Computer Sciences, Master I
École Normale Supérieure de Lyon
Université Claude Bernard Lyon I

Internship 2016–2017
STÉRIN Tristan
M.Sc. in Computer Sciences

Intraclass Apparel Retrieval with Deep Neural Features

Abstract : *Retrieve an apparel in a brand's catalogue given a photo of someone wearing it is a challenging and little studied problem. This task involves dealing with very few data at train time and state-of-the-art Computer Vision methods such as Deep Convolutional Neural Networks (DCNNs) are originally not suited for this type of "small-data" problems. In this work, we adopt a transfer learning approach and use the feature extraction power of ImageNet-trained DCNNs to solve this lack of data issue. The Self-Organizing Map unsupervised method helps us to visualize extracted features in an original way. Finally, we proceed to apparel retrieval by comparing different methods. Our best technique, FCAvg, outperforms traditional non-DCNNs approaches such as ORB Bag-Of-Words by achieving 89% at top-10 accuracy while the latter doesn't go beyond 30%. Being also very fast at request time, our retrieval method is a ready-to-use pipeline for potential industrial apparel Catalogue Matching applications.*

Keywords : *Apparel Retrieval, Machine Learning, Transfer Learning, Feature Extraction, DCNNs, SOM, Kohonen networks, Catalogue Matching.*

Under the supervision of:
Gilles SABAS
A7 Emailing
57 rue Grimaldi - 98000 Monaco - Principality of Monaco

Marc SEBBAN

University Jean Monnet, St-Etienne
Hubert Curien Lab
18 rue du Prof.Benoît Lauras - 42000 Saint-Etienne - France
http://perso.univ-st-etienne.fr/sebbanma/

August 18, 2017

Acknowledgements

I would really sincerely like to thank my advisor, Gilles Sabas, CEO at A7 for having given me the opportunity, resources and environment to realize this work. I also would like to thank my co-advisor and former Machine Learning teacher Marc Sebban for all what I've learned in his course and the advice he gave me during this internship. Finally, a big thank to my parents and all those I met during these three months, Marie, Angela, Caroline, Luciano, Vincenzo, Tristan, Vincent, your presence helped me work and being happy.

Contents

Introduction

Figure 1: A query photo and its associated catalogue image

Let's suppose I have the photo of someone wearing a dress that I really like. I would like to know whether my favorite brand has a dress which is similar to it. This situation is illustrated in Figure 1. We aim at designing a system that takes such queries as input and returns the most similar apparel in the brand dresses' catalogue. Stated as so, our problem is an instance of Content Based Image Retrieval (**CBIR**, [35]). We call it **intraclass** because we do not try to differentiate pants from dresses but dresses between themselves.

Research on apparel classification and retrieval is a relatively new field [21],[2],[15],[12],[22]. Authors unanimely agree that it is a hard CBIR problem. Indeed, the representation of a garment between a query picture and a standardized catalogue one can be widely different ; whether it is because of the human wearing it or because of the picture's context, Figure 1 showing both. This implies that in order to be successful, the system we built has to ensure Cross-Domain Adaptation [12]. That is the property of adapting knowledge from the catalogue domain, on which we have information, to the query domain, on which we don't. The main contributions of this field are contemporary to the great successes of Deep Convolutional Neural Networks (**DCNN** or **CNN**) in Computer Vision and often relie on it [15],[12],[22]. Indeed, these neural architectures are really good at Cross-Domain Adaptation.

Deep Convolutional Neural Networks, introduced in [19], have revolutionized Computer Vision when they outperformed by far traditionnal methods at the 2012 ImageNet Large-Scale Visual Recognition Challenge (**ILSVRC**). ImageNet [4] is a huge database of images – more than 14M – with about 20k classes, decomposed into roughly 30 super categories such as amphibians, vehicles or music instruments. The goal of the ILSVRC is to construct a classifier upon this data which correclty labels input pictures. With further refinements of the 2012 AlexNet architecture [18], such as GoogleLeNet [27], VGG [26] or ResNet [10], the total error went from 15% in 2012 to 3% in 2015 whereas, before the CNN era, it was approximatively of 30%. Their performing that well on such heterogeneous data implies that CNN are good at Cross-Domain Adaptation.

However, these systems are successful at Cross-Domain Adaptation because they are fed with a huge amount of data and are thus confronted to different domains. We are exactly is the opposite case: we have only one example per garment, all issued from the catalogue domain. Being specific to one brand dresses catalogue also makes the number of classes small, about 200 compared to the thousands of items per garment type in existing datasets [22], [15]. These two aspects are the main characteristics of our work : to the best of our knowledge there is no pre-existing work in that field at such a small scale. Learning with small data is an open problem with research efforts at the moment.

In this work, we tackle down the lack of data issue by using CNNs as **feature extractors** then present a way to visualize these features thanks to the Self-Organizing Map (**SOM**) and finally proceed to apparel retrieval with various techniques.

1 Feature Extraction

In this section, we firstly present the dataset on which we are going to work and then we show how we are going to use DCNNs as feature extractors.

1.1 Our Dataset

Figure 2: One of our dataset's query with its ROI and its associated catalogue image

Existing articles on apparel classification have proposed large dataset for this task [21],[2],[15],[12],[22]. In particular, [22] and [15] gathered data for what they call "Street-to-shop" and which is similar to our problem. However:

1. These datasets do not focus on one brand's catalogue in particular but on a large collection of them whereas the spirit of our work is really brand orientated: we want to deal with one specific catalogue.

2. These datasets contain a lot of query pictures but without Region Of Interest (**ROI**) information. That is we don't know were the garment is localized in the picture. Brand-new methods, Mask R-CNN [9], also using DCNNs, have given striking results for that problem. They could be, in the future, integrated in our pipeline. Nevertheless, at the present moment it would out of the scope of this work. Furthermore, for an industrial application, we can suppose that the user provides the ROI at query time, for instance, by circling the garment.

Consequently, we need a dataset which both satisfies our *catalogue* constraint and where garments are easily localizable in query pictures. We mined our own[1] from H&M dresses website[2], it contains around 200 dresses. This brand is convenient because they present each dresse with both a catalogue photo and a model wearing it. Furthermore, the models are all centered and a simple handcrafted heuristic for the ROI works fine, see Figure 2. Enventually, the photos have a really good definition which is nice in order to deal with details that often characterize garments.

[1]The mining script is available here https://github.com/tcosmo/dresscode/blob/master/get_dataset.py. Please note that you might not mine the exact same as ours since H&M collections change frequently.

