CS 512 Assignment 4: Report

Fall 2018

Herick Jayesh Asmani

A20399752

<u>Framework Used:</u> Keras.

Instructions to run programs:

1. Install various python packages such as numpy, OpenCv, TensorFlow, Keras, Matplotlib.
2. To train the model and visualize various plots (accuracy, loss, precision, recall) run cnn.py.
3. To visualize the model and train and evaluation plots, in your command prompt run tensorboard -- logdir ./models. It will give you a link. Copy and past that link into your web browser to see the visualization.
4. To classify unseen digits images run cnn_test.py. and provide path of the image.

Deliverable 1:

Since we are working with MNIST datasets, the output labels are 0 – 9 but we want that given an input image, the classifier should classify the image as either even or odd. So, for the classifier to classify input image as even or odd, we map the output labels to 1s and 0s i.e. map even numbers (0, 2, 4, 6, 8) to 1 and odd numbers (1, 3, 5, 7,9) to 0. Therefore, now we have binary output labels.

Model Details:

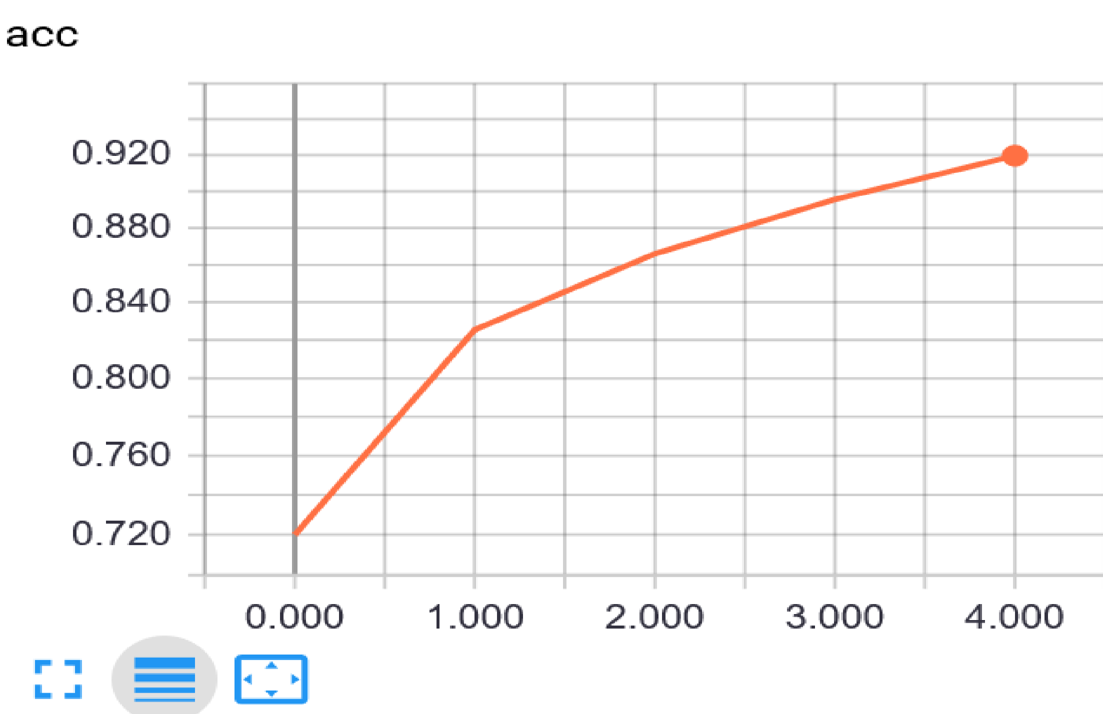
Based on the instructions given in the Assignment 4 description, following Model is built.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_2 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 102
=================================================================
Total params: 460,184
Trainable params: 460,184
Non-trainable params: 0
```

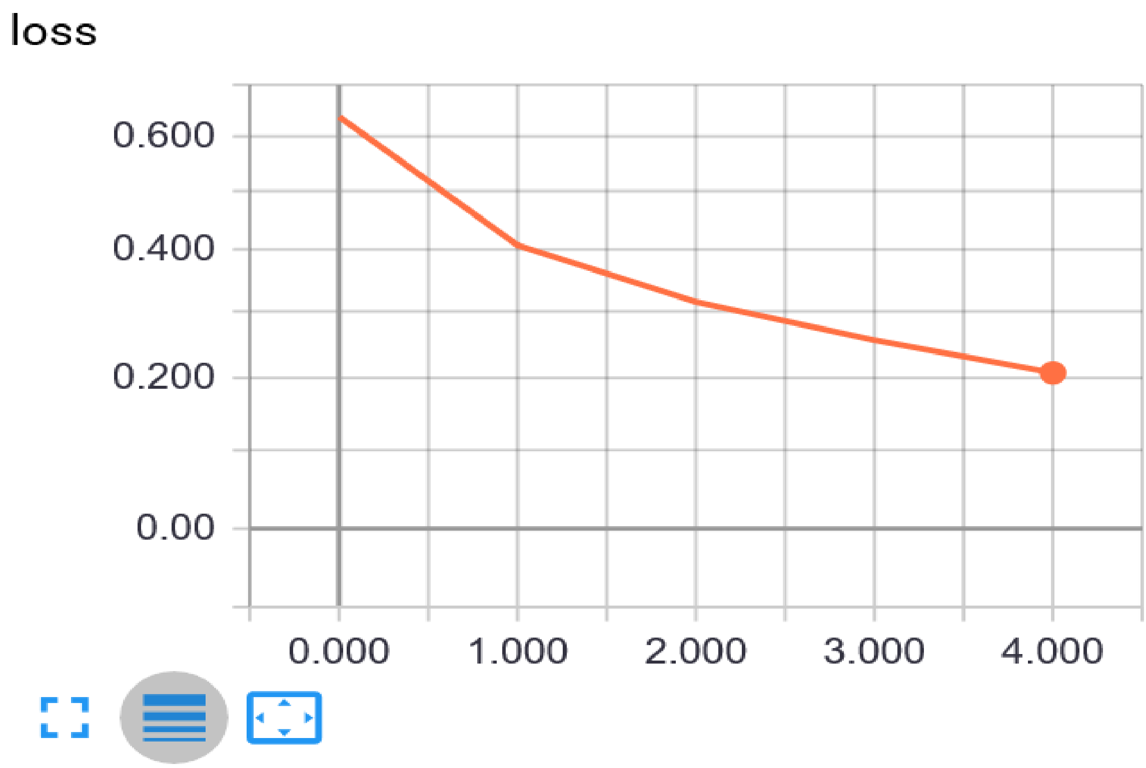
<u>Result:</u>

