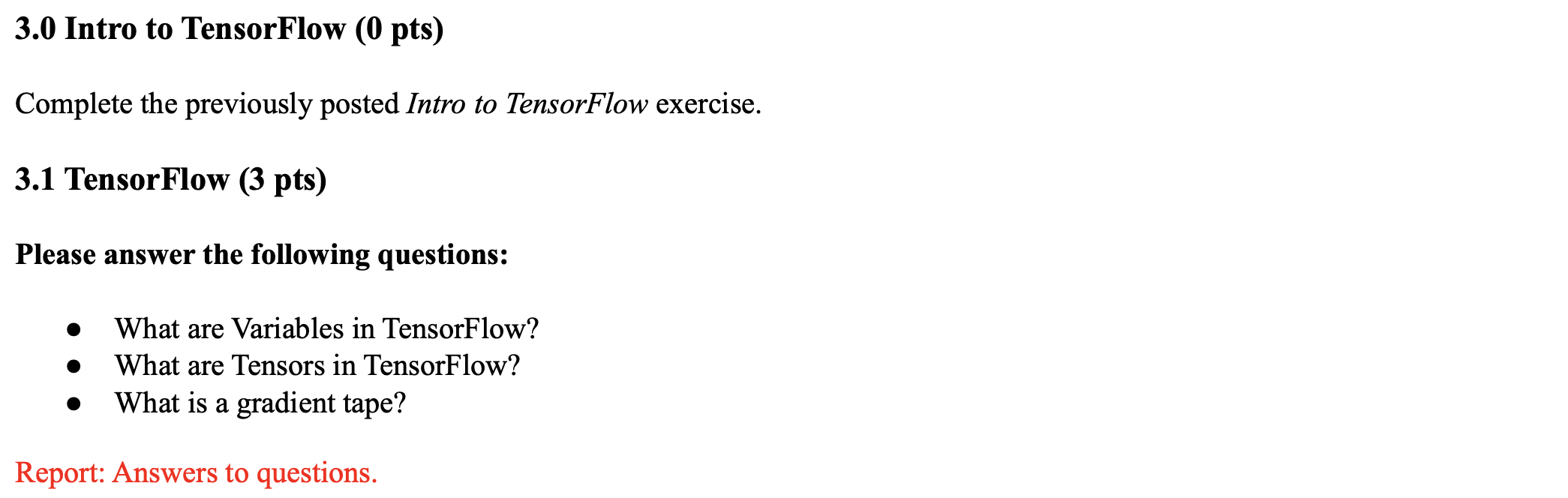
CMSC 636 Neural Nets and Deep Learning  
Spring 2022, Instructor: Dr. Milos Manic, *http://www.people.vcu.edu/~mmanic* Homework 3

*Student certification:*

*Team member 1:  
Print Name: Samah Ahmed Date: 14 April 2022   
I have contributed by doing the following: 3.1, 3.2*  
*Signed: Samah*

*Team member 2:  
Print Name: Md Touhiduzzaman Date: 15 April 2022   
I have contributed by doing the following: 3.3*  
*Signed: Touhid*

*Team member 2:   
Print Name: Maher Al Islam Date: 15 April 2022   
I have contributed by doing the following: 3.4  
Signed: Maher*

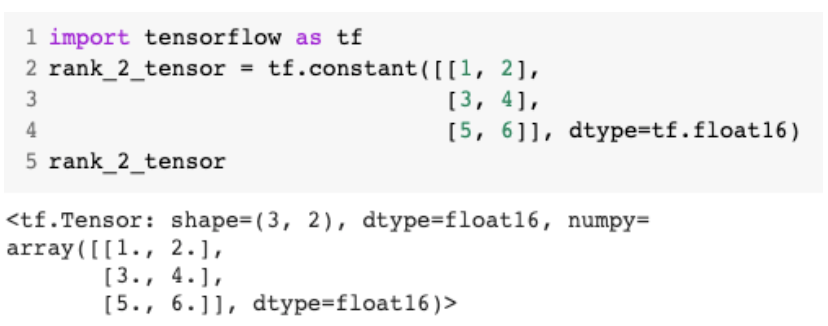


**Answer to the question 3.1:**

**TensorFlow Variable:**A TensorFlow **variable** is the recommended way to represent a shared, persistent state your program manipulates. This guide covers how to create, update, and manage instances of tf.Variable in TensorFlow.Variables are created and tracked via the tf.Variable class. A tf.Variable represents a tensor whose value can be changed by running ops on it. Specific ops allow you to read and modify the values of this tensor. Higher level libraries like tf.keras use tf.Variable to store model parameters.  
TensorFlow variables can be used to represent **model trainable parameters** like weights and biases of a neural network. For example:

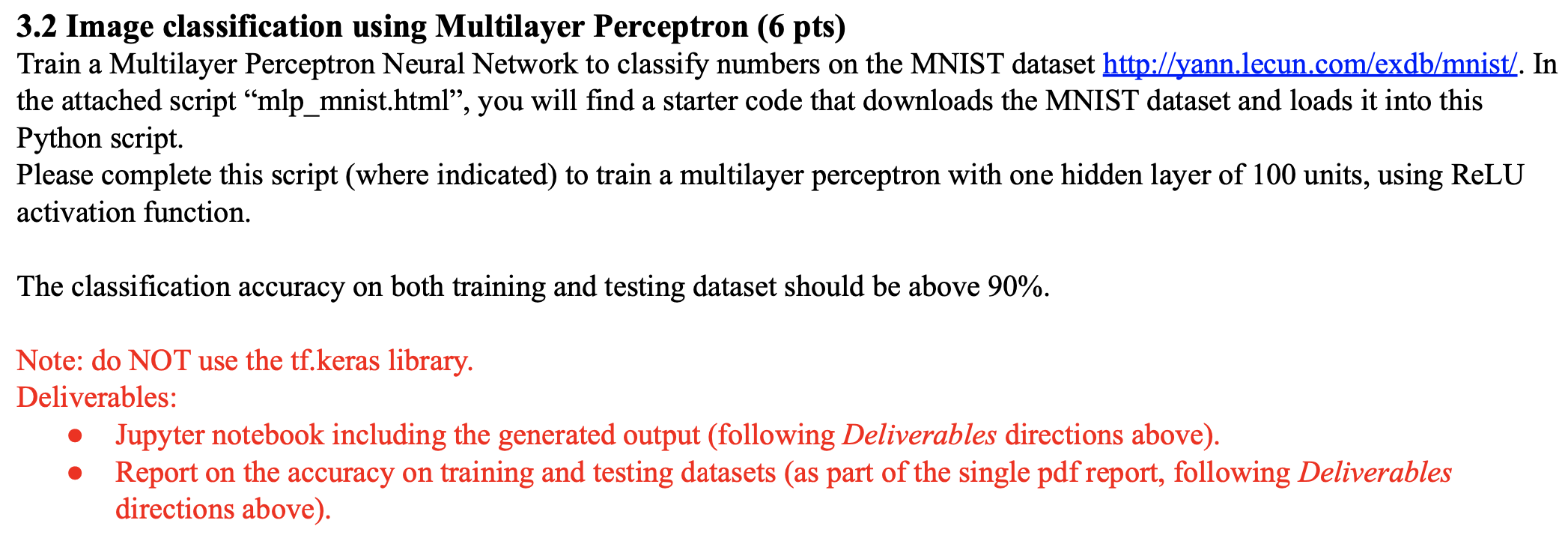


**Tensors in TensorFlow:**Tensors are multi-dimensional arrays with a uniform type (called a dtype). It can be used to represent **the data being fed into the model** and also the intermediate representations of the data as it passes through the model like output of activation functions.   
For example:



**Gradient Tape:**TensorFlow provides the tf.GradientTape API for automatic differentiation; that is, computing the gradient of a computation with respect to some inputs, usually tf.Variables. TensorFlow "records" relevant operations executed inside the context of a tf.GradientTape onto a "tape". TensorFlow then uses that tape to compute the gradients of a "recorded" computation using reverse mode differentiation.   
Example:

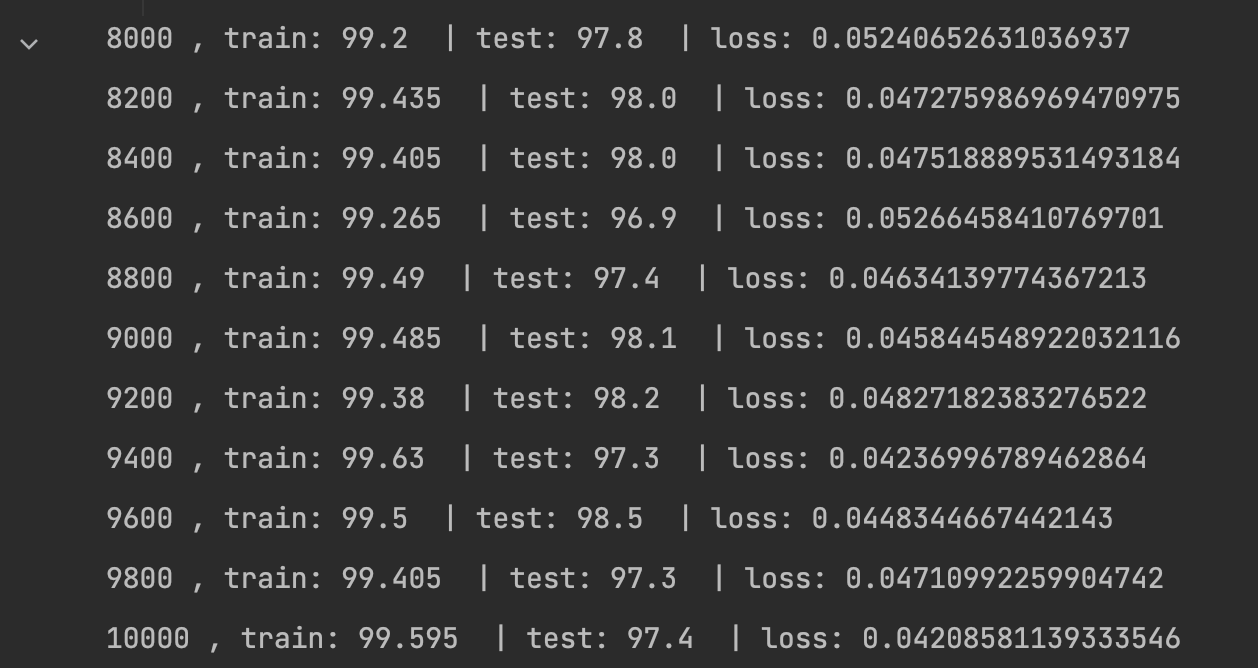


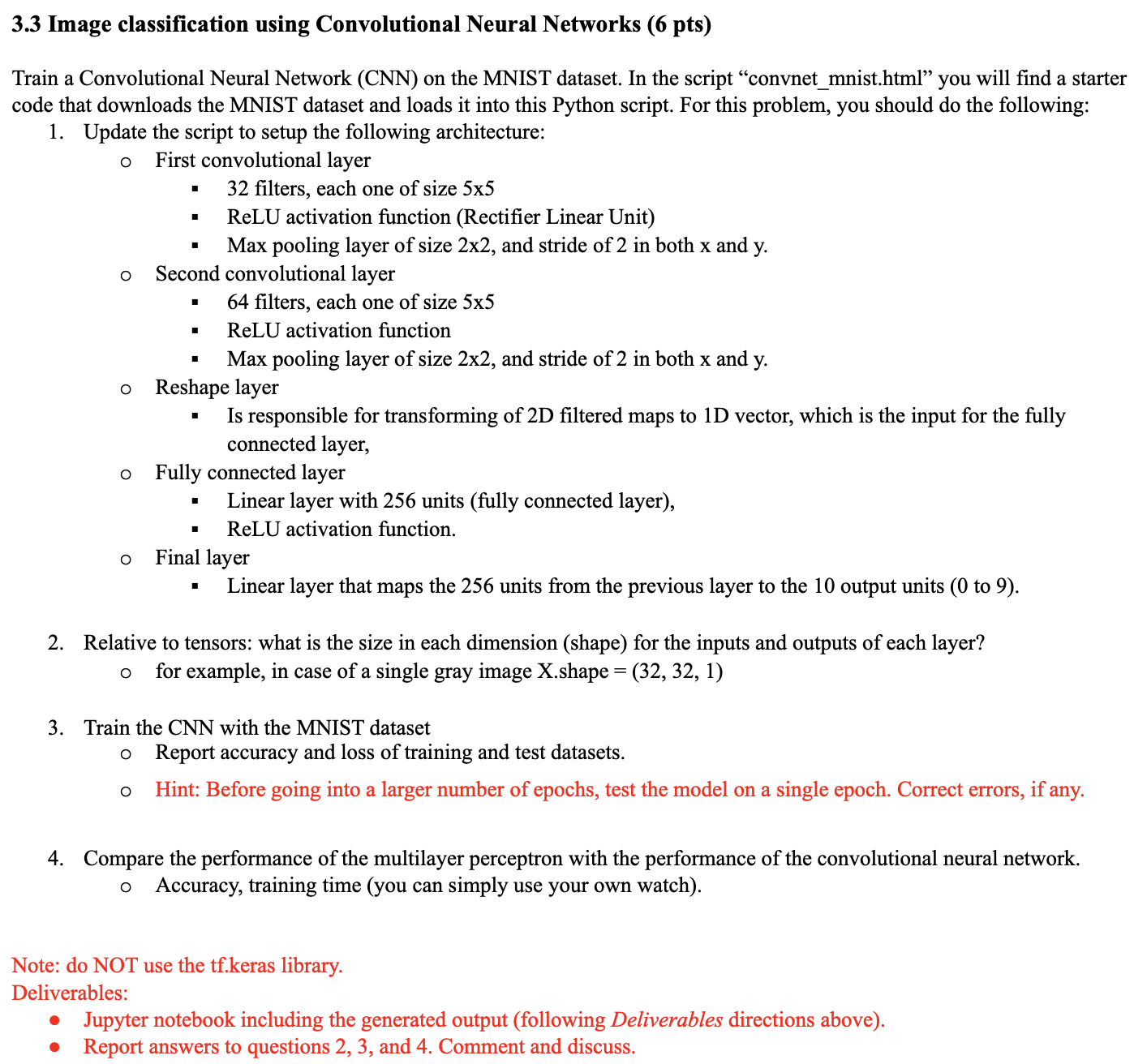


**Answer to the question no. 3.2:**Report on the