Rakshith Churchagundi Amarnath

CS 512 – Computer Vision

CWID: A20424771

Assignment 4 – CNN

Deliverable 1:

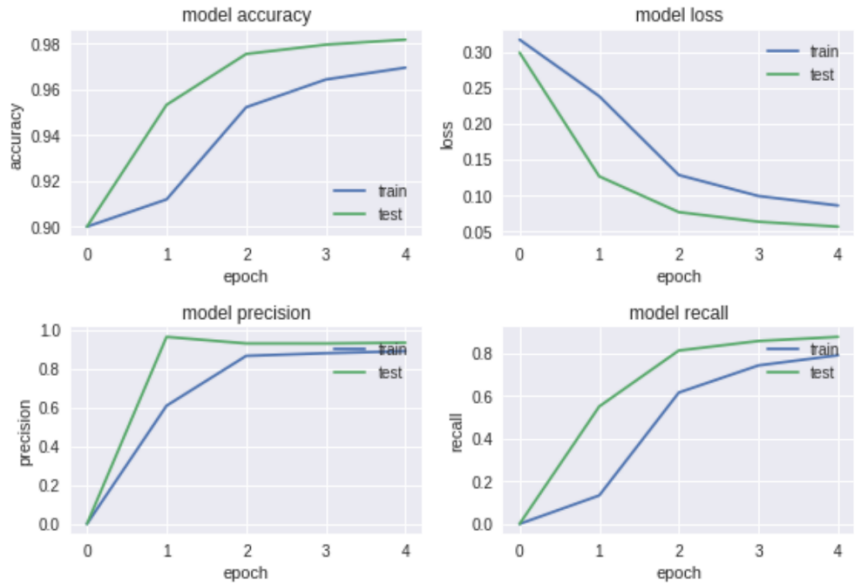
The MNIST (Modified National Institute of Standards and Technology) database is a large set of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning and computer vision. The black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

Here, we are taking this as the primary input to our program of (CNN) conventional neural network (CNN). The following implementations are done to our network.

- Training it using the MNIST data set
- Differentiating between training set - A set of examples used for learning, that is to fit the parameters of the classifier and test set - A set of examples used only to access the performance of a fully specified classifier.
- Adding convolution 2D layers and pool layers with specified filter and kernel size.
- Computing cross entropy, changing the drop rate
- Optimizing using gradients to a specific learning rate

The model training is done in 5 iterations and the parameters such as loss, accuracy, precision, recall will vary accordingly and its show below:

```
Test loss: 0.05622675534486771
Test accuracy: 0.981590002822876
Test precision: 0.9336127089500428
Test recall: 0.8774
```



The results for every iterations are as follows:

1st Iteration

- 12s - 195us/step - loss: 0.3170 - acc: 0.9000 - precision: 0.0000e+00 - recall: 0.0000e+00 - val_loss: 0.2989 - val_acc: 0.9000 val_precision: 0.0000e+00 - val_recall: 0.0000e+00

2nd Iteration

- 8s 140us/step - loss: 0.2381 - acc: 0.9119 - precision: 0.6092 - recall: 0.1333 - val_loss: 0.1264 - val_acc: 0.9532 - val_precision: 0.9638 - val_recall: 0.5508

3rd Iteration

- 8s 135us/step - loss: 0.1282 - acc: 0.9521 - precision: 0.8659 - recall: 0.6162 - val_loss: 0.0765 - val_acc: 0.9753 - val_precision: 0.9299 - val_recall: 0.8134

4th Iteration

- 8s 138us/step - loss: 0.0990 - acc: 0.9642 - precision: 0.8798 - recall: 0.7435 - val_loss: 0.0631 - val_acc: 0.9794 - val_precision: 0.9298 - val_recall: 0.8580

5th Iteration

- 8s 135us/step - loss: 0.0857 - acc: 0.9694 - precision: 0.8904 - recall: 0.7911 - val_loss: 0.0562 - val_acc: 0.9816 - val_precision: 0.9337 - val_recall: 0.8774

The total values will be as follows:

Test loss: 0.05622675534486771
Test accuracy: 0.981590002822876
Test precision: 0.9336127089500428
Test recall: 0.8774

