NAVEE PROJECT : REPORT

Vassili Chesterkine
vassili.chesterkine@student-cs.fr
Haoyi Han
haoyi.han@student-cs.fr
Carlos Santos García
carlos.santos@student-cs.fr
Nicolas Vitor Yuki Obara Yamakoshi
nicolas.yamakoshi@student-cs.fr
Ke Zhang
zhang.ke@student-cs.fr

supervised by
Vladyslav SYDOROV

June 13, 2021

Abstract

Despite recent progress in computer vision, some tasks of this field remain especially hard. In this paper, we will discuss few-shot classification, which consists in training a classifier that recognizes unseen classes during training with limited labeled examples. Though multiple strategies to tackle this task exist, we mainly explore the use of pre-trained embedding spaces where distances correspond to similarities between samples.

For our task, we use deep convolutional neural networks that we train on the available data, in order to construct our embeddings optimally, based on well-chosen loss functions and data augmentation. Our strategy is then to use classifiers such as k-nearest neighbors or simple perceptrons on this representation of images to classify unseen data with reasonable accuracy.

Notably, we achieve a considerable improvement in accuracy on various few-shot learning tasks, compared to our baseline model. This paper describes our strategy, experiments and results.

1 Introduction

Few-shot learning is increasingly popular as numerous applications arise, such as the analysis of medical images or face detection for identification purposes.

The objective of this project is to evaluate the feasibility of using pre-trained feature extractors to quickly categorize products based on images with limited amounts of data. Our client Navee works with fashion brands and brings them insights to their images available online. They seek to develop ways to extract information automatically from the images of their clients. Fashion brands sell numerous types of articles, and new ones are regularly released. It would consequently be a tedious task to labelize considerable amounts of images. That is why we would like to be able to build a model that could recognize an article, only using a small amount of labeled images as reference.

To tackle this problem, we first explored deep learning classical techniques applied to fashion images using the Kaggle Fashion Dataset, an open-source dataset that contained label fashion-oriented images. We tried different tools including feature extractors and classic convolutional neural networks. Yet, this dataset contained a large amount of label images for each class, and this introduction work was not proper few-shot learning. Having gained some experience on such methods, we then moved on to an actual few-shot learning framework, with images that Navee provided us.

This consisted in developing models on a dataset of images from Navee's clients, where every article had a small amount of labeled images. For each article, there were several photos, taken from several angles. Some were zoomed-in views of the article and some others were photos of models wearing that particular article, as illustrated in figure 8. Contrary to other frameworks used in research papers, our data was therefore not very homogeneous, as in some images, it is hard even for humans to detect and recognize the displayed item.

Our main approach to this few-shot learning problem was based on learning an embedding representation of the images, using convolutional networks. We trained our networks with the idea that the distance between the embeddings of two images should be consistent with the similarity of these images : images of the same article should live close by in the embedding space, and conversely images of different articles should be separated by a large distance. Most of our work revolved around strategies to improve the performance of our models on this task. Having trained this embedding function, we then focused on different ways to classify images using this new representation.

While we were free to try any method that we would deem relevant, the purpose of this project was to provide our client Navee with some useful insights about what techniques to explore further and even to use for their needs. We therefore mainly followed our supervisor's recommendations on what to look into, and not necessarily what we could read in related scientific papers.

Still, in the next section, we will proceed to a short review of the work that has been previously done on similar topics. In section 3, we give a formal definition of the problem and of our evaluation metrics. In sections 4, 5 and 6, we present different strategies that we used. Finally, in sections 7 and 8, we present our work and quantitative results, before concluding.

2 Related work

Few-shot learning (FSL) has been an active area of machine learning and computer vision for some time [1]. It aims at developing models capable to classify objects with little data to train on. There are two main approaches that are generally considered when trying to solve FSL problems.

The first is data augmentation. It is based on the idea that if you don't have enough data to build a reliable model, you should simply gather more data. To bypass the lack of labeled data for the task at hand, it is possible to use a large base-dataset that does not necessarily have the same classes as the FSL dataset, but which contains similar images. Another logical approach is to generate more data from the existing labeled data. This is what is commonly referred to as data augmentation. Classic data augmentation techniques include simple transformations performed on images : horizontal or vertical flips, rotations, random crops or color jitter [2]. These methods make it possible to increase the size of the FSL dataset, and contribute in reducing overfitting [3]. An alternative to these conventional methods is the use of Generative Adversarial Networks (GAN), in an attempt to create more data [4].

The other approach is meta-learning, which aims to improve the learning algorithm itself, contrary to what is done in more conventional AI tasks where the learning algorithm is fixed. Concretely, a good meta-learning model is expected to be capable of adapting and generalizing to new tasks well. In the meta-learning framework, learning is achieved through a set of training tasks. At each step of the training phase, parameters are updated based on a randomly selected task and the loss that is computed on it. Since the network is evaluated on a different task at each step, it must learn how to discriminate data in general. Meta-learning algorithms can broadly be divided into two large groups.

The first gathers models that aim to create representations (or embeddings) for which similar inputs have similar embeddings and conversely, dissimilar inputs have dissimilar embeddings [5]. This is often referred to as metric-learning, that is learning a distance function over objects. Early work mainly focused on pairwise comparators, which decide whether two inputs are drawn from the same class or not. Siamese networks [6] are an example of such models, which take as input several images, compute embeddings for each and are updated based on an adapted loss function, which aims to improve the representation of images in the embedding space. This topic will be further explored in 5.1 with the implementation of triplet networks [7]. Other metric-learning methods include matching networks [8] or prototypical network [9], the first of which is less robust to data imbalance (ie. classes with more samples are likely to dominate). The second group of models is based on using prior knowledge to force the learning algorithm to choose parameters that generalize well from few examples. Some models aim at choosing parameters that can be easily fine-tuned to new tasks through gradient descent steps. Meta-agnostic meta-learning (MAML) [10] is an example of this, and will be further explored in 6.1. Other models focus on the optimization process itself, as opposed to MAML, which revolves around finding a suitable initial set of parameters. Although we have not extensively looked into these types of models, the method presented in [11] might bring interesting results.

3 Problem statement

3.1 Few-shot classification task

We are given a dataset $D = (D_{support}, D_{query})$, where the dataset :

$$D_{support} = \big((x_i, y_i)_{i \in I}\big)$$

is such that for every given value y of labels $\#\{i \in I \mid y_i = y\}$ is small, generally about 5. $D_{support}$ and D_{query} are necessarily taken from the same distribution, ie. for any sample point (x, y), $\exists i \in I, y_i = y$.

For the sake of clarity and to provide a better framework to evaluate performance, let's now define the N-way-K-shot classification problem. This means that $D_{support}$ consists of N classes which each have exactly K samples. Intuitively, smaller values of K and greater values of N will be synonymous with harder tasks.

For a given N-way-K-shot classification problem, we seek to find $\hat{h_{N,K}} : x_i \mapsto y_i$, a function that maps one data point x_i, an image, to a label y_i, corresponding to that article. $\hat{h_{N,K}}$ should only be based on $D_{support}$. The performance of such an estimator $\hat{h_{N,K}}$ is evaluated through the computation of an accuracy metric $A(\hat{h_{N,K}})$ on the query set D_{query}, given by the simple following formula:

$$A(\hat{h_{N,K}}) = \frac{\#\{(x, y) \in D_{query} \mid \hat{h_{N,K}}(x) = y\}}{\#D_{query}}$$

which gives the proportion of test items that were correctly labeled by $\hat{h_{N,K}}$.

Additionally, we were given a dataset of images and labels D_{base} from a similar distribution. D_{base} can be seen as a training set that we could use in order to improve $\hat{h_{N,K}}$.

3.2 Image retrieval task evaluation

Some of the models presented in section 5 are used to learn a representation of images such that similar images get mapped to a space $\mathbb{R}^d$ where images depicting similar objects stay close to each other. To learn such a representation, we consider an auxiliary problem, called retrieval task, where given a query image and a support set of images from previously-unseen classes, we aim to retrieve all images of the support set that have the same label as the query image. For this purpose, we use D_{base}, which provides a significant number of samples for the model to learn a representation of images on. After this step, we can then used the learned representation to perform few-shot learning classification. For instance, since the embedding function should map the query image close to the images of the same label, it is possible to use the label of the image closest to the query image in the representation space as a prediction for the label of the query image.

It would not make sense to use accuracy to evaluate performance on the retrieval task, therefore a more relevant metric is needed. We will consequently use the mean average precision metric, as presented in the survey [12], to evaluate our retrieval models.

The main idea is that we compute for every sample (x_q, y_q) in our query dataset D_{query} the average precision $AP(x_q)$ of a binary classifier that predicts whether or not each image x_s from the support dataset $D_{support}$ belongs to the same category y_q as x_q or not. Mean average precision is then computed as follows:

$$mAP = \frac{1}{\#D_{query}} \sum_{x_q \in D_{query}} AP(x_q)$$

Average precision (AP) is obtained using the embeddings produced by the different models presented in 5: if the embedding produced by one of the architectures is called g, this embedding can usually be $g : \mathbb{R}^{224\times224\times3} \to \mathbb{R}^{256}$ and can be used to compute a similarity function d between two images x_1, x_2 by computing:

$$d(x_1, x_2) = \|g(x_1) - g(x_2)\|_2$$

This similarity function allows us to compute the similarity of the query image to any of the support images by considering $\{d(x_q, z), z \in D_{support}\}$ and perform classification to find the images similar to x_q and labelize them as y_q. To evaluate the quality of this binary classifier, we compute precision and recall scores for different similarity-threshold values. The trade-off between precision and recall gets illustrated in the precision-recall curve that plots them for different threshold values. Average precision (AP) summarizes this plot by computing the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight:

$$AP(x_q) = \sum_n (R_n - R_{n-1})Pn$$

where P_n and R_n are the precision and recall values at the n-th threshold.

4 Conventional computer vision task

This section presents our methodology for using several feature extractors combined with machine learning techniques to perform image classification on a fashion-oriented dataset.

4.1 Kaggle Fashion Product Images dataset

The dataset used in this section can be found using the link here. It is composed of 44.441 60×80 pixels fashion images. Each image is described by its pixels, as well as a masterCategory, a subCategory and an articleType. The label masterCategory is the broadest one, as there are only seven possible classes (Apparel, Accessories, Footwear, Personal Care, Free Items, Sporting Goods or Home). On the contrary, articleType is the most precise one. Some common examples of articleType are t-shirts, casual shoes or watches. The label subCategory is in-between, and some of its examples are topwear, shoes or belts. It is important to note that subCategory is a sub-category of masterCategory, that is each masterCategory gathers several subCategories. Similarly, articleType is a sub-category of subCategory, and of masterCategory by extension. An illustration of some of it images are available on figure 3. All of the images used in this section were used in grayscale to decrease computational intensity. The distribution of the labels is shown in charts 1 and 2.

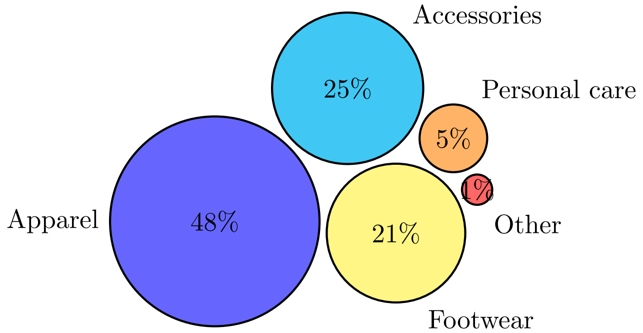


Figure 1: *masterCategory classes distribution in Kaggle dataset. The other categories are Sporting Goods, with 25 samples, and Home, with only one image. They were therefore discarded for our classification task.*

The objective of using Kaggle's dataset was to familiarize ourselves with deep learning classical techniques and architectures by performing image classification. Our end goal was to evaluate the performance of our models on the three different labels: masterCategory, subCategory and articleType. As expected,

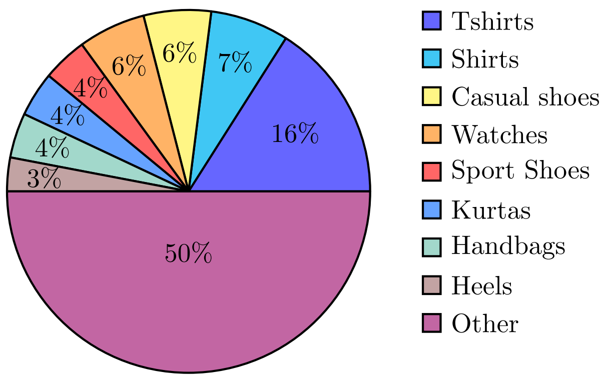


Figure 2: *Main articleType categories in Kaggle dataset. Other categories gathers 99 labels representing each 3% or less of the datasete.*

the most challenging task was predicting successfully articleType's for each image.

4.2 Feature extraction

Computer vision techniques generally rely on the extraction of meaningful features from the images, which are then used to decide which of the classes each image comes from. This can be done by using multi-layer perceptrons that take as an input the pixel values of the images, but some more relevant features can be obtained using convolutional layers, or other features. The structure we use is presented in figure 4.

4.2.1 Convolution Neural Networks

As presented in this survey [13], convolutional neural networks are systems that are able to extract features from data with convolution structures . CNNs can reliably extract complex features that represent the image in great detail on their own. This not only spares us from having to extract handcrafted features, which seems like a tedious task, but also provides features that are more efficient. The architecture of CNNs is inspired by visual perception: convolutional kernels represent different receptors that respond to particular features, and activation functions simulate the fact that above some threshold electric signals are transmitted between neurons. Kernels are governed by weights, parameters that are learnt during the training process to decrease the loss function. These types of networks have achieved impressive results in computer vision tasks, like the Imagenet competition, which has enabled some networks to be trained on more than 10 millions of images. This knowledge can be reused for our task by using their pre-trained networks, whose weights result from the aforementioned training.

In our work, we will be using VGG-16 [14] and different versions of ResNet (ResNet-18, ResNet-50)[15] networks, both presented in 2015. The former established the importance and relevance of using networks with increasing depth with smaller filters, while the latter introduced much deeper networks using residual blocks that can allow very accurate learning. These pre-trained networks have gained substantial knowledge that can be used in our classification task.

