TRIPLET NETWORK FOR FEW SHOT LEARNING

Daniel Teitelman, Gonen Weiss
Technion, Israel Institute of Technology
Haifa
{daniel.tei, gonen.weiss}@campus.technion.ac.il

ABSTRACT

In this project, we implemented the paper "Deep Metric Learning Using Triplet Network" by Elad Hoffer et al [1]. Using the method demonstrated in the paper we learned a linearly separated embedding of the "Fashion-MNIST" dataset [2] and evaluated the predictive power of the embedded data using linear classifiers. Afterward, we looked at the few-shot learning setting using the Triplet network, and evaluated its predictive power with respect to the number of shots (N-shots) and the number of classes (K-ways). Our implementation is publicly available in GitHub [1].

***K*eywords** Few shot learning · Triplet network · Metric learning · Fashion-MNIST

1 Introduction

In recent years, deep learning models have achieved state-of-the-art performance for many different tasks such as: image classification [3, 4], image segmentation [5], text generation [6], audio generation [7] and many more. One of the underlying assumptions is that deep, hierarchical models such as Convolutional Neural Networks (CNNs) create useful representations of data [8, 9], which later can be used to distinguish between available classes. During this project we are going to show how a simple Triplet network will allow us to learn a metric of the data, therefore allowing the use of linear classifiers. Moreover, the learned metric we allow to use the deep learning model as a pre-processing tool for the few-shot learning scenario. In the next sections, we will explore the Fashion-MNIST dataset used during this project, define metric learning, the triplet network, and introduce the few-shot learning setting.

1.1 The Fashion-MNIST 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. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits (definition taken from the Fashion-MNIST GitHub). We choose the Fashion-MNIST because the original paper Elad Hoffer et al. wrote used the following datasets in his paper: "MNIST, Cifar10, SVHN, STL10", therefore we wanted to use it on an unevaluated dataset. Another reason for the choice of the Fashion-MNIST came from the fact we wanted a harder dataset to evaluate the method on, but at the same time easy and fast to use. Both of these requirements are answered by this dataset as can be seen in the GitHub profile comparing Fashion-MNIST to other datasets. An example of the data present in the dataset can be seen in the figure 1.

1.2 Metric learning

Distance metric learning is a branch of machine learning that aims to learn distances from the data, which enhances the performance of similarity-based algorithms. The problem setting falls into two main categories depending on the type of supervision and available training data: supervised learning and weakly supervised learning. Some examples include: Mahalanobis distance, information-theoretic metric learning [10] , Sparse high dimensional metric learning, [11]. But