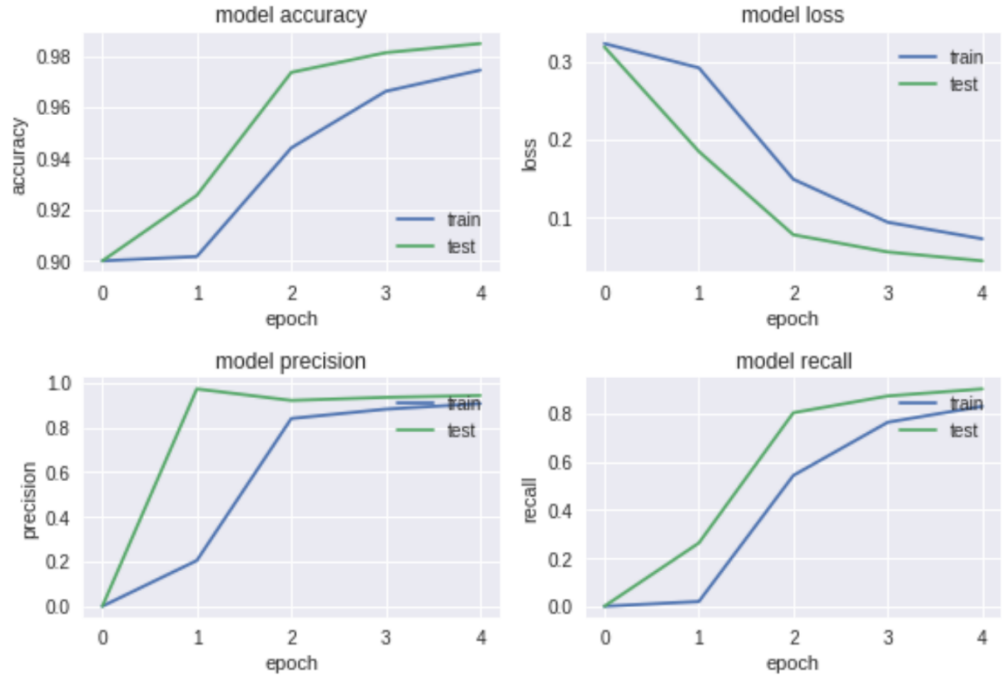
Deliverable 2: Parameter tuning

i. Changing the network architecture- Results after adding another layer to our model using

model.add(Conv2D(128, kernel_size=(5, 5) activation='relu', input_shape=input_shape))
model.add(Conv2D(128, (5,5), activation='relu'))

As we can see in the plots, the loss decreased when compared to the results we got using 2 filters, after adding the third, the total recall increased by 0.1

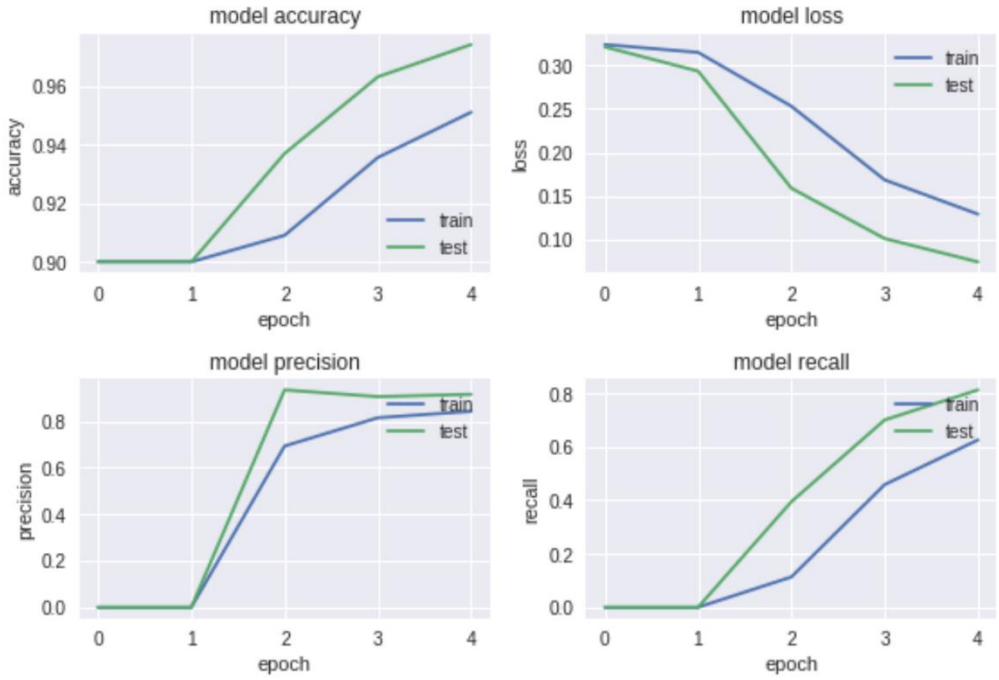
Test loss: 0.04531613793373108
Test accuracy: 0.9848300006866455
Test precision: 0.9432870603561402
Test recall: 0.9019



