CS512 Assignment 4: Report
Arpit Hasmukhbhai Patel
CWID: A20424085
Semester: Fall 2018
Department of Computer Science
Illinois Institute of Technology
November 03,2018

❖ Abstract

This programming assignment deals with implementation of a basic Convolutional Neural Network for classification. For, implementation it needs to use a GPU framework. Specifically, Keras or TensorFlow. After that we will train and evaluate a CNN to classify images of numbers in the MNIST dataset as either even or odd.

❖ Problem Statement

Deliverables 1: Custom CNN

1. In this we need to train CNN using following configuration such as: use MNIST train set: 55,000 samples, add 2 Convolutional and Pooling layers in which Layer 1should be 32 filters of kernel size 5x5 and Layer 2 should be 64 filters of kernel size 5x5, Pooling in both layers should down sample by a factor of 2, Dropout rate of 40%, use gradient descent optimization and a learning rate of 0.001, Train for 5 epochs
2. In this we need to report the training loss and accuracy for each step/iteration and plot the curve as our model trains. Also, we need to report the loss and accuracy value of the final training step
3. In this we need to report the loss, accuracy, precision and recall for MNIST test set

Deliverables 2: Parameter Tuning

Evaluate different variations of the basic network as described in the question and measure performance

Deliverable 3: Application

In this we need write a program using our pretrained custom CNN which should do the following:

1. Accept as input an image of a handwritten digit
2. Using OpenCV do some basic image pre-processing to prepare the image for our CNN
3. Use our CNN to classify the binary image
4. Our program should continuously request the path to an image file, process the image, and output to the console that class (even/odd) of the image
5. Program should terminate when 'q' or ESC key is entered

❖ Proposed Solutions

- ❖ You have learned how to build a CNN using either Keras or TensorFlow. I will now train and evaluate a CNN to classify images of numbers in the MNIST dataset as either even or odd.
- ❖ Note, images in the MNIST dataset are labeled by the corresponding integer. For example, a handwritten image of the digit '7' is labeled as the integer 7. Hence, I have to map the integer labeling (10 classes) of the images to a binary labeling (even/odd).
- ❖ Make sure that your program can save and load weights so that training can proceed with previous results. You will be required to log and plot some metrics as a function of training and evaluation iterations.

❖ Implementation

- ❖ We are using keras with backend as a tensorflow. First we load the data and change the label of data because we want to find odd and even number. If it is even then we classify it as 0 and if it is odd then we classify as 1. Then we only want to use the 55000 samples from mnist dataset for training so we take that with x_train = x_train[:55000] code. Then we reshape it to 28 by 28 pixels.
- ❖ Now, we create the first convolution neural network with 32 filters and kernel size of 5x5 with Conv2D function in keras. Now, we do pooling layer with downsample factor of 2.
- ❖ Now, we create the second convolution neural network with 64 filters and kernel size of 5x5. In both convolution layer we used activation function 'relu'. And we add the same pooling layer from above.
- ❖ So, after that we take dropout rate of 40% with model.add(Dropout(0.4)). Create a flatten layer and add sigmoid activation function.
- ❖ Learning rate is a rate at which model learns. If we take learning rate so high we can not get good model and if we take it too low it takes more time to learn so we want balance of both of those. Learning rate 0.001 is what I implemented in program as asked. For that we used gradient decent optimization.
- ❖ To count loss in the training we used crossentropy loss function. And we taken 5 epochs as asked in the assignments.
- ❖ We train the model with model.fit() function. After that I printed the plot how accuracy is increasing with training and loss is decreasing.
- ❖ After training we save the model with model.save() function of Keras. So this is for Deliverables 1
- ❖ In Deliverables 2 we make some changes like changing learning rate or number of layers and so on and check what it affects on training accuracy and loss and precision etc.
- ❖ In Deliverables 3 we load that saved model with load_model() and give an imput of an image. We do some changes in the image. We resize the image to 28x28 pixels because we trained the model in that pixel. We convert it to grayscale and do some blurring with GaussianBlur() function.

