cs577 Project Report

Team Members: Gonuguntla Venkata Sai Goutham

A20450688

Semester : Spring, 2021

Department of Computer Science

Illinois Institute of Technology

May 4, 2021

Detection of Diabetic Retinopathy using Deep Learning

Problem Statement:

The leading cause of blindness is diabetic retinopathy, millions of people are affected by this disease. One out of two people having diabetics suffer from diabetic retinopathy. Currently, in India technicians capture the images and rely on the highly-trained doctors to diagnose diabetic retinopathy by studying the fundus images whereas manually checking the fundus images is time consuming and costly. To overcome this, Aravind Eye Hospital in India wants to automatically screen the images of the disease and determine the condition severity

Proposed Solution:

In this process, Kaggle has created a competition Asia Pacific Tele-Ophthalmology Society (APTOS) sponsored blindness detection, California Health Care Foundation [CHCF] sponsored Diabetic Retinopathy Detection competition by providing around 3662 scans of eye fundus to train and test the model.

My solution to the above problem is to build a convolution neural network model to take the image as input and determine the severity of the disease.

Implementation:

Data:

I have used Kaggle dataset which is of 9.52GB in size and contains 3662 train and test fundus images. Along with images there is train.csv file which gives target label(diagnosis column) for each image present(id_code column) in train images directory.

There are 5 target labels ranging from 0 to 4 which tells the severity of diabetic retinopathy by seeing the fundus images. These numerical labels are encoded as categories which are given below:

{0 : 'No DR',

1 : 'Mild',

2 : 'Moderate',

3 : 'Severe',

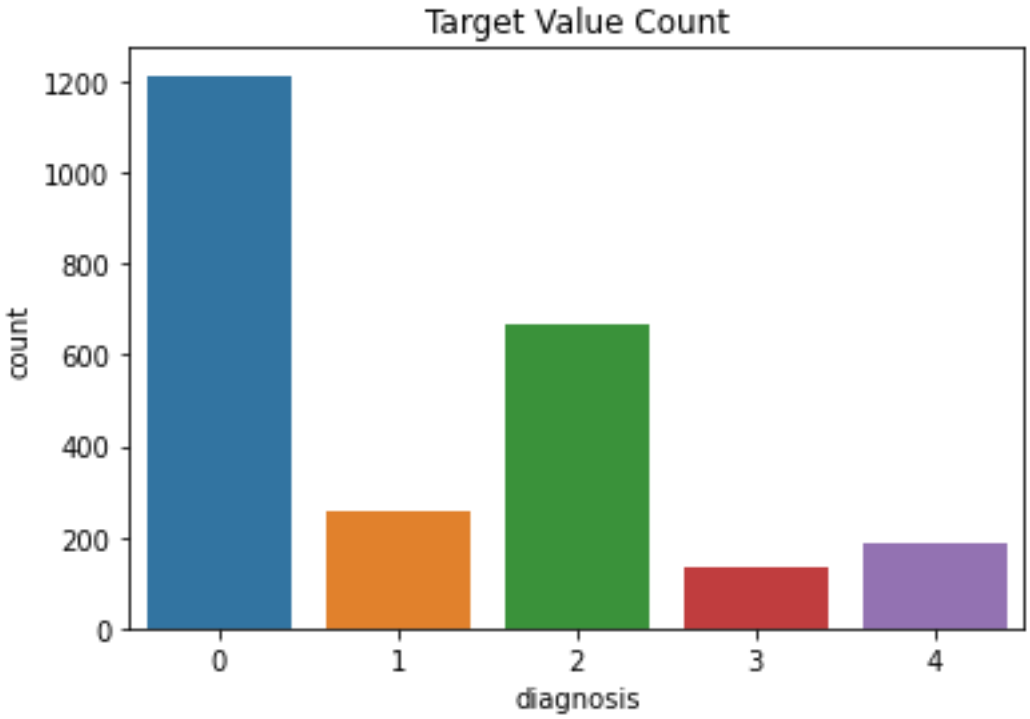
4 : 'Proliferative DR'}

To do this categorical encoding I have written encode_categorical function

```
def encode_categorical(x):
  labels = {0 : 'No DR',1 : 'Mild', 2 : 'Moderate',3 : 'Severe',4 : 'Proliferative DR'}
  return labels[x]
```

This function creates a new column encoded_diagnosis corresponding category for every row of the label column in the train_df(after reading train.csv file).

