Challenge 1
Annalisa Paladino

The dataset

In this first Challenge we will consider the ***FashionMNIST*** dataset, an *MNIST*-like dataset of grayscale images of fashion items, developed by Zalando Research in 2017. The dataset is composed of 784-dimensional images of size 28×28 pixels, it contains 60k training examples, and 10k testing examples. Each training example is accompanied with a respective label, which can either be:

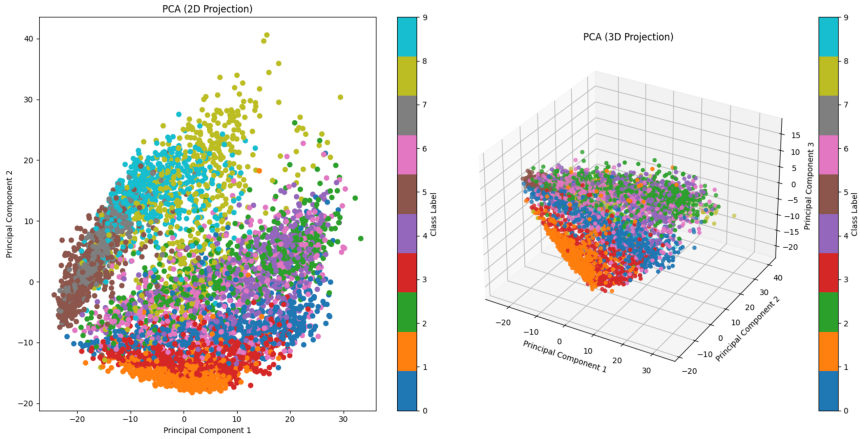
- 0 → T-shirt/top
- 1 → Trouser
- 2 → Pullover
- 3 → Dress
- 4 → Coat
- 5 → Sandal
- 6 → Shirt
- 7 → Sneaker
- 8 → Bag
- 9 → Ankle boot

For the following tasks, it's important to normalize the data with `StandardScaler()`. Moreover, since some of the following tasks can be memory- and time- demanding, I downsampled the dataset. The code containing the solution proposed for this Challenge is available on GitHub[1].

Section 1: Understanding data geometry

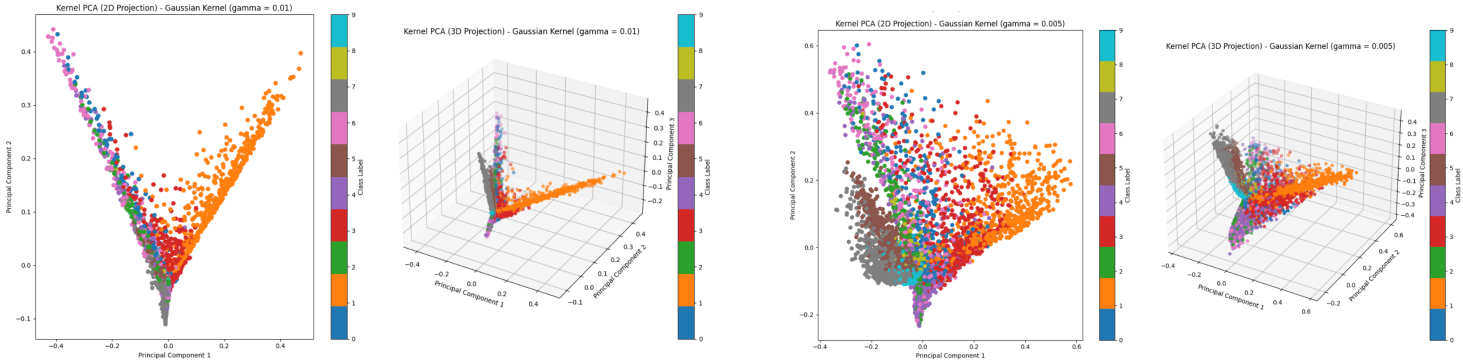
The goal of this first section is to develop a geometric understanding of the dataset, to do it we will perform the following steps:

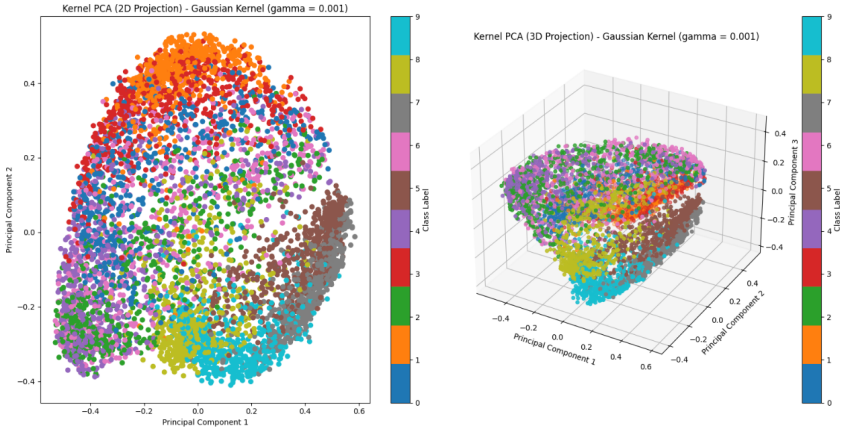
1. **(Linear) PCA:** PCA was performed by utilizing the first two and three principal components along with the true labels. From the plots below, it is evident that PCA successfully highlights distinct groups. Clusters corresponding to items such as trousers (orange), sandals (brown), ankle boots (light blue) and bags (light green) are clearly identifiable in these visualizations. Although certain clusters are discernible, there is noticeable overlap among some classes. This suggests that PCA alone may be insufficient, and the application of additional techniques should be considered.



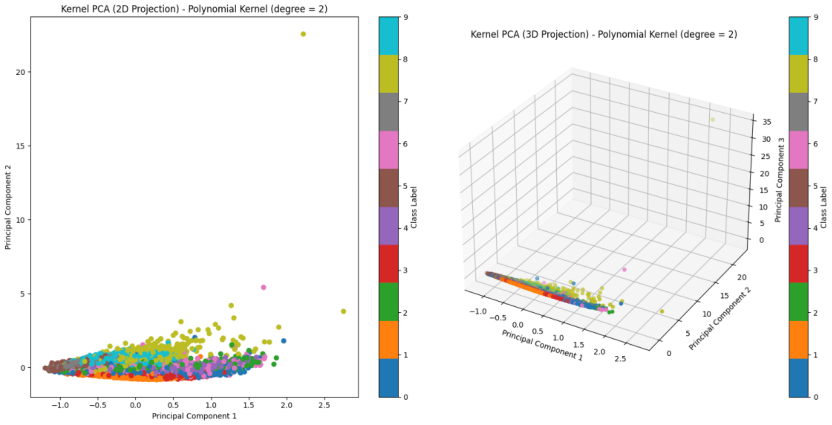
[1] https://github.com/annalisapaladino/AdvancedDeepLearning-KernelMethods/tree/main/Challenge%201

2. **Kernel-PCA with Gaussian Kernel:** Gaussian Kernel PCA was performed by utilizing the first two and three principal components along with the true labels. The plots below showcase only the most noteworthy results, while the complete set of experiments is available within the notebook, precisely the dispersion parameter were tuned by assigning to gamma the following values: 0.1, 0.05, 0.025, 0.01, 0.005, 0.0025, 0.001, 0.0005. It can be observed that higher gamma values result in suboptimal performance, whereas smaller values yield significantly better results. After testing multiple gamma values, the optimal value was determined to be 0.001. A clear distinction can be observed among certain classes: bags (light green), sandals (brown), sneakers (gray), trousers (orange), and ankle boots (light blue). Furthermore, an interesting observation is the proximity of sandals, sneakers, and ankle boots, all of which belong to the category of footwear, whereas trousers are positioned much farther away, reflecting their independence from footwear. Nonetheless, some overlap remains among the other classes.



