**Image Classification of the Fashion MNIST dataset**

A machine learning project report

BSMALEA1KU

|  |  |
| --- | --- |
| Gabriel Cristian Circiu Maciej Pawel Jalocha  Wenzel Keil | gaci@itu.dk  macja@itu.dk  weke@itu.dk |

# **Introduction**

We are tasked with investigating the automatic classification of clothing product images to predict their respective type. The goal is to determine which type of clothing product any given image is, based on several models trained on already existing images of different kinds of clothes. While we can observe, with our human eyes, differences in the shapes and sizes of these images, for machines it is not as natural, and so it is imperative that we approach this task with not only a human perspective but that of a statistical one as well, that will aid us in programmatically evaluate both high- and low-level features. There are, however, obstacles. While we are given thousands of images to train the models on, this is not enough to achieve results that can always classify correctly, and with the rise of high-quality images, storing more than a few thousand images will hold challenges of its own.

### The Fashion MNIST dataset

The dataset we base our work on, is a subset of the of the 2017 project from Zalando Research [reference?], consisting of 70,000 images of different clothing items. Each image is a 28x28 grayscale image, associated with a label from 10 classes. In our project, we are using a subset of only 15,000 of the images, associated with a label from 5 classes. The dataset is then split up into 10,000 images for training and 5,000 for testing the correctness of our models.

Our initial approach to the task is to perform a thorough exploratory data analysis (EDA) to better understand the data that we are working with. We have observed that there is an even distribution of classes [reference?], the images are taken from the same angle, and they are all laid out the same way for their respective class. Using this information, we could already build a template of sorts, gathered from the averages of all the images within each class, to see if there are any outliers. [reference?] While at first it may not be evident, in the 5th class, known as “Shirt”, we found that the template created was not distinct looking, and after inspection of the images, we have found that these are clothing products that do not fit the first 4 classes, namely: T-shirt/top; Trouser; Pullover; Dress.

Throughout the analysis, we have observed some discrepancy with some of the aspects of the images. Notably, the contrast and brightness. We have corrected for the contrast issue by naively standardizing the image luminosities by subtracting the mean and dividing by the standard deviation of the image, which we concluded would not cause more issues as half the pixels were black. [reference?]

# **Methodology**

The next step was to extract features that can be used to train the machine learning models. The first step in extracting features was given as a requirement for our project, which is based on dimensionality reduction obtained from principal component analysis (PCA). This allows us to find the components that explain the most variance within the dataset, while disregarding the rest within the data. While we could set how many components we would like to obtain and use as our features, we also must account for how much of the variance can we capture with those components. We opted to go for 71 components as it explained ~91% of the variance. The reasoning behind the choice was not arbitrary, but instead it was chosen based on [explain the plot and show it]. This allows us to use a small fraction of the data to still get results that approach the ground truth. This is like the well-known [Pareto Principle](https://en.wikipedia.org/wiki/Pareto_principle).

Alongside that, we extracted other high-level features as well, such as the circumferences, widths, and heights of the four quadrants of an image, including their variances, averages, minimums and maximums. This resulted in a total of [number?] more features. Finally, we have generated a template from the dataset, as mentioned before using the average pixel luminosity values of the images, and then evaluating how closely each images matches the templates, measuring the Euclidian distances of the pixels from the template created. This custom feature extraction method gave us 5 more features to work with in the end, all adding up to a total of [number?] features. All features were stored in .csv files to be easily accessible for later.

### Development of machine learning models

With all features prepared, it was time to develop all the required and some optional machine learning models. We will categorize these into three groups, namely M1, M2, and M3. M1 is the implementation of two *Decision Tree* models, one from scratch, and one using the *Scikitlearn* library. M2 is the implementation of two *Feed Forward Neural Network* models, once again one done from scratch, and one using the *Tensorflow* library. Finally, M3 is the implementation of a *Random Forest* model using the *Scikitlearn* library, a *Convolutional Neural Network* using the *Tensorflow* library, and a custom designed model built from scratch which is derived from a template matching method. For all models we will be using the entirety of our extracted features data, as that will allow us to get the best results.

Before moving forward with developing and training our models, we split our saved feature data into both training and validation subsets, first by simply using the *train\_test\_split* method within the Scikitlearn library and then later switching it out for k-fold cross validation using the *GroupKFold* method from within the Scikitlearn library. Our dataset was split up to 80% training and 20% validation for both cases. This will help give us a good estimate on how well our models will perform overall once we turn to use the entire training dataset.