After compiling the above model with categorical cross entropy loss (since I am using Softmax output) and gradient descent optimizer with learning rate of 0.001. The model is trained for 5 epochs and following training and test metrics curves are obtained.

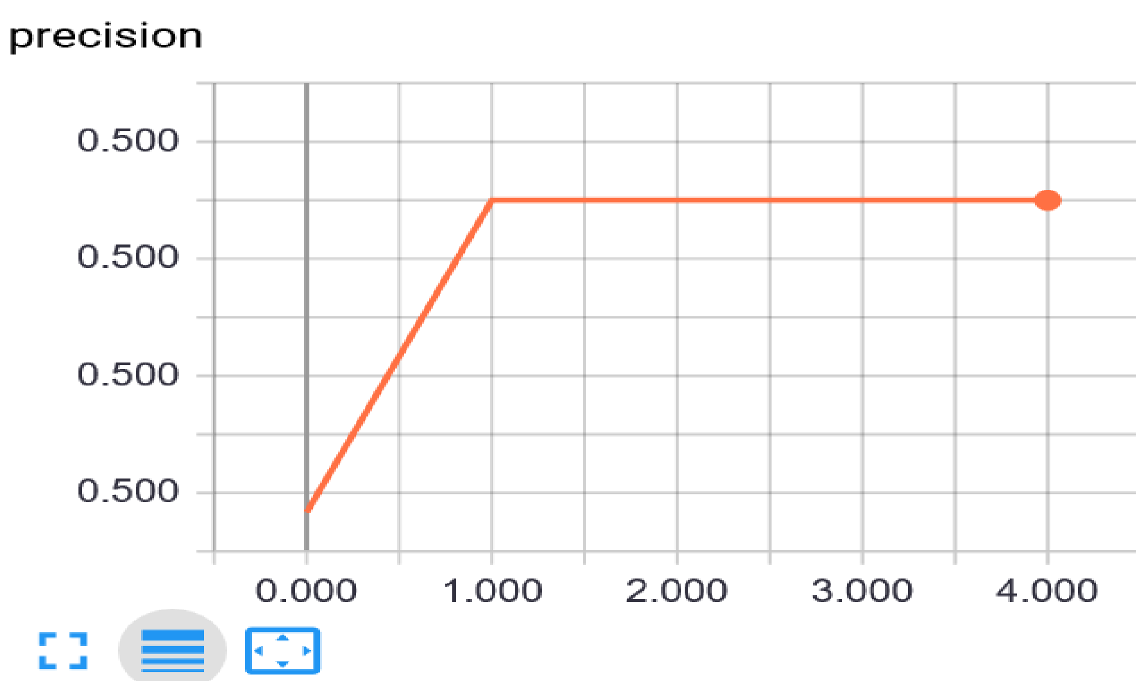
<u>Training Accuracy:</u>



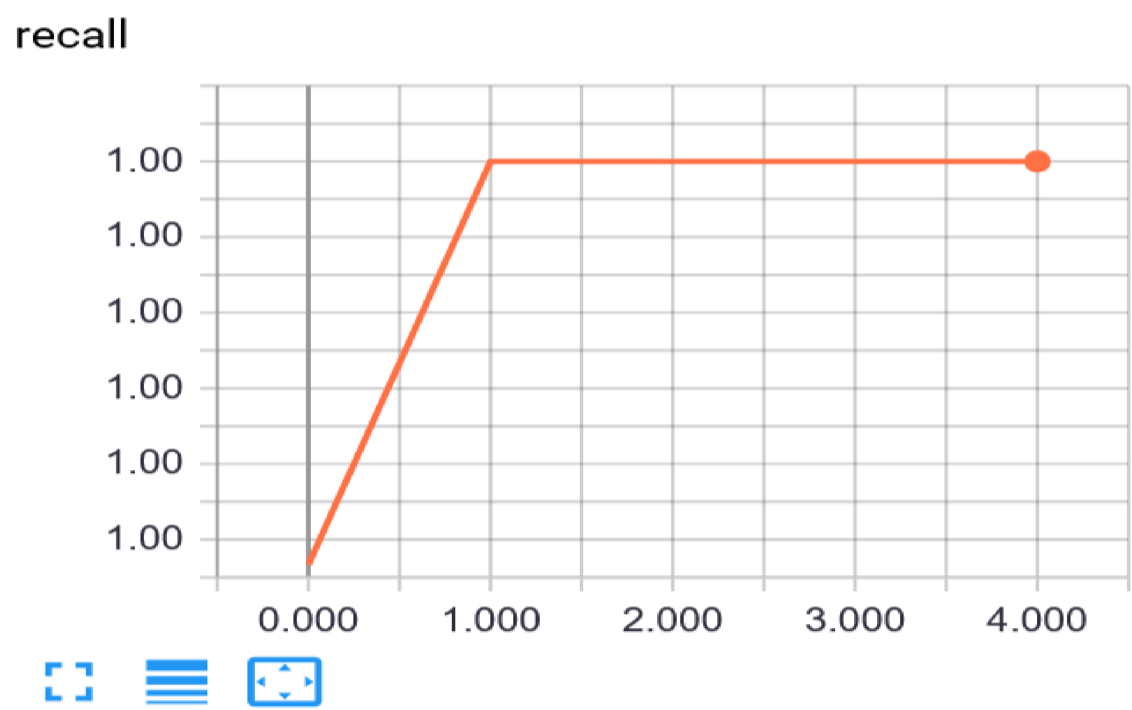
Training Loss:



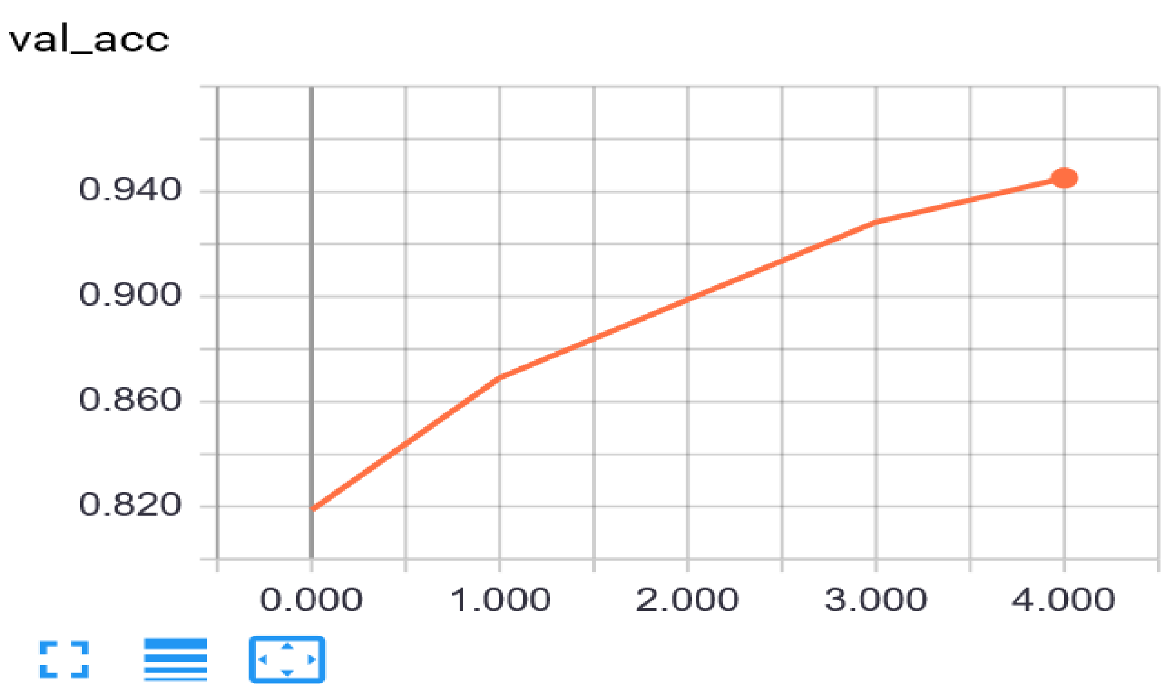
Training Precision:



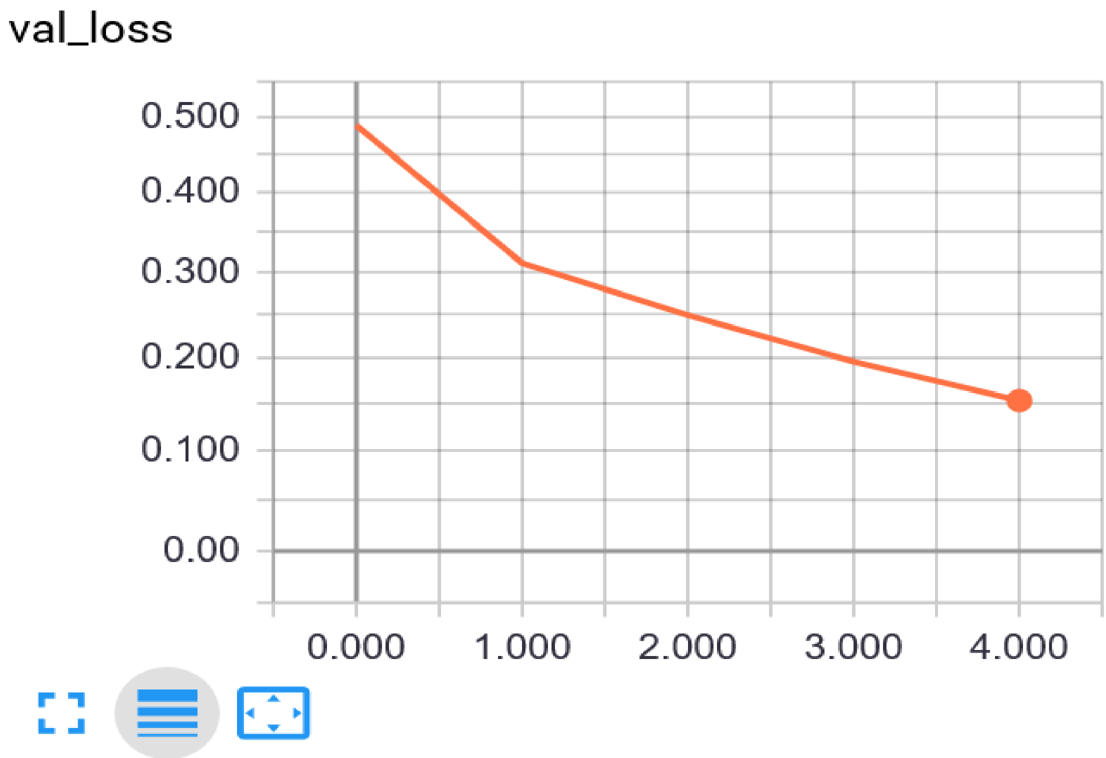
Training Recall:



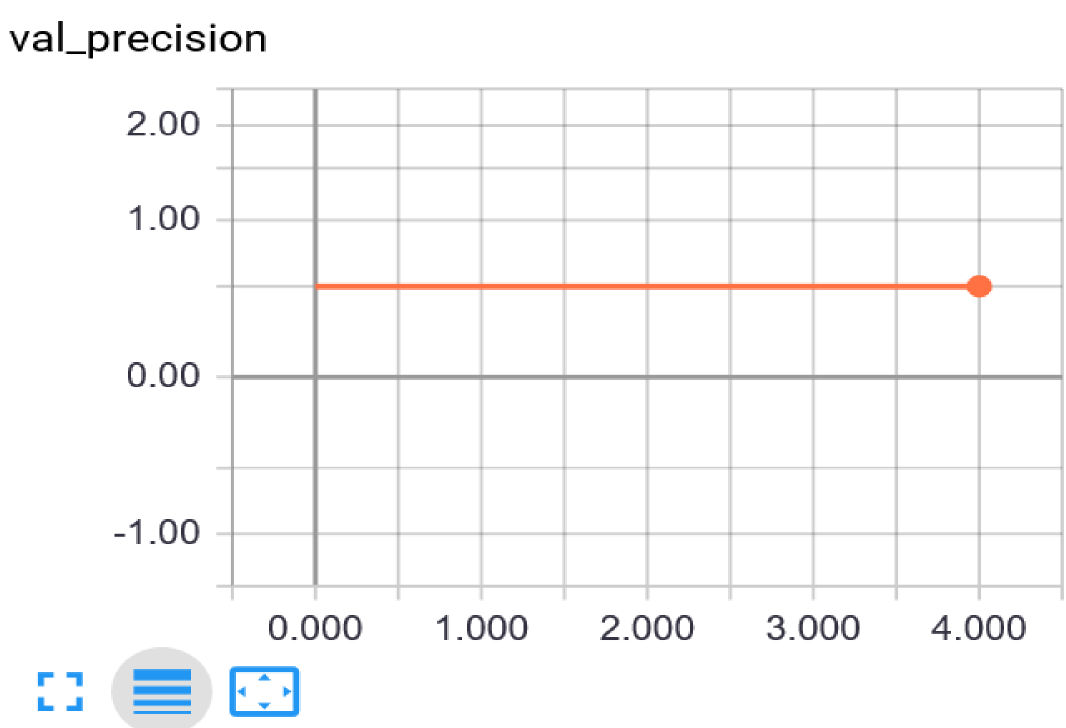
Test Accuracy:



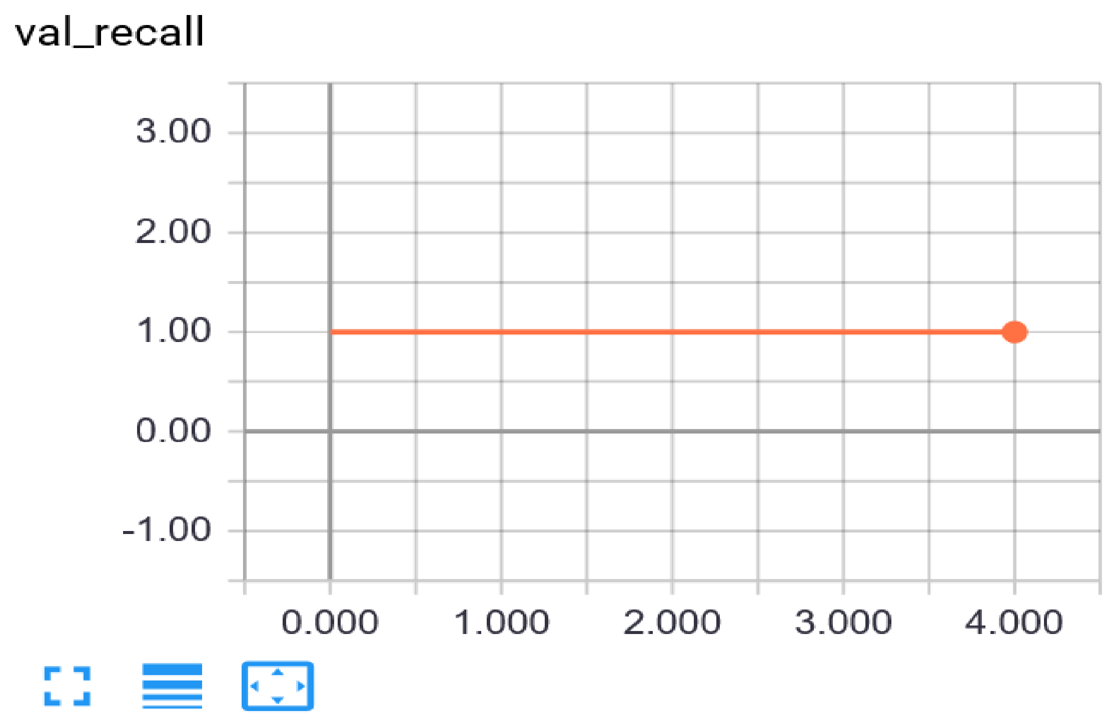
Test Loss:



Test Precision:



Test Recall:



The values of accuracy, loss, precision and recall for training and test of the above CNN model at the final step is as follows:

```
Epoch 5/5
60000/60000 [==============================] - 185s 3ms/step - loss: 0.2078 - acc: 0.9196 - precision: 0.5000 - recall: 1.000
0 - val_loss: 0.1530 - val_acc: 0.9452 - val_precision: 0.5000 - val_recall: 1.0000

Epoch 00005: val_loss improved from 0.19593 to 0.15301, saving model to ./models/model.weights.best.hdf5
Test loss:  0.1530118324995041
Test accuracy:  0.9452
Test precision:  0.5
Test recall:  1.0
```

Deliverable 3:

Algorithm Details:

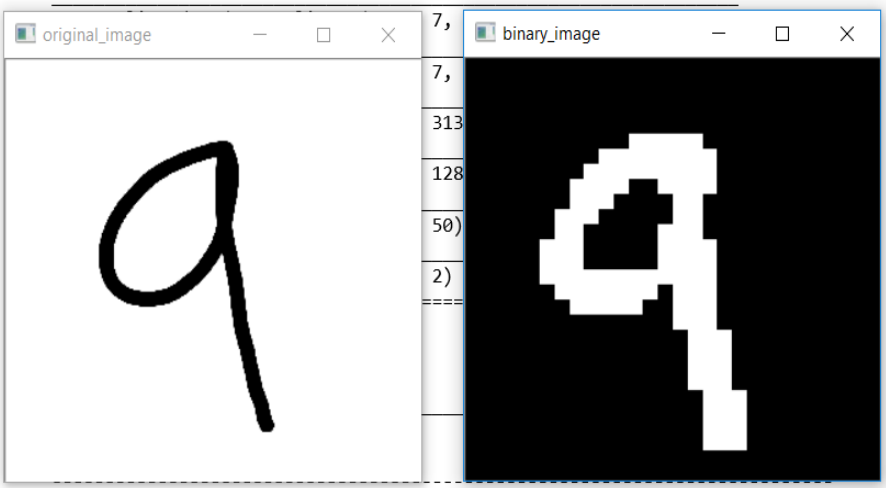
In order to classify the unseen handwritten digits in an image, we have to do some preprocessing. First, we obtain an image by giving a path of the desired image file. After obtaining the image, we resize the image to match the dimensions of the images on which we trained the CNN model i.e. we resize the image to 28x28. Now the CNN model was trained on binary images, so to classify the unseen image correctly, we first convert the RGB image to grayscale image and then apply Gaussian blur and Adaptive threshold technique to binarize the image. The algorithm calculates the threshold for a small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination. However, it thresholds the image as 0 and 255. Therefore, we binarize it by dividing the result by 255. Now the image is processed i.e. ready for classification. So, now we import the CNN Model we created in cnn.py and load its best weights for classification. Before feeding the image, we do reshaping of the image to (1, 28, 28, 1) because that's the dimension of the input, the CNN accepts and then we feed the image to this model and predict its class as either even (1) or odd (0).

Note: The unseen images used here were obtained by drawing digits on a white background using "Sketches" app from Sony.

<u>Results</u>:

The custom CNN model was able to correctly classify the digits except for some.

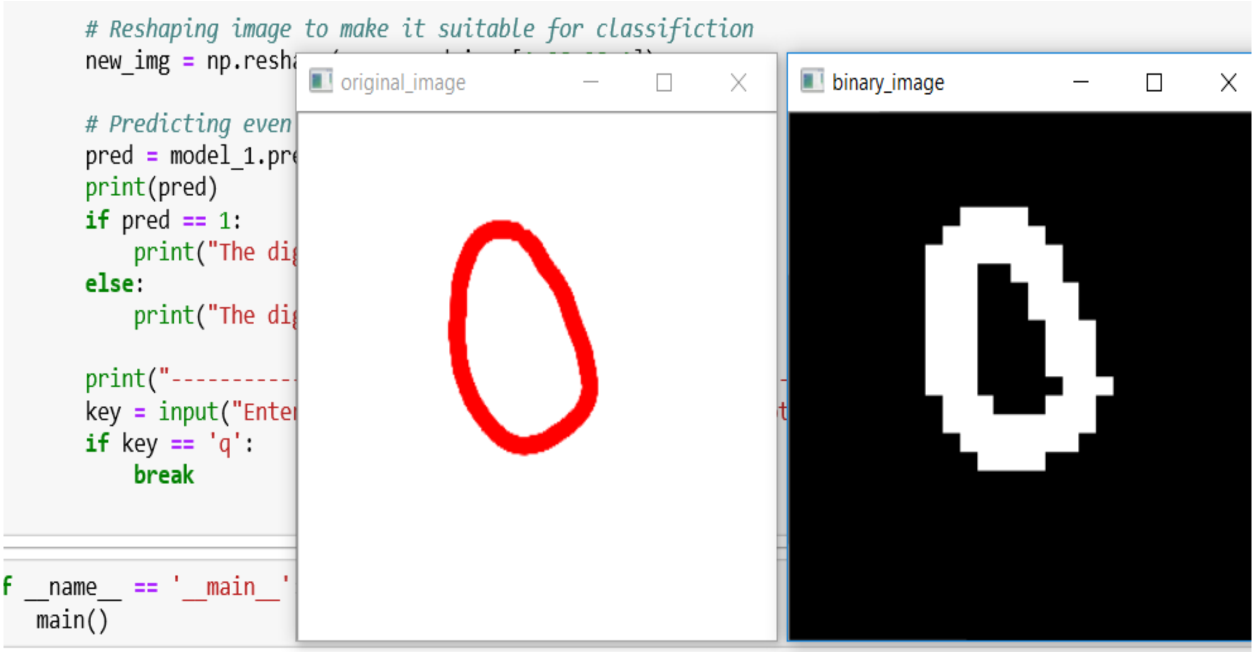
<u>9 – correctly classified as odd.</u>




```
Enter q if you want to exit else enter any other key to continue.
Type the Path of your image file: C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Assig
C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Assignments\Numbers\Sketches\Nine.png
True
```

```
Type the Path of your image file: C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Ass:
C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Assignments\Numbers\Sketches\Nine.png
True
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_5 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 102
=================================================================
Total params: 460,184
Trainable params: 460,184
Non-trainable params: 0
_________________________________________________________________
[0]
The digit is odd
------------------------------------------------------------------------
```

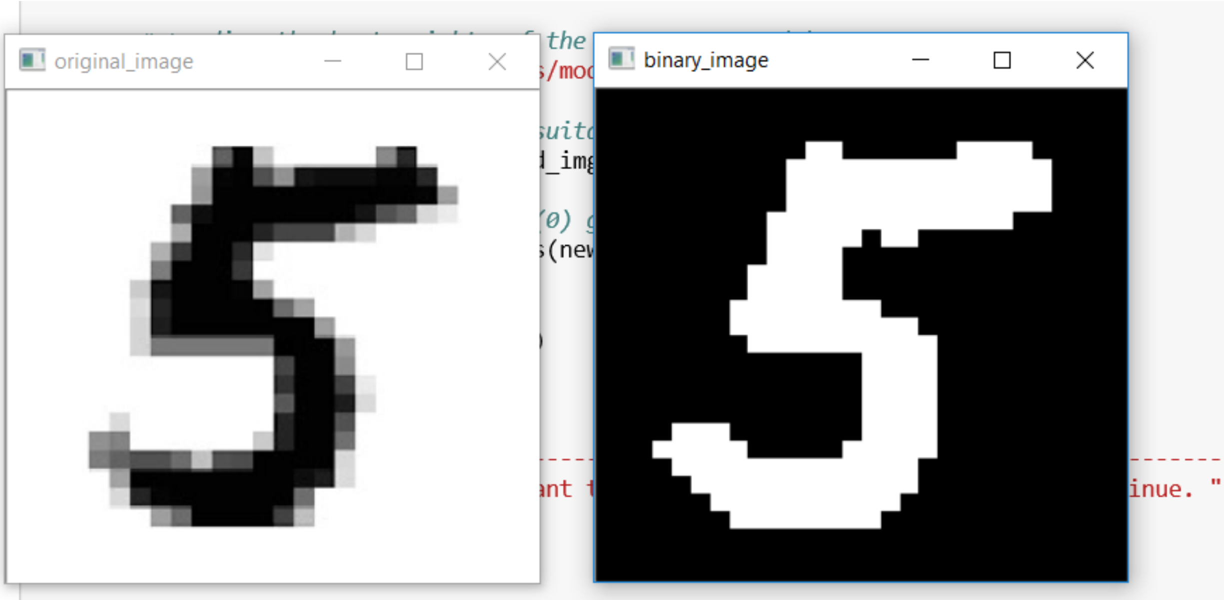
0 – correctly classified as even.




```
Type the Path of your image file: C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Assignments\Numbers
C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Assignments\Numbers\Sketches\Zero.png
True
```

```
Type the Path of your image file: C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Assi
C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Assignments\Numbers\Sketches\Zero.png
True
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_2 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 102
=================================================================
Total params: 460,184
Trainable params: 460,184
Non-trainable params: 0
_________________________________________________________________
[1]
The digit is even
```

5 – correctly classified as odd.

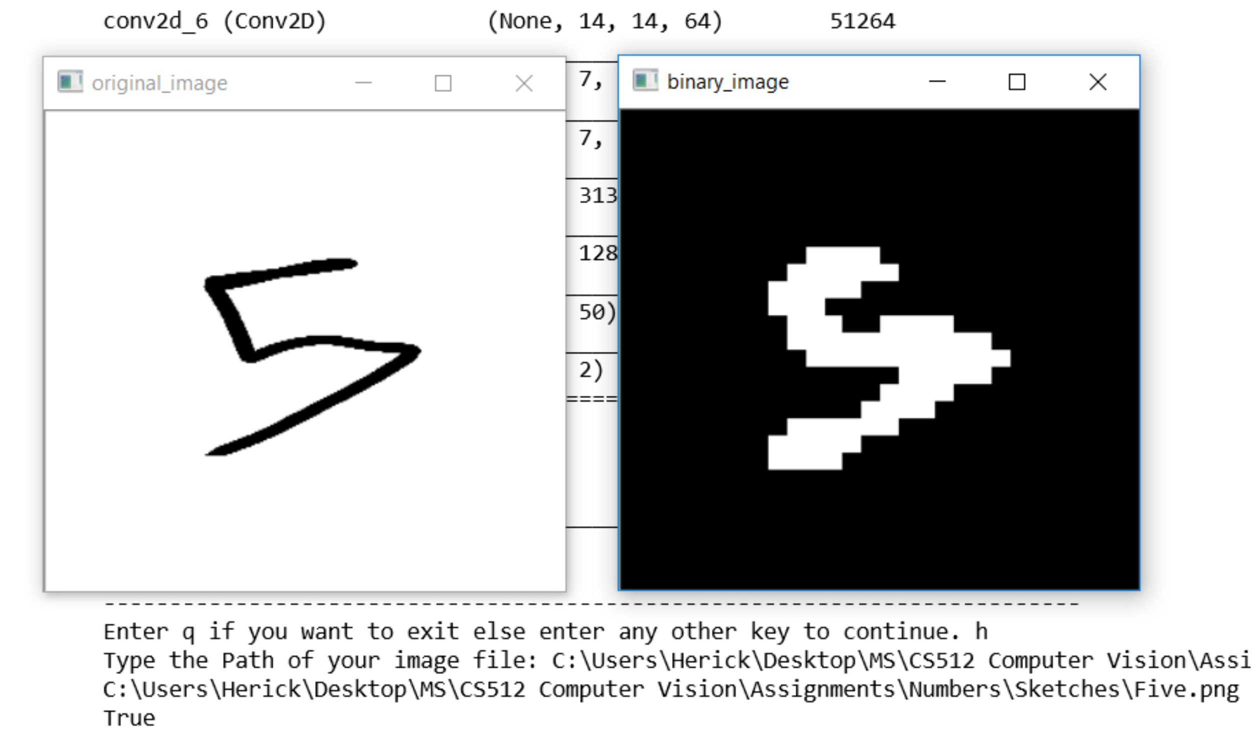



```
]: if __name__ == '__main__':
       main()
```

```
Type the Path of your image file: C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Ass
C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Assignments\Numbers\Sketches\F5.jpeg
True
```

```
Type the Path of your image file: C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Ass
C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Assignments\Numbers\Sketches\F5.jpeg
True
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_8 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 102
=================================================================
Total params: 460,184
Trainable params: 460,184
Non-trainable params: 0
_________________________________________________________________
[0]
The digit is odd
--------------------------------------------------------------------------
```

However, the below ‘5’ was incorrectly classified as even:




```
Enter q if you want to exit else enter any other key to continue. h
Type the Path of your image file: C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Ass
C:\Users\Herick\Desktop\MS\CS512 Computer Vision\Assignments\Numbers\Sketches\Five.png
True
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               401536
_________________________________________________________________
dense_11 (Dense)             (None, 50)                6450
_________________________________________________________________
dense_12 (Dense)             (None, 2)                 102
=================================================================
Total params: 460,184
Trainable params: 460,184
Non-trainable params: 0
_________________________________________________________________
[1]
The digit is even
--------------------------------------------------------------------------
```

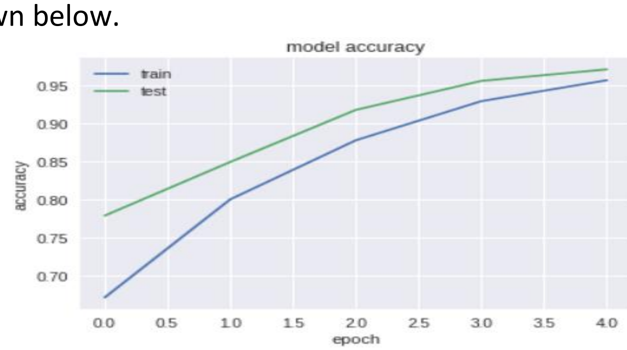
Deliverable 2:

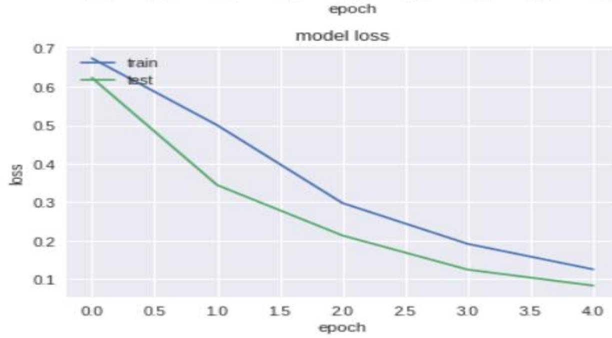
A. Changing the network architecture.

Following architecture was designed.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 128)         204928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 3, 256)         819456
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 1, 1, 256)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 1, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 102
=================================================================
Total params: 1,115,928
Trainable params: 1,115,928
Non-trainable params: 0
_________________________________________________________________
```

The above architecture resulted in increase in the number of trainable parameters compared to custom CNN model. Also its validation loss and accuracy improved as shown below.






```
[21] # load the weights that yielded the best validation accuracy
     model.load_weights('model.weights.best.hdf5')

     # evaluate and print test accuracy
     score = model.evaluate(X_test, new_Y_test, verbose=0)
     print('Test loss: ', score[0])
     print('Test accuracy:', score[1])
```

```
Test loss:  0.08361626062840223
Test accuracy: 0.9705
```

B. <u>Changing Receptive field and strides of custom CNN model.</u>

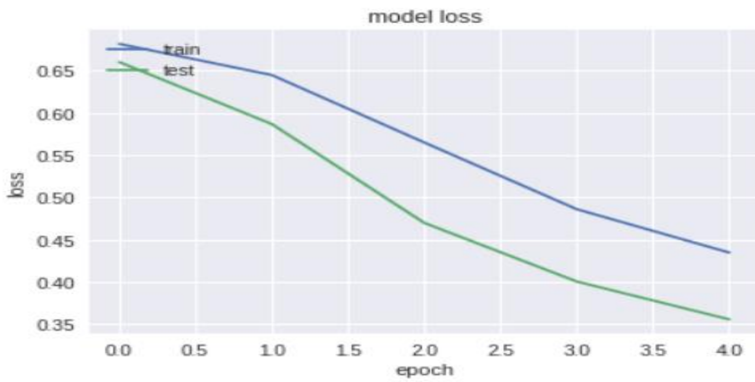
The Receptive field was set to 3 and stride was set to 2 as shown below.

```
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = 3, strides = 2, padding = 'same', activation = 'relu', input_shape = ip_shape))
model.add(MaxPooling2D(pool_size = 2, strides = 2))
#model.add(Dropout(0.4))
model.add(Conv2D(filters = 64, kernel_size = 3, strides = 2, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = 2, strides = 2))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(2, activation = 'softmax'))

model.summary()
```

This resulted in decrease in number of parameters compared to custom CNN.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 14, 14, 32)        320
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 4, 4, 64)          18496
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_8 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 102
=================================================================
Total params: 58,264
Trainable params: 58,264
Non-trainable params: 0
```

Also, accuracy and loss degraded compared to custom CNN model.

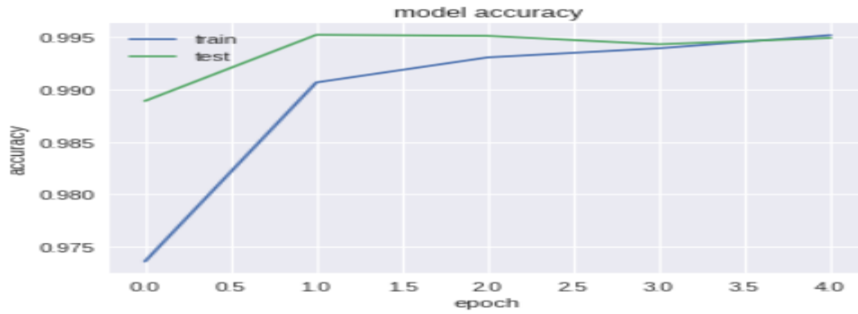


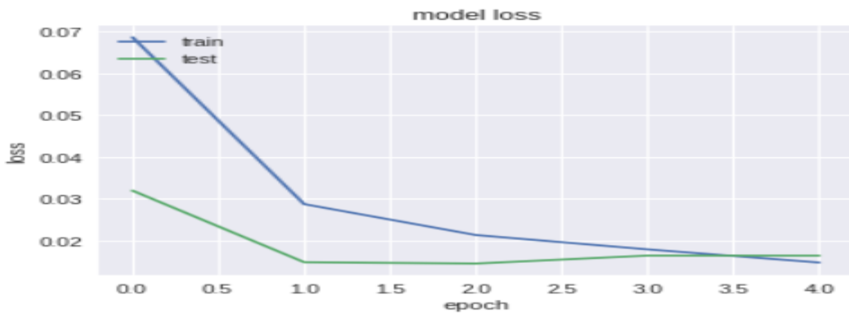

```
[26] # load the weights that yielded the best validation accuracy
     model.load_weights('model.weights.best.hdf5')

     # evaluate and print test accuracy
     score = model.evaluate(X_test, new_Y_test, verbose=0)
     print('Test loss: ', score[0])
     print('Test accuracy:', score[1])
```

```
Test loss:  0.35569756093025207
Test accuracy: 0.8472
```

C. <u>Changing the optimizer from gradient descent to Adam (lr=0.001)</u>

The accuracy and loss improved compared to custom CNN model.






```
[36] # load the weights that yielded the best validation accuracy
     model.load_weights('model.weights.best.hdf5')

     # evaluate and print test accuracy
     score = model.evaluate(X_test, new_Y_test, verbose=0)
     print('Test loss: ', score[0])
     print('Test accuracy:', score[1])
```

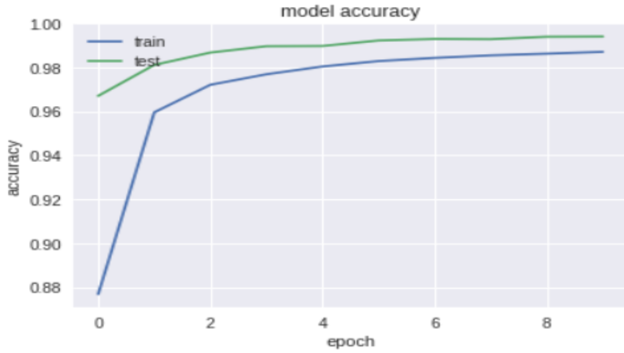
```
Test loss:  0.014513163709636137
Test accuracy: 0.9951
```

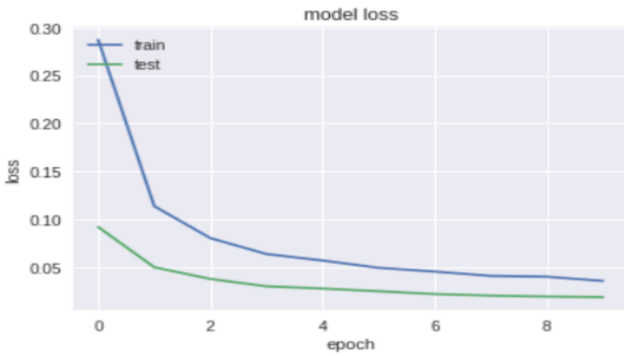
D. <u>Changing various parameters (num_filters, dropout = 0.5, lr = 0.01, epoch = 10)</u>

Following architecture is obtained.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 28, 28, 16)        416
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 14, 14, 32)        12832
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
dropout_7 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_19 (Dense)             (None, 128)               200832
_________________________________________________________________
dense_20 (Dense)             (None, 50)                6450
_________________________________________________________________
dense_21 (Dense)             (None, 2)                 102
=================================================================
Total params: 220,632
Trainable params: 220,632
Non-trainable params: 0
```

Number of parameters decreased compared to custom CNN model but accuracy and loss improved.





[57]
```
# load the weights that yielded the best validation accuracy
model.load_weights('model.weights.best.hdf5')

# evaluate and print test accuracy
score = model.evaluate(X_test, new_Y_test, verbose=0)
print('Test loss: ', score[0])
print('Test accuracy:', score[1])
```

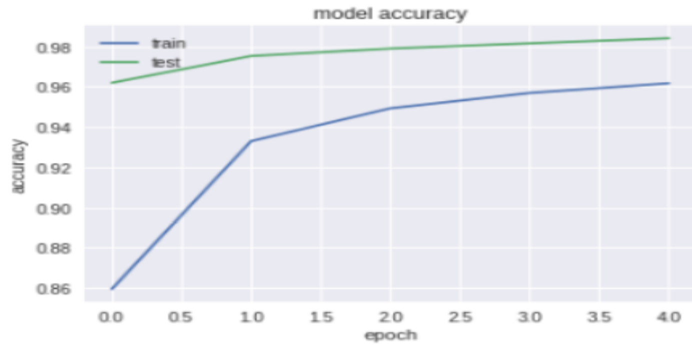
```
Test loss:  0.019207315197959544
Test accuracy: 0.994
```

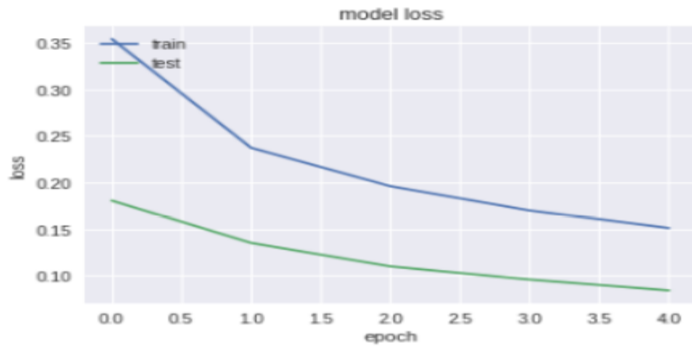
E. Adding batch normalization

Following results are obtained.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_7 (Conv2D)            (None, 28, 28, 32)        832       
_________________________________________________________________
batch_normalization_11 (Batc (None, 28, 28, 32)        128       
_________________________________________________________________
activation_11 (Activation)   (None, 28, 28, 32)        0         
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 64)        51264     
_________________________________________________________________
batch_normalization_12 (Batc (None, 14, 14, 64)        256       
_________________________________________________________________
activation_12 (Activation)   (None, 14, 14, 64)        0         
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 7, 7, 64)          0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 7, 7, 64)          0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 3136)              0         
_________________________________________________________________
dense_10 (Dense)             (None, 128)               401536    
_________________________________________________________________
batch_normalization_13 (Batc (None, 128)               512       
_________________________________________________________________
activation_13 (Activation)   (None, 128)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 50)                6450      
_________________________________________________________________
batch_normalization_14 (Batc (None, 50)                200       
_________________________________________________________________
activation_14 (Activation)   (None, 50)                0         
_________________________________________________________________
dense_12 (Dense)             (None, 2)                 102       
_________________________________________________________________
batch_normalization_15 (Batc (None, 2)                 8         
_________________________________________________________________
activation_15 (Activation)   (None, 2)                 0         
=================================================================
Total params: 461,288
Trainable params: 460,736
Non-trainable params: 552
_________________________________________________________________
```

```
Epoch 00005: val_loss improved from 0.09590 to 0.08429, savi
```






```
[38] # load the weights that yielded the best validation accuracy
     model.load_weights('model.weights.best.hdf5')

     # evaluate and print test accuracy
     score = model.evaluate(X_test, new_Y_test, verbose=0)
     print('Test loss: ', score[0])
     print('Test accuracy:', score[1])
```

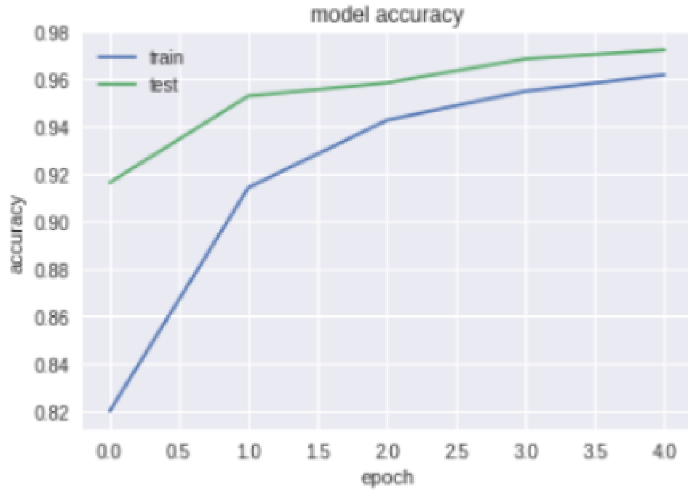
```
Test loss:  0.08429430634379387
Test accuracy: 0.9841
```

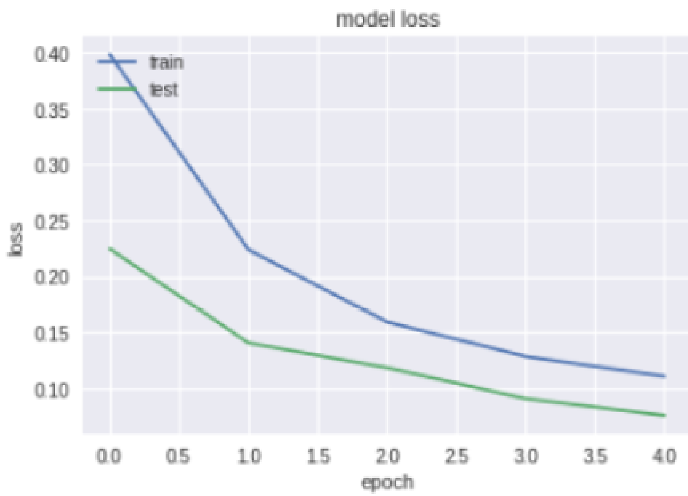
F. <u>Using different weight initializers (He)</u>

Xavier is by default weight initializer in Keras. So, Custom CNN contained Xavier initial weights.

Following result was obtained for He initializer.

1. He_normal:

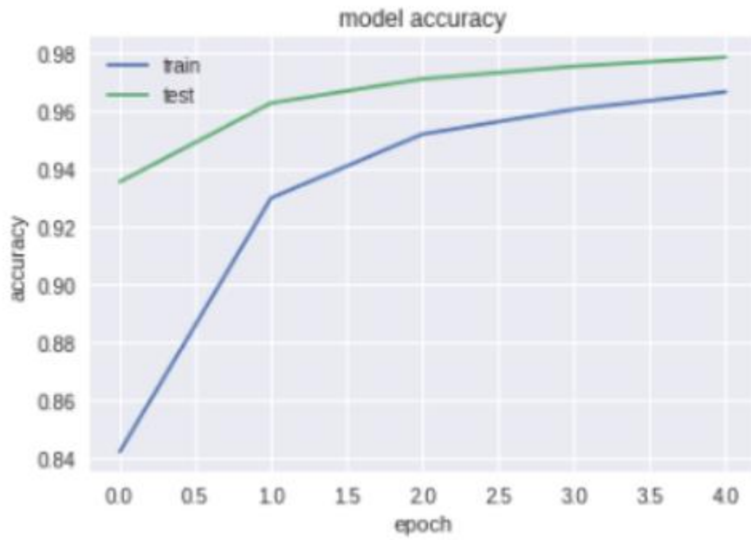


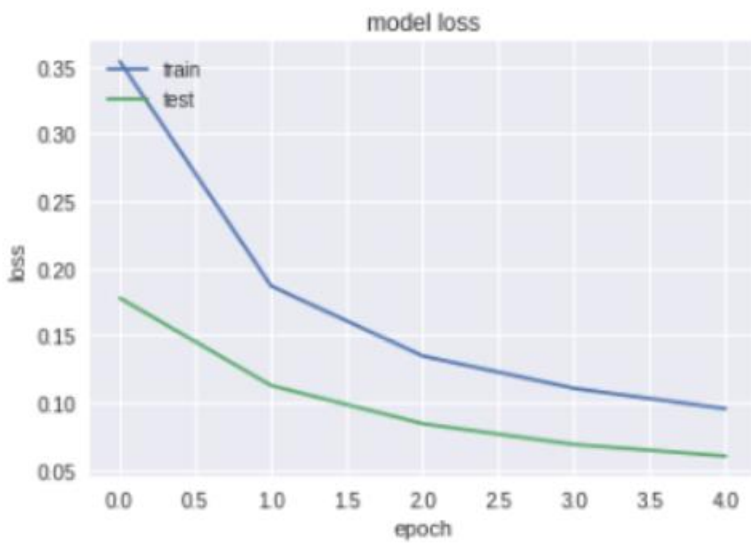



```
[54] # load the weights that yielded the best validation accuracy
     model.load_weights('model.weights.best.hdf5')

     # evaluate and print test accuracy
     score = model.evaluate(X_test, new_Y_test, verbose=0)
     print('Test loss: ', score[0])
     print('Test accuracy:', score[1])
```

```
Test loss:  0.07527381648942828
Test accuracy: 0.9724
```

2. He_uniform:






```
[69] # load the weights that yielded the best validation accuracy
     model.load_weights('model.weights.best.hdf5')

     # evaluate and print test accuracy
     score = model.evaluate(X_test, new_Y_test, verbose=0)
     print('Test loss: ', score[0])
     print('Test accuracy:', score[1])

     Test loss:  0.06012772022634744
     Test accuracy: 0.9784
```

References:

https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html

https://stackoverflow.com/questions/42112260/how-do-i-use-the-tensorboard-callback-of-keras

https://keras.io/optimizers/

https://keras.io/initializers/

https://keras.io/callbacks/

https://github.com/CHIKKIZZY/cs512_TA_CodeSamples/blob/master/AS4/train_with_colab.md