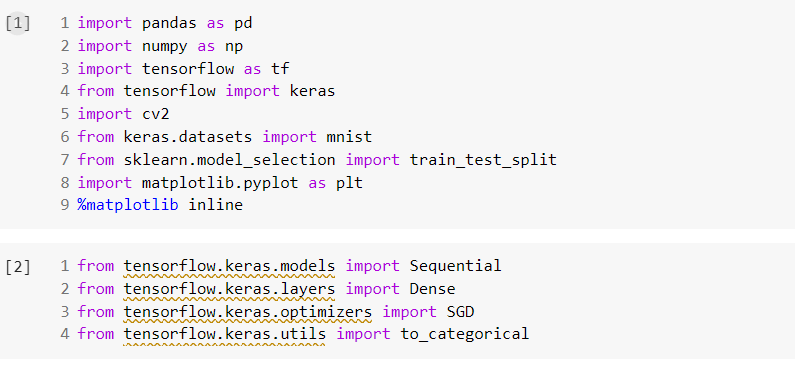
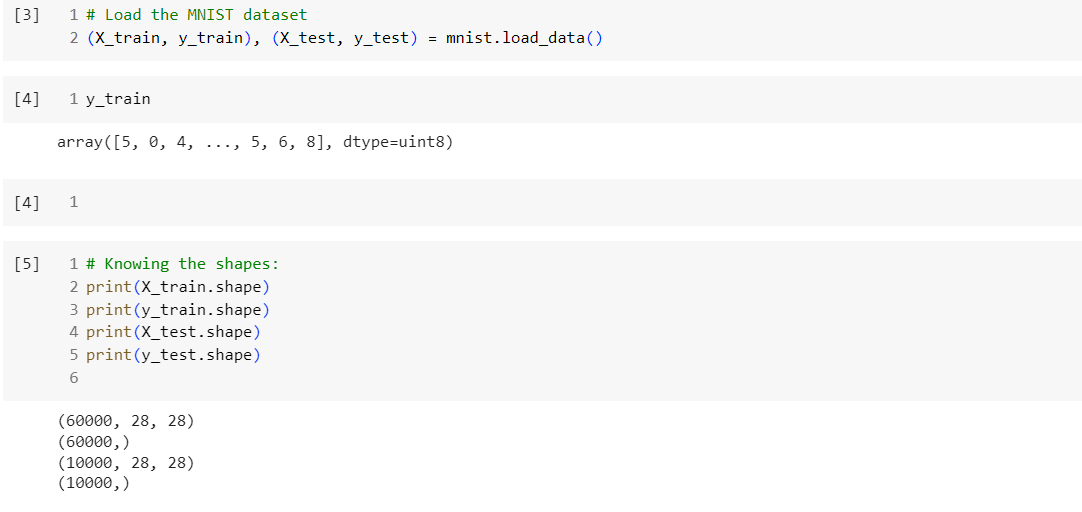
***Application of Keras to build, compile, and train a neural network as a multi-class classifier for MNIST dataset (0 vs. 1 vs. 2 vs. 3):***

**a) Use mnist function in keras.datasets to load MNIST dataset and split it into training and testing sets. Then, randomly select 20% of the training images along with their corresponding labels to be the validation data**

