# INFOMCV Assignment 4

# Taher Jarjanazi & Martino Fabiani (Group 20)


## Description and motivation of the baseline model and four variants

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 6, 28, 28] | 156 |
| ReLU-2 | [-1, 6, 28, 28] | 0 |
| MaxPool2d-3 | [-1, 6, 14, 14] | 0 |
| Conv2d-4 | [-1, 16, 10, 10] | 2,416 |
| ReLU-5 | [-1, 16, 10, 10] | 0 |
| MaxPool2d-6 | [-1, 16, 5, 5] | 0 |
| Conv2d-7 | [-1, 120, 1, 1] | 48,120 |
| ReLU-8 | [-1, 120, 1, 1] | 0 |
| Flatten-9 | [-1, 120] | 0 |
| Linear-10 | [-1, 84] | 10,164 |
| Linear-11 | [-1, 10] | 850 |
| Softmax-12 | [-1, 10] | 0 |

Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0

Input size (MB): 0.002991
Forward/backward pass size (MB): 0.111755
Params size (MB): 0.235390
Estimated Total Size (MB): 0.350136

For the baseline model we construct the architecture based on the LeNet5 network architecture, although with some deviations from the original paper. We refer to this model as LeNet5 1.0 henceforth. The first layer is a 2D convolutional layer where a single-channel input of 28x28 is processed. Here a filter of size 5x5 is run over the input with stride of 1 and zero-padding of 2 in order to keep the same input dimensions. The output of Conv2d-1 is a 6-channels 28x28 activation map. Conv2d-1 is followed by a Rectified Linear Unit (ReLU) activation function to introduce nonlinearity to the network, enabling the network to learn complex patterns in the data more effectively. Next, max pooling is applied to reduce the dimensions of the input to 14x14. The fourth layer is another 2D convolutional layer, denoted as Conv2d-4. Here, the input from the previous layer, with 6 channels and dimensions of 14x14, undergoes convolution with 16 filters of size 5x5. This convolutional operation yields an output tensor of dimensions 16x10x10. Subsequently, a ReLU activation function is applied elementwise.

After the activation, max pooling is performed with a kernel size of 2x2 and a stride of 2. This pooling operation reduces the dimensions of the input by a factor of 2, resulting in an output tensor of dimensions 16x5x5.

The seventh layer is a pivotal component of the LeNet5 1.0 architecture. It consists of a 2D convolutional layer denoted as Conv2d-7. Operating on the output from the previous layer, which comprises 16 channels with spatial dimensions of 5x5, this convolutional layer employs a set of 120 filters. Each filter has a spatial extent of 5x5, resulting in convolutions being applied across the entire input. The output of this layer is a tensor with a depth of 120 and spatial dimensions reduced to 1x1.

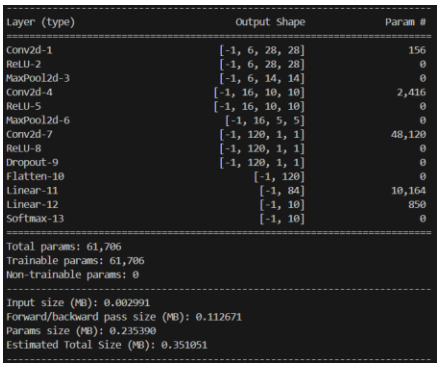
Following the convolution operation, ReLU is applied elementwise to the output tensor.

Subsequently, the output tensor is flattened into a vector form using the Flatten operation. This transformation reshapes the tensor from a multi-dimensional array into a one-dimensional vector while preserving the total number of elements.

The flattened vector is then fed into two fully connected layers, denoted as Linear-10 and Linear-11. Linear-10 comprises 84 neurons, facilitating the extraction of high-level features from the flattened input. Finally, Linear-11, with 10 neurons, serves as the output layer responsible for generating the classification scores for each class.

The SoftMax function is applied to the output of Linear-11, producing a probability distribution over the 10 classes. This distribution represents the model's confidence in each class prediction.