The below graph shows the target column distribution:



From the graph, we can see that there are around 1200 images of training data for 0 label which is No Diabetic Retinopathy images. This shows that data is slightly imbalanced in the target values.

Let us also see how the training images look like to do so I have written plot_batch_images function which takes ImageDataGenerator as input and plot 20(here 20 batch size) images:

```
def plot_batch_images(generator):
  fig=plt.figure(figsize=(8, 8))
  for i,j in generator:
    for im in range(len(i)):
      fig.add_subplot(4, 5, im+1)
      plt.imshow(i[im])
    break
  plt.show()
```

The above image shows the function description for plot_batch_images.

Now, we have dataframe in which the id of training images and their label is present. To read these images and do some preprocessing, I have used ImageDataGenerator from keras to perform necessary pre-processing and data augmentations if required.

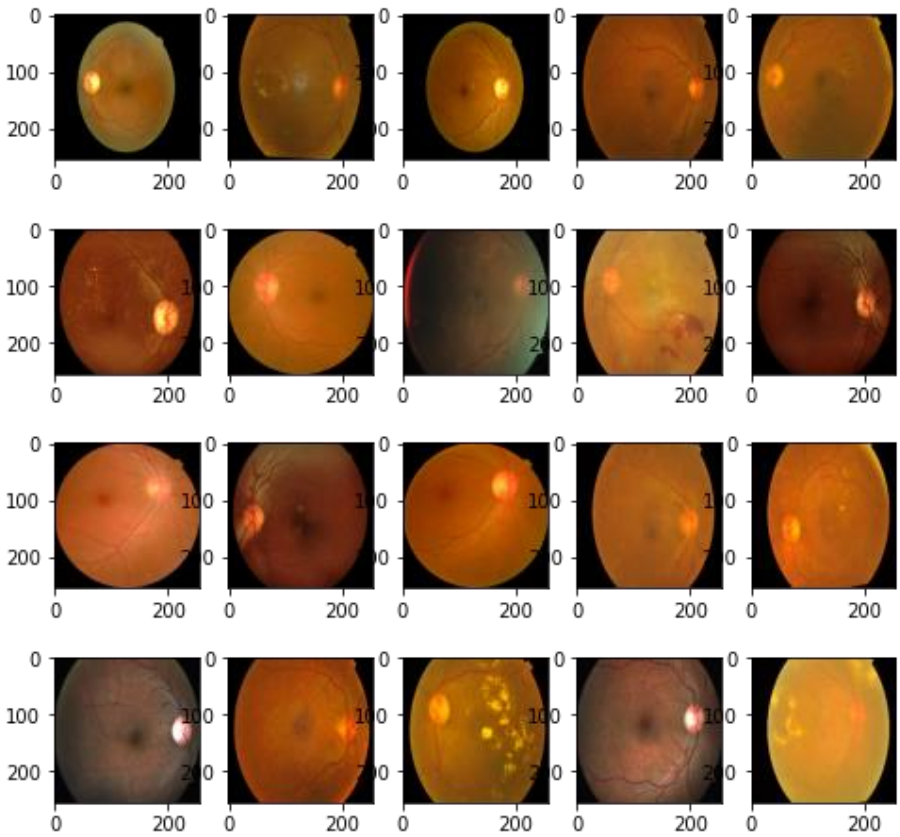
I have performed stratified split the training data to training and validation splits in which validation split is about 0.33 size of training data available. As the split is stratified sampling the classes are equally distributed in training, validation data so that model wont be biased for a single class.

```
train_datagen=ImageDataGenerator(rescale=1./255)
test_datagen=ImageDataGenerator(rescale=1./255)
val_datagen=ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_dataframe(dataframe = train_df,x_col='path',y_col='encoded_diagnosis',class_mode='categorica
val_generator = val_datagen.flow_from_dataframe(dataframe = val_df,x_col='path',y_col='encoded_diagnosis',class_mode='categorical',bat
test_generator = test_datagen.flow_from_dataframe(dataframe = test_df,x_col='path',y_col='encoded_diagnosis',class_mode='categorical',
```

```
Found 2453 validated image filenames belonging to 5 classes.
Found 1209 validated image filenames belonging to 5 classes.
Found 1928 validated image filenames belonging to 1 classes.
```

The above code snippet is ImageDatagenerator for train, val and test data. There is no data augmentation included here only pre-processing is done by dividing 255, which the converted image array contains values ranging from 0 to 1 and target values are in the form categorical encoded labels. As we can see after splitting the training we have 2453 images for training and 1209 for validation data.