[2]This exact page: http://www2.hm.com/fr_fr/femme/catalogue-par-produit/robes.html?product-type=ladies_dresses&image=model&sort=stock&offset=0&page-size=400

1.2 Deep Convolutional Neural Networks

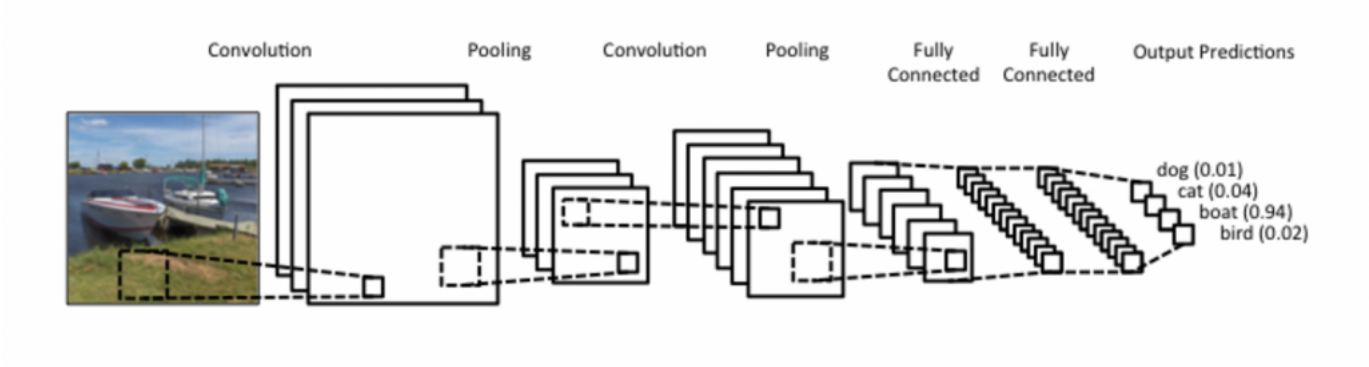

Figure 3: A traditional DCNN architecture

Deep Convolutional Neural Networks are a class of Artificial Neural Networks (**ANN**) that were originally designed to deal with images. They take images as inputs and process them through a succession of Convolution, Pooling and Rectification Layers. At the end of that process, a Fully Connected network, similar to the Multi-Layer Perceptron [7] (**MLP**), is learnt in order to achieve classification. This process is depicted in Figure 3.

Here's a brief summary of what each of these components does:

1. Convolution Layer (**conv**): it is the essential component of a DCNN. Each conv is a bank of parametrized filters which compute discrete convolution over its input channels. The parameters of these filters are learnt which makes the strength of DCNNs.

2. Pooling Layer (**pool**): pooling layers compress the information contained in a convolutionized channel. They act by taking the max or the average in neighbourhoods of their input channels.

3. Rectification Layer (**ReLu**): it is known since the 80' [11] that what makes the power of ANN is the use of non-linearities. In that spirit, ReLu layers act non linearly on their inputs, for instance by replacing negative values with 0s.

4. Fully Connected Layer (**fc**): at the end of the process the data is shrinked into relatively low dimensional – typically around 1000 dimensions – vectors, called **feature vectors**. These feature vectors, considered as abstract representation of the original images are the input of standard fully connected classifiers such as MLPs. The last layer of this MLP is often a softmax which gives a probability distribution over the classes in the dataset.

DCNN, like most ANN, are trained in a supervised manner upon a loss function describing how far are the DCNN predictions from ground truth labels. It is this supervision[3] process that requires a lot of data.
For a more detailled and technical introduction to DCNNs please refer to [5].

1.3 Feature Extraction with DCNNs

We have extremely few data in comparison of what it takes to train a DCNN. In order to use DCNN's power we are going to use them in a **transfer learning** setting [14]. That is to use a network that already has been trained on ImageNet in order to abstractly describe our images.

[3] Amusingly enough, the 2012 ILSVRC winner AlexNet model was called SuperVision.

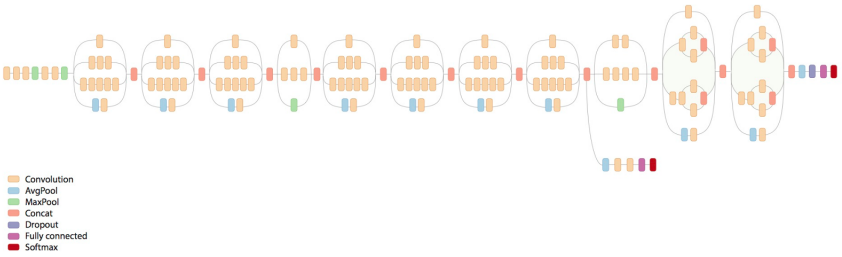

Figure 4: InceptionV3 Architecture

What we want to work with are exactly the **feature vectors** that appear just before the fc layer. Intuitively, a model that works well on ImageNet must produce feature vectors that abstractly describes any image. Furthermore, two images with close visual content should have close feature vectors. We use as a feature extractor the InceptionV3 model [28]. It is an improvement of GoogLeNet [27] which integrates complex design, see Figure 4. It is almost at the state-of-the-art on the ImageNet challenge and we can easily find pre-trained versions of it on the web [29]. We specifically use the `pool_3:0` layer which produces 2048 sized feature vectors.

Feature extraction with DCNNs for CBIR purpose has been explored by [34]. However they use features of the entire image. It is something we cannot do, especially for the query because our Inception model has seen humans while training. Consequently, even with the ROI specification, the features we get are those describing humans because we often see human attributes as hands or arms in the ROI. Also, intuitively, garments are often characterized by a very specific part of it. What we truly want are the features of the characteristic part of the garment. We need to know were to look.

We tried to develop the so-called KDRP approach described in [30]. It consists in randomly sampling zones with high SIFT [24] keypoints density. However it did not produce convincing results on our data. The solution we adopted, less elegant, is simply to take all the features of fixed-size overlapping sampling windows as shown in Figure 5. We are left with about 80 feature vectors per image (both catalogue and query). We call theses ensembles of features **features sets**. The use of GPUs, `2 x GeForce GTX 1080TI` has been crucial for the feature extraction process efficiency.

Figure 5: Overlapping sliding windows over catalogue and query image

2 Feature Visualization with The Self-Organizing Map

This part is not crucial in the apparel retrieval pipeline and could be skipped at first read. It exposes how the Self-Organizing Map (**SOM**) can help us visualizing the features we have extracted. To the best of our knowledge, there is no pre-existing work in our field using SOMs to visualize DCNNs features.