❖ Results

Deliverables– 1

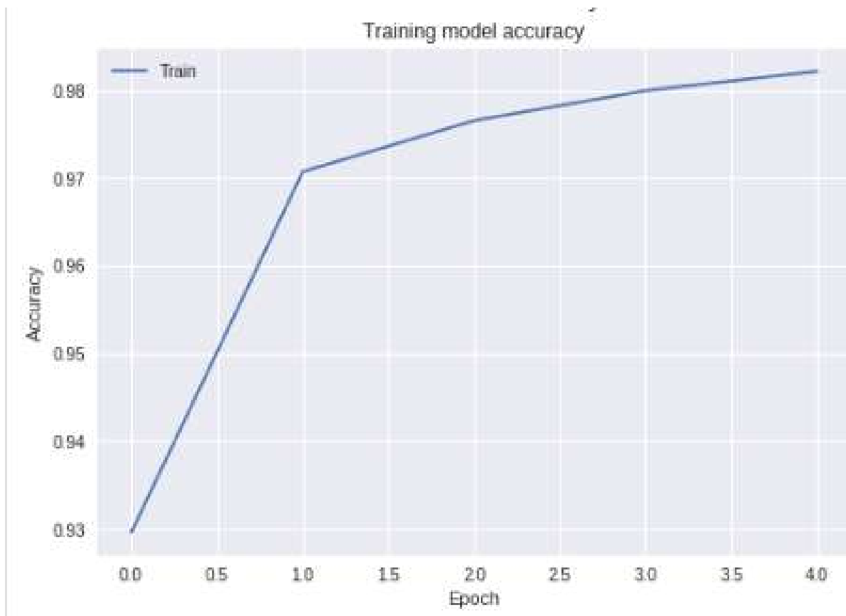
1. Trained CNN using the configuration mentioned in the question

```
#Trainig for 5 epochs
epochs = 5

# building a linear stack of layers with the sequential model
model = Sequential()
model.add(Conv2D(32, kernel_size=(5,5),input_shape=(28, 28, 1), activation = 'relu')) #Layer 1 consist of 32 filters of kernal size 5 X 5
model.add(MaxPooling2D(pool_size=(2, 2))) #pooling and downsample by a factor of 2
model.add(Conv2D(64, kernel_size=(5,5), activation = 'relu')) #Layer 2 consist of 64 filters of kernal size 5 X 5
model.add(MaxPooling2D(pool_size=(2, 2))) #pooling and downsample by a factor of 2
model.add(Dropout(0.4)) #droupout rate of 40%

# Flatten Layer
model.add(Flatten())

model.add(Dense(1))

#using sigmoid activation
model.add(Activation("sigmoid"))

#using gradient descent optimizer and a learning rate of 0.001
sgd = optimizers.SGD(lr=0.001)

# compile the sequential model
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy', precision, recall])

# training the model and saving metrics in history
history = model.fit(x_train, y_train, epochs= epochs , validation_data=(x_test, y_test))
```

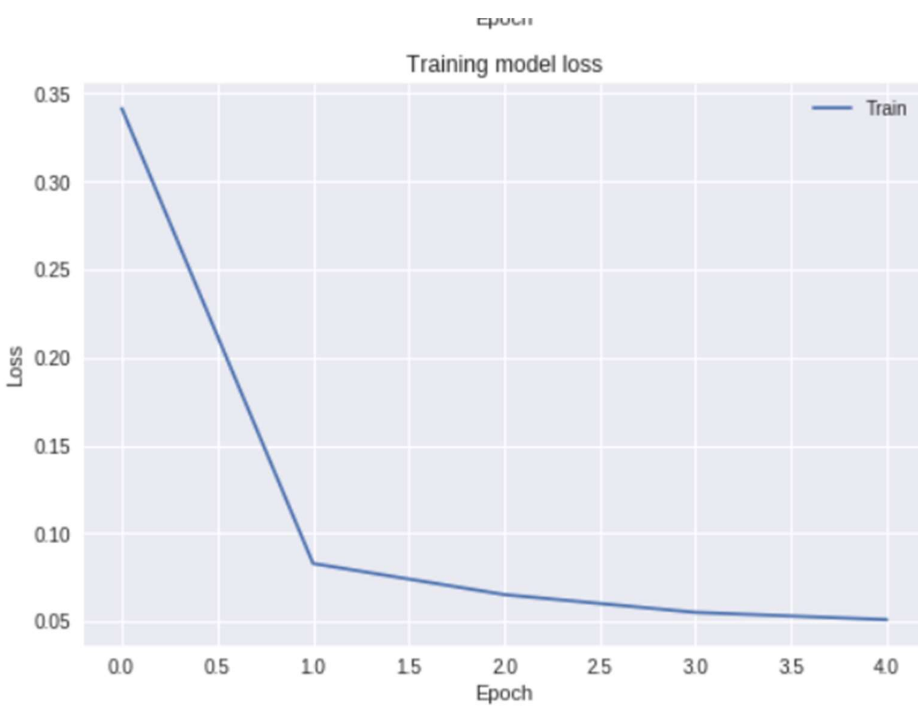
2.