accuracy on training and testing datasets:

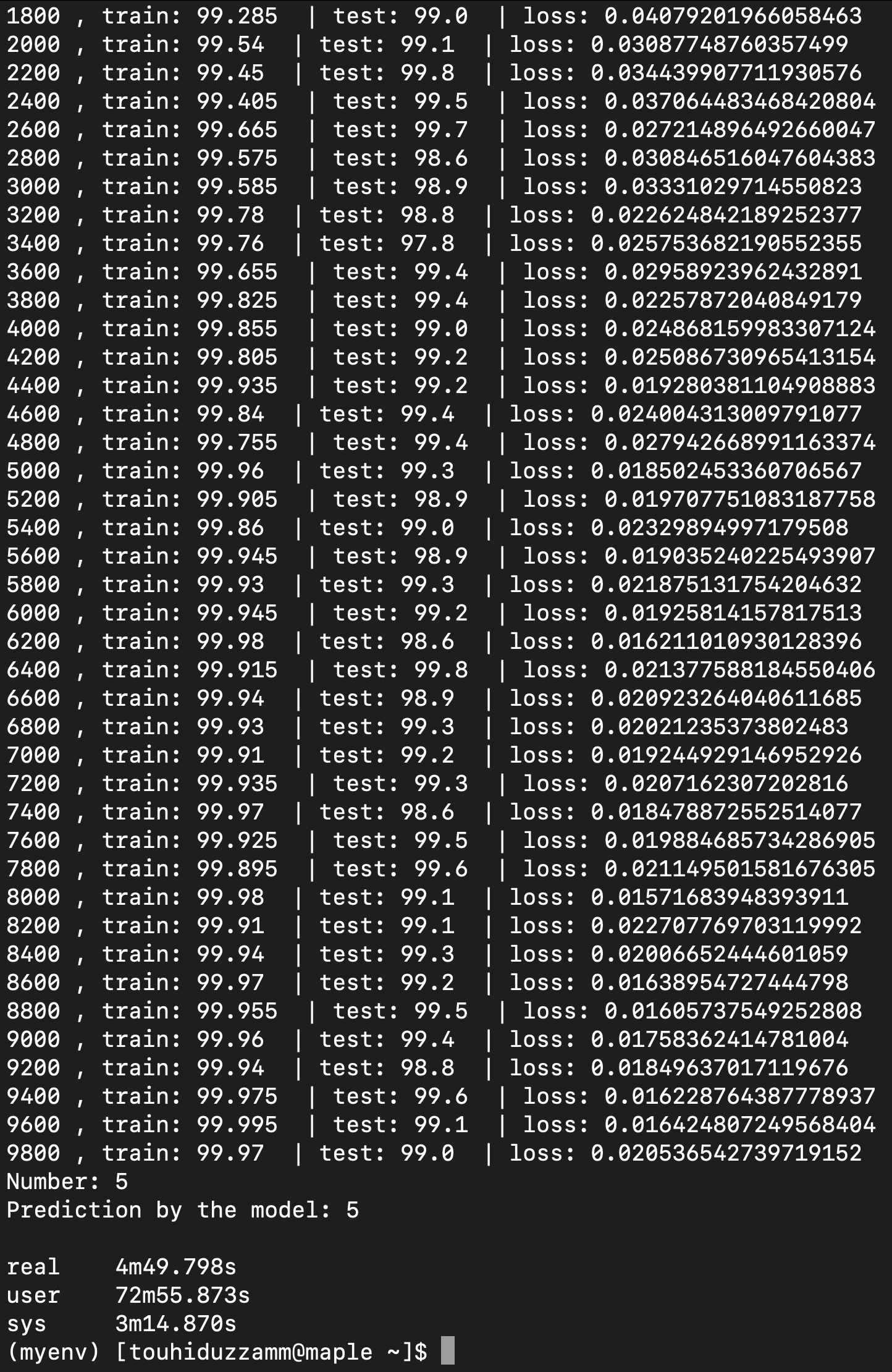
* Train accuracy: 99.595%
* Test accuracy: 97.4%