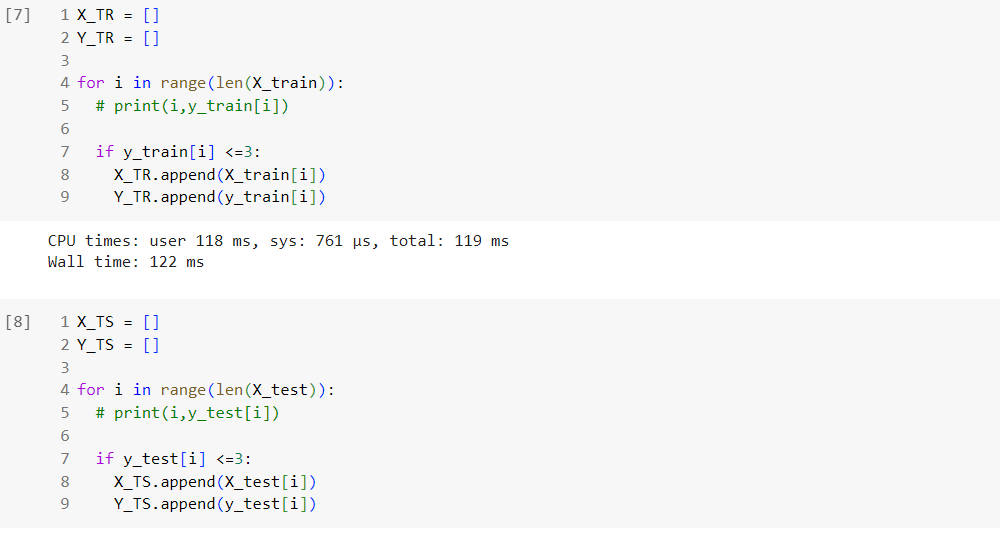
* Importing the required datasets:

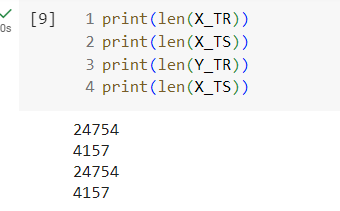


Loading the dataset:

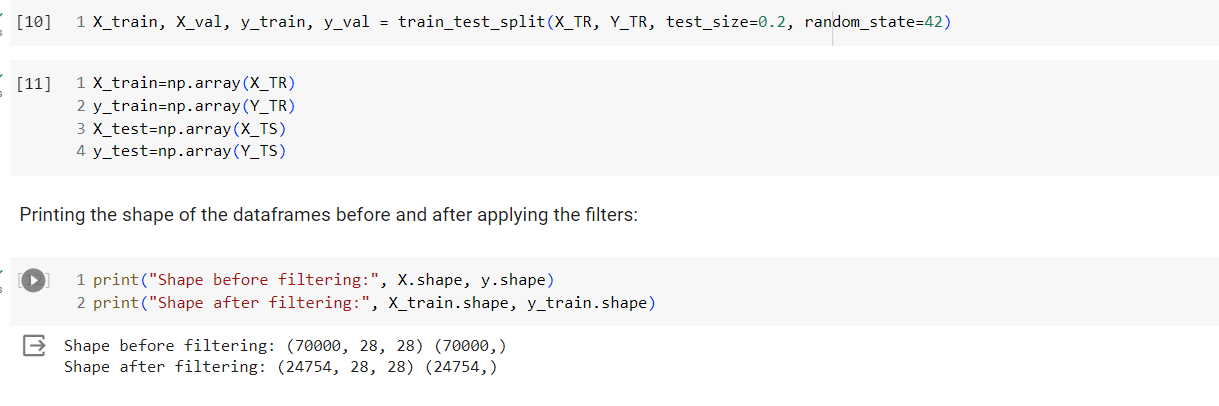


* Filtering the data (0 vs. 1 vs. 2 vs. 3):

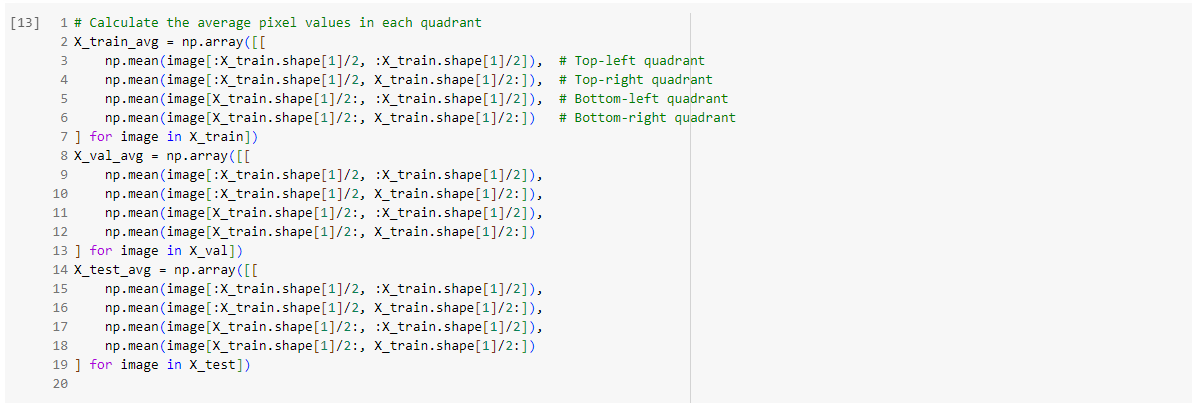




* randomly select 20% of the training images along with their corresponding labels to be the validation data.

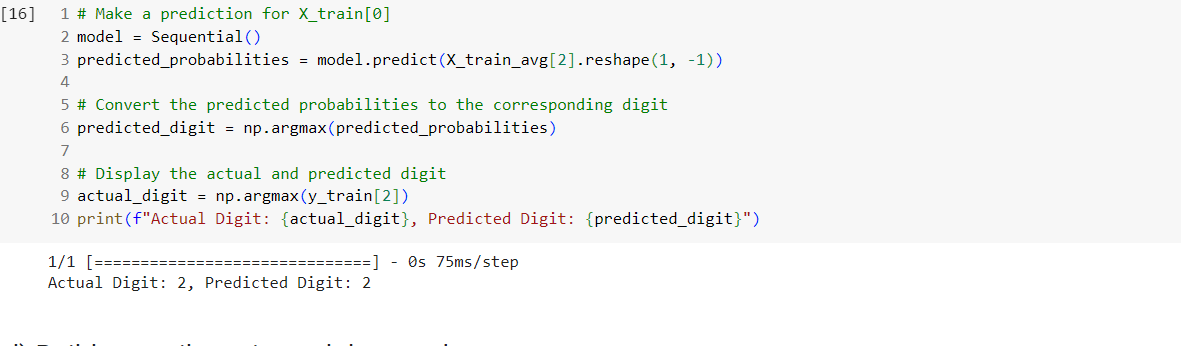


**b) Feature extraction: average the pixel values in the quadrants in each image to generate a feature vector of 4 values for each image.**

****

**c) Convert the label vectors for all the sets to binary class matrices using to\_categorical() Keras function**

****

****

**d) Build, compile, train, and then evaluate:**

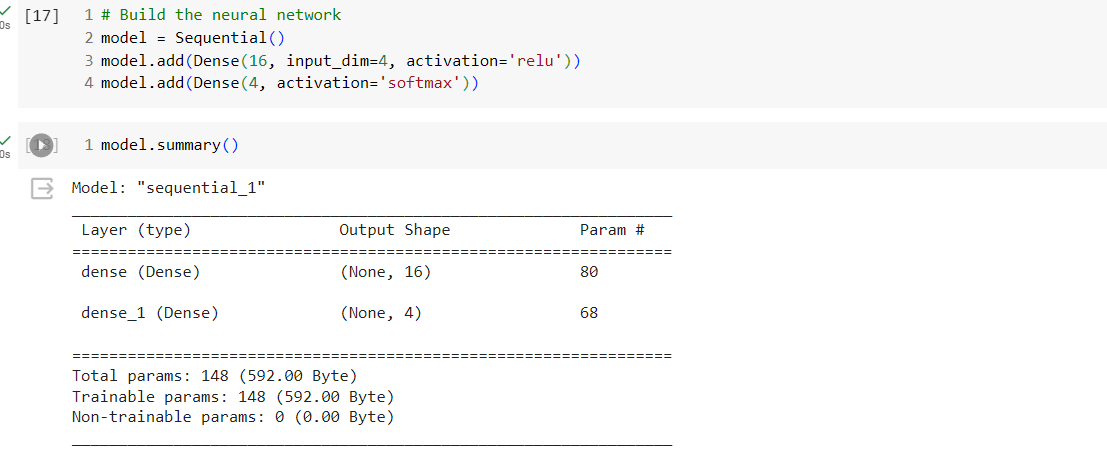
**i. Build a neural network with 1 layer that contains 16 nodes using the Keras library.**