The first variation (LeNet5 2.0) of the baseline model adds a Dropout layer (denoted as Dropout-9) after the seventh layer (Conv2d-7) before flattening the output. This serves a crucial purpose in regularizing the network to prevent overfitting, where it works by randomly setting a fraction of input units (in this case 0.1) to zero during the training process, effectively disabling them. This prevents the network from becoming overly reliant on specific activations or features, thus promoting robustness and generalization to unseen data. The choice of putting the Dropout layer before the Flatten-10 layer is because all the nonlinear activations and feature extraction layers are before it. Also, a dropout rate of 0.1 is meant to regularize the network without decreasing the performance.

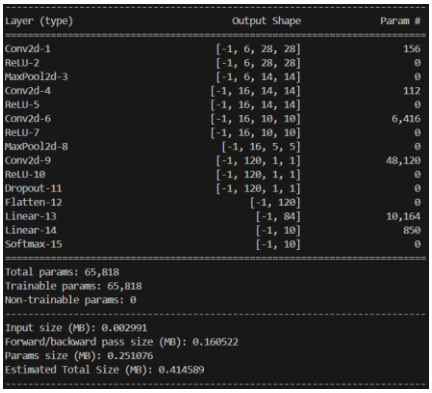


| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 6, 28, 28] | 156 |
| ReLU-2 | [-1, 6, 28, 28] | 0 |
| MaxPool2d-3 | [-1, 6, 14, 14] | 0 |
| Conv2d-4 | [-1, 16, 10, 10] | 2,416 |
| ReLU-5 | [-1, 16, 10, 10] | 0 |
| MaxPool2d-6 | [-1, 16, 5, 5] | 0 |
| Conv2d-7 | [-1, 120, 1, 1] | 48,120 |
| ReLU-8 | [-1, 120, 1, 1] | 0 |
| Dropout-9 | [-1, 120, 1, 1] | 0 |
| Flatten-10 | [-1, 120] | 0 |
| Linear-11 | [-1, 84] | 10,164 |
| Linear-12 | [-1, 10] | 850 |
| Softmax-13 | [-1, 10] | 0 |

Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0

Input size (MB): 0.002991
Forward/backward pass size (MB): 0.112671
Params size (MB): 0.235390
Estimated Total Size (MB): 0.351051

The second variation (LeNet5 3.0) builds on top of LeNet5 2.0 where we add a convolutional layer after the first max pooling layer, where a filter of size 1x1 is applied to preserve the spatial dimensions of 16x10x10. This layer is followed by a ReLU activation function. The additional convolutional layer aims at increasing the complexity of the network and adding more parameters in the hope of discovering more useful patterns in the input.

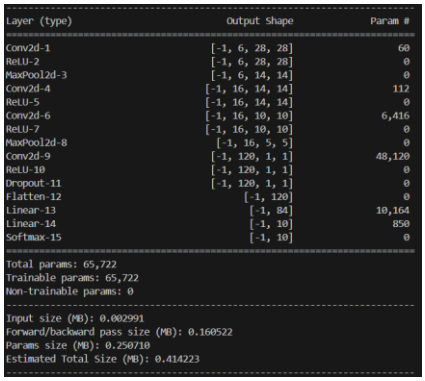


| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 6, 28, 28] | 156 |
| ReLU-2 | [-1, 6, 28, 28] | 0 |
| MaxPool2d-3 | [-1, 6, 14, 14] | 0 |
| Conv2d-4 | [-1, 16, 14, 14] | 112 |
| ReLU-5 | [-1, 16, 14, 14] | 0 |
| Conv2d-6 | [-1, 16, 10, 10] | 6,416 |
| ReLU-7 | [-1, 16, 10, 10] | 0 |
| MaxPool2d-8 | [-1, 16, 5, 5] | 0 |
| Conv2d-9 | [-1, 120, 1, 1] | 48,120 |
| ReLU-10 | [-1, 120, 1, 1] | 0 |
| Dropout-11 | [-1, 120, 1, 1] | 0 |
| Flatten-12 | [-1, 120] | 0 |
| Linear-13 | [-1, 84] | 10,164 |
| Linear-14 | [-1, 10] | 850 |
| Softmax-15 | [-1, 10] | 0 |