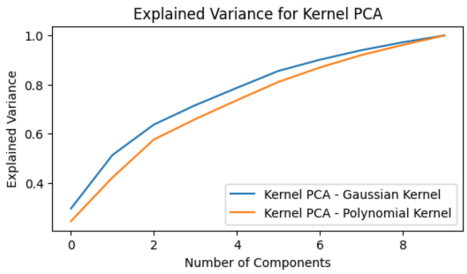


3. **Kernel-PCA with Polynomial Kernel:** Polynomial Kernel PCA was performed by utilizing the first two and three principal components along with the true labels. The degree of the polynomial kernel was tuned by testing values of 1, 2 and 3. The best results were achieved with a degree of 2, as higher degrees (equal to or greater than 3) did not yield any notable improvements.

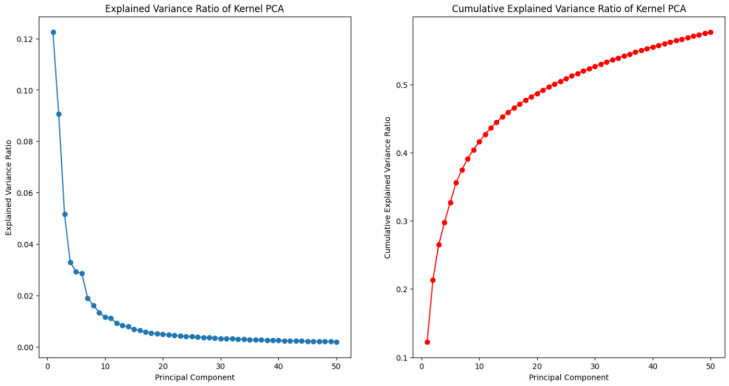


Section 2: Bridging *unsupervised* and *supervised*

To identify the best result in explaining data geometry obtained in the previous section, we refer to the **Explained Variance**. The plot displays the explained variance for Kernel PCA with both gaussian and polynomial kernels; however, the explained variance for standard PCA was also computed in the notebook but didn't have a superior performance compared to Kernel PCA. As you can see in the plot below, the best result found was the Gaussian Kernel with a value of gamma of 0.001.

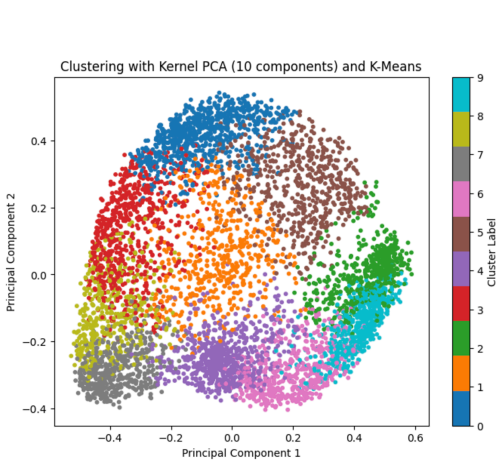


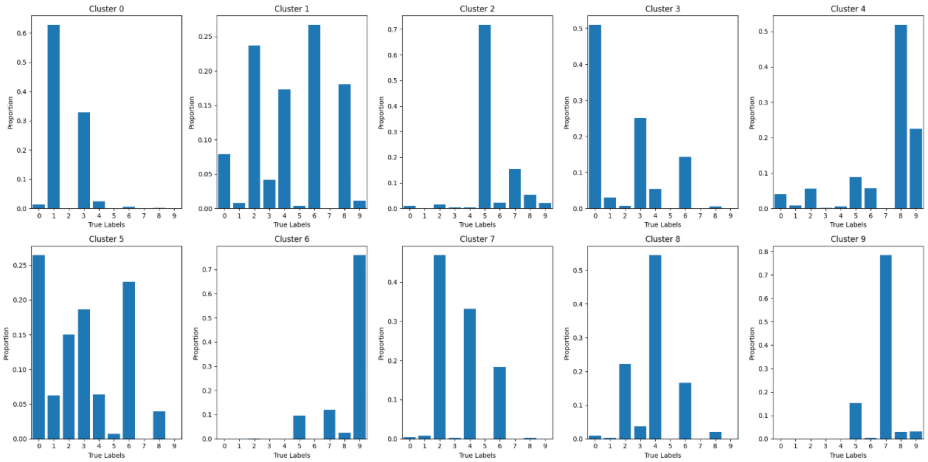
Now, considering the first 10 components of the kernel-PCA with Gaussian kernel and gamma=0.001, we can see that 10 components reflect the actual *knee* of the spectrum associated to the principal components