These networks will be used as feature extractors, before classifying the images from the dataset.

4.2.2 Histograms of Oriented Gradients

First introduced by INRIA researchers in [16], Histograms of Oriented Gradients (HOG) are feature descriptors, originally used to perform human detection, but that have been successful in object recognition tasks. These feature descriptors provide a representation of an image that simplifies it by extracting useful information from its original form, typically to follow up with a classification task.

The idea behind HOG features is to encode not only the presence of edges in an object but also their direction, by extracting both the location and orientation of gradients on the image. This gradients are computed in local portions, on which a histogram is then created considering both the magnitude and the orientation of these gradients.

Images need to be preprocessed, typically cropped and reduced to a fixed ratio size. In our implementation, we used the original 60×80 images from the dataset. Additional preprocessing involving colour correction can be done, as mentioned in [16], but the gains are minimal so this step was skipped. After computing HOG features, the obtained vector belongs in our setup to $\mathbb{R}^{1728}$ and will be used to perform the classification task. An example of the oriented gradients obtained is presented in figure 5.

4.3 Multi-class classification

Classification is done automatically on CNN structures, by adding fully-connected layers on top of the convolutional layers, which allows to output a probability density over the possible classes.

When using HOG extractors, the feature vectors are fed into a k-nearest neighbors classifier or to a multi-layer perceptron in order to perform the final classification. With this final step, we can compute a prediction for each image, classifying it as one of the possible labels, depending on the level at which we wanted to classify : masterCategory, subCategory and articleType.

Figure 3: *Small sample of images from Kaggle Product Images dataset.*

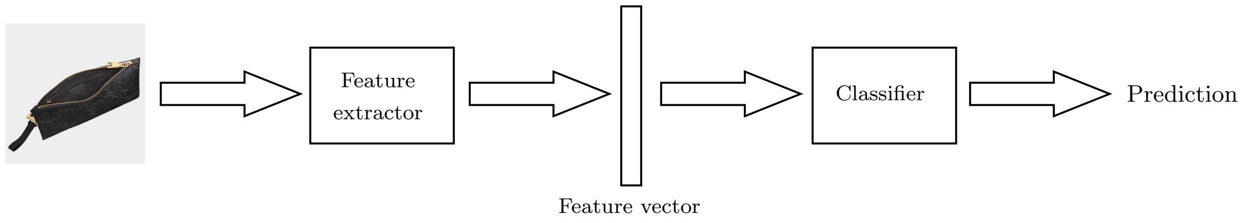


Figure 4: *Basic architechture used in section 4. Feature extractors we used include deep convolutional networks (VGG, ResNet) and classifiers can be nearest neighbors algorithms, but also multilayer perceptrons and ensemble classifiers to improve performance.*

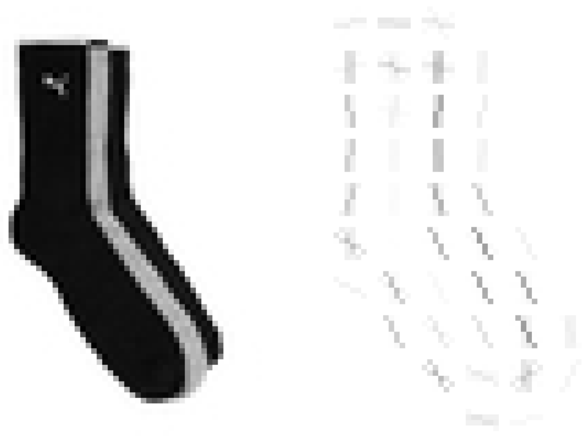


Figure 5: *Oriented gradients obtained with a sock image. The information encoded with this system is then transformed into a feature vector in $\mathbb{R}^{1728}$ that is used to perform the classification.*

Additional tests were made using ensemble learning techniques with multiple classifiers to improve the overall performance of our system. We will discuss this in the following section.

4.4 Networks fine-tuning and advanced techniques

Here are the different approaches we tried to improve our performance on this classification task.

4.4.1 Network fine-tuning

As a few thousand images were available for training, we used the images available to fine-tune ResNet-50 and predict the different labels.

4.4.2 Ensemble learning - Voting

Empirically, using different classifiers as an ensemble system tends to give better results than using each one of the classifiers alone. The idea is to combine the information of each predictive output to produce a single final output, in order to improve the overall performance. To create a such ensemble, we would typically elaborate different models separately, try to reach maximum performance with each, and then use each prediction to make a better-informed final prediction. The two main techniques used are soft and hard voting.

In hard voting, every individual classifier votes for a class, and the label that gathers the majority of votes wins. This encourages the use of an odd number of classifiers to avoid ties.

In soft voting, every individual classifier provides a probability value that a specific sample belongs to a particular class, and the predicted label corresponds to the one that, in average, has the biggest probability. This technique is particularly adapted to the use of neural networks, that usually have a softmax layer that allows to output a probability density over the target classes. We therefore used this approach and used three independently trained ResNet-50 models to

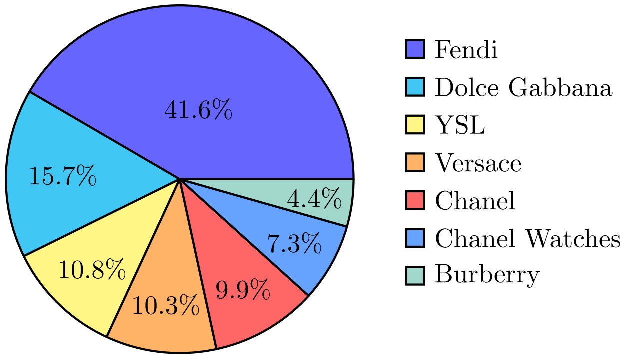


Figure 6: *Brands distribution in the dataset*

improve the final results.

5 Representation learning

Unlike in the previous part, the dataset that is used here is a *real* dataset, provided with Navee, containing various images from their partner brands. A chart of the distribution of all images amongst the 7 brands is presented in figure 6. Each brand contains a certain number of categories, each one having its own subcategories. Overall, this amounts to 3967 classes across all 7 brands, all of which a variable number of images. Below is a histogram that represents the number of images per class.

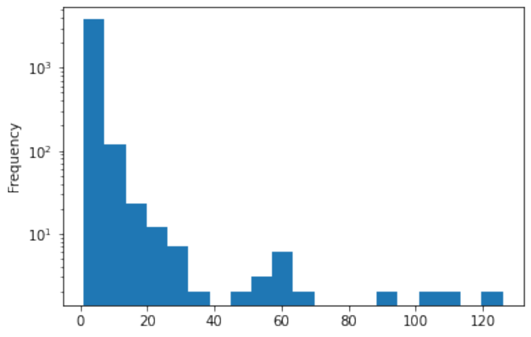


Figure 7: *Number of images per class*

This suggests that the majority of classes contains less than 20 images per category. To come back on the figure of 5 images we gave to define few-shot learning in 3.1, the number of classes with 5 images or less is 2926, that is about 74% of all classes. An example of the images each class can contain is available in figure 8.

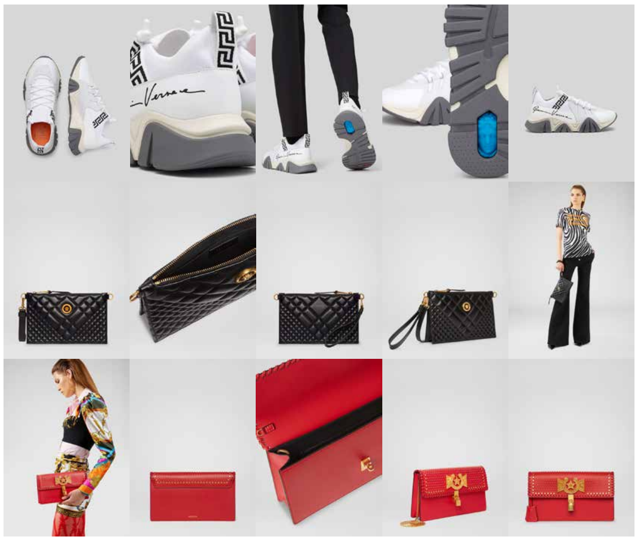


Figure 8: *Examples of images from 3 different classes from Versace. Articles are seen from different points of view and are sometimes difficult to recognize when displayed on someone, which makes the classification task even harder.*

To work on this dataset, we have split it into a base set and a test set. The former contains images that will be used to train our models to learn an adapted representation. The latter is divided into two separate validation sets, one containing all available images from Givenchy, the second one containing a subset of images from Versace (50 classes, the rest is left in the train set) for us to evaluate our models on. The base set and the test sets thus form a partition of the available images. This allows us to respectively run tests on images from a completely new and unseen distribution and on images that come from the same distribution as some images of our train set. Then, each test set itself was split in 2 : a support set and a query set, which allow us to compute mAP, as discussed in 3.2.

We aim at finding an embedding function f, which maps an image x to an embedding vector $f(x) \in \mathbb{R}^d$. Additionally, for two input images x and y which represent the same real-life object, the euclidean distance between $f(x)$ and $f(y)$ should be small and conversely, the same euclidean distance should be large if x and y represent different objects. For this purpose, we have used convolutional networks to extract features from images and have trained them using specific loss functions.

Intuitively, the loss function should favor the creation of clusters, formed with images from the same category, and separate clearly different clusters.

The following section discusses our choices of convolutional networks as well as loss functions, with their respective pros and cons.

5.1 Siamese Networks

A classic approach to metric learning involves neural network architectures which contain two or more identical sub-networks, which share the same characteristics and parameters. Each sub-network is used to compute an embedding for an input, and all embeddings are then compared with an appropriate loss function, which leads to an update of the parameters across all sub-networks. Such an architecture is presented in figure 9. As siamese networks learn a similarity functions between inputs, it is possible to classify classes that were not in the training dataset without any additional training of the network, which is interesting for our retrieval task.

There are several ways to implement siamese networks, we will discuss some in the following sections.

5.2 Contrastive loss

Contrastive loss was first introduced in 2005 by Yann Le Cun et al. in [17]. It is a loss function that is based on the comparison of the representations of two training data samples. It uses pairs of inputs, either positive pairs (an anchor sample x^a and a positive sample x^p which belong to the same category) or negative pairs (x^a and x^n which belong to different categories). The objective is to learn a representations of the inputs where the distance between similar training samples is small, and conversely, the distance between different inputs is greater. Concretely, contrastive loss has the following expression, given a pair of inputs (x, x'):

$$L = \begin{cases} d(x, x') \text{ if } (x, x') \text{ is a positive pair} \\ \max(0, m - d(x, x')) \text{ else} \end{cases}$$

where d is usually the euclidean distance in the representation space, and m is a hyperparameter called margin, which is in theory the minimal distance that we try to force between two inputs of a negative pair.

For positive pairs, the loss will be 0 when both embeddings are strictly identical and will be non-negative as long as both embeddings are different. For negative pairs, the loss will be 0 when the distance between both embeddings is greater than the margin m, and will be non-negative otherwise.

In our implementation we used the following equivalent formula : given a input pair (x, x') and a label $y = 1$, equal to 1 if (x, x') is positive and equal to 0 otherwise, the loss is

$$L = y \times d(x, x') + (1 - y) \times \max(0, m - d(x, x'))$$

5.3 Triplet loss

Given a triplet of images $(x^a, x^p, x^n) \in D^3_{train}$ where x^a, x^p belong to the same category and x^n to a different category, let's denote a distance function d defined as $d(x, y) = ||f(x) - f(y)||^2_2$. We would like the embedding function to meet the following condition :

$$d(x^a, x^p) + m \leq d(x^a, x^n) \quad (1)$$

where the margin m is a chosen parameter that enforces a certain distance between positive pairs (anchor and positive) and negative pairs (anchor and negative). Subsequently, the triplet loss function associated to a triplet $(x^a, x^p, x^n) \in D^3_{train}$ is as follows :

$$L(x^a, x^p, x^n) = \left[d(x^a, x^p) + m - d(x^a, x^n)\right]_+ \quad (2)$$

Let's denote $\mathcal{T}$ the set of all triplets, formed with images from the training set. We want to minimize the following quantity :

$$L = \sum_{(x^a, x^p, x^n) \in \mathcal{T}} L(x^a, x^p, x^n)$$

Yet, this is practically impossible, as generating all possible triplets is far too compute-intensive. In fact, if we only consider 100 categories with 5 images each, there would be just shy of 1.5 million triplets to consider for each training epoch. Also, among all these triplets, many would already satisfy (1) and would therefore not contribute to the training. It thus becomes clear that not all triplets should be used for training in order to improve convergence. Instead, a selection of useful triplets should be carried out before computing gradients.

5.4 Triplet selection

Based on the the definition of the loss function, there are 3 types of triplets :

- easy triplets : the anchor and the positive are close together, the negative is far from both, ie. $d(x^a, x^p) + m < d(x^a, x^n)$ and the loss is 0.
- hard triplets : the negative is closer to the anchor than the positive, ie. $d(x^a, x^n) < d(x^a, x^p)$ and the loss is greater than m.
- semi-hard triplets : the negative is not closer to the anchor than the positive, but the difference of both distances is smaller than m, ie. $d(x^a, x^n) \in [d(x^a, x^p), d(x^a, x^p) + m]$, thus the loss is between 0 and m.

