Machine Learning Engineer Nanodegree Capstone Project

Aayush Bhaskar

25th June, 2018.

Project Definition:

Project Overview:

The project that I have undertaken is to show the use of Deep Learning technology in the field of healthcare, by creating a model to predict the degree of Diabetic Retinopathy in a person's eye, using the eye scans. Since the Machine Learning wave has swept across the world, they have carved their niche in the healthcare domain too. IBM's artificial intelligence system Watson has been used in the healthcare and medical sector since 2013, and Google's Deep Mind Health has also made a lot of advances in the field of medical support. Microsoft's Inner Eye initiative started in 2010, is presently working on image diagnostic tools.

Diabetic retinopathy, also known as diabetic eye disease, is a medical condition in which damage occurs to the retina due to diabetes mellitus and is a leading cause of blindness. There have been several works on the employment of machine learning to the detection of diabetic retinopathy. Detection of Diabetic Retinopathy is a rather long detection procedure and might take up to a week to get the results, but with the use of ML, we can get the predictions under 30 minutes.

A few years ago, Google research team studied whether ML could be applied for detection of Diabetic Retinopathy or not. Then on 29th November 2016 Google published an article stating: "Today, in the Journal of the American Medical Association, we've published our results: a deep learning algorithm capable of interpreting signs of DR in retinal photographs, potentially helping doctors screen more patients, especially in underserved communities with limited resources." This article can be referenced here: https://www.blog.google/topics/machine-learning/detecting-diabetic-eye-disease-machine-learning/.

This problem, being that of the healthcare sector has ample data available for training a model for the detection of diabetic eye disease. One such dataset was hosted by Kaggle 3 years ago in one of its competitions. The dataset can be obtained here: https://bit.ly/2Kimuvr and its details will be discussed further in the report. I have used a part of the original dataset because my laptop doesn't have a Nvidia GPU.

Problem Statement:

The presence of diabetic retinopathy (DR) in each image is rated 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 problem here to be solved is to create an automated analysis system capable of assigning a score of diabetic retinopathy based on this scale. This system will take the aid of Machine learning to solve the problems by classifying the input images accordingly and assigning the required score to it. The problem dataset contains images, and looking at its basic concept, this problem, at its core, is an image classification problem. So it would be logical to solve this problem using an efficient CNN model. So, the solution would be to create a relevant CNN model and get the scores for the new images using our classifier.

Metrics:

The metrics for the project is finding how accurate the predictions for the model have been through each step of training of the model. The code that has been used for calculating the accuracy for each step is as follows:

```
cnn_mod=model()
cross_entropy=-tf.reduce_sum(y*tf.log(cnn_mod))
epoch_batch=tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
corr_pred=tf.equal(tf.argmax(cnn_mod, 1), tf.argmax(y, 1))
acc=tf.reduce_mean(tf.cast(corr_pred, tf.float32), name='acc')
```