[1] https://github.com/danielt17/Triplet-network-few-shot-learning

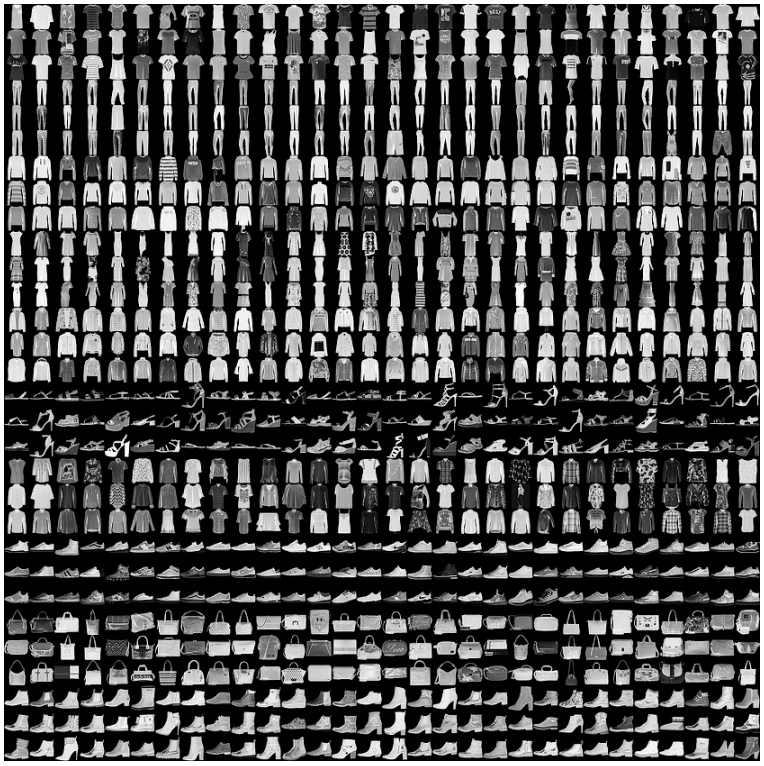

Figure 1: An example of how the data looks, each class takes three-rows

more recently deep learning techniques for metric learning called deep metric learning became popular including the following techniques: Siamese network [12], triplet network [1] and CosFace [13]. In the next section, we will describe the triplet network architecture more precisely as it is the model we worked with during this project.

1.3 Few shot learning

Few shot learning (FSL) is a type of machine learning problem where the training dataset contains limited information [14]. Where most ML models achieve high accuracy the more data we have, our motivation is achieving high accuracy while using a small amount of data of some given classes. To continue our work we need to introduce some definitions, which will allow us to evaluate the model over standard few-shot learning metrics. Instead of training a classifier using examples and labels in the classical setting, we do the following procedure: first, we learn a metric by one of the methods described above. Secondly, we create a set of examples our model was not trained on called a support set. A support set is defined by two parameters called: K-way-N-shot, Where each task includes K classes with N examples of each. Thirdly, we have a set of examples we want to predict their respective class called a query set, to allow the prediction to take place we use the support set, and the pre-trained model. The three steps described the heart of the few shot learning task, an example of the process can be seen in the figure2.

2 Triplet network

A triplet network as defined by Elad Hoffer et al. is comprised of 3 instances of the same feed-forward network (In our case a simple CNN) with shared parameters. We feed the Triplet network with 3 samples: anchor x, positive x^+, and negative x^-. Anchor is a baseline image of a given class, positive is an image of the same class as an anchor and the negative sample is an image of a different class than an anchor. The objective of the Triplet network is to minimize the L_2 distance between the anchor and the positive sample, while maximizing the distance between the anchor and the negative sample. In a similar fashion to the definition by Elad Hoffer et al., we can write the problem mathematically in the following way (Θ are the network weights). A description of the network architecture can be seen in the figure 3.

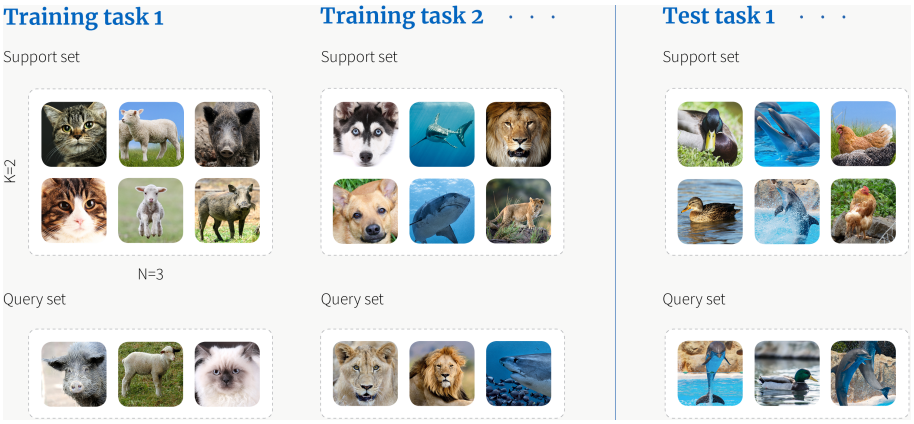

Figure 2: An algorithm is trained using a series of training tasks. Here, each task is a 3-way-2-shot classification problem because each training task contains a support set with three different classes and two examples of each. During training the cost function assesses performance on the query set for each task in turn given the respective support set. At test time, we use a completely different set of tasks, and evaluate performance on the query set, given the support set. Note that there is no overlap between the classes in the two training tasks cat, lamb, pig, dog, shark, lion and between those in the test task duck, dolphin, hen, so the algorithm must learn to classify image classes in general rather than any particular set.