Total params: 65,818
Trainable params: 65,818
Non-trainable params: 0

Input size (MB): 0.002991
Forward/backward pass size (MB): 0.160522
Params size (MB): 0.251076
Estimated Total Size (MB): 0.414589

The third variation (LeNet5 4.0) builds on top of LeNet5 3.0 where the kernel size of the 1st convolutional layer is changed from 5x5 to 3x3 while changing the zero-padding to 1 to maintain the spatial dimensions. A smaller kernel is used to extract more features from the image itself and because the input dimensions are already small in this input, a smaller kernel size is also sufficient and more appropriate for getting more local features.



| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 6, 28, 28] | 60 |
| ReLU-2 | [-1, 6, 28, 28] | 0 |
| MaxPool2d-3 | [-1, 6, 14, 14] | 0 |
| Conv2d-4 | [-1, 16, 14, 14] | 112 |
| ReLU-5 | [-1, 16, 14, 14] | 0 |
| Conv2d-6 | [-1, 16, 10, 10] | 6,416 |
| ReLU-7 | [-1, 16, 10, 10] | 0 |
| MaxPool2d-8 | [-1, 16, 5, 5] | 0 |
| Conv2d-9 | [-1, 120, 1, 1] | 48,120 |
| ReLU-10 | [-1, 120, 1, 1] | 0 |
| Dropout-11 | [-1, 120, 1, 1] | 0 |
| Flatten-12 | [-1, 120] | 0 |
| Linear-13 | [-1, 84] | 10,164 |
| Linear-14 | [-1, 10] | 850 |
| Softmax-15 | [-1, 10] | 0 |

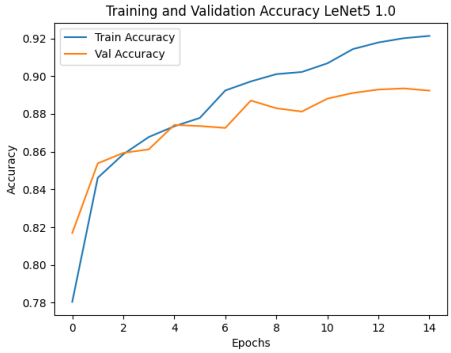
Total params: 65,722
Trainable params: 65,722
Non-trainable params: 0

Input size (MB): 0.002991
Forward/backward pass size (MB): 0.160522
Params size (MB): 0.250710
Estimated Total Size (MB): 0.414223

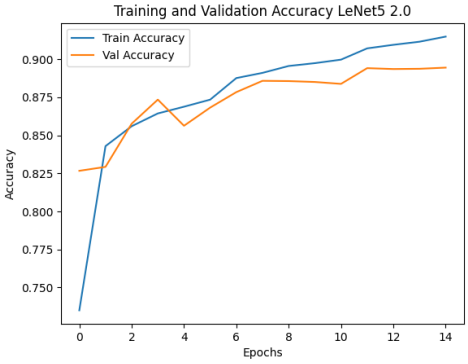
The fourth and final variation (LeNet5 5.0) builds on top of LeNet5 4.0 where we introduce a batch normalization layer after every convolutional layer. Batch normalization helps keep a stable convergence while speeding up the training process.