ii. **Output after changing the receptive parameters and adding strides** – An integer or a tuple of a single integer which specifies the stride length of the convolution. The stride values are set to (1,1) and following changes can be observed in the following graph plots.

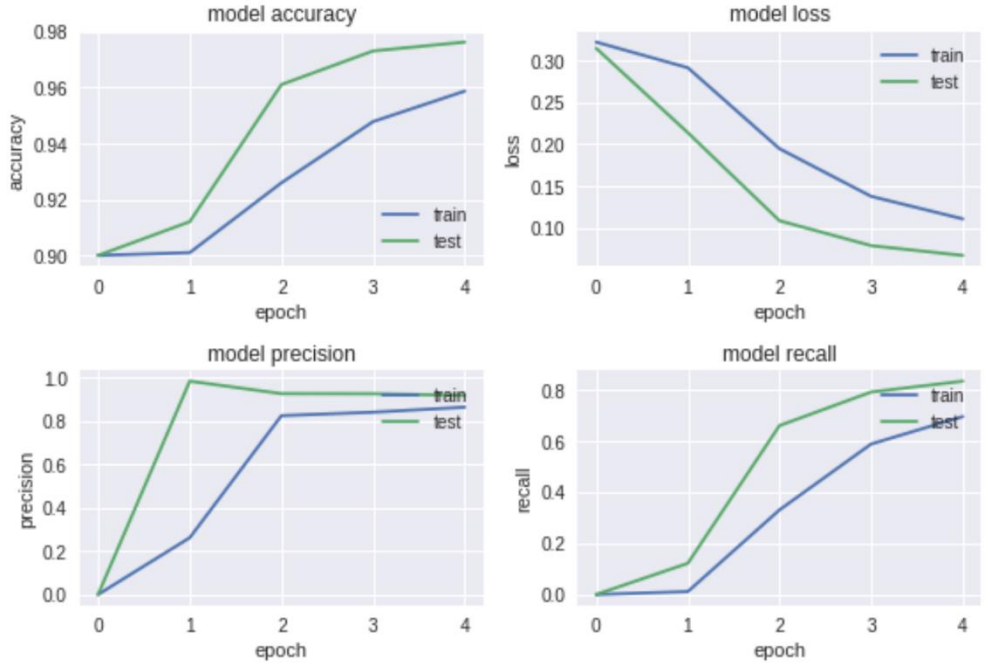
The accuracy, total recall of the model will be stagnant for the first iteration for both test and train set.

```
Test loss: 0.07467112967669964
Test accuracy: 0.973970000076294
Test precision: 0.9161312808990478
Test recall: 0.8124
```



iii. **Changing optimizers – Various parameters are changed to train the model giving more preferences to one parameter also.**

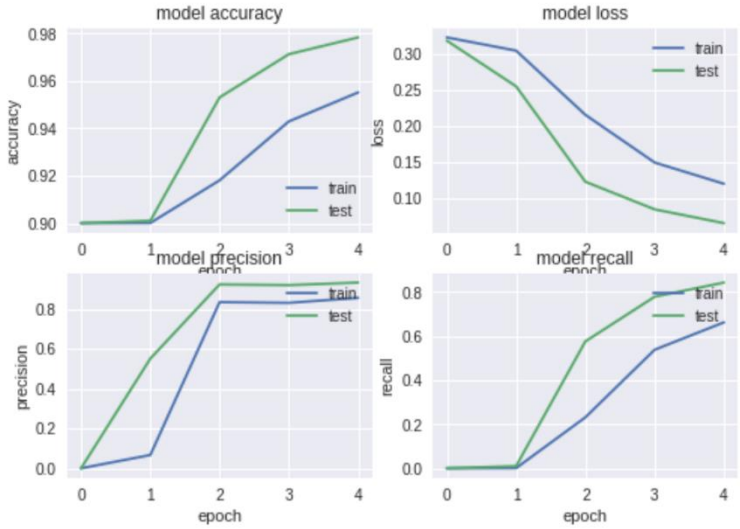
```
Test loss: 0.06761271526813507
Test accuracy: 0.9762500012397766
Test precision: 0.9194893237113952
Test recall: 0.8344
```