The Self-Organizing Map The Self-Organizing Map or Kohonen network is a neural model that has been introduced in the 80' by computer scientist Teuvo Kohonen [16]. Since the intrudoctory paper, the SOM has known a lot of refinements such as in [17]. A very extensive litterature on the subject, both theoretical and applied, has been written about the SOM [6]. The SOM is a very efficient tool for high-dimensional data visualization. It will organize on a 2D map high dimensional feature vectors based on their similarity. One of its main advantage is that it is an **unsupervised** method: no additional information than the raw data is needed for training. Surprinsigly, this model seems to be a bit out of fashion nowadays.

Our implementation of SOMs can be accessed both in the Appendix A and here.

2.1 The SOM Model

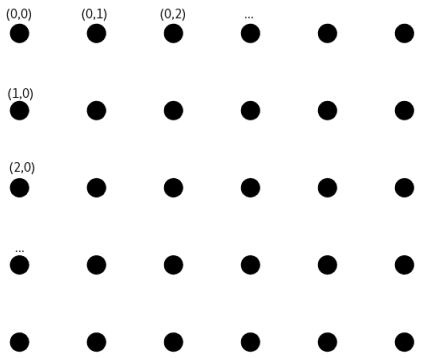

Figure 6: The rectangular lattice supporting a $(5, 6, d)$-SOM

In it's simpliest version, a SOM can be describe by a set of points, called cells, on a regular rectangular **h*w** lattice, see Figure 6. Each one of these points is associated with a corresponding **feature vector** of dimension **d**. Thus, the triplet (h, w, d) fully describes the structure of the SOM.

Training. Training a SOM consists in presenting it d-dimensional data and iteratively update the feature vectors on the map so that they get closer to the data. Information is spread into neighbourhoods on the map so that close cells have close feature vectors. In order to ensure convergence, the effect of each change and its influence on neighbours dimishes with time.

Concretely, at each time t, a vector D_t is randomly selected from the data and each cell with feature vector W_t is updated according to the following rule:

$$W_{t+1} = W_t + L(t) * N(\delta, t) * (D_t - W_t)$$

With:

1. $L(t)$ the **learning rate**, of the form $L(t) = L_0 * e^{\frac{-t}{\lambda}}$. With L_0 the initial learning rate and λ the time scaling factor. They are hyper-parameters of the training.

2. $N(\delta, t)$ the **neighbouring penalty** where δ is the euclidian distance between the coordinates of the cell we are currently updating and the coordinates of the Best Matching Unit (**BMU**). The BMU is the cell on the SOM having the closest W_t to D_t before update. $N(\delta, t)$ is of the Gaussian form:

$$N(\delta, t) = e^{-\frac{\delta^2}{2\sigma(t)^2}}$$

The spatial scaling factor $\sigma(t)$ has a really close expression than $L(t)$: $\sigma(t) = \sigma_0 * e^{\frac{-t}{\lambda}}$ with the same λ. We call σ_0 the initial neighbourhood, it's a hyper-parameter aswell. As time passes by, this neighbourhood shrinkes toward the BMU and the changes on the SOM become more and more local.

Starting and stopping. An important choice before training a SOM is how to initialize it and when to stop the training. There are various ways of initializing a SOM such as: uniformly at random, with a gaussian distribution, with a random sample of the data vectors etc. In the following experiments we always have uniformly initialized the SOM. We chose to stop the learning process when the changes only significally affect the BMU, that is when $\sigma(t) < 1$.

Choosing the hyper-parameters. We have 3 hyper-parameters: L_0, λ, σ_0. Like in many Machine Learning algorithms, choosing the hyper-parameters is a crucial point in order to ensure a good training. Here's the common way to choose them:

1. L_0: having $L_0 = 1$ directly replaces W_t by D_t at $t = 0$. Experimentally, choosing $0.5 < L_0 < 2$ is often a good fit.

2. λ: given our stopping criterion λ drives the total time of learning. It is also crucial in the evolving dynamic of the SOM. Typically choosing $50 < \lambda < 1000$ was a good fit.

3. σ_0: it is very common to set the initial neighbourhood to half the size of the SOM. For instance we would choose $\sigma_0 = 20$ for a $(40, 40, d)$-SOM.

2.2 Visualizing our Features Sets

The code that generates the following visualization is available in this notebook. On purpose, the online version is ran on a different dress for diversity.

Figure 7: A dress from our catalogue with a cat on it

The SOM will enable us to see what the DCNN "sees" on our garment features sets. Here, we do not use the SOM as an analytical tool but only for this visualization purpose. It substracts a bit of the mystery surrounding the DCNNs features. We'll take as an example the features set extracted

from the dress in Figure 13. This dress has the characteristic of having a very recognizable cat on it. We extract all the sliding-window features as explained in Section 1.3, they constitute our features set. We train on these 2048-dimensional features a $(8, 8, 2048)$-SOM with $L_0 = 1$, $\lambda = 200$, $\sigma_0 = 4$. The training takes 416 iterations for 1.37s, note that our implementation is not optimized for GPU.

Here is what we obtain:

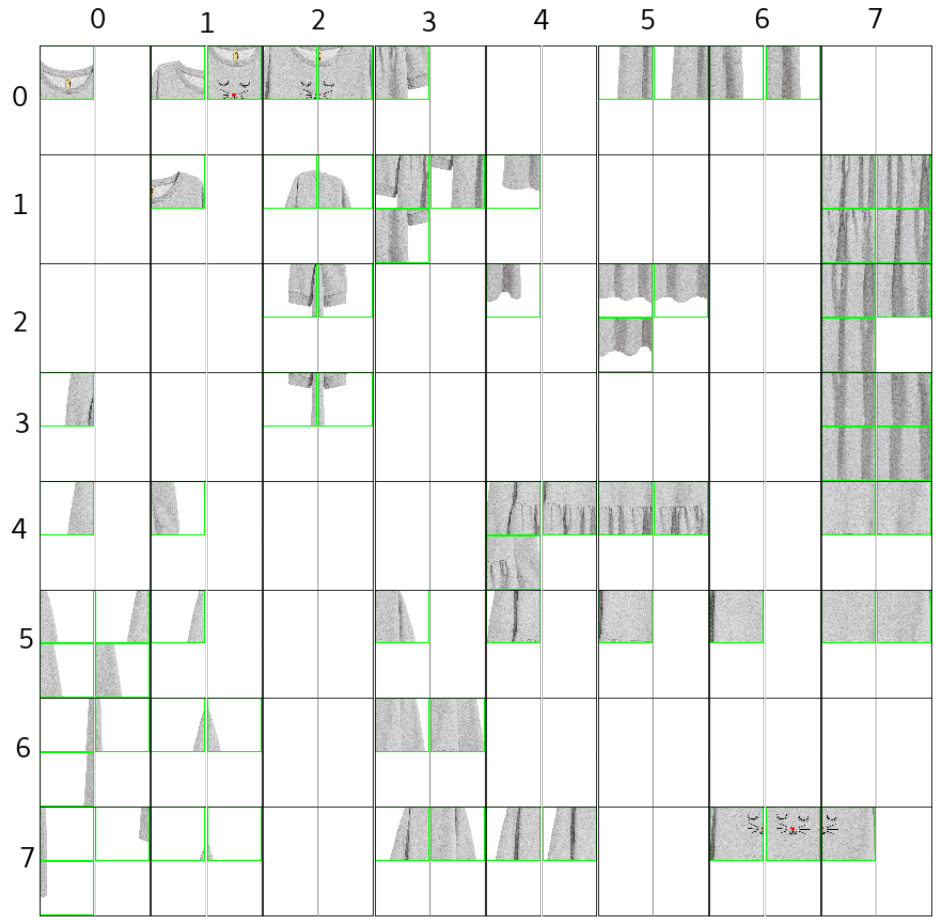

Figure 8: Visualization of the SOM trained on the cat dress

This visualization is done by simply aggregating the garment's feature vectors on their BMU and then plot the corresponding original sub-image. Please do not mind the grey vertical lines we couldn't get rid of.

Thanks to the SOM, we can see that the features divide the garment in categories that we, humans, understand:

1. Top left: we see all what's collar-related.
2. Around $(2, 2)$, all what's related to the ankles (the features seem symmetry invariant!) and the sleeves.
3. More in the center, around $(4, 4)$ the characteristic folds of the dress.
4. Bottom right: the cat! Interestingly enough, even parts of the cat are aggregated there whereas on $(0, 1)$ and $(0, 2)$ the collar is dominant on the cat for the DCNN.
5. Bottom left: all the "junk" features that were mainly blank.

6. Empty cells: We also notice there are few empty cells such as $(1, 0)$ or $(0, 7)$. Their feature vector is not the BMU of any of our data.

2.3 Emergent SOM and U-Matrix

A different type of behavior appears when we consider much larger SOMs. When almost all the data has its own BMU the setting is called Emergent SOM. As suggested by [23], considering the U-Matrix leads to a new type of visualization. The U-Matrix is a matrix of the same (h, w) dimension as the SOM where each cell holds the sum of the distances between the corresponding SOM feature vector and it's 4 adjacent neighbours. It gives a notion of montains separating similarity plains.

We train a $(50, 50, 2048)$-SOM on our data with $L_0 = 1$, $\lambda = 200$, $\sigma_0 = 24$. Training runs for 775 iteration and takes 1m23s. We get the following contour diagram:

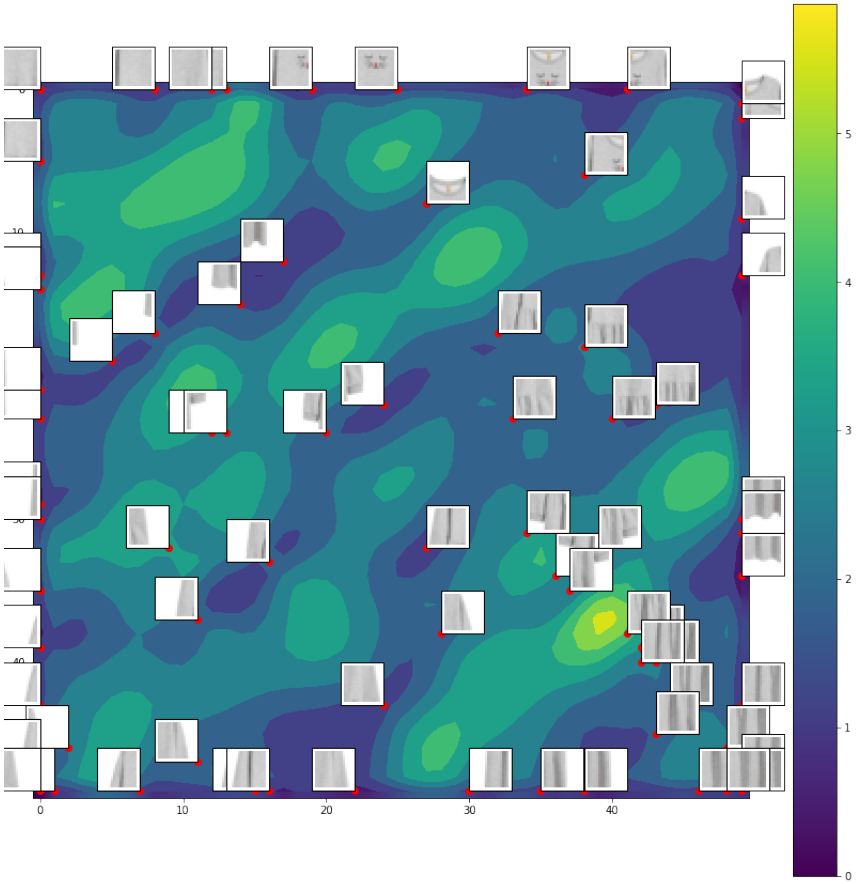

Figure 9: Visualization of the ESOM trained on the kitty dress

We still observe a nice repartition of our features but conclusions are a bit different. Especially for the distinction between the cat on the collar and the cat by its own: these features appear to be a lot closer, without an evident mountain to separate them. We observe a large plain containing the dress's drops. As we'll mention in Section 3.3, this kind of representation could lead to analytical tools.

You'll find in Appendix B the corresponding visualization for the associated query cat dress image. We see that the topologies of these SOM and ESOM are quite different than from the catalogue dress. Despite of their visualization power, SOM and ESOM doesn't give a straightforward way to solve our problem. During these internship, I've written a blog post containing more details and experiments with SOMs, it can be found here.

3 Apparel Retrieval

Now that we have a collection of features for each item in the catalogue, we must retrieve one particular dress given its associated query and this query's feature vectors. We are going to experiment several retrieval methods and compare them. Our measurement of success is *top-10 accuracy*: we want each garment to be among the 10 first proposals. Nevertheless, we also display results at top-1,3,5 accuracy. We also compute the average time per query for each method.