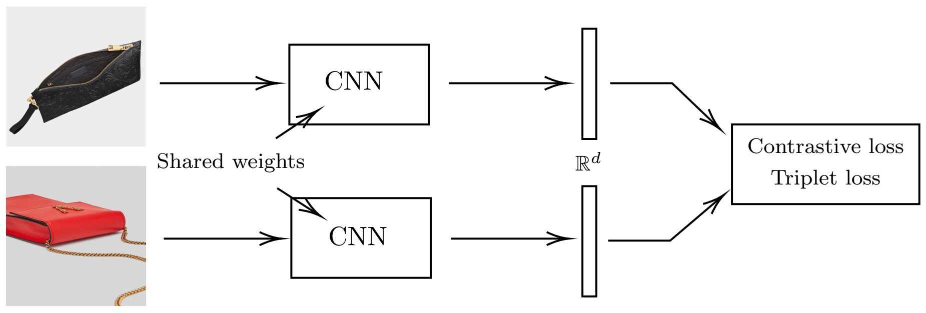


Figure 9: *Siamese networks structure: the choice of the loss depends on the chosen structure.*

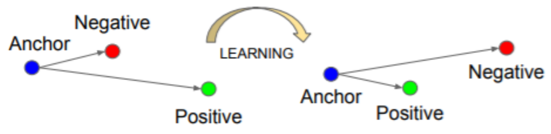


Figure 10: *Triplet loss attempts at penalizing embedding functions where the distance between the anchor and the positive is not significantly smaller than the distance between the anchor and the negative.*

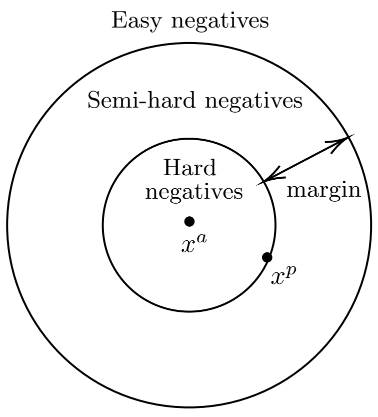


Figure 11: *Types of triplets*

This is illustrated in figure 11. As argued by the FaceNet paper authors [18], selecting semi-hard or hard triplets will have a considerable impact on our results. There are two main approaches to triplet selection :

- offline selection : generate triplets offline, compute triplet loss for all triplets based on the latest weights of the feature extractor, and select semi-hard or hard triplets only for training.
- online selection : given a mini-batch of inputs, compute embeddings and, for instance, for each anchor, choose the positive which maximizes $d(x^a, x^p)$ and the negative which minimizes $d(x^a, x^n)$ to create a 'hard' triplet.

Though it is not obvious that online selection is the more efficient method, this paper [19] claims that this strategy yields optimal performance. As this is dependent on the dataset, as well as harder to code in our implementation and still demands high amounts of computational power (which was not available to us), we chose another approach to online mining. Given a mini-batch, we would compute all losses and select the top k % of greatest losses to compute gradients on, where k is a hyperparameter that we selected.

5.5 Classification using embeddings

Having obtained an embedding functions which maps an image to a representation in a high-dimensional space, we can now come back to our primary task, that is classifying unseen images using these representations. For this purpose, we have tried two different methods, which seemed quite logical to implement : k-nearest neighbors classification and classification using a simple neural network.

5.5.1 k-Nearest Neighbors

The principle of this method is fairly straightforward. Considering unseen from new classes, and some query image that belongs to one these classes, we can compute the embeddings of all images and use a strategy based on k-nearest neighbors to identify the class of the query image. As we have built our embedding function to map similar inputs close to each other and dissimilar inputs with some distance, this strategy should yield interesting results.

5.5.2 Perceptron

It is also possible to use a multi-layer perceptron (MLP) to classify images based on their embeddings. For this approach, we chose a simple neural network with only 1 hidden layer.

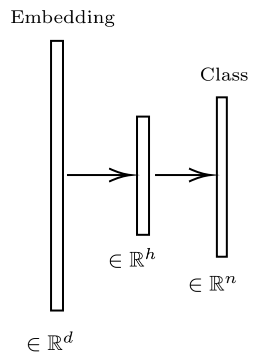


Figure 12: *Structure of the MLP : d is the dimension of the embeddings, h is the dimension of the hidden layer and n is the number of classes of the query set*

The idea here is that we can not only use the representation that was learned with our siamese networks, but also fine-tune this representation to best adapt to unseen data. The downside of this approach is that unlike with the kNN-based strategy, which required virtually no additional training, it requires some time and effort to fine-tune representations to the unseen data. In fact, many hyperparameters can be tweaked and lead to different performance : the dimension of the hidden layer, the learning rate of the optimizer, the regularization parameters (dropout and L2 regularization).

Additionally, as suggested in [20] for image retrieval, we decided to use a dimensionality reduction technique, in conjunction with the MLP. In fact, reducing the dimension of our embeddings could help remove some redundancy, making it a simpler task for the network to classify correctly and improving accuracy, as well as reducing computation time, which is always desirable.

6 Other techniques

As the project expanded and our understanding of the project grew, we explored other few-shot learning techniques, like Model-Agnostic Meta-Learning and Prototypical Networks.

6.1 Model-Agnostic Meta-Learning

Model-Agnostic Meta-Learning (MAML) was first introduced by Chelsea Finn et al. in [10]. The aforementioned work proposes an algorithm to train a model on multiple learning tasks, for it to solve new learning tasks using only few training samples and few gradient descent steps. Fundamentally, it's a method to create a model whose parameters are easy to fine-tune with the minimum amount of data and iterations possible.

For our application of FSL classification, the training dataset composed of images of luxury articles is split into a training dataset composed of Tasks, where each Task $\mathcal{T}_i$ is a n-way k-shot classification problem on its own, with a query image and a support set. The model architecture can be any one that can be trained with *gradient descent* (hence the model-agnostic nomenclature) and in our case, it was a convolutional neural network. Its parametrized function is denoted by f_θ, while its parameters are denoted by θ.

The training method starts with a sample batch of tasks $\mathcal{T}_i$. For each task $\mathcal{T}_i$, the model's parameters θ are copied on their current state and updated to θ_i', which is the updated parameter vector for this specific task and it's calculated as:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta),$$

where α is the step size hyperparameter and $\mathcal{L}_{\mathcal{T}_i}(f_\theta)$ is the loss evaluated on task $\mathcal{T}_i$ using θ as the model's parameter. The equation above represents a single step on gradient descent for simplicity, but the usage of multiple updates can be done with some slight modification on the equation and further iterations.

Next, after all θ_i' for all tasks from the batch have been calculated, θ is updated as follows:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$$

where β is the meta step size hyperparameter and $\mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ is the loss evaluated on task $\mathcal{T}_i$ using θ_i' as the model's parameter. This represents the meta learning step, which is a single gradient descent step. More steps can be done to improve performance, since this is the step that actually trains the model to better adapt to all future tasks that are similar to the training examples. Moreover, this whole process from the sampling of batches of tasks until the θ update represents an iteration over an epoch.

6.2 Prototypical Networks

Prototypical Networks are a relatively new neural network model adapted to FSL classification, introduced by Jake Snell et al. in [9]. It is a simple method with a very intuitive structure, as the authors indicated that this is because all of the required non-linearity can be learned within the embedding function.

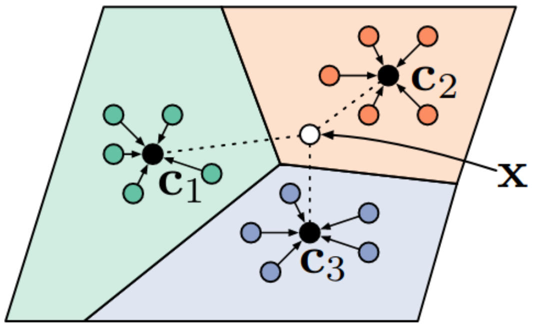


Figure 13: *Illustration of prototypical networks in the embedding space*

To explain the idea of this method, we assume each category can be associated to a *prototypical* point as center point around which all the points of this category are situated. Then, following the definition of FSL, we aim to learn to fit this prototypical center by a few samples via metric functions. First, we map the data in the support set to an embedding space. Then, we calculate the average of all the embedded images of the same category, which will yield the embedding of the prototype of the category. Finally, for the prediction, we map the data into the embedding space and calculate its distance to each prototype, then we choose the one with the minimal distance to the predicted prototype, i.e. the category to which the data belongs. This method is illustrated in figure 13.

Given a support set $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $x_i \in \mathbb{R}^D$ is an n-dimensional vector and $y_j \in [\![1, K]\!]$ is one of K categories. We denote S_k the subset of the k-th category in S. Then we define the prototype (center point) of the k-th category as

$$c_k = \frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} f_\phi(x_i)$$

where $f_\phi : \mathbb{R}^n \to \mathbb{R}^d$ is the encoder function to learn.

When doing the prediction of data in query set, we choose a metric function $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ and calculate the probability

$$p_\phi(y = k|x) = \frac{\exp(-d(f_\phi(x), c_k))}{\sum_{k'} \exp(-d(f_\phi(x), c_{k'}))}$$

And we aim at minimizing the loss function

$$J(\phi) = -\log p_\phi(y = k|x)$$

In conclusion, this algorithm can be summarized as shown below:

```
Algorithm 1: Prototypical Networks
  Data: the training set
        $T = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ where
        $y_j \in [[1, K]]$ and $T_k \subset T$ such that
        $\forall (x_i, y_i) \in T_k, y_i = k$
  Result: the loss $J$ for a randomly generated
          training episode
  randomly select class indices for episode;
  for $k \leftarrow 1$ to $N_c$ do
      randomly select support examples of $N_s$
        categories that constitute $S_k$;
      randomly select query examples of $N_q$
        categories that constitute $Q_k$;
      compute prototype (point) $c_k$ from support
        examples $S_k$ by formula above;
  end
  $J \leftarrow 0$;
  for $k \leftarrow 1$ to $N_c$ do
      for $(x, y)$ in $Q_k$ do
          $J \leftarrow J + \frac{1}{N_c N_q}\Big[d(f_\phi(x), c_k) +$
            $\log \sum_{k'} \exp(-d(f_\phi(x), c_k))\Big]$;
      end
  end
```

7 Preliminary results

As presented in section 4 the obtained results using ResNet and HOG features extractors, combined with different techniques, to perform the image classification task on Kaggle dataset. The original dataset was split into a train dataset (80%) and a test set (20%).

7.1 Classification on different levels of granularity

Pre-trained embeddings, as well as HOG extractors turned out to be sufficiently rich to allow very good results on the available data, as shown in table 1. These results are satisfactory and even on the most challenging task, where 107 classes are available, the accuracy is fairly good. Exploring some of the incorrect predictions, as done in figure 14, shows that the mislabeled examples are also hard to labelize for humans, suggesting that our systems are close to human-performance. As the task of predicting articleType is the most difficult one, we will focus from now-on on this particular label.

	masterCategory	subCategory	articleType
ResNet-50	98%	94%	82%
HOG + kNN	98%	91%	80%

Table 1: *Test accuracies obtained with the two main methods explored in section 4 on predicting masterCategories, subCategories and articleTypes for Kaggle dataset images.*

7.2 Classifying using HOG Features

As presented in table 1, HOG extraction needs to be combined with a classification technique in order to predict the target labels. This has been done by using both k-nearest neighbors (kNN) and a multi-layer perceptron (MLP) after encoding the images into feature vectors. These feature vectors were normalized to improve the performance of these algorithms. Results comparing both techniques are shown in table 2, and suggest that MLP can improve the overall performance, but need to be retrained if new images get added into the data.

7.3 Voting classifiers

Using three different pre-trained ResNet-50 models to which we added one fully connected layer, that was randomly initialized with three different seeds, we fine-tuned the models to predict articleType on Kaggle's dataset. Results are shown in table 3. This encourages us to use voting with various classifiers, as will be seen in section 8.

	Train Accuracy	Test Accuracy
HOG + MLP	97%	83%
HOG + kNN	89%	80%

Table 2: *Test accuracies obtained using HOG features, using an MLP and k-nearest neighbors algorithm. The perceptron is composed of 4 layers of 1728, 256, 150 and 107 neurons respectively.*

Model	Test Accuracy
1st ResNet-50	82.1%
2nd ResNet-50	79.4%
3rd ResNet-50	80.2%
Ensemble Voters	**83.9%**

Table 3: *Test accuracies obtained with individual classifiers based on ResNet-50 as well as with a soft voter ensemble. The accuracy of this last voter is slightly better, resulting in a better classifier overall.*

7.4 Layer Removal for embedding exploration capabilities

As mentioned in [21], it is generally thought that bottom layers of pre-trained networks contain general information applicable to different datasets and tasks, while upper layers of the pre-trained network contain more abstract information relevant to a specific dataset and task.

We therefore tested the removal of upper VGG-16 seeking for a boost in performances that we indeed obtained. The main drawback of this method is the difficulty to know in advance the optimal number of layers to remove, which depends not only on the task you want your system to learn but also on your available data. We therefore performed an exhaustive grid search to find the optimal number of layers to remove from a pre-trained VGG-16 network with important image recognition knowledge in it, which then forward-passed each one of the images we trained on. The data from Versace was split into train (80%) and test (20%) datasets, and normalized before feeding it into a k-nearest neighbors algorithm to make predictions for the five main labels each image could belong to (similar to masterCategories in Kaggle dataset). The possible labels were: shoes, bags, wallets, belts, and small leather goods. As a first step towards the FSL task, we also tested the influence of the number of images per category the system should get in order to improve its capabilities. The obtained results depended on the particular images we chose for each of the five layers, so a 20-seed average was done in order to obtain the expected accuracy for each possible configuration. These results can be found on figure 15. The low accuracy obtained with only five possible labels compelled us to using different techniques than using pre-trained extractors and fine-tuning them to obtain useful embeddings of our images, as the ones presented in section 8.