a. training loss and accuracy for each step/iteration and plot

```
Using TensorFlow backend.
Train on 55000 samples, validate on 10000 samples
Epoch 1/5
55000/55000 [==============================] - 21s 375us/step - loss: 0.3415 - acc: 0.9320
Epoch 2/5
55000/55000 [==============================] - 19s 350us/step - loss: 0.0828 - acc: 0.9703
Epoch 3/5
55000/55000 [==============================] - 19s 353us/step - loss: 0.0652 - acc: 0.9768
Epoch 4/5
55000/55000 [==============================] - 19s 351us/step - loss: 0.0550 - acc: 0.9808
Epoch 5/5
55000/55000 [==============================] - 19s 353us/step - loss: 0.0509 - acc: 0.9819
```



Plot of training model accuracy



Plot of training model loss

b. Loss and accuracy value of the final training step

```
55000/55000 [==============================] - 8s 150us/step


 Loss and accuracy of final training step
Train loss: 0.026033725902878425
Train accuracy: 0.9912
```

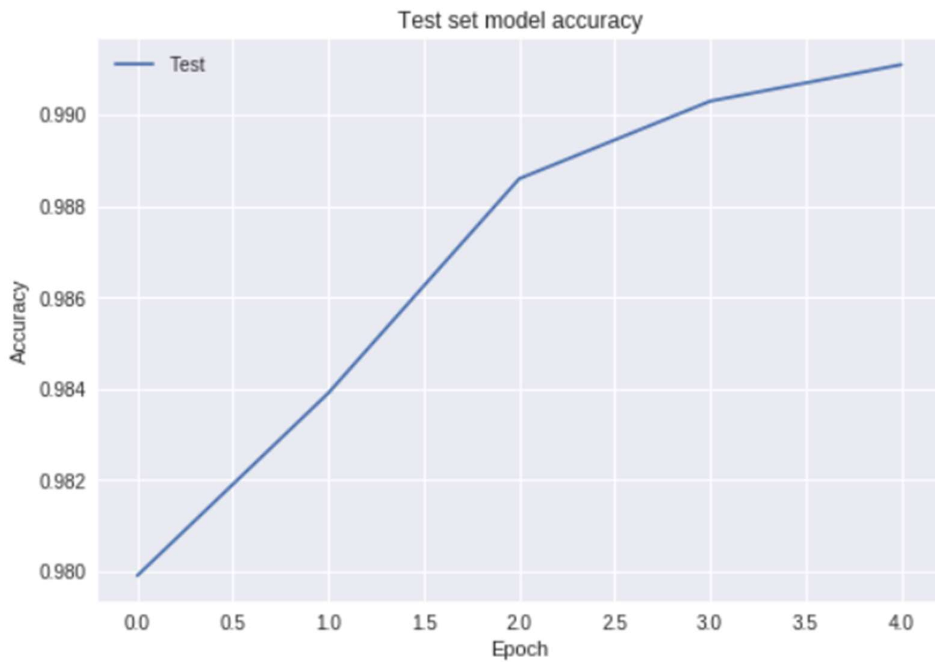
3.