All of the following experiments were conducted in python with frameworks `tensorflow` for DCNNs, `opencv` for image manipulation, `sklearn` for classifiers, `numpy` and `scipy` for calculus. The following hardware setting was used: `cpus: 2 x Intel Xeon 1.7 GHz, ram: 32GB, gpus: 2 x GTX 1080 Ti`.

3.1 Methods Description and Results

Reference Methods. As a way to compare the results we are going to obtain, two reference methods also have been implemented. They do not use our sub-sampled DCNN features sets.

1. Traditional Bag-Of-Words (**BOW**) method [20], with k-means clustered dictionaries based on ORB features [25]. This kind of method that uses mathematical features, in the spirit of SIFT [24] or SURF [1], were the most used before the DCNN era. We don't use the query ROI information when implementing this method. The notebook of this experiment is to be found here.

2. DCNN features extracted from the whole images (**TakeAll**) – no sub-sampling – on both queries and items, the retrieval is performed by nearest cosine similarity. The code[4] is to be found here.

Methods on our features sets. All the following methods use the features sets. They are implemented on this notebook.

1. Nearest Neighbours with Majority Vote (**kNNMaj**): we treat the features of the catalogue images as one big set. For each query, for each feature vector, we find its nearest neighbour's class in the catalogue and then perform majority vote on all the query's feature vectors.

2. Memory Vector and Nearest Neighbour (**MemVec**): we use the ideas expressed in [13] in order to summarize each features set with a sum-memory vector. We then perform nearest neighbours retrieval. However the Penrose-Moore memory vectors also described in [13] where not convincing on our data.

3. Fully Connected with Majority Vote (**FCMaj**): we train a fully connected one layer (**ReLu**) softmax neural network. It associates each feature vectors to a probability distribution over catalogue images. Consequently, it has 2048 inputs and as many outputs as the catalogue has items, 196. We use a 1024-sized hidden layer. We then perform retrieval by doing a majority vote among the most probable prediction of the neural network on each of the query's feature vector.

4. Fully Connected with Average probability (**FCAvg**) : with the same neural network, we average the probability distributions of all the feature vectors on a features set and perform top-k retrieval in this averaged distribution.

5. Random Forest with Majority Vote (**RFMaj**) : same as FCMaj but, instead of a neural network, a Random Forest [3] as a classifier. Random Forests were proven efficients for our kind of problem in [2].

[4] This code is extracted from previous work, it hasn't the same global architecture as the rest.

6. Random Forest with Average probability (**RFAvg**) : same as FCAvg but, instead of a neural network, a Random Forest [3] as a classifier.

	Accuracies				
Methods	top-1	top-3	top-5	top-10	time per request (s)
BOW	7%	14%	20%	25%	0.01
TakeAll	-	-	-	30%	-
kNNMaj	53%	72%	77%	84%	0.7
MemVec	51%	64%	71%	80%	0.1
FCMaj	60%	74%	80%	86%	**0.002**
FCAvg	**61%**	**75%**	**82%**	**89%**	**0.002**
RFMaj	40%	51%	57%	69%	0.1
RFAvg	49%	67%	76%	85%	0.1

Figure 10: Results of described retrieval methods on our dresses dataset

Results. The results are shown in Figure 10. With BOW, the gap in efficiency between using old-school features and neural features appears clearly. With TakeAll, we conclude that taking features on entire images is not satisfying – as prevsited in Section 1.3. With both the Fully Connected and the Random Forest approaches, we see that the Avg method is preferable than majority vote. This averaging method can be seen as an ensemble technique which is more resistant to noise and false positive matches. Among all the retrieval techniques that we tested, FCAvg performs the best – we selected the best neural model we had trained. Fully Connected techniques also do very well on our timing measurement. However, timing is not a really fair comparison as only FC methods ran on GPU.

The FCAvg method requires a non-negligeable, but still small, time budget at train time – around 5min. It appears to be fast and efficient at request time. Consequently it would be a good fit for an industrial application.

3.2 Error Analysis

Figure 11: Summary of FCAvg errors

We are going to see whether we can explain the 11% error rate of the FCAvg method. Figure 11 summarize these errors as follow: it plots each ill-classified query with its associated garment in a box. They are shown in increasing order of predicted rank, which is written in red. Finally, in each box, the top left figure is a label enumerating these errors while the bottom left figure is the id of the garment in our dataset. In the rest, we will only refer to the rank – red – and label – top left – of an error. The mean rank of ill-classified queries is pretty high: 35. These ranks go from 11 to 143, let's recall that a success is being ranked in the 10 first proposals. Firstly, we can remark that a lot of ill-classified garments are black, it can be explained by the fact that our dataset contains a lot of black dresses thus to distinguish them is harder. Secondly we can spot four main reasons for errors:

1. Physiology of the model: the model deforms the garment when wearing it. Also, the model can obstruct or modificate crucial details. As in errors labelled 9, 10, 11, 13, 14, 16, 17, 18, 21.

2. Pose orientation of the model: similarly, our method seems to be quite pertubated when the model has not a frontal pose. As in errors labelled 5, 6, 15, 19, 20.

3. The ROI: sometimes the element we want to capture are not or just partially in the ROI. It is the less dramatic kind of error because the ROI should be specified by the user. It is the case

for errors labelled: 1, 2 and 3. Generally, by adapting the ROI we achieve retrieval for errors 1, 2, 3, 8, 9 leading to a retrieval of 91% for FCAvg.

4. Fine-grained details: sometimes the characteristic elements of the garment can be hard to catch with our sub-sampling method. As in errors labelled 4, 12.

However this analysis is, for most of it, intuitive. Therefore it has a limited explanatory power and doesn't tell why, for instance, error 1 is pretty well ranked whereas most of its ROI is centered on an off-topic giant cat. At the same time, other errors which seem pretty fine, like 16, are really badly ranked.

3.3 Going Further

In this section we inspect four of the main directions this work could take in the future.

Figure 12: Style transfer example, from [32]

Texture Networks. Texture Networks [31], [32], are a very successful method used in a completely different DCNN field: Style Transfer. Style Transfer consist in mixing the style of an image with the content of an another, Figure 12 issued from [32], illustrates this process. What could be interesting for our work is the concept of Texture Loss, introduced in [8], that Texture Networks use to get those results. This Texture Loss function says how two images differ in texture. This pseudo metric is based on statistical information coming from DCNN's convolution features' maps. In term of apparels we can intuitively think that such a texture metric could be interesting for retrieval – even if some research tempers the importance of texture in apparel recognition [36]. We tried to integrate the texture loss in our process without success. We still think it could be of use.

