Detection of Diabetic Retinopathy using CNN

Jayaram Kuchibhotla
Department of Computer Science
University at Buffalo
Person #:50208766
jayaramk@buffalo.edu

Abstract

This project report describes about the deep learning solution implemented for classifying the Fundus images (retina scans) into five different categories corresponding to five different levels of intensity of the Diabetic Retinopathy (DR) disease. The problem statement is presented by Kaggle. Convolutional Neural Network (CNN) is implemented for the classification task. The report has the following steps 1) Introduction 2) Description of the dataset 3) Explanation of Preprocessing 4) Architecture of the model 5) Results 6) Problems and Possible Improvements 7) Conclusion

1 Introduction

CNNs are widely used in the field of Computer Vision for classifying images as they are highly able in extracting the features from an image and provide high prediction accuracies. Diabetic Retinopathy (DR) is detected by manual observation of multiple features in the scanned color images of the eye retina. This task is time consuming and difficult. Application of CNNs helps in faster and effective diagnosis of the disease.

2 Description of the dataset

The dataset is provided by EyePacs, a free platform for retinopathy screening, which consists of 35123 fundus images (color scans eye retina). The data is divided into five categories
0- NoDR
1- Mild
2- Moderate
3- Severe
4- Proliferative DR

After removing the corrupted images, the dataset is organized as follows:

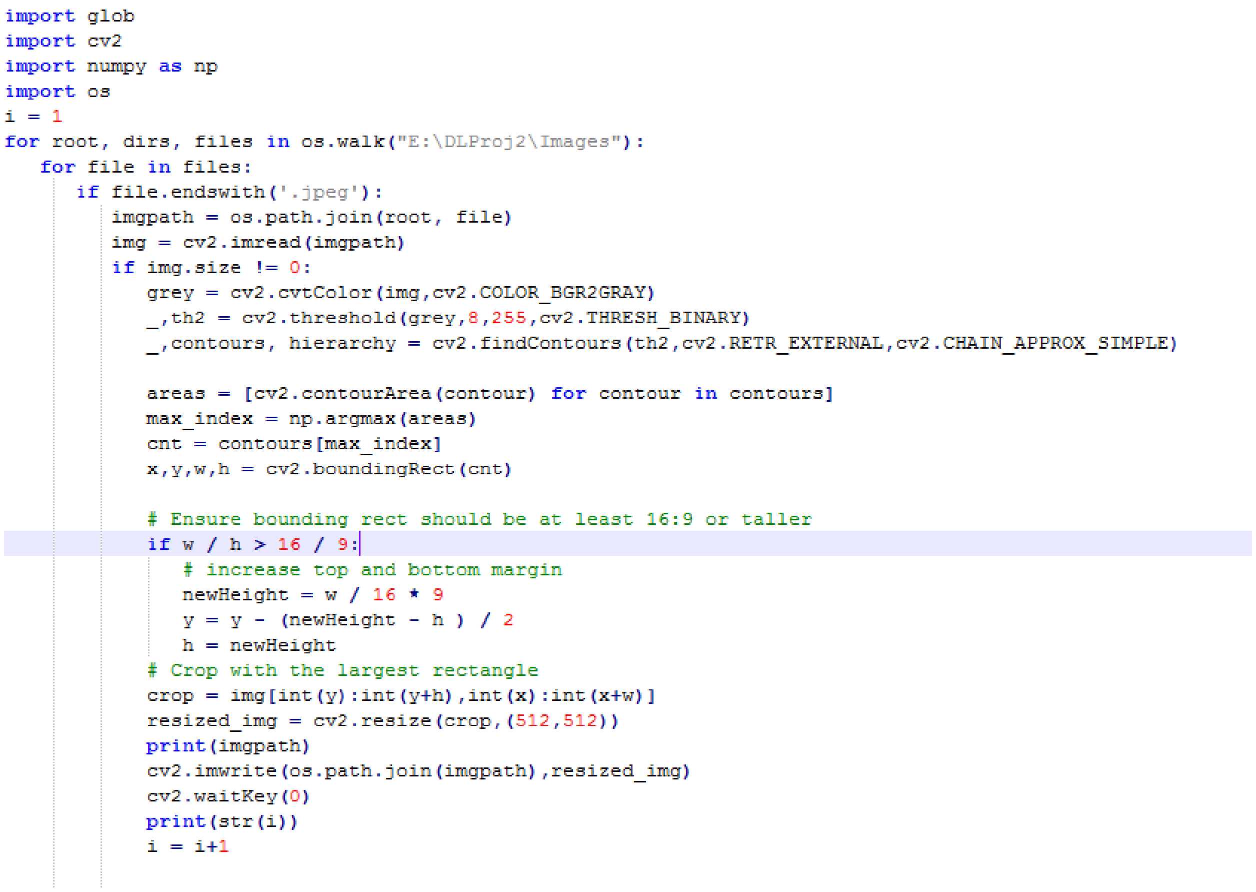
Partition	Train	Test	Total
Number of images	28097	7025	35122