$$TripletNet(x, x^+, x^-) = \begin{bmatrix} ||Net(x) - Net(x^+)||_2 \\ ||Net(x) - Net(x^-)||_2 \end{bmatrix}$$

$$min_\Theta ||Net(x) - Net(x^+)||_2, max_\Theta ||Net(x) - Net(x^-)||_2$$

In the paper Elad Hoffer et al. experienced a hard time (which we also experienced) training the model using the simple definition of the triplet loss described above, Therefore we transitioned to a probabilistic formulation of the triplet loss which, describes the problem as a simple 2-class classification problem, in which we try to minimize the anchor-positive term while maximizing the anchor-negative term. Below the idea is formalized in a mathematical way:

$$Loss(d_+, d_-) = ||(d_+, d_- - margin)||_2^2$$

where

$$d_+ = \frac{e^{||Net(x)-Net(x^+)||_2}}{e^{||Net(x)-Net(x^+)||_2} + e^{||Net(x)-Net(x^-)||_2}}$$

and

$$d_- = \frac{e^{||Net(x)-Net(x^-)||_2}}{e^{||Net(x)-Net(x^+)||_2} + e^{||Net(x)-Net(x^-)||_2}}$$

From here we have a differentiable loss term which can easily use with the back-propagation algorithm, one might notice the a "good" model will learn a use-full representation if the following happens $\frac{d_+}{d_-} \rightarrow 0$ which happens iff $\frac{||Net(x)-Net(x^+)||_2}{||Net(x)-Net(x^-)||_2} \rightarrow 0$.

3 Results & experiments

In this section, we will discuss our results. Firstly, we will look at the re-implementation of the paper by Elad Hoffer et al and the results achieved on the Fashion-MNIST dataset. Secondly, we will look at further work done by us, taking the model further and using it in the few-shot learning scenario. Additionally, we discuss the network architecture and different hyperparameters taken into account during implementing and training our models.

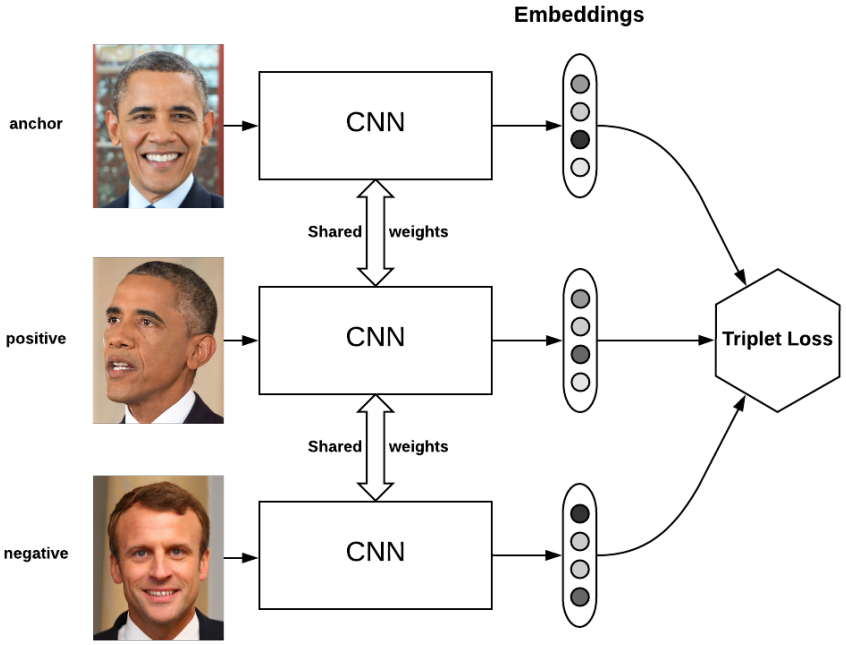

Figure 3: A standard Triplet network architecture, Triplet loss on two positive faces (Obama) and one negative face (Macron)

3.1 Network architecture & hyper-parameters

While training our model we used the following neural network architecture as can be seen in the table 1.