The Self-Organized-Map. In Section 2, we introduced the SOM as a tool for visualization. However, the Emergent SOM shown in Section 2.3 can be used for analytical purpose. Indeed various clustering methods [23], [33], have been developed in order to decompose the ESOM in clusters in an unsupervised way. We could use it in order to automatically segmentize our garments according to the sub-sampled features and perform retrieval by comparing segments. For instance, comparing collars with collars, sleeves with sleeves etc...

More Queries. The need we had for a simple ROI on the query side left us with a few queries to test our system. A lot of queries can be mined on the internet, especially with social networks. However we would not have the associated ROI. As suggested in Section 1.1, the use of modern method such as Mask R-CNN would enable us to automatize the search of the ROI and perform high-scale testing. As this method is really new, it is a challenge to fully understand it and to reproduce the result of the introductory article [9] which relies on a quite thick litterature. An other point is that, as our system has been trained on high quality data, it's not obvious that it would behave well with standard quality images. Nevertheless, we tested the system with photos of friends taken from a mediocre-quality cellphone and had good proposal results. If high scale testing showed that quality was an issue, we could use standard image distortion techniques in order to perturbate our samples and hopefully make the system more robust.

General Catalogue Matching. If we abstract a bit from our apparel application field we can derive a generalization of the problem we are trying to solve. We call it **Catalogue Matching**. Catalogue Matching consists in having:

1. A specialized, thus pretty small, *catalogue* of n same-class items. By n small we mean for instance $n < 10000$.

2. For each item, less than k standard visual descriptions of it with k really small. For instance $k < 10$ – here $k = 1$.

3. Unknown queries of these objects in natural settings.

The object of Catalogue Matching being to correctly associate queries to catalogue items. As mentioned in the Introduction, being a "small data" problem, it is very challenging to solve it with current tools in Machine Learning. On top of good features extractors, unsupervised methods, such as SOMs, are very likely to be crucial tools in order to solve Catalogue Matching. Quite obviously, this problem has a lot of industrial instances.

Conclusion

In this work, we constructed a custom dataset in order to solve Catalogue Matching in the case of apparel intraclass retrieval. We saw that this problem was inherently challenging because of the little data it implies to work with and that it had been littlely addressed in the associated litterature. We have seen how DCNNs, used as feature extractors, could be applied on such small datasets in a transfer learning way. Thanks to the SOM, we have explored an unsupervised tool in order to visualize our DCNNs features in a way we havn't found in existing work. Finally we tested and compared several retrieval methods, among those, the Fully Connected layer with Average probability appeared to be superior according to accuracy and timing criterions. In a nutshell, we built an apparel retrieval pipeline which could be tested, as it, in an industrial application.

On a more personal side, we re-enforced our technical knowledge about DCNNs and had a first experience with "small-data" problems. We also were introduced to the Self-Organizing Map and its high-dimensional data visualization power. Our experiments with the SOM gave us the intuition that such unsupervised methods could help to understand the black boxes DCNNs currently are.

References

[1] Herbert Bay, Tinne Tuytelaars, and Luc J. Van Gool. SURF: speeded up robust features. In *Computer Vision - ECCV 2006, 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006, Proceedings, Part I*, pages 404–417, 2006.

[2] Lukas Bossard, Matthias Dantone, Christian Leistner, Christian Wengert, Till Quack, and Luc J. Van Gool. Apparel classification with style. In *Computer Vision - ACCV 2012, 11th Asian Conference on Computer Vision, Daejeon, Korea, November 5-9, 2012, Revised Selected Papers, Part IV*, pages 321–335, 2012.

[3] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[5] Adit Deshpande. A beginner's guide to understanding convolutional neural networks, 2016. Available at https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/.

[6] Kaski et al. Bibliography of self-organizing map (som) papers: 1981-1997, 1997. Available at http://cis.legacy.ics.tkk.fi/research/som-bibl/vol1_4.pdf.

[7] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT 1998, Madison, Wisconsin, USA, July 24-26, 1998.*, pages 209–217, 1998.

[8] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 262–270, 2015.

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[11] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[12] Junshi Huang, Rogério Schmidt Feris, Qiang Chen, and Shuicheng Yan. Cross-domain image retrieval with a dual attribute-aware ranking network. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1062–1070, 2015.

[13] Ahmet Iscen, Teddy Furon, Vincent Gripon, Michael G. Rabbat, and Hervé Jégou. Memory vectors for similarity search in high-dimensional spaces. *CoRR*, abs/1412.3328, 2014.

[14] Andrej Karpathy. Transfer learning, 2017. Available at http://cs231n.github.io/transfer-learning/.

[15] M. Hadi Kiapour, Xufeng Han, Svetlana Lazebnik, Alexander C. Berg, and Tamara L. Berg. Where to buy it: Matching street clothing photos in online shops. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 3343–3351, 2015.

[16] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, January 1982.

[17] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78:1464–1480, 1990.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[20] Jialu Liu. Image retrieval based on bag-of-words model. *CoRR*, abs/1304.5168, 2013.

[21] Si Liu, Zheng Song, Guangcan Liu, Changsheng Xu, Hanqing Lu, and Shuicheng Yan. Street-to-shop: Cross-scenario clothing retrieval via parts alignment and auxiliary set. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 3330–3337, 2012.

[22] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 1096–1104, 2016.

[23] Jörn Lötsch and Alfred Ultsch. Exploiting the structures of the u-matrix. In *Advances in Self-Organizing Maps and Learning Vector Quantization - Proceedings of the 10th International Workshop, WSOM 2014, Mittweida, Germany, July, 2-4, 2014*, pages 249–257, 2014.

[24] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[25] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski. ORB: an efficient alternative to SIFT or SURF. In *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 2564–2571, 2011.

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

[27] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9, 2015.

[28] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826, 2016.

[29] Tensorflow. Inceptionv3 pre-trained model. Available at https://www.tensorflow.org/tutorials/image_recognition.

[30] J. T. Turner, Kalyan Moy Gupta, Brendan Morris, and David W. Aha. Keypoint density-based region proposal for fine-grained object detection and classification using regions with convolutional neural network features. *CoRR*, abs/1603.00502, 2016.

[31] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1349–1357, 2016.

[32] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. *CoRR*, abs/1701.02096, 2017.

[33] Juha Vesanto and Esa Alhoniemi. Clustering of the self-organizing map. *IEEE Trans. Neural Netw. Learning Syst.*, 11(3):586–600, 2000.