3 Preprocessing

The size of each image is of the order 3000*2000 pixels. Almost every image consists of a black border of significant size. Preprocessing is done to remove this border and retain only the necessary contents of the image. Thereafter all the images are resized to a uniform size of 512*512 pixels for each of the R,G,B frames .

OpenCV is used for the pre-processing .The main steps involved are listed as below:
1. Reading the image from the folder
2. Make a copy of the image and convert the copy to Grey Image.
3. Find the contours in the grey image and calculate the area of each contour
4. Use the dimensions of the maximum area contour in bounding rectangle and crop the original image
5. Resize the crop to 512*512 and save it to folder

The code snippet for the above process is as below:



```python
import glob
import cv2
import numpy as np
import os
i = 1
for root, dirs, files in os.walk("E:\DLProj2\Images"):
   for file in files:
      if file.endswith('.jpeg'):
         imgpath = os.path.join(root, file)
         img = cv2.imread(imgpath)
         if img.size != 0:
            grey = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            _,th2 = cv2.threshold(grey,8,255,cv2.THRESH_BINARY)
            _,contours, hierarchy = cv2.findContours(th2,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

            areas = [cv2.contourArea(contour) for contour in contours]
            max_index = np.argmax(areas)
            cnt = contours[max_index]
            x,y,w,h = cv2.boundingRect(cnt)

            # Ensure bounding rect should be at least 16:9 or taller
            if w / h > 16 / 9:
               # increase top and bottom margin
               newHeight = w / 16 * 9
               y = y - (newHeight - h ) / 2
               h = newHeight
            # Crop with the largest rectangle
            crop = img[int(y):int(y+h),int(x):int(x+w)]
            resized_img = cv2.resize(crop,(512,512))
            print(imgpath)
            cv2.imwrite(os.path.join(imgpath),resized_img)
            cv2.waitKey(0)
            print(str(i))
            i = i+1
```

Sample Image before pre-processing

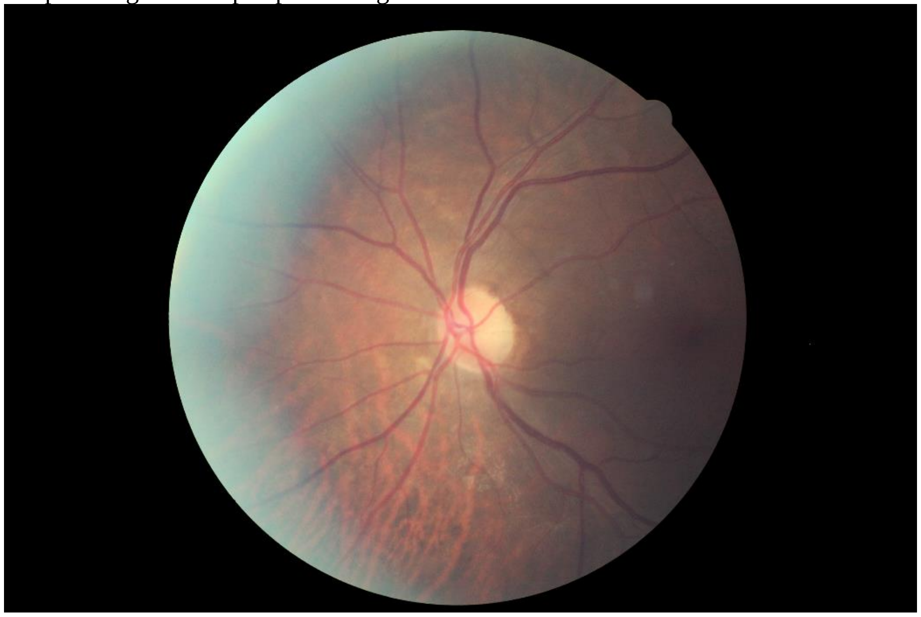
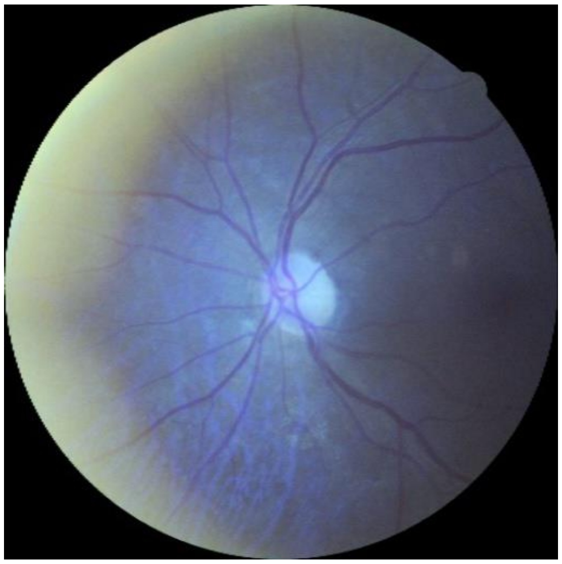