Layer number [#]	Layer Type	Filters [#]	Activation	Kernel size [#]	Padding [#]	Stride [#]	Neurons [#]	Dropout rate
1	Conv2D + Batchnorm2D	32	ReLU	3	1	-	-	-
2	MaxPooling2D	-	-	2	-	2	-	-
3	Conv2D+Batchnorm2D	64	ReLU	3	-	-	-	-
4	MaxPooling2D	-	-	2	-	-	-	-
5	FC	-	Linear	-	-	-	600	-
6	Dropout2D	-	-	-	-	-	-	0.25
7	FC	-	Linear	-	-	-	120	-
8	FC	-	Linear	-	-	-	50	-

Table 1: Backbone neural network model used in the Triplet Network architecture

The following optimizer and hyper-parameter settings were used during training: Adam optimizer with the following parameters $lr = 1 \cdot 10^{-6}, \beta_1 = 0.9, \beta_2 = 0.999$, an exponential decay learning rate scheduler with $\gamma = 0.99$, batch size of 64 examples and 500 epochs, train set size: 60000, test set size 12000. These hyperparameters were used for both experiments, while the difference between them was in the training set, in the metric learning experiment we used all 10 classes, while in the few-shot learning experiment we used only 6 classes for training and evaluated our model over the 4 unseen classes. The 4 unseen classes are: Shirt, Sneaker, Bag, and Ankle boot. When training the models (both for metric learning and few-shot learning case) we monitored the following metrics: train loss, test loss, similarity, and dissimilarity. Where the last two metrics gave us the most insight into our model metric learning ability. Both training procedures can be seen in figures 4 and 5

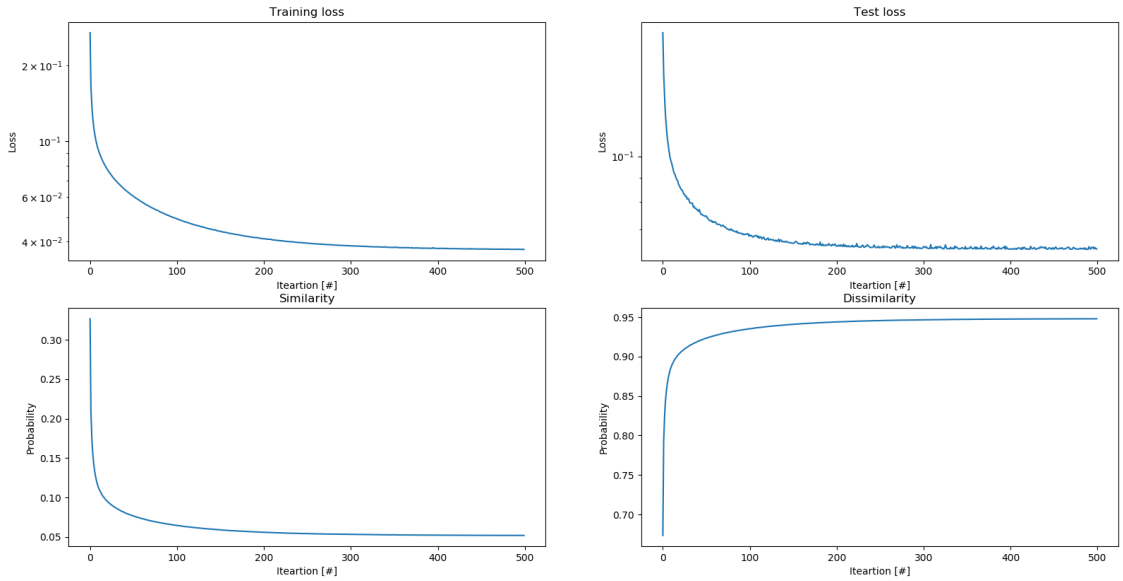

Figure 4: Training procedure for metric learning. Upper left: training loss, Upper right: test loss, Bottom left: similarity, Bottom right: dissimilarity.

