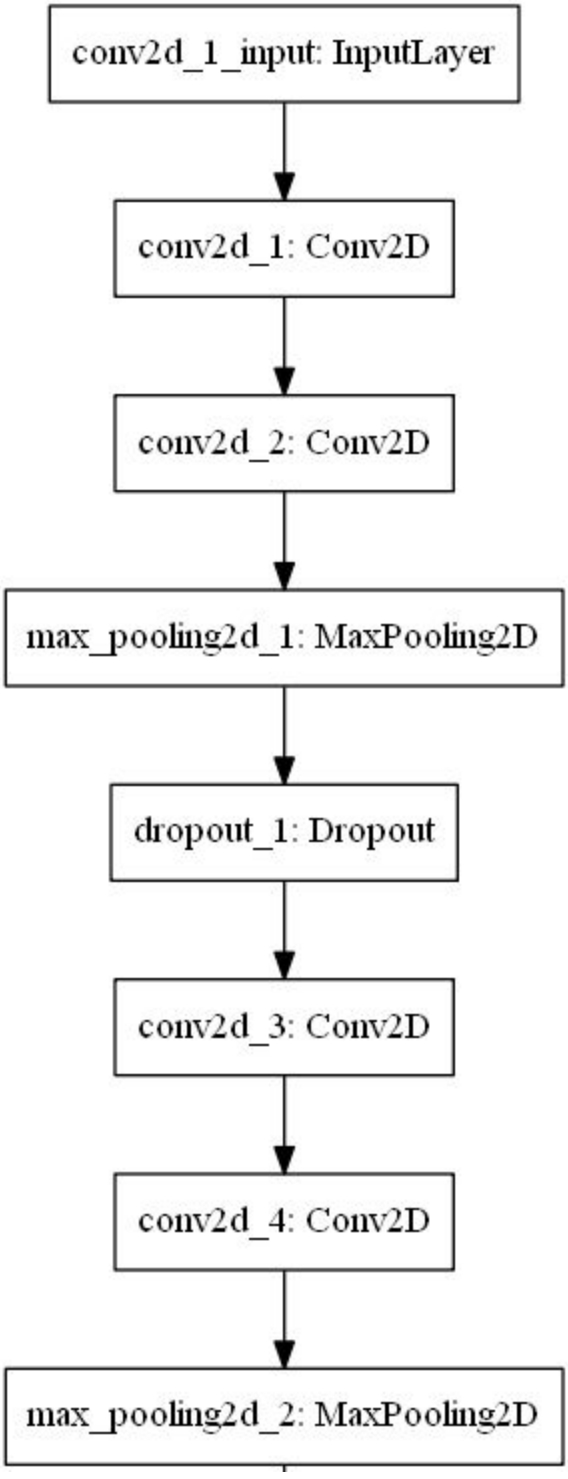
Data Preprocessing:

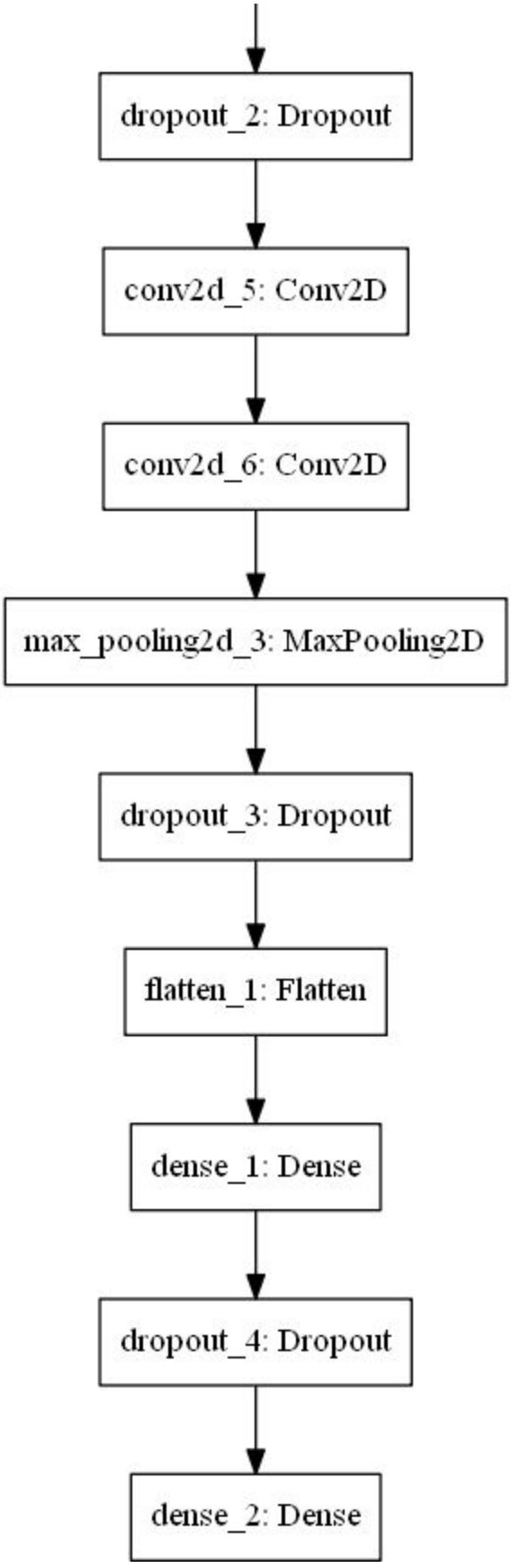
While reading data, for each image we check the dimensions of the image. We want all the images to be of equal sizes. The images are resized to a given pixel size so that all images are of the same size. After resizing if they don't match the desired size we throw an error. This preprocessing is done to help have uniformity in all images so that it does not affect the training of data. As mentioned before, each image is named 'subjectId_(left/right)'. So we map the image name to it's pixels which are used for processing. This mapped array is used to only extract the patient id and the pixels for both the left and right eye images.

We also need to match the patient id from the csv file and images. We only keep images that are unique. The data has been divided into 75% for training and 25% for validation testing. The image is augmented so that it can be identified from all possible angles.