Image after removing the back borders and resizing:



The dimension of the above image is (512*512) which is given as input to the CNN Model

4 Model Architecture

The model architecture is similar to the VGGNET or OxfordNET. The Layer details are show below. The batch size is constant for every layer in the network, which is 20.

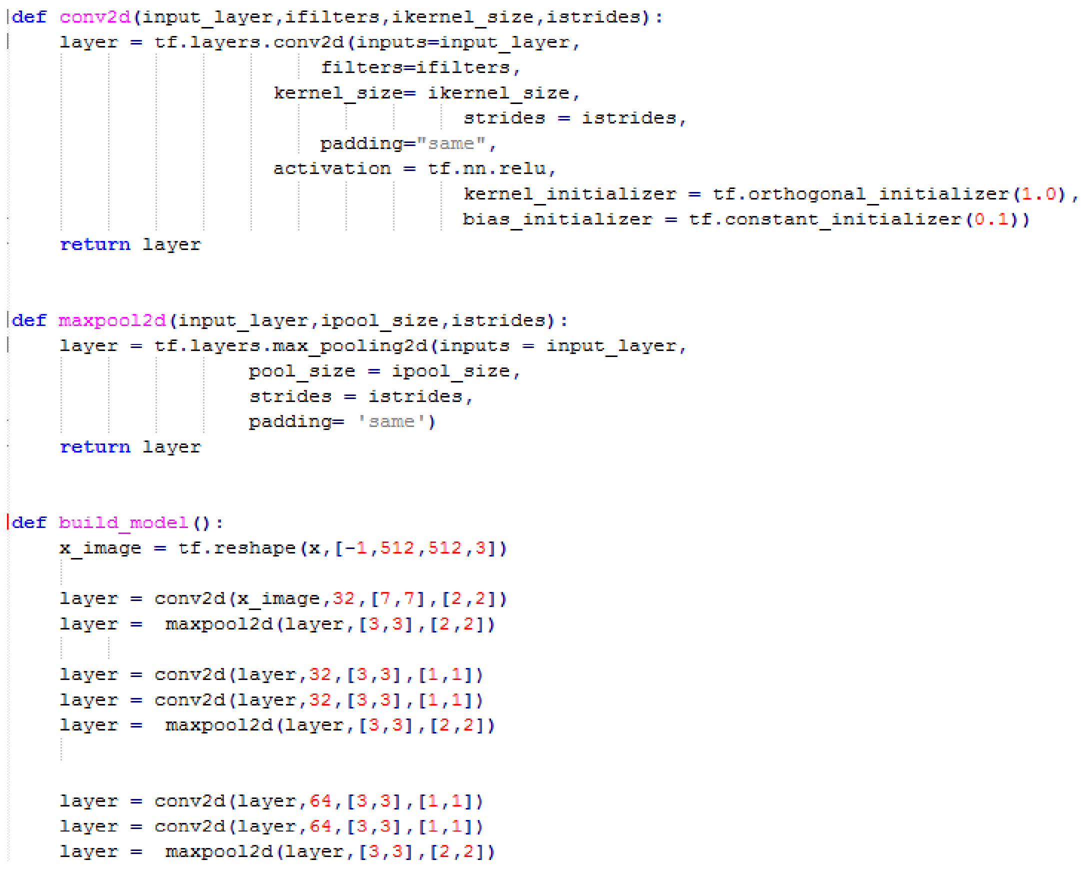
Nr	Name	Channels	Width	Height	Filter Size	Stride
0	Input	3	512	512		
1	Conv	32	256	256	7*7	2*2
2	*Max pool*	32	127	127	3*3	2*2
3	Conv	32	127	127	3*3	1*1
4	Conv	32	127	127	3*3	1*1
5	*Max pool*	32	63	63	3*3	2*2
6	Conv	64	63	63	3*3	1*1
7	Conv	64	63	63	3*3	1*1
8	*Max pool*	64	31	31	3*3	2*2

