**Homework 2 – Problem 2.1**

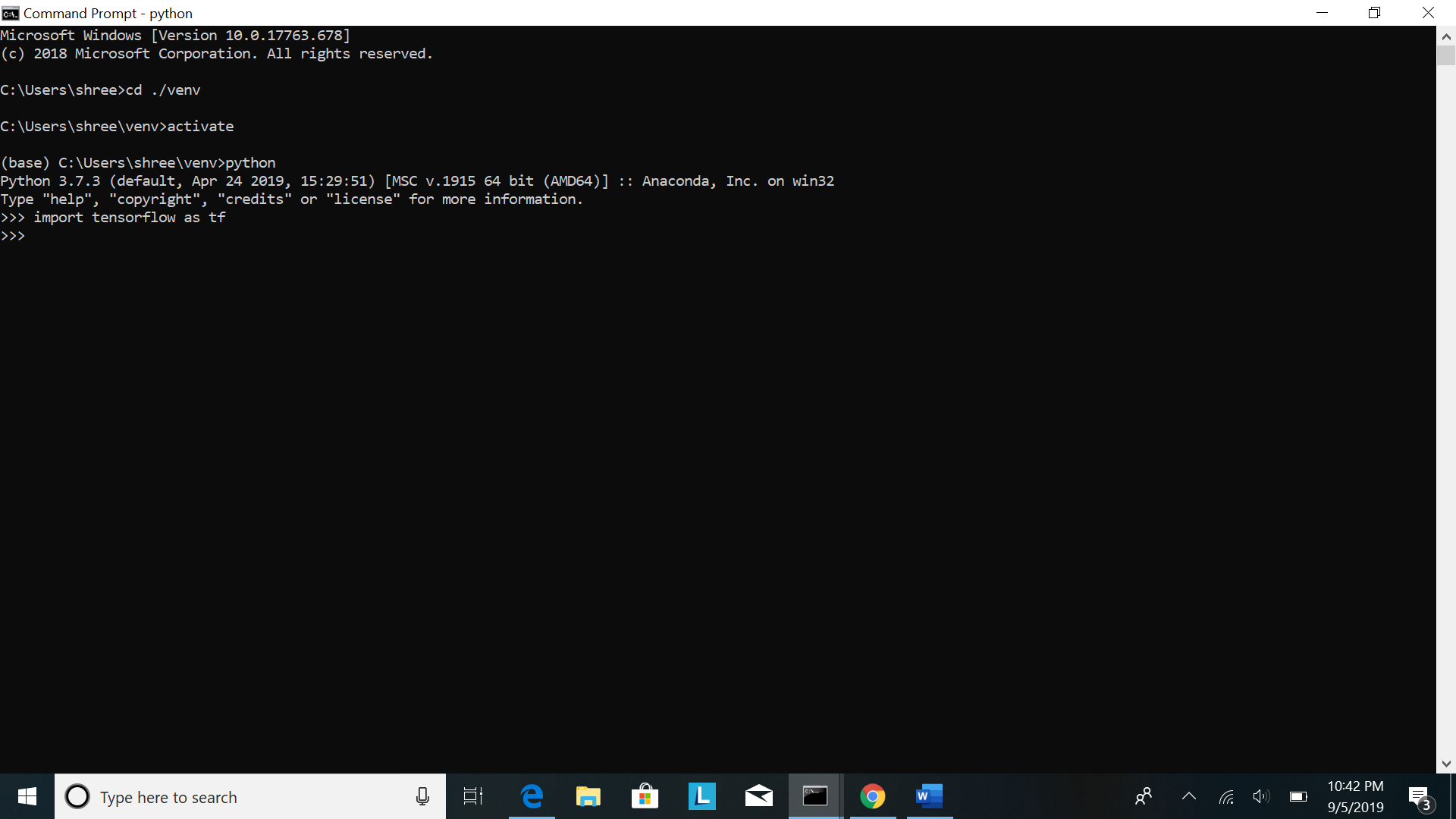
**Introduction:**

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:



**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>

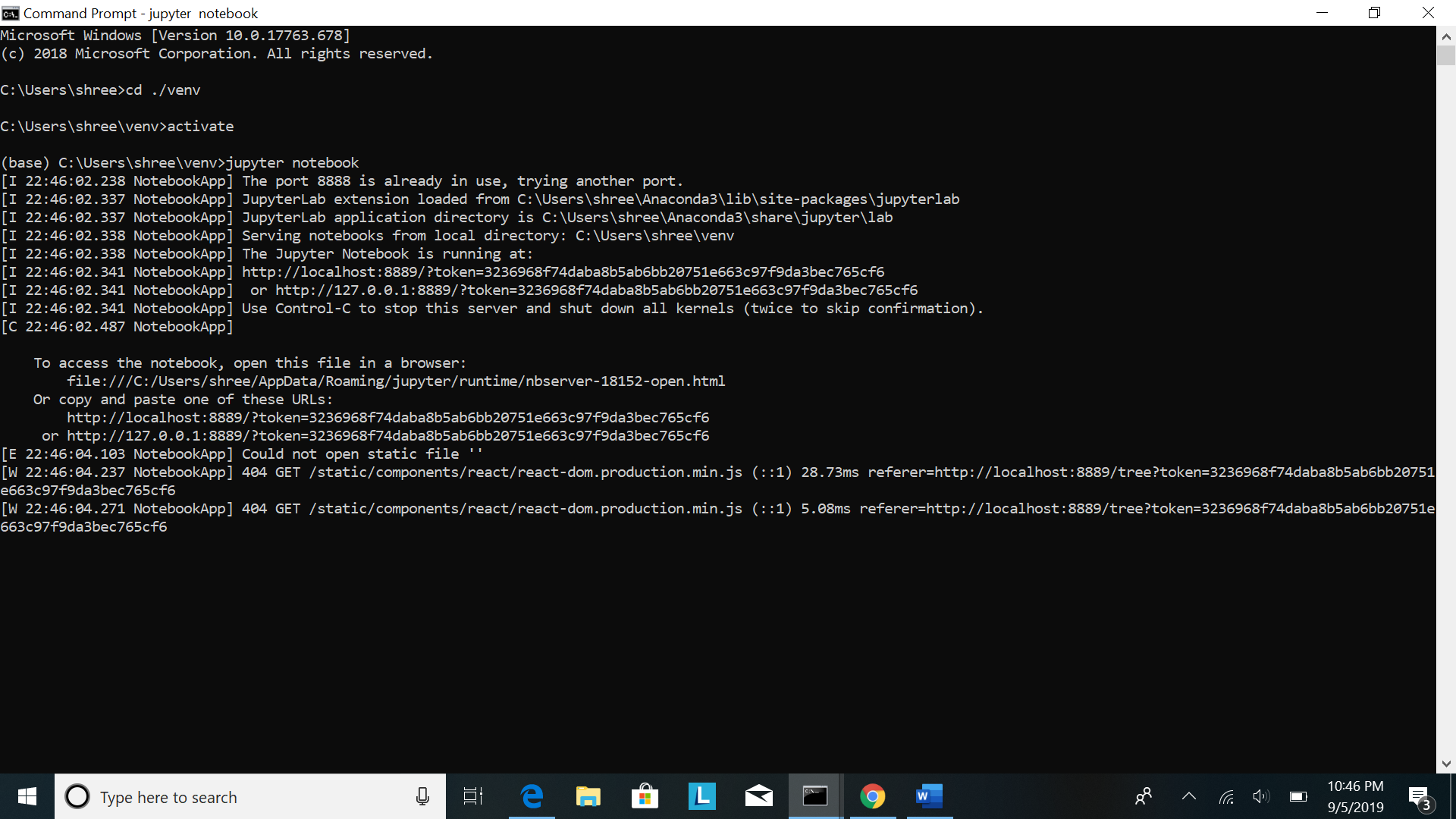
For the dataset I used the datasets available in Tensorflow itself. They are available in the tensorflow.keras.datasets package. The two datasets I used are listed in table1 and are follows:

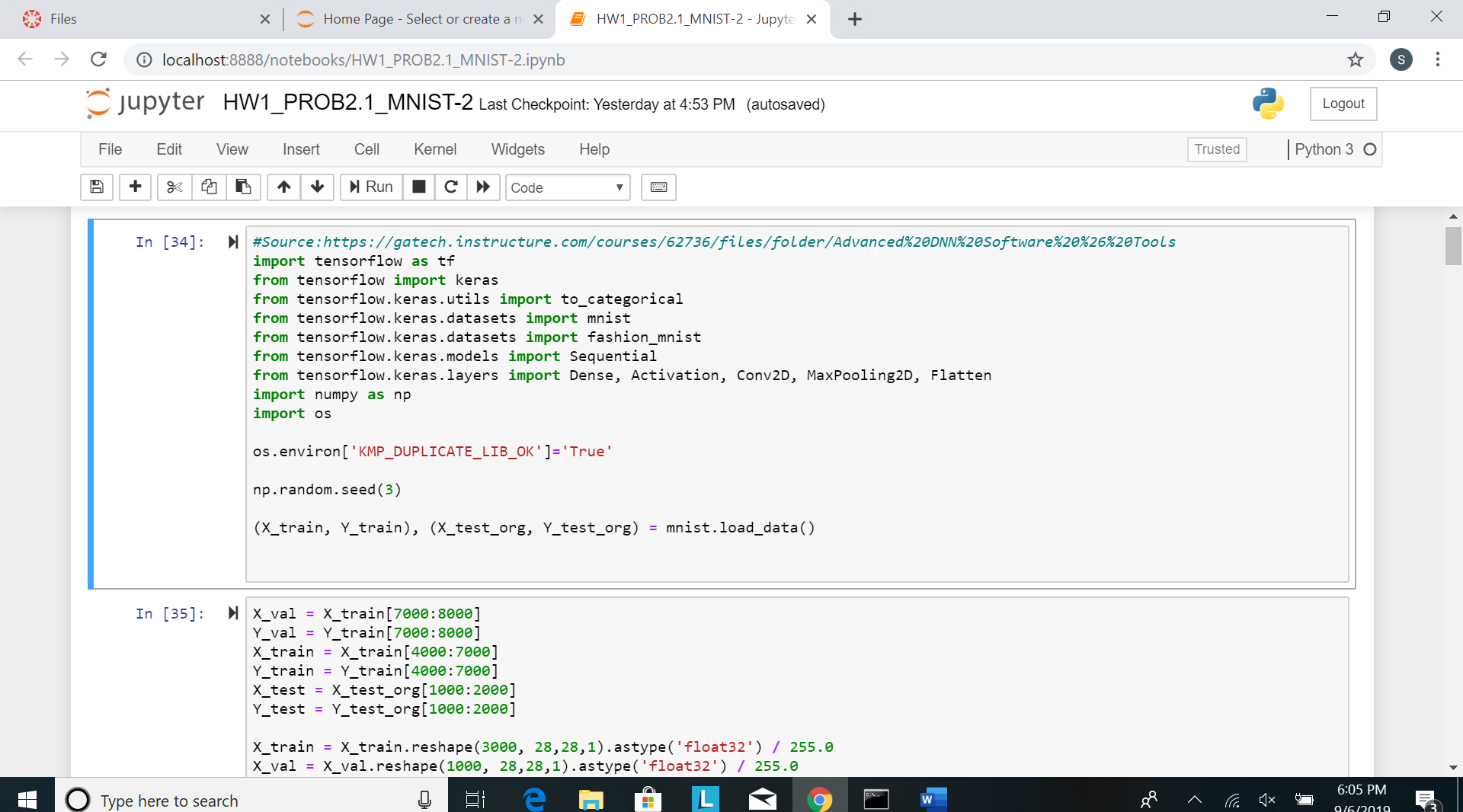
Table 1:

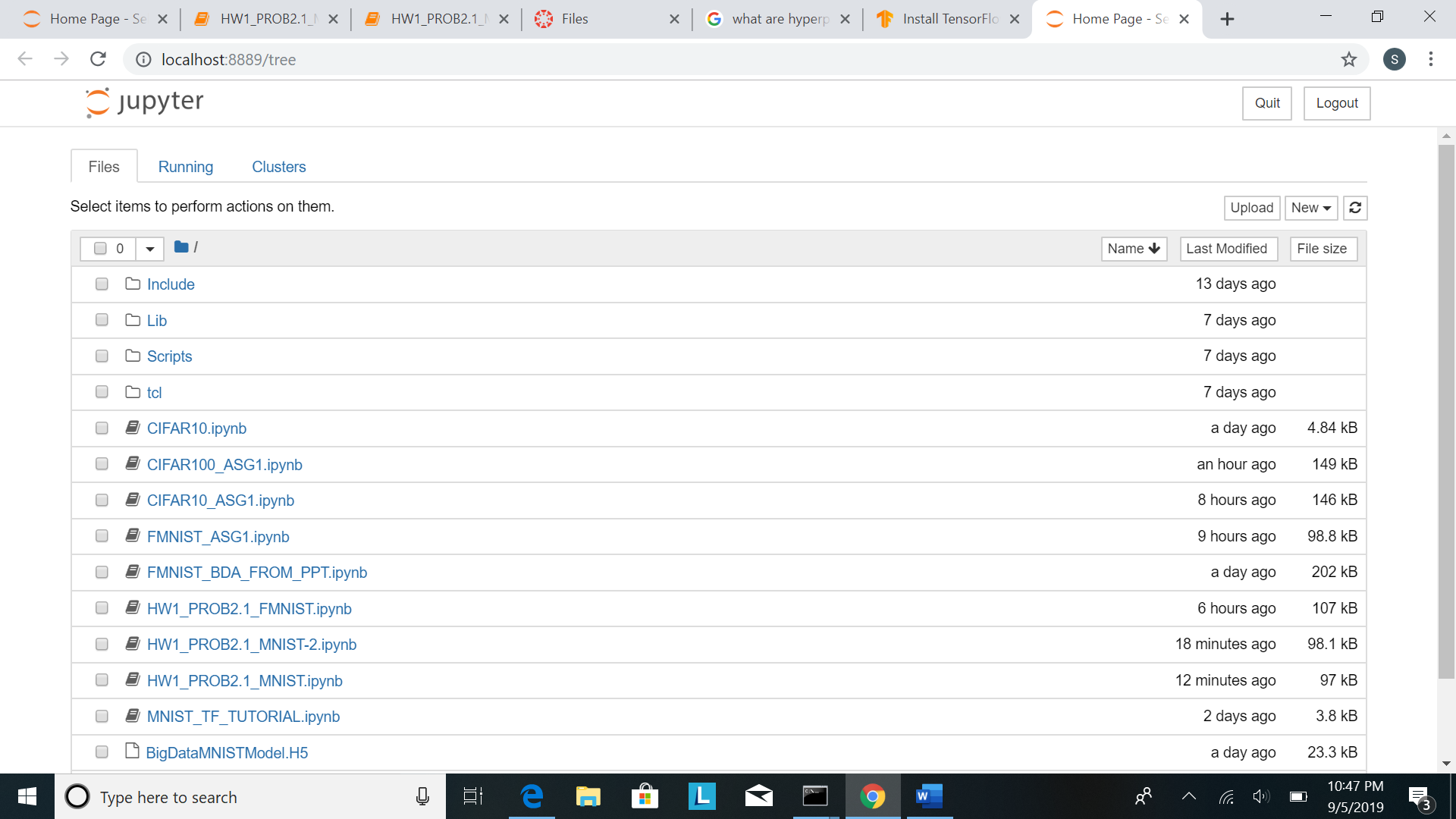
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name | Classes | Resolution of each image | #Training images | #Testing images | Storage size of dataset | Storage size of each image |
| MNIST | 10 | 28\*28 pixel grayscale | 60000 | 10000 | 54.95MB | 1KB |
| FMNIST | 10 | 28\*28 pixel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





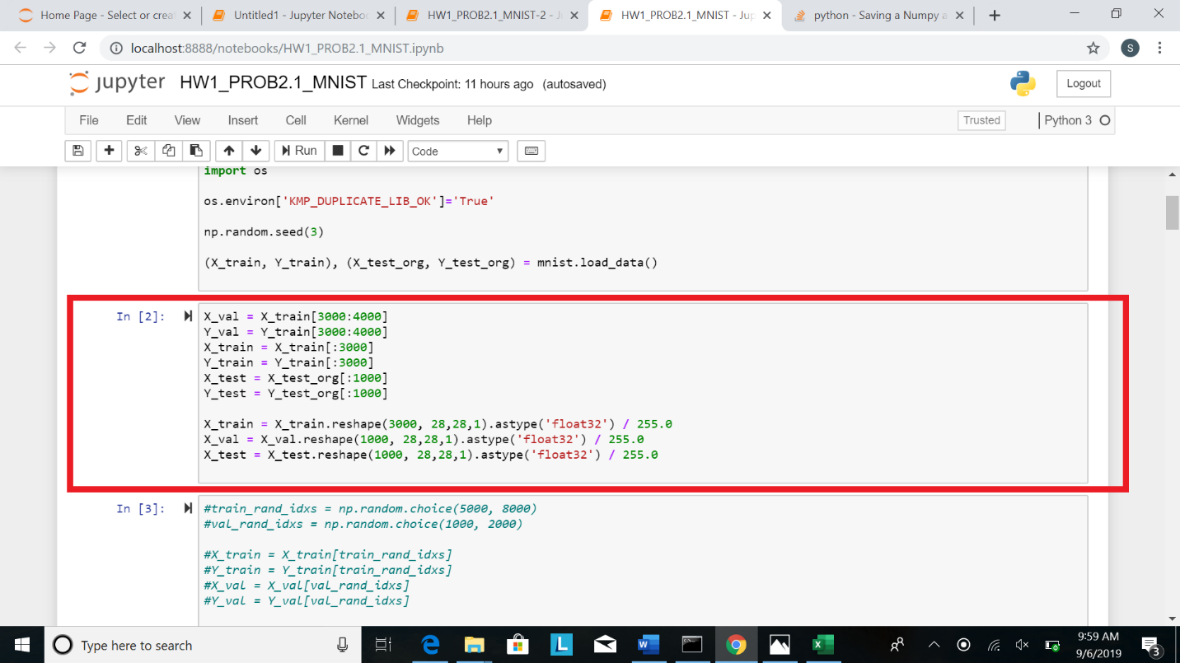


**3. Input Analysis:**

1. **the input dataset**

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 images in 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 images and labels with index 0-4000, the next 4000 images and labels with index 4000-8000 and so on, will give us 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 images and labels. I have also similarly split the FMNIST dataset and used it to create my own dataset of 5000 images. I have also created one dataset of greater size where I have taken 100000 images from the MNIST dataset. The results are summarized in the table below.

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



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 15000-25000) |  | 10 | 28\*28 pixel grayscale | 8000 | 2000 | 1KB |

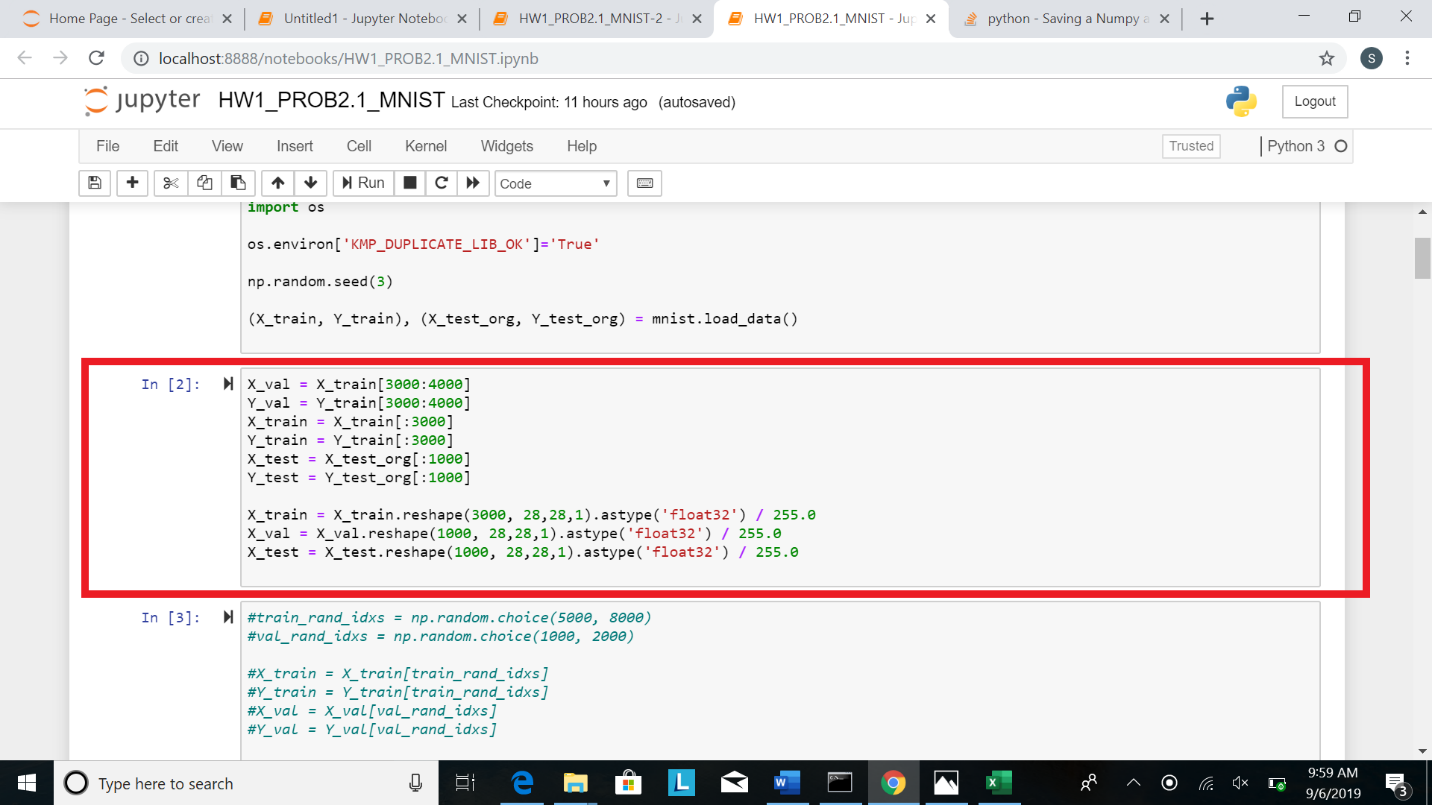
1. **choose and show 2 sample images per class for all K classes in each of the four datasets**