Model used for training:

For creating a training model we use Deep Convolutional Neural networks. Given below is the plot of the model which states the steps involved in creating the Convolutional Neural Network.


conv2d_1_input: InputLayer
conv2d_1: Conv2D
conv2d_2: Conv2D
max_pooling2d_1: MaxPooling2D
dropout_1: Dropout
conv2d_3: Conv2D
conv2d_4: Conv2D
max_pooling2d_2: MaxPooling2D


dropout_2: Dropout
conv2d_5: Conv2D
conv2d_6: Conv2D
max_pooling2d_3: MaxPooling2D
dropout_3: Dropout
flatten_1: Flatten
dense_1: Dense
dropout_4: Dropout
dense_2: Dense

The basic steps for CNN are Convolution of the image then MaxPooling then Convolution and Max Pooling again. The mentioned steps can be repeated any number of times as required and the final layer is the dense flatten layer. In our model, the first step is repeated thrice after which it is flattened. The activation function used for all the layers was rectified linear unit or ReLu. And for compiling the model, we have used categorical cross entropy for the loss.

Below is the summary of our model which describes each layer of the model and the parameters of the model. As we can see the input image size (resized to 128*128).

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 32)      896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 126, 126, 32)      9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 63, 63, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 61, 61, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 30, 30, 64)        36928
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               6423040
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 2565
=================================================================
Total params: 6,565,029
Trainable params: 6,565,029
Non-trainable params: 0
```

Training:

Since the amount of data is huge, we kept a batch size of 32 and have trained it on 150 Epochs. Given below are snippets of the training and the epochs with the accuracy values.

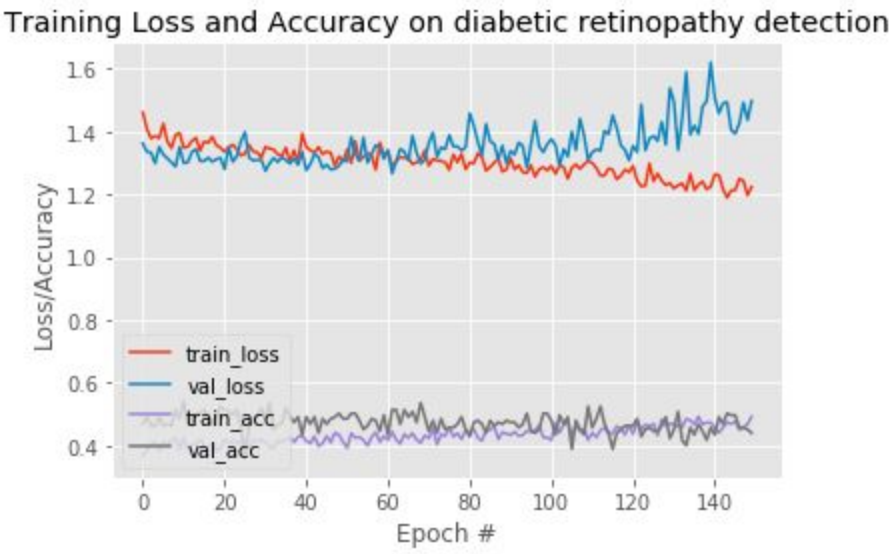
```
training network...
Epoch 1/150
22/22 [==============================] - 72s 3s/step - loss: 1.4609 - acc
uracy: 0.3695 - val_loss: 1.3614 - val_accuracy: 0.4689
Epoch 2/150
22/22 [==============================] - 78s 4s/step - loss: 1.4073 - acc
uracy: 0.3878 - val_loss: 1.3351 - val_accuracy: 0.4915
Epoch 3/150
22/22 [==============================] - 65s 3s/step - loss: 1.3774 - acc
uracy: 0.4045 - val_loss: 1.3323 - val_accuracy: 0.4633
Epoch 4/150
22/22 [==============================] - 72s 3s/step - loss: 1.3900 - acc
uracy: 0.4120 - val_loss: 1.2986 - val_accuracy: 0.4633
Epoch 5/150
22/22 [==============================] - 81s 4s/step - loss: 1.3739 - acc
uracy: 0.3827 - val_loss: 1.3506 - val_accuracy: 0.4915
Epoch 6/150
22/22 [==============================] - 73s 3s/step - loss: 1.4329 - acc
uracy: 0.4135 - val_loss: 1.3265 - val_accuracy: 0.4633
Epoch 7/150
22/22 [==============================] - 73s 3s/step - loss: 1.3608 - acc
uracy: 0.4003 - val_loss: 1.3140 - val_accuracy: 0.4689
Epoch 8/150
22/22 [==============================] - 79s 4s/step - loss: 1.3386 - acc
