Introduction

In this milestone, we focus on implementing and evaluating several deep learning models for the task of image classification using the Fashion-MNIST dataset. Our objective is to recognize various fashion items using **Multi-Layer Perceptron** (MLP), **Convolutional Neural Networks** (CNN), and **Transformer models**. Additionally, we explore the impact of **Principal Component Analysis** (PCA) on MLP performance. We assess the results based on accuracy, macro F1-score, and runtime analysis to find the best overall algorithm.

Methodology

For data preparation, we utilized the **Fashion-MNIST** dataset, which consists of 60,000 training images and 10,000 test images, each labeled with one of 10 categories of fashion items. This set is well known in machine learning and researchers often say: "If it doesn't work on MNIST, it won't work at all". The images were normalized to ensure consistent input features, and the dataset was split into training, validation, and test sets, with 20% of the training data allocated for validation.

Models Implementation

The Multi-Layer Perceptron was implemented with **two linear layers**, with a reduction of the data of two-third by the first layer and the second layer gives us an output an output of the number of classes. The output of the first layer first goes through a **rectified linear unit function** (ReLU) before being passed to the second layer.

The Convolutional Neural Network consists of **two convolutional layers** followed by **three fully connected layers**. The first convolutional layer processes the input channels and produces 6 output channels with a 3x3 kernel, stride of 1, and padding of 1. The second convolutional layer takes these 6 channels and produces 16 output channels with similar kernel, stride, and padding settings. The fully connected layers transform the data through dimensions 120, 84, and finally to the number of classes.
In the **forward propagation method**, the input tensor undergoes a series of transformations. Initially, the input passes through the first convolutional layer, followed by a ReLU activation and max-pooling, which halves the spatial dimensions. This process repeats with the second convolutional layer. The resulting tensor is then flattened to prepare it for the fully connected layers. The flattened tensor is sequentially passed through the first and second fully connected layers, each followed by a ReLU activation. Finally, the output layer generates logits representing the unnormalized scores for each class. These logits can then be used to predict the class of the input image.

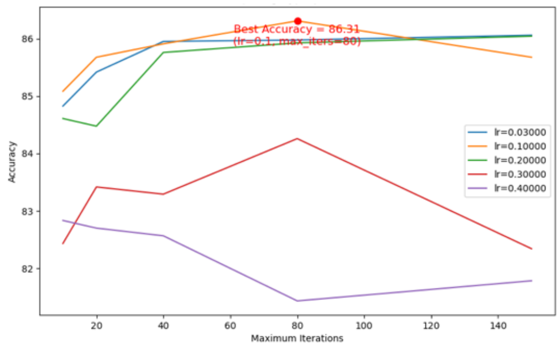
The Transformer model, implemented using attention mechanisms, was designed to capture long-range dependencies in the data. Key hyperparameters included the number of attention heads, layers, and learning rate.
The **initialization** method sets up the patch size, a linear mapper to transform patches to the hidden dimension, a learnable classification token, positional embeddings for spatial information, multiple transformer blocks and a final multi-layer perceptron for classification.
In the **forward** method, input images are divided into patches, which are then linearly mapped to the hidden dimension. A classification token is added to the sequence, followed by positional embeddings. The sequence is processed through a series of transformer blocks. Finally, the classification token is used to produce the output logits via an MLP.

Principal Component Analysis was applied for dimensionality reduction before feeding data into the MLP. This allowed us to evaluate the impact of PCA on model performance and training time. We've decided to take default value for the number of principal components which is a hundred as it seems to perform pretty good.

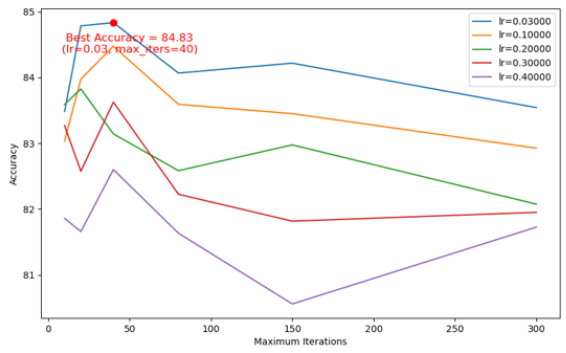
Results

The performance of the MLP, CNN, and Transformer models were evaluated using the accuracy. We've different hyperparameters to see how the model performs when tuning it.

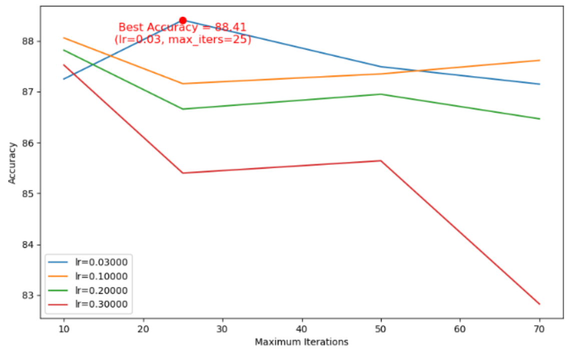
For the **MLP**, we found the best accuracy when taking a learning rate (LR) of 0.1 and a maximum of 80 iterations, giving a result of 86.31%.



Adding the **PCA** to MLP, the accuracy drops to 84.83% with a learning rate of 0.03 and 40 iterations max.



The **CNN** outperformed the MLP, achieving an accuracy of 88.41% by taking 0.1 as the learning rate and 25 iterations max.



The **Transformer** model performed the worst, achieving an accuracy of 74,86%, with a learning rate of 0.2 and 30 max Iter. We haven't done a graph for the transformer as it took to many times to run and isn't useful given the results we found.

To add up, looking at the results when training the model, we find accuracy that goes up to 100% and losses of 0. This means that our models sometimes overfit the training sample which will end up with a worst overall performance of the model.

Time

Analyzing time complexity gives us an important insight when choosing the best algorithm.

Task	MLP	MLP + PCA	CNN	Transformer
Fit time	52.3s	9.4s	108s	15'092s
Predict time	0.06s	0.02s	0.18s	43s

This demonstrate PCA's effectiveness in reducing computational complexity without significantly impacting performance. We can also see that transformer isn't really viable for our problem since it performs worst (for the parameters we tried) and takes a tremendous amount of time.

Conclusion

With the best hyperparameters we achieved to find, here are the results for each algorithm:

MLP	MLP + PCA	CNN	Transformer
86.31% F1-score = 0.866	84.83% F1-score = 0.847	88.41% F1-score = 0.879	74,86% F1-score = 0.739

One of the main challenges was tuning the hyperparameters, especially for the Transformer model. Balancing the trade-off between training time and model performance was crucial.
At the end, CNN performed the best in term of accuracy. But depending on the use case, if we need fast prediction and have limited amount of resource, MLP + PCA gives us a better overall performance with a fitting and predicting time more than 3 times faster.

We think there is still a lot of improvement that can be done concerning the architecture of our models, like changing the number of layer and the activation function that we apply.