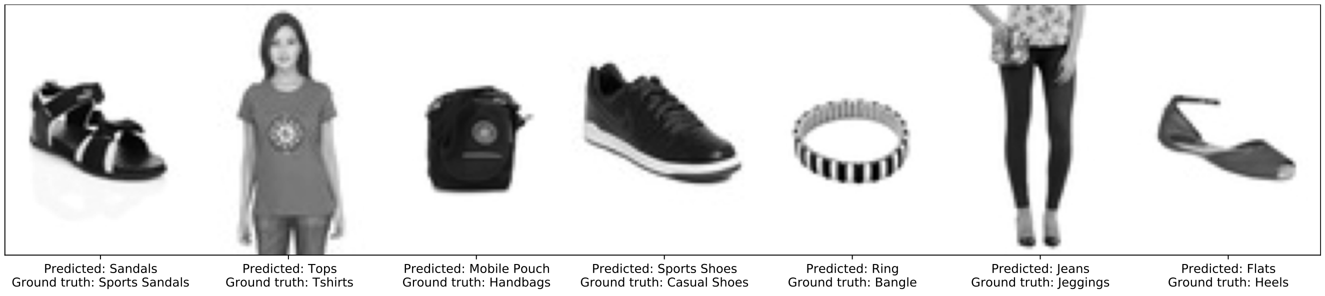


Figure 14: *Some examples of the mislabeled images using HOG extractors and k-nearest neighbors algorithm: as can be seen, the system seems close to human performance, and some of the predictions are difficult to distinguish from human ones.*

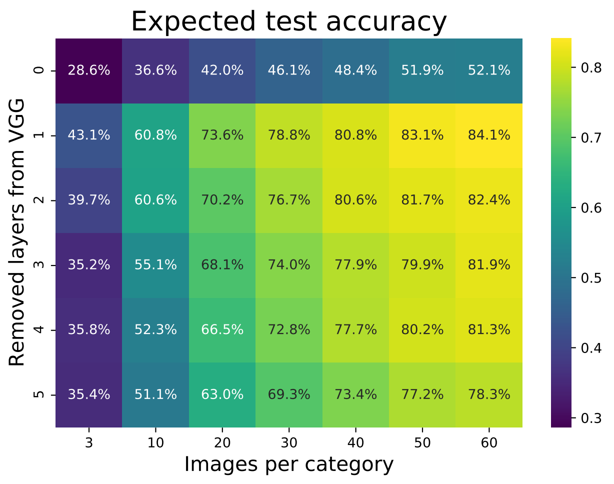


Figure 15: *Test accuracies obtained using normalized VGG embeddings of Versace images, removing some of the last layers of VGG and setting the number of images per target label. For each combination of K and N, 20 tests were made to average the accuracies that are highly dependant on the selected labels and images. As could be expected, the accuracy improves with the number of images per class, and it seems that removing VGG's last layer produces the biggest performance gain.*

8 Experiments and results

8.1 Methodology

For the retrieval task, we fixed our base set and test set to be able to evaluate our performance consistently. We used all images from all brands except a subset of Versace and all Givenchy images to train our models on. This provided us with 2 evaluation sets : the first contained images of 50 articles from Versace, drawn from a distribution of images closer to the one used for training. The second exclusively contained images from 309 unseen classes. This aimed at replicating two possible use-cases for Navee: one where new articles from an already-known brand are launched (Versace's case) and one where a new brand appears on the market and their articles need to be classified. Also, we made sure to keep the base set and the evaluation set disjoint to avoid any bias.

We trained our systems using batches of roughly 250 images, setting the number of pairs or triplets of images used to train during a full epoch at 10000. These pairs/triplets were generated on the fly when training, picked at random and changed at every epoch. When loaded, data augmentation techniques were applied to the images (random horizontal flips, as well as rotations, random crops) to get as diverse combinations of images as possible. Most of the tests we conducted were run on a GTX 1070 Ti GPU, but we also used Google Colab and Google Cloud services for training our systems. The different experiments that we conducted spanned over more than 340 hours of computation time, as each epoch could last between 7 and 10 minutes to be finished, depending mainly on the hardware and on the amount of data augmentation. We also set pseudo-random generator seeds for to enable proper comparisons between different systems.

During training, weights were saved and mean average precision was computed on each of the two validation datasets after each epoch. This allowed us to monitor the improvements our networks were making during train time, and to determine the optimal number of epochs to train on.

8.2 Different extractors with contrastive siamese network

We have tested both using the initial convolutional extractor presented by Gregory Koch for one-shot image recognition in [6], illustrated in figure 17, as well as using ResNet-50 and ResNet-18 pre-trained extrac-

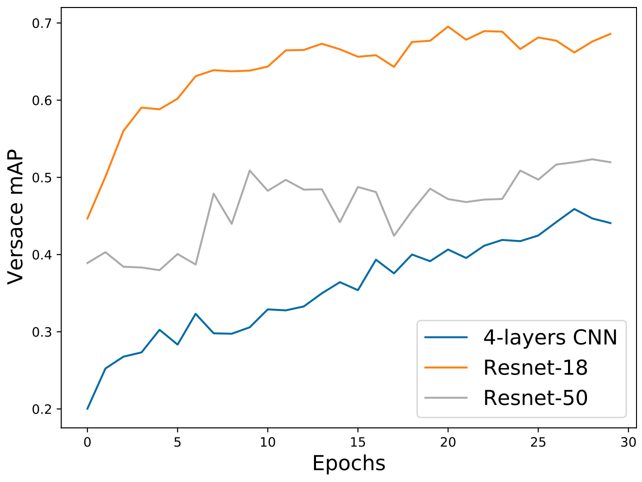


Figure 16: *Mean average precision on Versace computed at each epoch during 30 epochs of training that suggests the relevance of using middle-sized but still complex enough ResNet-18 as the main feature extractor for siamese systems.*

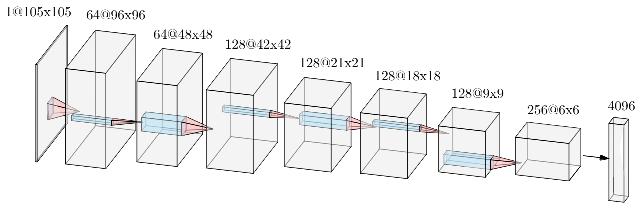


Figure 17: *Convolutional neural network presented by Gregory Koch et al. in [6] for one-shot image recognition. This network is composed of 4 convolutional layers, as well as max-pooling layers, and embeds any image into an $\mathbb{R}^{4096}$ vector. This is used as the CNN extractors before computing the contrastive loss on the embedded images.*

tors to compare performances. The results we obtained showed that deeper and more complex networks did boost the retrieval performance of our systems, but fine-tuning larger networks such as ResNet-50 was difficult and did not provide sufficient performance gains. We empirically proved that for our use-case, a shallower CNN such as ResNet-18 performed better. The comparison of these results is available on figure 16. It is worth mentioning that using different feature extractors forced us to using different sizes images: the 4-layers CNN presented in [6] took $105 \times 105 \times 3$ images as input, while ResNet was fed $224 \times 224 \times 3$ images, which helped in improving the performances.

8.3 Hyperparameters influence

Our implementation of siamese networks relied on several important parameters that were tested here in order to try to improve the performance of our systems. The different tests here presented were done by trying for each of the parameters different values at a time, letting the others unchanged. Setting the pseudo-random generator's seed ensured the reproducibility of our different tests, through the consistency of the initialization of random weights and of the order on which couples/triplets were sampled from the data. As these tests were computationally intensive, we decided to run them on the smaller convolutional network presented in 17, to allow us to iterate quickly. We assume here that the similarity of the systems using siamese networks can allow us to generalize from the data reasonably well and to apply the obtained knowledge to ResNet-based siamese networks. Some final tests were however done on ResNet-18 networks. The parameters we focused on were:

- the learning rate λ of the network
- the proportion p_{same} of positive pairs, containing images from the same category: when using contrastive loss, image couples can either come from the same class ($y = 1$) or be drawn from different articles ($y = 0$). This problem does not occur when using triplet loss since each triplet sample contains an anchor, a positive and a negative image.
- the margin m used in contrastive and triplet loss
- the proportion p_{mining} of hard examples from each batch the network is trained on at each epoch: selecting the data points on which the networks struggles the most, as presented in section 5, can be very helpful during training.

In addition, we conducted experiments with more complex models using ResNet to check the influence of training our networks on different datasets, and other techniques inspired from section 4 were only the bottom-end layers of the pre-trained extractors we used were unfrozen.

For the sake of readability, all figures mentioned in the following sections are available in the appendix section of this report, but can be accessed through the links.

8.3.1 Learning rate

When using stochastic gradient descent, the updates of the weights of the model are made using a parameter called the learning rate that controls how much to change the model after calculating the error of the system. Choosing it small enough ensures the correct convergence of the system to a minimum, but the smaller the learning rate, the longer the network takes to get there. Increasing the learning rate can therefore improve the speed of the system but make it more prone to unstable training processes and thus damage the overall performance. Hence, it is an important parameter to test when configuring neural networks. The base model used a learning rate $\lambda = 5 \times 10^{-5}$, and we tried different values in log scale from 10×10^{-6} to 10×10^{-4}. The results are shown in figures 25a and 25b, and have allowed to validate our initial choice of $\lambda = 5 \times 10^{-5}$.

8.3.2 Proportion of positive pairs

We explored the influence of changing the proportion of couples of images coming from the same label ($y = 1$) in our train set, as we thought this parameter could influence how the embeddings could be created by our contrastive loss networks. As can bee seen in figures 26a and 26b, this parameter surprisingly seems to have very little influence on performance, and was therefore set, as in the base case, to $p_{same} = 0.4$. In fact, it is important to make sure that the proportion of positive pairs is not too small to allow the network to learn how to map images from the same class. Typically, pairs cannot be drawn at random and should be selected carefully.

8.3.3 Margins

The margin m used in both triplet and contrastive loss appeared as a parameter that we could adjust to try enhance the performance of our embedding system. Intuitively, setting a greater margin would ensure greater separation of clusters in the embedding space. The results shown in figures 27a and 27b suggest that, as with p_{same}, the margin has no significant influence in the performances of our system. We conjecture that the influence of the margin is highly dependant on the dimension of the embedding space. Further experiments should have been made to check how these parameters influence together the performance of our systems. We therefore validate our initial choice of $m = 2$.

8.3.4 Hard mining proportion

As mentioned in section 5, selecting hard examples to train on in each batch can improve the performance of the system: we adapt the weights of the network only considering the examples on which the representation is not satisfactory, ie. those which have a non-zero loss. We have thus explored the proportion p_{mining} of images the network is trained on, sorted from highest to lowest loss values. As an example, a value $p_{mining} = 0.2$ means that the network will update its weights only by considering the worst 20% of images from each batch, i.e. the 20% of samples where the loss is the greatest. The results we obtained, represented in figures 28a and 28b, show that choosing $p_{mining} = 0.5$ ensures obtaining a good compromise between loss convergence and retrieval performances on this dataset for a batch of 200 pairs.

8.3.5 Embedding space dimension

Images are projected to $\mathbb{R}^d$, and as mentioned for the margin, the dimension of this space could have an influence on how useful the embeddings are for the retrieval task. We explore in figures 29a and 29b how this parameter influences retrieval performances, and conclude that $d = 256$ could provide the best performance possible, as well as reducing the computational costs of projecting images to larger spaces.

8.3.6 Fine-tuning ResNet

After using the smaller CNN presented in figure 17 to perform the previous tests, we decided to use ResNet-18 as a more robust extractor, as suggested by figure 16. We checked on how many residual blocks of ResNet should be unfrozen during training to improve the performances. As can be checked in figures 30a and 30b, a major gain in performances is obtained by allowing a full fine-tuning of the network, which will be done for the final model.

8.3.7 Training data: more is better

We compared using only Versace images to train siamese networks as well as using all the data available to check whether this last scenario was providing a substantial gain of knowledge to our system. Results shown in figures 31a and 31b confirm the usefulness of training the system to larger and more diverse datasets, even when the smaller dataset comes from a closer distribution to Versace validation test. In fact, it seems that training on additional data, that does not necessarily come from the same distribution as the evaluation data, can still be useful and bring a slight boost in performance.

8.4 Retrieval task final results

The final parameters of our networks, as discussed in 8.3, have been selected checking the retrieval performance of our siamese networks on Versace support/query datasets. These datasets are however close to the base set, since images from Versace have been already shown to our network during train time. We therefore consider Givenchy support and query datasets, that contains 309 unseen classes, much closer to a real possible application of this system. We therefore explore how the knowledge gained on other brands can be translated into a completely new dataset.

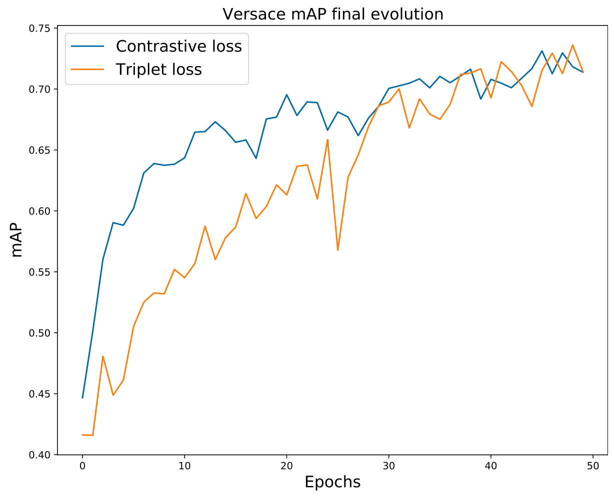
Having selected the final parameters, we trained our networks during around 100 epochs to gain as much knowledge as possible. We reduced the learning rate during training to improve the convergence time, dividing it by 10 after each 30 epochs for contrastive loss. Our final results are shown in figures 18a and 18b. Table 4 gathers the final results for the retrieval task. As can be seen from the figures, the retrieval task on Givenchy is considerably harder.

This unsatisfactory performance might be improved with a better selection of what pairs (or triplets) are used to train. In our implementation, we selected our training samples randomly, not ensuring that every article was suitably represented in each batch or each epoch. As suggested in [18], it could be valuable that within a mini-batch, each article that is present appears in at least a minimal number of training samples (40 for [18]). Additionally, to ensure that every article's representation is properly learned during training, it would be beneficial to select triplets in such a way that every class appears during an epoch. Finally, the proposition of [22] to fix batches for 5 epochs might be interesting to ensure that the network learns an appropriate representation of these images.