uracy: 0.4179 - val_loss: 1.3014 - val_accuracy: 0.4633
Epoch 9/150
22/22 [==============================] - 74s 3s/step - loss: 1.3807 - acc
uracy: 0.4238 - val_loss: 1.2878 - val_accuracy: 0.5141
Epoch 10/150
22/22 [==============================] - 68s 3s/step - loss: 1.4054 - acc
uracy: 0.3886 - val_loss: 1.3501 - val_accuracy: 0.4859
Epoch 11/150
22/22 [==============================] - 68s 3s/step - loss: 1.3481 - acc
uracy: 0.4247 - val_loss: 1.2991 - val_accuracy: 0.5424
Epoch 12/150
22/22 [==============================] - 68s 3s/step - loss: 1.3511 - acc
uracy: 0.3856 - val_loss: 1.3015 - val_accuracy: 0.4689
Epoch 13/150
22/22 [==============================] - 106s 5s/step - loss: 1.3773 - ac
curacy: 0.4030 - val_loss: 1.3345 - val_accuracy: 0.4633
```

```
Epoch 138/150
22/22 [==============================] - 59s 3s/step - loss: 1.2376 - acc
uracy: 0.4677 - val_loss: 1.4789 - val_accuracy: 0.4463
Epoch 139/150
22/22 [==============================] - 59s 3s/step - loss: 1.2195 - acc
uracy: 0.4736 - val_loss: 1.4956 - val_accuracy: 0.4181
Epoch 140/150
22/22 [==============================] - 58s 3s/step - loss: 1.2301 - acc
uracy: 0.4736 - val_loss: 1.6192 - val_accuracy: 0.4576
Epoch 141/150
22/22 [==============================] - 59s 3s/step - loss: 1.2611 - acc
uracy: 0.4589 - val_loss: 1.5061 - val_accuracy: 0.4350
Epoch 142/150
22/22 [==============================] - 57s 3s/step - loss: 1.2538 - acc
uracy: 0.4413 - val_loss: 1.4575 - val_accuracy: 0.4915
Epoch 143/150
22/22 [==============================] - 58s 3s/step - loss: 1.2188 - acc
uracy: 0.4575 - val_loss: 1.4889 - val_accuracy: 0.4689
Epoch 144/150
22/22 [==============================] - 58s 3s/step - loss: 1.1905 - acc
uracy: 0.4751 - val_loss: 1.4931 - val_accuracy: 0.5028
Epoch 145/150
22/22 [==============================] - 58s 3s/step - loss: 1.2050 - acc
uracy: 0.4692 - val_loss: 1.4043 - val_accuracy: 0.4972
Epoch 146/150
22/22 [==============================] - 60s 3s/step - loss: 1.2134 - acc
uracy: 0.4773 - val_loss: 1.3926 - val_accuracy: 0.4972
Epoch 147/150
22/22 [==============================] - 58s 3s/step - loss: 1.2492 - acc
uracy: 0.4501 - val_loss: 1.4263 - val_accuracy: 0.4576
Epoch 148/150
22/22 [==============================] - 56s 3s/step - loss: 1.2314 - acc
uracy: 0.4591 - val_loss: 1.4918 - val_accuracy: 0.4576
Epoch 149/150
22/22 [==============================] - 61s 3s/step - loss: 1.1970 - acc
uracy: 0.4688 - val_loss: 1.4358 - val_accuracy: 0.4520
Epoch 150/150
22/22 [==============================] - 56s 3s/step - loss: 1.2190 - acc
uracy: 0.4924 - val_loss: 1.4971 - val_accuracy: 0.4407
```

We have provided the snippets of the first thirteen epochs and the last twelve epochs for an idea of the running. On the training data we develop an accuracy of 49.24%

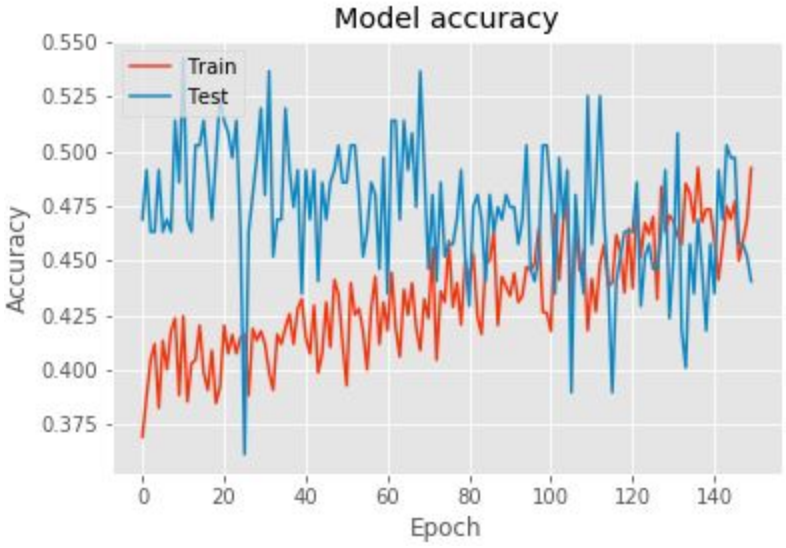
Generating accuracy plots

```
Generating plots...
```



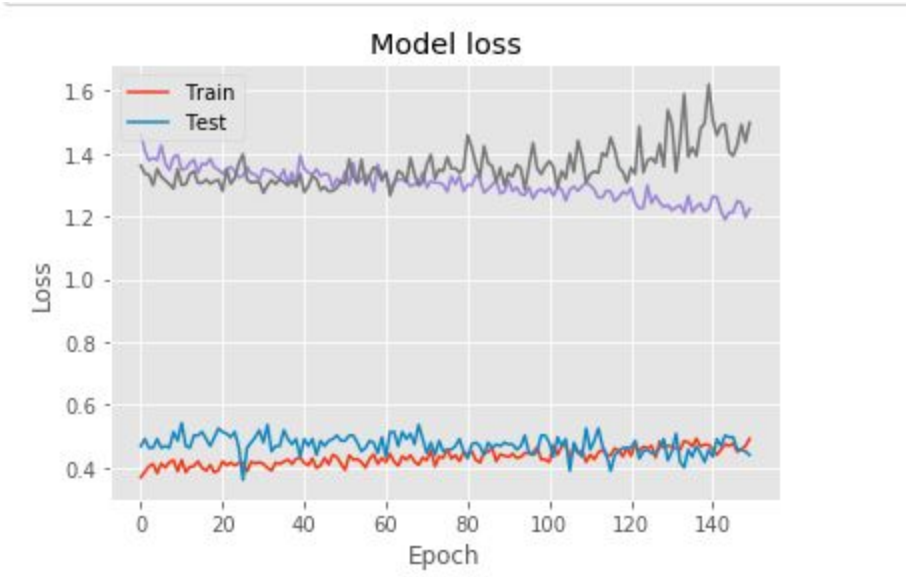
This is the plot of the training loss vs validation loss and training vs testing accuracy.

Around 60 epochs the model gave the best results as the training accuracy equals the validation accuracy as seen above.