[34] Ji Wan, Dayong Wang, Steven Chu-Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, and Jintao Li. Deep learning for content-based image retrieval: A comprehensive study. In *Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014*, pages 157–166, 2014.

[35] Wikipedia. Content-based image retrieval, 2017. Available at https://en.wikipedia.org/wiki/Content-based_image_retrieval.

[36] Qin Zou, Zheng Zhang, Qian Wang, Qingquan Li, Long Chen, and Song Wang. Who leads the clothing fashion: Style, color, or texture? A computational study. *CoRR*, abs/1608.07444, 2016.

Appendix A Python SOM Implementation

```
import numpy as np
import itertools

class SOM(object):

def __init__(self,h,w,dim_feat):
    """
        Construction of a zero-filled SOM.
        h,w,dim_feat: constructs a (h,w,dim_feat) SOM.
    """
    self.shape = (h,w,dim_feat)
    self.som = np.zeros((h,w,dim_feat))

    # Training parameters
    self.L0 = 0.0
    self.lam = 0.0
    self.sigma0 = 0.0

    self.data = []

    self.hit_score = np.zeros((h,w))

def train(self,data,L0,lam,sigma0,initializer=np.random.rand,frames=None):
    """
        Training procedure for a SOM.
        data: a N*d matrix, N the number of examples,
              d the same as dim_feat=self.shape[2].
        L0,lam,sigma0: training parameters.
        initializer: a function taking h,w and dim_feat (*self.shape) as
                     parameters and returning an initial (h,w,dim_feat) tensor
        frames: saves intermediate frames if not None.
    """
    self.L0 = L0
    self.lam = lam
    self.sigma0 = sigma0

    self.som = initializer(*self.shape)

    self.data = data

    for t in itertools.count():
        if frames != None:
            frames.append(self.som.copy())

        if self.sigma(t) < 0.5:
            print("final t:", t)
            #print("quantization error:", self.quant_err())
            break

        i_data =  np.random.choice(range(len(data)))
```

```
            bmu = self.find_bmu(data[i_data])
            self.hit_score[bmu] += 1

            self.update_som(bmu,data[i_data],t)

    def quant_err(self):
        """
            Computes the quantization error of the SOM.
            It uses the data fed at last training.
        """
        bmu_dists = []
        for input_vector in self.data:
            bmu = self.find_bmu(input_vector)
            bmu_feat = self.som[bmu]
            bmu_dists.append(np.linalg.norm(input_vector-bmu_feat))
        return np.array(bmu_dists).mean()

    def find_bmu(self, input_vec):
        """
            Find the BMU of a given input vector.
            input_vec: a d=dim_feat=self.shape[2] input vector.
        """
        list_bmu = []
        for y in range(self.shape[0]):
            for x in range(self.shape[1]):
                dist = np.linalg.norm((input_vec-self.som[y,x]))
                list_bmu.append(((y,x),dist))
        list_bmu.sort(key=lambda x: x[1])
        return list_bmu[0][0]

    def update_som(self,bmu,input_vector,t):
        """
            Calls the update rule on each cell.
            bmu: (y,x) BMU's coordinates.
            input_vector: current data vector.
            t: current time.
        """
        for y in range(self.shape[0]):
            for x in range(self.shape[1]):
                dist_to_bmu = np.linalg.norm((np.array(bmu)-np.array((y,x))))
                self.update_cell((y,x),dist_to_bmu,input_vector,t)

    def update_cell(self,cell,dist_to_bmu,input_vector,t):
        """
            Computes the update rule on a cell.
            cell: (y,x) cell's coordinates.
            dist_to_bmu: L2 distance from cell to bmu.
            input_vector: current data vector.
            t: current time.
        """
        self.som[cell] += self.N(dist_to_bmu,t)*self.L(t)
```

```
                        *(input_vector-self.som[cell])

def update_bmu(self,bmu,input_vector,t):
    """
        Update rule for the BMU.
        bmu: (y,x) BMU's coordinates.
        input_vector: current data vector.
        t: current time.
    """
    self.som[bmu] += self.L(t)*(input_vector-self.som[bmu])

def L(self, t):
    """
        Learning rate formula.
        t: current time.
    """
    return self.L0*np.exp(-t/self.lam)

def N(self,dist_to_bmu,t):
    """
        Computes the neighbouring penalty.
        dist_to_bmu: L2 distance to bmu.
        t: current time.
    """
    curr_sigma = self.sigma(t)
    return np.exp(-(dist_to_bmu**2)/(2*curr_sigma**2))

def sigma(self, t):
    """
        Neighbouring radius formula.
        t: current time.
    """
    return self.sigma0*np.exp(-t/self.lam)
```