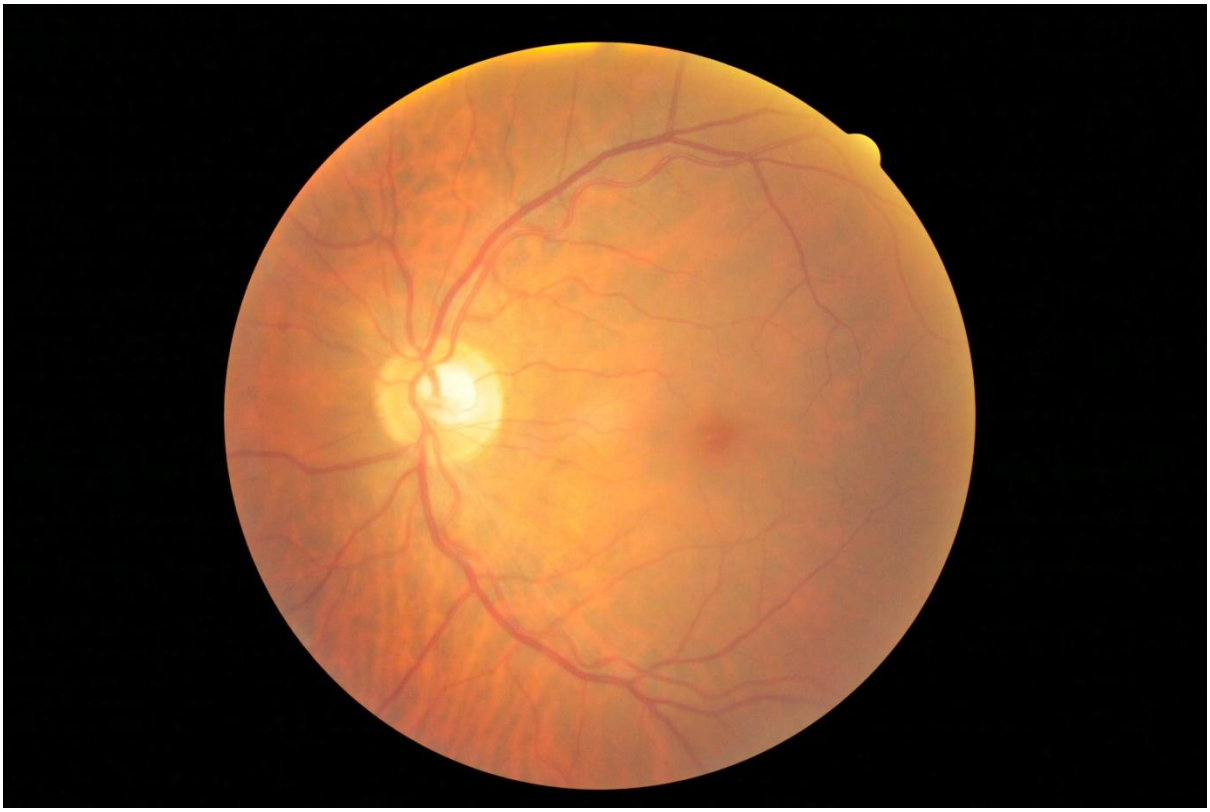
This code basically takes the predictions by calling the model for each step, and then it calculates the mean of the total number of correct predictions that have occurred during each epoch for each batch.
The 'tf' here stands for the Tensorflow library which has been used in order to write the code for the model. Hence, calculating the mean accuracy for each step is the metrics used for this model.
Also, graphs have been plotted for the cross-entropy as well as the accuracy of the model, which will be visualized later in the report.

Analysis:

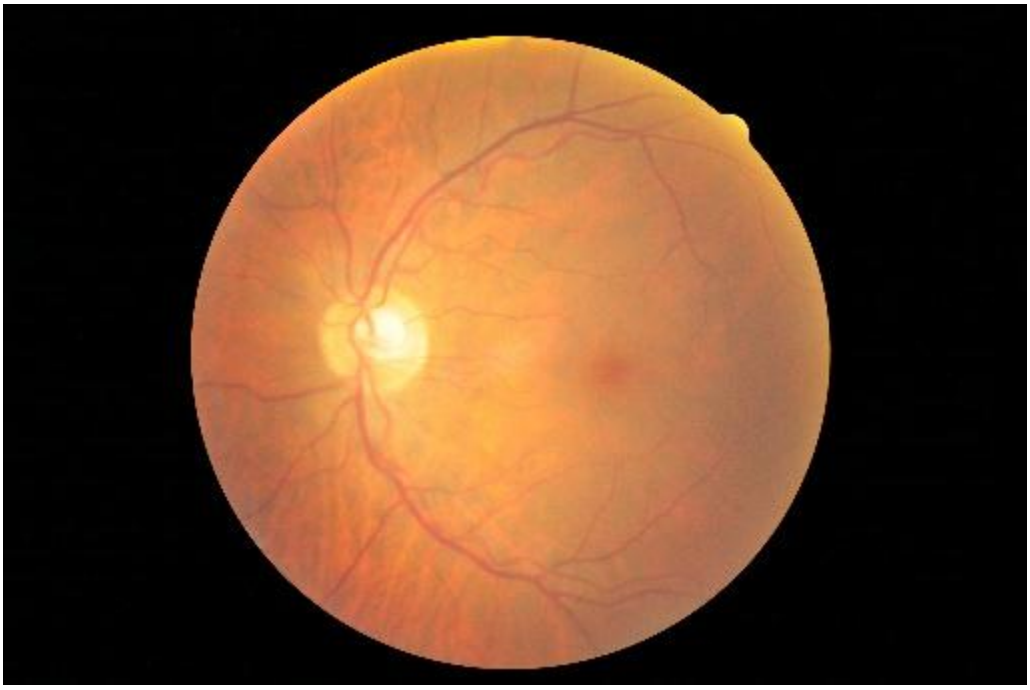
Data Exploration and Visuals:

As described earlier, the dataset that has been used is from a Kaggle contest. The original dataset has a total of 35,000 scans of the retina scans, and their intensity of the diabetic eye disease has been numbered in the scale 0-4 as explained above. This dataset is available in parts on the official contest site, and it has been divided into 5-6 parts so that one can download each part easily. The images come zipped in .005 format file, which has been unzipped and the folder containing the scans has been extracted using 7zip software. The image labels on the scale 0-4 have been provided in a separate csv file, along with the image name. The dataset contain eye scans in pairs, i.e. one for the left and one for the right eye, and they may have different intensities of the disease. So, basically we have scans from 17,500 people.

I have used 1,000 images from this dataset to train my Deep learning model. Now, each image in the dataset is a high quality scan of the eyeball. Each image is a RGB image (3 channel image) having a resolution 3168X4752 pixels. The image before preprocessing looks as below:



Each image in the dataset has a very large resolution, so what we do is, we preprocess it, and lower its resolution to 340X512 pixels. The image looks as below:



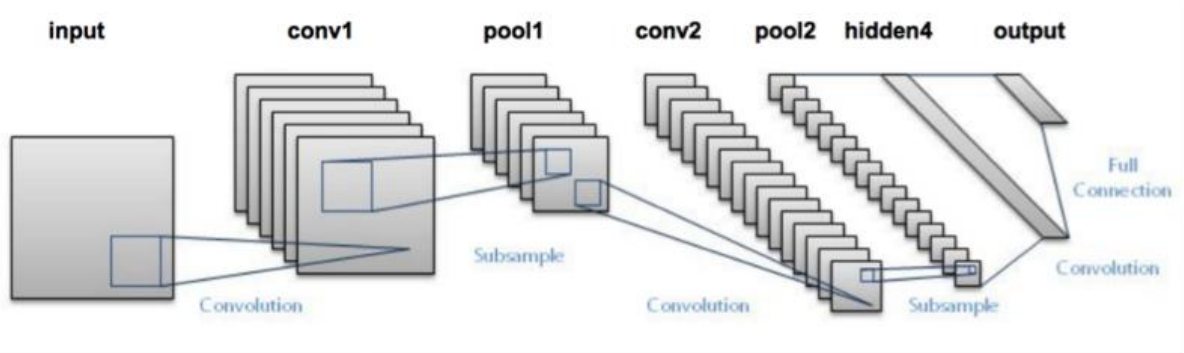
The black borders have been chopped off and the quality has gone down. These images have been used to train the model. We have approximately 200 images under each label 0-4 in the dataset.
The feature of each image is of course, the intensity of each pixel that is present in the image. That is the feature that has been used to train the model for prediction.

Algorithms and Techniques:

As described above, the project at its core is using images to classify the score of the diabetic retinopathy that is present in the eye. So, it is a project of image classification, and the thing that works best on an image classification model is a Convolution Neural Network (CNN) model.