Now, using the plot_batch_images function I have plotted single batch of training images:



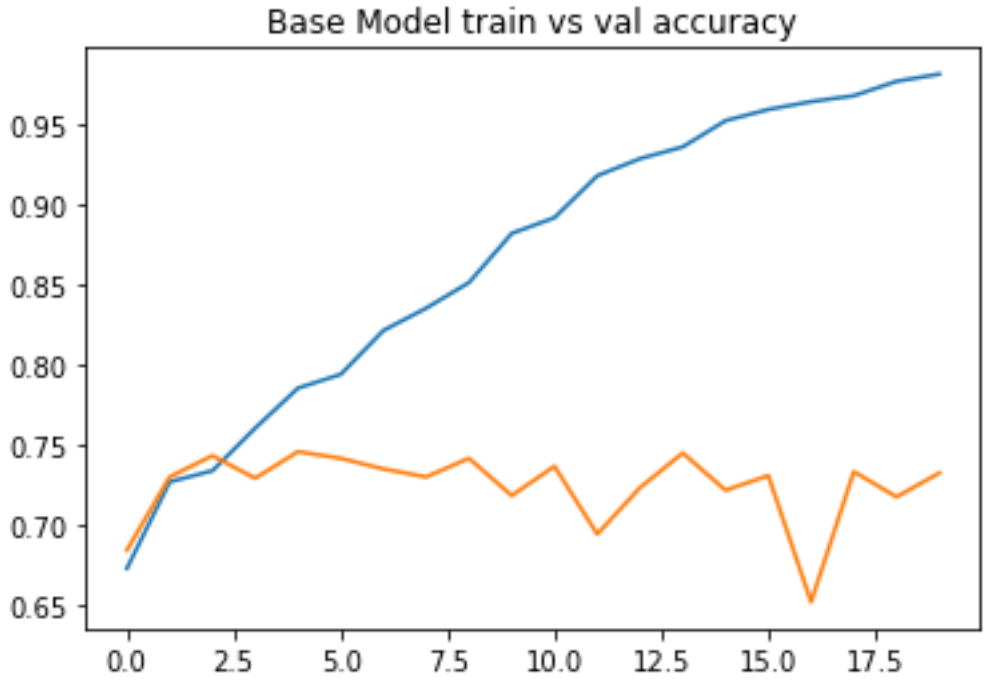
The above images are different condition of diabetic retinopathy. We can see that there are different colors, contrast and brightness of images available.

Model Building:

I have built a base model with couple of conv2d layers, maxpooling layers and fully connected layer for the prediction with softmax activation. The architecture is shown below:

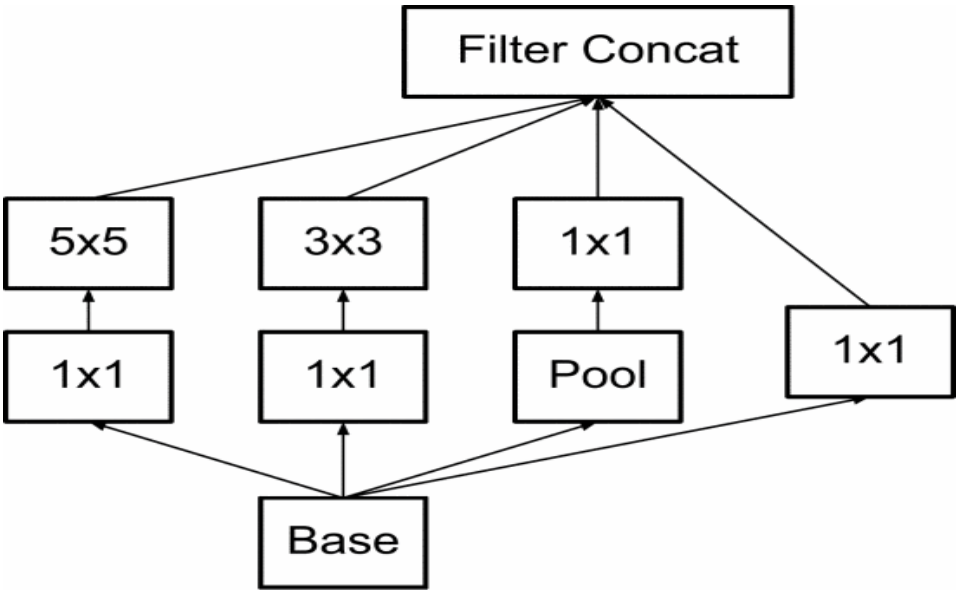
```
[ ]  model = Sequential()
     model.add(Conv2D(64,(3,3),activation='relu',input_shape=(256, 256, 3)))
     model.add(MaxPooling2D((2,2)))
     model.add(Conv2D(128,(3,3),activation='relu'))
     model.add(MaxPooling2D((2,2)))
     model.add(Flatten())
     model.add(Dense(512,activation='relu'))
     model.add(Dense(5,activation='softmax'))
```

This baseline model has around 70% accuracy which is highly overfitted as we can see that from first epoch itself the validation accuracy is higher than training and after few epochs the training score went high and validation score is very low.

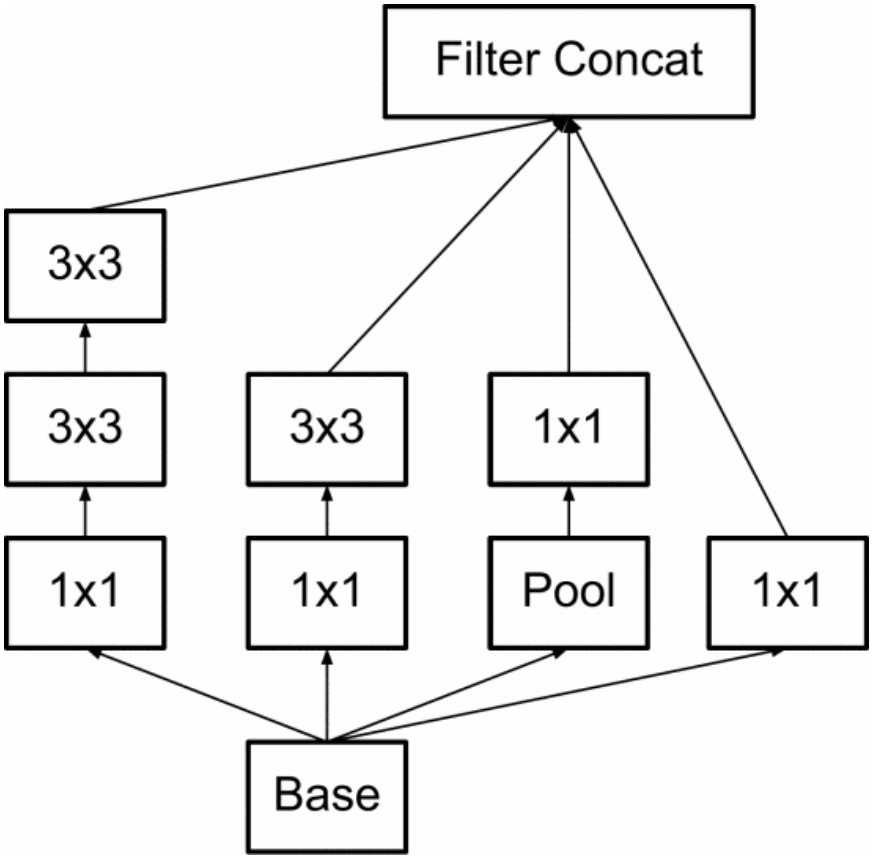


Custom Inception V3 :