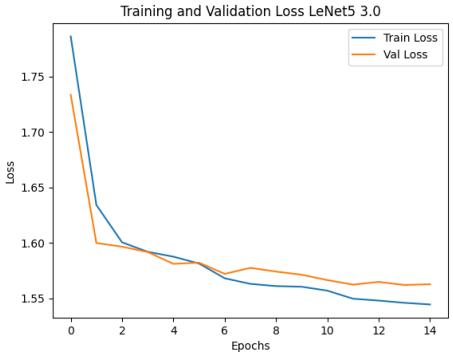
```
Layer (type)                 Output Shape              Param #
================================================================
Conv2d-1                     [-1, 6, 28, 28]              60
BatchNorm2d-2                [-1, 6, 28, 28]              12
ReLU-3                       [-1, 6, 28, 28]               0
MaxPool2d-4                  [-1, 6, 14, 14]               0
Conv2d-5                     [-1, 16, 14, 14]            112
BatchNorm2d-6                [-1, 16, 14, 14]             32
ReLU-7                       [-1, 16, 14, 14]              0
Conv2d-8                     [-1, 16, 10, 10]          6,416
BatchNorm2d-9                [-1, 16, 10, 10]             32
ReLU-10                      [-1, 16, 10, 10]              0
MaxPool2d-11                  [-1, 16, 5, 5]               0
Conv2d-12                    [-1, 120, 1, 1]         48,120
BatchNorm2d-13               [-1, 120, 1, 1]            240
ReLU-14                      [-1, 120, 1, 1]              0
Dropout-15                   [-1, 120, 1, 1]              0
Flatten-16                        [-1, 120]               0
Linear-17                          [-1, 84]          10,164
Linear-18                          [-1, 10]             850
Softmax-19                         [-1, 10]               0
================================================================
Total params: 66,038
Trainable params: 66,038
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.002991
Forward/backward pass size (MB): 0.233459
Params size (MB): 0.251915
Estimated Total Size (MB): 0.488365
----------------------------------------------------------------
```

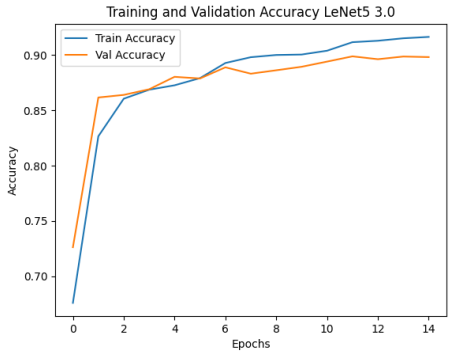
## Training and validation loss for all five models

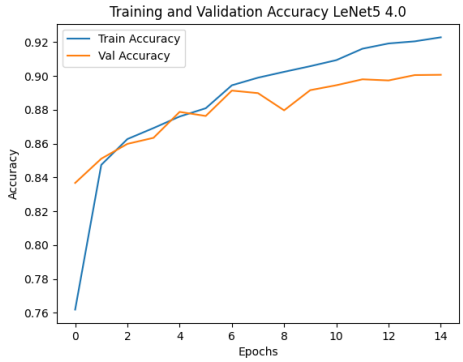



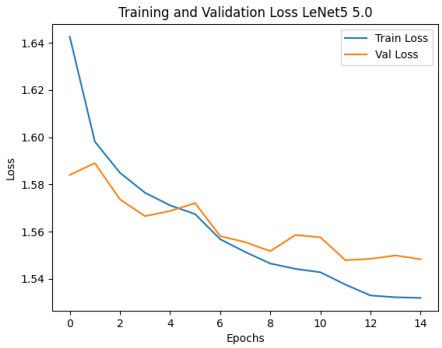


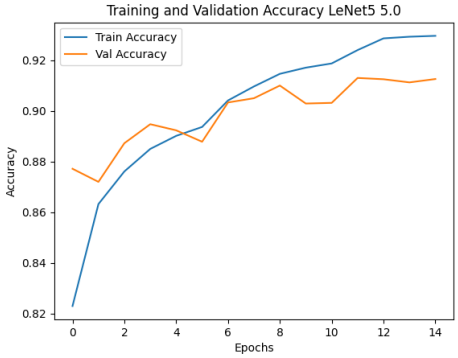



**Link to model weights**

Models

**Table with training and validation top-1 accuracy for all five models**

| Model name | Training top-1 accuracy (%) | Validation top-1 accuracy (%) |
|---|---|---|
| LeNet5 1.0 (Baseline) | 92.131250 | 89.233333 |
| LeNet5 2.0 | 91.479167 | 89.441667 |
| LeNet5 3.0 | 91.631250 | 89.808333 |
| LeNet5 4.0 | 92.283333 | 90.066667 |
| LeNet5 5.0 | 92.962500 | 91.258333 |