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

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

1. **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\*5000 = 4000 and 0.2\*5000 = 1000. Further, I have split the training data to contain 1000 validation images and 3000 training images.



1. **Default Neural network model and hyperparameters**

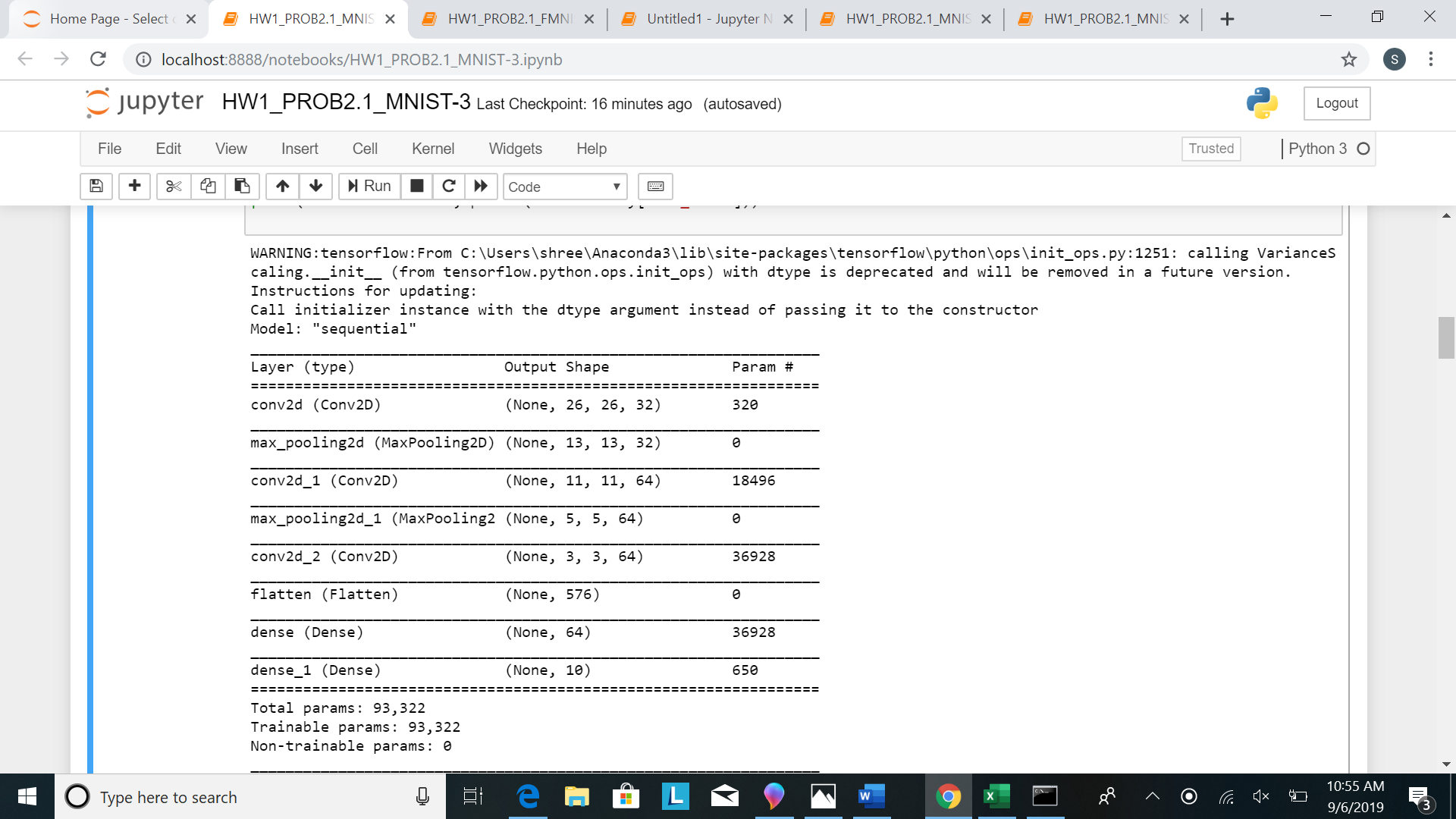
The default structures of the classifier used for the classification task of the 4 datasets is as follows:

***Classifier 1***

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

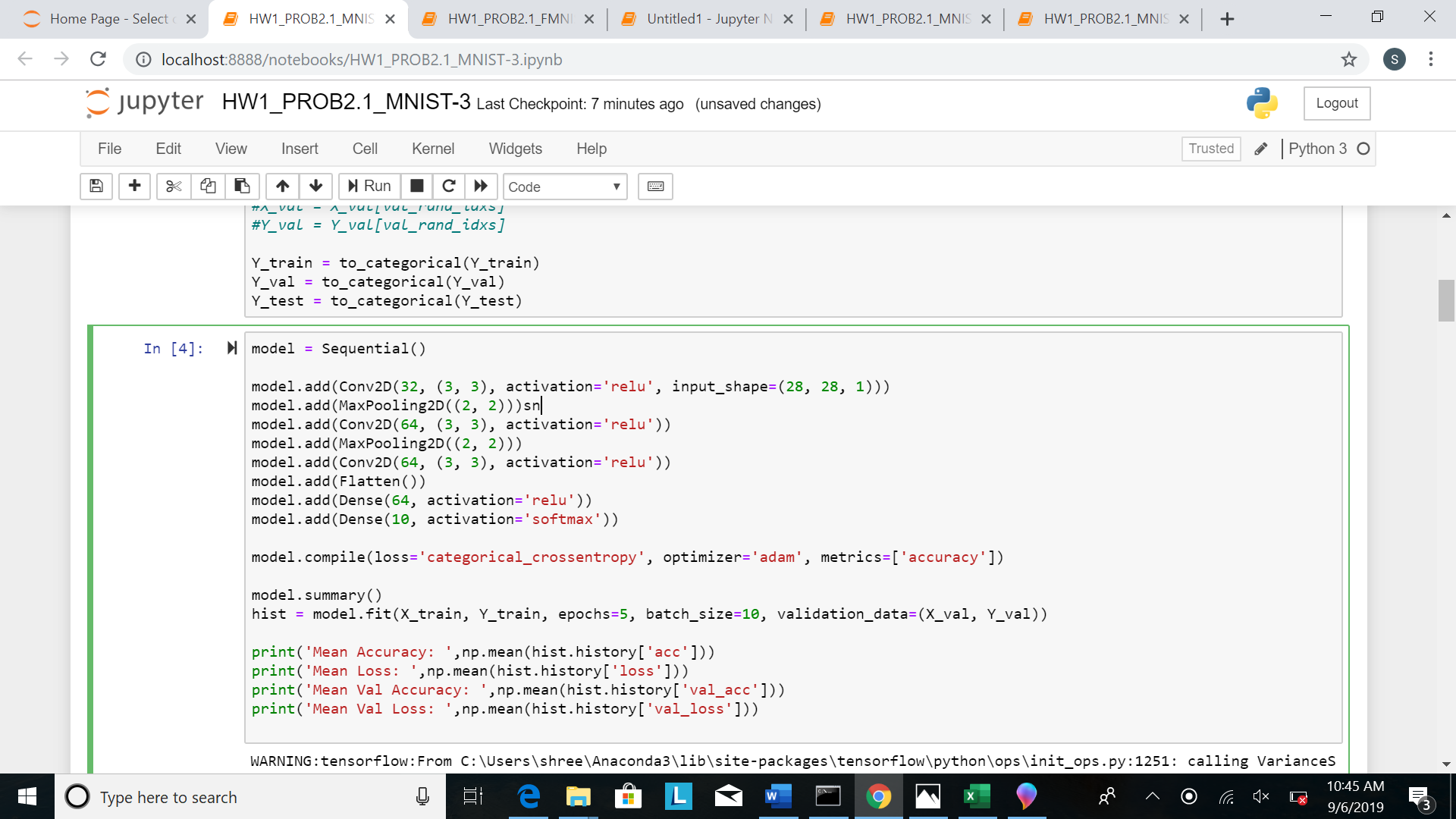
* Convolution layer consisting of 32 3\*3 filters with ‘Relu’ activation function
* 2\*2 Pooling layer
* Convolution layer consisting of 64 3\*3 filters with ‘Relu’ activation function
* 2\*2 Pooling layer
* Convolution layer consisting of 64 3\*3 filters with ‘Relu’ activation function
* Flatten
* Dense 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 loss function used in this classifier is - 'categorical\_crossentropy', the optimizer is - 'adam'.