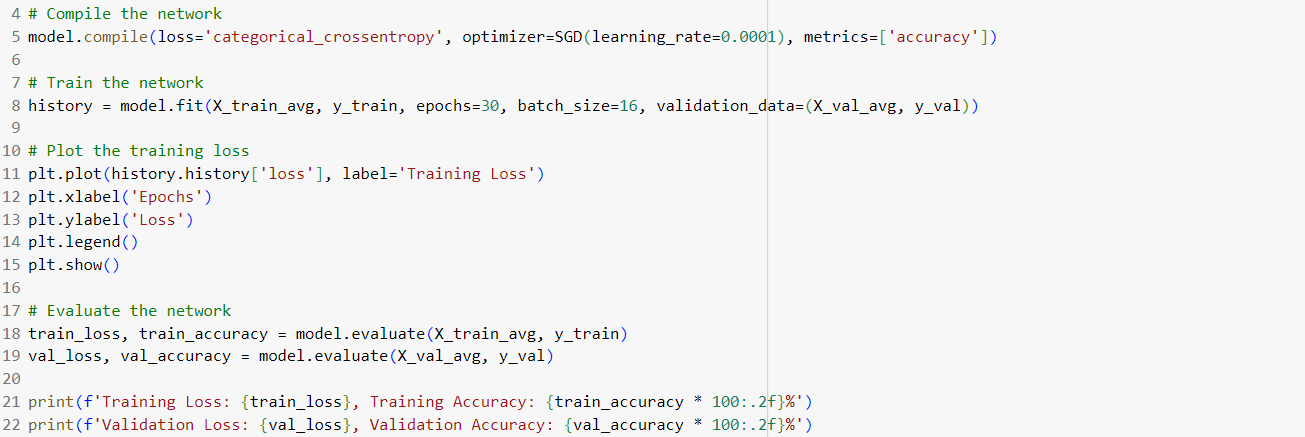
**ii. Compile the network. Make sure to select a correct loss function for this classification problem. Use stochastic gradient descent learning (SGD, learning rate of 0.0001). Explain your selection of the loss function.**

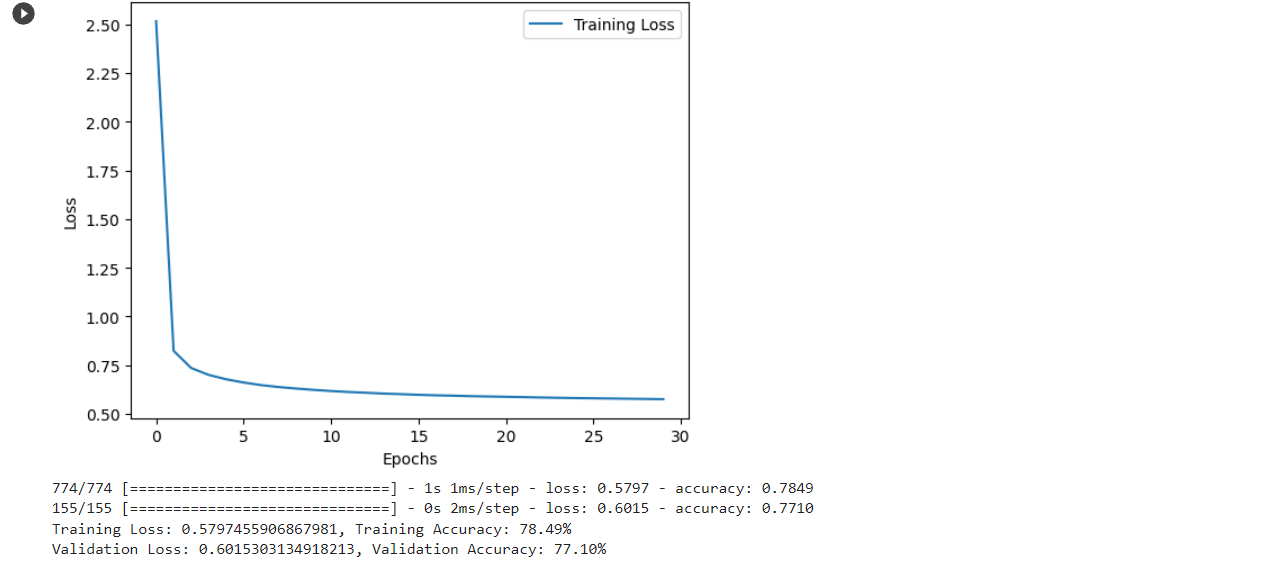
* **We used categorical\_crossentropy loss function**
* **categorical\_crossentropy is used for multi-class classification problems, where each input sample can belong to multiple classes**