But, before using a CNN the image has to be thoroughly processed so that it can be used. First of all, we need to read in each image as a 3-dimensional array, so that it can be fed into the model. Also, we need to reduce its dimensions as described earlier. We do all these using the OpenCV image processing library with the help of numpy library.

So, this project employs a CNN model to achieve its goal. A CNN model generally uses a convolution layer, which basically is used to emulate the way our eyes see the images. A convolution model takes a part of the image and processes it, just like our neurons, which see specific parts of the image. Then comes the pooling layer, which processes the output from the convolution layer and reduces its size so that the data doesn't become too bulky. Next comes the dropout layer, a flattening layer, and finally the fully connected layers (dense layers), which give the output by using an activation function like softmax, tanh etc.
I have used softmax function here. The model will be described in details in the later section of the report.



This is how a basic CNN model architecture looks like. The one used in the project is a much more complex model.

Benchmark:

Considering the Kaggle contest held 3 years ago, the best solution of the contest had an accuracy of 86% on test data, and 55% on new data (from a different source). This would serve as the benchmark for the solution, considering the use of only one part of the dataset. The best solution to the problem can be found here: https://www.kaggle.com/kmader/vgg16-640hr-nloss-retinopathy/code.
This a pretty high accuracy, and my model will try to get as high an accuracy as possible, so that it can be compared and evaluated against the benchmark.

Methodology:

Data Preprocessing:

The dataset contains 1,426 different scans of the eye and the intensity of the disease on a scale of 0-4 associated with it. The images have a resolution of 3168X4752 pixels. The following preprocessing steps have been taken to make the data ready to be fed into the neural network.

1. **Each image of the dataset has been read into the program as a 3-dimensional array, so that it can be processed further.**
2. **The data array for each image is of the dimension 3168X4752X3, too big to be processed and also containing excessive black borders, so we reduce the size of the images to an array of dimensions 340X512X3.**
3. **The final step is normalizing the images for training, i.e. making each pixel's intensity between 0 and 1, by dividing the pixels by the value (Max Pixel Value-Min Pixel Value).**
4. **The dataset is then divided into training and testing sets to be used for training and evaluating the data.**

The code for the preprocessing steps is as follows:

```
import cv2 import numpy as np
x=[]
y=[]
for i in range (0,len(df)-1):
    #df is the list containing the image names and labels
    img = cv2.imread("E://train//"+df[i][0]+".jpeg")
    img = cv2.resize(img,(512,340))
    img = np.array(img).reshape((340,512,3))
    x.append(img)
    y.append(df[i][1])
    i+=1
x=np.array(x)
y=np.array(y).reshape(len(y),1)


def normalize(x):
    max_x,min_x=x.max(),x.min()
    norm_x=np.zeros(tuple(x.shape))
    norm_img=x.shape[0]
    for nr in range(norm_img):
        norm_x[nr,...]=(x[nr,...]-float(min_x))/(float(max_x-min_x))
    return norm_x
```

The image before and after preprocessing have already been shown above.
The arrays x and y are numpy arrays containing the data features and the labels for the data respectively. Now they need to be split into training and testing data. Since, the dataset contains 1,426 images; I have used 1,000 images for training, in batches of 100 to train the network. Also, I have used 400 images to test the efficiency of my model.
The code for the above processing is as below:

```
from sklearn.model_selection import train_test_split as tts
x_train, x_test, y_train, y_test = tts(x, y, test_size=0.2987,
random_state=42)
X_train=np.split(x_train,10)
y_train=np.split(y_train,10)
```

```
X_test=x_test[:400]
Y_test=y_test[:400]
x_test=np.split(X_test,4)
y_test=np.split(Y_test,4)
```

This dataset doesn't contain any abnormalities or outliers. Hence it is now safe to use the arrays X_train[] and y_train to train my CNN model.

