# Image Classification on Zalando Fashion-MNIST dataset

Anthony Palmieri
anthony.palmieri@studenti.unipd.it

## 1. Introduction

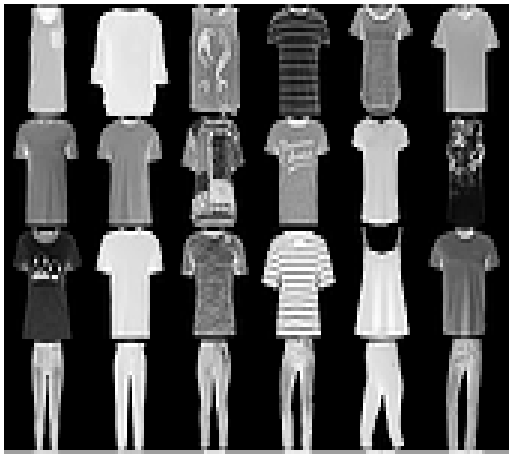
With this report and related notebook we focused on a multi-class image classification problem, in particular the one proposed by the Zalando Fashion-Mnist dataset.
Object classification is a fairly easy task for us, but it's been proved to be quite complex for machines.
Classifying images consists of associating to each image a specific class, in our case we had to choose among 10 classes.
The objective is to find a classification algorithm capable to reach a reasonable accuracy level.
In our case, using *Support Vector Machine* (after dimensionality reduction via *PCA*) we were able to obtain a 99.88% accuracy on training set, 89.96% accuracy on validation set, and 89.57% accuracy on test set.
We also experimented with different algorithms such as: *Decision Tree, Random Forests, k-Nearest-Neighbours, Multinomial Logistic Regression* and *Neural Networks*, but none of these performed as good as SVM.
To implement SVM though, we had to reduce the number of features beforehand. In fact, the size of the dataset turned out to be quite problematic, leading us to experiment with some feature reduction techniques such as *PCA* and *HOG*[1].
Prior to feature reduction Neural Networks resulted in the best model, achieving an accuracy score of 88.4% on test set.
In this report we decided to analyze both of these models, we'll start by focusing on the Neural Network approach and then move on to the SVM, introducing dimensionality reduction along the way.

## 2. The Dataset

The object of study is the Fashion-Mnist dataset released by Zalando Research.
Fashion-Mnist is a dataset of Zalando's article images, consisting of a training set of 60000 examples and a test set of 10000 examples. Each example is a $28 \times 28$ grayscale image, to which a label from 0 to 9 is associated.

[1]HOG is not properly a dimensionality reduction technique, although it can reduce the number of features when dealing with images, that's why we used it.

In fact, we have 10 classes: t-shirt, top, trousers, pullover, dress, coat, shirt, sneaker, bag and ankle boot.
Below, a representative image of the dataset.



Before starting working on the dataset it was necessary to flatten the images in order to transform them from $28 \times 28$ matrices to numpy arrays of length $28 \cdot 28$. We also further split the training set so to have 48000 instances for training and the remaining 12000 instances for validation.
In particular the training set consists of a $48000 \times 784$ matrix, very large dimensions which heavily weighted on the performances of some of the models we tried.

## 3. Neural Networks approach

First of all we checked for classes unbalance.
Turns out the number of instances in each class is about 4800, so there's no need to implement undersampling or oversampling procedures, such as *SMOTE* for instance.
Moreover, being the classes perfectly balanced and equally important, we decided to use accuracy as a mean to measure the performances of the deployed algorithms.
Recall that this would have been wrong if classes were highly unbalanced.
For some of the proposed algorithms it was also necessary to scale the features, given they were too sparse.

We tried with both a *MinMaxScaler* and a *StandardScaler*, obtaining better results with the latter.
Feature scaling was particularly crucial for Neural Networks, model with which we obtained the best results before feature reduction.
For constructing the model we relied on the Sequential and Dense libraries provided by Keras. We implemented the function MLP_definer to carry out a GridSearch over the number of hidden layers of the networks, and number of neurons per hidden layer, achieving the best results with a network consisting of:

- input layer with 784 neurons and activation function *tanh*
- hidden layer with 100 neurons and activation function *tanh*
- output layer with 10 neurons and activation function *softmax*

(Slightly worse results can also be achieved using *Relu* instead of *tanh*, consider though that *Relu* is computationally more efficient.)

In particular, using softmax as activation function on the output layer allows us to have a probabilistic interpretation of the results.
We used the *Categorical Cross Entropy* as loss function and *mini-batch Stochastic Gradient Descent* as optimiser, with batch size equals to 16.
Using Cross Entropy as loss function is quite a reasonable choice, considering this same function arises naturally when trying to maximize the likelihood function of the model. Nonetheless we also tried with other loss functions, which can be found here, obtaining worse scores though.
In choosing the optimizer we opted for a mini-batch Stochastic Gradient Descent, which is essentially a middle way between batch Gradient Descent and Stochastic Gradient Descent: a size $N$ random sample is extracted from the training set, and on this random sample batch Gradient Descent is applied, iterating all over again until the convergence of the algorithm.
We also implemented an *Early Stopping* mechanism, with Patience equals to 5, monitoring the validation accuracy.
The model was able to reach a 96.45% accuracy on training set, 89.36% accuracy on validation set, and 88.4% accuracy on test set.
Even though the model is built onto 30 epochs, experimenting with even greater number of epochs we see that validation accuracy keeps swinging around the same thresholds, so much so that the Early stopping mechanism arrests the fitting after barely 17 epochs.
Below, the graphs for Accuracy and Loss on training and validation set, without Early Stopping.