**Discussion of results in terms of baseline model**

Initially, starting the discussion by comparing the base configuration with the first variation, which adds a dropout layer of 0.1, it is possible to observe a slight increase in accuracy during the validation phase (89.23% for model 1 and 89.44% for model 2), showing how the addition of dropout contributed to performance improvement, indicating a reduction in overfitting and an enhancement in the model's generalization to new data.

Continuing with the second variation, the main change involves adding an additional convolutional layer. Specifically, in the second model, there are two convolutional layers after the first max-pooling layer. It can be observed that this increase in network complexity led to an improvement in validation accuracy, reaching 89.81%. This may have allowed the model to extract more sophisticated features, leading to greater discrimination capability during the training phase.

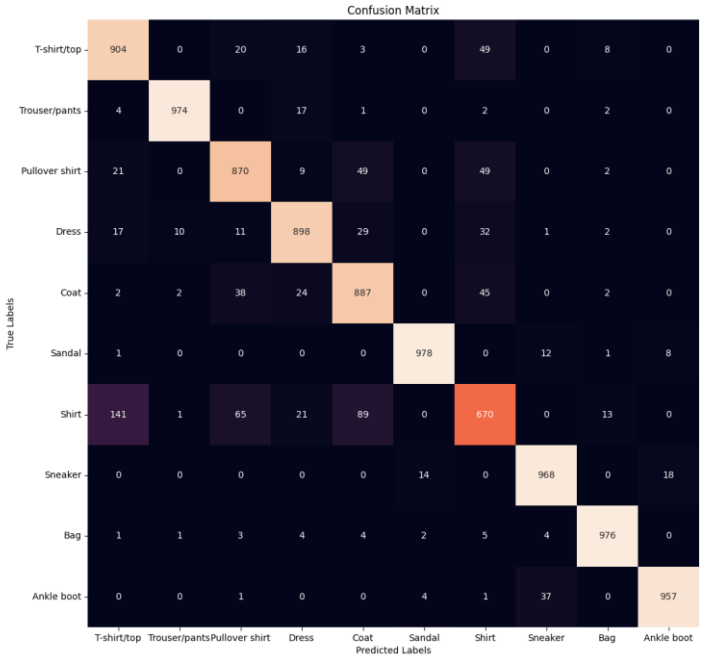
In the third variation, a reduction in kernel size from 5x5 to 3x3 was made. As evident, this allowed the network to capture finer details within the images by extracting relevant features, resulting in an improvement in validation accuracy, reaching 90.06%.

Finally, the last variation consists of introducing batch normalization in the convolutional layers. This technique has proven to be very effective, stabilizing the learning process and accelerating the convergence of the network. This translates into a significant increase in model performance, as evidenced by an accuracy of 91.26% during testing and 92.96% during training, further contributing to reducing overfitting, as indicated by the greater difference in accuracy.