### M1: Decision Trees

Decision Trees are a supervised machine learning model that [short explanation?].

[Scratch implementation and correctness?]

As for the library implementation, this was done using the Scikitlearn library, and within it, the *DecisionTreeClassifier* method. For the hyperparameters, we used *max\_depth=50*, which means that we can have only 50 branches away from the root of our tree. Going higher resulted in overfitting.

### M2: Feed Forward Neural networks

[short explanation?]

[Scratch implementation and correctness?]

As for the library implementation, we used the *Tensorflow* library, which allowed us to easily build a neural network with minimal effort. To set up the model, we set the batch size to 32, and epochs to 10. As for the network itself settled with 3 hidden layers, alongside the input and output layers. The hidden layers used 32 neurons and the ReLU activation function, while the output layer used the Softmax activation function. For our optimization, we used the Stochastic Gradient Descent with a learning rate of 0.01.

### M3.1: Random Forest

The Random Forest classifier model was implemented using the Scikitlearn library, and within it, the *RandomForestClassifier* method, which is an ensemble method, meaning it can utilize multiple Decision Tree models and combine them to reach a more accurate result. For the hyperparameters, we used *n\_estimators=100*, which means we get an ensemble of a hundred trees. While higher estimators give better results, it comes with diminishing results and the computation required significantly increases.

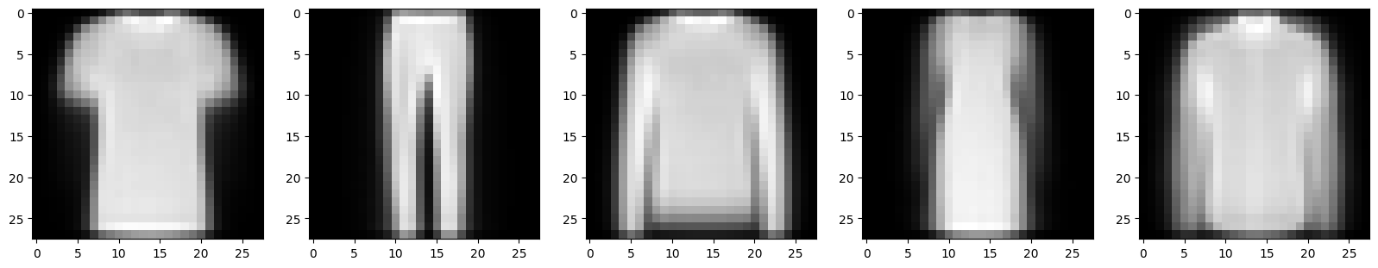
### M3.2: Convolutional Neural Network

[Explanation and library implementation description?]

### M3.3: Template Matching

One of the methods evaluated was a template matching approach, where a representative “standard” template is derived for each category and then used for comparison against new images.

Since the original input images were not grayscale, and differences in color intensity could skew the data (e.g., jeans appearing primarily blue and thus having lower average pixel values), the first step was to standardize each image. This was done by subtracting the mean and dividing by the standard deviation of the 28x28 brightness values in each image, ensuring that each image had a mean of zero and a standard deviation of one. Such normalization helps equalize the contribution of all features and reduces the risk of scale-related biases or noise. Below are the two templates before and after standardizing the data.

A close-up of a person's body

Description automatically generated

With the data standardized, the training set was split according to its respective categories. For each category, a template was created by averaging pixel values at each position in a 28x28 grid across all images belonging to that category. This process yielded five distinct templates—one per category—to be used during inference.

To classify a test image, we measured how closely it resembled each category’s template. Specifically, we subtracted a given template from the test image and computed the Euclidean distance. The expectation was that the correct template would produce the smallest distance, effectively “nullifying” the values when compared with the corresponding category’s pattern. The category associated with the template that yielded the lowest score was then assigned to the test image.

To ensure the correctness of our implementation, we performed several checks. First, we verified that each standardized image had a mean of zero and a standard deviation of one by calculating and reviewing summary statistics. We also inspected a small sample of templates visually to confirm that the averaged pixel values produced a coherent representation of the corresponding category.