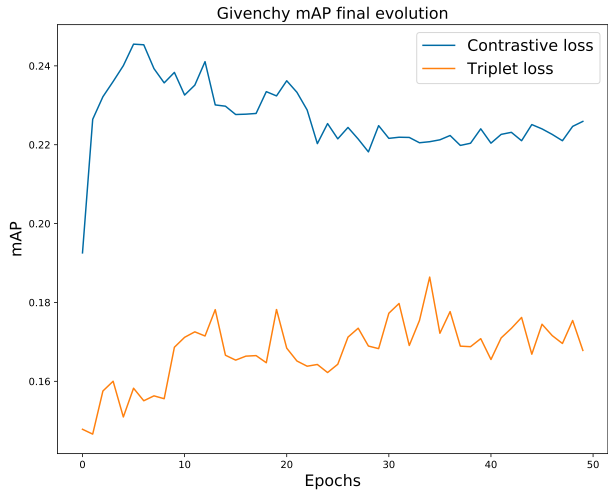
Siamese model	Versace mAP (N=50)	Givenchy mAP (N=309)
Contrastive loss	0.711	**0.217**
Triplet loss	**0.714**	0.168

Table 4: *Final mean average precision results on Versace and Givenchy datasets. N represents the number of categories considered in each dataset.*

Still, we provide representations which suggest that our approach is indeed effective. Figures 20a and 20b display the distribution of the distance between pairs of images from each evaluation dataset, when using an embedding function based on pre-trained ResNet-18 and when using an embedding function trained with our triplet loss method. For each dataset, we sample all possible pairs of distinct images and compute the Euclidean distance the embeddings of each pair. When using the pre-trained version of ResNet-18, we use PCA to reduce the output dimensionality of the CNN from 512 to 256. In the top plot, we see that the representation of images is far from optimal as positive pairs of images are not particularly close to each other in the embedding space. After training, the overlap between both distributions is somewhat reduced, which suggests that the learned representation of images is closer to what we want for the image retrieval task. In addition, some particular examples are presented in



(a) *Versace support set*



(b) *Givenchy support set*

Figure 18: *Obtained results of our final models on Versace and Givenchy support datasets. Only first 50 epochs are plotted, but contrastive loss network was trained for another 40 epochs.*

figure 19, as well as the euclidean distances computed in the embedding spaces.

Finally, the embeddings of PCA, projected in a 2-dimensional space using PCA are represented in figures 32 and 33, which once again suggest that our training favored the creation of clusters in the embedding space.

In 8.5, we will explore how these two systems can be used for the few-shot classification task and how they perform.

8.4.1 Prototypical Networks retrieval results

Concerning prototypical networks, we did not manage to obtain the level of performance obtained with siamese networks. The methodology used to train this network was different: for each of the selected brands, we split the data we had for it in a train dataset (70%) and test set (30%). We trained prototypical networks on Versace, Burberry, Dolce&Gabbana and YSL, and results are available in table 5. As not every brand had at least 5 images per label, we did not manage to compute the last values for mAP. This could have been done, with more time available to work on this model, using data augmentation to produce new images from the available ones.

mAP (%)	**20-way 1-shot**	**60-way 1-shot**	**60-way 5-shot**
VERSACE	26.46	24.97	28.29
Burberry	10.81	10.59	10.60
D&G	15.83	16.30	*None*
YSL	29.83	29.14	*None*

Table 5: *mAP obtained using Prototypical Networks for several available brands on different FSL parameters. Missing values can be attributed to a lack of data for some articles on particular brands, but could have been avoided by using data augmentation techniques. Yet, the lack of promising results and the computational intensity of the task kept us from testing this model to its full potential.*

Exploring prototypical networks was an opportunity for us to test a popular few-shot learning technique, but as retrieval results for this architecture was not promising enough, we focused on few-shot classification using MAML and siamese networks, explored in 8.5.

8.5 Few-shot classification task

This section explores to what extent the models we obtained help in the few-shot classification task. We evaluate the performance of our models through the

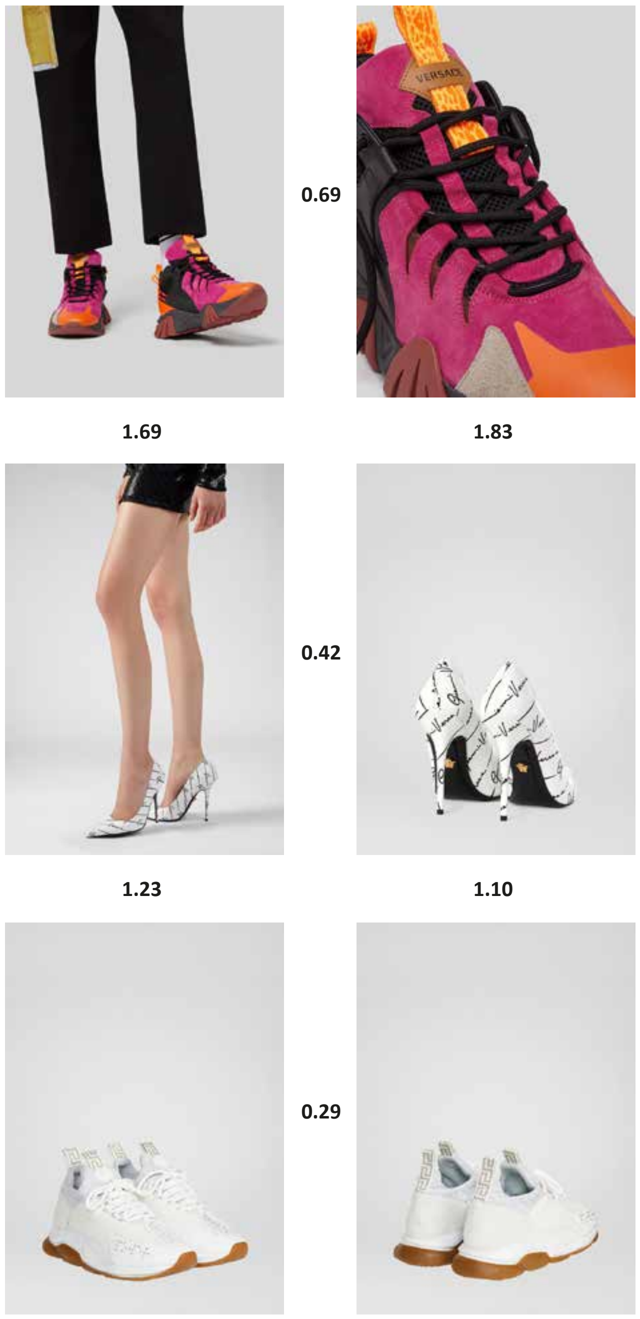
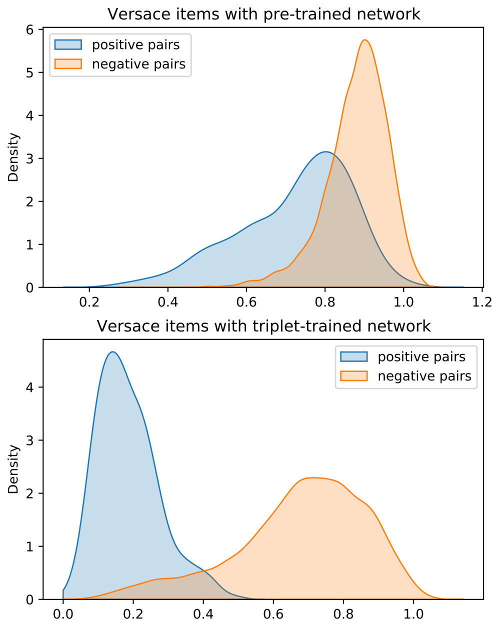
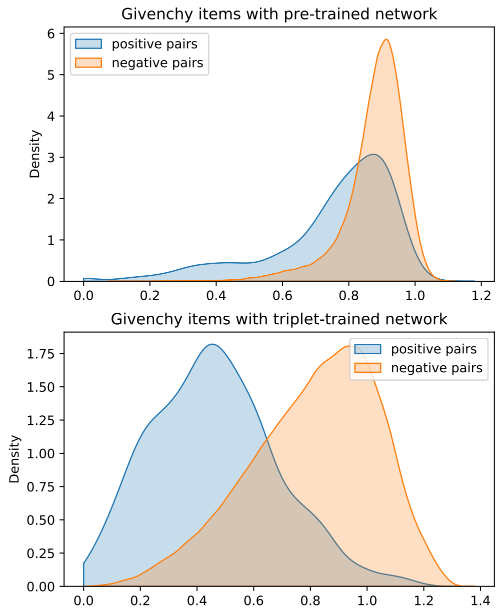


Figure 19: *Pairs of images from three randomly selected Versace articles, as well as euclidean distances computed between their representation in the embedding space. As can be seen, a threshold of 0.7 would classify correctly every article.*



(a) *Versace support set*



(b) *Givenchy support set*

Figure 20: *Distribution of distance between positive pairs of images (same label) and negative pairs of images in the embedding space.*

accuracy metric in a N-shot K-way classification task, for a variety of values for N and K. As explained in section 5, we used two approaches for this task. The first involves the use of kNN on the embeddings computed with our siamese networks. The second is based on a simple perceptron that takes embeddings as input, before giving a class as its output.

Results for the first approach are available on figures 23 and 24. For each combination of N and K, we compute the embeddings of all K images for every label, using random data augmentation when K is greater than the actual number of images available for a given class. After this, images from the query set are given a label by applying a k-Nearest Neighbors classification algorithm. Here k was set to 1 after an examination of possible values suggested that the best performance was obtained with $k = 1$.

The approach that used an MLP to classify demanded considerably more work. In order to find a performing architecture, we ran grid searches on the parameters of this MLP : learning rate, learning rate decay, regularization parameters and dimension of the hidden layer, Additionally, as mentioned before, we used PCA as a dimensionality reduction technique to adjust the size of the input. We therefore inspected various values for this input dimension in order to get the best performance we could from that model. These grid searches were conducted using only a fraction of all query images, so as to be able to evaluate our optimal parameters on unseen query images and thus have a more realistic estimation of our performance. Finally, we ran several simulations with different seeds for our model with optimal parameters to further ensure the reliability of the evaluation scores.

To give a better idea of the general performance of our models on various N-way-K-shot tasks, in the following paragraph, we use an arbitrary evaluation metric that is the average the accuracies over all tasks considered in the previous heatmaps (figures 23 and 24). On average, our networks have boosted by 23% the classification performance of our network on Versace's unseen data, going from 60% average accuracy for a baseline model that use pre-trained ResNet-18 as as its embedding function, to 83% for our siamese models (83.1% for contrastive, 83.6% for triplet). While the approach that used perceptrons to classify slightly affected the overall performance (82.1% accuracy), it did bring a small improvement to the performance on some tasks, particularly when there was between 3 and 5 training examples. On the Givenchy retrieval task however, our systems only improved by around 10% the accuracy of off-the-shelf ResNet-18, that had on average 38% accuracy that was improved into 47% for the triplet-trained network and 52% for the contrastive one.

Also, we tried using an ensemble learning approach by combining embeddings from both the aforementioned siamese network models. The strategy we used consisted in concatenating both 256-dimensional embeddings of an image to create one 512-dimensional embeddings, which could then be used for classification. Once again, it was possible to use either a kNN-based strategy or a simple perceptron for the classification, possibly relying on PCA to pre-process embeddings. For lack of time, we only performed kNN-based classification on these new 512-dimensional embeddings. The results we obtained for this are displayed in figures 21a and 21b and are not conclusive since the average accuracy scores on Versace and Givenchy are respectively 82,3% and 49,0%. Yet, this model still outperforms some of our former models on some tasks, and can therefore be considered for these tasks. It is also important to note that further exploring some of the ideas we proposed just above might bring a slight boost in performance for that method.

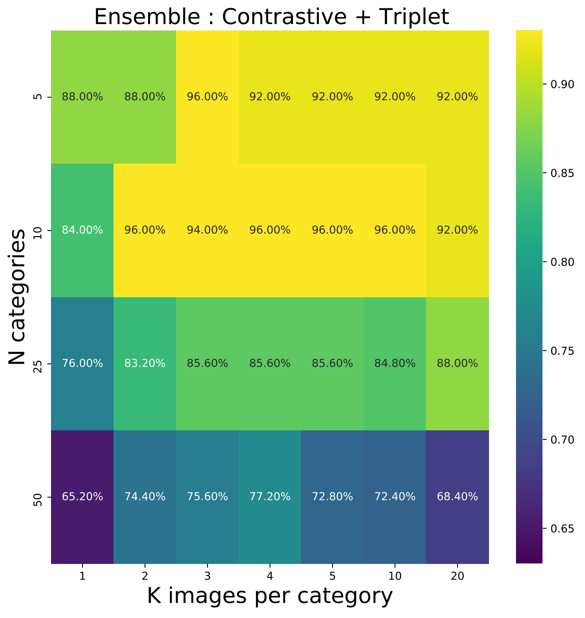
Overall, it seems that none of the models we developed is better than any other at all tasks, and consequently, it would make sense to use the heatmaps that we provided to select the best model when confronted to a particular N-way-K-shot problem.

Regarding MAML, the results weren't as impressive as the ones achieved by other architectures presented in this paper, such as HOG extractors and siamese networks. However, the performance for FSL is notably better than some baseline performances for usual classification techniques, like ResNet-18. For performance reasons and hardware-related issues, the model wasn't trained to the fullest potential, with this being a possible cause for the low performance. Furthermore, these issues also hindered the evaluation of the model on all the possible combinations of K and N, like it was done for other architectures. Figure 22 shows the N-way K-shot learning accuracies grid for this architecture, evaluated on the same Versace support dataset as the other models.

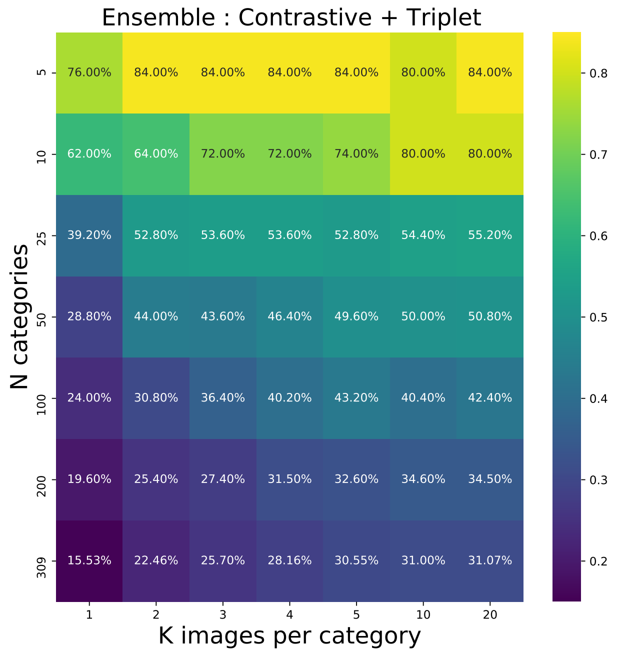
9 Conclusion

In this work, we explore computer vision techniques and propose the use of siamese networks using margin losses to perform few-shot classification of fashion images. Using few labelled images given by Navee, we conduct extensive experiments and give detailed analysis and visualizations showing that our model achieves superior retrieval performance to existing deep learning off-the-shelf methods.