## Discussion of models evaluated on the test set

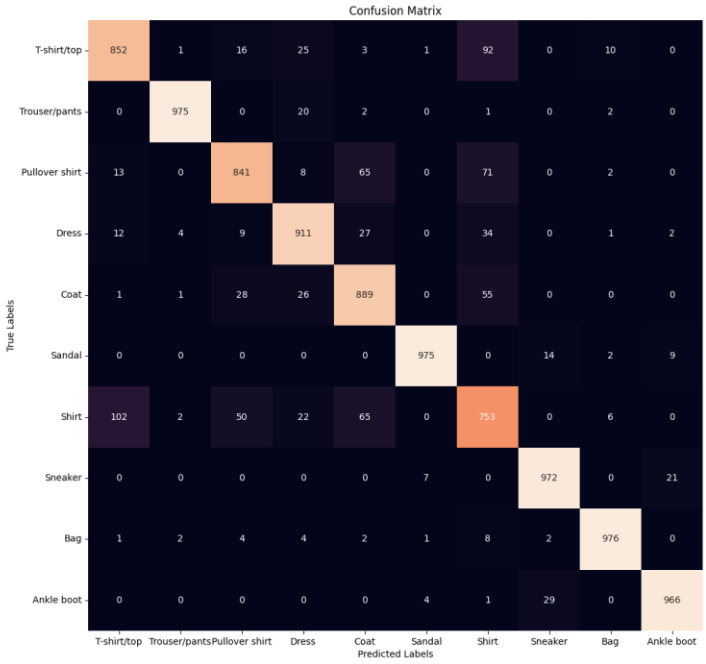
In the initial phase of this project, the structure of the developed neural networks was based on the Fashion MNIST dataset. Specifically, a split into training and validation sets was created from the training dataset. After various experiments and reaching an optimal structure, the final training phase was conducted, considering this time the combined training and validation dataset for training and testing on the test dataset, highlighting these results:

Test set metrics with model trained on train only:  Accuracy: 90.820000%, Avg loss: 1.553494



Train set metrics with model trained on train + validation: Accuracy: 92.820000%, Avg loss: 1.533038

Test set metrics with model trained on train + validation: Accuracy: 91.100000%, Avg loss: 1.550143



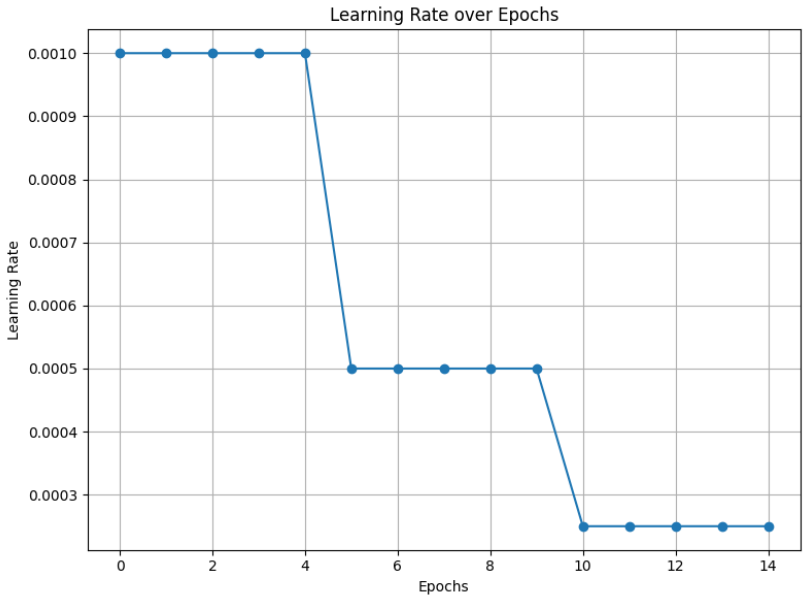

As observed, this difference is minimal. This occurs because the training dataset provided for Fashion MNIST exhibits a homogeneous distribution of images, allowing the network to have enough examples for each class to learn to generalize effectively. Indeed, when using only the training dataset, the results shown are already excellent, as it corresponds to 80% of the images designated for training, indicating how the network has learned to correctly classify images belonging to the remaining 20%. Clearly, what changes with the addition of the validation set during training is the increased number of input images per class, increasing the images used in the learning process while essentially maintaining the same distribution per class, slightly improving the performance of the network.

Also, when looking at the training performance of each model, we observe that there is quite a large difference between metrics of the train set and the test set. For the model trained only using the train set, the difference in accuracy is over 2% between the test set and the train set, while the difference is a bit smaller for the model trained using the train and validation sets (under 2%). This indicates a degree of overfitting.