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

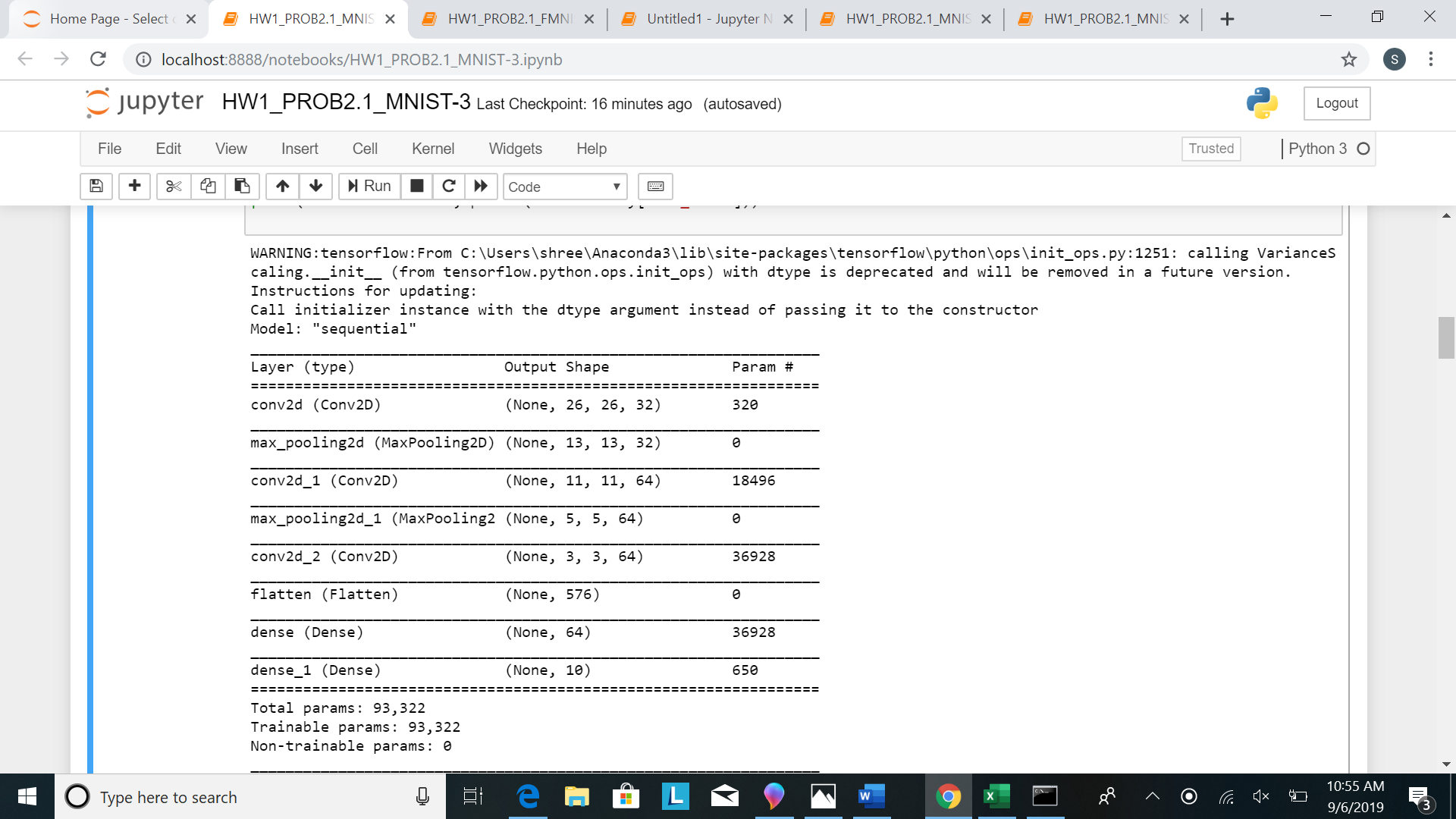


***Classifier 2:***

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

* Convolution layer consisting of 32 3\*3 filters with ‘Relu’ activation function
* 2\*2 Pooling layer
* Convolution layer consisting of 64 3\*3 filters with ‘Relu’ activation function
* 2\*2 Pooling layer
* Convolution 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 |



The loss function used in this classifier is - 'categorical\_crossentropy', the optimizer is - 'adam'.

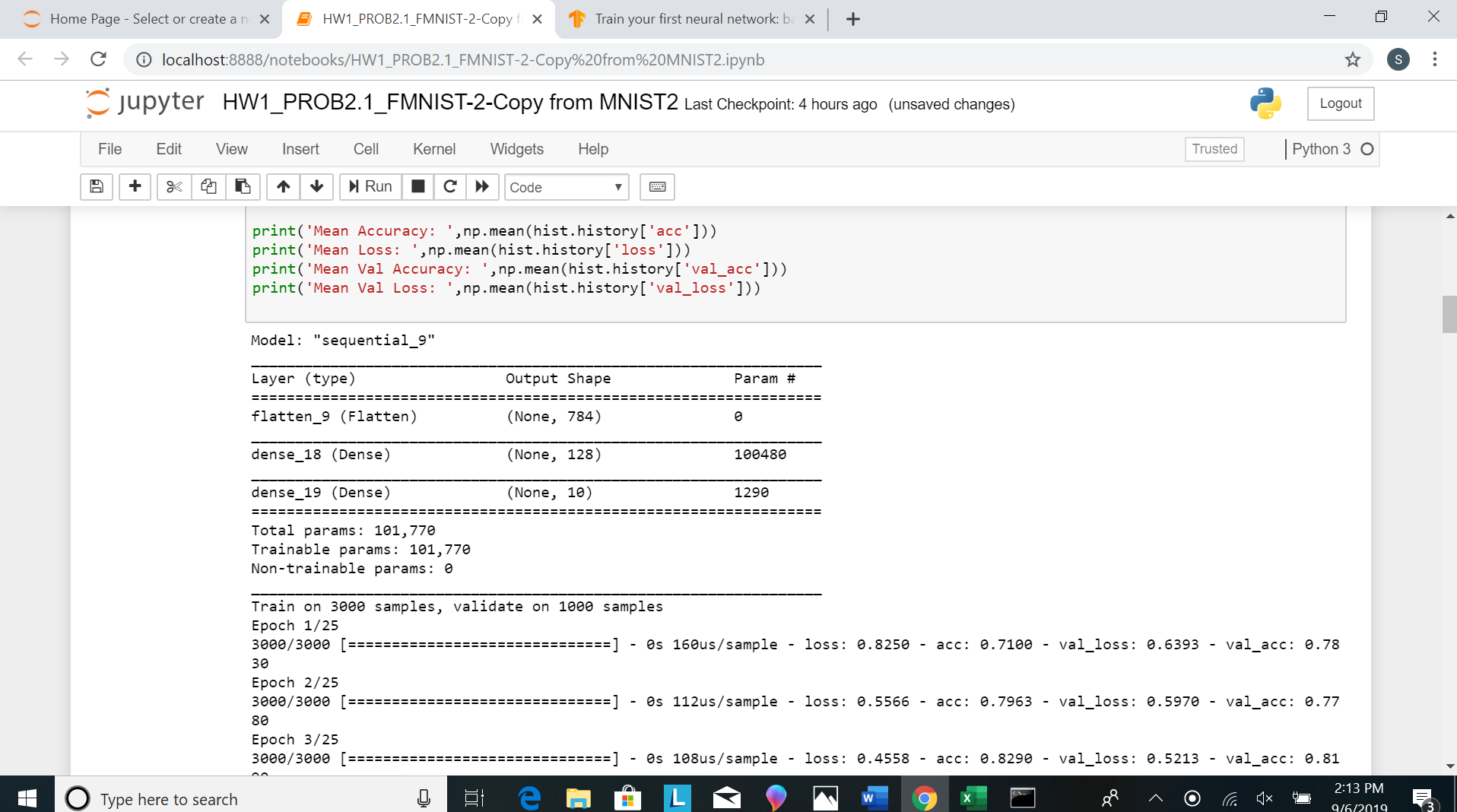
The number of epochs = 25 and the batch size = 10

***Classifier 3:***

This classifier consists of layers as follows:

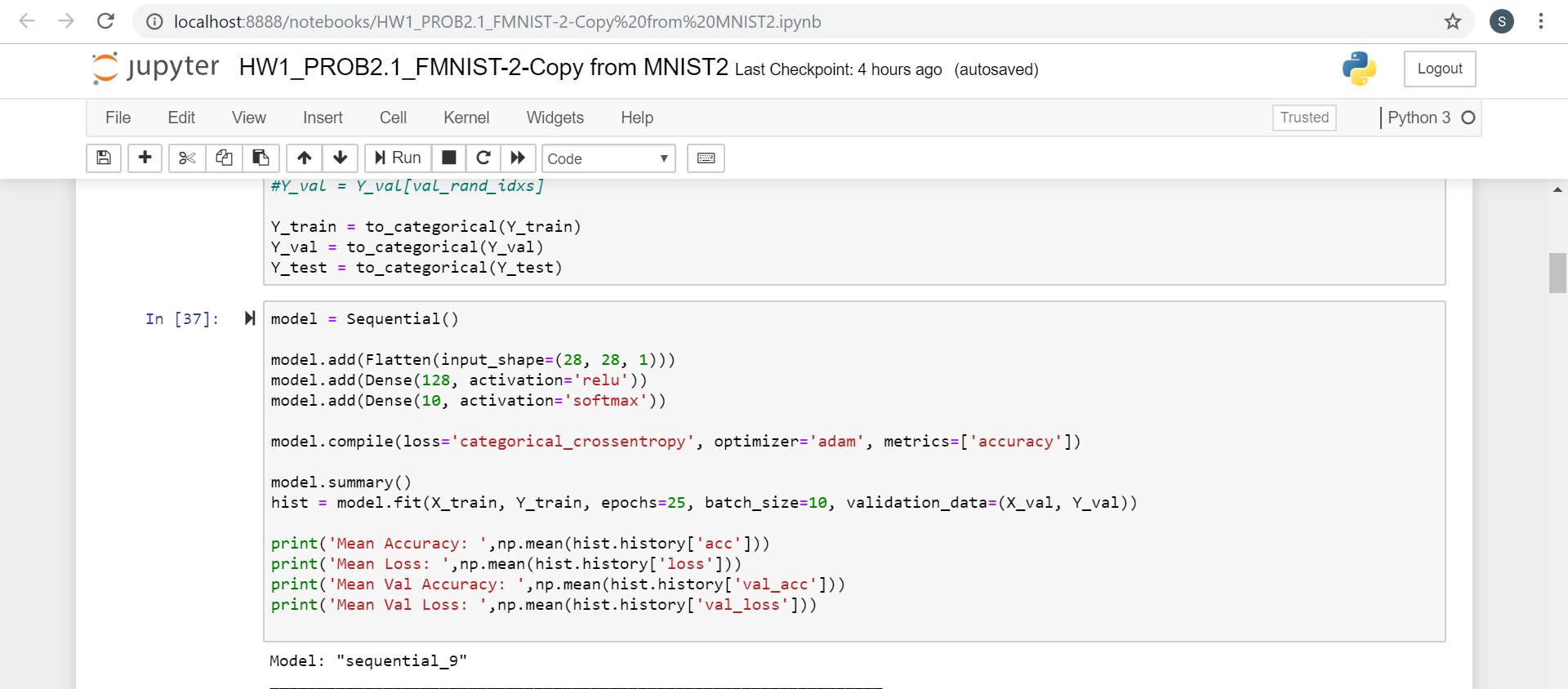
* Flatten
* Dense with 128 neurons and ‘Relu’ activation function
* Dense with 10 neurons and ‘Softmax’ activation function

|  |  |
| --- | --- |
| 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 loss function used in this classifier is - 'categorical\_crossentropy', the optimizer is - 'adam'.

