Machine Learning Report - FASHION MNIST CHALLENGE

Darko Ivanovski 1243085
darko.ivanovski@studenti.unipd.it

Davide Ghiotto 1236660
davide.ghiotto.2@studenti.unipd.it

1. Introduction

This is the report for the Machine Learning Project about the Fashion MNIST Challenge [2]. The objective of the challenge is to correctly classify images, from Zalando's article images, between one out of ten classes, using machine learning methods and techniques studied in the Machine Learning Course. It is, essentially, a "Multi-class classification" problem.
It is an important task because this dataset is presented as a new standard for benchmarking machine learning algorithms (the claim is that the original MNIST dataset with handwritten digits was too easy for modern algorithms). Solving this problem will be a good starting point to dive deep inside the image classification world.
In this report we present how we achieved an accuracy of 0.9276 using a Convolutional Neural Network, starting with a simple basic structure first and adding complexity until we could make sense of it. Moreover, we will discuss about alternative methods which offer different trade-offs about training/testing time, model complexity and overall performances. All of our work is available on kaggle [1].

2. Dataset

Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 *grayscale* image, associated with a label from 10 classes, representing the type of clothing that the image depicts.
We further split the training set in two parts: a *training set* (80%) and a *validation set* (20%). In this way we were able to take a glance of our performances on a set different from the training one.
A single example of this dataset is stored as a continuous sequence of pixels, resulting in an array of 784 (28x28) values between 0 and 255.

2.1. Preprocessing

We applied two different types of preprocessing to the data:

- **feature scaling**: we normalized the input data to avoid preferring a particular set of features over the whole input. This step was important especially for the KNN and SVM algorithms that calculate the distance between points.
- **reshaping**: CNNs expect a specific input structure to work properly. So, we reshaped the original input consisting of an array of 784 values per image, in a matrix of dimensions 28x28x1 (where the last dimension is the *depth* of the image). It's important to specify that we did this reshaping only for the CNN.

3. Method

We choose a Convolutional Neural Network (CNN) as our leading machine learning model. We think it's the right approach since CNNs are good models when it comes to image classification. In fact, CNNs use relatively little pre-processing compared to other image classification algorithms and in this way they take advantage of the hypothesis on the input structure. In addition, we have a lot of cleaned and well-organized training data that can train the network to reach good accuracy.
We implemented two versions of CNN:

- **Single Convolutional Layer**: simple version with only the basic structure, consisting of different layers:
  - convolutional: with 64 filters as the first core layer;
  - pooling: we use *max-pooling* of size 2x2 to reduce the number of parameters passed to the *fully connected layer* and to control overfitting;
  - fully connected: with 256 neurons;
  - loss: we use a *softmax* loss layer to finally predict a single class of K mutually exclusive classes.

  In addition to these layers, we added some regularization methods, such as: *dropout layer* and *lr-regularization* to avoid overfitting. We trained this network on 20 epochs.
- **Double Convolutional Layer**: the simple CNN was not capable of capturing the real complexity of the problem, so we tried a little more powerful model: this

network has 2 convolutional layer (the first with 64 filters, and the second one with 128 filters) each followed by a *max-pooling* layer (2x2). Then, after the second convolutional layer, we added a *dropout* layer (p=0.35). And, at the end, we maintained the fully connected layer with 256 neurons and the *softmax* loss layer. So, we boosted only the convolutional aspect of the network, and the result was an improvement on the accuracy by over 1%. We trained this network on 30 epochs.

The Figure 1 represents the accuracies of the single-layer and the double-layer networks. The Figure 2, instead, plots together the loss value for the relative networks.

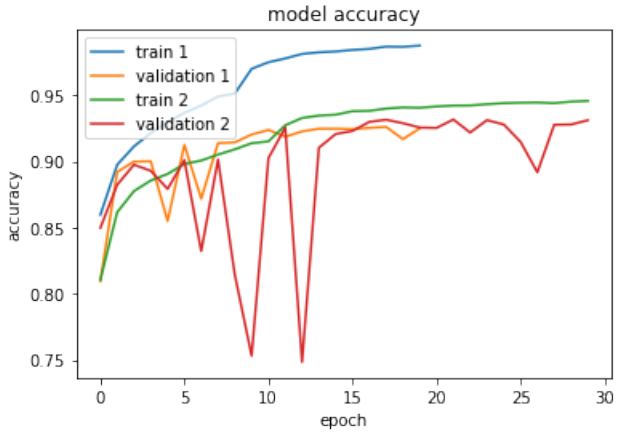

Figure 1. Comparison on the accuracy of the two CNNs

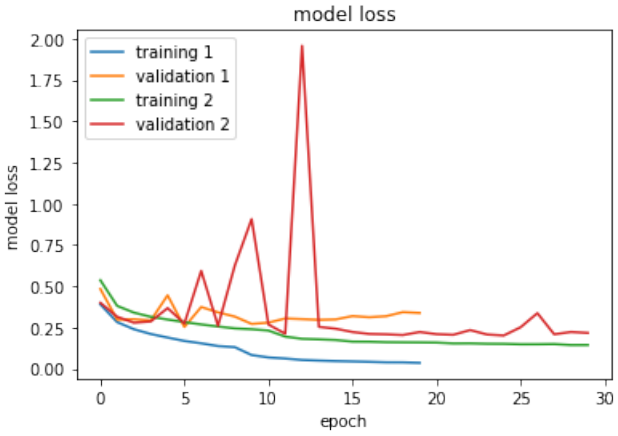

Figure 2. Comparison on the loss of the two CNNs

As we can see from the graphs, the first network tends towards a more overfitted solution. We can spot the occurrence of this phenomenon noticing that the accuracy of the training set keeps raising as data is provided, and the loss gradually tends towards zero. However, the validation set does not behave in the same way: the difference between the loss of the training and the validation becomes larger and larger as we increase the number of epochs.
Instead, the double convolutional layer network has an higher loss value for the training set, but the loss for the validation set is lower and really close to the training one. This is clear sign that adding a convolutional layer, in our case, reduces the overfitting problem.

3.1. Parameters choice

We now want to explain the reasons behind the parameters that we choose, such as the number of filters in the convolutional layers.
Our main focus was to improve the accuracy of the model with just one convolutional layer in order to better understand the role of the parameters, and then applying our discoveries to add another convolutional layer. So, we first tried to use a very different number of filters in the convolutional layer, ranging from 16 filters to 1024. We noticed that we achieved the best result with 64 filters, and that adding or subtracting filters worsens the accuracy.
Then, we maintained fixed the 64 filters and tried to adjust the number of neurons in the fully connected layer. As before, we tried to use numbers in the range 16-1024, finding that the best result is given with 256 neurons.
Once we have set the number of filters and the number of neurons for our two core layers, we tried to adjust the dropout rate to have the best possible result. The dropout was added after the fully connected layer, and we obtained the best result with a rate of $p = 0.35$.
Finally, we added the *lr-regularization*: we reduced the LR by a factor of 0.2 after a plateau of 3 epochs without improvement. We noticed that this regularization has a very little impact in the accuracy of the model, but still improves the results by a little bit.

In the end, the best accuracy reachable by a CNN with just the basic structure was 0.9158. To improve the results, as we said, we added a convolutional layer. To do this, we maintained fixed all of the parameters above, which gave the best possible results, and tried to adjust the number of filters of this second convolutional layer. Our best result was obtained choosing 128 filters, halfway between the 64 filters of the first layer and the 256 neurons of the fully connected one.
Moreover, we added another dropout with a rate of $p = 0.35$ after the two convolutional layers. In this way, our model has a dropout after the convolutional layers and a dropout after the fully connected layer.
Given the increased complexity of this model to the previous one, we trained this model over 30 epochs versus the 20 epochs of the simple one.

In this way, we obtained our final results with an accuracy of 0.9276.

Beside this main model, we tested other alternatives approaches, as explained in the next section.

4. Experiments

As mentioned, we experimented with several methods to compare the results with the CNN. For these algorithms we use the dataset without the reshaping process, only with the normalize data, in order to feed the algorithms the right way. Here we present a list of the methods, ordered by increasing accuracy, with their best version based on hyperparameters optimizations tests:

- **Logistic Regression (LG)**: it's a fast algorithm that can train and test in very little time, but the accuracy was low due to the limitation of this model when it comes to a large amount of features.

- **K-Nearest Neighbors (KNN)**: we reached the highest accuracy with K=10 and *minkowski* distance (p=2) (euclidean distance). However, it did not performed so well. Our intuition was that "similar images should be near each other" , but the number of features was so big (28*28) that we may occur in the "curse of dimensionality" problem, where distances mean less and less, because every training point is "far away" from the other.

- **Random Forest (RF)**: we tested this simple model to compare its accuracy with the results of more complex methods as CNNs or SVMs. Training and testing time were very low and the accuracy was relatively high based on the simple model that the RF uses.

- **Support Vector Machine (SVM)**: we tried the SVM because generates a model that it is easier to interpret than a CNN. We tested the SVM with the *gaussian* and the *polynomial* kernels. The former brought to a more overfitted solution compared to the latter, where we achieved the best accuracy with a fifth degree polynomial.

- **Inspired CNN**: we wanted to compare our best model with others. This dataset is very famous and it was easy to find already implemented CNNs on the web, so we borrowed the model from [3] and tested it. We reached the highest accuracy but the complexity of the model was also very high. In fact, we noticed that adding a bunch of layers improves the accuracy, but we were not able to explain in a reasonable way why that works. Our main goal was to build and optimize a network that we understand in every aspect, not only to obtain the highest possible accuracy adding complex features with no clue of how they work.

The accuracies of the discussed methods are summarized in the table 1.

Method	Training time	Testing time	Accuracy
Logistic Regression	37 s	0 s	0.8468
KNN	-	788s	0.8531
Random Forest	99s	1s	0.8802
SVM	637s	226s	0.9011
CNN 1C	140s	5s	0.9158
CNN 2C	223s	5s	0.9276
Inspired CNN	781s	5s	0.9414

Table 1. Best accuracy achieved with different methods

References

[1] Davide Ghiotto and Darko Ivanovski. Kaggle notebook. https://www.kaggle.com/davideghiotto/best-algo/.

[2] Zalando Research. Fashion-mnist. https://github.com/zalandoresearch/fashion-mnist.

[3] Aditya Vartak. Kaggle notebook. https://www.kaggle.com/adityav5/cnn-on-fashion-mnist-approx-95-test-accuracy.