We used scikit-learns shuffled k-fold split to partition our training-data into 5 subsets as recommended in our lecture on the [[find lecture]], taking 4 of these as our training set and the remaining one as our validation set repeating the process 5 times so that each subset was once the validation set. To control our results, we relied on scikit-learn library’s reporting methods to get a confusion matrix for each k-fold split and reporting the mean confusion matrix at the end with the respective variance.

Before we standardized the data our testing results looked as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Category 1 | Category 2 | Category 3 | Category 4 | Category 5 |
| Precision | 0.722 | 0.8921 | 0.6404 | 0.7547 | 0.4524 |
| Std. | 0.0251 | 0.0143 | 0.02 | 0.0148 | 0.0394 |
|  |  |  |  |  |  |
| Recall | 0.7255 | 0.8979 | 0.6501 | 0.8346 | 0.3915 |
| Std. | 0.0235 | 0.0147 | 0.0225 | 0.0212 | 0.0074 |
|  |  |  |  |  |  |
| F1 | 0.7235 | 0.8949 | 0.6446 | 0.7923 | 0.4188 |
| Std. | 0.0203 | 0.0129 | 0.0087 | 0.0047 | 0.016 |

And an accuracy of 0.6984 with a standard deviation of 0,0076

After standardizing our data, the general scores for the validation set improved significantly:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Category 1 | Category 2 | Category 3 | Category 4 | Category 5 |
| Precision | 0.7459 | 0.9838 | 0.7402 | 0.7909 | 0.5874 |
| Std. | 0.0274 | 0.0076 | 0.0204 | 0.0116 | 0.0228 |
|  |  |  |  |  |  |
| Recall | 0.8109 | 0.9051 | 0.7957 | 0.8826 | 0.4699 |
| Std. | 0.0222 | 0.0113 | 0.0149 | 0.0182 | 0.0204 |
|  |  |  |  |  |  |
| F1 | 0.7768 | 0.9427 | 0.7669 | 0.8342 | 0.5221 |
| Std. | 0.0222 | 0.0044 | 0.0165 | 0.0125 | 0.0209 |

With an accuracy of 0.7721 with a standard deviation of 0.0102

Giving us personally better results as we have initially expected and an overall improvement over the initial approach without standardization.

Since the template matching method is based purely on averaging pixel intensities for each category and directly computing Euclidean distances to classify test images, it does not introduce any tunable hyperparameters such as learning rates or regularization parameters. The choice of template size (28x28) is derived directly from the input images and thus is fixed

Additionally, these distance scores were stored as high-level features for use in other classification methods.

# **Results**

Once we implemented and trained all our models on the extracted features, we documented the results and some interesting observations that

This section presents the results of the project.

* Discuss results of all methods, and all the results, most importantly accuracy and auc.
* Loss/Cost function results over epochs
* Compare what do the results show and what they mean, which one we foun to be most important
* Random forest good, but neural network even better
* Explain the difference our feature extractions made
* Explain the difference of using PCA or other features

Required: Interpretation and discussion of the results Your report should include a thorough discussion of the performance of each of the methods applied. In particular, you should compare the methods’ performance and guide the reader in interpreting the results. Use your expert knowledge to explain the results; for instance, why do particular methods perform better than others?

## Results template matching

Mean confusion matrix

[[799. 4. 21. 102. 74.]

[ 30. 885. 17. 59. 9.]

[ 15. 1. 769. 14. 201.]

[ 43. 5. 3. 860. 89.]

[211. 2. 257. 88. 442.]]

Accuracy 0.751,

Precision: [0.7276867, 0.98662207, 0.72071228, 0.76580588, 0.54233129]

Recall: [0.799, 0.885, 0.769, 0.86, 0.442]

F1: [0.76167779, 0.93305219, 0.74407354, 0.81017428, 0.48705234]

# **Discussions**

This section discusses the implications of the results.

* What do the results mean?
* Are our models good?
* Are there any metrics that show areas that could be improved?
* Would these models be qualified to be used in the real world?
* PCA helps reduce computation required
* Extracting high level features could also help

# **Conclusion**

While all classification results seem to be promising, we have observed that Neural Networks provide the best results, and the Random Forest classifier also does exceptionally well, considering its simplicity of expanding upon the Decision Tree classifier.

Moving forward, we would like to explore the entire Fashion MNIST dataset, and not only refine our existing models for even better classification, but also implement new ones that could tackle much larger problems in the future, or even develop generative models to acquire new insights in the world of machine learning.