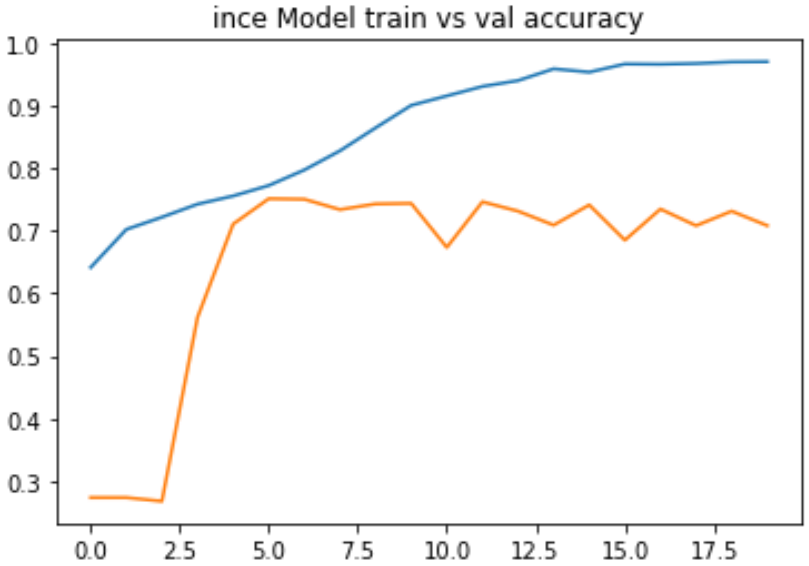
To handle this, we need some sophisticated model architectures. So I have considered building similar to Inception v3 architecture, which is version of GoogleNet architecture. Basic Inception architecture idea, figure shown below. Idea is to try different convolution filters on single input that is 1x1,3x3,5x5 and concatenate the output from all these convolutions to gather more feature extractions of previous layer images input.



The reference paper ”Re- thinking the inception architecture for computer vision” suggested that replacing the 5 x 5 convolution by series of two 3x3 convolutions would gather similar features from the images and also reduce the number of computations. As the 5x5 convolutions is 25/9 2.98 more computationally expensive than 3x3 convolutions. So the 5x5 convolutions are replaced by 2 layer 3x3 convolutions as shown below. The below image is same architecture as above but replaced by 2 3x3 convolution filters.

