

Homework 2 – Problem 2.1

Introduction:

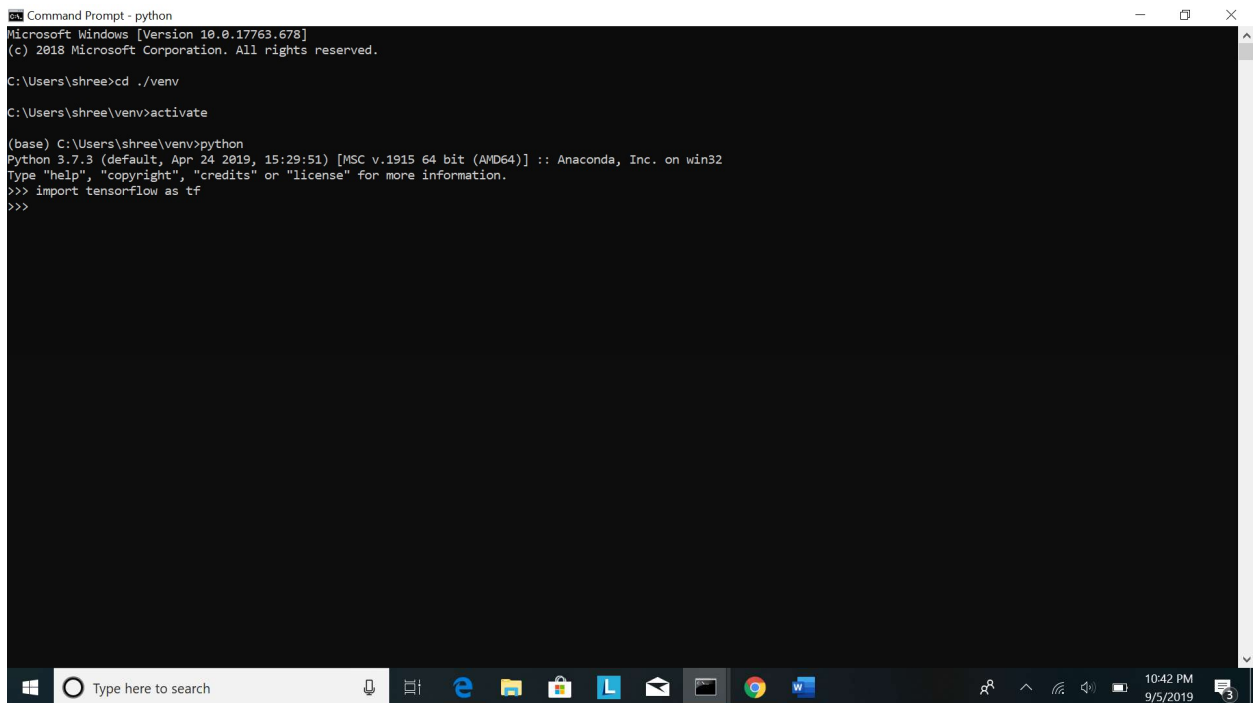
For this homework assignment, I have completed problem 2.1 to get some hands-on experience with the Deep learning Framework, Tensorflow, by building a k-class Image Classifier, here k=10.

The link to the Github repository with the .ipynb files is - <https://github.com/shreeshaa/BDA-homework-1>

Installation of Tensorflow:

I started by installing Tensorflow on my machine using the official website link <https://www.tensorflow.org/install/pip> which requires python3.6, pip and virtual environment.

Following are the screenshots after successful installation of python and Tensorflow:



```
Command Prompt - python
Microsoft Windows [Version 10.0.17763.678]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\shree>cd ./venv
C:\Users\shree\venv>activate

(base) C:\Users\shree\venv>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import tensorflow as tf
>>>
```

Deliverables:

1. provide URL of your open source code package and dataset download.

For the Tensorflow code of the CNN classifier I used the following Tensorflow tutorial link as reference and used it to build the k class CNN classifier:

https://www.tensorflow.org/beta/tutorials/images/intro_to_cnns

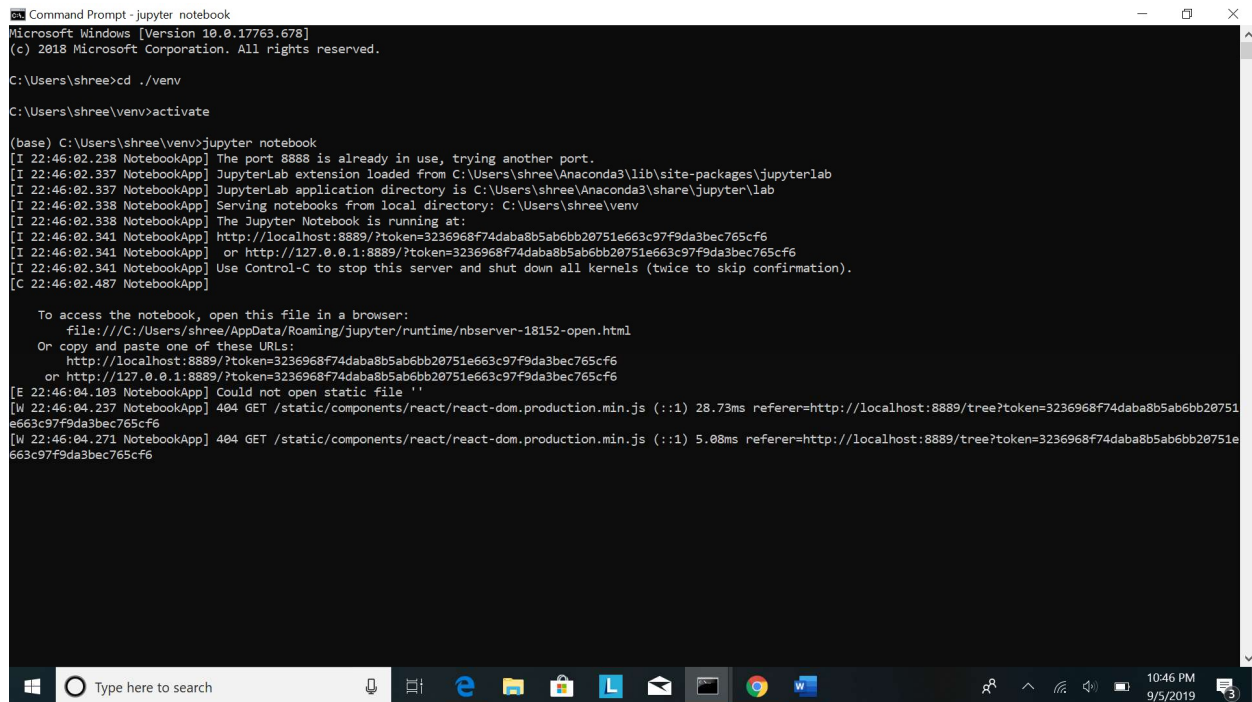
For the dataset I used the datasets available in Tensorflow itself. They are available in the tensorflow.keras.datasets package. The two datasets I used are listed in table1 and are follows:

Table 1:

Name	Classes	Resolution of each image	#Training images	#Testing images	Storage size of dataset	Storage size of each image
MNIST	10	28*28 pixel grayscale	60000	10000	54.95MB	1KB
FMNIST	10	28*28 pixel grayscale	60000	10000	55MB	1KB

2. Screen shots of your execution process/environments

Next, I started building my CNN classifier for subproblem 2.1 using Tensorflow in Jupyter Notebook. Following are the screenshots of the environment:



```

Command Prompt - jupyter notebook
Microsoft Windows [Version 10.0.17763.678]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\shree>cd ./venv

C:\Users\shree\venv>activate

(base) C:\Users\shree\venv>jupyter notebook
[I 22:46:02.238 NotebookApp] The port 8888 is already in use, trying another port.
[I 22:46:02.337 NotebookApp] JupyterLab extension loaded from C:\Users\shree\Anaconda3\lib\site-packages\jupyterlab
[I 22:46:02.337 NotebookApp] JupyterLab application directory is C:\Users\shree\Anaconda3\share\jupyter\lab
[I 22:46:02.338 NotebookApp] Serving notebooks from local directory: C:\Users\shree\venv
[I 22:46:02.338 NotebookApp] The Jupyter Notebook is running at:
[I 22:46:02.341 NotebookApp] http://localhost:8889/?token=3236968f74daba8b5ab6bb20751e663c97f9da3bec765cf6
[I 22:46:02.341 NotebookApp] or http://127.0.0.1:8889/?token=3236968f74daba8b5ab6bb20751e663c97f9da3bec765cf6
[I 22:46:02.341 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).

[C 22:46:02.487 NotebookApp]

To access the notebook, open this file in a browser:
    file:///C:/Users/shree/AppData/Roaming/jupyter/runtime/nbserver-18152-open.html
Or copy and paste one of these URLs:
    http://localhost:8889/?token=3236968f74daba8b5ab6bb20751e663c97f9da3bec765cf6
    or http://127.0.0.1:8889/?token=3236968f74daba8b5ab6bb20751e663c97f9da3bec765cf6
[E 22:46:04.103 NotebookApp] Could not open static file ''
[W 22:46:04.237 NotebookApp] 404 GET /static/components/react/react-dom.production.min.js (::1) 28.73ms referer=http://localhost:8889/tree?token=3236968f74daba8b5ab6bb20751e663c97f9da3bec765cf6
[W 22:46:04.271 NotebookApp] 404 GET /static/components/react/react-dom.production.min.js (::1) 5.08ms referer=http://localhost:8889/tree?token=3236968f74daba8b5ab6bb20751e663c97f9da3bec765cf6
  
```

The top screenshot shows a Jupyter Notebook titled "HW1_PROB2.1_MNIST-2" with the following code:

```
In [34]: #Source:https://gatech.instructure.com/courses/62736/files/folder/Advanced%20DNN%20Software%20%26%20Tools
import tensorflow as tf
from tensorflow.keras import keras
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatten
import numpy as np
import os

os.environ['KMP_DUPLICATE_LIB_OK']='True'

np.random.seed(3)

(X_train, Y_train), (X_test_org, Y_test_org) = mnist.load_data()

In [35]: X_val = X_train[7000:8000]
Y_val = Y_train[7000:8000]
X_train = X_train[4000:7000]
Y_train = Y_train[4000:7000]
X_test = X_test_org[1000:2000]
Y_test = Y_test_org[1000:2000]

X_train = X_train.reshape(3000, 28,28,1).astype('float32') / 255.0
X_val = X_val.reshape(1000, 28,28,1).astype('float32') / 255.0
```

The bottom screenshot shows the Jupyter file browser view with the following files and folders:

Name	Last Modified	File size
Include	13 days ago	
Lib	7 days ago	
Scripts	7 days ago	
tol	7 days ago	
CIFAR10.ipynb	a day ago	4.84 kB
CIFAR100_ASG1.ipynb	an hour ago	149 kB
CIFAR10_ASG1.ipynb	8 hours ago	146 kB
FMNIST_ASG1.ipynb	9 hours ago	98.8 kB
FMNIST_BDA_FROM_PPT.ipynb	a day ago	202 kB
HW1_PROB2.1_FMNIST.ipynb	6 hours ago	107 kB
HW1_PROB2.1_MNIST-2.ipynb	18 minutes ago	98.1 kB
HW1_PROB2.1_MNIST.ipynb	12 minutes ago	97 kB
MNIST_TF_TUTORIAL.ipynb	2 days ago	3.8 kB
BigDataMNISTModel.H5	a day ago	23.3 kB

3. Input Analysis:

- the input dataset (size, resolution, storage size in KB or MB per image, storage size of dataset in MB or GB).**

I created my datasets using the main datasets listed in Table1. I split the MNIST datasets to create 4 new datasets. As we know, the MNIST and FMNIST datasets are present in a random order. Thus, splitting the X_train and Y_train arrays into subarrays containing the first 4000 and the next 4000 images, will give us 4 datasets containing random set of images, evenly distributed across each class. Similarly, the X_test and Y_test arrays have been split to contain the first 1000 and the next 1000 outputs.

The code for splitting is as given in the screenshot below:

```

import os

os.environ['KMP_DUPLICATE_LIB_OK']='True'

np.random.seed(3)

(X_train, Y_train), (X_test_org, Y_test_org) = mnist.load_data()

In [2]: X_val = X_train[3000:4000]
        Y_val = Y_train[3000:4000]
        X_train = X_train[:3000]
        Y_train = Y_train[:3000]
        X_test = X_test_org[:1000]
        Y_test = Y_test_org[:1000]

        X_train = X_train.reshape(3000, 28, 28, 1).astype('float32') / 255.0
        X_val = X_val.reshape(1000, 28, 28, 1).astype('float32') / 255.0
        X_test = X_test.reshape(1000, 28, 28, 1).astype('float32') / 255.0

In [3]: #train_rand_idxes = np.random.choice(5000, 8000)
        #val_rand_idxes = np.random.choice(1000, 2000)

        #X_train = X_train[train_rand_idxes]
        #Y_train = Y_train[train_rand_idxes]
        #X_val = X_val[val_rand_idxes]
        #Y_val = Y_val[val_rand_idxes]

```

Datasets used:

Name	Classes	Resolution of each image	#Training images	#Testing images	Storage size of each image
FMNIST(first 5000 images with index 1-5000)	10	28*28 pixel grayscale	4000	1000	1KB
MNIST (first 5000 images with index 1-5000)	10	28*28 pixel grayscale	4000	1000	1KB
MNIST (next 5000 images with index 5000-10000)	10	28*28 pixel grayscale	4000	1000	1KB
MNIST (images with index 10000-15000)	10	28*28 pixel grayscale	4000	1000	1KB
MNIST(images with index 15000-20000)	10	28*28 pixel grayscale	4000	1000	1KB

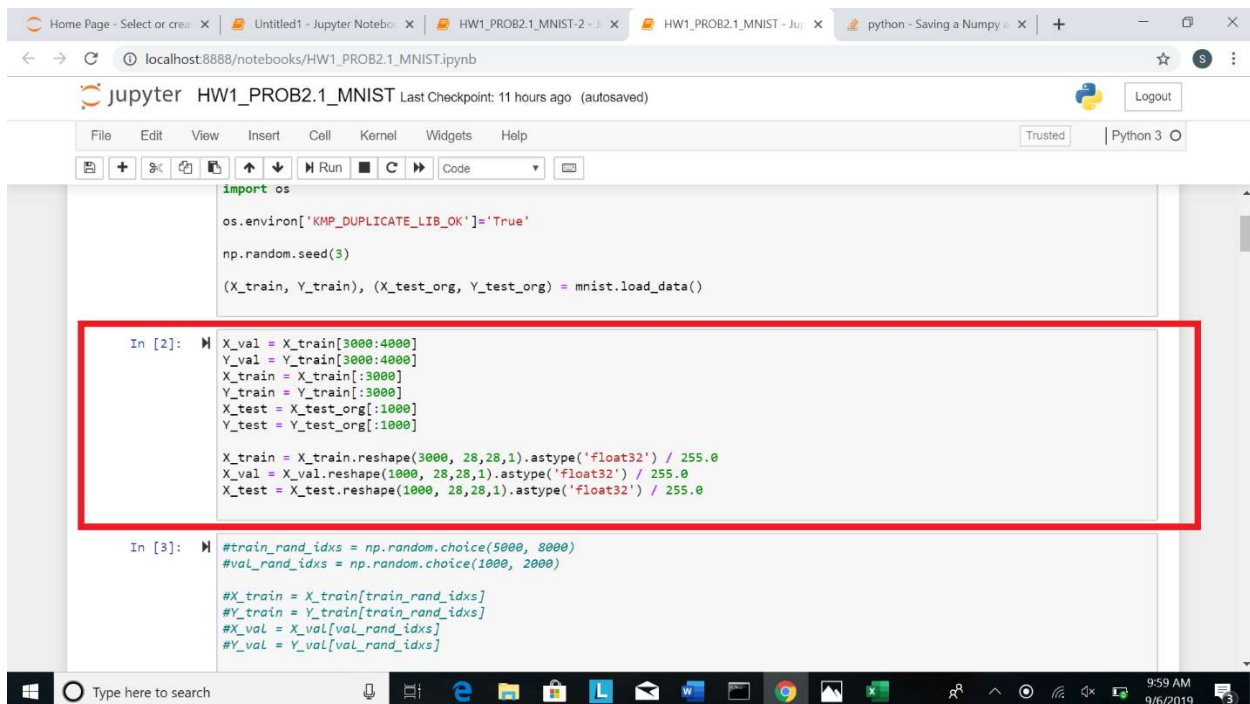
b. choose and show 2 sample images per class for all K classes in each of the four datasets

The classes are the same in all the 4 datasets. And they are as given below:

Class 1 - 0	0	0
Class 2 - 1	1	1
Class 3 - 2	2	2
Class 4 - 3	3	3
Class 5 - 4	4	4
Class 6 - 5	5	5
Class 7 - 6	6	6
Class 8 - 7	7	7
Class 9 - 8	8	8
Class 10 - 9	9	9

c. the training v.s. testing data split ratio and size used in your CNN training.

The question asked to use a ratio of 8:2 for training and testing. Thus, I have split each dataset containing 5000 images into sizes, $0.8 \times 5000 = 4000$ and $0.2 \times 5000 = 1000$. Further, I have split the training data to contain 1000 validation images and 3000 training images.



```
import os

os.environ['KMP_DUPLICATE_LIB_OK']='True'

np.random.seed(3)

(X_train, Y_train), (X_test_org, Y_test_org) = mnist.load_data()

In [2]: X_val = X_train[3000:4000]
        Y_val = Y_train[3000:4000]
        X_train = X_train[:3000]
        Y_train = Y_train[:3000]
        X_test = X_test_org[:1000]
        Y_test = Y_test_org[:1000]

        X_train = X_train.reshape(3000, 28, 28, 1).astype('float32') / 255.0
        X_val = X_val.reshape(1000, 28, 28, 1).astype('float32') / 255.0
        X_test = X_test.reshape(1000, 28, 28, 1).astype('float32') / 255.0

In [3]: #train_rand_idx = np.random.choice(5000, 8000)
        #val_rand_idx = np.random.choice(1000, 2000)

        #X_train = X_train[train_rand_idx]
        #Y_train = Y_train[train_rand_idx]
        #X_val = X_val[val_rand_idx]
        #Y_val = Y_val[val_rand_idx]
```

d. Default Neural network model and hyperparameters

The default structures of the classifier used for the classification task of the 4 datasets is as follows:

Classifier 1

This classifier consists of 8 layers. This classifier has a LeNet5-like data structure where we have convolution and pooling layers and finally, 2 fully connected layers. They are as follows:

- Convolutional layer consisting of 32 3*3 filters with 'Relu' activation function
- 2*2 Pooling layer
- Convolutional layer consisting of 64 3*3 filters with 'Relu' activation function
- 2*2 Pooling layer
- Convolutional layer consisting of 64 3*3 filters with 'Relu' activation function
- Flatten
- Dense with 64 neurons and 'Relu' activation function
- Dense with 10 neurons and 'Softmax' activation function

Neuron Size	3
Number of weight filters	32+64+64=160
Size of weight filters	3
Batch size	10
Number of epochs	5
Number of iterations	400

The screenshot shows a Jupyter Notebook window titled 'HW1_PROB2.1_MNIST-3'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and code execution. The main area displays a warning message from TensorFlow regarding the deprecated 'dtype' argument in the 'Variances' class. Below the warning, a summary of the neural network layers is shown, including the layer type, output shape, and number of parameters.

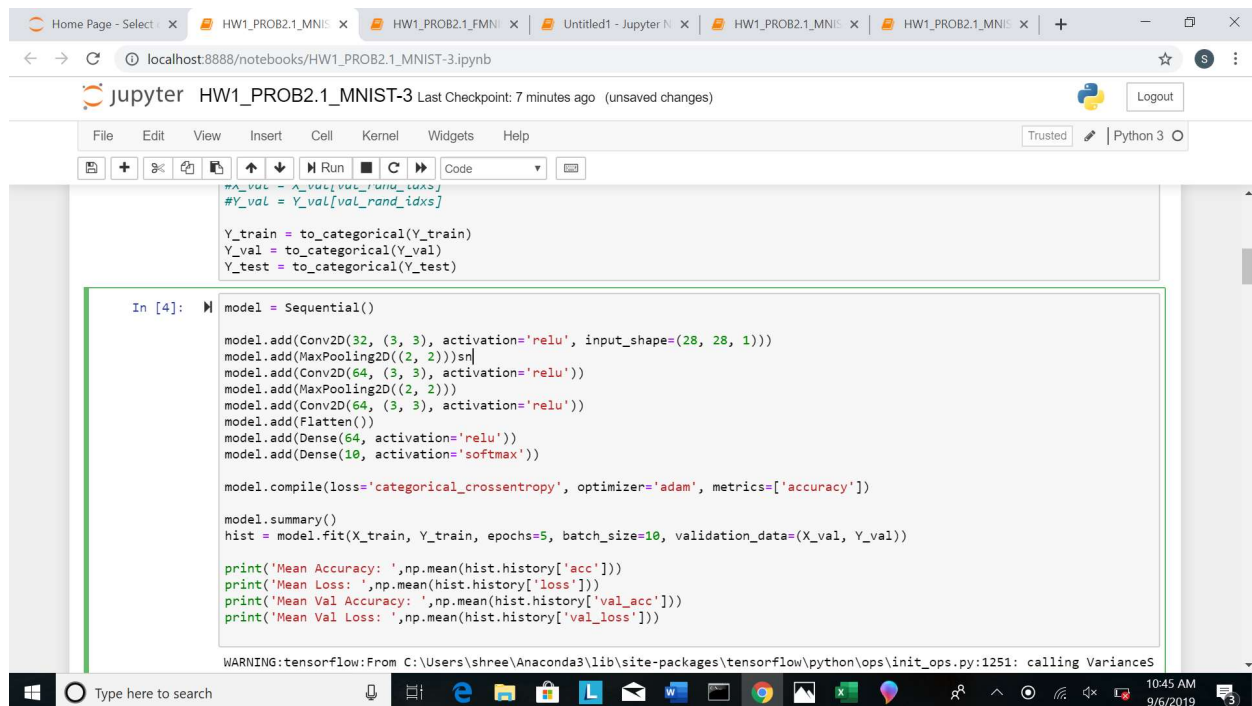
```
WARNING:tensorflow:From C:\Users\shree\Anaconda3\lib\site-packages\tensorflow\python\ops\init_ops.py:1251: calling Variances
calling.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36928
dense_1 (Dense)	(None, 10)	650

Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0

The loss function used in this classifier is - 'categorical_crossentropy', the optimizer is - 'adam'.

The number of epochs = 5 and the batch size = 10



The screenshot shows a Jupyter Notebook window titled 'HW1_PROB2.1_MNIST-3'. The code in the notebook defines a sequential model with three convolutional layers, two pooling layers, a flatten layer, and two dense layers. The model is compiled with 'categorical_crossentropy' loss and 'adam' optimizer. It is then trained for 5 epochs with a batch size of 10. The code also includes print statements to display the mean accuracy and loss for both training and validation sets.

```
#X_val = X_val[val_rand_idxs]
#Y_val = Y_val[val_rand_idxs]

Y_train = to_categorical(Y_train)
Y_val = to_categorical(Y_val)
Y_test = to_categorical(Y_test)

In [4]: model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
hist = model.fit(X_train, Y_train, epochs=5, batch_size=10, validation_data=(X_val, Y_val))

print('Mean Accuracy: ', np.mean(hist.history['acc']))
print('Mean Loss: ', np.mean(hist.history['loss']))
print('Mean Val Accuracy: ', np.mean(hist.history['val_acc']))
print('Mean Val Loss: ', np.mean(hist.history['val_loss']))
```

WARNING:tensorflow:From C:\Users\shree\Anaconda3\lib\site-packages\tensorflow\python\ops\init_ops.py:1251: calling VarianceS

Classifier 2:

This classifier is like classifier 1 except number of epochs = 25. The layers in this classifier are as follows:

- Convolutional layer consisting of 32 3*3 filters with 'Relu' activation function
- 2*2 Pooling layer
- Convolutional layer consisting of 64 3*3 filters with 'Relu' activation function
- 2*2 Pooling layer
- Convolutional layer consisting of 64 3*3 filters with 'Relu' activation function
- Flatten
- Dense with 64 neurons and 'Relu' activation function
- Dense with 10 neurons and 'Softmax' activation function

Neuron Size	3
Number of weight filters	32+64+64=160
Size of weight filters	3
Batch size	10
Number of epochs	25
Number of iterations	400

WARNING:tensorflow:From C:\Users\shree\Anaconda3\lib\site-packages\tensorflow\python\ops\init_ops.py:1251: calling VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version. Instructions for updating: Call initializer instance with the dtype argument instead of passing it to the constructor

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36928
dense_1 (Dense)	(None, 10)	650

Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0

The loss function used in this classifier is - 'categorical_crossentropy', the optimizer is - 'adam'.

The number of epochs = 25 and the batch size = 10

Classifier 3:

This classifier consists of layers as follows:

- Flatten
- Dense with 128 neurons and 'Relu' activation function
- Dense with 10 neurons and 'Softmax' activation function

Neuron Size	-
Number of weight filters	128+10 = 138
Size of weight filters	-
Batch size	10
Number of epochs	25
Number of iterations	400

The screenshot shows a Jupyter Notebook interface with the following content:

```
print('Mean Accuracy: ', np.mean(hist.history['acc']))
print('Mean Loss: ', np.mean(hist.history['loss']))
print('Mean Val Accuracy: ', np.mean(hist.history['val_acc']))
print('Mean Val Loss: ', np.mean(hist.history['val_loss']))
```

Model: "sequential_9"

Layer (type)	Output Shape	Param #
flatten_9 (Flatten)	(None, 784)	0
dense_18 (Dense)	(None, 128)	100480
dense_19 (Dense)	(None, 10)	1290

Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0

Train on 3000 samples, validate on 1000 samples

```
Epoch 1/25
3000/3000 [=====] - 0s 160us/sample - loss: 0.8250 - acc: 0.7100 - val_loss: 0.6393 - val_acc: 0.78
30
Epoch 2/25
3000/3000 [=====] - 0s 112us/sample - loss: 0.5566 - acc: 0.7963 - val_loss: 0.5970 - val_acc: 0.77
80
Epoch 3/25
3000/3000 [=====] - 0s 108us/sample - loss: 0.4558 - acc: 0.8290 - val_loss: 0.5213 - val_acc: 0.81
100
```

The loss function used in this classifier is - 'categorical_crossentropy', the optimizer is - 'adam'.

The number of epochs = 25 and the batch size = 10

The screenshot shows a Jupyter Notebook interface with the following code:

```
#Y_val = Y_val[val_rand_idxs]

Y_train = to_categorical(Y_train)
Y_val = to_categorical(Y_val)
Y_test = to_categorical(Y_test)

In [37]: model = Sequential()

model.add(Flatten(input_shape=(28, 28, 1)))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
hist = model.fit(X_train, Y_train, epochs=25, batch_size=10, validation_data=(X_val, Y_val))

print('Mean Accuracy: ', np.mean(hist.history['acc']))
print('Mean Loss: ', np.mean(hist.history['loss']))
print('Mean Val Accuracy: ', np.mean(hist.history['val_acc']))
print('Mean Val Loss: ', np.mean(hist.history['val_loss']))
```

Model: "sequential_9"

Classifier 4:

This classifier consists of layers as follows:

- Convolutional layer consisting of 64 2*2 filters with 'Relu' activation function
- 2*2 Pooling layer
- Dropout layer with rate = 0.3
- Convolutional layer consisting of 32 2*2 filters with 'Relu' activation function
- 2*2 Pooling layer
- Dropout layer with rate = 0.3

- Flatten
- Dense with 256 neurons and 'Relu' activation function
- Dropout layer with rate = 0.5
- Dense with 10 neurons and 'Softmax' activation function

Neuron Size	2
Number of weight filters	64+32=96
Size of weight filters	2
Batch size	20
Number of epochs	20
Number of iterations	200

Model: "sequential_9"		
Layer (type)	Output Shape	Param #
conv2d_21 (Conv2D)	(None, 28, 28, 64)	320
max_pooling2d_15 (MaxPooling)	(None, 14, 14, 64)	0
dropout_3 (Dropout)	(None, 14, 14, 64)	0
conv2d_22 (Conv2D)	(None, 14, 14, 32)	8224
max_pooling2d_16 (MaxPooling)	(None, 7, 7, 32)	0
dropout_4 (Dropout)	(None, 7, 7, 32)	0
flatten_8 (Flatten)	(None, 1568)	0
dense_16 (Dense)	(None, 256)	401664
dropout_5 (Dropout)	(None, 256)	0
dense_17 (Dense)	(None, 10)	2570
Total params: 412,778		
Trainable params: 412,778		
Non-trainable params: 0		

The loss function used in this classifier is - 'categorical_crossentropy', the optimizer is - 'adam'.

The number of epochs = 20 and the batch size = 20

```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
+ %< [Run] [Code]
Y_train = to_categorical(Y_train)
Y_val = to_categorical(Y_val)
Y_test = to_categorical(Y_test)

In [53]: model = Sequential()

model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu', input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
hist = model.fit(X_train, Y_train, epochs=20, batch_size=20, validation_data=(X_val, Y_val))

print('Mean Accuracy: ', np.mean(hist.history['acc']))
print('Mean Loss: ', np.mean(hist.history['loss']))
print('Mean Val Accuracy: ', np.mean(hist.history['val_acc']))
print('Mean Val Loss: ', np.mean(hist.history['val_loss']))

```

e. Default error threshold

We are not using default error threshold, we are instead using number of epochs in all our models. Hence, our definition of convergence is when the specified number of epochs are done, as opposed to an accuracy percentage or error value/threshold. Thus, the number of epochs in each model is the error threshold for us.

4. Analysis:

a. You are asked to provide a table to compare the 4 CNN classifiers

Dataset	Model 1 with epochs = 5	Model 2 with epochs = 25	Model 3	Model 4
MNIST[0-5000]	Training Time - 6s, Training Accuracy - 0.93, Testing Time - 1s, Testing Accuracy - 0.95	Training Time - 4.92s, Training Accuracy- 0.98, Testing Time - 1s, Testing Accuracy - 0.972	Training Time-0.9s, Training Accuracy- 0.97, Testing Time-0s, Testing Accuracy-0.913	Training Time-3.94s, Training Accuracy-0.94, Testing Time-1s, Testing Accuracy-0.967
MNIST[5000-10000]	Training Time - 5.8 s, Training Accuracy - 0.936, Testing Time-1s, Testing Accuracy - 0.924	Training Time-5.28s, Training Accuracy-0.984, Testing Time-1 s, Testing Accuracy-0.96	Training Time-1s, Training Accuracy-0.979, Testing Time-0s, Testing Accuracy-0.887	Training Time-6.6s, Training Accuracy-0.945, Testing Time-0s, Testing Accuracy-0.95
MNIST[10000-15000]	Training Time - 6s, Training Accuracy - 0.936, Testing Time - 1 sec, Testing Accuracy - 0.948	Training Time – 5s, Training Accuracy-0.98, Testing Time-1s, Testing Accuracy-0.96	Training Time-0.8s, Training Accuracy-0.982, Testing Time-0s, Testing Accuracy-0.904	Training Time-7.1s, Training Accuracy-0.94, Testing Time-0.959, Testing Accuracy-1s
MNIST[15000-25000]	Training Time-3s, Training Accuracy- 0.962 , Testing Time - 0sec, Testing Accuracy - 0.965	Training Time-5.12s, Training Accuracy-0.98, Testing Time-0s, Testing Accuracy-0.98	Training Time-1s, Training Accuracy-0.9318, Testing Time-0s, Testing Accuracy-0.929	Training Time-4.95s, Training Accuracy-0.952, Testing Time-0s, Testing Accuracy-0.959
FMNIST[5000-10000]	Training Time - 6.2s, Training Accuracy - 0.7755 , Testing Time - 1s, Testing Accuracy-0.834	Training Time-5.4s, Training Accuracy-0.91, Testing Time - 0s, Testing Accuracy - 0.85	Training Time-1.2s, Training Accuracy-0.90, Testing Time-0s, Testing Accuracy-0.834	Training Time-2.05s, Training Accuracy-0.813, Testing Time-1s, Testing Accuracy=0.872

b. You are asked to record the trained model size in MB for both classifiers in the above table for all 4 CNN classifiers

- i. Model1 – 1.14 MB
- ii. Model2– 1.14MB
- iii. Model2 – 1.219MB
- iv. Model3 – 4.881MB

c. Test accuracy and time for outlier dataset

Model 1	Model 2	Model 3	Model 4
Test Accuracy-0.115, Test Time-186 micro sec/sample	Test Accuracy-0.054, Test Time-189 micro sec/sample	Test Accuracy-0.106, Test Time-58 micro sec/sample	Test Accuracy-0.099, Test Time-216 micro sec/sample

To perform an outlier test, data from the FMNIST dataset was provided as an input to the classifier 1 trained and optimized for the MNIST dataset.

On using the FMNIST dataset as the test dataset for a first model with 5 epochs trained with the MNIST dataset we got a very low accuracy of 0.115.

```
In [65]: (OX_train, OY_train), (OX_test, OY_test) = fashion_mnist.load_data()
OX_test = OX_test[:1000]
OY_test = OY_test[:1000]
OY_test = to_categorical(OY_test)
OX_test_reshape = OX_test.reshape(1000, 28,28,1).astype('float32') / 255.0
test_loss, test_acc = model.evaluate(OX_test_reshape, OY_test, batch_size=10)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)

1000/1000 [=====] - 0s 186us/sample - loss: 4.1174 - acc: 0.1150
Test loss: 4.117417442798614
Test accuracy: 0.115
```

Please find below the 5 examples used for Outlier testing:

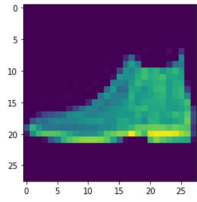
These are images from the FMNIST dataset supplied to our model 1 trained with MNIST dataset.

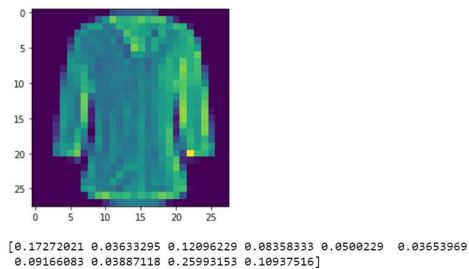
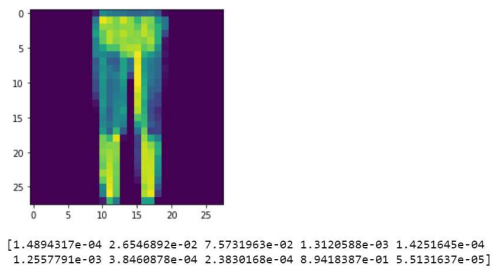
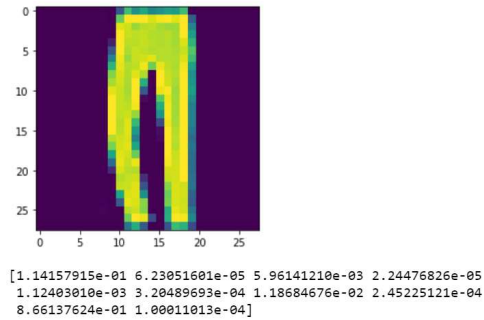
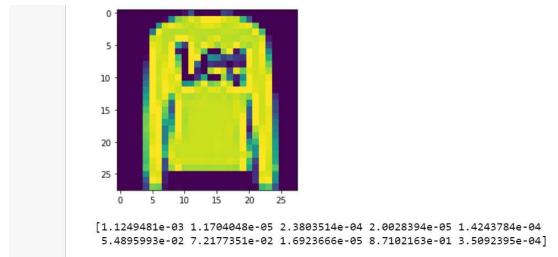
```
In [9]: (OX_train, OY_train), (OX_test, OY_test) = fashion_mnist.load_data()
OX_test = OX_test[:1000]

OX_test_reshape = OX_test.reshape(1000, 28,28,1).astype('float32') / 255.0
predictions = model.predict(OX_test_reshape)

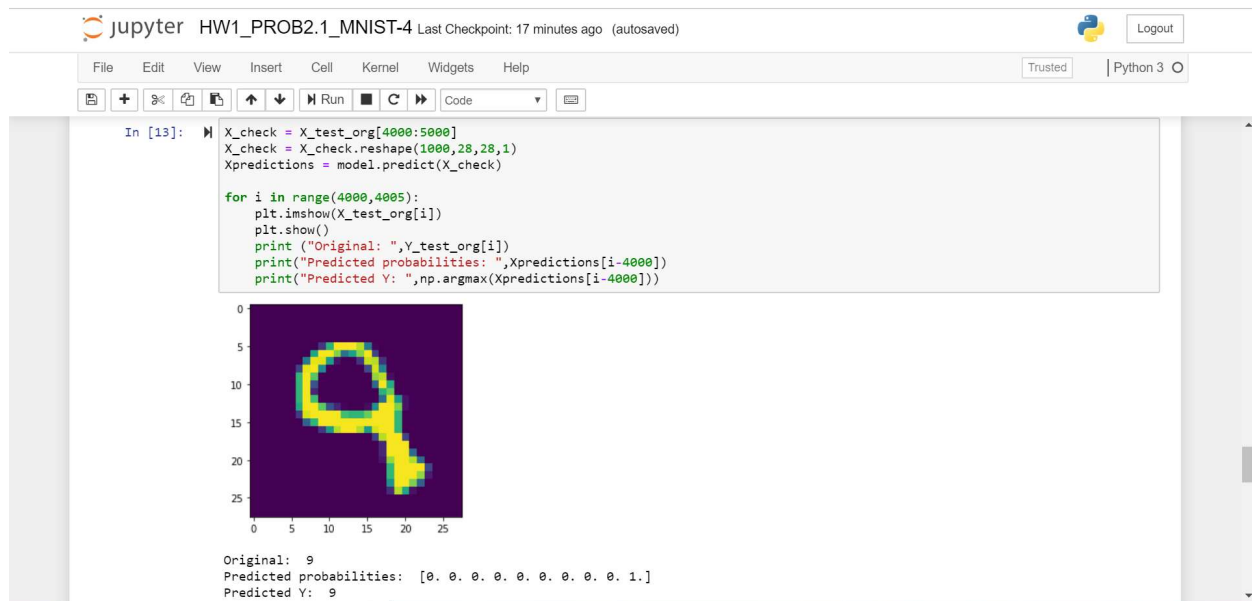
In [10]: for i in range(5):
plt.imshow(OX_test[i])
plt.show()
print(predictions[i])

[4.9073310e-03 4.8056915e-02 8.6369818e-01 5.0074114e-03 7.0940790e-05
 6.2821000e-03 6.8286315e-02 2.1732425e-04 3.4250855e-03 4.8412490e-05]
```



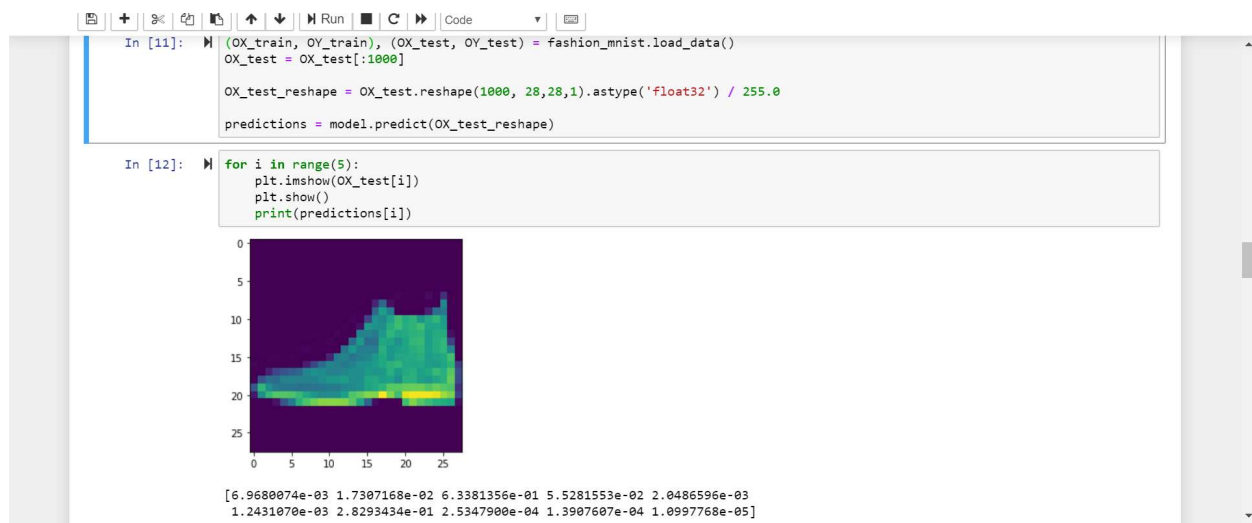


When an image from the mnist dataset itself, outside of the training data was provided, and the probabilities computed, as output by the model were observed, we saw that one of the classes has a distinctly high probability as compared to the others, see image below:



As we can see in the image, all the values of array Y are 0 and only one value is 1, which corresponds to the class for number 9

However, the same cannot be said when an image from the FMNIST dataset is provided to the model. All the 10 probability values were similar and had very low values.

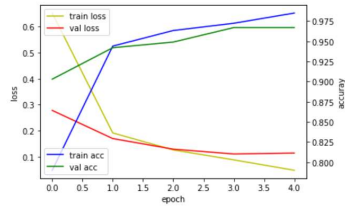


Thus, we can see how the outlier test succeeds. The given model has very low accuracy and does not have a distinctly high probability for a particular class, when an image outside the dataset is provided as input.

d. You are asked to make at least three observations from your experimental comparison of the 4 CNN classifiers

1. Let's look at how the classifier 1 performs on each of the 4 datasets.

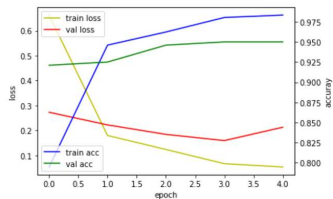
The results of training this classifier using first dataset are as follows:



Statistics:

Mean Accuracy: 0.93093336
Mean Loss: 0.2212755664913081
Mean Val Accuracy: 0.94560003
Mean Val Loss: 0.16049843223730567
Test loss: 0.14692081421962938
Test accuracy: 0.95

The results of training this classifier using the second dataset are as follows:

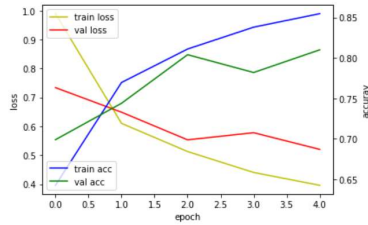


Statistics:

Mean Accuracy: 0.9333334
Mean Loss: 0.21710453927413614
Mean Val Accuracy: 0.9384
Mean Val Loss: 0.21013142218880237
Test loss: 0.29821544875972905
Test accuracy: 0.924

Similar results were observed for the third dataset.

We found that no real difference was observed on training a classifier with same number of images from the same dataset even though we pick a random set of images. However, if we change the dataset itself, for example, if we use the first 5000 training and testing images of the FMNIST dataset, we see a difference in the training of the classifier. Refer to the graph below:



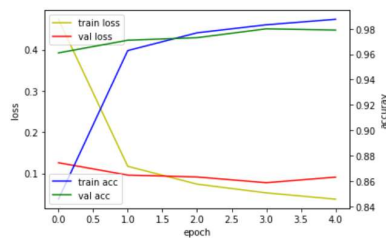
Statistics:

Mean Accuracy: 0.7832
Mean Loss: 0.5896927571423973
Mean Val Accuracy: 0.76780003
Mean Val Loss: 0.6067126127295196
Test loss: 0.4799393391003832
Test accuracy: 0.834

When the main dataset was changed from MNIST to FMNIST, we observed a drop in the testing accuracy of approximately 10%. This is probably because the Neural Network Classifier we have selected, is similar to the one used in the Tensorflow tutorial for the MNIST database. Thus, this model probably has optimized for that dataset and does not work as well on the other dataset.

- Second difference in accuracy was observed when we trained the model using two times the training data. Average training accuracy went up from 0.93 to 0.96 when we used 10000 entries in the training and testing datasets instead of 5000.

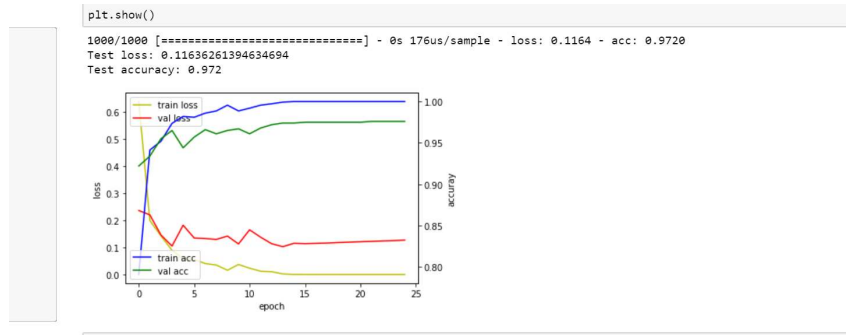
```
1000/1000 [=====] - 0s 166us/sample - loss: 0.1370 - acc: 0.9650
Test loss: 0.13701344164659532
Test accuracy: 0.965
```



Dataset	Model 1
MNIST[0-5000]	Training Time - 6s, Training Accuracy - 0.93, Testing Time - 1s, Testing Accuracy - 0.95
MNIST[5000-10000]	Training Time - 5.8 s, Training Accuracy - 0.936, Testing Time-1s, Testing Accuracy - 0.924
MNIST[10000-15000]	Training Time - 6s, Training Accuracy - 0.936, Testing Time - 1 sec, Testing Accuracy - 0.948
MNIST[15000-25000]	Training Time-3s, Training Accuracy- 0.962 , Testing Time - 0sec, Testing Accuracy - 0.965
FMNIST[5000-10000]	Training Time - 6.2s, Training Accuracy - 0.7755 , Testing Time - 1s, Testing Accuracy-0.834

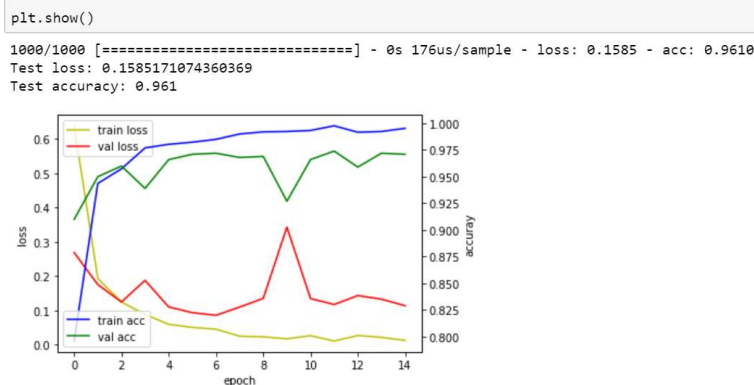
- Now, let's see how the results are affected by a change in the number of epochs. Following was observed when we trained model 2 using dataset 1. As we can see in the image, the graph is

much smoother than when epochs = 5 and seems to have reached a constant value towards the end.



Also, a lot of spikes and drops are observed in the dataset at the beginning. This is probably because more learning is learnt in the earlier epochs, as compared to the later stages because we get closer to convergence. Another observation is that since the graph tends to flatten after 15 epochs, we do not necessarily need 25. Same results can probably be observed with 15 epochs as well.

When we change the number of epochs to 15 and test the data, we observe a testing accuracy of 0.96 which is still very close to the one obtained with 25 epochs of 0.97, as opposed to the testing accuracy of 0.93 observed with 5 epochs.



This has been summarized in the table below:

	Epochs = 5	Epochs = 25	Epochs = 15
MNIST[0-5000]	Training Accuracy - 0.93, Testing Accuracy - 0.95	Training Accuracy- 0.98, Testing Accuracy - 0.972	Training Accuracy- 0.97, Testing Accuracy - 0.96

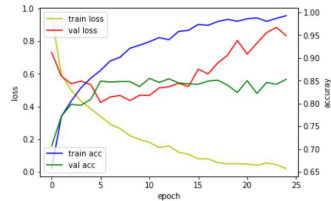
Thus, we observe that as the value of the epochs increases, the benefit obtained by increasing the number of epochs reduces.

- When the model 2 was trained using the FMNIST dataset, the accuracy was still low and as we can see in the graph, the accuracy and loss have not flattened (or converged). Thus, this model is probably not the best for the FMNIST dataset. This was the conclusion from the earlier model

with epoch = 5 as well. Thus, for the FMNIST dataset, we probably need a different architecture for the classifier.

```
loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')
plt.show()

1000/1000 [=====] - 0s 163us/sample - loss: 0.7940 - acc: 0.8580
Test loss: 0.7940268568826309
Test accuracy: 0.858
```



- On looking at the results for the dataset with 10000 training samples, no major difference is observed in accuracy in model 2 as that observed in model 1. When the number of epochs is high, the effect of an increase in size of the training sample is not as pronounced as it is when the number of epochs is lower.

The following has been summarized in the table below:

Dataset	Model with epochs = 5	Model with epochs = 25
MNIST[0-5000]	Training Time - 6s, Training Accuracy - 0.93, Testing Time - 1s, Testing Accuracy - 0.95	Training Time - 4.92s, Training Accuracy- 0.98, Testing Time - 1s, Testing Accuracy - 0.972
MNIST[5000-10000]	Training Time - 5.8 s, Training Accuracy - 0.936, Testing Time-1s, Testing Accuracy - 0.924	Training Time-5.28s, Training Accuracy- 0.984, Testing Time-1 s, Testing Accuracy-0.96
MNIST[10000-15000]	Training Time - 6s, Training Accuracy - 0.936, Testing Time - 1 sec, Testing Accuracy - 0.948	Training Time – 5s, Training Accuracy- 0.98, Testing Time-1s, Testing Accuracy- 0.96
MNIST[15000-25000]	Training Time-3s, Training Accuracy- 0.962 , Testing Time - 0sec, Testing Accuracy - 0.965	Training Time-5.12s, Training Accuracy- 0.98, Testing Time-0s, Testing Accuracy- 0.98
FMNIST[5000-10000]	Training Time - 6.2s, Training Accuracy - 0.7755 , Testing Time - 1s, Testing Accuracy-0.834	Training Time-5.4s, Training Accuracy- 0.91, Testing Time - 0s, Testing Accuracy - 0.85

- We observe that the number of epochs makes no difference to the size of the model in KB by comparing the size of models 1 and 2.
- The running time of the classifier which has only the Flatten and the Dense layers i.e. layers with no convolution filter that only have fully connected neuron layers is significantly lesser.