**Department of Computing**



**CS-405 Deep Learning**

**Class: BESE-12B**

**Lab 04: Hyperparameter Tuning in Neural Networks**

**Date: 11th October 2024**

**Time: 2:30 pm to 5:00 pm**

**Instructor: Dr. Hashir Moheed Kiani**

**Submitted by**

**Jamal Ahmad Khan (366394)**

**Muhammad Hassaan Qaisar (369095)**

**Lab Report**

# **Introduction**

This lab focuses on the practical implementation of machine learning concepts, specifically around neural networks and hyperparameter tuning. It guides students through essential machine learning tasks such as data preprocessing, model building, training, and evaluation using the PyTorch framework. The lab emphasizes the importance of selecting appropriate hyperparameters, which play a crucial role in optimizing the performance of neural networks. By experimenting with different hyperparameter settings, students gain hands-on experience in improving model accuracy and efficiency. This lab provides a foundational understanding of how to tune and fine-tune models for various machine learning tasks.

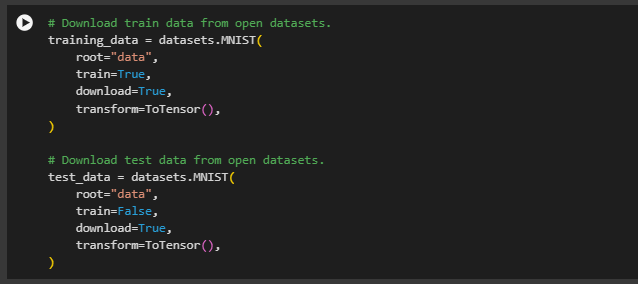
# **Task 1**

## **MNIST Dataset**

The MNIST dataset is one of the most well-known datasets in the field of machine learning and deep learning, often used as a benchmark for image classification tasks. It contains 70,000 grayscale images of handwritten digits (0-9), with 60,000 images for training and 10,000 for testing. Each image is 28x28 pixels in size, and the goal is to correctly classify the digits. Due to its simplicity and size, MNIST is widely used for training and testing various machine learning algorithms, particularly in the early stages of model development.

## **Downloading MNIST Dataset**

The following code snippet downloads the MNIST dataset, a collection of handwritten digit images, directly from the open datasets available in PyTorch. The datasets.MNIST function is used to download both the training and testing data, specifying the root directory where the data will be saved. The transform=ToTensor() argument ensures that the images are converted into tensor format, which is essential for using them in deep learning models. By setting download=True, the dataset is automatically downloaded if it's not already present.

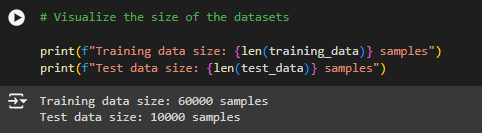


## **Visualizing some of the Input Images**

## C:\Users\jamal\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\mnist.jpeg

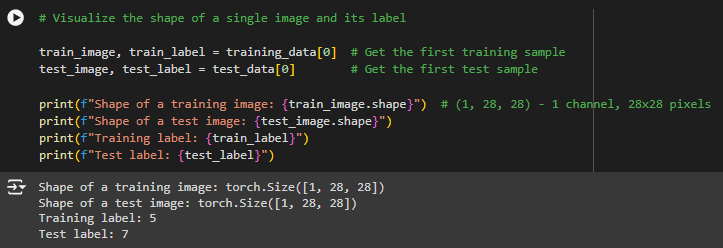
## **Size of the Dataset**

The following code snippet shows that how many entries does MNIST dataset has in its training and test sections.



## **Shape of a Single Image and its Label**

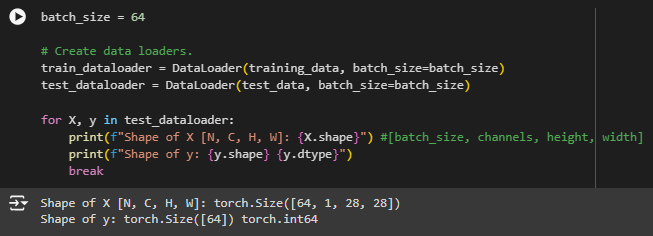
The following code snippet shows the shape of a single image and its label.



The shape of the training set is (6000,2). This means that we have 6000 thousand samples with 2 entities. The first entity is the image and the second entity is its label. For example:

* **training\_data[0][0]** is the first image of training set
* **training\_data[0][1]** is the label of the first image in the training set

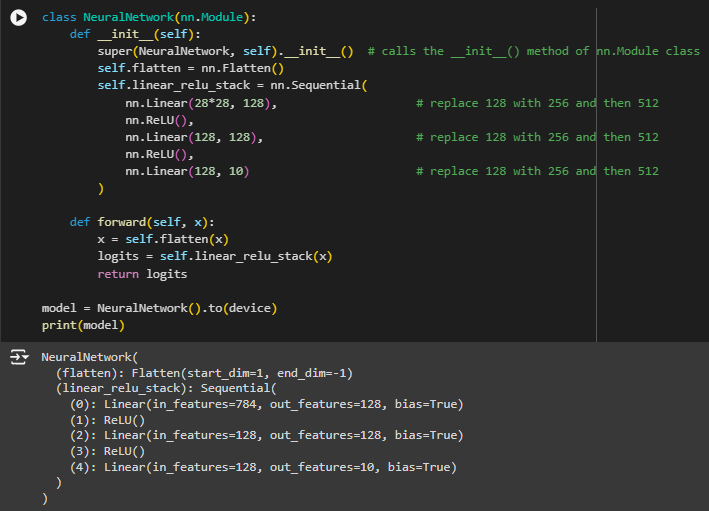
Now, we pass the Dataset as an argument to DataLoader. This wraps an iterable over our dataset, and supports automatic batching, sampling, shuffling and multiprocess data loading. Here we define a batch size of 64, i.e. each element in the dataloader iterable will return a batch of 64 features and labels.

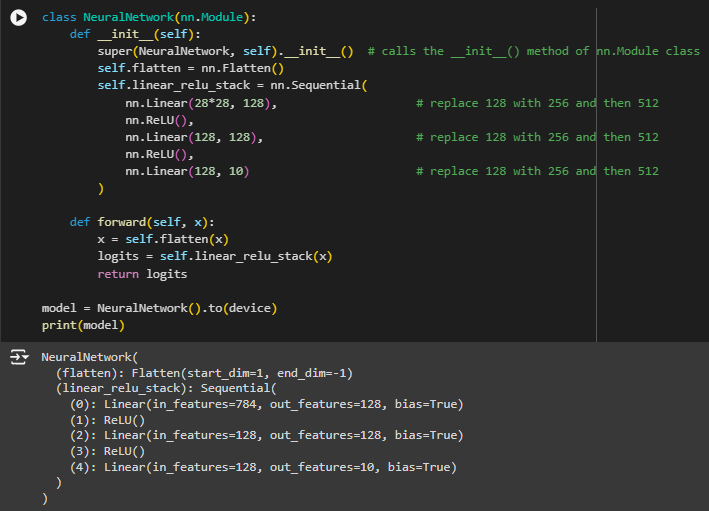


## **Define the Model**

To define a neural network in PyTorch, we create a class that inherits from nn.Module. We define the layers of the network in the \_\_init\_\_ function and specify how data will pass through the network in the forward function. To accelerate operations in the neural network, we move it to the GPU if available.

In the following code, I have first used the 128 neurons model and test it with different variation of hyperparameters and then similarly for 256 and 512 neurons.





## **Observations and Findings**

The following table shows all the different combinations of models neurons and different combinations of hyperparameters that are used to give various sort of results and one can look up at the values and can find the best one for its usage.

### **Model with 128 Neurons**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Optimizer | Learning Rate | Epochs | Accuracy (%) | Val. Loss |
| SGD | -3 | 5 | 69.9 | 1.889621 |
| SGD | -3 | 10 | 79.1 | 0.830572 |
| SGD | -3 | 50 | 91.5 | 0.291083 |
| SGD | -4 | 5 | 15.8 | 2.287481 |
| SGD | -4 | 10 | 10.1 | 2.282166 |
| SGD | -4 | 50 | 68.5 | 1.844733 |
| Adam | -3 | 5 | 97.5 | 0.082814 |
| Adam | -3 | 10 | 97.2 | 0.116796 |
| Adam | -3 | 50 | 98.1 | 0.166093 |
| Adam | -4 | 5 | 93.8 | 0.209387 |
| Adam | -4 | 10 | 95.9 | 0.139255 |
| Adam | -4 | 50 | 97.7 | 0.088755 |

### **Model with 256 Neurons**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Optimizer | Learning Rate | Epochs | Accuracy (%) | Val. Loss |
| SGD | -3 | 5 | 67.4 | 1.856814 |
| SGD | -3 | 10 | 83.1 | 0.687489 |
| SGD | -3 | 50 | 92.0 | 0.281274 |
| SGD | -4 | 5 | 23.0 | 2.284172 |
| SGD | -4 | 10 | 40.0 | 2.264763 |
| SGD | -4 | 50 | 66.0 | 1.758352 |
| Adam | -3 | 5 | 96.5 | 0.138005 |
| Adam | -3 | 10 | 98.0 | 0.086092 |
| Adam | -3 | 50 | 98.1 | 0.161446 |
| Adam | -4 | 5 | 95.5 | 0.151744 |
| Adam | -4 | 10 | 96.8 | 0.102958 |
| Adam | -4 | 50 | 98.0 | 0.105934 |

### **Model with 512 Neurons**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Optimizer | Learning Rate | Epochs | Accuracy (%) | Val. Loss |
| SGD | -3 | 5 | 70.3 | 1.671867 |
| SGD | -3 | 10 | 84.8 | 0.618288 |
| SGD | -3 | 50 | 92.2 | 0.273662 |
| SGD | -4 | 5 | 24.1 | 2.282232 |
| SGD | -4 | 10 | 36.1 | 2.261677 |
| SGD | -4 | 50 | 75.1 | 1.543590 |
| Adam | -3 | 5 | 96.9 | 0.132185 |
| Adam | -3 | 10 | 97.8 | 0.104475 |
| Adam | -3 | 50 | 98.2 | 0.196091 |
| Adam | -4 | 5 | 96.6 | 0.112004 |
| Adam | -4 | 10 | 97.5 | 0.079524 |
| Adam | -4 | 50 | 98.2 | 0.116876 |

Now, if we look at all these values, and carefully selecting the model that gives us highest accuracy with low loss would be our possible achievable goal. So, by carefully selecting the model that gives so, is the following:

**Model with 512 neurons, having Adam optimizer with learning rate of 1e-3 that gives us an average accuracy of 98.25 with an average validation loss of 0.196091**

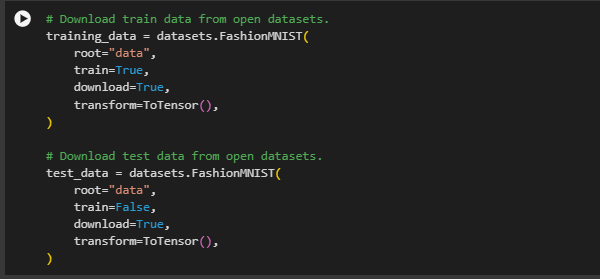
# **Task 2**

## **Fashion MNIST Dataset**

The Fashion MNIST dataset is a more complex alternative to the MNIST dataset, designed to challenge models with more realistic data. It contains 70,000 grayscale images of fashion items (such as shirts, shoes, and bags), also divided into 60,000 training and 10,000 testing images, with each image being 28x28 pixels. The dataset has 10 categories of fashion items, and like MNIST, it serves as a benchmark for image classification tasks. Fashion MNIST is often used to test models' robustness on more diverse and complex visual data compared to the simpler digit classification task in MNIST.

## **Downloading Fashion MNIST Dataset**

The following code snippet is used to download the Fashion MNIST dataset, which contains images of clothing items such as shoes, shirts, and bags. The datasets.FashionMNIST function is called to retrieve both the training and testing datasets, with the images being transformed into tensors using ToTensor(). The dataset is saved in the specified root directory, and the download=True option ensures that the dataset is downloaded if it doesn't exist locally.

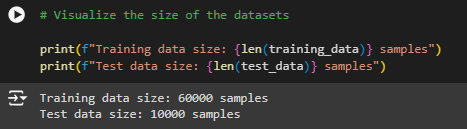


## **Visualizing some of the Input Images**

## C:\Users\jamal\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\fashion mnist.jpeg

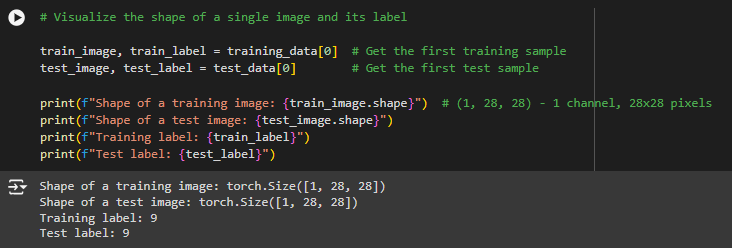
## **Size of the Dataset**

The following code snippet shows that how many entries does Fashion MNIST dataset has in its training and test sections.



## **Shape of a Single Image and its Label**

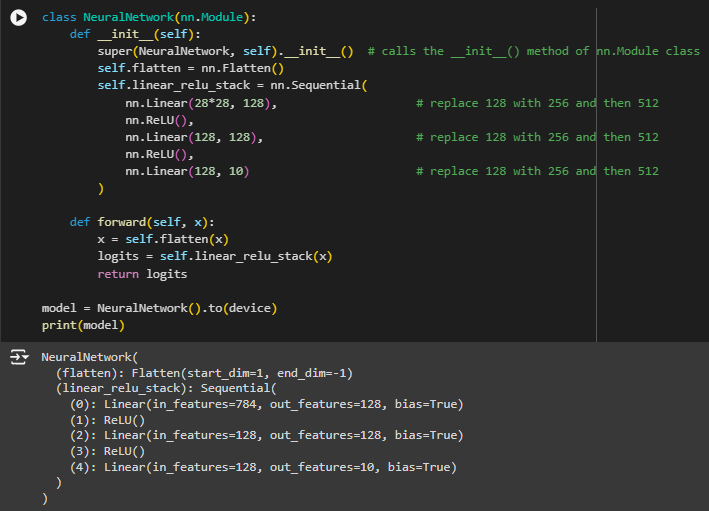
The following code snippet shows the shape of a single image and its label.

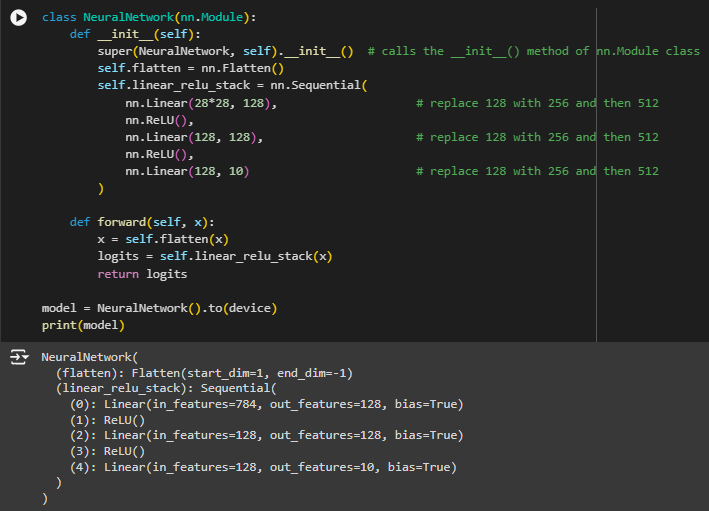


## **Define the Model**

To define a neural network in PyTorch, we create a class that inherits from nn.Module. We define the layers of the network in the \_\_init\_\_ function and specify how data will pass through the network in the forward function. To accelerate operations in the neural network, we move it to the GPU if available.

In the following code, I have first used the 128 neurons model and test it with different variation of hyperparameters and then similarly for 256 and 512 neurons.





## **Observations and Findings**

The following table shows all the different combinations of models neurons and different combinations of hyperparameters that are used to give various sort of results and one can look up at the values and can find the best one for its usage.

### **Model with 128 Neurons**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Optimizer | Learning Rate | Epochs | Accuracy (%) | Val. Loss |
| SGD | -3 | 5 | 60.1 | 1.268713 |
| SGD | -3 | 10 | 68.4 | 0.824849 |
| SGD | -3 | 50 | 82.1 | 0.502641 |
| SGD | -4 | 5 | 19.1 | 2.275382 |
| SGD | -4 | 10 | 28.1 | 2.209133 |
| SGD | -4 | 50 | 63.3 | 1.224722 |
| Adam | -3 | 5 | 87.0 | 0.352771 |
| Adam | -3 | 10 | 87.5 | 0.350339 |
| Adam | -3 | 50 | 87.9 | 0.647922 |
| Adam | -4 | 5 | 84.4 | 0.435537 |
| Adam | -4 | 10 | 86.0 | 0.394178 |
| Adam | -4 | 50 | 88.1 | 0.337178 |

### **Model with 256 Neurons**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Optimizer | Learning Rate | Epochs | Accuracy (%) | Val. Loss |
| SGD | -3 | 5 | 63.4 | 1.140300 |
| SGD | -3 | 10 | 70.1 | 0.813381 |
| SGD | -3 | 50 | 82.1 | 0.502094 |
| SGD | -4 | 5 | 26.9 | 2.257945 |
| SGD | -4 | 10 | 40.4 | 2.198068 |
| SGD | -4 | 50 | 63.4 | 1.126220 |
| Adam | -3 | 5 | 87.2 | 0.345128 |
| Adam | -3 | 10 | 87.5 | 0.363967 |
| Adam | -3 | 50 | 88.5 | 0.758998 |
| Adam | -4 | 5 | 85.7 | 0.408205 |
| Adam | -4 | 10 | 87.1 | 0.365625 |
| Adam | -4 | 50 | 88.7 | 0.366195 |

### **Model with 512 Neurons**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Optimizer | Learning Rate | Epochs | Accuracy (%) | Val. Loss |
| SGD | -3 | 5 | 65.3 | 1.080217 |
| SGD | -3 | 10 | 71.2 | 0.780698 |
| SGD | -3 | 50 | 82.7 | 0.489227 |
| SGD | -4 | 5 | 34.4 | 2.232200 |
| SGD | -4 | 10 | 42.2 | 2.165706 |
| SGD | -4 | 50 | 64.3 | 1.084387 |
| Adam | -3 | 5 | 86.7 | 0.364216 |
| Adam | -3 | 10 | 87.6 | 0.389106 |
| Adam | -3 | 50 | 88.4 | 0.833555 |
| Adam | -4 | 5 | 86.5 | 0.376754 |
| Adam | -4 | 10 | 87.6 | 0.344080 |
| Adam | -4 | 50 | 87.9 | 0.513383 |

Now, if we look at all these values, and carefully selecting the model that gives us highest accuracy with low loss would be our possible achievable goal. So, by carefully selecting the model that gives so, is the following:

**Model with 512 neurons, having Adam optimizer with learning rate of 1e-3 that gives us an average accuracy of 88.40% with an average validation loss of 0.833555**

**THE END**