In addition, we also explore popular few-shot learn-



(a) *Versace*



(b) *Givenchy*

Figure 21: *N-shot K-way accuracies obtained on Versace and Givenchy support datasets after averaging the accuracies obtained for each combination of N and K with ensemble learning. As each task depends on the selected labels, each classification task was performed 50 times, setting random numbers generator seeds to ensure comparability and reproducibility.*

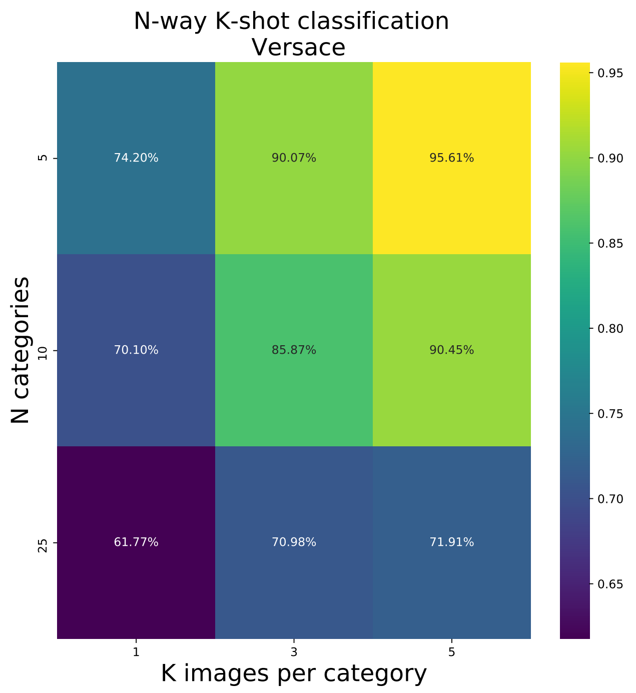


Figure 22: *N-way K-shot accuracies obtained on Versace support dataset for the MAML model after averaging the accuracies obtained for each combination of N and K.*

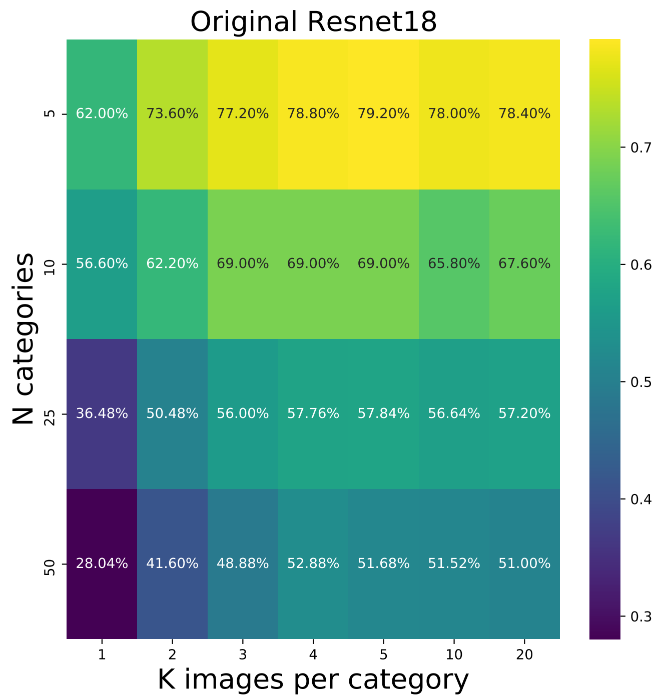
ing techniques like prototypical networks and meta-learning models and apply them to our classification task. In our work however, we only explore techniques that benefit from labeled data samples. Semi or self-supervised methods could leverage large amounts of unlabeled data available on the internet to produce proper embeddings by exploiting that additional and rich data. Further work could therefore be done in this direction, starting for example with variational auto-encoders, unsupervised systems that could benefit from large amounts of unlabeled images.

10 Source code

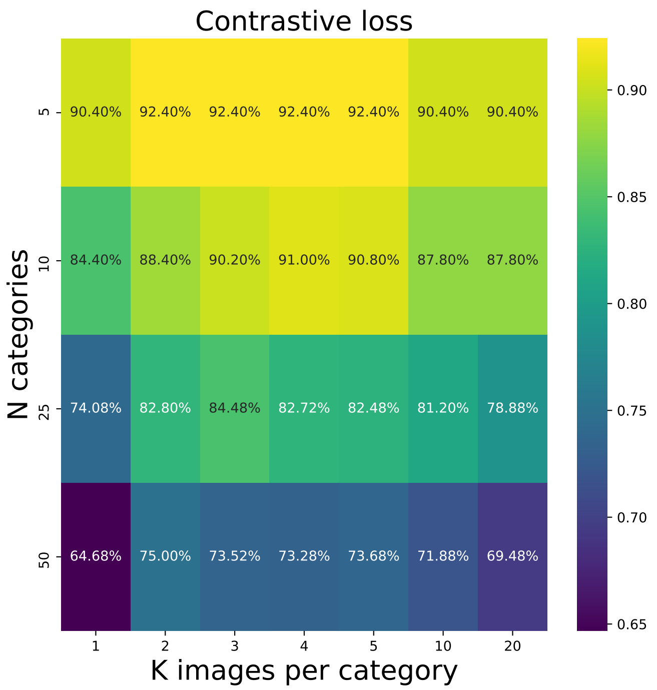
Most of the models we implemented were obtained using Pytorch framework, though other frameworks were occasionally used. A summary of our code, as well as some of the results and final models can be found here: `https://github.com/carlossantosgarcia/few-shot-classification`.

11 Appendix

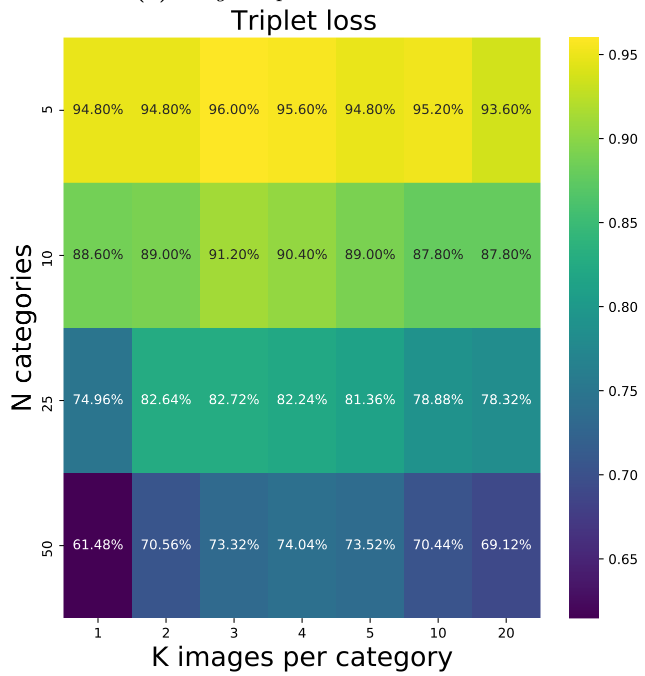
The next pages contain some figures displaying the obtained results during our hyperparameters search, as well as other interesting figures.



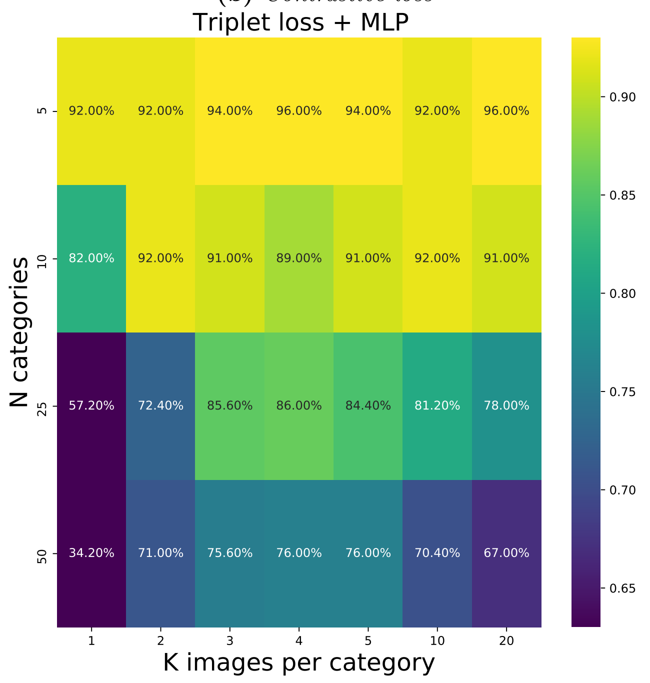
(a) *Original pre-trained ResNet-18*



(b) *Contrastive loss*

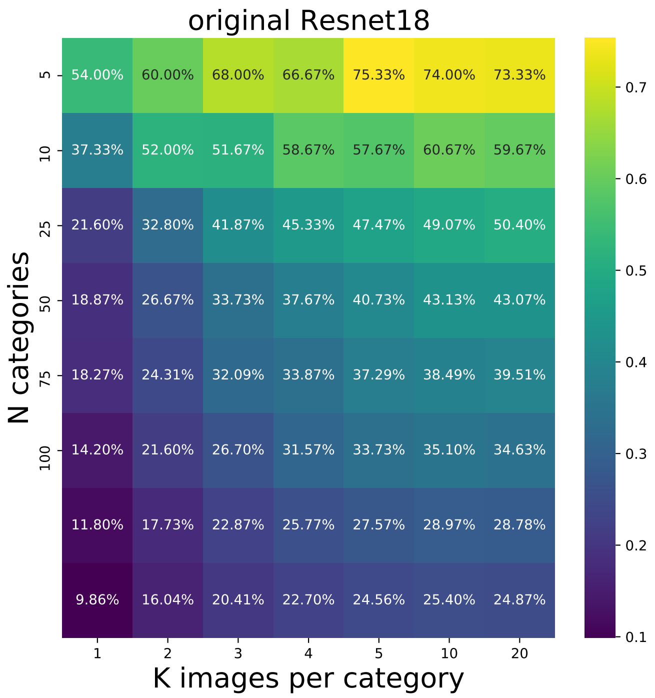


(c) *Triplet loss*

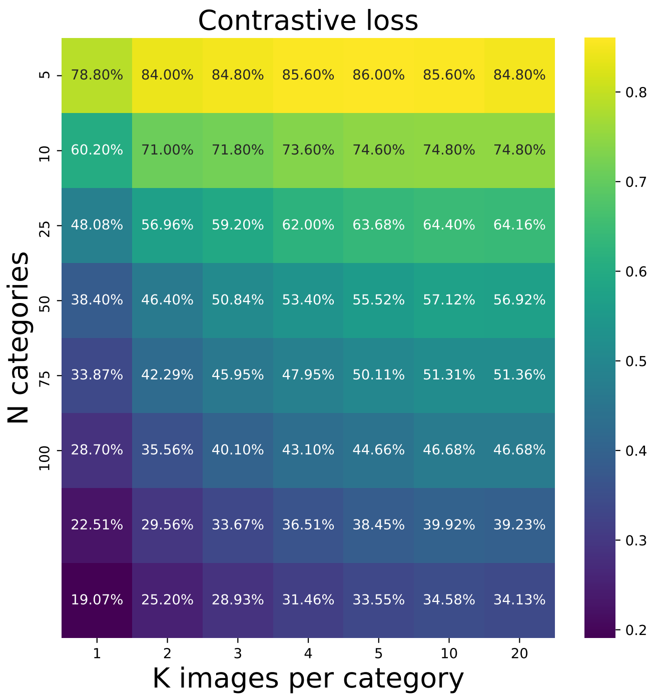


(d) *Triplet loss + MLP on embeddings*

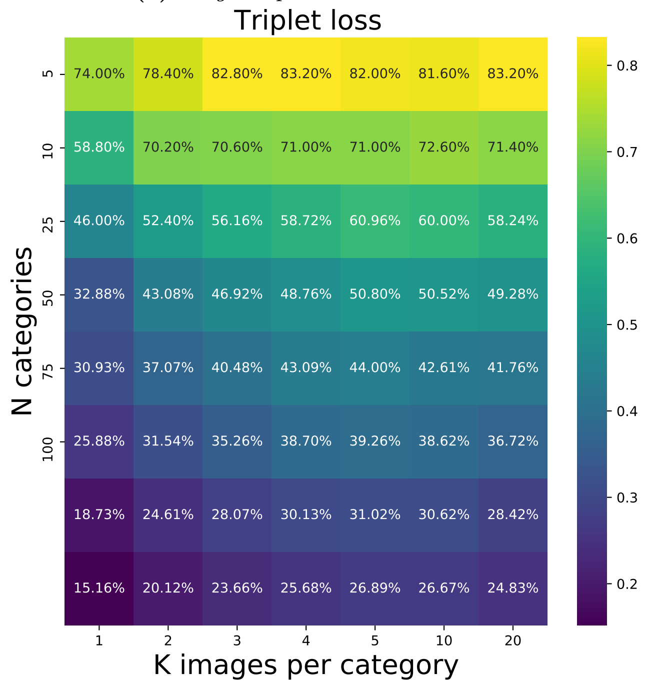
Figure 23: *N-way K-shot accuracies obtained on Versace support dataset after averaging the accuracies obtained for each combination of N and K. As each task depends on the selected labels, each classification task was performed 50 times, setting random numbers generator seeds to ensure comparability and reproducibility.*