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

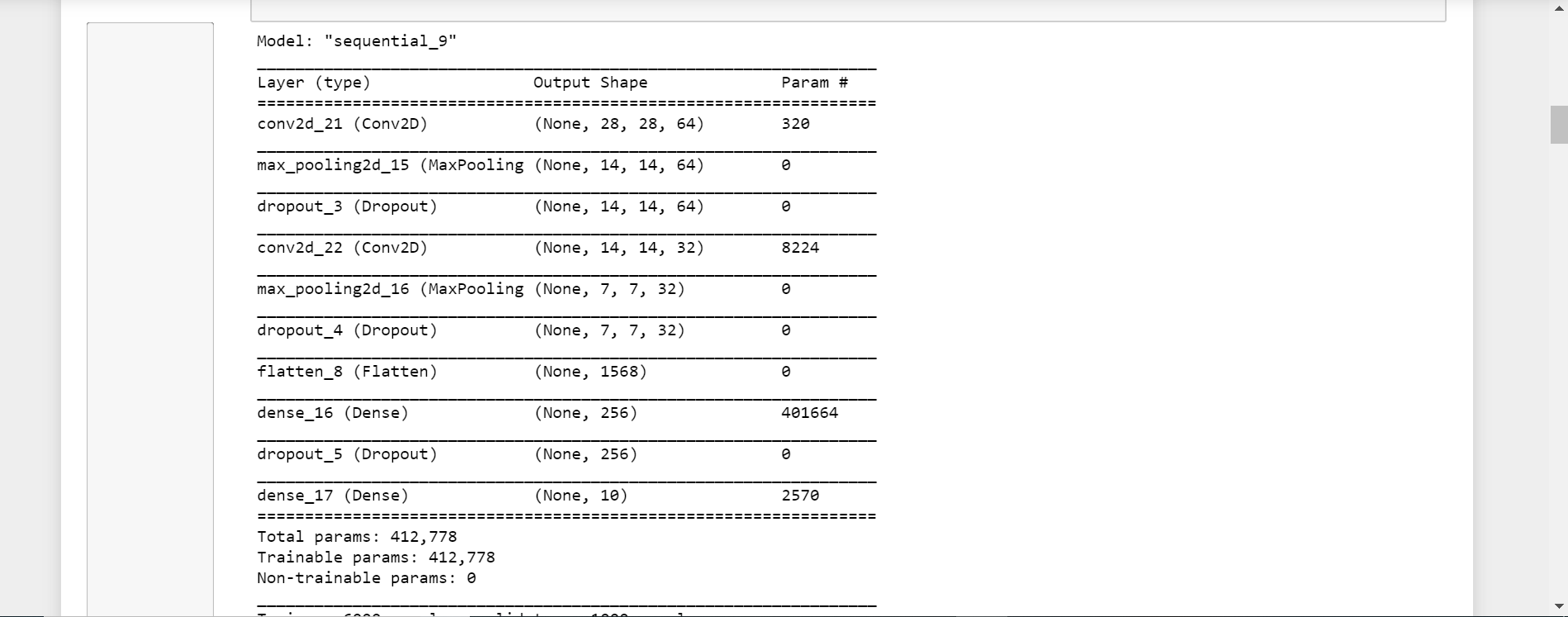


***Classifier 4:***

This classifier consists of layers as follows:

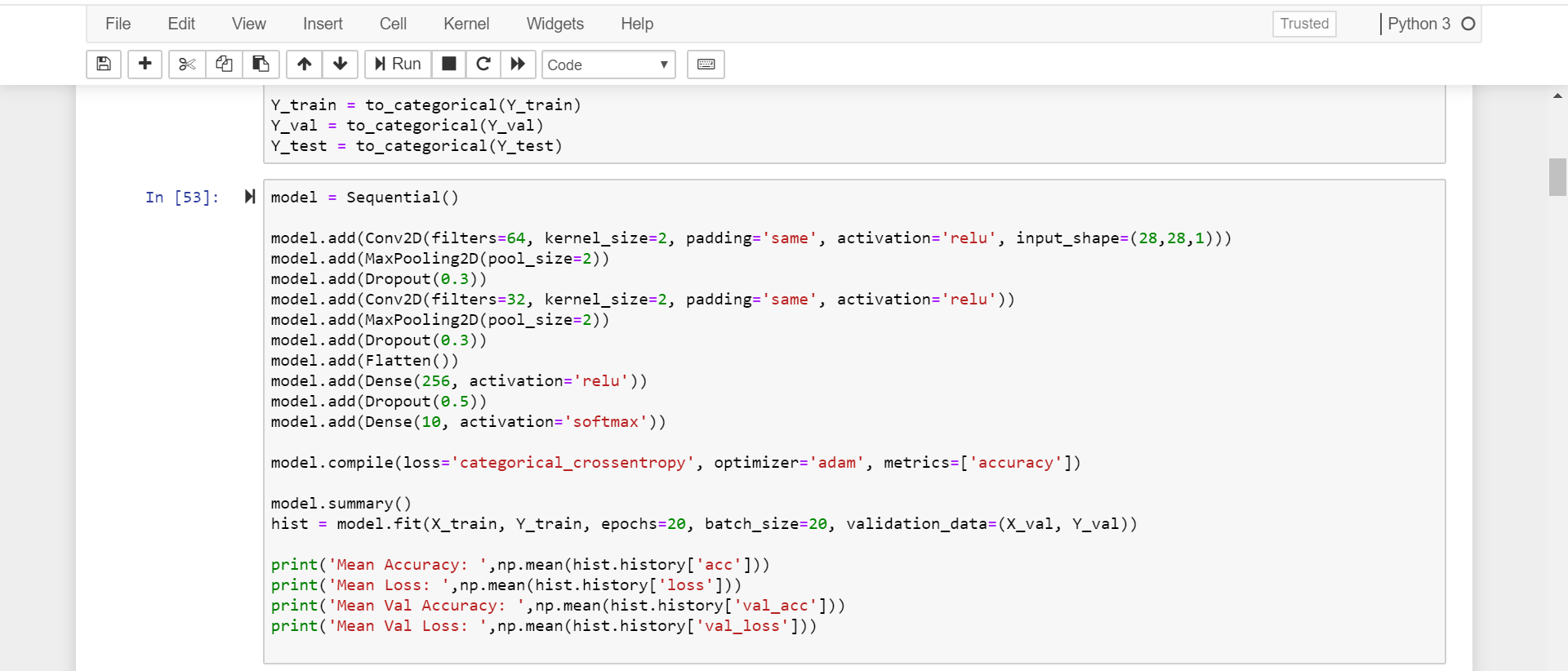
* Convolution layer consisting of 64 2\*2 filters with ‘Relu’ activation function
* 2\*2 Pooling layer
* Dropout layer with rate = 0.3
* Convolution layer consisting of 32 2\*2 filters with ‘Relu’ activation function
* 2\*2 Pooling layer
* Dropout layer with rate = 0.3
* Flatten
* Dense with 256 neurons 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 |



The loss function used in this classifier is - 'categorical\_crossentropy', the optimizer is - 'adam'.