By considering only the first 10 components of the kernel-PCA to assign 10 labels to the resulting datapoints we considered two different approaches: KMeans and Agglomerative clustering.

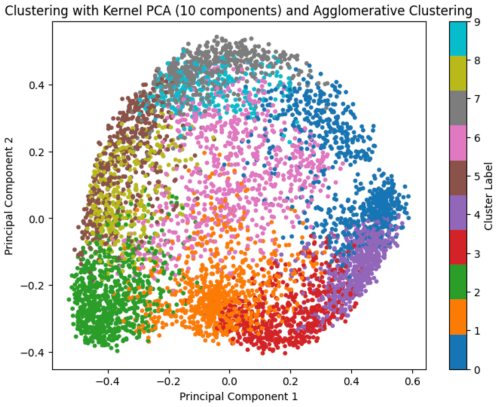
By applying **KMeans** clustering with 10 clusters, to see how well the label-assignment reflects the true labels, we can check the following plots:

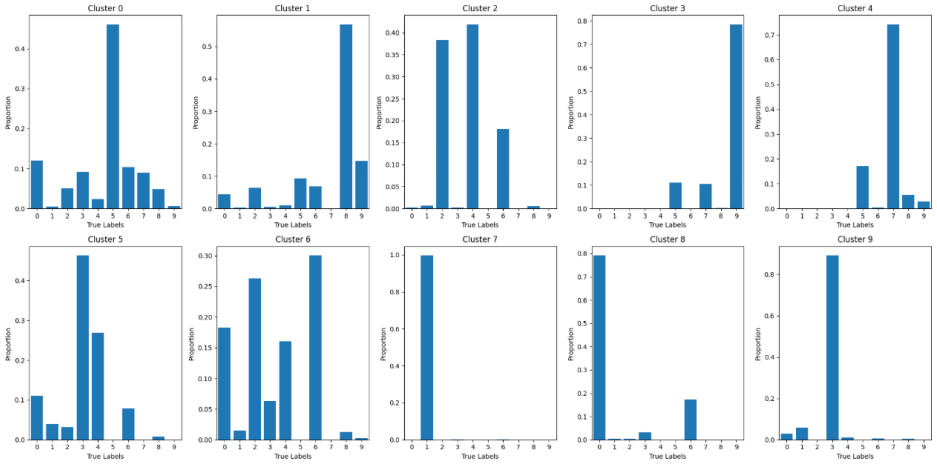




We can see from the label distribution in the cluster that there is a good separation between classes, but mixed distributions are present in some of them.

By applying **Agglomerative** clustering with 10 clusters we obtain the following results:





Again we can see that there is a good separation between classes, but some of them still have a mixed distribution.

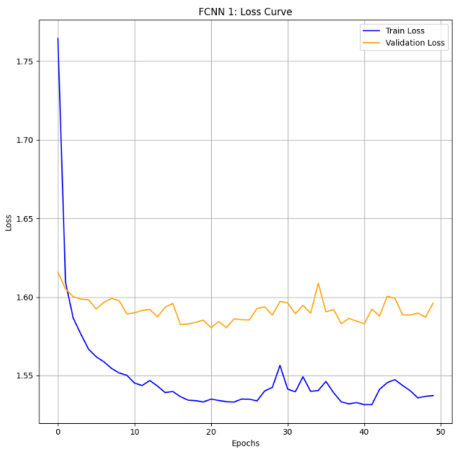
To determine which of the two approaches is more suitable, the *average intracluster variance* can be computed. The approach that minimizes this metric is considered optimal. Based on this criterion, agglomerative clustering is found to be the preferred method.