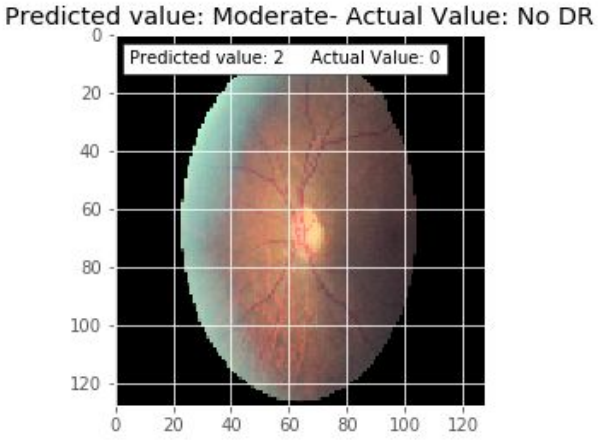


Model accuracy

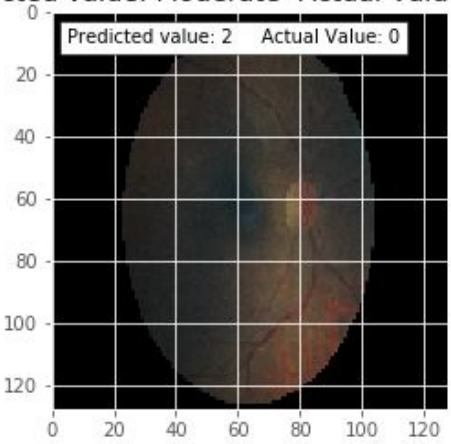
Plot for model loss of the test and train data:



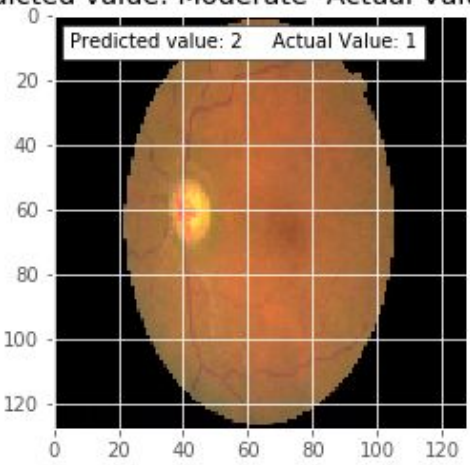
The predicted value and the actual value of at least 25 images (30 included) are given below:



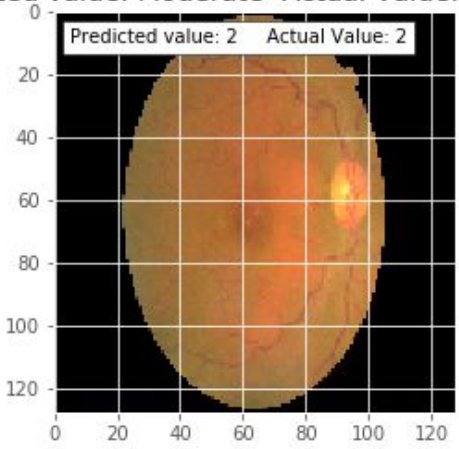
Predicted value: Moderate- Actual Value: No DR



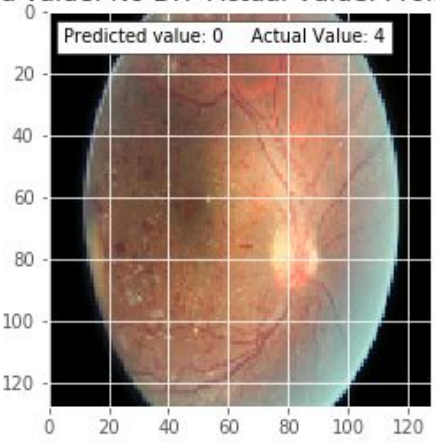
Predicted value: Moderate- Actual Value: Mild



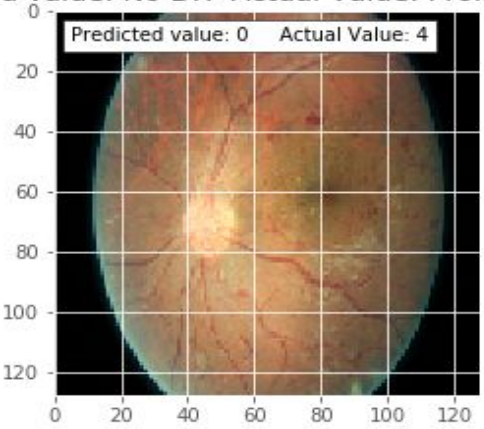
Predicted value: Moderate- Actual Value: Moderate



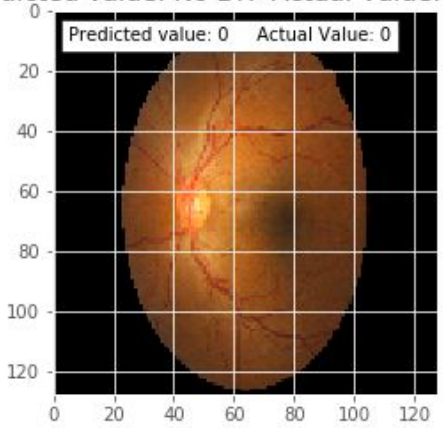
Predicted value: No DR- Actual Value: Proliferative DR



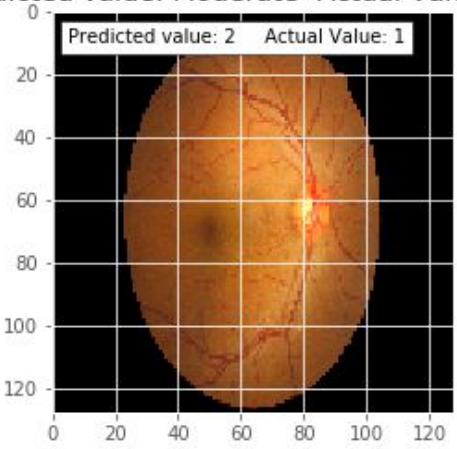
Predicted value: No DR- Actual Value: Proliferative DR



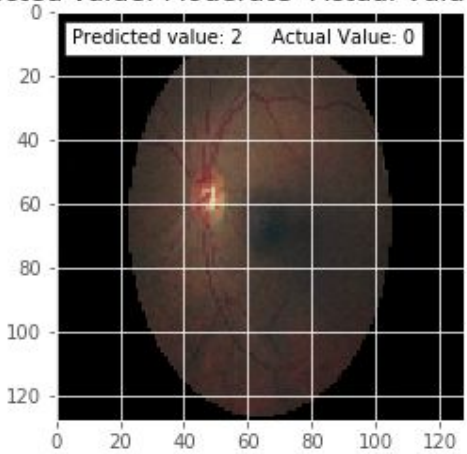
Predicted value: No DR- Actual Value: No DR



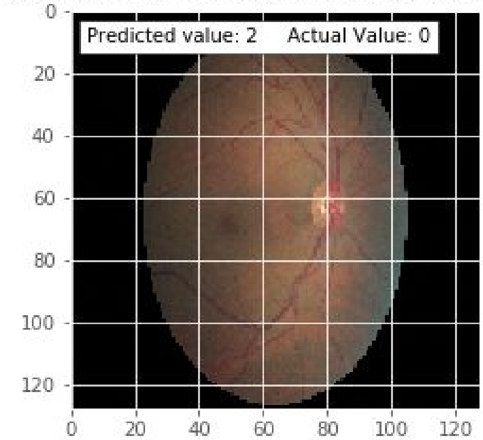
Predicted value: Moderate- Actual Value: Mild



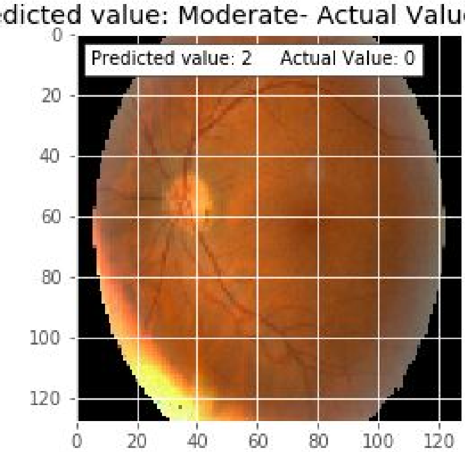
Predicted value: Moderate- Actual Value: No DR



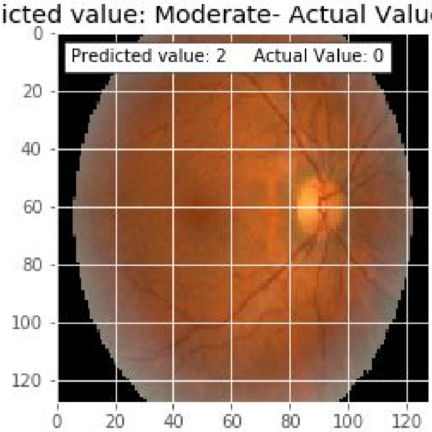
Predicted value: Moderate- Actual Value: No DR



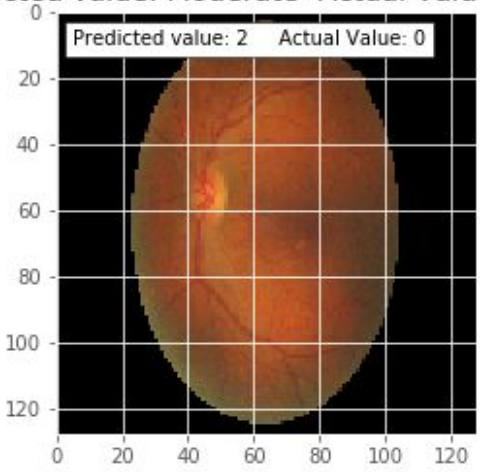
Predicted value: Moderate- Actual Value: No DR



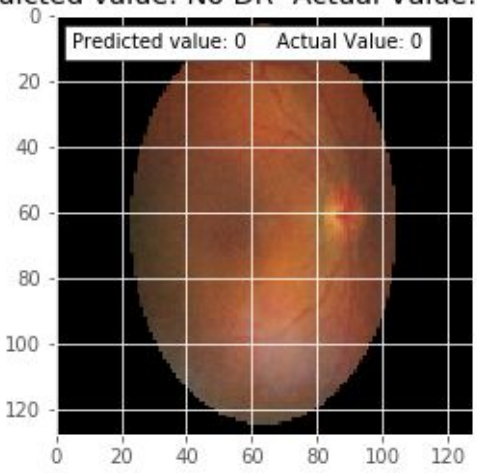
Predicted value: Moderate- Actual Value: No DR



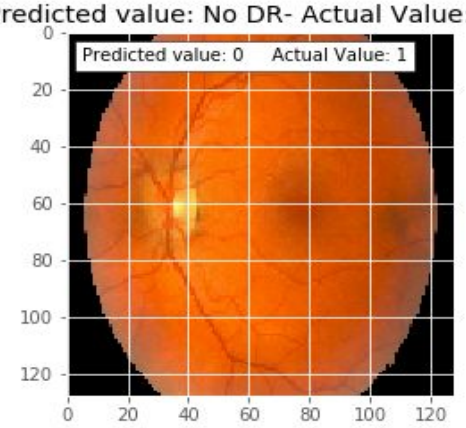
Predicted value: Moderate- Actual Value: No DR



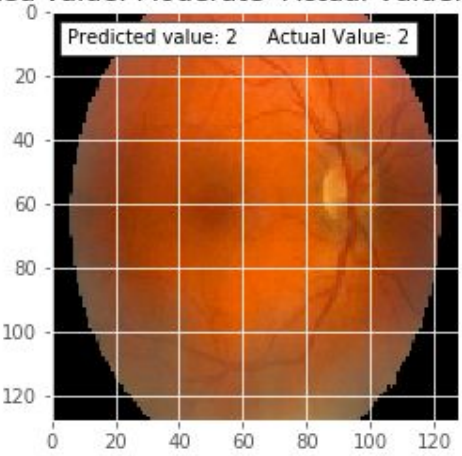
Predicted value: No DR- Actual Value: No DR



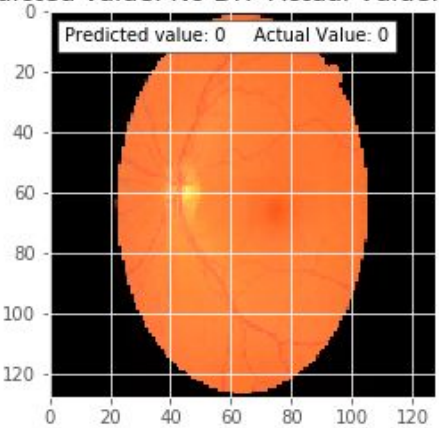
Predicted value: No DR- Actual Value: Mild



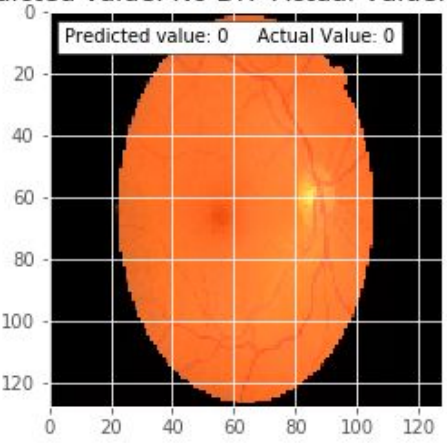
Predicted value: Moderate- Actual Value: Moderate



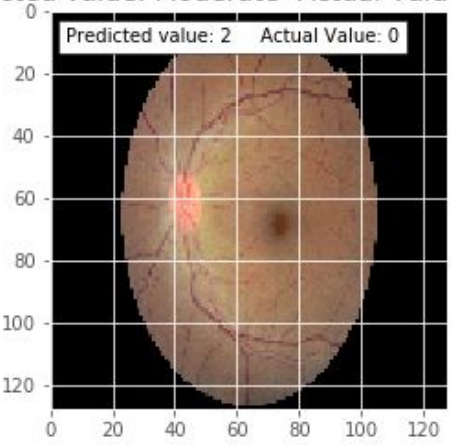
Predicted value: No DR- Actual Value: No DR



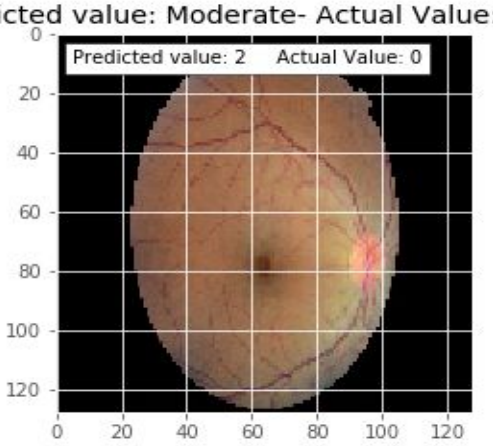
Predicted value: No DR- Actual Value: No DR



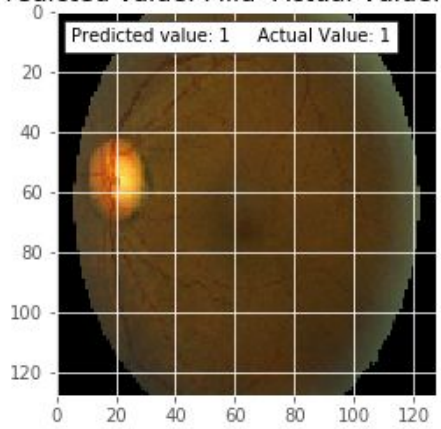
Predicted value: Moderate- Actual Value: No DR



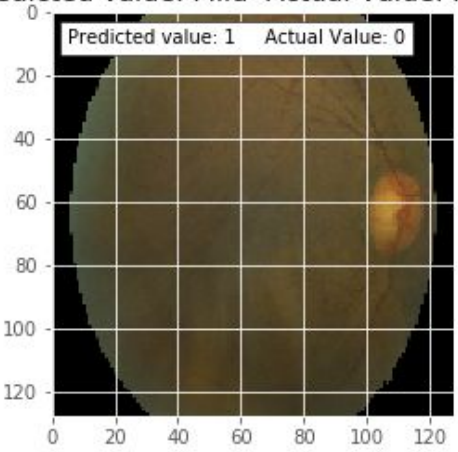
Predicted value: Moderate- Actual Value: No DR



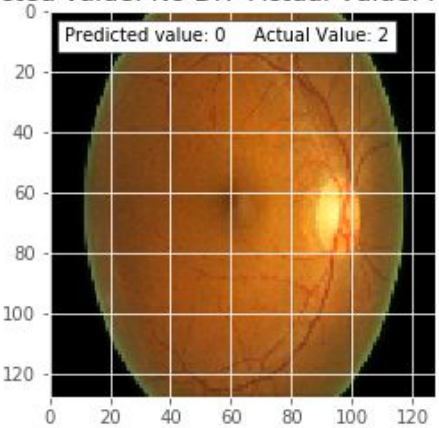
Predicted value: Mild- Actual Value: Mild



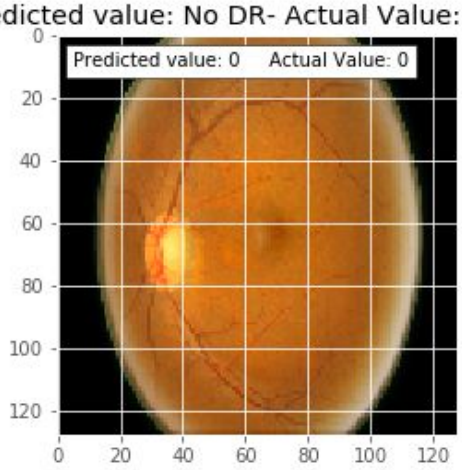
Predicted value: Mild- Actual Value: No DR



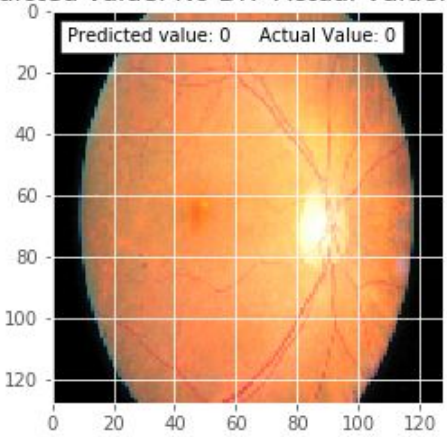
Predicted value: No DR- Actual Value: Moderate



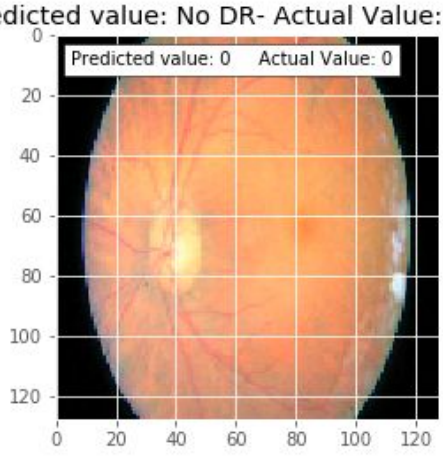
Predicted value: No DR- Actual Value: No DR



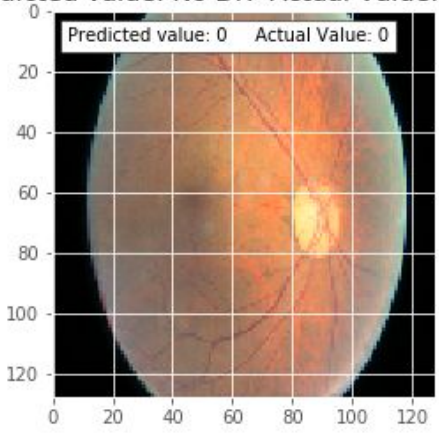
Predicted value: No DR- Actual Value: No DR



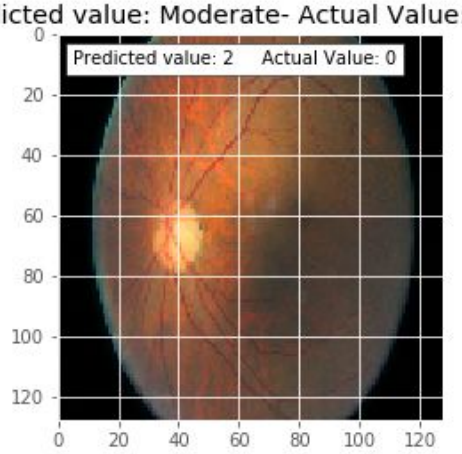
Predicted value: No DR- Actual Value: No DR



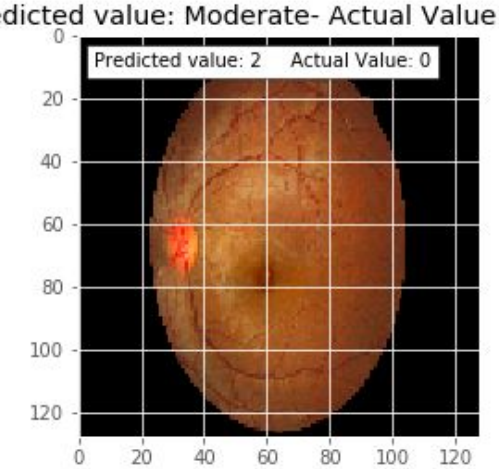
Predicted value: No DR- Actual Value: No DR



Predicted value: Moderate- Actual Value: No DR



Predicted value: Moderate- Actual Value: No DR



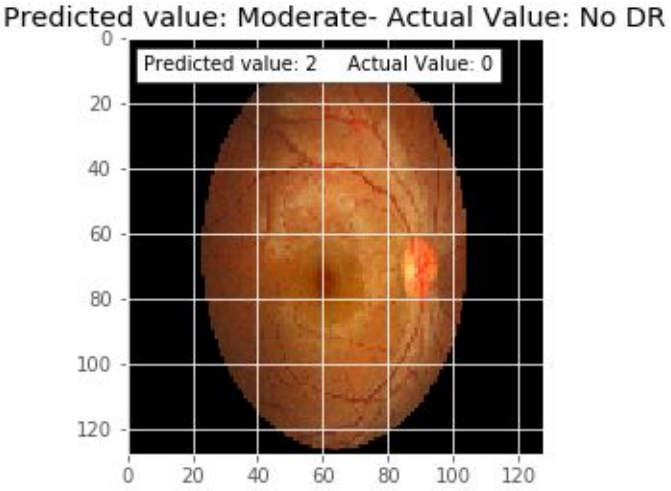


Table containing details of parameter testing and tuning:

Model No.	**Iteration**	**Parameters**	**Training and Testing accuracy**
I.	5	Number of layers = 11 Filters for layer 1 = 32 Kernel Size Layer 1= (3, 3) Activation Function = ReLU Filters for layer 2 = 32 Kernel Size Layer 2 = (3, 3) Activation Function = ReLU Pool size layer 3 = (2, 2) Filters for layer 4 = 64 Kernel Size Layer 4= (3, 3) Activation Function = ReLU Filters for layer 5 = 64 Kernel Size Layer 5= (3, 3)	Training accuracy = 41.20 Testing accuracy = 49.14

		Activation Function = ReLU Pool size layer 6 = (2, 2) Filters for layer 7 = 64 Kernel Size Layer 7= (3, 3) Activation Function = ReLU Filters for layer 8 = 64 Kernel Size Layer 8= (3, 3) Activation Function = ReLU Pool size layer 9 = (2, 2) No. of units of layer 10 = 512 Activation Function = ReLU Activation Function for dense layer 10 = softmax	