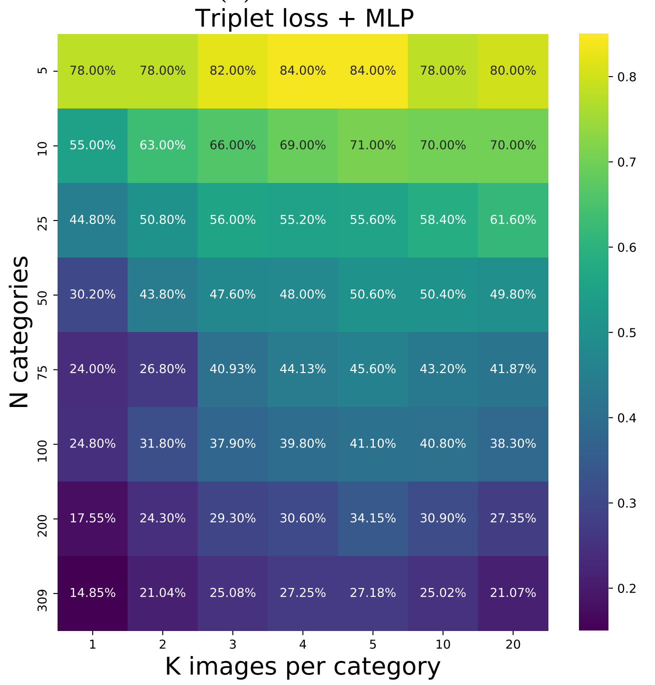
(a) *Original pre-trained ResNet-18*



(b) *Contrastive loss*

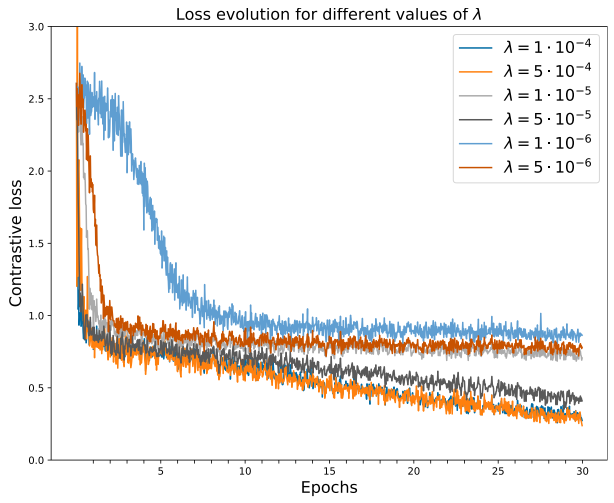


(c) *Triplet loss*

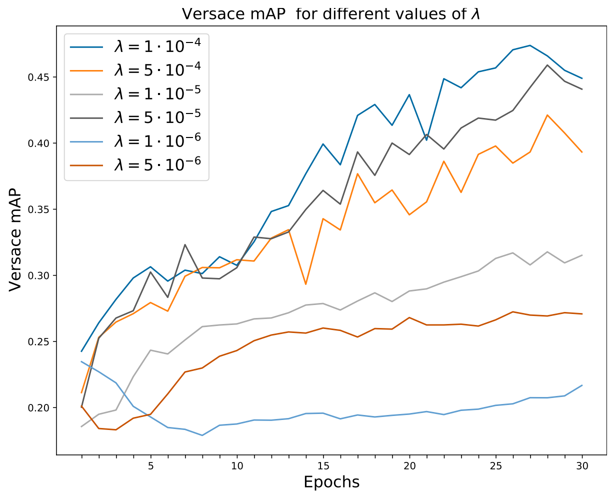


(d) *Triplet loss + MLP*

Figure 24: *N-shot K-way accuracies obtained on Givenchy support dataset after averaging the accuracies obtained for each combination of N and K. As each task depends on the selected labels, each classification task was performed 50 times, setting random numbers generator seeds to ensure comparability and reproducibility.*

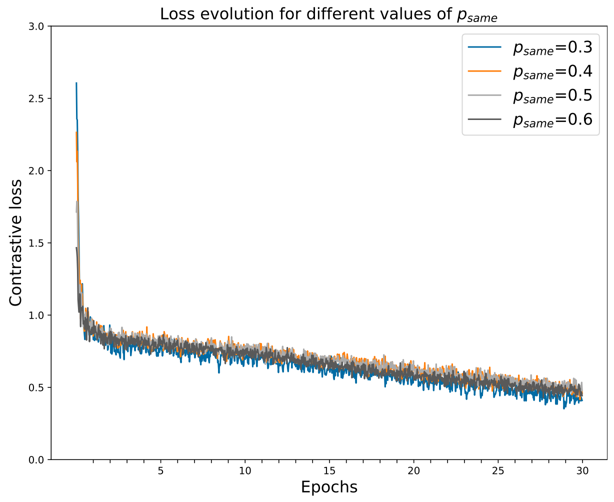


(a) *Contrastive loss*

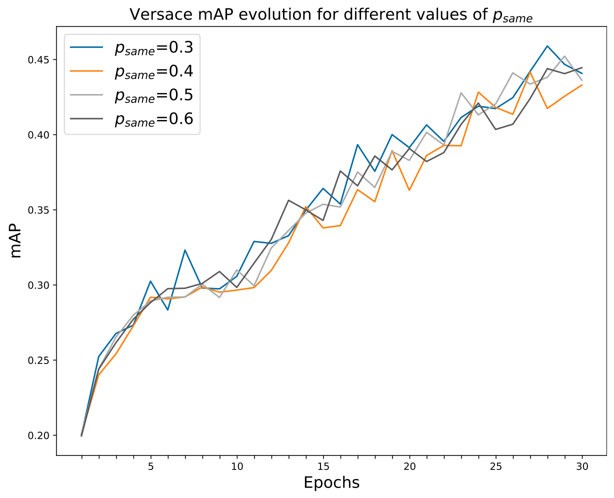


(b) *Versace mAP*

Figure 25: *Contrastive loss and mAP for our siamese networks during 30 training epochs with different values for the learning rate λ, all other parameters unchanged. This parameter seems to influence the performance of our models and suggests the use of $\lambda = 5.10^{-5}$ in order to get the best performance.*

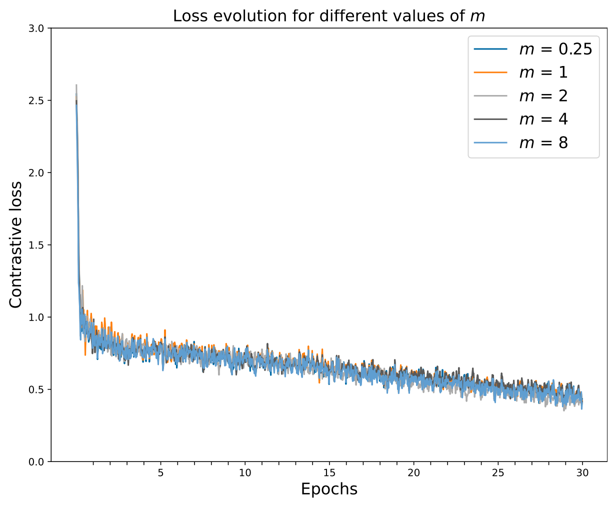


(a) *Contrastive loss*

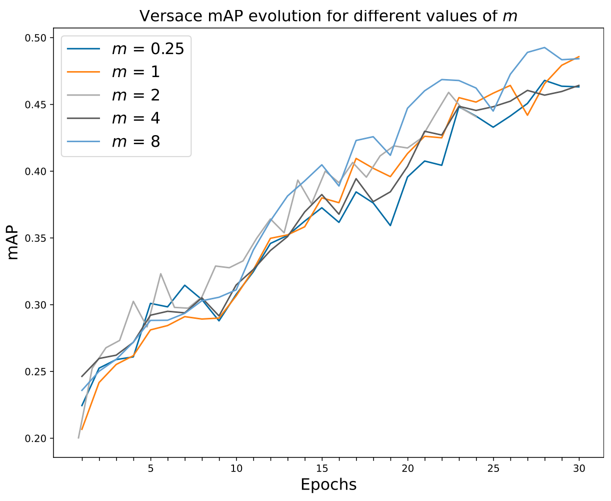


(b) *Versace mAP*

Figure 26: *Contrastive loss and mAP for our siamese networks during 30 training epochs with different values of p_{same}, all other parameters unchanged. This parameter does not seem to have a significant influence on performance.*

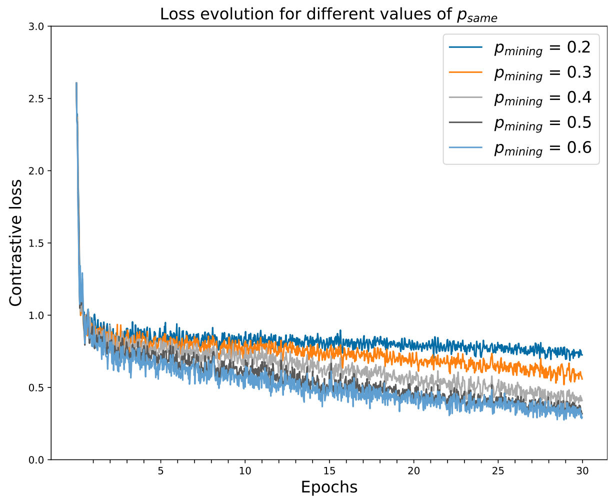


(a) *Contrastive loss*

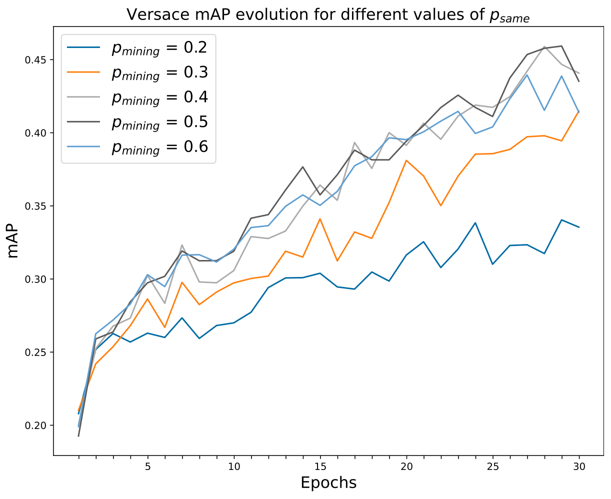


(b) *Versace mAP*

Figure 27: *Contrastive loss and mAP for our siamese networks during 30 training epochs with different values for the margin m, all other parameters unchanged. As with p_{same}, this parameter does not seem to have a significant influence on performance.*

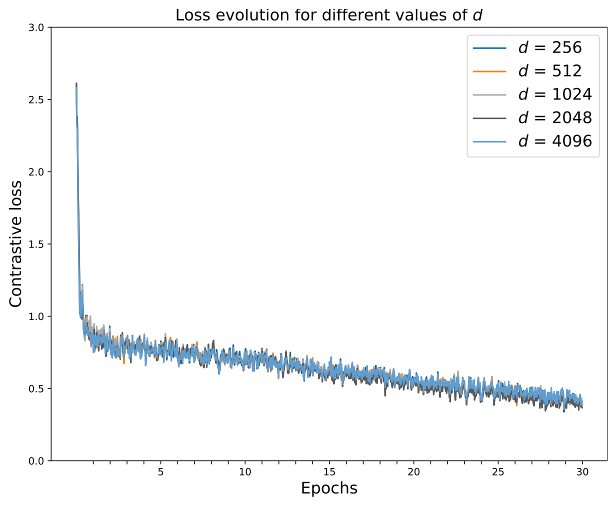


(a) *Contrastive loss*

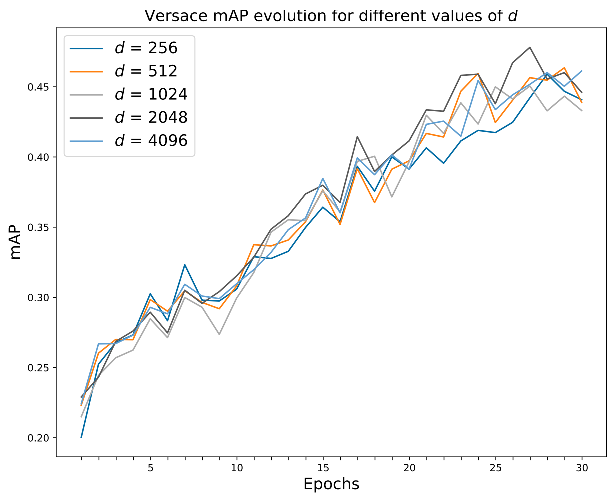


(b) *Versace mAP*

Figure 28: *Contrastive loss and mAP for our siamese networks during 30 training epochs with different values of p_{same}, all other parameters unchanged. As could be expected, the smaller p_{same}, the harder the task to train the network and the slower the decrease of the loss. As happened with p_{same}, this parameter seems to have no particular influence on the network performance.*

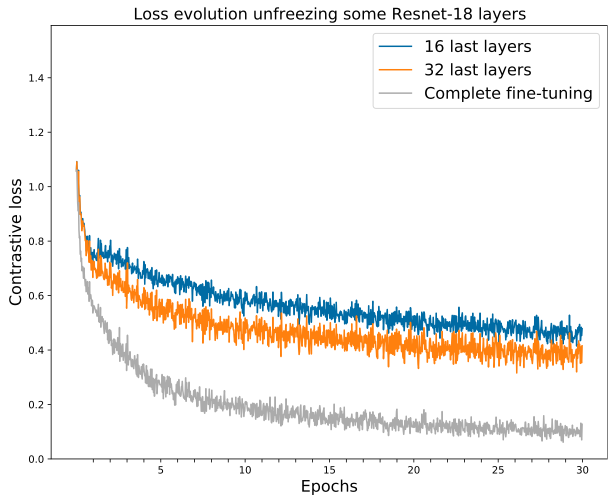


(a) *Contrastive loss*

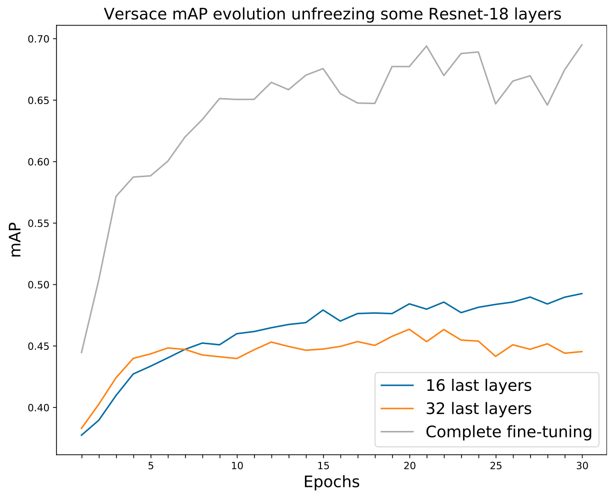


(b) *Versace mAP*

Figure 29: *Contrastive loss and mAP for our siamese networks during 30 training epochs with different values of d, all other parameters unchanged.*

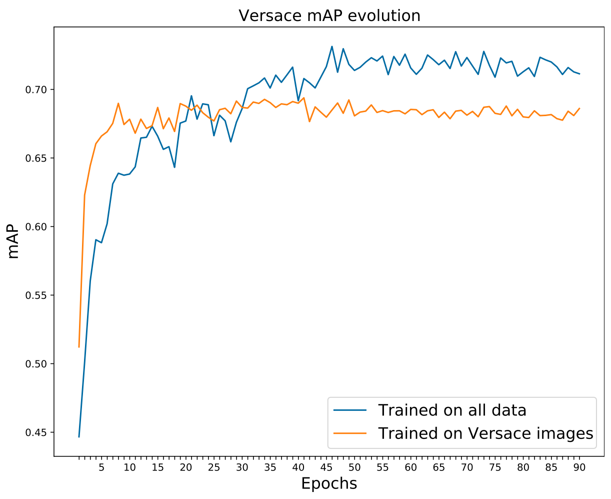


(a) *Contrastive loss*

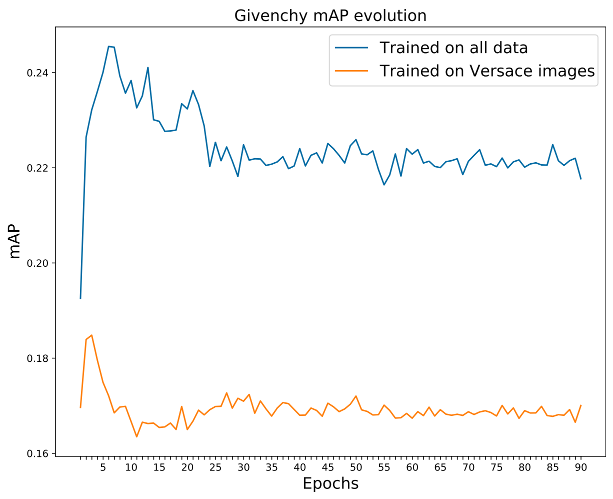


(b) *Versace mAP*

Figure 30: *Contrastive loss and mAP for our siamese networks during 30 training epochs with different ResNet-18 layers unfrozen, all other parameters unchanged. A major boost in retrieval performances is obtained by fine-tuning the complete network.*



(a) *Versace mAP*



(b) *Givenchy mAP*

Figure 31: *Comparison of the retrieval performances of siamese networks on Versace and Givenchy datasets (with 50 and 309 classes respectively) after being trained in different sized datasets.*

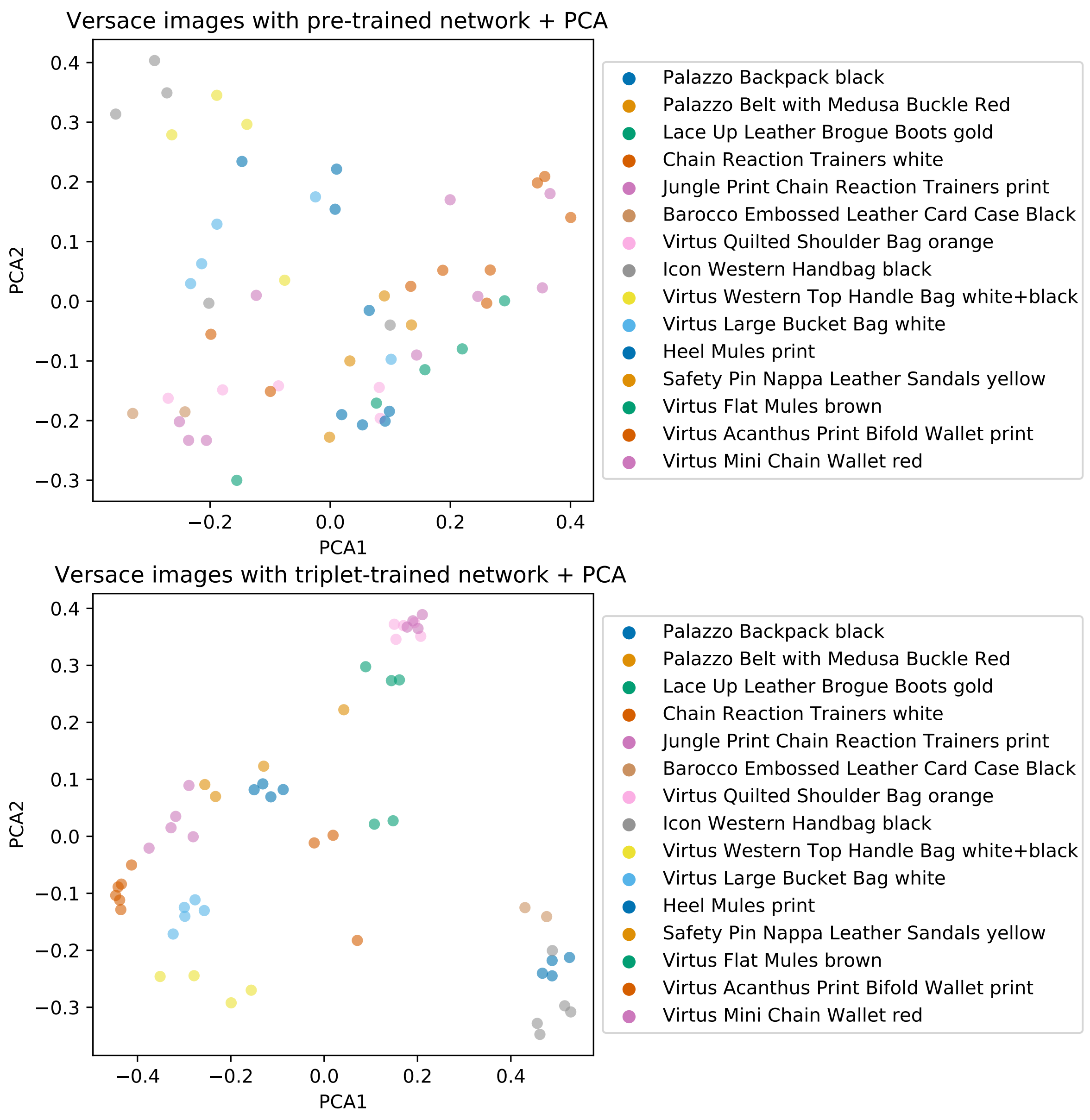


Figure 32: *Embedding of images from 10 random classes from Versace embedded with a pre-trained network (top) or a triplet-trained network (bottom), represented in a 2-dimensional space through PCA.*

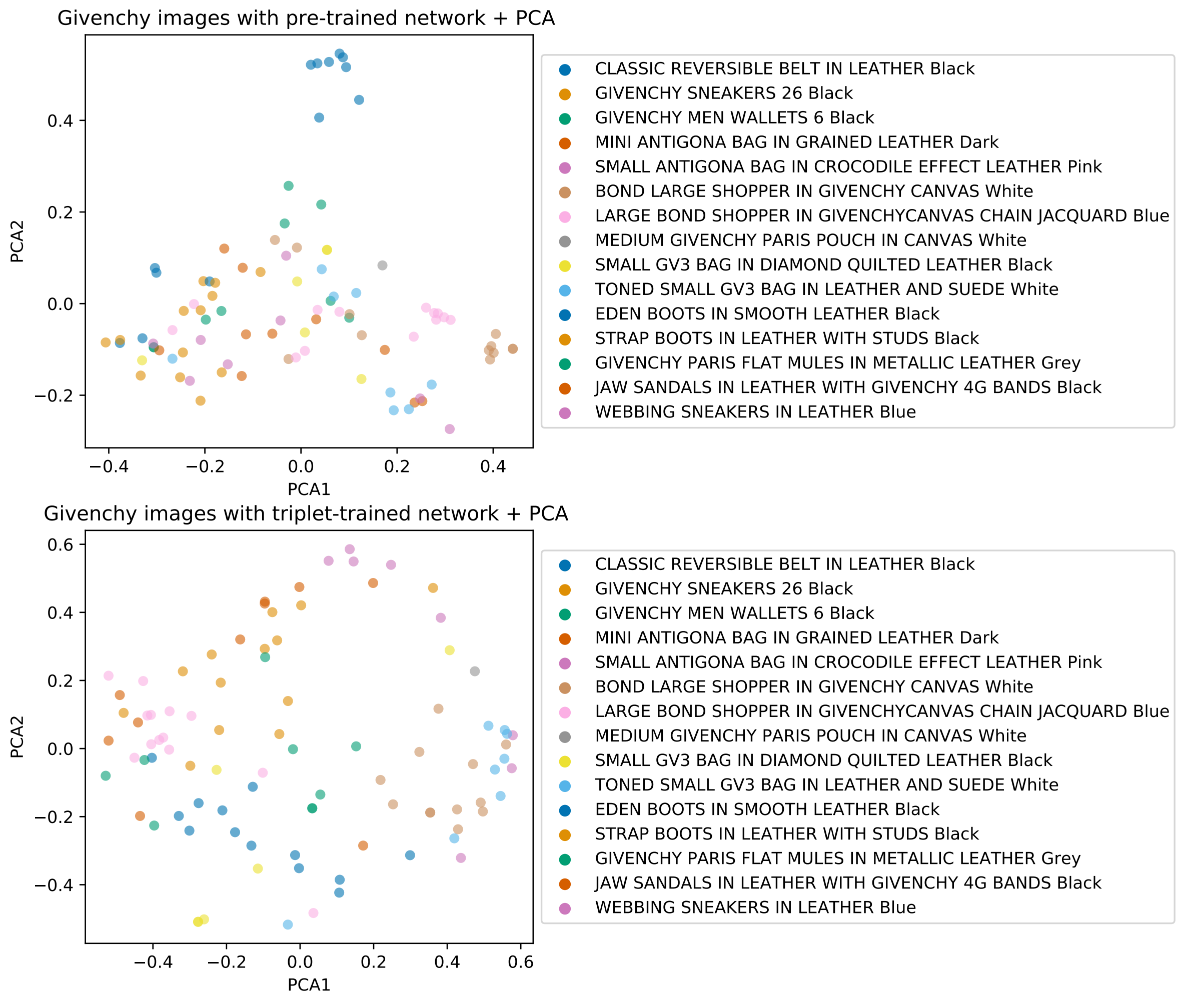


Figure 33: *Embedding of images from 10 random classes from Givenchy embedded with a pre-trained network (top) or a triplet-trained network (bottom), represented in a 2-dimensional space through PCA.*

References

[1] Wei-Yu Chen et al. *A Closer Look at Few-shot Classification*. 2019. eprint: arXiv:1904.04232.

[2] Connor Shorten and Taghi M. Khoshgoftaar. *A survey on Image Data Augmentation for Deep Learning*. 2019. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. 20112. URL: https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[4] Saman Motamed, Patrik Rogalla, and Farzad Khalvati. *Data Augmentation using Generative Adversarial Networks (GANs) for GAN-based Detection of Pneumonia and COVID-19 in Chest X-ray Images*. 2020. eprint: arXiv:2006.03622.

[5] Juan Luis Suárez-Díaz, Salvador García, and Francisco Herrera. *A Tutorial on Distance Metric Learning: Mathematical Foundations, Algorithms, Experimental Analysis, Prospects and Challenges (with Appendices on Mathematical Background and Detailed Algorithms Explanation)*. 2018. eprint: arXiv:1812.05944.

[6] Gregory R. Koch. "Siamese Neural Networks for One-Shot Image Recognition". In: 2015.

[7] Elad Hoffer and Nir Ailon. *Deep metric learning using Triplet network*. 2014. eprint: arXiv:1412.6622.

[8] Oriol Vinyals et al. *Matching Networks for One Shot Learning*. 2016. eprint: arXiv:1606.04080.

[9] Jake Snell et al. "Prototypical Networks for Few-shot Learning". In: *NeurIPS* abs/1703.05175 (2017). URL: https://arxiv.org/abs/1703.05175.

[10] Chelsea Finn et al. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *CoRR* abs/1703.03400 (2017). URL: http://arxiv.org/abs/1703.03400.

[11] Nikhil Mishra et al. *A Simple Neural Attentive Meta-Learner*. 2017. eprint: arXiv:1707.03141.

[12] Wei Chen et al. *Deep Image Retrieval: A Survey*. 2021. eprint: arXiv:2101.11282.

[13] Zewen Li et al. *A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects*. 2020. eprint: arXiv:2004.02806.

[14] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. eprint: arXiv:1409.1556.

[15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. eprint: arXiv:1512.03385.

[16] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. IEEE. DOI: 10.1109/cvpr.2005.177. URL: https://doi.org/10.1109/cvpr.2005.177.

[17] Yann Le Cun et al. *Dimensionality reduction by learning an invariant mapping*. 2005. URL: https://nyuscholars.nyu.edu/en/publications/dimensionality-reduction-by-learning-an-invariant-mapping.

[18] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: (2015). DOI: 10.1109/CVPR.2015.7298682. eprint: arXiv:1503.03832.

[19] Alexander Hermans, Lucas Beyer, and Bastian Leibe. *In Defense of the Triplet Loss for Person Re-Identification*. 2017. eprint: arXiv:1703.07737.

[20] Filip Radenović, Giorgos Tolias, and Ondřej Chum. *Fine-tuning CNN Image Retrieval with No Human Annotation*. 2017. eprint: arXiv:1711.02512.

[21] Weiming Zhi et al. "Layer Removal for Transfer Learning with Deep Convolutional Neural Networks". In: Nov. 2017.

[22] Jiedong Hao et al. "DeepFirearm: Learning Discriminative Feature Representation for Fine-grained Firearm Retrieval". In: *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, Aug. 2018. DOI: 10.1109/icpr.2018.8545529. URL: https://doi.org/10.1109/icpr.2018.8545529.