Image Classification: Fashion-MNIST (Zalando)

Alvise Dei Rossi
alvise.deirossi@studenti.unipd.it

1. Introduction

The task of image recognition (i.e. classifying which object is shown in an image) is a core task in computer vision, as it makes possible a large number of downstream applications (tagging automatically pictures, assisting visually impaired people and much more) and has also become the go-to task on which to test machine learning classifiers. For many years the MNIST handwirtten digit classification dataset has been by far the most used dataset to benchmark these algorithms. Nowadays, even though it's still widely used, MNIST is considered too easy, in fact an accuracy of over 99% can easily be achieved. In many cases an algorithm's performance on it doesn't really transfer to real computer vision tasks; simply put, if an algorithm works on MNIST, it may still totally fail on other more complex datasets. For this reason nowadays it's more common to test image recognition algorithms on the Fashion-MNIST Zalando dataset which has the same accessibility of MNIST, sharing many aspect of it, such as training/test sets sizes and number of features, but appears to be a more challenging alternative for benchmarking. In this project different models are evaluated on the Fashion-MNIST dataset, looking to obtain the best possible result. Both performance and time-consumption to train and tune the model are taken into consideration in order to find the best candidate for the task. In particular the impact of the optimization of various hyper-parameters for a MLPClassifier is explored. An accuracy of over 88% on the test set is achieved by doing so.

2. Dataset

The Fashion-MNIST dataset is made up of a selection of Zalando's articles images. Just like the MNIST dataset it consists of a training set of 60,000 instances and a test set of 10,000 instances. Every instance is 28 pixels in height and 28 pixels in width, for a total of 784 pixels. Each pixel has a value ranging from 0 to 255 associated indicating the lightness of that pixel. To every example is also associated a target integer between 0 and 9, corresponding to a class label as shown by Table 1. Both the training and the test set are pretty much balanced, no class is overrepresented or underrepresented in either of those, thus accuracy is used to evaluate models' performance most of the times. For the purpose of this project all the images are flattened to 784-dimensional vectors; min-max normalization of the features is also performed.

Label	Description
0	T-shirt/Top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandals
6	Shirt
7	Sneaker
8	Bag
9	Ankle boots

Table 1. Class labels in Fashion-MNIST dataset.

In order to evaluate all the classifiers appropriately, the training set is split in two parts: 50000 instances are used to actually train the model, while the second part (the validation set, composed of 10000 instances) is used to evaluate it on and optimizing it, tuning hyper-parameters.

3. Procedure

To obtain a good model, initially some classifiers from different categories (e.g. linear, SVM, decision tree etc.) are trained using mostly default parameters and a subset of the data available. This is done to quickly shortlist the most promising ones. Other than the performance on the validation set, the time required to train and evaluate the models are also kept into account as some models tend to badly scale to larger datasets, hence posing a limitation once many different hyper-parameters configurations have to be tested. Finally, when the most promising model is selected, the parameters are tuned to try to improve performance, avoiding overfitting or high bias. We can understand how to do so by plotting the accuracy and/or loss and observing how they vary as we change the parameters one at the time. Another possibility is to try to find the best possible model by launching a grid search over many parameters at once. In the case of multiclass classification tasks, when the targets

are integers, as it is in the Fashion-MNIST dataset, the loss of the model is calculated by using the sparse categorical crossentropy loss formula (1):

$$J(\theta) = -\frac{1}{M} * \sum_{i=1}^{M} log(P_{model}[y_i \in C_{y_i}]) \qquad (1)$$

where M is the number of instances for which the loss is calculated and $P_{model}[y_i \in C_{y_i}]$ is the probability given by the model for the correct label for that instance. Checking how the loss function changes for the training and validation sets as we train the model may allow us to spot when we're overfitting the model hence failing to generalize to new data or when we're in a high bias configuration.

Important hyper-parameters that usually have to be considered in order to optimize the model are the learning rate and the regularization term; beside them, depending on the model chosen, many parameters may have an impact on the overall performance: in the case of NNs it's good practice to test the model for different choices of the dropout rate, the activation function and even trying different architectures of the network. The number of epochs the model is trained for can also be considered as a value to optimize in the case early stopping isn't directly implemented. Finally, when the model has been fine-tuned correctly, error analysis is performed to try to understand in which cases the model fails to recognise differences between classes and possibly to figure out ways to improve the training process in future models.

4. Experiments

To quickly shortlist models that can be appropriate to be used for the task, some tests were performed using different algorithms. This was done using default parameters for the models and a subset of 10000 instances of the data to speed-up the process since in this step of the process all that is needed is to get a rough idea for what to focus on later. The results obtained are reported in Table 2. Clearly the SGDClassifier and the Decision Tree algorithms are underfitting the data, in particular the latter. The KNN model had a slightly better performance but the long evaluating time and the fact that other models seemed to be both more

Classifier	Score	Train Time	Eval Time
SGDClassifier	0.8094	16.56s	0.03s
Logistic Regr.	0.8302	28.23s	0.03s
KNN	0.8221	5.44s	189.28s
Decision Tree	0.7482	5.42s	0.03s
SVM Classifier	0.8300	25.16s	43.62s
MLPClassifier	0.8519	26.51s	0.29s

Table 2. Models' performances.

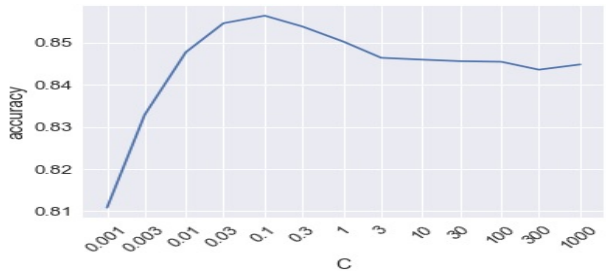

Figure 1. Log. regression: accuracy vs different values of C param.

time-efficient and accurate ruled it out as an option too. The best result was obtained for the simple 1-hidden layer neural network reaching a score of 0.8519. But, since both the logistic regression and the SVM classifiers gave good results too, they were further explored as options. Many logistic regression models were subsequently trained using k-fold cross validation for different values of the C regularization term parameter (ranging from 0.001 to 1000), this time on the entire training set. The results obtained are illustrated in Figure 1: the model reached the peak cross-validation accuracy of 0.85614 for a C value of 0.1. A SVM classifier was then trained on the entire training set aswell, using a linear kernel and achieving an accuracy of 0.8515, close to the best logistic regression model and non-optimazed neural network performances. Due to the nature of the problem though, this classifier was very slow to train and evaluate: in fact the SVM algorithm doesn't scale well to large datasets and, in the case of multiclass classification tasks such as this one, the implementation used automatically splits the problem in many binary classification problems (one vs one reduction), training 45 binary classifiers in order to make predictions. Further improvement for SVM may certainly be possible, for example choosing other kernels, but it was decided that optimizing the hyper-parameters of a neural network could give the best results in far less time.

Consequently, to begin with this process, three types of NN architectures were tested: in Figure 2 are plotted the

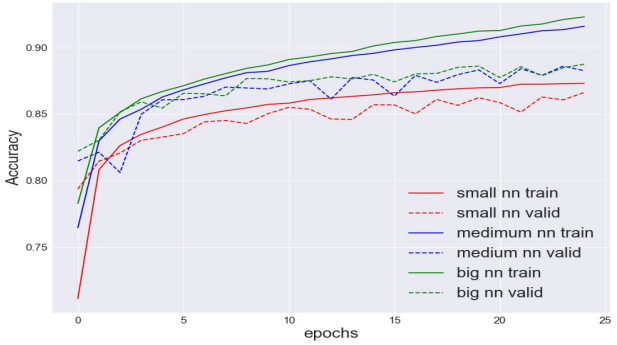

Figure 2. Difference of performances between NN architectures.

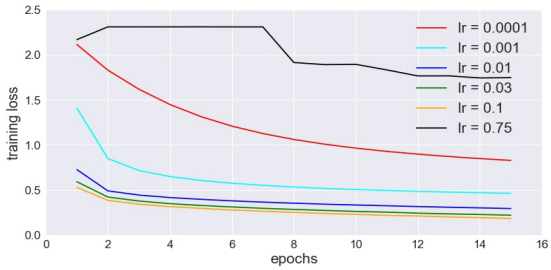

Figure 3. Effect of different learning rates.

learning curves for these models. In all the models the input layer consisted of a layer of 784 units, corresponding to the number of pixels, and an output layer of 10 units, corresponding to the number of classes. As activation function the relu function was used, while to obtain the probabilities of the classes in the final layer the softmax function was used. The smallest architecture consisted in a model with a single hidden layer with 16 units and clearly underfitted the data, lagging behind the performance of the two bigger models, which reached a validation accuracy of about 0.885. Between these last two, the "medium" network (with 2-hidden layers made up of 512 units) seemed to be a better choice as it achieved about the same score (and loss) of the bigger net (2-hidden layers, 2048 units each), in way less time and being slightly less prone to overfit the data.

Subsequently a brief study of the effect of the learning rate, shown in Figure 3, led to the choice of a value of 0.03. Using the same procedure, appropriate values were also chosen for the regularization term (0.0002) and the dropout rate (0.3) since previously the model was clearly overfitting; the importance of the impact of these parameters can easily be noticed in Figure 4. While the model is still slightly overfitting, the effect is reduced and is further suppressed by choosing to stop training after 30 epochs, as validation loss won't decrease more. A brief comparison of the usage of the sigmoid function instead of relu was also made, but the sigmoid function led only to a slower to learn and less effective model, hence was discarded. Finally, having selected the best hyper-parameters, the model was trained on the entire training set and tested on the test set, achieving an accuracy of 0.8812.

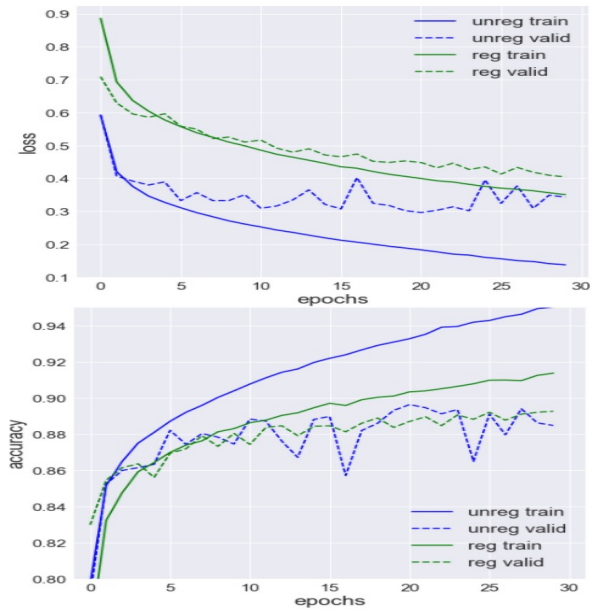

Figure 4. comparison before/after regularization and dropout.

To gain insight about where improvements could be made, a brief study of the confusion matrix was performed. As shown in Table 3, the model is having trouble classifying correctly shirts: these are in fact often miss-labeled as t-shirts, pullovers and coats and vice-versa. Some examples of mislabeled shirts can be seen in Figure 5. It's easy to see how the model missclassifies some of these instances, as sometimes they indeed look alike to other classes, and most likely a human classifier would mislabel some of them aswell. An idea to improve performance in future models could be to feed more shirt examples to the model that look like other classes but aren't, so to help it to distinguish between them.

Label	Precision	Recall	F1 Score	Support
0	0.84	0.87	0.85	1023
1	0.99	0.98	0.98	988
2	0.84	0.78	0.81	1008
3	0.91	0.91	0.91	1021
4	0.72	0.91	0.81	1050
5	0.97	0.97	0.97	996
6	0.79	0.60	0.68	970
7	0.95	0.95	0.95	955
8	0.97	0.97	0.97	968
9	0.96	0.97	0.97	1021
accuracy			0.89	10000

Table 3. Classification report.

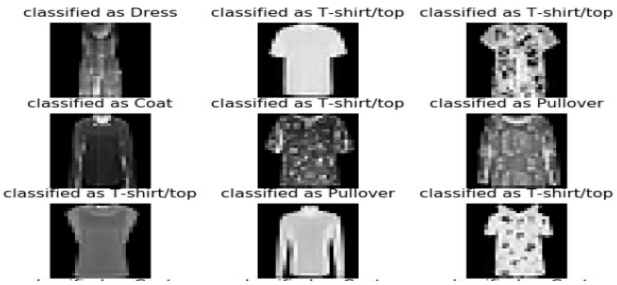

Figure 5. Examples of mislabeled shirt instances.