I.	10	Number of layers = 11 Filters for layer 1 = 32 Kernel Size Layer 1= (3, 3) Activation Function = ReLU Filters for layer 2 = 32 Kernel Size Layer 2 = (3, 3) Activation Function = ReLU Pool size layer 3 = (2, 2) Filters for layer 4 = 64 Kernel Size Layer 4= (3, 3) Activation Function = ReLU Filters for layer 5 = 64 Kernel Size Layer 5= (3, 3) Activation Function = ReLU Pool size layer 6 = (2, 2) Filters for layer 7 = 64 Kernel Size Layer 7= (3, 3) Activation Function = ReLU Filters for layer 8 = 64	Training accuracy = 38.86 Testing accuracy = 49.51

		Kernel Size Layer 8= (3, 3) Activation Function = ReLU Pool size layer 9 = (2, 2) No. of units of layer 10 = 512 Activation Function = ReLU Activation Function for dense layer 10= softmax	
I.	30	Number of layers = 11 Filters for layer 1 = 32 Kernel Size Layer 1= (3, 3) Activation Function = ReLU Filters for layer 2 = 32 Kernel Size Layer 2 = (3, 3) Activation Function = ReLU Pool size layer 3 = (2, 2) Filters for layer 4 = 64 Kernel Size Layer 4= (3, 3) Activation Function = ReLU Filters for layer 5 = 64 Kernel Size Layer 5= (3, 3) Activation Function = ReLU Pool size layer 6 = (2, 2) Filters for layer 7 = 64 Kernel Size Layer 7= (3, 3) Activation Function = ReLU Filters for layer 8 = 64 Kernel Size Layer 8= (3, 3) Activation Function = ReLU Pool size layer 9 = (2, 2) No. of units of layer 10 = 512 Activation Function = ReLU Activation Function for dense	Training accuracy = 41.79 Testing accuracy = 53.57

		layer 10= softmax	
II.	5	Number of layers = 10 Filters for layer 1 = 32 Kernel Size Layer 1= (3, 3) Activation Function = ReLU Pool size layer 2 = (2, 2) Filters for layer 3 = 64 Kernel size layer 3 = (3, 3) Activation function layer 3 = ReLU Pool sie layer 4 = c(2, 2) Filters for layer 5 = 128 Kernel size layer 5 = c(3, 3) Activation function layer 5 = ReLU Pool size layer 6 = c(2, 2) No. of units layer 7 = 2048 Activation function layer 7 = ReLU No. of units layer 8 = 7 Activation Function layer 8 = ReLU No. of units layer 9 = 512 Activation Function layer 9 = ReLU Activation Function Dense Layer 10 = softmax	Training accuracy = 16.05 Testing accuracy = 21.73