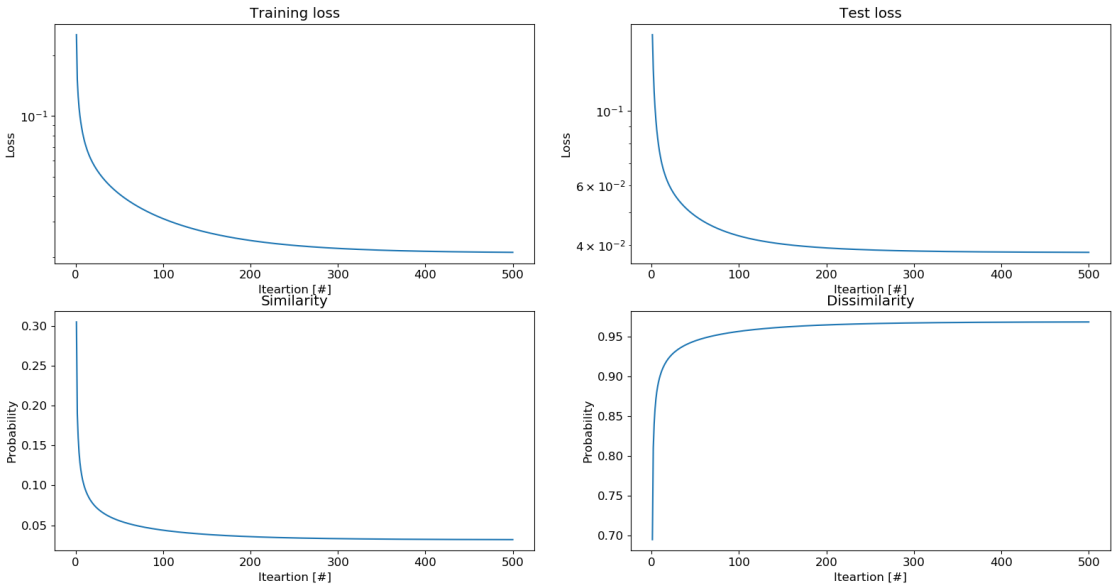

Figure 5: Training procedure for few shot learning. Upper left: training loss, Upper right: test loss, Bottom left: similarity, Bottom right: dissimilarity.

3.2 Metric learning using Triplet network

As described beforehand we will discuss our results about the metric learning ability of the triplet network architecture, but beforehand a disclosure should be said as we didn't have a lot of computing power we could not train big deep learning models, therefore our will achieve worse results than a big deep neural network. Though we can still witness the effect in the T-SNE visualization [15] of the embedded vectors as can be seen in figures 6 and 7. Our visualization will show that our models can help us make the dataset linearly separable which will be supported by accuracy scores of the embedded feature vectors, versus the accuracy of the same deep neural network over the "clean" images as can be seen by table 2.