9	Conv	128	31	31	3*3	1*1
10	Conv	128	31	31	3*3	1*1
11	Conv	128	31	31	3*3	1*1
12	Conv	128	31	31	3*3	1*1
13	*Max pool*	128	15	15	3*3	2*2
14	Conv	256	15	15	3*3	1*1
15	Conv	256	15	14	3*3	1*1
16	Conv	256	15	15	3*3	1*1
17	Conv	256	15	15	3*3	1*1
18	Maxpool	256	7	7	3*3	2*2
19	Dropout	256	7	7		
20	Flatten	256	7	7		
21	Dense	1024				
22	Leaky Relu	1024				
23	Droput	1024				
24	Dense	5				
25	Softmax	5				

Details:

- Adam Optimizer is used in training this model
- Weights are initialized by orthogonal initializer
- Cross Entropy Loss Function is utilized
- Learning Rate of value 0.0001 is used

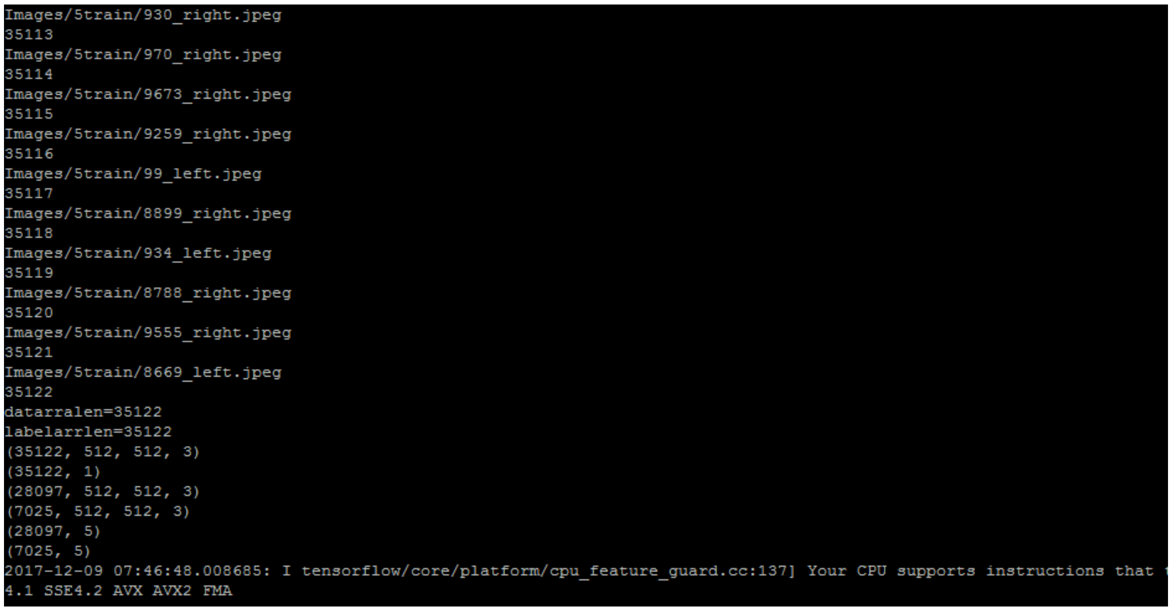
Code snippet that implements the above architecture:



```
def conv2d(input_layer,ifilters,ikernel_size,istrides):
    layer = tf.layers.conv2d(inputs=input_layer,
                          filters=ifilters,
                      kernel_size= ikernel_size,
                                      strides = istrides,