II.	10	Number of layers = 10 Filters for layer 1 = 32	Training accuracy =

		Kernel Size Layer 1= (3, 3) Activation Function = ReLU Pool size layer 2 = (2, 2) Filters for layer 3 = 64 Kernel size layer 3 = (3, 3) Activation function layer 3 = ReLU Pool sie layer 4 = c(2, 2) Filters for layer 5 = 128 Kernel size layer 5 = c(3, 3) Activation function layer 5 = ReLU Pool size layer 6 = c(2, 2) No. of units layer 7 = 2048 Activation function layer 7 = ReLU No. of units layer 8 = 7 Activation Function layer 8 = ReLU No. of units layer 9 = 512 Activation Function layer 9 = ReLU Activation Function Dense Layer 10 = softmax Activation Function = ReLu	13.76 Testing accuracy = 20.27
II.	30	Number of layers = 10 Filters for layer 1 = 32 Kernel Size Layer 1= (3, 3) Activation Function = ReLU Pool size layer 2 = (2, 2)	Training accuracy = 16.92 Testing accuracy =

		Filters for layer 3 = 64 Kernel size layer 3 = (3, 3) Activation function layer 3 = ReLU Pool sie layer 4 = c(2, 2) Filters for layer 5 = 128 Kernel size layer 5 = c(3, 3) Activation function layer 5 = ReLU Pool size layer 6 = c(2, 2) No. of units layer 7 = 2048 Activation function layer 7 = ReLU No. of units layer 8 = 7 Activation Function layer 8 = ReLU No. of units layer 9 = 512 Activation Function layer 9 = ReLU Activation Function Dense Layer 10 = softmax	23.19

From the table above we also see that Model 1 was better than model 2 (removed from the python file).

Conclusion:

We notice that the testing accuracy is much higher than the training accuracy and so the model does not overfit. From the table above we also see that Model 1 was better than model 2 (removed from the file). From the images of the predicted value and the actual value we notice that even though there are certain times that the images have predicted no DR when there is proliferative retinopathy, the