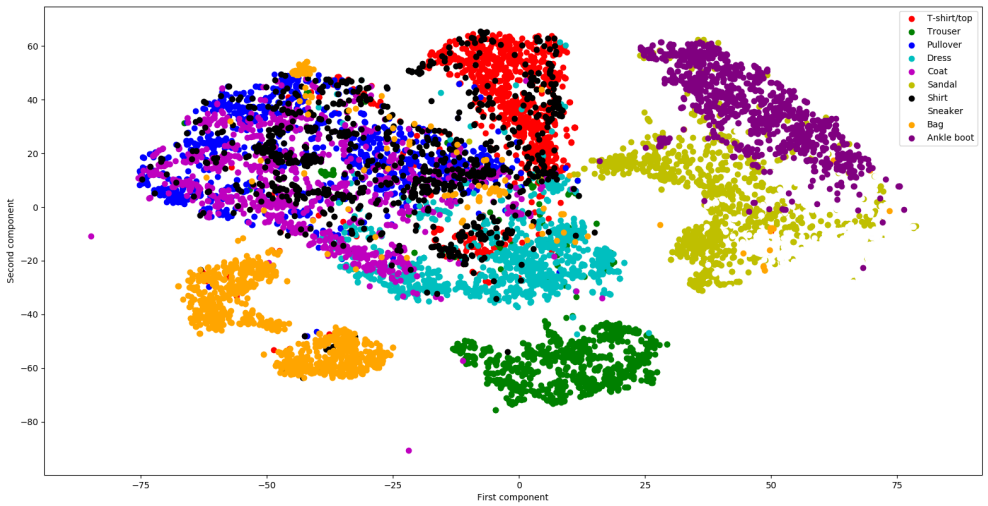

Figure 6: T-SNE (768 to 2 dimensions) of raw Fashion-MNIST dataset, a lot of classes are not linearly separable

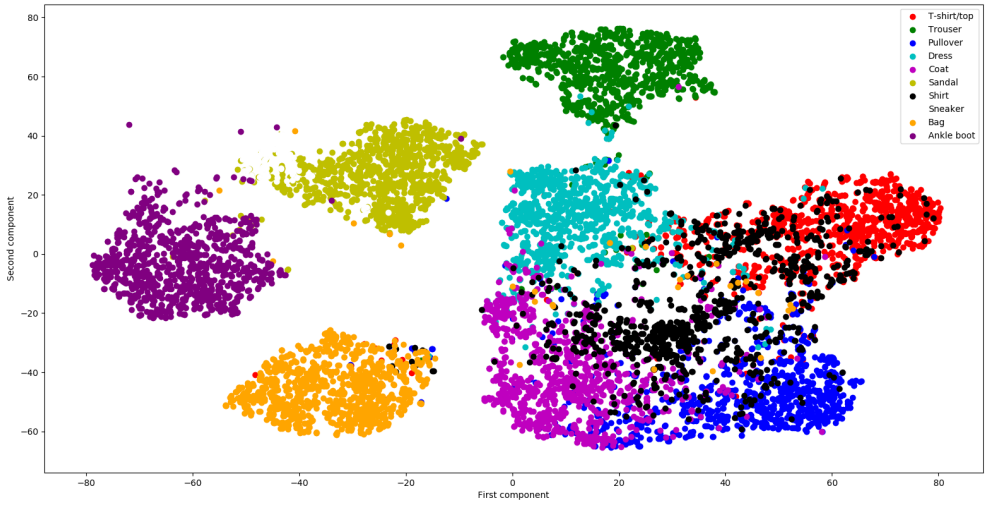

Figure 7: T-SNE (50 to 2 dimensions) of embedded vector out of triplet network, most classes are now linearly separable.

3.3 Triplet network for few-shot learning

As discussed earlier in this subsection we describe the results of the few-shot learning experiment our evaluation procedure, will use standard few-shot learning metrics. Firstly, accuracy with respect to the number of shots, Secondly, accuracy with respect to the number of classes. Our expectation is that when the number of shots grows the accuracy rises, on the other hand when the number of classes increases we expect the accuracy to decrease. The results can be seen in figure 8.

We can see our hypothesis are correct, in addition, we see that after 10 shots we get already high accuracy which is the work point for this classifier in the few-shot learning case. Moreover, we see that increasing the number of classes gives a big penalty to the predictive capabilities of the model, from here a reasonable work point would be 3 classes in the support set.