                          padding="same",
                      activation = tf.nn.relu,
                                      kernel_initializer = tf.orthogonal_initializer(1.0),
                                      bias_initializer = tf.constant_initializer(0.1))
    return layer


def maxpool2d(input_layer,ipool_size,istrides):
    layer = tf.layers.max_pooling2d(inputs = input_layer,
                    pool_size = ipool_size,
                    strides = istrides,
                    padding= 'same')
    return layer


def build_model():
    x_image = tf.reshape(x,[-1,512,512,3])

    layer = conv2d(x_image,32,[7,7],[2,2])
    layer =  maxpool2d(layer,[3,3],[2,2])

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


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

5 Results:

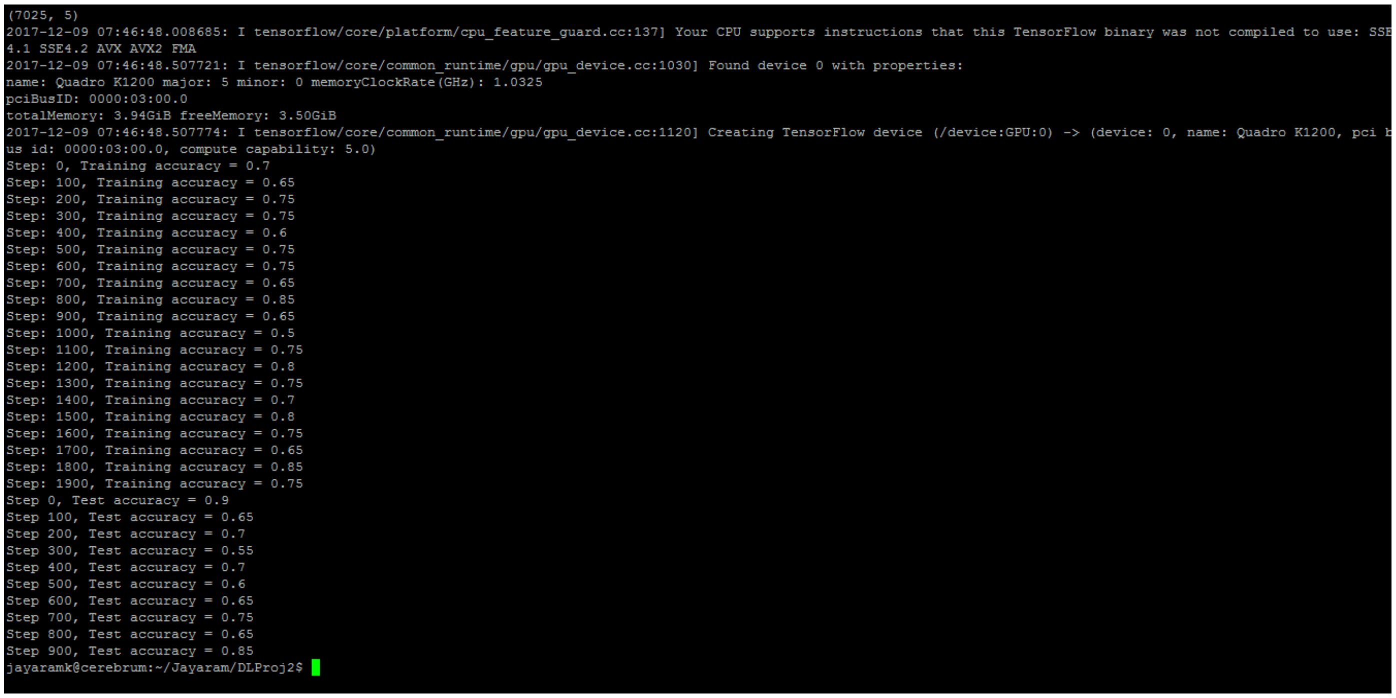
Snapshot showing
1. Reading of the Processed Images
2. Shapes of the total dataset, training and testing sets for both input data and labels