One final thing, the labels have been one hot encoded for the categories to compare in the classification stage. For e.g. 3 can be represented as [0 0 0 1 0] and 0 as [1 0 0 0 0]. The code is as follows:

```
def one_hot_encode(x):
    from sklearn.preprocessing import OneHotEncoder
    enc=OneHotEncoder(n_values=5)
    ohc_labels=enc.fit_transform(np.array(x).reshape(-1,1)).toarray()
    return ohc_labels
```

Implementation:

Since I am going to use CNN for the model, let me describe how a general CNN model looks like. It has the following layers:

1. **Convolution layers**
2. **Pooling layers**
3. **Dropout layer**
4. **Flattening layer**
5. **Dense layers**
6. **Output layer**

Now, out of these, we can have multiples layers of each category, but generally we use a single dropout and a single flattening layer. There has to be one output layer at the end. The number of other layers depends on how accurate we want the model to be, without making it an over-fitted model.
For writing the codes of this model, I have used tensorflow.layers and tensorflow.nn modules. In the following paragraphs, I have described the features used in each layers. They are as follows:

For the Convolution 2D layers, the arguments used are:

1. **An input tensor (array) for the layer**
2. **Filters, i.e. the dimensionality of the output space**
3. **Kernel Size, i.e. the height and width of the convolution window of the layer**
4. **Strides, specifying the stride of the convolution window along the height and width**
5. **Padding, whose value is set to 'same'.**

For the Pooling layers, we have used max pooling 2d layers, and the arguments used are:

1. **An input tensor**
2. **Pool Size, specifying the size of the pooling window**
3. **Strides**
4. **Padding, value set to 'same'.**

The dropout layer has only 2 arguments, input tensor and the dropout rate. Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent over-fitting.

The flattening layer, which basically flattens the input tensor to a 2D tensor, has a single input, the tensor which needs to be flattened.

Then we have the dense layers, containing the following arguments:

1.) **The input tensor**
2.) **The dimensionality of the output tensor**

Finally, we have the output activation layer, for which we have used a softmax function for activation.

So, the initial CNN structure was to use 3 convolution layers and 3 maxpool layers alternatively, after which there was a dropout and a flattening layer, then there were 3 dense layers, and then finally the output softmax function.

The final model has been thoroughly refined and there are various more layers in the model now. The final model will be discussed in the next section.
Now, to being with the execution of the model, we first define a cross-entropy function as follows:

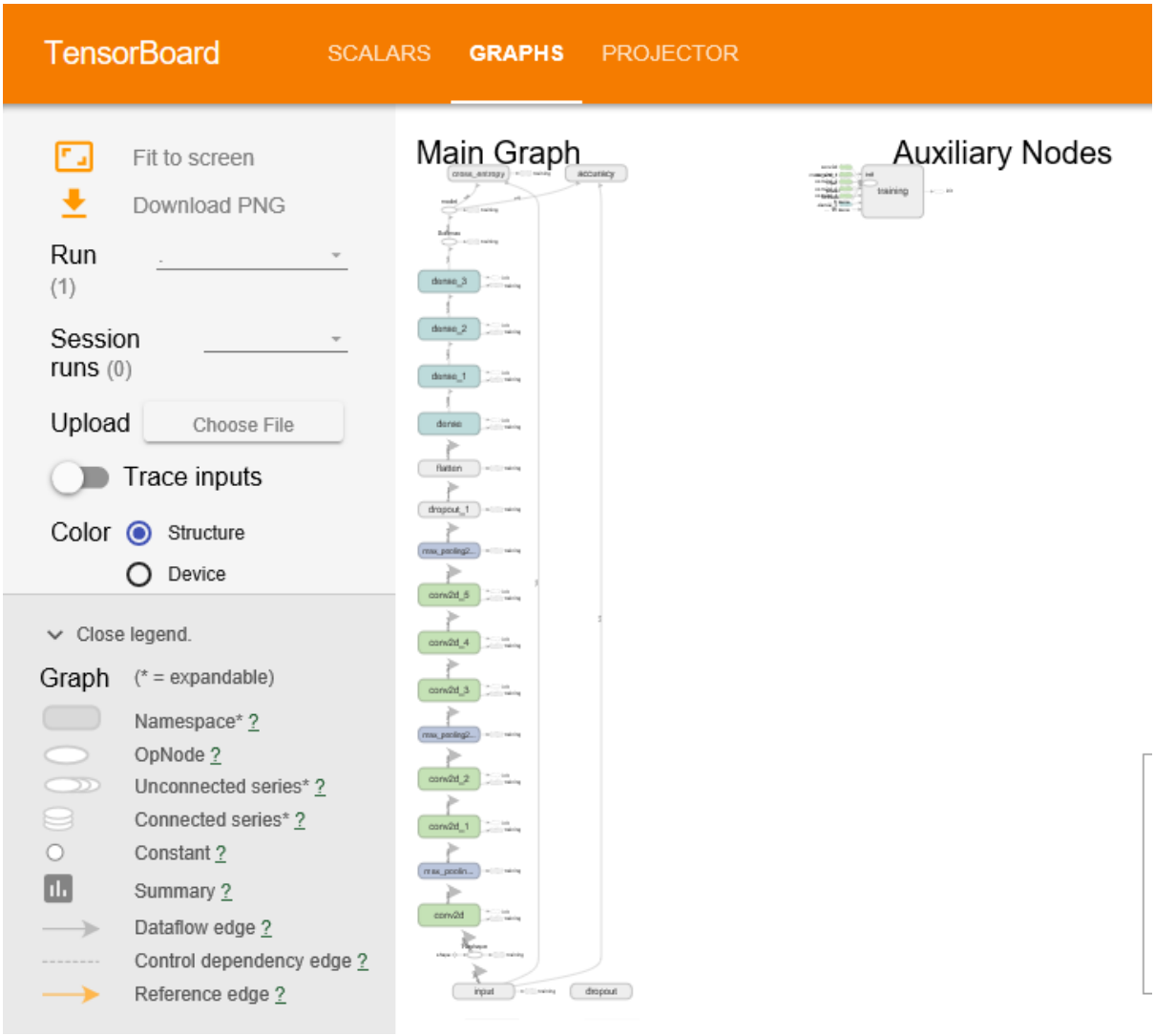
```
cross_entropy=-tf.reduce_sum(y*tf.log(model()))
```

So, now that we have the cross entropy of the model, we next feed it into our optimizer so that we can get the output of this training iteration. We have used Adam Optimizer, with a learning rate of 0.0001 to optimize the training of our CNN model. After this we have calculated the value of our accuracy by finding the number of correct predictions. The code is as follows:

```
epoch_batch=tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
corr_pred=tf.equal(tf.argmax(cnn_mod, 1), tf.argmax(y, 1))
acc=tf.reduce_mean(tf.cast(corr_pred, tf.float32), name='acc')
```

Thus, we have trained the model and also found out the metrics to evaluate the model. I'll discuss the refinements in the next section.

The model's final graph looks as below:



Refinements:

The initial model described above, with a learning rate of 0.001 gave a maximum validation accuracy of 66% and a testing accuracy of 53.24%, and it was ran for 10 epochs.
The following model was then made as the final model, with a learning rate of 0.0001, a total of 15 epochs, and it gave a maximum validation accuracy of 83% and a testing accuracy of 67.99%. The learning rate below 0.0001 made the model quite a bit slow and also the model got a slightly lower accuracy. 0.0001 works just perfect for my model. The model is as below:

1. **A convolution layer**
2. **A maxpool layer**
3. **2 convolution layers**
4. **A maxpool layer**
5. **3 convolution layers**
6. **A maxpool layer**
7. **A dropout layer**
8. **A flattening layer**
9. **4 dense layers with 256, 512, 1024, and 5 outputs respectively.**
10. **A softmax activation layer.**

The initial model had dense layers having 192, 384 and 5 outputs respectively, followed by a softmax function.