Model	Input type	Accuracy (test set)
DNN	Raw images	91.01%
Linear SVM	Embedded vector	88.41%
KNN (n=100)	Embedded vector	88.16%

Table 2: Different machine learning models accuracy over the Fashion-MNIST dataset with respect to input-type (Raw images, versus embedded vector via the triplet network)

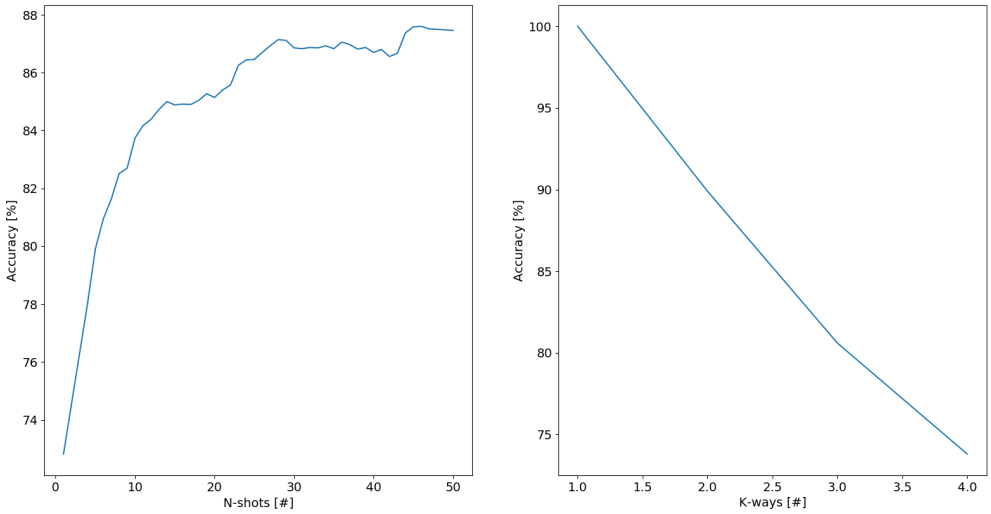

Figure 8: K-way-N-shot evaluation of out model, results were averaged over 1000 queries. Left: N shot evaluation number of classes $k_{way} = 3$. Right: K way evaluation number of shots $n_{shot} = 5$.

4 Discussion & conclusion

During this project, we imported the "Deep Metric Learning Using Triplet Network" by Elad Hoffer et al. to modern PyTorch [16] from LUA and evaluated our results on a different data set called "Fashion MNIST", during this experiment we saw as expected that the metric learning done by the triplet network allows linear classifiers to achieve high accuracy. Additionally, we contributed by showing the model can be used for few-shot learning achieving high accuracy on the Fashion-MNIST for unseen classes during test time. The project taught us a lot about the world of metric learning and few-shot learning two fields we have not focused on during the course. Additionally, from the project, we understood that few-shot learning might be one of the few machine learning techniques that will allow deep learning models to work in highly specialized fields with a small amount of data.

Acknowledgments

The project was done to fulfill the requirements of the "Deep Learning" (046211) course, which was very interesting and fun to attend during the semester.

References

[1] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network, 2018.

[2] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[4] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[5] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[7] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

[8] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

[9] Geoffrey E. Hinton. Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11:428–434, 2007.

[10] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 209–216, New York, NY, USA, 2007. Association for Computing Machinery.

[11] Guo-Jun Qi, Jinhui Tang, Zheng-Jun Zha, Tat-Seng Chua, and Hong-Jiang Zhang. An efficient sparse metric learning in high-dimensional space via l_1-penalized log-determinant regularization. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 841–848, New York, NY, USA, 2009. Association for Computing Machinery.

[12] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, page 737–744, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[13] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition, 2018.

[14] Yaqing Wang, Quanming Yao, James Kwok, and Lionel M. Ni. Generalizing from a few examples: A survey on few-shot learning, 2020.

[15] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.