**iii. Train the network for 30 epochs and a batch size of 16.**

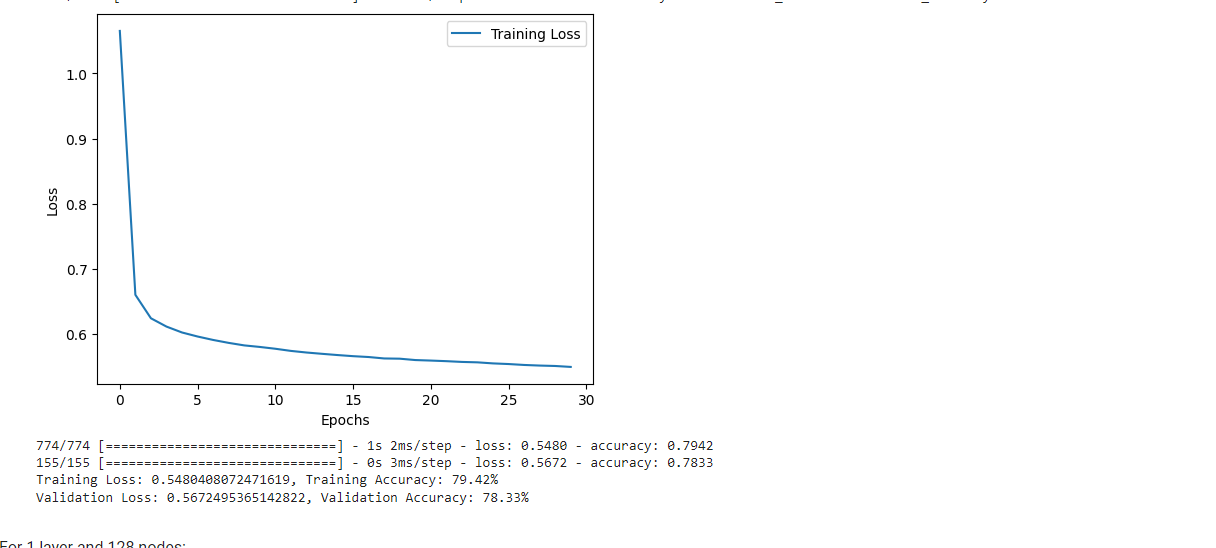
**iv. Plot the training loss (i.e., the learning curve) for all the epochs. v. Use the evaluate() Keras function to find the training and validation loss and accuracy.**