iv. **Using weight initializers**

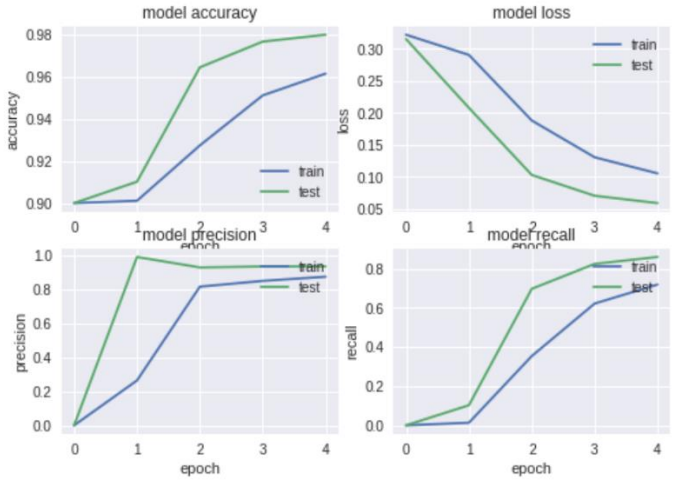
model.add(Dense(177, activation='relu'))

model.add(Dense(64, kernel_initializer = 'random_uniform', bias_initializer)

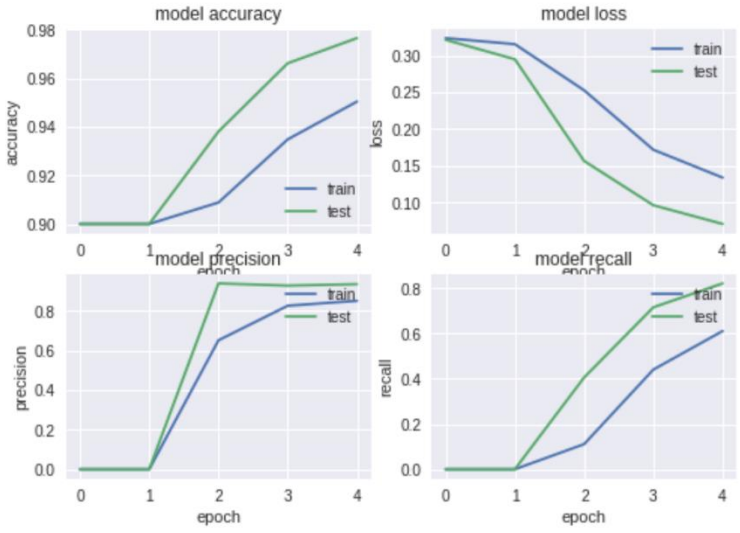


Using he normalization

keras.initializers.he_normal(seed=None)

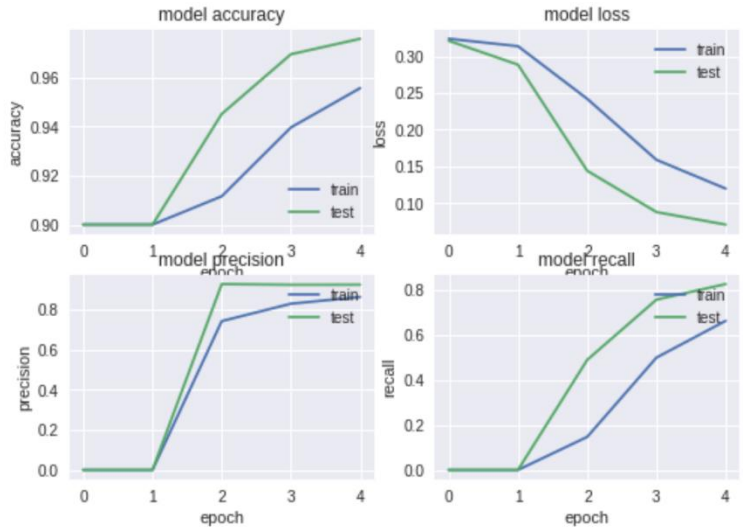


v. Batch normalization



vi. **VGG16** -Pretrained Model Concept used

keras.applications.vgg16.VGG16(include_top=True, weights='imagenet', input_tensor=None, input_shape=None, pooling=None, classes=1000)



3. Deliverable 3: Application

i. Accepting the image –

imge=cv2.imread(path,1)
The image path is set and its given as an input with the above command,

ii. imge = cv2.resize(imge, (28,28))
lower_reso = cv2.resize(imge, (28,28))

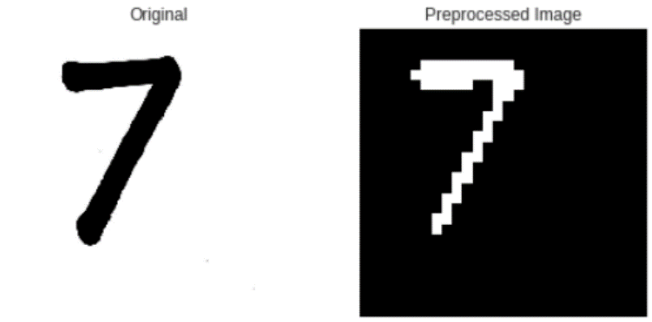
The image is being resized so that It will be fit like other images in the database using this above command.

iii. Converting image from GrayScale to Binary

```
gray_scale = cv2.cvtColor(lower_reso,cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray_scale,(3,3),0)

retval, threshold = cv2.threshold(blur, 70, 255, cv2.THRESH_BINARY)
```

The image is converted from a grayscale to a binary one using GaussianBlur()

iv. The original and preprocessed image displayed separately.




```
Using TensorFlow backend.
The label is:  odd
```

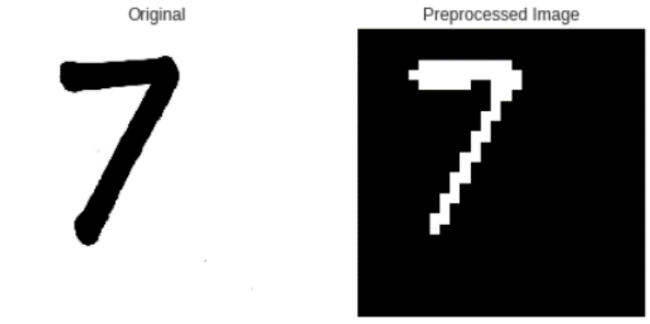
v.
```
from keras.models import load_model

model = load_model('/content/drive/My Drive/apps/cnn.h5')
```

The CNN model is being loaded to classify the images.

vi. This command allows us to provide the path of our input image so that it can be classified.

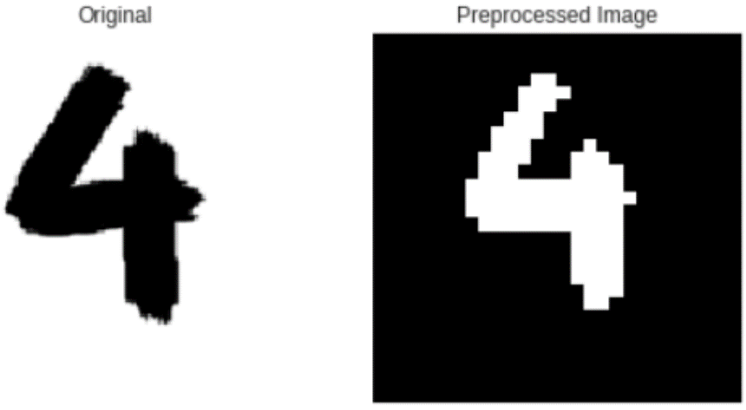
```
path = input("\nPlease enter file path or press esc or q for exit: ")
```

vii. The following images shows the model's output of classifying image




```
Using TensorFlow backend.
The label is:  odd
```

The above image shows that the hand written digit ‘7’. The output is predicted by the model as odd




```
The label is:  even
```

The above image shows that the hand written digit ‘4’. The output is predicted by the model as even.

Pressing ‘q’ will execute the quit command and the program will be terminated