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



1. **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:**

1. **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 |

1. **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

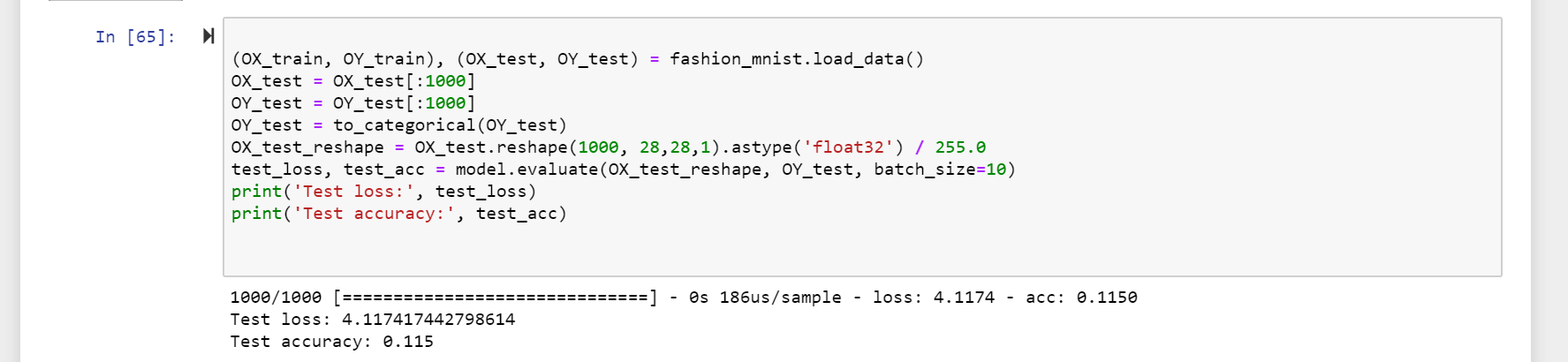
iv.Model3 – 4.881MB

1. **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 |

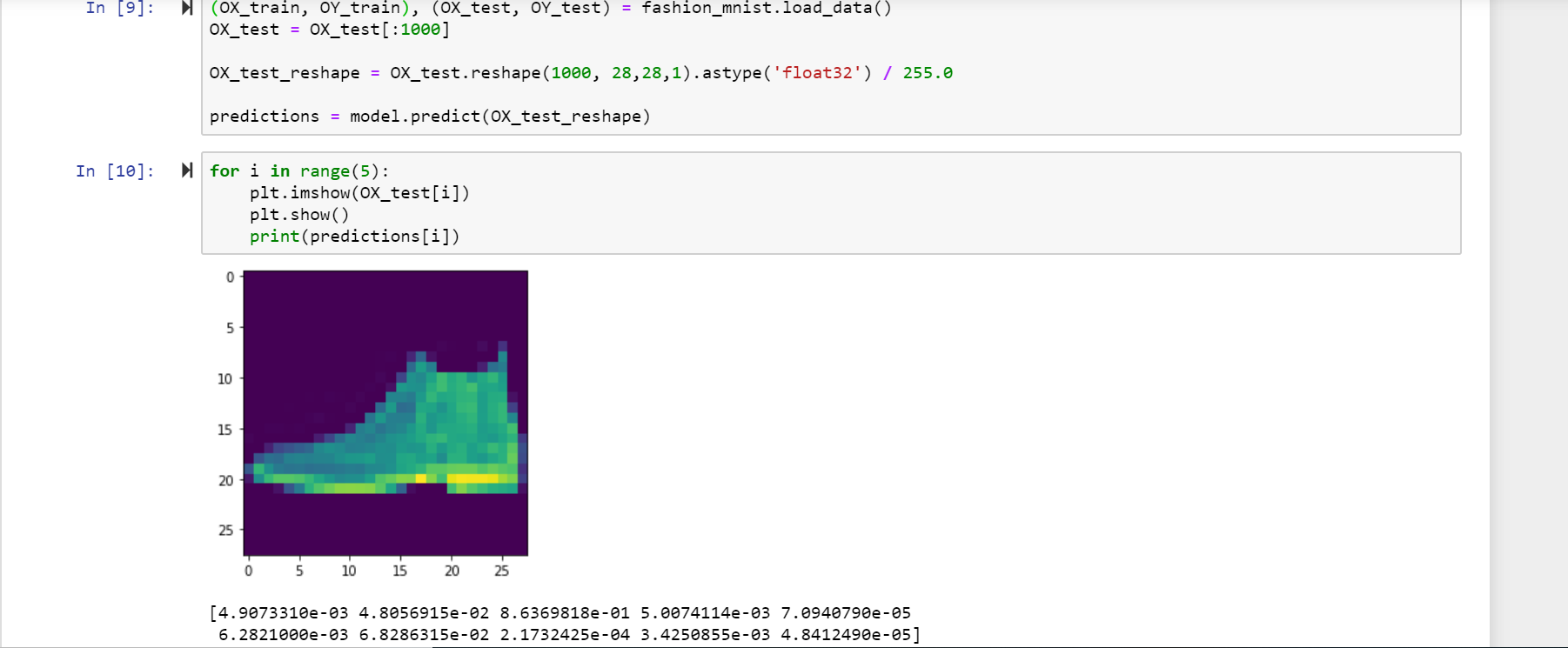
To perform an outlier test, data from the FMNIST dataset was provided as an input to the classifier 1 trained and optimized for the MNIST dataset.

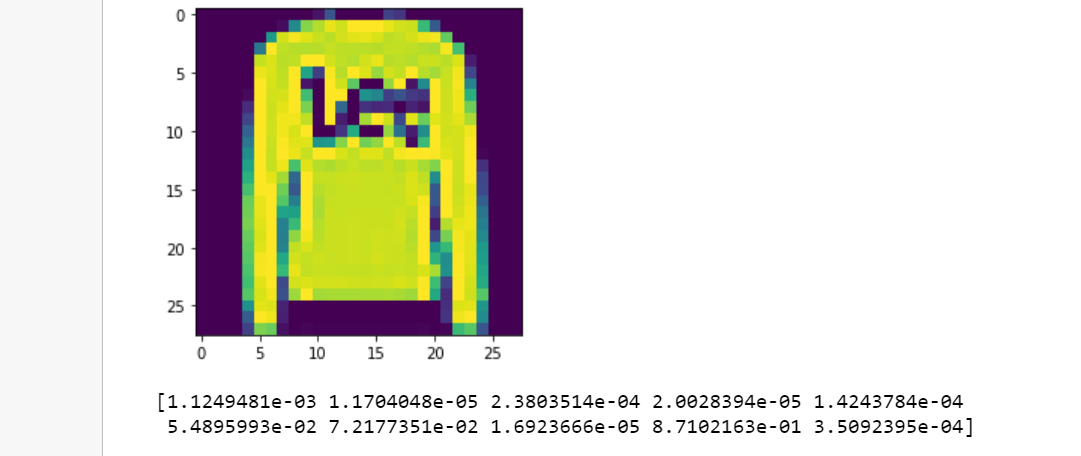
On using the FMNIST dataset as the test dataset for a model trained with the MNIST dataset we got a very low accuracy of 0.115.

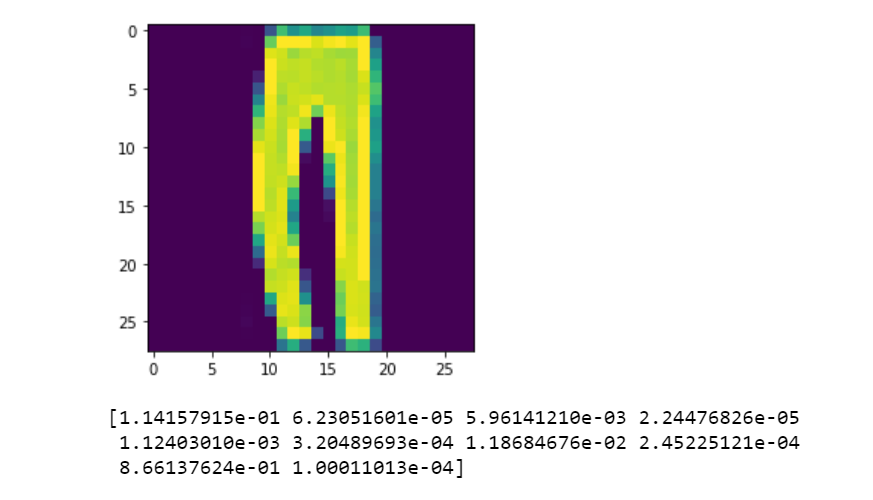


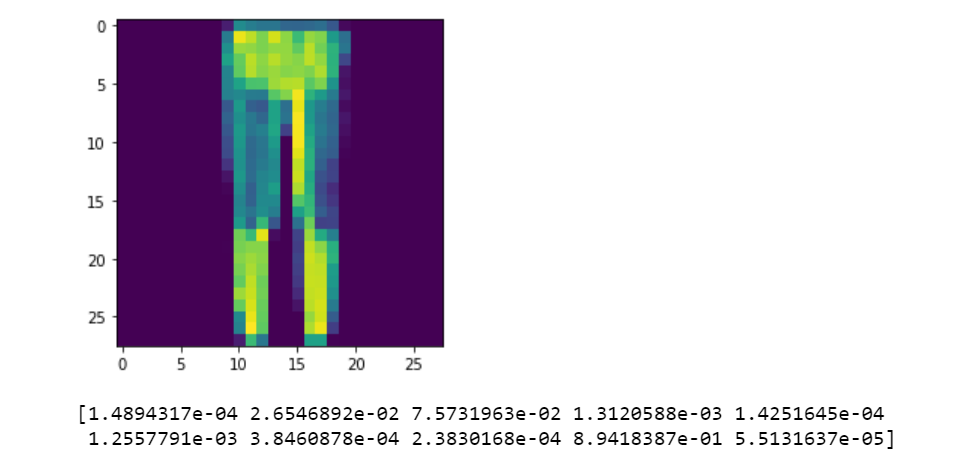
Please find below 5 examples of Outlier testing:

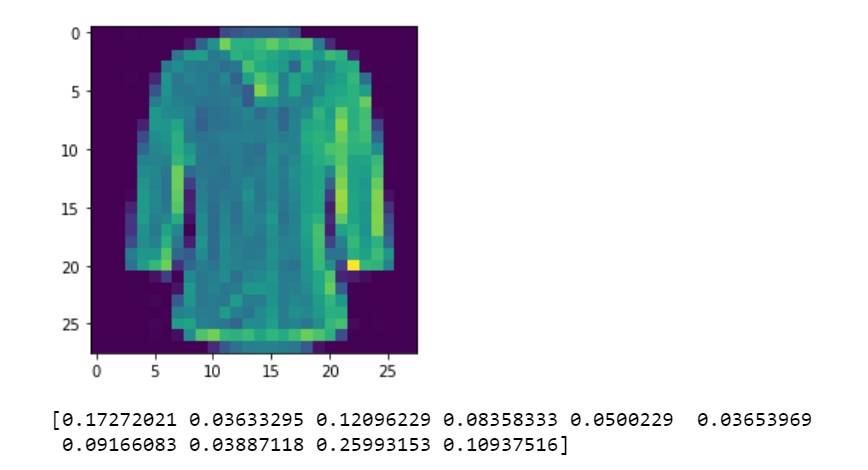
These are images from the FMNIST dataset supplied to our model 1 trained with MNIST dataset.