Results:

Model Evaluation and Validation:

The final model looks as follows:

```
def model():
    img_inp=tf.reshape(x,[-1,340,512,3])
    layer=conv2d(img_inp,32,[7,7],[2,2])
    layer=maxpool2d(layer,[5,5],[2,2])

    layer=conv2d(layer,32,[5,5],[1,1])
    layer=conv2d(layer,64,[3,3],[1,1])
    layer=maxpool2d(layer,[3,3],[2,2])

    layer=conv2d(layer,64,[3,3],[1,1])
    layer=conv2d(layer,128,[3,3],[1,1])
    layer=conv2d(layer,256,[3,3],[1,1])
    layer=maxpool2d(layer,[3,3],[2,2])

    layer=tf.layers.dropout(layer,0.4)

    layer=tf.layers.flatten(layer)

    layer=tf.layers.dense(layer,256)
    layer=tf.layers.dense(layer,512)
    layer=tf.layers.dense(layer,1024)

    layer=tf.layers.dense(layer,5)

    layer=tf.nn.softmax(layer)

    return layer
```

This model achieves a highest validation accuracy of 83%, and a testing accuracy of 67.99%.

This model works the best when it has a learning rate of 0.0001 or 1e-4, a learning rate below that makes the model quite slow and a bit less accurate, and a higher learning rate would make the model's training converge faster making it an under-fitted model, again lowering the accuracy of the model.

The model having a testing accuracy of 67.99% over 400 samples, can be safely said be faring well to different kinds of unseen data. Hence it can be said that the model provides a generalized solution to the detection of diabetic retinopathy in a person's eye. Since the dataset as well as the test data contains images taken from various different angles, the model fares well, and the results are not affected from the small changes in a particular test data.

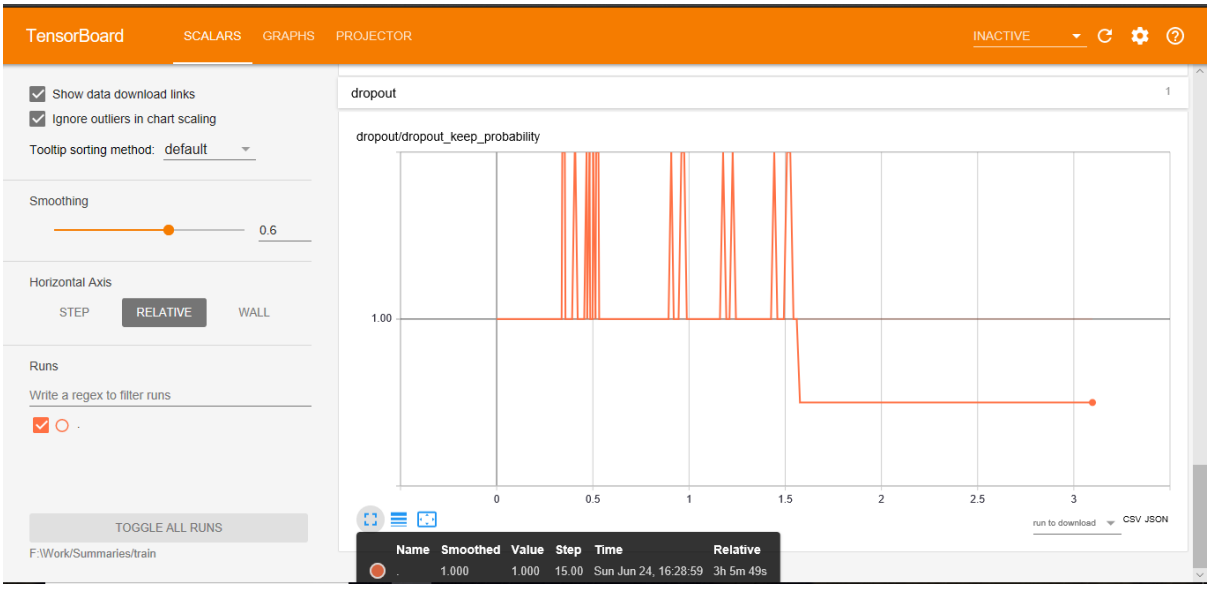
Justification:

The benchmark solution mentioned earlier in the report had a validation accuracy of 86% and a test accuracy of 55% on unknown data. My model fares almost as well as the benchmark solution, giving 83% and 67.99% accuracies respectively. It can be said that the solution provided by me can successfully detect a person's degree of diabetic retinopathy disease and hence this solve the problem statement posed by the project. The solution fares better than the benchmark model, with higher accuracy in testing unknown samples.

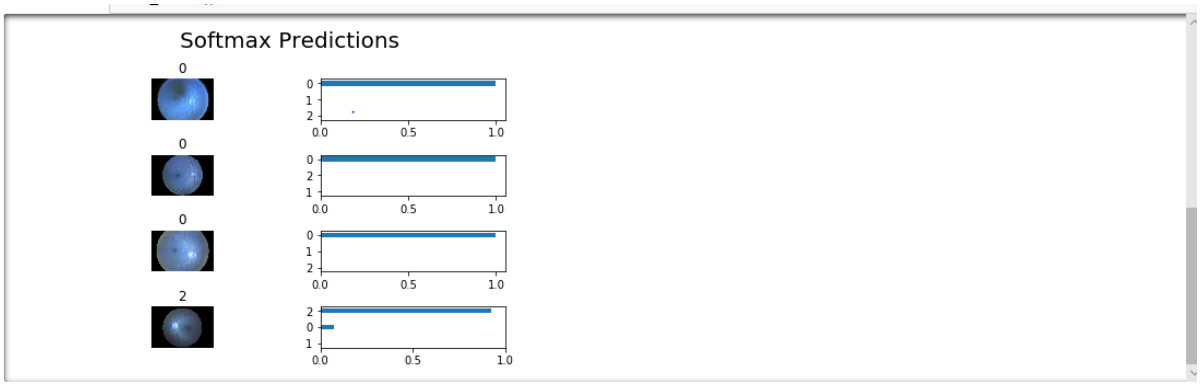
Visualizations:

The graph for the validation accuracy of the model.

The graph for the cross-entropy of the model.



The graph for the keep probability of the model.



A few examples of the data predictions by the model.

Conclusions:

I have trained a CNN model using 1,000 images from Kaggle, each of size 3168X4752 pixels, reduced to 340X512 pixels and used for training. The final model gives a final accuracy of 67.99%. The results are promising, but not so much as to used for evaluating patients' disease as of now.

I thoroughly enjoyed building the CNN model and also enjoyed learning to use tensorboard for the visualizations.

The next step would be train the model on an even bigger dataset, and to find some better means to preprocess the data, so that the most important information areas, i.e. areas around the retina can be specifically extracted and used for training. All these things along with higher number of epochs can be

achieved by training the model on a GPU (I have used Intel i7 CPU for all the processing done here). I would also like to create a frontend UI so that it would have the feel of a beautiful app.

References:

1. https://ieeexplore.ieee.org/document/7914977/
2. https://www.sciencedirect.com/science/article/pii/S1877050916311929
3. https://arxiv.org/pdf/1703.10757
4. https://www.blog.google/topics/machine-learning/detecting-diabetic-eye-disease-machine-learning/
5. https://bit.ly/2Kimuvr **(link for the dataset)**
6. https://tensorflow.org
7. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
8. https://en.wikipedia.org/wiki/Cross_entropy
9. https://www.tensorflow.org/tutorials/layers