**Choice tasks**

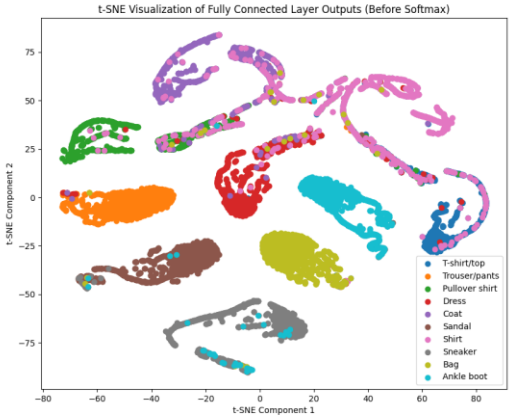
CHOICE 1: Decreasing Learning Rate:

For the development of the dynamic adjustment of the learning rate value the following procedure was adopted. For each call of the function, a check is performed on the corresponding epoch, allowing the modification of the parameter only every 5 of them ( lr = lr * (0.5 ** (epoch // 5)) ). Once this is verified and the next parameter is computed, the value is replaced within the process and the parameter is stored for subsequent representation. This function is called after each training iteration.
An example of the dynamic representation of the learning rate variation during a learning process is as follows:




CHOICE 3: Multi-Output Feedback:
To obtain more accurate feedback and a broader view during the learning process, three different output layers have been added within the neural network. These, within LeNet, are respectively positioned following the second convolutional layer ("self.output_conv1 = nn.Linear(16 * 5 * 5, 10)"), following the second pooling layer ("self.output_pool2 = nn.Linear(16 * 5 * 5, 10)"), and following the third convolutional layer ("self.output_conv3 = nn.Linear(120, 10)"). These allow us to visualize the accuracy level at different stages of the process, enabling a more detailed analysis of learning.

CHOICE 5: t-SNE visualization:
We use the t-SNE technique to visualize the last fully connected layer of LeNet5 5.0 when running the test set through it. We map the embeddings to a 2-dimensional space, and we observe that the visualization is consistent with the first confusion matrix shown above. Classes such as Bag and Ankle boot have relatively few points mistaken for other classes and have distinct clusters in the visualization. For other classes such as Shirt, which is the worst performing class in the confusion matrix, we see that there is great overlap between it and the classes Coat and T-shirt/top. This of course make sense these clothing items, especially with the low-resolution input we have, can be mistaken for each other more easily compared to other classes.



CHOICE 6: Fashion Product Images Dataset:

The implementation of this dataset during the testing phase following the training of the neural network on the fashion MNIST dataset first required a preprocessing phase.

First, it was necessary to make the output consistent with the training input. To do this, all images classified with labels not belonging to the MNIST training dataset were filtered out since the new dataset contains more than the 10 classes used in our base dataset. For this reason, the filtered labels did not have the same label name, so it was necessary to compare those that were recognized as the most similar and consistent categories with the training data to make the process efficient. Some classes were challenging such as the “Ankle boot” class which had no direct correspondence with the fashion product categories, so we had to pick the next closest category “Formal Shoes”. Subsequently, using the style.csv file which identifies all the images in the dataset, all product IDs were filtered, keeping those corresponding to the images that were considered correct, and the corresponding images were saved along with a new CSV file containing the product IDs with their respective labels but renamed according to the fashion MNIST criteria. However, before being saved, the images undergo processing that changes the background colour from white to black, and then a conversion to grayscale is performed to make the images consistent with the MNIST dataset.

The last necessary procedure occurred during the construction of the dataset since the labels as they were saved cannot be used in a neural network. For this reason, a mapping was carried out, assigning each label a value between 0 and 9 in the same way as the training dataset, in order to make the testing classification meaningful. Then, the process of loading this dataset via PyTorch was standard. A custom class was created, and the dataset was loaded by reading the pre-processed image and assigning the corresponding label. Following the training of our LeNet, the testing phase on the new dataset resulted in an accuracy level of 19.89% and an average loss of 2.254. The corresponding confusion matrix is the following: