Supervised Learning with Fashion MNIST

Clara de Mattos Szwarcman*

Pontifícia Universidade Católica do Rio de Janeiro
clara_szw@hotmail.com

Ian Albuquerque Raymundo da Silva†

Pontifícia Universidade Católica do Rio de Janeiro
ian.albuquerque.silva@gmail.com

December 13, 2017

I. Introduction

The MNIST dataset is considered the "hello world" of Machine Learning. Current results achieve over 99.7% accuracy using different techniques [12] [5] [10] [3] [8]. Also, many good techniques do not work well on this dataset because of its simplicity [4]. With that in mind, the Fashion MNIST dataset [13] was created. It has same size and format of the MNIST dataset meaning that it should be as simple to use, but its images consist of fashion items which are supposedly harder to classify.

The goal of this work is to explore different techniques of supervised learning for classifying the images in the Fashion MNIST dataset. We want to measure the accuracy of our tests and compare them with different results on the same dataset, while also trying to obtain some intuition behind the data.

II. The Dataset

The Fashion MNIST dataset is composed by 70.000 grayscale (0-255) images of size 28x28 labeled into the following 10 different classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle boot. Figure 1 is an example of one of the images in the dataset.

For the implementation we used Python and Scikit-Learn [2] and TensorFlow [1] libraries. The input for the algorithms are the pixels values.

Figure 1: *Example of an image in the dataset*

III. Preparation

The Fashion MNIST is already subdivided into two sets. A set of 60.000 images (the training set) and a set of 10.000 images (the test set). The idea is to learn with the training set and then evaluate the performance of our trained model with the test set.

We will be using the standard accuracy as a

*Clara's student ID: 1310351
†Ian's student ID: 1310451

Algorithm Name	Best Accuracy (%)
SVC	89.7
Gradient Boosting	88.8
Random Forest	87.9
MLP	87.7
K-Neighbors	86.0
Logistic Regression	84.0
SGD	82.9
Decision Tree	80.1

Table 1: *Fashion MNIST Scikit-Learn Benchmark [9]*

metric for our tests:

$$accuracy = \frac{\#(correct\_classifications)}{\#(test\_set\_size)} \quad (1)$$

The dataset is already provided as a list of (28*28)+1 sized vectors corresponding to each of the 28*28 pixels of the image plus 1 entry for the classification. Since each pixel value was an integer from 0 to 255, we have opted to divide each value by 255 to achieve features that goes from 0 to 1.

IV. Known Results

The Fashion MNIST repository contains some benchmarks using the python library scikit-learn [9]. They are not meant to be efficient or good, but they work as a baseline for our project. The best result (corresponding to the best selection of parameters tested) of each algorithm are displayed in table 1.

The current state of art for the Fashion MNIST dataset is a recent one, with 96.3% accuracy using Wide Residual Networks using random erasing data augmentation [14] and other image pre-processing techniques. We will consider this as the ceiling accuracy for our project.

V. Approaches

i. Random Classifier

To test our framework, we implemented a random classifier that simply guesses one of the 10 classes. The accuracy was 10% as expected.

ii. K-Nearest-Neighbors (KNN)

ii.1 Own implementation

Our first attempt at the problem was implementing the K-Nearest-Neighbors algorithm ourselves. We tried using the Euclidean distance as the metric and K = 50 as the parameters for the algorithm. For that, we obtained an accuracy of 75.80%, quite far from the scikit-learn benchmark.

This algorithm was pretty fast to implement. Since its training consists in only saving the training set, it is pretty fast to train. However, it does take a lot of time to classify each example.

ii.2 Distance from Mean

Inspired by the KNN algorithm, we implemented a "distance from mean" algorithm in which for the training we take the mean of each group of instances of the same class and for the testing we simply classify as the nearest of the 10 means obtained in the training part.

This is basically a worse version of the KNN and the results were worse accordingly. We obtained an accuracy of 67.68%. The benefits were the fact that it was easy to implement, fast to train and fast to classify.

ii.3 Scikit-Learn Implementation

We also tested for the Scikit-Learn implementation for the KNN algorithm. This time, we tested for K = 5 neighbors using the Manhattan distance. Our obtained accuracy was 86.23%, the exact accuracy from the benchmark, as expected.

ii.4 Scikit-Learn w/ Haar Transform Pre-Processing

For the next test, we tried to pre-process the dataset by changing the feature space. We maintained the parameters K = 5 and Manhattan distance, but also used the Haar transform to change the feature space. The idea was to use features that described the frequencies in the image, allowing a more global description of the image. However, it did not achieve good

results, resulting in an accuracy of 84.0%, lower than the algorithm without the Haar Transform pre-processing.

iii. Single Layer Perceptron

iii.1 Standard Implementation

We implemented ourselves a single layer perceptron and trained it with only 1 iteration over the dataset. We did not invest a lot of effort in this approach because it relies on the assumption that the dataset is linearly separable for the algorithm to converge, which clearly is not the case for the dataset. We obtained an accuracy of 75.8% for this approach

iii.2 MIRA (Margin-infused relaxed algorithm)

We made some changes to the update step of the algorithm so it took into account how much each error should affect the weights of the perceptron, using the Margin-infused relaxed algorithm technique. We obtained slightly better result, with an accuracy of 76.99%.

iv. SVM (Scikit-Learn Implementation)

iv.1 Polynomial Kernel

For the SVM we used the Scikit-Learn implementation. We first tested with a 3rd degree polynomial kernel and with a penalty parameter C = 10, since they were the parameters used for the benchmark. We obtained an accuracy of 87.23%, which is slightly different from the accuracy described in the benchmark (89,7%).

iv.2 Polynomial Kernel w/ Pre-Processing

We tried to improve the results by pre-processing the dataset by downscaling the images to 7x7, normalizing the dataset by feature (pixel) and also adding to the algorithm input rotations of the original dataset from -10 to 10 degrees, varying degree by degree. With that technique, we obtained an accuracy of 84.91%, which was worse than without the pre-processing. We suspect that down sampling the dataset to 7x7 images caused too much information loss.

Because of that, we changed the down sampling to 14x14 and also changed the rotation interval to the interval -5 to 5 degrees. The new accuracy was 89.39%, an improvement of about 2% compared to the example without the pre-processing.

It is also important to note that the pre-processing heavily improved the accuracy of the algorithm when using a small subset of the original dataset. For instance, for 100 instances, the accuracy was improved from 43.69% to 64.21% and for 1000 instances it was improved from 59.10% to 78.56%. It is probably due to the fact that adding the rotation benefits a lot small datasets that do not contain enough data for the generalization necessary for good results.

iv.3 Gaussian Kernel

We also tested for a gaussian kernel, with parameter C = 10^6 and gamma = $4 \cdot 10^{-7}$ since it achieved good results from previous works. As expected, we obtained better accuracy, achieving 90.13% for those parameters.

iv.4 Gaussian Kernel w/ Rotations and Canny Filterl

By adding rotations from -5 to 5 degrees, without down sampling, and also applying the canny filter (an edge detection filter) for the dataset, we achieved a result of 80.36%. Even though we thought that applying the canny filter was a good idea to improve the significance features of the image, it clearly was not the case.

v. Convolutional Neural Networks

The Convolutional Neural Networks (CNNs) are the current state of the art for the Fashion-MNIST dataset with 96.3% accuracy [14]. To be able to train faster and with larger networks,

we used the GPU NVIDIA GTX 980M with CUDA.

The first CNN that we used was based on the 1994s LeNet5 [6], one of the first CNNs, that has a 99.05% accuracy for the MNIST dataset [7]. We used the same layers and kernel sizes but with more features on each convolutional layer. 2 shows the architecture used for the first CNN test.

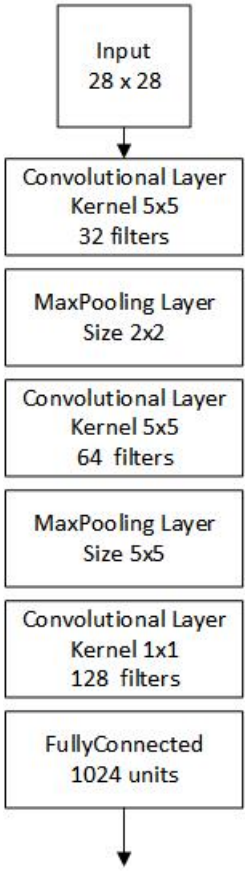

Figure 2: *LeNet5 based Architecture*

We trained the network for 200 epochs and got an accuracy of 89.02%, then, to increase the examples in the training set, we added rotations from -5 to 5 degrees, degree by degree, and got 89.81%.

Seeking to improve our accuracy, we used a more complex and deeper CNN as base, the Very Deep Convolutional Networks [11] used in the ImageNet Challenge. Since this is a deep CNN and made for 224x224 RGB images, we adapted to a smaller network with a receptive field of 28x28 and fewer units on the fully connected layer. 3 shows the architecture of the new CNN.

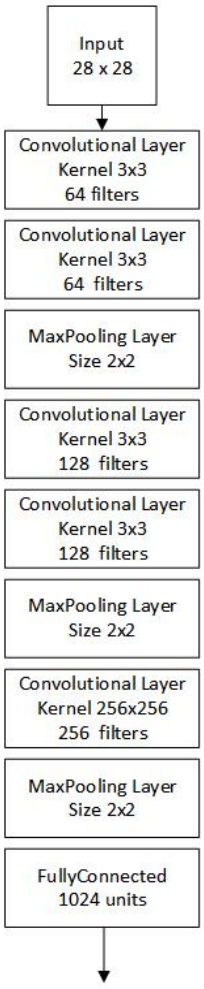

Figure 3: *VGG based Architecture*

The accuracy obtained from training the new network for 300 epochs was 90.06%, and for 938 epochs, 91.27%. When we tried to increase the number of epochs again, the accuracy dropped slightly, so we went to another approach. 4 shows the value of the loss rate given the number of steps.

Our next approach was to use Random Erasing as data augmentation, since it was used in

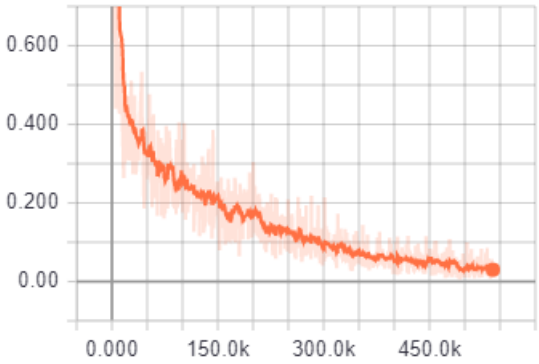

Figure 4: *Loss Rate for VGG based network*

the state-of-the-art implementation. Random Erasing randomly selects a rectangle region in an image and erases its pixels with random values, which reduces the risk of over-fitting. For 300 epochs the accuracy was .

VI. Conclusions

After running all the different algorithms we could realize the importance of a powerfull resource for processing, in our case the GPU. We also have seen the importance of choosing good pre-processing techniques, that can either boost or decrease the quality of the test. Also, we have witnessed the efficiency of the Convolutional Neural Networks for a task of this kind, because they got the higher results sometimes even in less time.

Our results were above the ones in the benchmark, but still lower than the state of art. The state of art used a more complex network, with more pre-processing, which is what we believe that was the cause for the difference between our accuracy.

Our next steps to improve the accuracy of the result are to add more data augmentation combined with the Random Erasing (i.e flipping, rotation and cropping) and to implement and test different networks with TensorFlow, such as the ResNet, which is similar to the CNN used in the state of the art implementation. We are also extending the currents implementations to more complex data sets like CIFAR-10 and CIFAR-100, that are datasets with 32x32 RGB images with respectively 10 and 100 different labeled classes.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. *CoRR*, abs/1309.0238, 2013.

[3] Jia-Ren Chang and Yong-Sheng Chen. Batch-normalized maxout network in network. *CoRR*, abs/1511.02583, 2015.

[4] François Chollet. Many good ideas will not work well on mnist (e.g. batch norm). inversely many bad ideas may work on mnist and no transfer to real cv. Tweet, April 2017.

[5] Jurgen Schmidhuber Dan Ciresan, Ueli Meier. Multi-column deep neural

networks for image classification. *International Conference of Pattern Recognition*, page 3642–3649, February 2012.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[7] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[8] C.-Y. Lee, P. W. Gallagher, and Z. Tu. Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree. *ArXiv e-prints*, September 2015.

[9] Zalando Research. Fashion mnist scikit-learn benchmark. http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/.

[10] Ikuro Sato, Hiroki Nishimura, and Kensuke Yokoi. APAC: augmented pattern classification with neural networks. *CoRR*, abs/1505.03229, 2015.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[12] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proc. International Conference on Machine learning (ICML'13)*, 2013.

[13] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[14] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *CoRR*, abs/1708.04896, 2017.