****

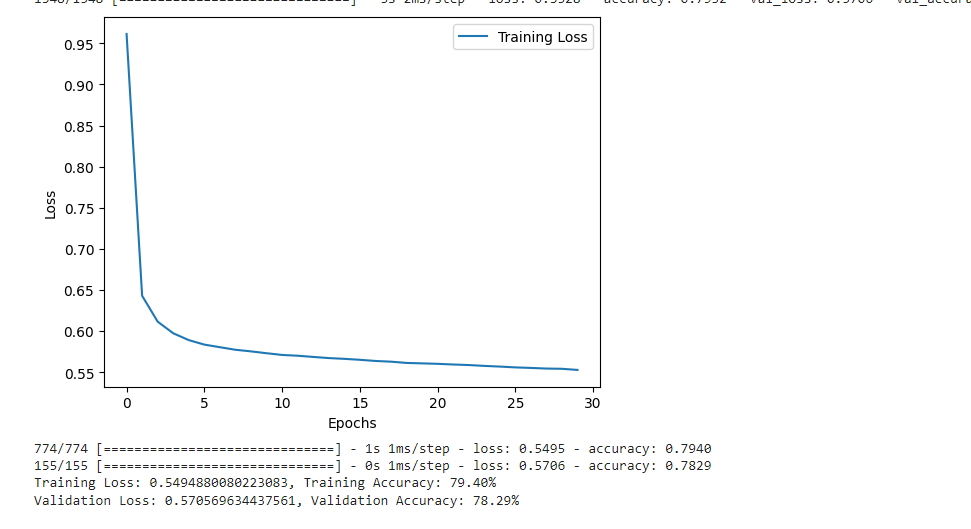
****

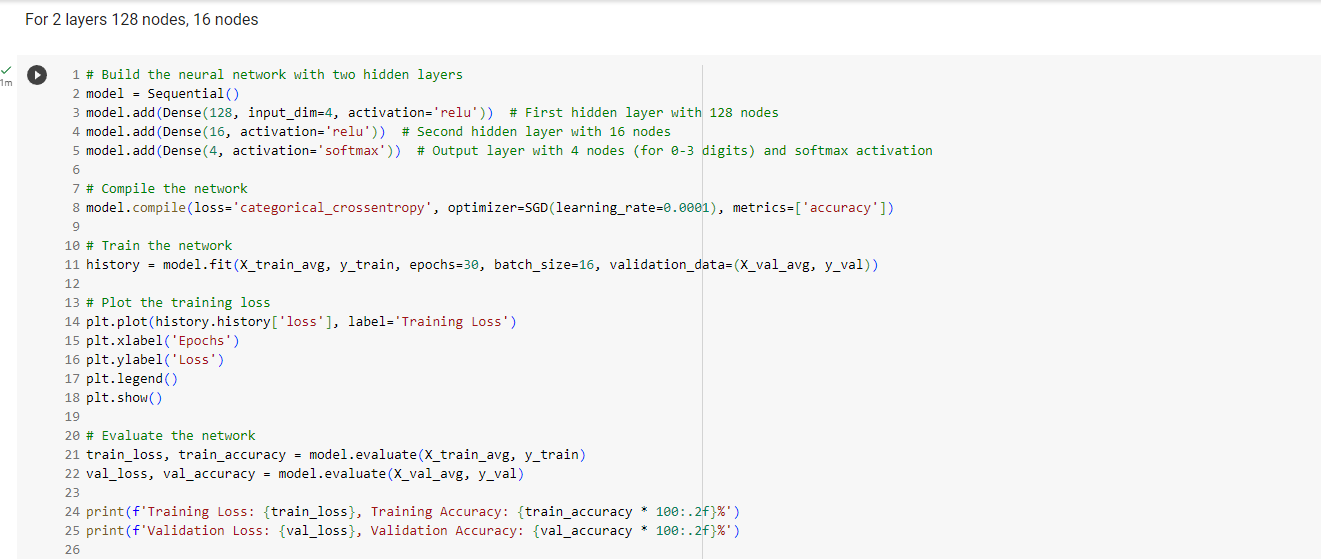
****

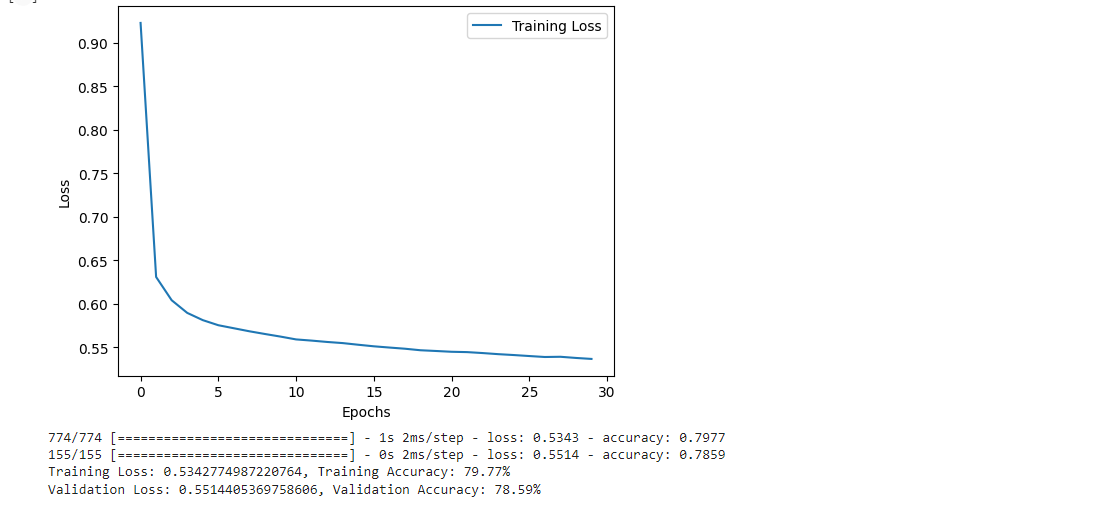
****

****

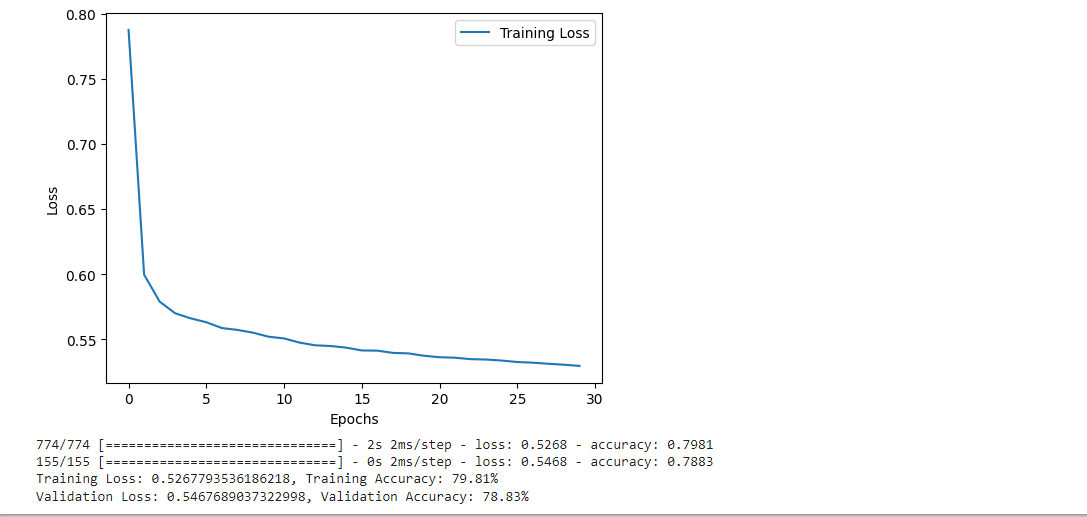
****

****

****

****

****

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model #** | **Details** | **Training** | | **Validation** | |
| loss | accuracy | loss | accuracy |
| 1 | 1 layer 16 nodes | 0.565 | 79.21% | 0.600 | 77.68% |
| 2 | 1 layer 64 nodes | 0.540 | 79.72% | 0.571 | 78.02% |
| 3 | 1 layer 128 nodes | 0.541 | 79.88% | 0.566 | 78.59% |
| 4 | 2 layers  128 nodes,  16 nodes | 0.53 | 80.00% | 0.558 | 78.25% |
| 5 | 2 layers  128 nodes,  64 nodes | 0.540 | 79.77% | 0.569 | 78.43% |

**f) What behavior do you observe in the training loss and the validation loss when you increase the number layers and nodes in the previous table. Which model is more suitable in this problem? Explain.**

From the results we get, it is clear that the behaviour of training and validation loss changes when we change the number of layers and nodes in the neural network. Based on the provided results, the 2-layer model with 128 nodes in the first layer and 16 nodes in the second layer appears to be the most suitable for this problem. It offers the following advantages:

It achieves the lowest validation loss among the models.

It has the highest validation accuracy, indicating better generalization.

It exhibits less overfitting compared to the 1-layer models, as evidenced by the smaller gap between training and validation losses.

* 1. Evaluate the selected model in part (e) on the testing set and report the testing loss and accuracy.

Testing Loss: 0.5267793536186218, Testing Accuracy: 79.81%

