CS512 Assignment 4: Report

Parthkumar Patel
CWID: A20416508
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

First, we will train and evaluate a CNN to classify images of numbers in the MNIST dataset as either even or odd. Images in the MNIST dataset are labelled by the corresponding integer. For example, a handwritten image of the digit '7' is labelled as the integer 7. Hence, you will have to map the integer labelling (10 classes) of the images to a binary labelling (even/odd).

After, that we will build the linear stack of layers using sequential model where we will add 2 layers where first one is 32 filter of kernel size 5*5 and second one 64 filters of kernel size 5*5. Now, we will down sample it by factor of 2 and set the other parameters and will use gradient decent optimization and a learning rate of 0.001 and train it for 5 epochs.

After training it for 5 epochs, now we will train it for final training step and calculate the loss, accuracy, precision and recall. Now, we will test it using 10000 images and we will get the result of the above listed parameter for MNSIT test images. Now , we will save that created model using keras's save.model function and load it into another file to test our handwritten images to check whether it will classify it into odd or even image.

# ❖ Results

**Deliverables– 1**

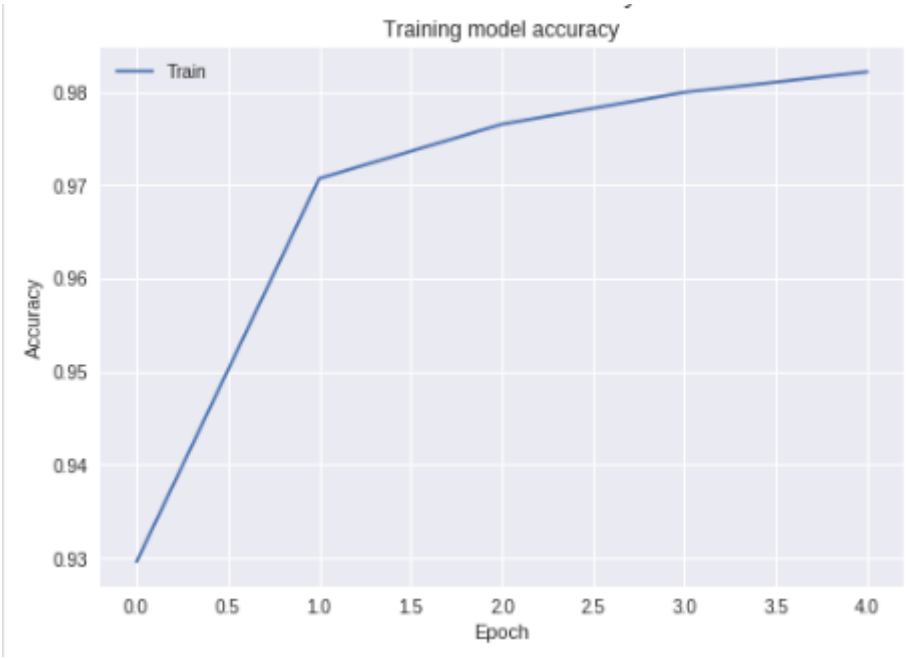
1. Trained CNN using the configuration mentioned in the question

```
#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 s
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())

# Use final dense layer to get two outputs
model.add(Dense(1))

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

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

# compiling 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))
```

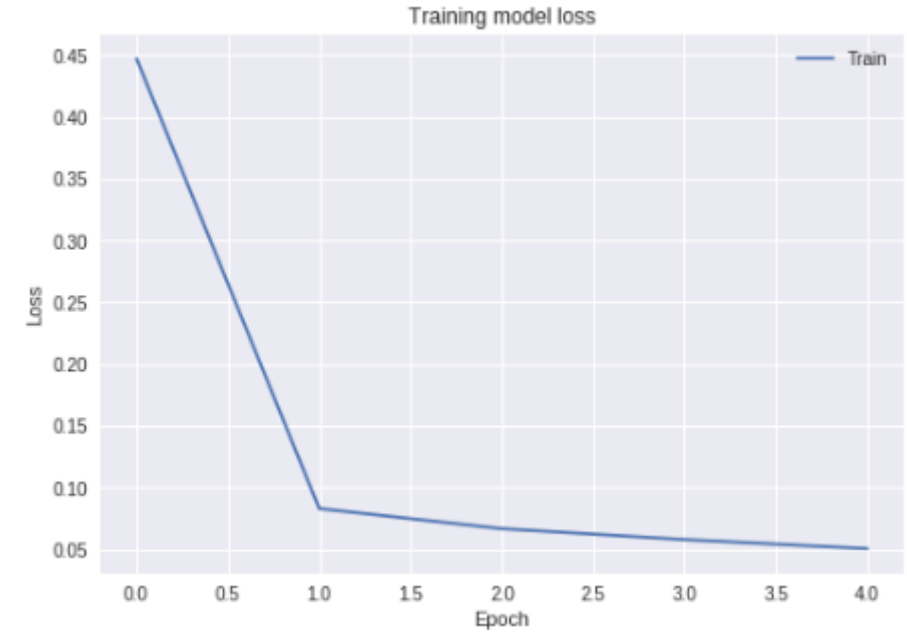
2.

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

```
Train on 55000 samples, validate on 10000 samples
Epoch 1/5
55000/55000 [==============================] - 15s 264us/step - loss: 0.3316 - acc: 0.9301
Epoch 2/5
55000/55000 [==============================] - 14s 257us/step - loss: 0.0879 - acc: 0.9679
Epoch 3/5
55000/55000 [==============================] - 14s 258us/step - loss: 0.0657 - acc: 0.9767
Epoch 4/5
55000/55000 [==============================] - 14s 258us/step - loss: 0.0581 - acc: 0.9799
Epoch 5/5
55000/55000 [==============================] - 14s 259us/step - loss: 0.0503 - acc: 0.9827
```




Plot of training model accuracy




Plot of training model loss

b. Loss and accuracy value of the final training step

```
55000/55000 [==============================] - 6s 118us/step


 Loss and accuracy of final training step
Train loss: 0.028436766319383276
Train accuracy: 0.9904
```

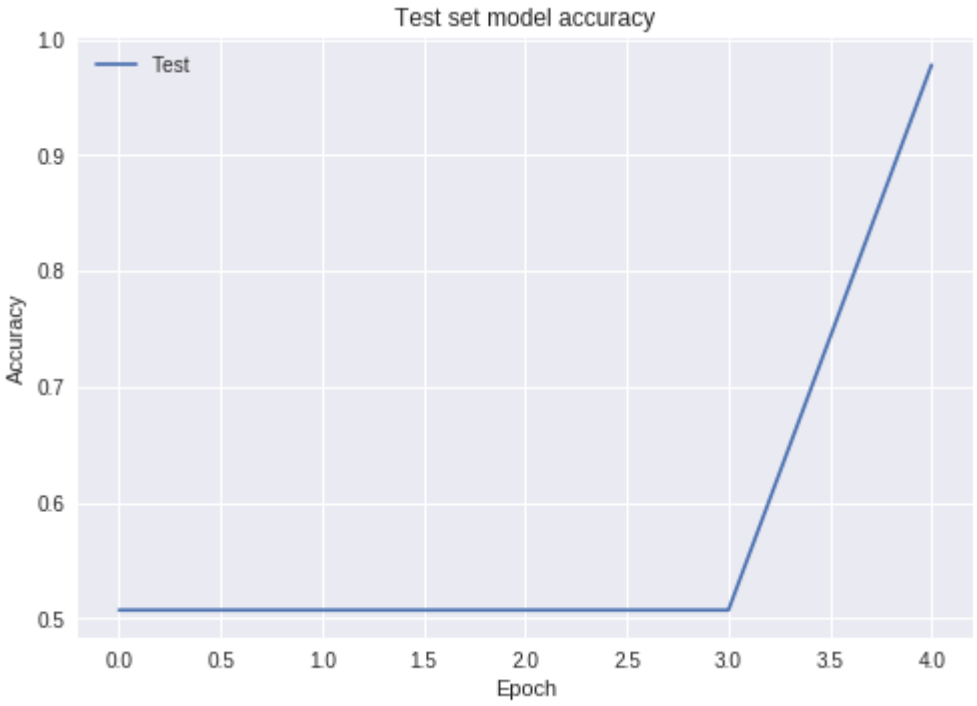
3.

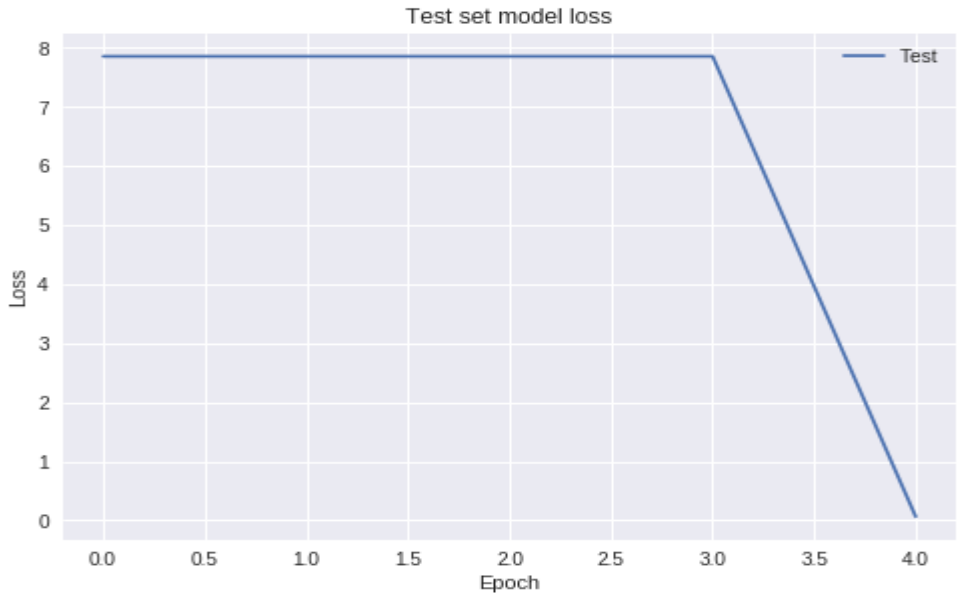
a. Report the loss, accuracy, precision and recall for MNIST test set

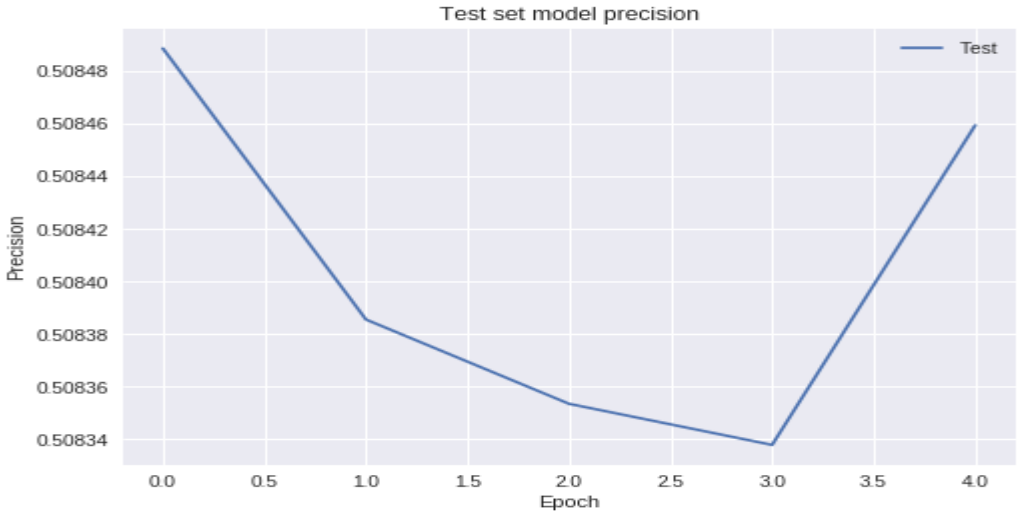
```
Evaluation of Testing set:


10000/10000 [==============================] - 1s 120us/step
Test Loss:8.178321719360351     Test Accuracy:0.4926    Test Precision:0.870097441005706       Test Recall:0.025089589014649392
```

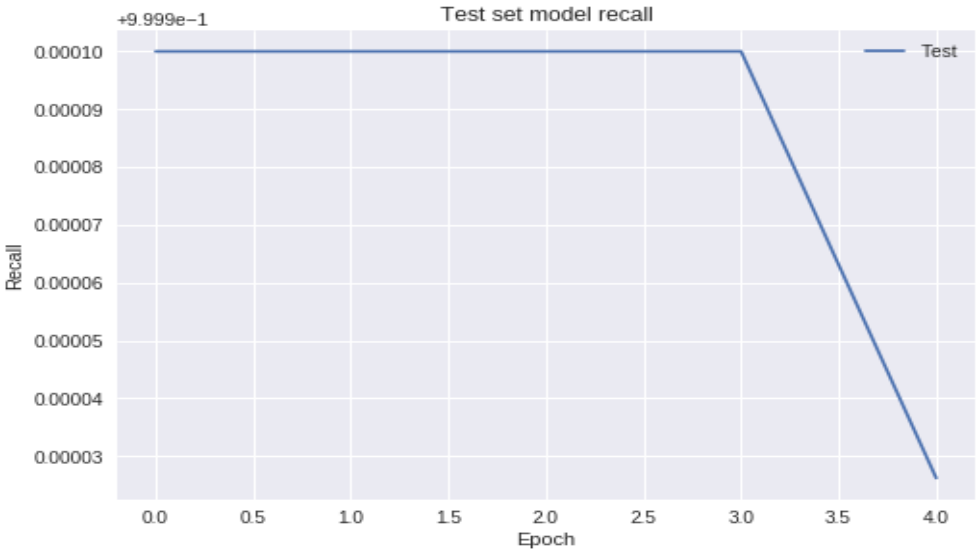
b. For each epoch trained compute above mentioned metrics

```
Train on 55000 samples, validate on 10000 samples
Epoch 1/5
55000/55000 [==============================] - 15s 271us/step - loss: 8.1936 - acc: 0.4915 - precision: 0.7850 - recall: 0.0453 - val_loss: 8.1783 - val_acc: 0.4926 - val_precision: 0.8383 - val_recall: 0.0330
Epoch 2/5
55000/55000 [==============================] - 14s 263us/step - loss: 8.1953 - acc: 0.4915 - precision: 0.8454 - recall: 0.0308 - val_loss: 8.1783 - val_acc: 0.4926 - val_precision: 0.8531 - val_recall: 0.0308
Epoch 3/5
55000/55000 [==============================] - 15s 264us/step - loss: 8.1953 - acc: 0.4915 - precision: 0.8567 - recall: 0.0299 - val_loss: 8.1783 - val_acc: 0.4926 - val_precision: 0.8628 - val_recall: 0.0300
Epoch 4/5
55000/55000 [==============================] - 15s 266us/step - loss: 8.1953 - acc: 0.4915 - precision: 0.8644 - recall: 0.0296 - val_loss: 8.1783 - val_acc: 0.4926 - val_precision: 0.8663 - val_recall: 0.0294
Epoch 5/5
55000/55000 [==============================] - 15s 268us/step - loss: 8.1953 - acc: 0.4915 - precision: 0.8663 - recall: 0.0294 - val_loss: 8.1783 - val_acc: 0.4926 - val_precision: 0.8676 - val_recall: 0.0294
```




Test set model loss
Test
Loss
Epoch
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Test set model precision
Test
Precision
Epoch
0.50834
0.50836
0.50838
0.50840
0.50842
0.50844
0.50846
0.50848
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Test set model recall
+9.999e−1
Test
Recall
Epoch
0.00003
0.00004
0.00005
0.00006
0.00007
0.00008
0.00009
0.00010
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

**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 of the model is decreased from 0.98 to 0.50.

- 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:0.02530967883798294   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
```

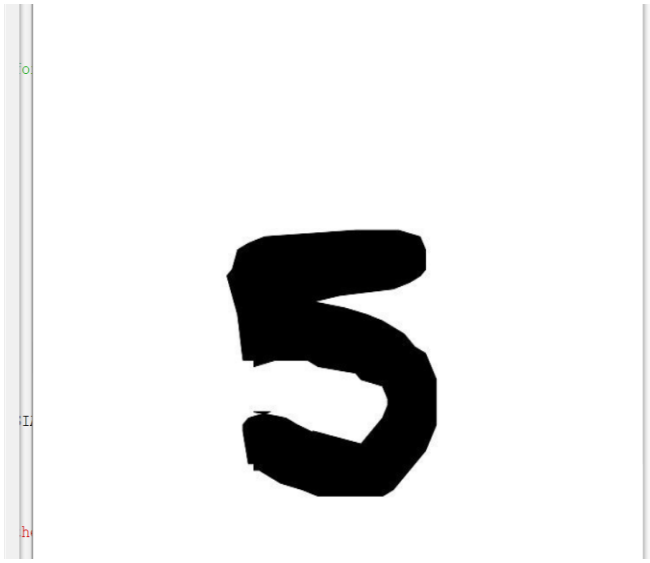
- Optimizer = Adam

```
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:
5.jpg
[[0.9954639]]
odd
>>>
```



```
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:
7.jpg
[[0.98492056]]
odd
>>>
```