```
Images/5train/930_right.jpeg
35113
Images/5train/970_right.jpeg
35114
Images/5train/9673_right.jpeg
35115
Images/5train/9259_right.jpeg
35116
Images/5train/99_left.jpeg
35117
Images/5train/8899_right.jpeg
35118
Images/5train/934_left.jpeg
35119
Images/5train/8788_right.jpeg
35120
Images/5train/9555_right.jpeg
35121
Images/5train/8669_left.jpeg
35122
datarralen=35122
labelarrlen=35122
(35122, 512, 512, 3)
(35122, 1)
(28097, 512, 512, 3)
(7025, 512, 512, 3)
(28097, 5)
(7025, 5)
2017-12-09 07:46:48.008685: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that
4.1 SSE4.2 AVX AVX2 FMA
```

- Training is done using a batch size of 20 for 2000 iterations
- Testing is done using a batch size of 20 for 1000 iterations

Snapshot showing the details of Training and Testing:



```
(7025, 5)
2017-12-09 07:46:48.008685: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use: SSE
4.1 SSE4.2 AVX AVX2 FMA
2017-12-09 07:46:48.507721: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1030] Found device 0 with properties:
name: Quadro K1200 major: 5 minor: 0 memoryClockRate(GHz): 1.0325
pciBusID: 0000:03:00.0
totalMemory: 3.94GiB freeMemory: 3.50GiB
2017-12-09 07:46:48.507774: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1120] Creating TensorFlow device (/device:GPU:0) -> (device: 0, name: Quadro K1200, pci b
us id: 0000:03:00.0, compute capability: 5.0)
Step: 0, Training accuracy = 0.7
Step: 100, Training accuracy = 0.65
Step: 200, Training accuracy = 0.75
Step: 300, Training accuracy = 0.75
Step: 400, Training accuracy = 0.6
Step: 500, Training accuracy = 0.75
Step: 600, Training accuracy = 0.75
Step: 700, Training accuracy = 0.65
Step: 800, Training accuracy = 0.85
Step: 900, Training accuracy = 0.65
Step: 1000, Training accuracy = 0.5
Step: 1100, Training accuracy = 0.75
Step: 1200, Training accuracy = 0.8
Step: 1300, Training accuracy = 0.75
Step: 1400, Training accuracy = 0.7
Step: 1500, Training accuracy = 0.8
Step: 1600, Training accuracy = 0.75
Step: 1700, Training accuracy = 0.65
Step: 1800, Training accuracy = 0.85
Step: 1900, Training accuracy = 0.75
Step 0, Test accuracy = 0.9
Step 100, Test accuracy = 0.65
Step 200, Test accuracy = 0.7
Step 300, Test accuracy = 0.55
Step 400, Test accuracy = 0.7
Step 500, Test accuracy = 0.6
Step 600, Test accuracy = 0.65
Step 700, Test accuracy = 0.75
Step 800, Test accuracy = 0.65
Step 900, Test accuracy = 0.85
jayaramk@cerebrum:~/Jayaram/DLProj2$
```

The final accuracy values:
1. At the 2000th Iteration, **Training accuracy** = **75%**
2. At the 1000th Iteration, **Testing accuracy** = **85%**

6 Problems and Possible improvements:

- It was observed that batch size greater than 20 such as32 or 64 results in resource allocation error on Quadro K1200 GPU which has 4 GB Memory .Hence the batch size is restricted to 20

- The preprocessing of images was time consuming, one second per one image. The main reason was the imread() function which involved copying of larger size of arrays

- The usage of shuffle function and split_train_test_data functions of sklearn caused memory error due to the large size of the data. Shuffle is avoided and indices are used for splitting the datasets

- Data Augmentation: The size of the dataset can be increased by changing parameters such as changing such as brightness, contrast values. The operations such as flipping and rotating can also help to augment the dataset size. Larger datasets can help in better training of the model and better prediction accuracy. This is a possible improvement that can be made in the project

7 Conclusion

This problem statement and project helped me to learn about image processing and explore about various CNN architectures.

8 References

[1] https://www.kaggle.com/c/diabetic-retinopathy-detection

[2]http://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/

[3] http://yerevann.github.io/2015/08/17/diabetic-retinopathy-detection-contest-what-we-did-wrong/

[4] http://jeffreydf.github.io/diabetic-retinopathy-detection/

[5] https://github.com/jayaram1125/Classification-/blob/master/ConvolutionalNeuralNetwork.py

[6]http://www.subsubroutine.com/sub-subroutine/2016/9/30/cats-and-dogs-and-convolutional-neural-networks