Section 3: (Supervised) classification

Now we have to consider the dataset composed of the original images, with the label assigned in the previous section by agglomerative clustering. To predict the label of a new image we need to define and learn a classifier on the data/label pairs, specifically: kernel-SVM, fully-connected NN and CNN.
The goal is to tune different configurations for each model and compare their performances, the best models will be selected based on the validation accuracy.

1. **Kernel-SVM:** For each kernel, a grid search was conducted over relevant hyperparameters
   - *RBF Kernel:* the values for gamma were selected basing on the same range used during Kernel PCA with Gaussian Kernel. The optimal model identified has a gamma value of 0.0025, achieving a validation accuracy of 0.94. This model also represents the best-performing configuration across all tested kernels.
   - *Polynomial Kernel:* this kernel was evaluated with combinations of degree, coef0, and a fixed gamma value ('scale'). The best model in this case has degree 3 and a value of coef0 of 1, achieving a validation accuracy of 0.93.
   - *Sigmoid Kernel:* this kernel tested combinations of gamma and coef0. The best model in this case has gamma=0.01, coef0=1, achieving a validation accuracy of 0.45. This kernel has the worst performance across all tested kernels.

2. **Fully-connected NN:** with FCNN it's possible to try different scenarios. Below you can find two different hyperparameter configurations
    - *FCNN 1:*

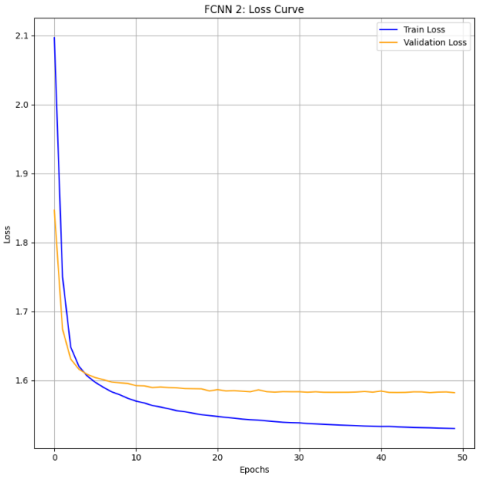


Hidden size: 128 neurons per layer (2 hidden layers)
Learning rate: 10^{-3}
Number of epochs: 50
Activation function: ReLU (except for the output layer: Softmax)
Batch size: 64
Optimizer: Adam
Criterion: Cross Entropy Loss

We reach a training accuracy of 0.93 and a validation accuracy of 0.84

From the plot it's possible to see how the training loss decreases initially but begins to fluctuate significantly after around 20 epochs, this indicates potential instability in learning, due to the fact that we used a smaller hidden size and a higher learning rate.
The validation loss remains relatively constant and does not decrease significantly over the epochs, this suggests that the model may not be generalizing well to the validation data. The lack of alignment between training and validation loss trends implies poor generalization.

    - *FCNN 2:*

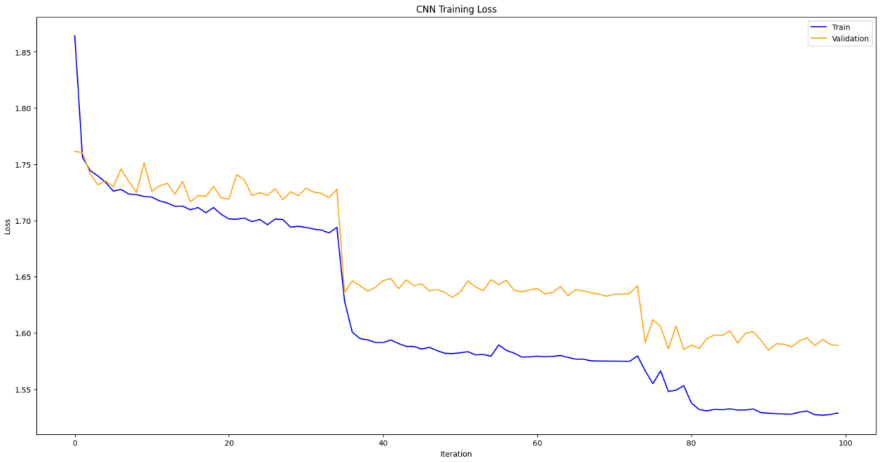


Hidden size: 256 neurons per layer (2 hidden layers)
Learning rate: 10^{-4}
Number of epochs: 50
Activation function: ReLU (except for the output layer: Softmax)
Batch size: 64
Optimizer: Adam
Criterion: Cross Entropy Loss

We reach a training accuracy of 0.93 and a validation accuracy of 0.89

The training loss decreases steadily and smoothly, showing consistent learning throughout the 50 epochs, this indicates a well-behaved training process.
The validation loss decreases significantly and stabilizes after around 10 epochs. The close alignment between training and validation loss suggests good generalization without overfitting. FCNN 2 demonstrates better overall performance, as the training and validation losses are both minimized effectively and show similar trends.

3. **CNN:**



Convolutional Block:

- First layer: Input channels = 1 (grayscale), Output channels = 32, Kernel size = 3x3
- Second layer: Input channels = 32, Output channels = 64, Kernel size = 3x3
- Each convolutional layer is followed by: ReLU and a MaxPooling layer (kernel size = 2)

FC Block:

After flattening the feature maps, two fully connected layers:

- First layer: Transforms the feature map into a 128-dimensional vector
- Second layer: Outputs a 10-dimensional vector corresponding to class probabilities

The final output is passed through a Softmax activation

Epochs: 100
Learning rate: 10^{-3}

We reach a training accuracy of 0.93 and a validation accuracy of 0.87

From the plot it's possible to see that there is a gap between training and validation loss could indicate slight overfitting or a limitation in the model's capacity to generalize perfectly to the validation set. The decreasing trend in loss suggests that the model is effectively converging.

Comparison:

The best-performing classifier identified in this study is the Kernel-SVM. Its superior performance can likely be attributed to the application of kernel PCA during the dimensionality reduction process. However, Kernel-SVM is not the most scalable approach, particularly for larger datasets, where Convolutional Neural Networks (CNNs) may be more appropriate.

Section 4: Wrap-up!

Now it's possible to evaluate the overall accuracy of the pipeline on the *test set* of *FashionMNIST* by comparing the predicted labels from the three classifiers built in *Section 3* with the true labels. As suggested by the assignment, the most abundant labels for each group of kernel-PCA-labelled datapoints have been used to assign a true label name to those determined from kernel-PCA. The overall accuracies obtained are:

	Accuracy
SVM	0.5995
FCNN 1	0.5830
FCNN 2	0.5986
CNN	0.6051

The results have a low accuracy, around 60% for all the techniques. The reason why we obtain such a low accuracy is that the labels obtained by the agglomerative clustering are not exactly the same as the true labels.

Section 5: A *fully-supervised* approach

Now I repeated the steps of *Section 3* using the true labels of the dataset for training (to prevent redundancy, I will omit repeating the steps, as they remain identical to those outlined in the preceding section). Those are the results:

	Hybrid pipeline accuracy	**Fully-supervised accuracy**
SVM	0.60	0.85
FCNN 1	0.58	0.85
FCNN 2	0.60	0.85
CNN	0.61	0.87

By comparing the results from the fully-supervised approach with those obtained through the hybrid pipeline, it is evident that the models trained using true labels achieve significantly higher accuracies than those derived from the hybrid pipeline. This observation suggests that the fully-supervised approach is more effective in understanding the relationships between features and labels.