

ASSIGNMENT 4 REPORT

VISMAYA VEERAMANJU KALYAN

CS512 FALL18

A20423189

Using Tensorflow Estimators

DELIVERABLES 1: Custom CNN

1.

To execute and run the model: Python cnn.py

Trained the 55,000 samples with 2 convolutional and pooling layers

Layer 1: 32 filters of size 5*5

Layer 2: 64 filter of size 5*5

Pooling by a factor of 2

Dropout rate of 40%

Uses gradient descent optimization and a learning rate of 0.001

Trained for 5 epochs

The model is saved as `cnn_model` in the `models` folder

2.

a. The loss decreases as the model trains.

Loss at 1 epoch = 0.42776573

Loss at 2 epoch = 0.38741004

Loss at 3 epoch = 0.31164566

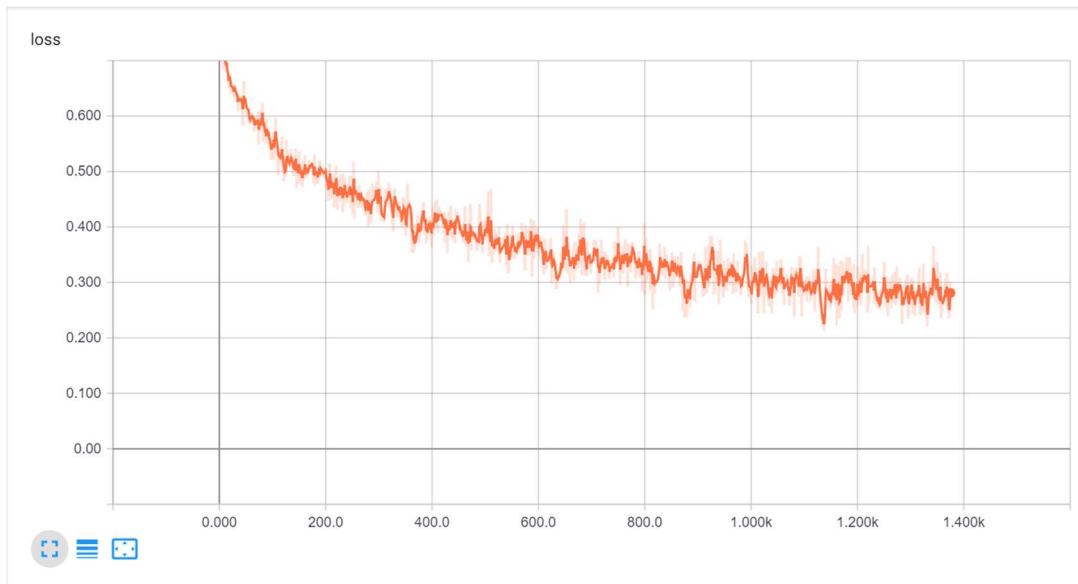
Loss at 4 epoch = 0.36886513

Loss at 5 epoch = 0.3015162

To visualize the training and evaluation results using tensorboard execute:

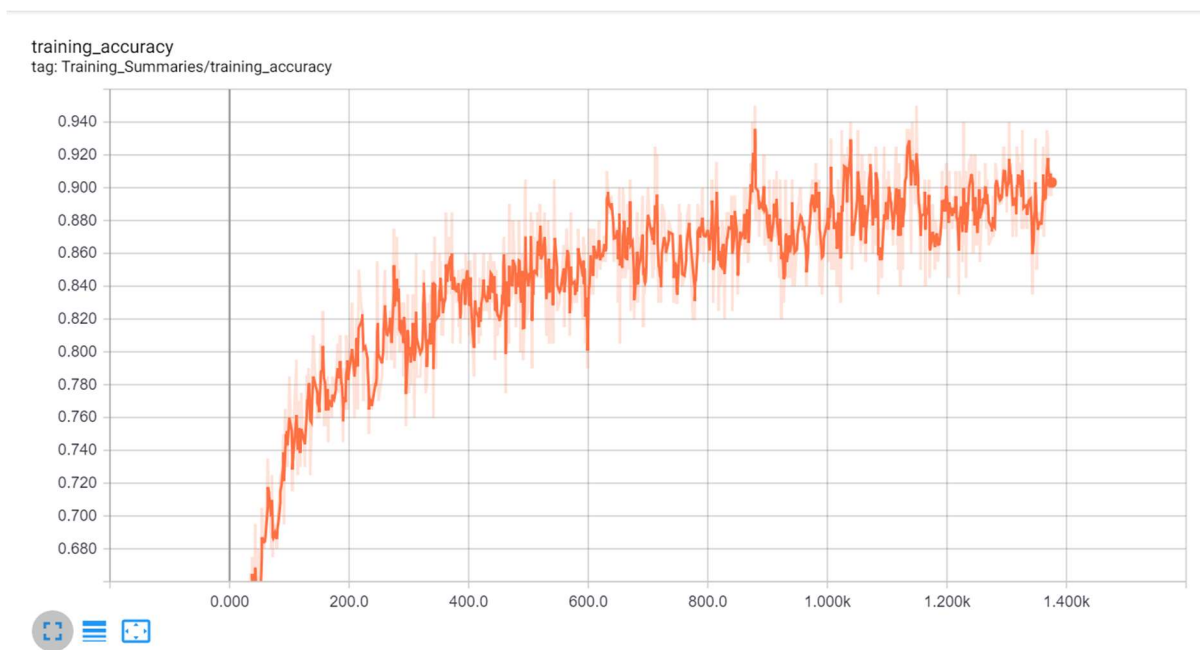
Tensorboard - -logdir=path/to/models/cnn_model

The graph shows the loss at each step/iterations.



The accuracy of the model increases as the model trains

The graph below shows the accuracy at each step.



b.

loss = 0.3015162

accuracy = 0.8950

at the final step.

3.

a.

Epoch: 1

{'accuracy': 0.8369, 'loss': 0.4236007, 'precision': 0.82834256, 'recall': 0.8559322, 'global_step': 275}

Epoch: 2

{'accuracy': 0.8686, 'loss': 0.3459785, 'precision': 0.8616776, 'recall': 0.8827355, 'global_step': 550}

Epoch: 3

{'accuracy': 0.8879, 'loss': 0.3009121, 'precision': 0.8794394, 'recall': 0.902838, 'global_step': 825}

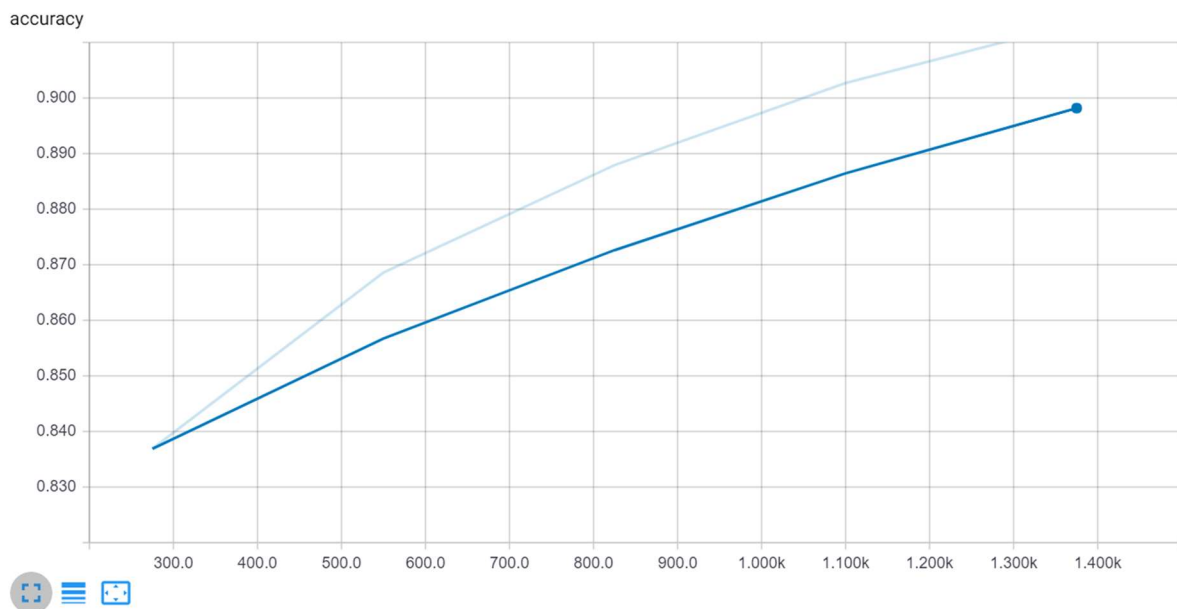
Epoch: 4

{'accuracy': 0.9027, 'loss': 0.2688621, 'precision': 0.90135056, 'recall': 0.907568, 'global_step': 1100}

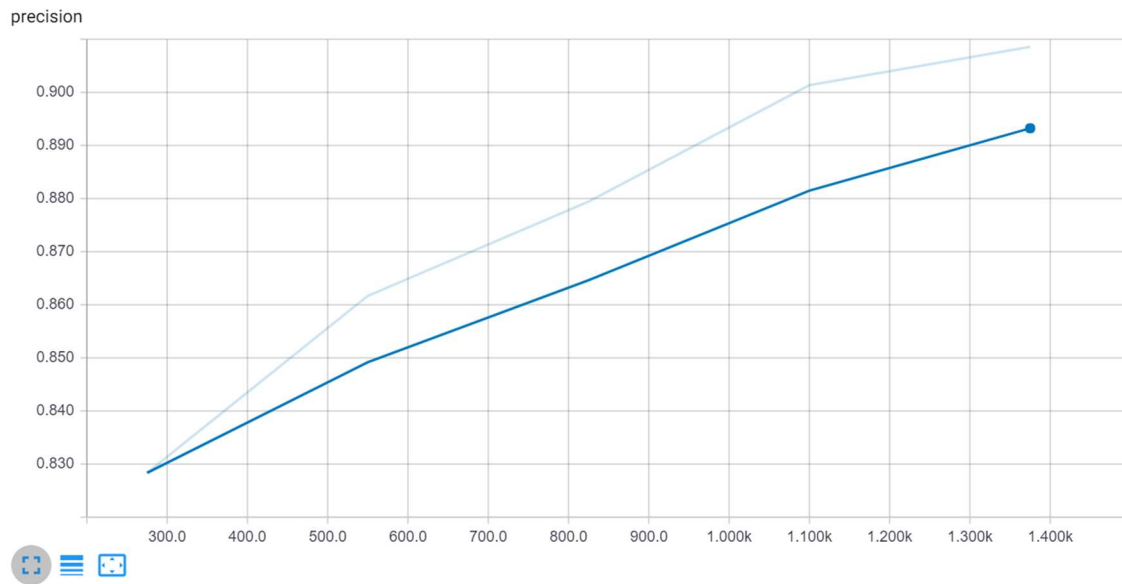
Epoch: 5

{'accuracy': 0.9135, 'loss': 0.24379912, 'precision': 0.90856147, 'recall': 0.9223492, 'global_step': 1375}

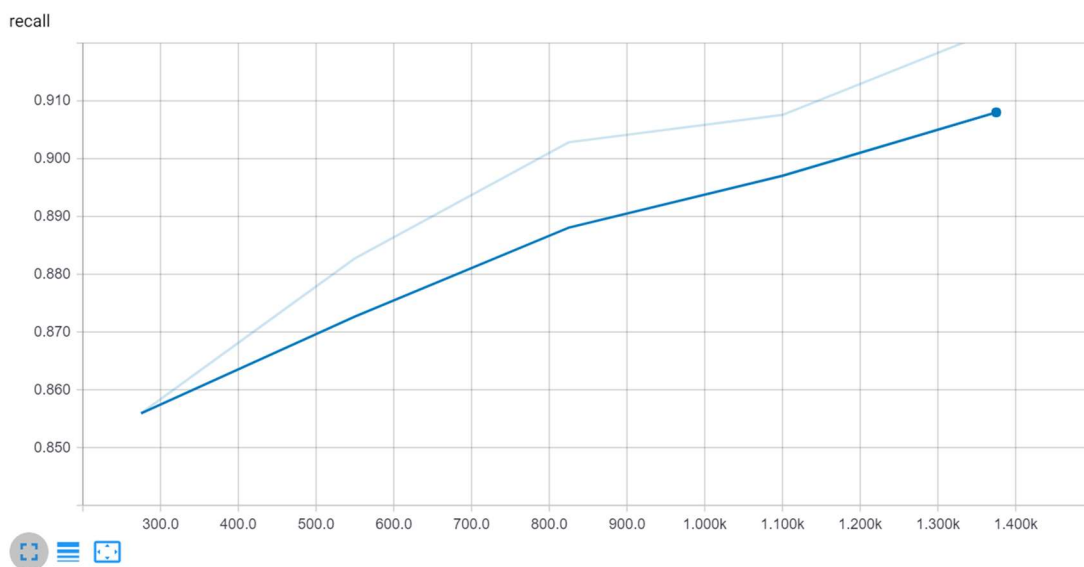
The graph shows the accuracy of the evaluation data of 5 epochs



The graph shows the Precision of the evaluation data of 5 epochs



The graph shows the recall of the evaluation data of 5 epochs



b.

'accuracy': 0.9135,

'loss': 0.24379912,

'precision': 0.90856147,

'recall': 0.9223492

Of the final epoch.

Deliverables 2:

Changing the receptive field

Using bigger kernel size will lead to loss of some details hence reducing the accuracy. Hence a kernel of 5 is a good value for the filter size for the used image size.

Kernel 5	Kernel 10	Kernel 3
Runtime: 41 seconds 'accuracy': 0.9135, 'loss': 0.24379912, 'precision': 0.90856147, 'recall': 0.9223492	Runtime: 42 seconds 'accuracy': 0.9052, 'loss': 0.2485592, 'precision': 0.9122702, 'recall': 0.89968467	Runtime: 34 seconds 'accuracy': 0.9088, 'loss': 0.24770962, 'precision': 0.90708137, 'recall': 0.9138746

Changing the optimizer:

Gradient Decent Optimizer:

Runtime: 46 seconds

'accuracy': 0.9135,

'loss': 0.24379912,

'precision': 0.90856

'recall': 0.9223492

Adam Optimizer:

Runtime: 57 seconds

'accuracy': 0.9926,

'loss': 0.023368156,

'precision': 0.98828125,

'recall': 0.99724084,

Hence using *Adam Optimizer* gives us better result as the accuracy on the evaluation set increased and the loss function decreased. The model also requires little more time compared to Gradient decent optimizer.

Changing various parameters

Dropout :

- Reduces node interaction(co-adaptation)
- reduces overfitting
- increases training speed

'0' dropout implying no dropouts: accuracy is 0.8882 for 5th epoch of evaluation data.

Here the model is overfitted and hence we get a lower accuracy compared to 0.4 dropout where 'accuracy' is 0.9135 for the same example.

0.4 is a good value for dropout as keeping a bigger number would lead to dropping more input nodes.

Learning rate:

The size of the steps taken during the gradient descent is called the learning rate.

high learning rate can cover more ground each step, but risk overshooting the lowest point as the slope of the hill is constantly changing.

very low learning rate can confidently move in the direction of the negative gradient since it is recalculating so frequently. A low learning rate is more precise, but calculating the gradient is time-consuming, so it will take a very long time to get to the bottom.

learn_rate = 1.4	learn_rate = 0.01	learn_rate = 0.001	learn_rate = 0.0001
Runtime: 41 seconds 'accuracy': 0.5074 'loss': 0.6931489, 'precision': 0.5074, 'recall': 1.0,	Runtime: 41 seconds 'accuracy': 0.9774, 'loss': 0.06575417, 'precision': 0.98577154, 'recall': 0.9694521,	Runtime: 46 seconds 'accuracy': 0.9135, 'loss': 0.24379912, 'precision': 0.90856 'recall': 0.9223492	Runtime: 42 seconds 'accuracy': 0.8289, 'loss': 0.47528094, 'precision': 0.8119087, 'recall': 0.86263305,

First, with low learning rates, the loss improves slowly, then training accelerates until the learning rate becomes too large and loss goes up.

The model could be improved by using a learning rate of 0.01.

Number of epochs:

We can make it a better model by increasing the number of epochs. The recommended value is 20,000 steps which is equal to 73 epochs in which case have a very high accuracy and a low loss making our model better compared to the deliverable 1 model.

Epochs 5	Epochs 10	Epochs 20	Epochs 73(20000steps)
----------	-----------	-----------	-----------------------

Runtime: 46 seconds 'accuracy': 0.9135, 'loss': 0.24379912, 'precision': 0.90856 'recall': 0.9223492	Runtime: 79 seconds 'accuracy': 0.9379, 'loss': 0.18103005, 'precision': 0.94273216, 'recall': 0.9343713	Runtime: 155 seconds 'accuracy': 0.9672, 'loss': 0.10557537, 'precision': 0.971957, 'recall': 0.96314543	Runtime: 610 seconds 'accuracy': 0.9845, 'loss': 0.0451717, 'precision': 0.989258, 'recall': 0.9800946
--	--	--	--

Using different weight initializer (fast_conv, Xavier, he_rec):

The accuracy of the model decreases by 10% when using Xavier compared to fast_conv or he_rec.

Using he_rec weight initializer would increase the accuracy and reduce the loss of the model compared to fast_conv used in the model. Hence the model using he_rec would be comparatively better than the deliverable 1 model.

fast_conv	Xavier	he_rec
'accuracy': 0.9135, 'loss': 0.24379912, 'precision': 0.90856147, 'recall': 0.9223492	'accuracy': 0.8115, 'loss': 0.4492977, 'precision': 0.79731494, 'recall': 0.8427276,	'accuracy': 0.939, 'loss': 0.17775545, 'precision': 0.9323906, 'recall': 0.9485613,

DELIVERABLES 3

To execute the program type in the command

python cnn_test.py

PROBLEM STATEMENT:

To write a program to use the pretrained custom CNN as described in the DELIVERABLE 1.

1. Accepts an input handwritten image which contains only 1 digit.
2. Process the image and display the original & processed image.
3. Use CNN to classify the image.
4. It should continuously request path to the image and output odd/even on the console.
5. Quit when q is pressed.

PROPOSED SOLUTION:

The program uses the model trained stored in the models folder. Function `cnn_model_fn` is imported from `cnn.py`.

```

model_fn = cnn.cnn_model_fn
# Create the Estimator
classifier = tf.estimator.Estimator(model_fn=model_fn, model_dir=model_dir,
params=parameters)

```

The program accepts the filename and reads the image as a grayscale. The image is then preprocessed to the desired format for the model prediction

```

predict_input_fn = tf.estimator.inputs.numpy_input_fn(x={"x":output},
shuffle=False)

```

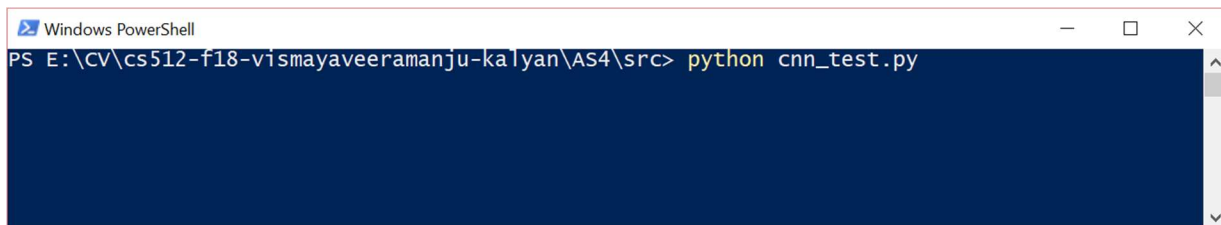
Which is used as an input to the predict function.

```

predict_results = classifier.predict(predict_input_fn)

```

IMPLEMENTATION DETAILS:



As the program runs it outputs asking you to enter the file name or 'q' to quit.

The image read is resized to a width & height of 28 & 28 as the model takes the image of that particular size.

The image is then processed using functions like

1. GaussianBlur with a filter size of 5.

```

blur = cv.GaussianBlur(inputImage,(5,5),0)

```

2. adaptiveThreshold

In this, the algorithm calculates the threshold for 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.

`cv.ADAPTIVE_THRESH_GAUSSIAN_C`: Threshold value is the weighted sum of neighbourhood values where weights are a gaussian window.

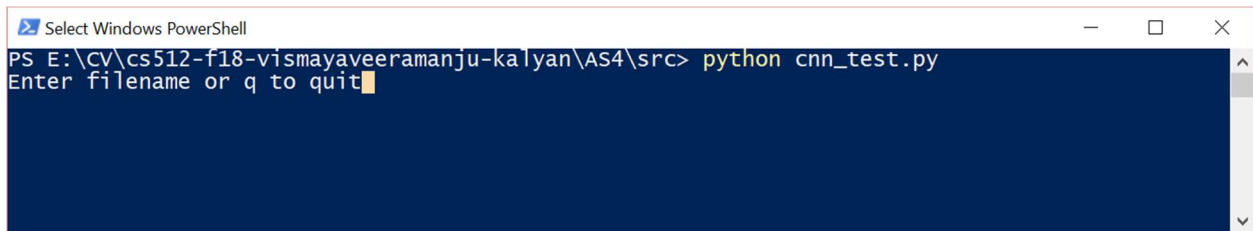
Block Size - It decides the size of neighbourhood area. 11

C - It is just a constant which is subtracted from the mean or weighted mean calculated. 2


```
Thresh= cv.adaptiveThreshold(blur,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,cv.THRESH_BINARY_INV,11,2)
```

The thresholded image is to convert the 28*28 image to the format of 784*1. As the required format is 1*784 we then reshape it the required form. The output of the reshaped image is fed as an input to predict using the tensorflow estimator of the CNN model. The result of which tells us which class the image belongs to.

RESULTS AND DISCUSSION:



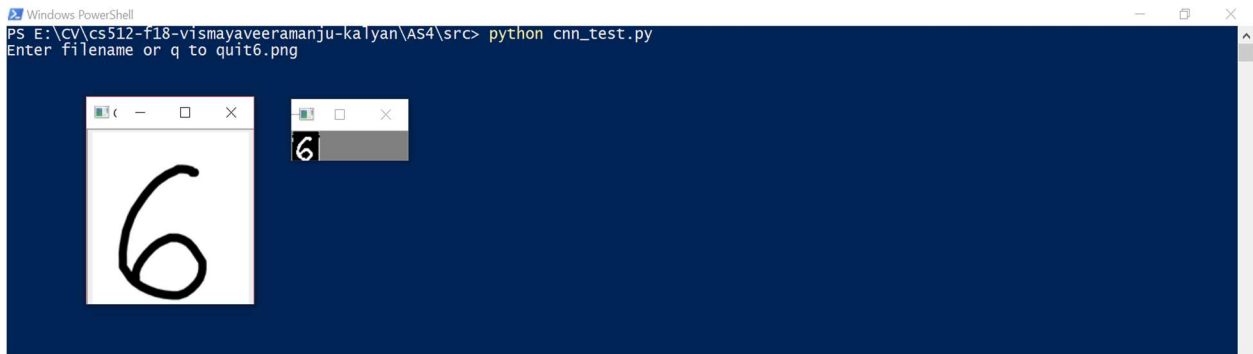
```
Select Windows PowerShell
PS E:\CV\cs512-f18-vismayaveeramanju-kalyan\AS4\src> python cnn_test.py
Enter filename or q to quit
```

There r 2 images for testing

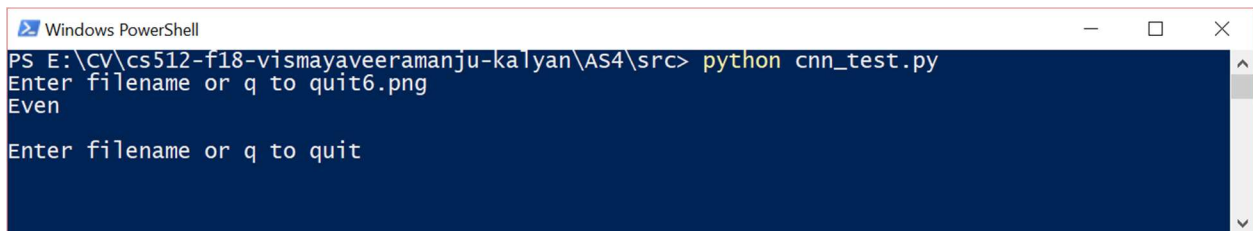
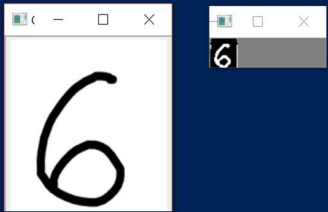
- 5.png
- 6.png.

The program uses the trained model to classify the image as odd or even.

EVEN CASE

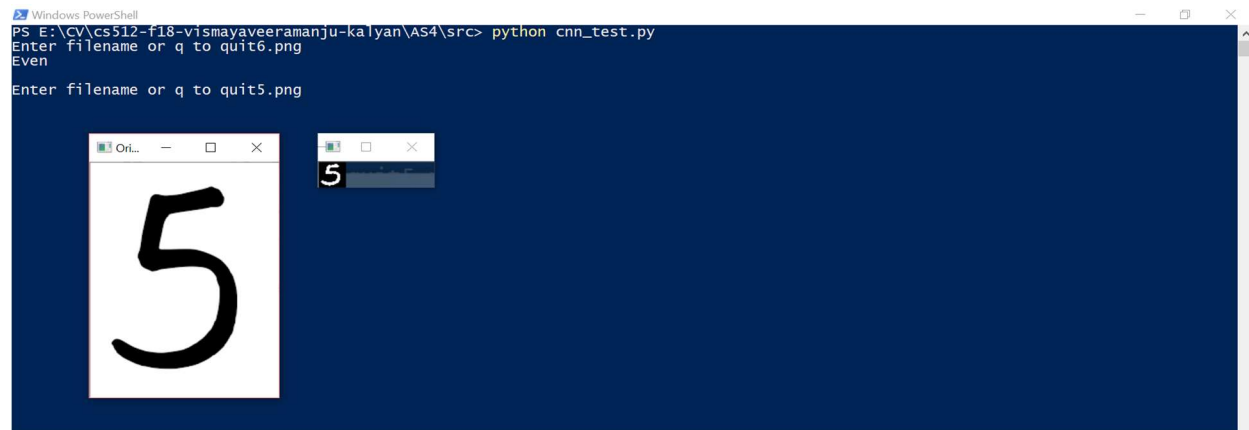


```
Windows PowerShell
PS E:\CV\cs512-f18-vismayaveeramanju-kalyan\AS4\src> python cnn_test.py
Enter filename or q to quit6.png
Even
```

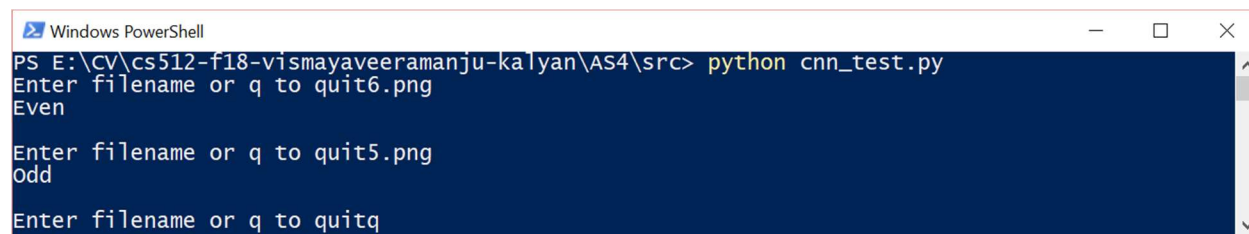


```
Windows PowerShell
PS E:\CV\cs512-f18-vismayaveeramanju-kalyan\AS4\src> python cnn_test.py
Enter filename or q to quit6.png
Even
Enter filename or q to quit
```

ODD CASE



```
Windows PowerShell
PS E:\CV\cs512-f18-vismayaveeramanju-kalyan\AS4\src> python cnn_test.py
Enter filename or q to quit6.png
Even
Enter filename or q to quit5.png
```



```
Windows PowerShell
PS E:\CV\cs512-f18-vismayaveeramanju-kalyan\AS4\src> python cnn_test.py
Enter filename or q to quit6.png
Even
Enter filename or q to quit5.png
Odd
Enter filename or q to quitq
```

REFERENCE:

Tensorflow documentation

<https://www.tensorflow.org/tutorials/estimators/cnn>

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

opencv documentation

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

<https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0>

colab vedio

<https://www.youtube.com/watch?v=4BVpzY6prJ0>