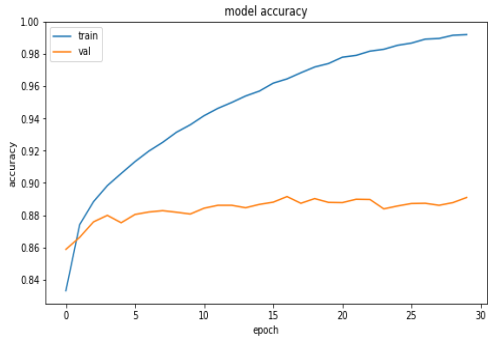



Figure 1: Model Accuracy on Training and Validation sets

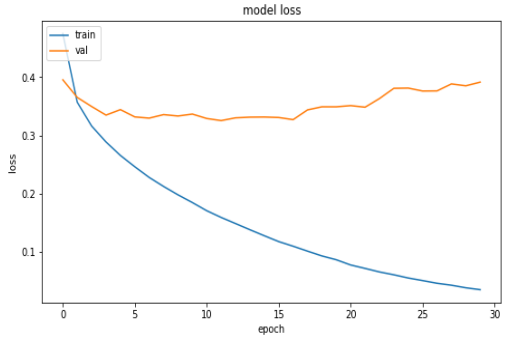



Figure 2: Model Loss on Training and Validation sets

We also observe that while validation accuracy keeps bouncing around the same values, validation loss on the contrary starts increasing after a while.
Introducing the Early Stopping was actually a good choice.

## 4. Dimensionality Reduction and SVM

We tried to tackle the problem also with several other algorithms, such as: Decision Tree, Random Forest, k-Nearest-Neighbours, Support Vector Machine and Multinomial Logistic Regression.
For each one of these models we implemented a GridSearch over an appropriate grid of parameters.
The size of the dataset was particularly limiting, considering some of these GridSearch took more than half an hour to be carried out.
Support Vector Machine and Multinomial Logistic Regression in particular turned out to be basically impracticable[2] on the original dataset, due to its excessive size.
These considerations led us towards dimensionality reduction techniques, such as PCA and HOG.
PCA carries out a dimensionality reduction via Singular Value Decomposition of the training set matrix.

[2]Here we mean that the run time was getting very long, and not knowing when or if it would have finished, we discarded the models.

HOG on the other hand, is a feature descriptor used in computer vision and Image Processing for object detection purposes. We used it as a mean of dimensionality reduction.
Reducing the number of features allowed us to successfully implement even Support Vector Machine and Multinomial Logistic Regression.
In particular, using PCA, Support Vector Machine was able to achieve an accuracy score of 89.57% on test set, the best we could observe.
To define the classifier we used a *regularization parameter* $C$ equals to 10, and Radial Basis Function as *Kernel function*.
The regularization parameter $C$ controls underfitting and overfitting of the model: small values of C push the model towards underfitting whereas large values of C lead the model towards overfitting.
Radial Basis Function Kernel instead, is one of the most common kernel functions among kernelized machine learning algorithms, with kernel matrix given by

$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right).$$

(Recall that such Kernel can also be interpreted as similarity measure between samples.)
The model was able to reach an accuracy of 99.88% on training set, 89.96% on validation set and 89.57% on test set.
Below, the confusion Matrix and also some of the misclassified examples.

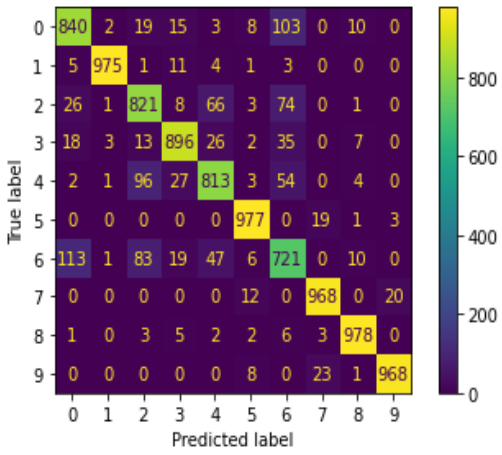



Figure 3: Confusion Matrix of the Model

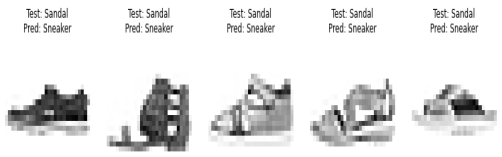



Figure 4: Sandals predicted as Sneakers

Actually, these errors are quite questionable. Some of these sandals I would have said were sneakers. These are errors we can tolerate.
Also note that the most problematic classes are the 0th one and the 6th one, corresponding respectively to T-shirt/Top and Shirts. Even in this case though, some of the errors are quite questionable.

## 5. Conclusions

Ultimately we couldn't do any better, we couldn't surpass 89.57% accuracy on test set, even though using Convolutional Neural Networks one can easily achieve a 93%.
We also note that the dimensionality reduction techniques we carried out, weren't effectively able to make any major improvements.
In this sense, instead of reducing the number of features (which is not that excessive), it would actually be interesting to see how resampling strategies could affect the performances of the deployed algorithms, however this is something we won't discuss.

## References

[1] Christopher M. Bishop (2006) *Pattern Recognition And Machine Learning*, Springer.