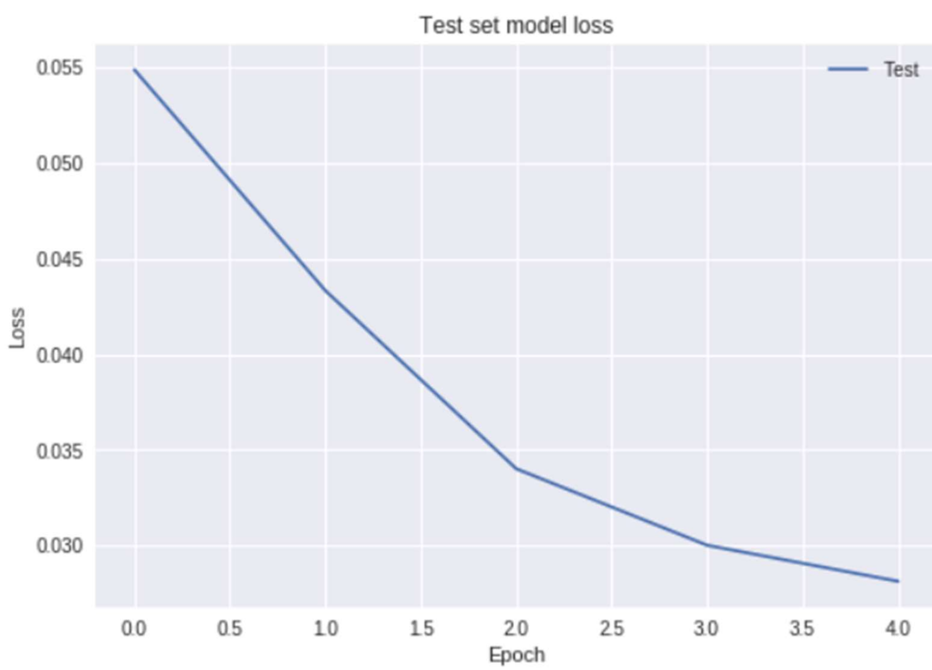
a. Report the loss, accuracy, precision and recall for MNIST test set

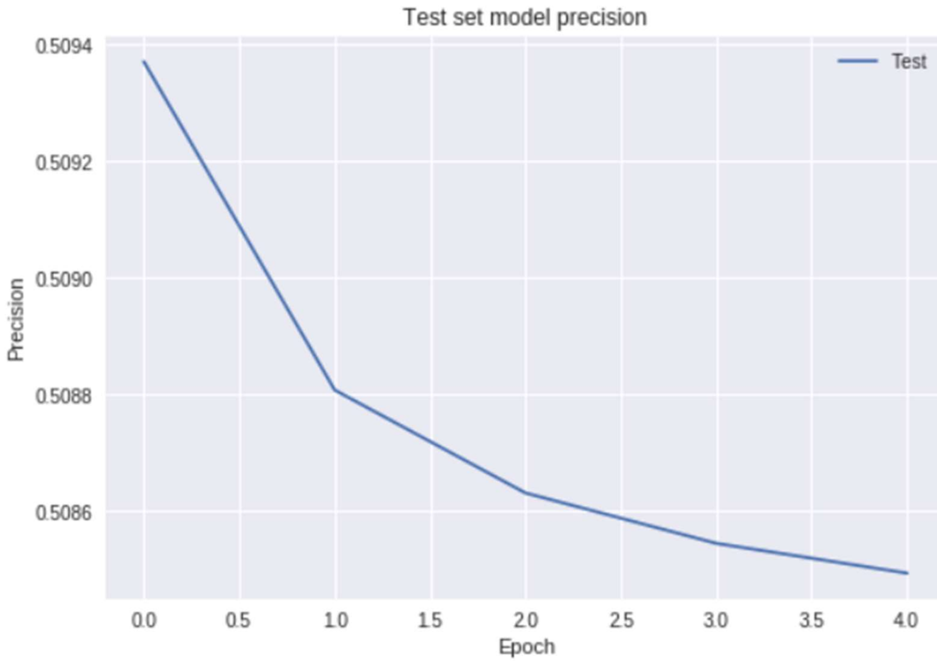
```
Evaluation of Testing set:

10000/10000 [==============================] - 2s 157us/step
Test Loss:0.028120689794793725  Test Accuracy:0.9911    Test Precision:0.5084591895103454       Test Recall:0.9999080164909363
```