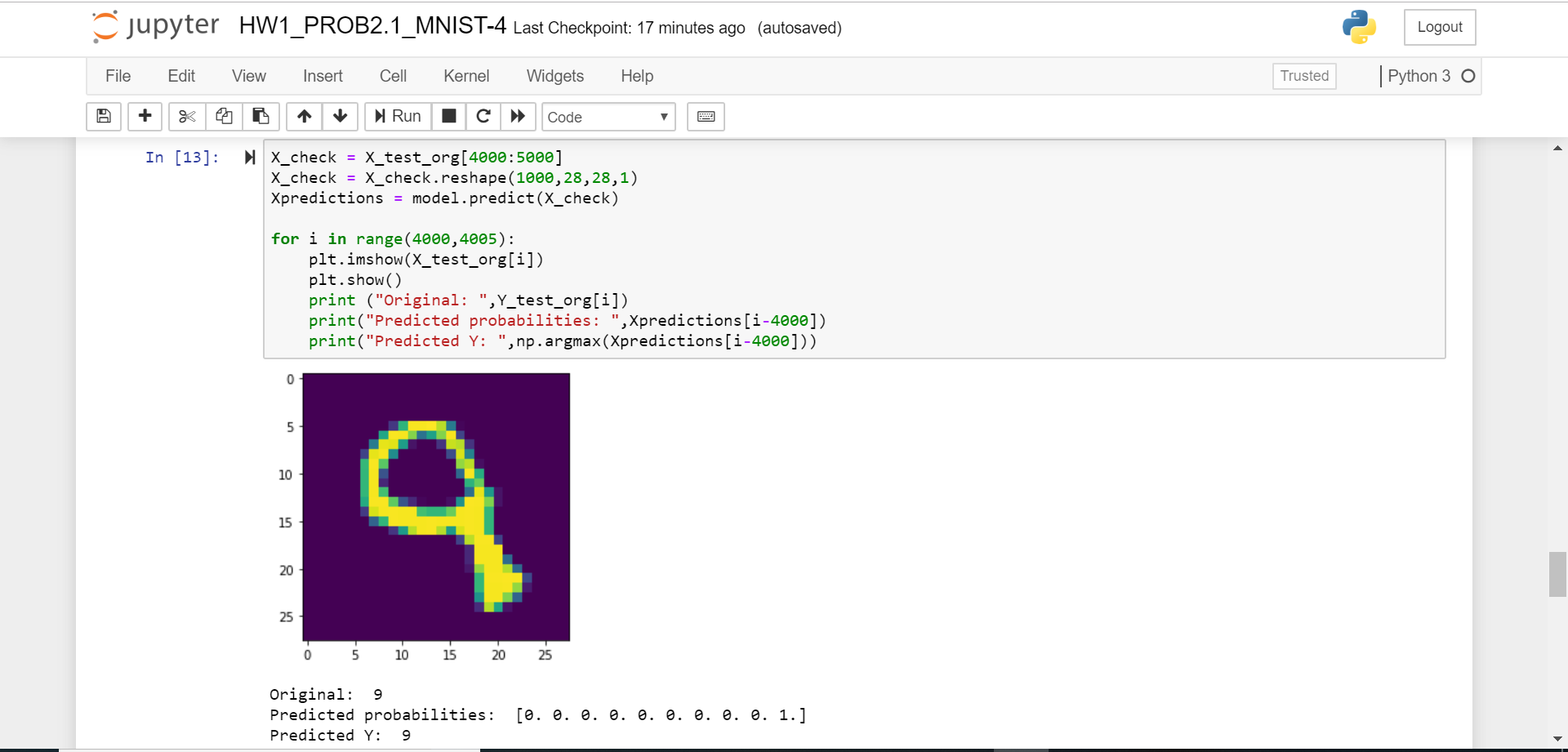






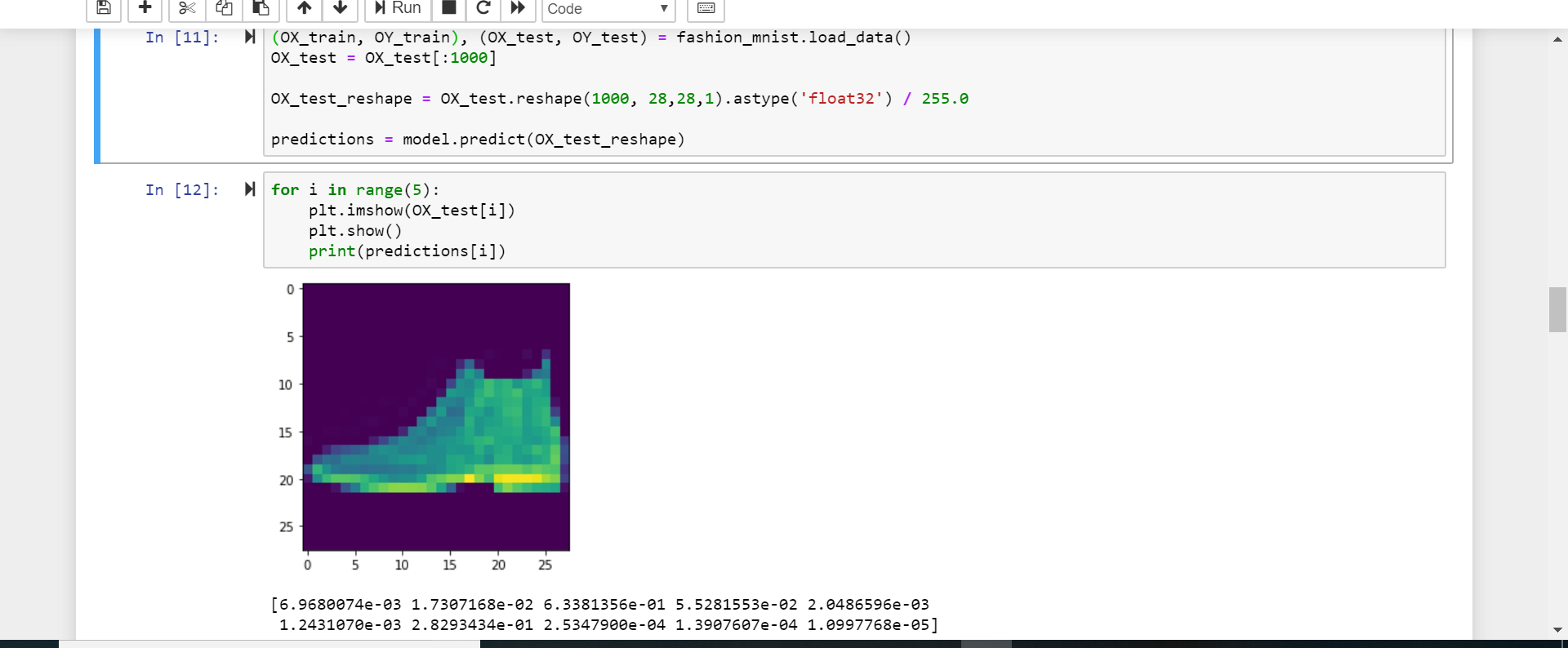


When an image from the MNIST dataset itself, outside of the training data was provided, and the probabilities computed, and output predicted by the model was observed, we saw that one of the classes has a distinctly high probability as compared to the others, see image below:



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

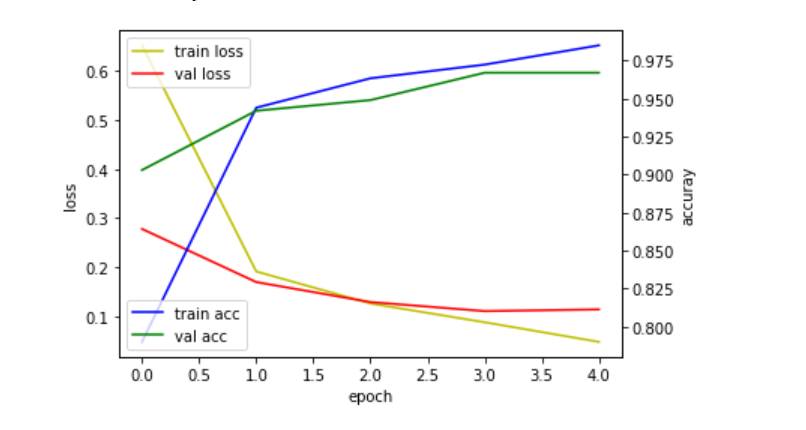
However, the same cannot be said when an image from the FMNIST dataset is provided to the model. All the 10 probability values were similar and had very low values.



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 a random image outside the dataset is provided as input.

1. **You are asked to make at least three observations from your experimental comparison of the 4 CNN classifiers**
2. Let’s look at how the classifier 1 performs on the 4 datasets.

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



Statistics:

Mean Accuracy: 0.93093336

Mean Loss: 0.2212755664913081

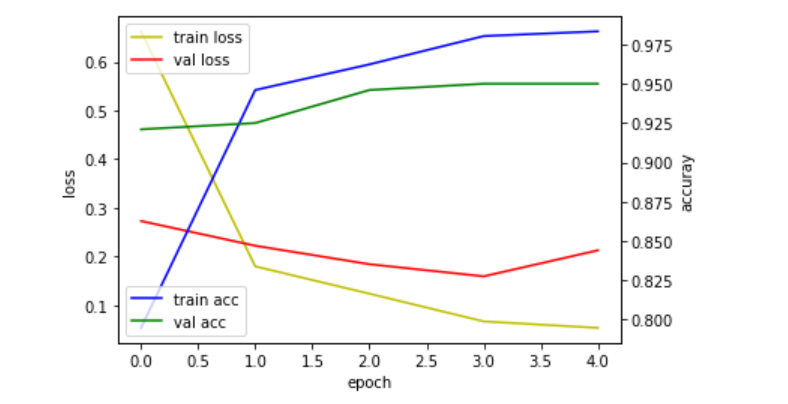
Mean Validation Accuracy: 0.94560003

Mean Validation 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 Validation Accuracy: 0.9384

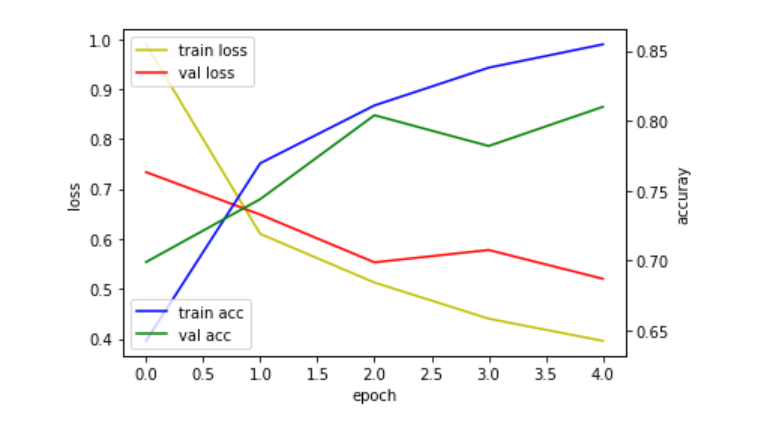
Mean Validation Loss: 0.21013142218880237