The provided script is updated & attached as “mnist\_mlp.ipynb” file. A screenshot of the results of the last epochs is:

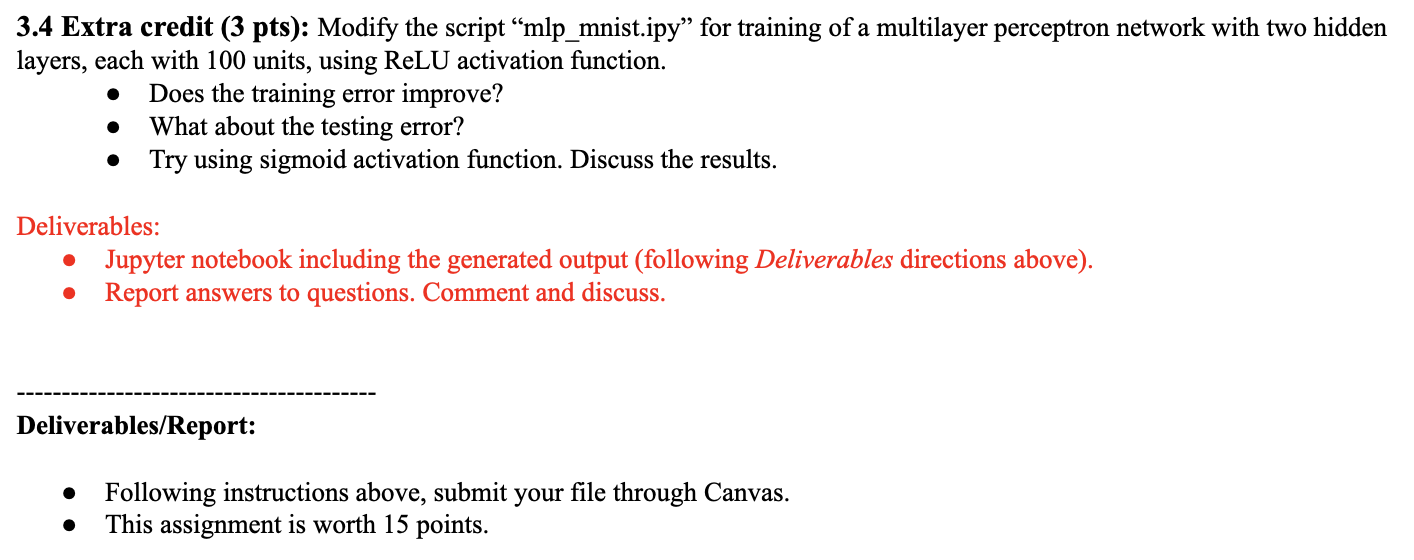




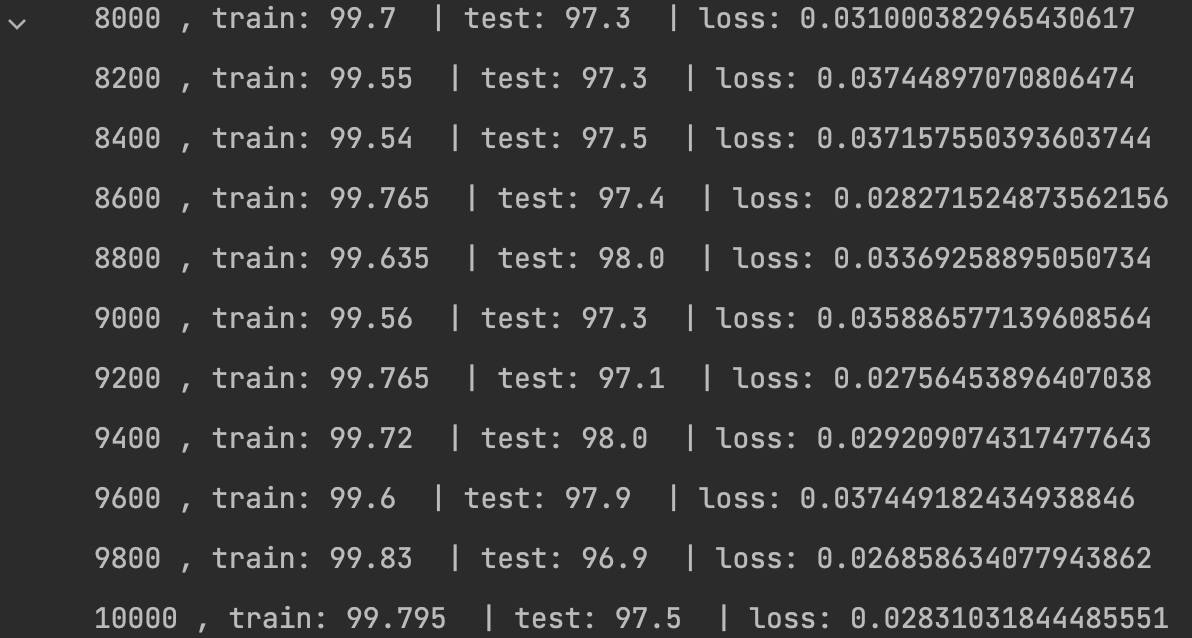
**Answer to the question no. 3.3:**

1. The script is updated as per the requirements and it is included with this submission as “mnist\_convnet.ipynb” file.
2. We have 5 layers in total, each with the following specification os input & output dimensions (shapes):
   1. First convolutional layer
      1. Input: 1
      2. Output: 32
   2. Second convolutional layer
      1. Input: 32
      2. Output: 64
   3. Reshape layer: [-1, 4\*4\*64]
   4. Fully connected layer
      1. Input: 4\*4\*64
      2. Output: 256
   5. Final layer
      1. Input: 256
      2. Output: 10
3. The implemented CNN network is trained with the MNIST dataset in local machine & a GPU enabled VCU server.   
   In a local machine without any use of GPU, we have got 99.845% training and 99.4% testing accuracy with 0.022966726017184556 as the final loss.  
   In the Maple server (a GPU enabled VCU shared server), we trained the network with 10000 epochs and have got 99.97% training and 99.0% testing accuracy with 0.02 loss at the final epoch. A screenshot of that run given below:  
   
4. MLP vs CNN in terms of Accuracy & Training Time:

The accuracy is found to be better with CNN than in MLP with the same dataset and same number of epochs. Even, CNN yields higher accuracy with half number of (5000) epochs than the MLP (10000 epochs) experiment.  
But, CNN requires more time to get trained than the MLP implementation. More than 6 minutes and 32 seconds of runtime was required for the CNN algorithms 5000 epochs to complete, which is a lot higher than the 10000 epochs of MLP requiring 1 minute and 27 seconds in the local machine. We achieved slightly better runtime for CNN by running it in the GPU enabled Maple server, which is 4 minute and 49.79 seconds.



**Answer to the question no. 3.4:**

The script is updated and attached with this submission as “mnist\_mlp\_34.ipynb”.  
We have noticed that the training accuracy of this updated version slightly improves to 99.795% than the previous 99.595%. Also, the testing accuracy improves a little to 97.5% from 97.4%.  


But using the sigmoid function does not change the accuracy that much than the ReLU function. Therefore, the ReLU activation function of the two hidden layers is a much better fit and the sigmoid function is an underfit in comparison to the MLP-2-hidden-layer.

~~~~~~ [The End] ~~~~~~