b. For each epoch trained compute above mentioned metrics

```
Train on 55000 samples, validate on 10000 samples
Epoch 1/5
55000/55000 [==============================] - 21s 375us/step - loss: 0.3415 - acc: 0.9320 - precision: 0.5123 - recall: 0.9965 - val_loss: 0.0549 - val_acc: 0.9799 - val_precision: 0.5094 - val_recall: 0.9994
Epoch 2/5
55000/55000 [==============================] - 19s 350us/step - loss: 0.0828 - acc: 0.9703 - precision: 0.5091 - recall: 0.9996 - val_loss: 0.0433 - val_acc: 0.9839 - val_precision: 0.5088 - val_recall: 0.9997
Epoch 3/5
55000/55000 [==============================] - 19s 353us/step - loss: 0.0652 - acc: 0.9768 - precision: 0.5084 - recall: 0.9998 - val_loss: 0.0340 - val_acc: 0.9886 - val_precision: 0.5086 - val_recall: 0.9998
Epoch 4/5
55000/55000 [==============================] - 19s 351us/step - loss: 0.0550 - acc: 0.9808 - precision: 0.5086 - recall: 0.9998 - val_loss: 0.0300 - val_acc: 0.9903 - val_precision: 0.5085 - val_recall: 0.9999
Epoch 5/5
55000/55000 [==============================] - 19s 353us/step - loss: 0.0509 - acc: 0.9819 - precision: 0.5084 - recall: 0.9999 - val_loss: 0.0281 - val_acc: 0.9911 - val_precision: 0.5085 - val_recall: 0.9999
55000/55000 [==============================] - 8s 150us/step
```

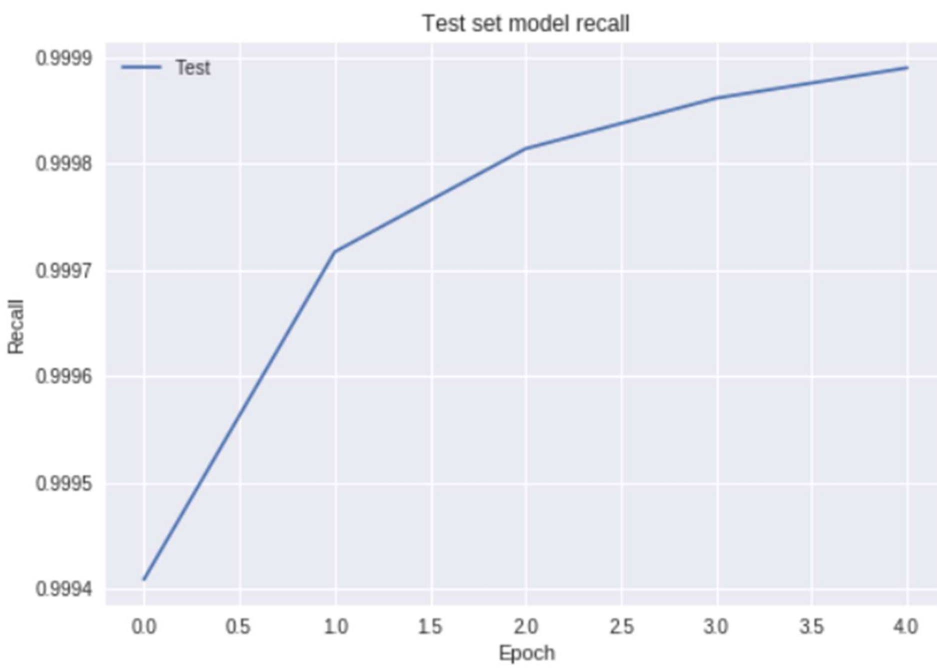

Test set model accuracy
Test
0.990
0.988
0.986
0.984
0.982
0.980
Accuracy
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch


Test set model loss
Test
0.055
0.050
0.045
0.040
0.035
0.030
Loss
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch



Test set model precision
Test
Precision
0.5094
0.5092
0.5090
0.5088
0.5086
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch

Epoch



Test set model recall
Test
Recall
0.9999
0.9998
0.9997
0.9996
0.9995
0.9994
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch

Deliverables 2:

- Learning Rate: 0.01

```
Train on 55000 samples, validate on 10000 samples
Epoch 1/5
55000/55000 [==============================] - 15s 268us/step - loss: 7.8351 - acc: 0.5085 - precision: 0.5082 - recall: 0.9994 -
Epoch 2/5
55000/55000 [==============================] - 14s 257us/step - loss: 7.8364 - acc: 0.5085 - precision: 0.5079 - recall: 1.0000 -
Epoch 3/5
55000/55000 [==============================] - 14s 258us/step - loss: 7.8364 - acc: 0.5085 - precision: 0.5080 - recall: 1.0000 -
Epoch 4/5
55000/55000 [==============================] - 14s 258us/step - loss: 7.8364 - acc: 0.5085 - precision: 0.5084 - recall: 1.0000 -
Epoch 5/5
55000/55000 [==============================] - 14s 258us/step - loss: 7.8364 - acc: 0.5085 - precision: 0.5085 - recall: 1.0000 -
55000/55000 [==============================] - 6s 116us/step


 Loss and accuracy of final training step
Train loss: 7.83640719687722
Train accuracy: 0.5084545454502106



Evaluation of Testing set:


10000/10000 [==============================] - 1s 116us/step
Test Loss:7.85321912612915      Test Accuracy:0.5074    Test Precision:0.5083225183486938       Test Recall:1.0
```

When we changed the learning rate of the model from 0.001 to 0.01 then the loss is increased and accuracy is decreased from 0.98 to 0.50. So learning rate 0.001 is better then 0.01.

- Dropout – 30%

```
Train on 55000 samples, validate on 10000 samples
Epoch 1/5
55000/55000 [==============================] - 15s 267us/step - loss: 6.1876 - acc: 0.5991 - precision: 0.7635 - recall: 0.2862 -
Epoch 2/5
55000/55000 [==============================] - 14s 257us/step - loss: 0.1107 - acc: 0.9604 - precision: 0.5450 - recall: 0.6690 -
Epoch 3/5
55000/55000 [==============================] - 14s 257us/step - loss: 0.0730 - acc: 0.9745 - precision: 0.5248 - recall: 0.8096 -
Epoch 4/5
55000/55000 [==============================] - 14s 256us/step - loss: 0.0571 - acc: 0.9804 - precision: 0.5191 - recall: 0.8659 -
Epoch 5/5
55000/55000 [==============================] - 14s 256us/step - loss: 0.0489 - acc: 0.9836 - precision: 0.5163 - recall: 0.8965 -
55000/55000 [==============================] - 6s 116us/step


 Loss and accuracy of final training step
Train loss: 0.027314712133309382
Train accuracy: 0.990909090909091



Evaluation of Testing set:


10000/10000 [==============================] - 1s 118us/step
Test Loss:0.0282995004942175    Test Accuracy:0.9894    Test Precision:0.5139981072425842       Test Recall:0.9228913857460022
```

When we changed the drop out from from 40% to 30% then the precision is decreased from around 0.87 to 0.51.

- Epochs: 10

```
Epoch 4/10
55000/55000 [==============================] - 14s 257us/step - loss: 0.0651 - acc: 0.9775 - precision: 0.5083 - recall: 1.0000
Epoch 5/10
55000/55000 [==============================] - 14s 255us/step - loss: 0.0568 - acc: 0.9804 - precision: 0.5085 - recall: 1.0000
Epoch 6/10
55000/55000 [==============================] - 14s 255us/step - loss: 0.0511 - acc: 0.9822 - precision: 0.5085 - recall: 1.0000
Epoch 7/10
55000/55000 [==============================] - 14s 256us/step - loss: 0.0471 - acc: 0.9831 - precision: 0.5085 - recall: 1.0000
Epoch 8/10
55000/55000 [==============================] - 14s 255us/step - loss: 0.0428 - acc: 0.9856 - precision: 0.5084 - recall: 1.0000
Epoch 9/10
55000/55000 [==============================] - 14s 256us/step - loss: 0.0418 - acc: 0.9855 - precision: 0.5084 - recall: 1.0000
Epoch 10/10
55000/55000 [==============================] - 14s 256us/step - loss: 0.0386 - acc: 0.9864 - precision: 0.5084 - recall: 1.0000
55000/55000 [==============================] - 6s 115us/step


 Loss and accuracy of final training step
Train loss: 0.020452919308942826
Train accuracy: 0.9938727272727272



Evaluation of Testing set:


10000/10000 [==============================] - 1s 115us/step
Test Loss:0.025309678837982994   Test Accuracy:0.9919    Test Precision:0.5083759575843811      Test Recall:0.9999916866302491
```

- Epochs: 3

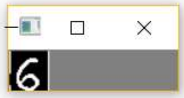
```
Train on 55000 samples, validate on 10000 samples
Epoch 1/3
55000/55000 [==============================] - 15s 268us/step - loss: 0.5155 - acc: 0.9223 - precision: 0.5277 - recall: 0.9905
Epoch 2/3
55000/55000 [==============================] - 14s 257us/step - loss: 0.0906 - acc: 0.9681 - precision: 0.5113 - recall: 0.9984
Epoch 3/3
55000/55000 [==============================] - 14s 256us/step - loss: 0.0706 - acc: 0.9751 - precision: 0.5100 - recall: 0.9991
55000/55000 [==============================] - 6s 116us/step


 Loss and accuracy of final training step
Train loss: 0.036894038953395054
Train accuracy: 0.9875636363636363



Evaluation of Testing set:


10000/10000 [==============================] - 1s 117us/step
Test Loss:0.0382737568805227    Test Accuracy:0.986    Test Precision:0.509482096195221       Test Recall:0.9994521673202514
```

- Optimizer = Adam

```
Train on 55000 samples, validate on 10000 samples
Epoch 1/5
55000/55000 [==============================] - 17s 304us/step - loss: 0.3618 - acc: 0.9476 - precision: 0.5319 - recall: 0.9932 -
Epoch 2/5
55000/55000 [==============================] - 16s 288us/step - loss: 0.0647 - acc: 0.9789 - precision: 0.5119 - recall: 0.9989 -
Epoch 3/5
55000/55000 [==============================] - 16s 284us/step - loss: 0.0599 - acc: 0.9803 - precision: 0.5106 - recall: 0.9994 -
Epoch 4/5
55000/55000 [==============================] - 16s 284us/step - loss: 0.0557 - acc: 0.9823 - precision: 0.5101 - recall: 0.9996 -
Epoch 5/5
55000/55000 [==============================] - 16s 284us/step - loss: 0.0541 - acc: 0.9830 - precision: 0.5096 - recall: 0.9997 -
55000/55000 [==============================] - 6s 115us/step


 Loss and accuracy of final training step
Train loss: 0.028247765320996668
Train accuracy: 0.9906909090909091



Evaluation of Testing set:


10000/10000 [==============================] - 1s 116us/step
Test Loss:0.027219966612639836  Test Accuracy:0.9905    Test Precision:0.509289197921753       Test Recall:0.9997444877624512
```

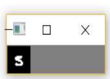
Deliverable 3:

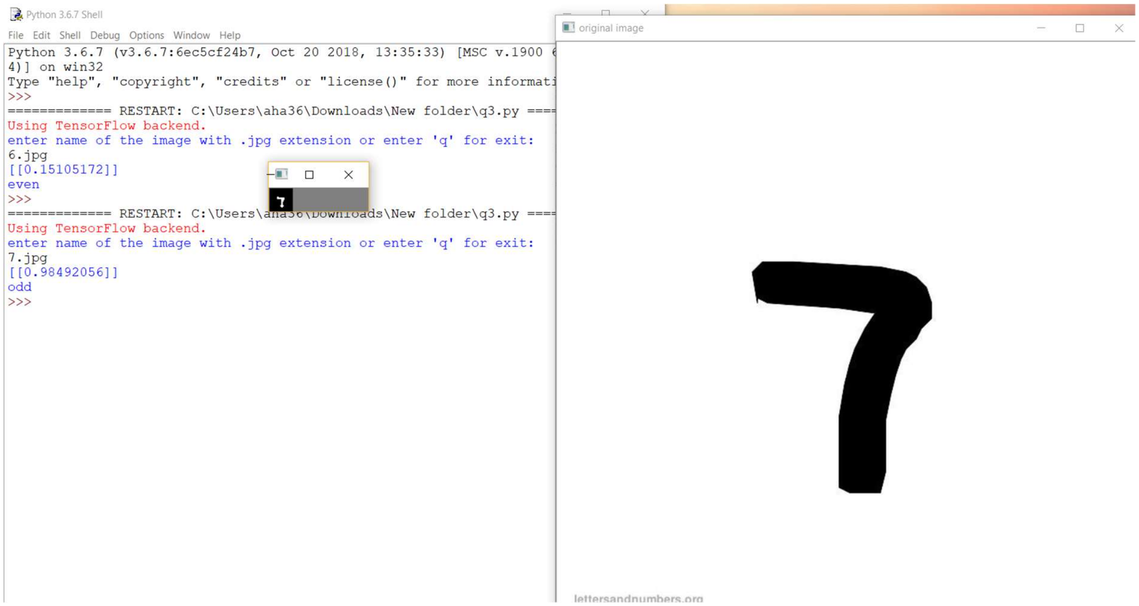
```
Using TensorFlow backend.
enter name of the image with .jpg extension or enter 'q' for exit:
6.jpg
[[0.15105172]]
even
>>>
```

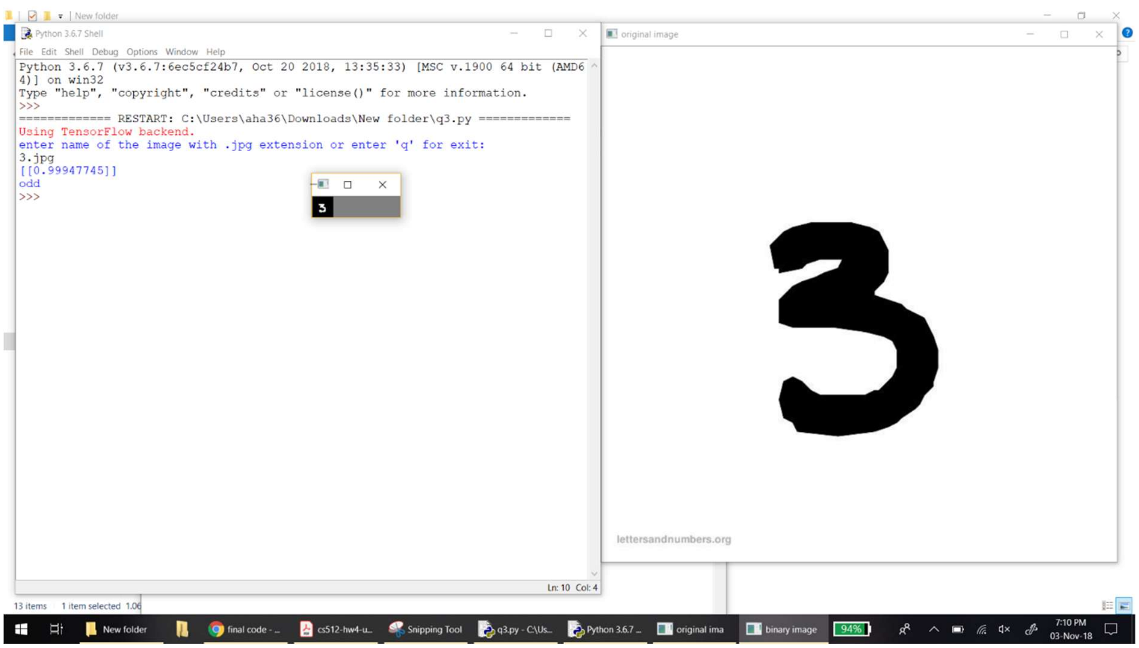


```
Using TensorFlow backend.
enter name of the image with .jpg extension or enter 'q' for exit:
3.jpg
[[0.99947745]]
odd
```

```
Using TensorFlow backend.
enter name of the image with .jpg extension or enter 'q' for exit:
5.jpg
[[0.9954639]]
odd
>>>
```




Python 3.6.7 Shell
File Edit Shell Debug Options Window Help
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 6
4)] on win32
Type "help", "copyright", "credits" or "license()" for more informati
>>>
============= RESTART: C:\Users\aha36\Downloads\New folder\q3.py ====
Using TensorFlow backend.
enter name of the image with .jpg extension or enter 'q' for exit:
6.jpg
[[0.15105172]]
even
>>>
============= RESTART: C:\Users\aha36\Downloads\New folder\q3.py ====
Using TensorFlow backend.
enter name of the image with .jpg extension or enter 'q' for exit:
7.jpg
[[0.98492056]]
odd
>>>
original image
7
lettersandnumbers.org


New folder
Python 3.6.7 Shell
File Edit Shell Debug Options Window Help
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64 bit (AMD6
4)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:\Users\aha36\Downloads\New folder\q3.py =============
Using TensorFlow backend.
enter name of the image with .jpg extension or enter 'q' for exit:
3.jpg
[[0.99947745]]
odd
>>>
original image
3
lettersandnumbers.org
Ln: 10 Col: 4
13 items 1 item selected
New folder
final code -
cs512-hw4-u
Snipping Tool
q3.py - C:\Us
Python 3.6.7
original ima
binary image
94%
7:10 PM
03-Nov-18