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

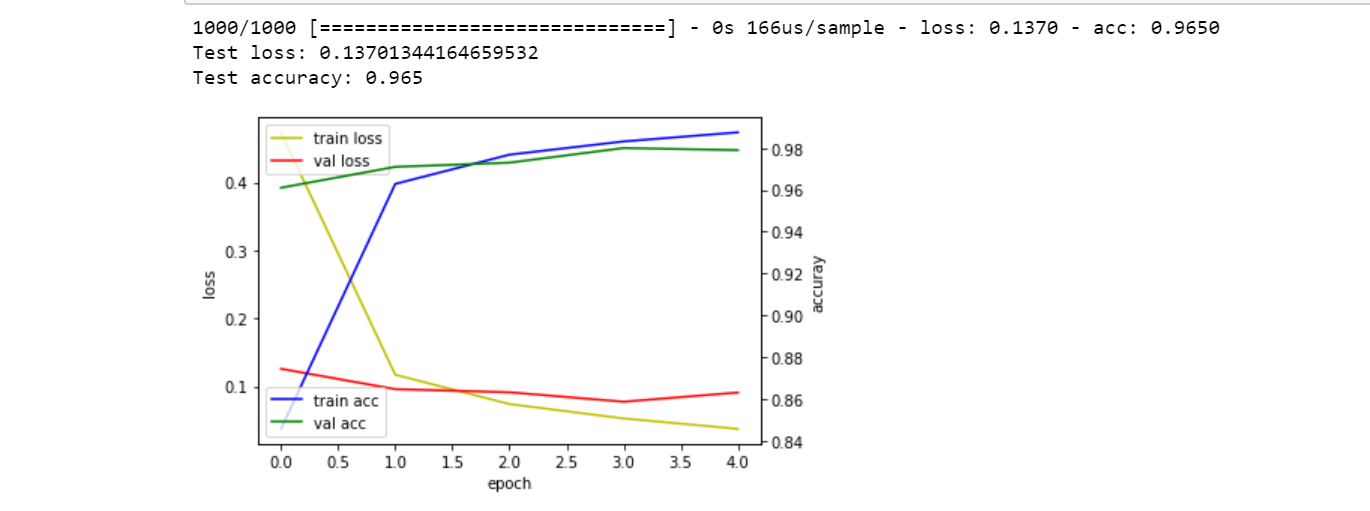
Mean Val Loss: 0.6067126127295196

Test loss: 0.4799393391003832

Test accuracy: 0.834

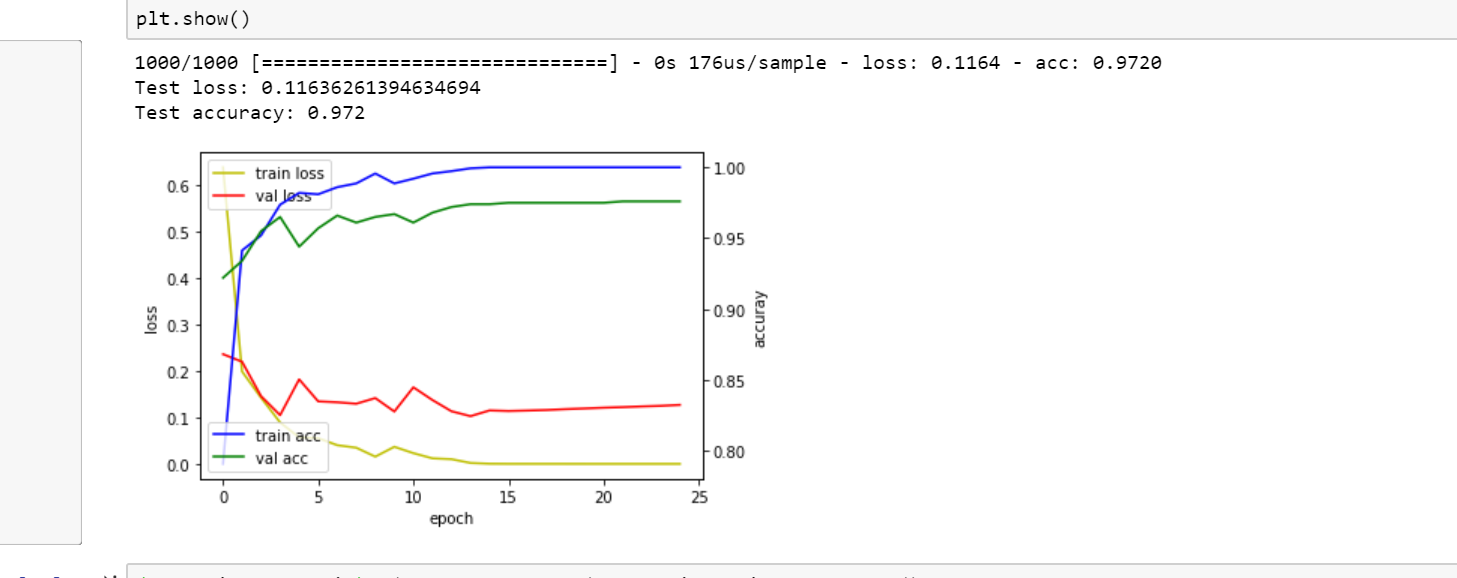
When the main dataset was changed from MNIST to FMNIST, we observed a drop in the testing accuracy of approximately 10%. This is probably because the Neural Network Classifier we have selected, is similar to the one used in the Tensorflow tutorial for the MNIST database. Thus, this model probably has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

1. 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.



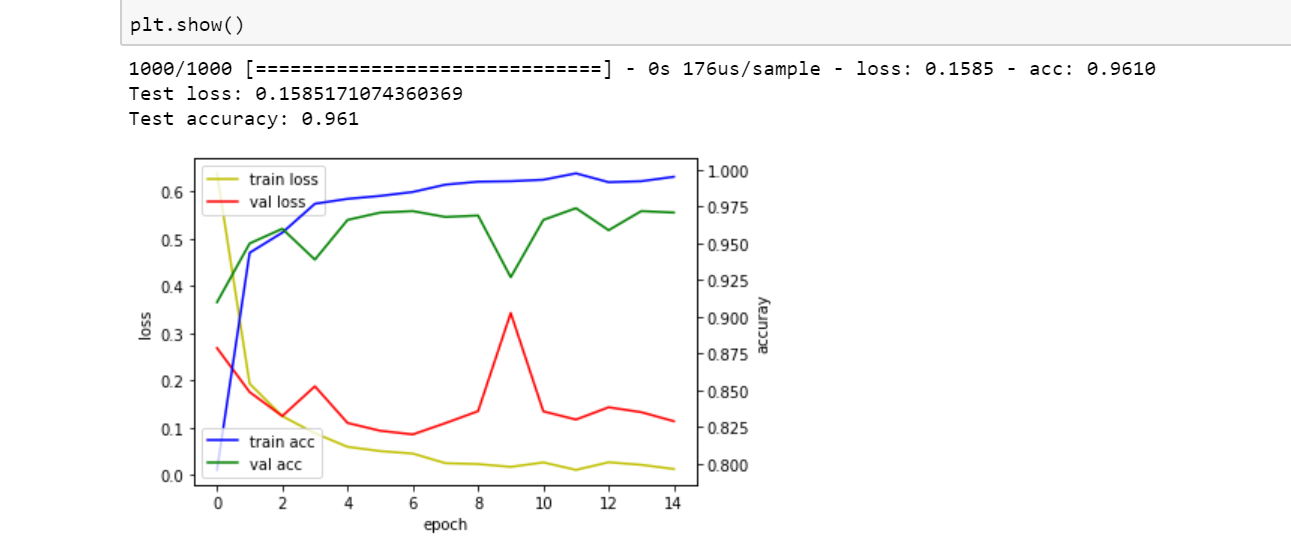
|  |  |
| --- | --- |
| Dataset | Model 1 |
| MNIST[0-5000] | Training Time - 6s, Training Accuracy - 0.93, Testing Time - 1s, Testing Accuracy - 0.95 |
| MNIST[5000-10000] | Training Time - 5.8 s, Training Accuracy - 0.936, Testing Time-1s, Testing Accuracy - 0.924 |
| MNIST[10000-15000] | Training 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 |

1. 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 as compared to later. 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.

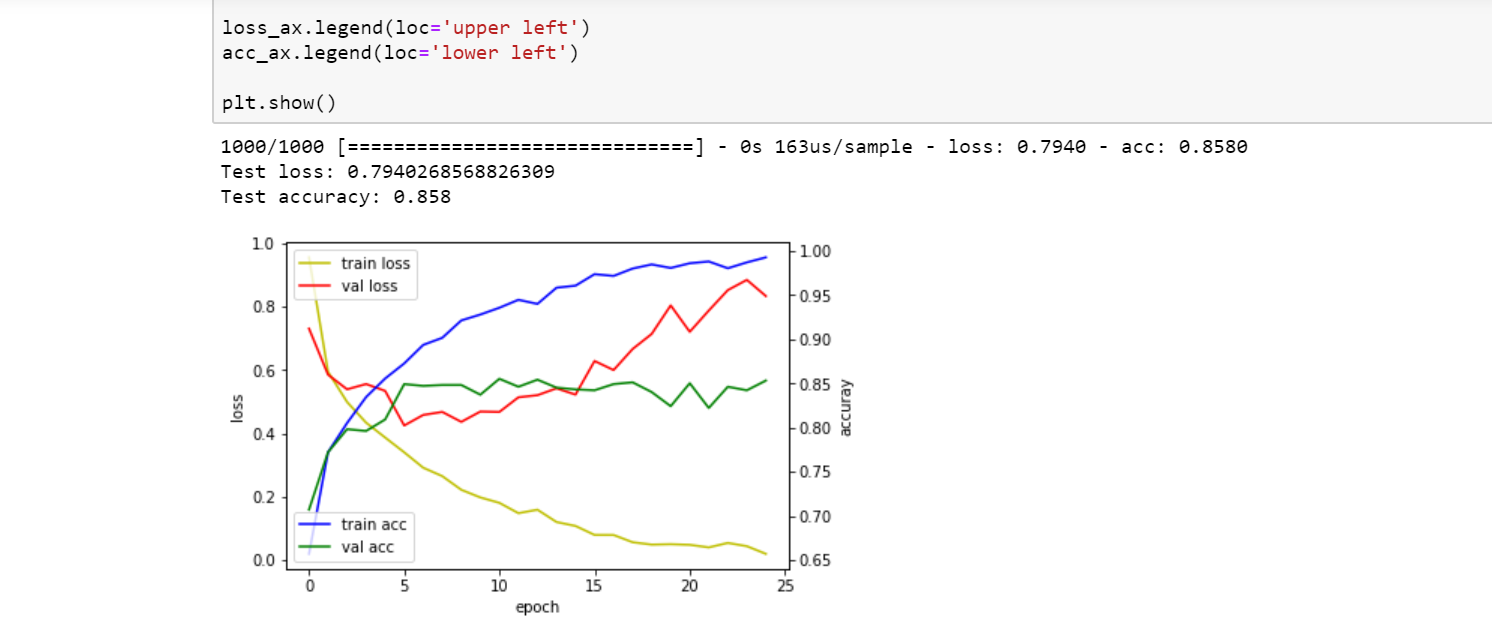


This has been summarized in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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.

1. 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.



1. 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 the 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 |
|  |  |  |
|  |  |  |

1. 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.
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.