Appendix B Query SOM and ESOM visualization

Figure 13: The query associated to the cat dress

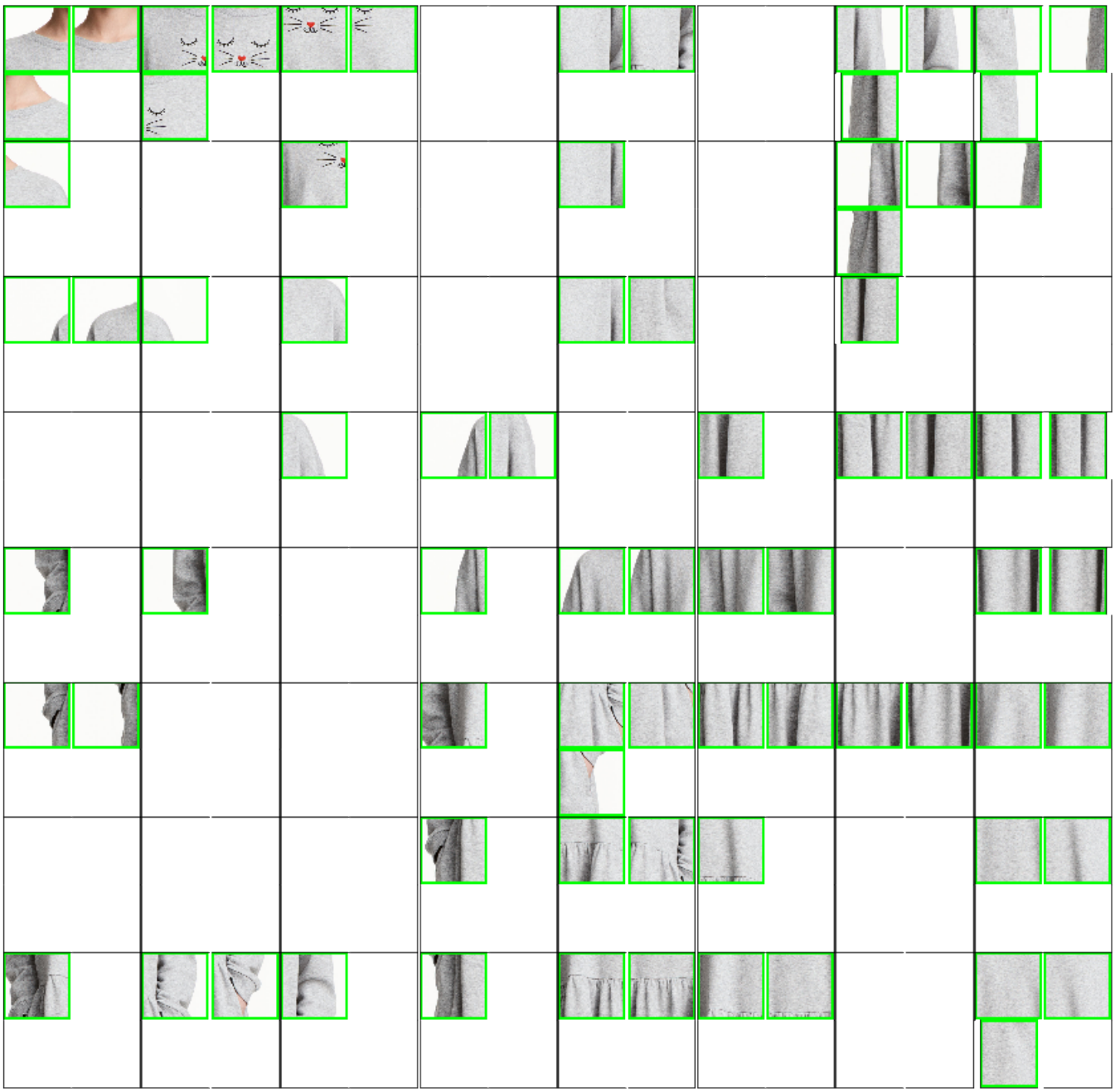

Figure 14: Visualization of the SOM trained on the cat dress query

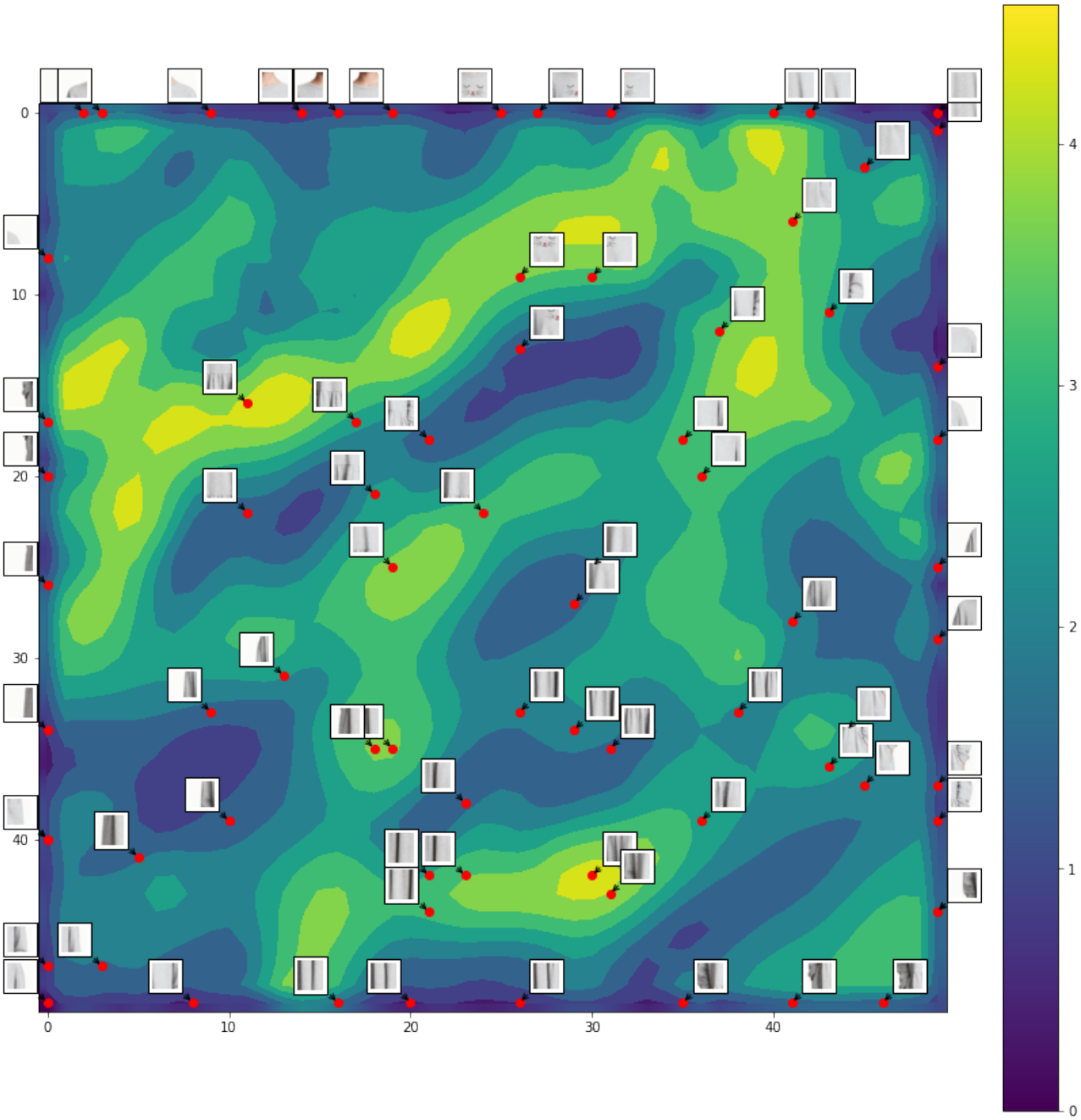

Figure 15: Visualization of the ESOM trained on the cat dress query

Appendix C Institutional Context

This internship took place at A7 Interactive, a Monaco-based IT company, under the supervision of its CEO, Gilles Sabas. The internship was co-advised by Marc Sebban, Professor in Computer Sciences and Deputy Director of the LabHC laboratory at the University Jean-Monnet, St-Etienne.