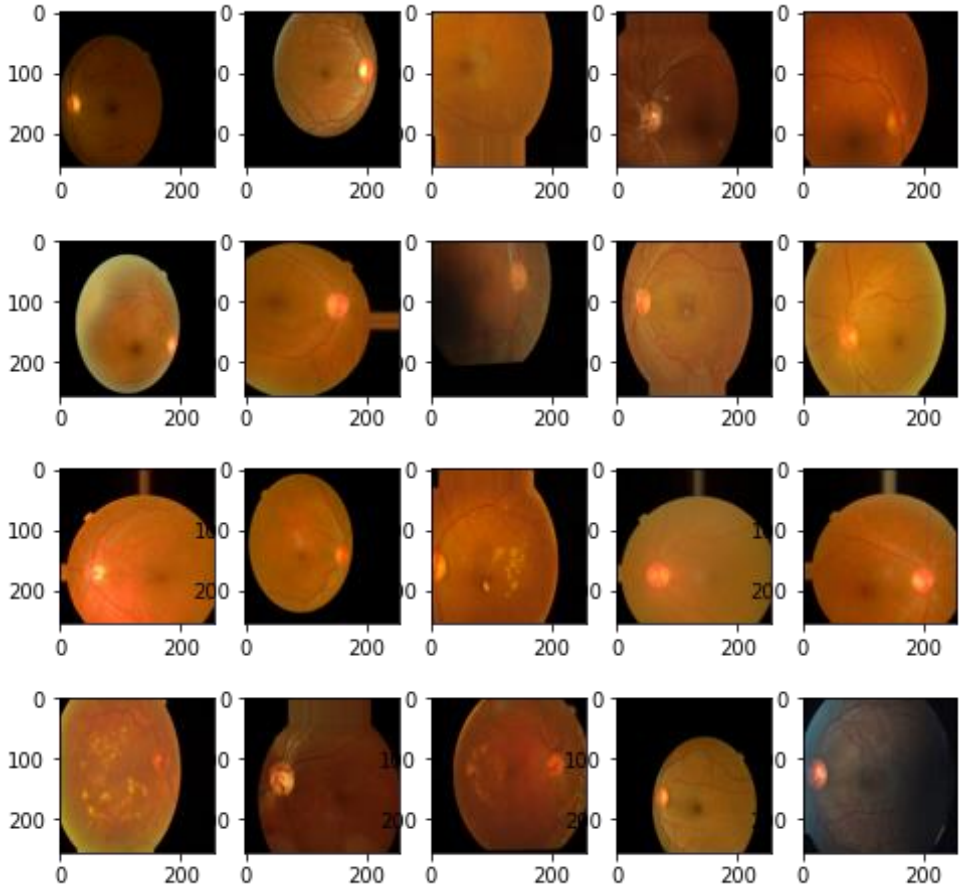


And, we can even further expand the replacement by using 1 x n and n x 1 convolution on the deeper layers when the grid size is between 12 and 20. This improves the performance though computationally expensive.



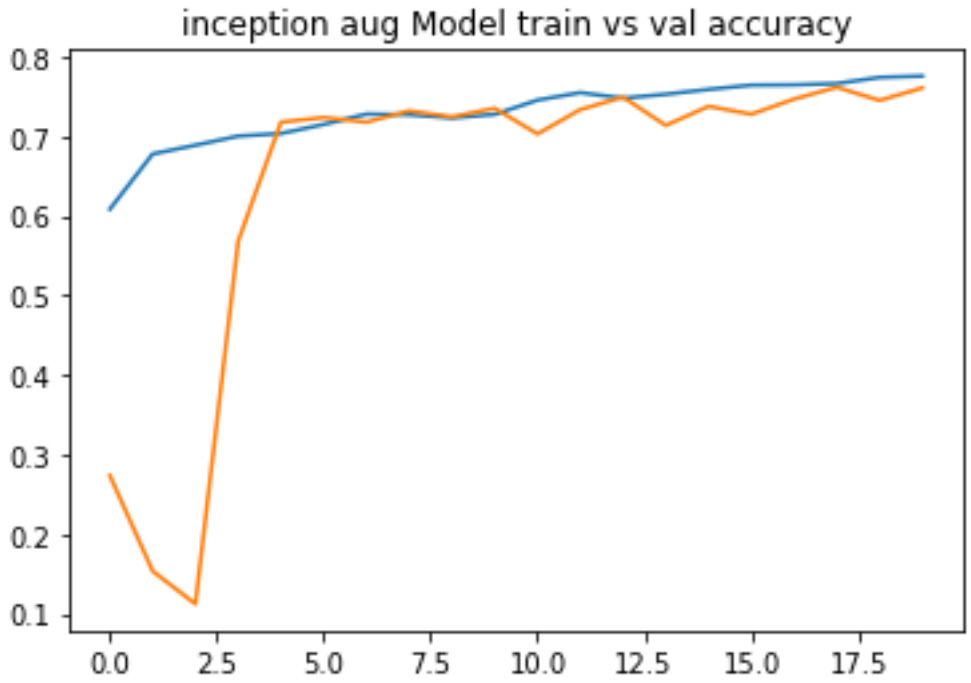
This custom inception model has around 30 million trainable parameters. I have trained model for 20 epochs on the same training data and achieved 75% accuracy before the model starts overfitting. The above graph is the training vs validation accuracy for the custom inception model built.

To increase the accuracy and reduce the overfitting I have performed some data augmentations using ImageDataGenerator. I have applied width shift, height shift and horizontal flip on each of images and after the augmentations the images are shown below.



As you can see the images are augmented, in this way the model can be trained in different scenarios and achieve higher model performance.

Now on the same model architecture using this images I have trained model for 20 epochs and achieved around 80% accuracy without any further overfitting.



As we can see data augmented model performed better with 80% accuracy and if we train even further the model accuracy might increase more.

I have also trained the model using pre – trained model available in Keras package. I am able to achieve 75% accuracy but with some overfitting.

Results and Discussion:

Model	**Accuracy**
Base Model	70%
Custom Inception v3	75%
Custom inception v3 – data augmented	80%

Of all models custom inception model with data augmentation performed very well with 80% accuracy.

References:

- Kaggle Dataset : APTOS 2019 Blindness Detection | Kaggle
- Reference paper to build custom inception v3 model : ”Re- thinking the inception architecture for computer vision”.
- Keras inception - https://keras.io/api/applications/inceptionv3/