model has mostly either predicted accurately or predicted a serious level for a not so serious level which is still better than predicting a lower level of DR when actually it is an advanced level of DR. False positives are better than false negatives. The accuracy using Convolution Neural Networks alone was not impressive and we could improve the accuracy by combining with other methods.

Challenges:

One of the challenges of the project was the limitations of the computational resources we had available. Deep CNN models take a long time to train, and the resources necessary to train them become expensive over time. Because of these time and computational constraints, there was a tradeoff between training time and accuracy.

The other challenge was the acquisition of data. The resources to accommodate the huge amount of data was limited and so local systems with more storage had to be used. The use of the UTD server and box created authentication errors and we did not want to manually take a subset of the data as it would be difficult to properly subset the data with the right distribution of images of different levels of the disease.

Resizing the images to the appropriate size so that they interface with the CNN model properly.

Future Work:

Using Transfer Learning models like ResNet, VGG16, etc., can help improve accuracy. These CNN architectures are standard models used in the past and have worked out well in a lot of identification and prediction research using CNN.

As there are very less images for the classes 3 and 4 so our model got less information about these classes. We need more data for these classes.

Further cleaning of data needs to be done. Attempting to crop by black background to clean data will enhance the model and the test accuracy.

Another area of work could be to combine different classifiers using an ensemble approach like ACNN that is AdaBoost-CNN.