Graduation internship report

Web application for a generic instance search using spatial locality on a large database from a single example

Aurélien Greffard

July-September 2014

IMAC Engineering School
Université Paris-Est-Marne-la-Vallée, France

Visual Cognitive Systems laboratory
University of Ljubljana, Slovenia

Table of Contents

1

Introduction

As a student of the French IMAC Engineering school[1], I had the opportunity to go abroad during my last year of study. I chose the University of Ljubljana, Slovenia, in the faculty of Computer and Information Science[2], where I could attend excellent courses of Master level during two semesters.

During the year, I was offered the opportunity to pursue my graduation internship in the Visual Cognitive Systems laboratory[3] on a research project, following the courses I attended in the University, especially the courses Machine Perception and Advanced Topics in Computer Vision, taught by the assistant professor Matej Kristan. This also turned out to be a great opportunity in my educational background, in the continuity of the IMAC school.

This report relates my work during this internship, and details the technical implementations of my project.

Acknowledgments

I want to thank my tutor Matej Kristan[4] for his supervision and his help throughout the internship. Luka Čehovin[5] for his feedback and advice and Domen Tabernik[6] for his support during the deployment phase on the servers.

I also thank all other researchers and members of the ViCoS laboratory for their warm welcome and their constant goodwill to communicate in English, allowing me to learn a lot during these three months in the laboratory.

[1] http://www.ingenieur-imac.fr/
[2] http://www.fri.uni-lj.si/en/
[3] http://www.vicos.si/
[4] http://www.vicos.si/People/Matejk
[5] http://www.vicos.si/User:Lukacu
[6] http://www.vicos.si/User:Doment

2

Description of the internship

1 The laboratory

The Visual Cognitive Systems Laboratory [7] is a research laboratory attached to the faculty of Information and Computer Science of the University of Ljubljana, in Slovenia. It is involved in basic research in visually enabled cognitive systems, with emphasis on visual learning and recognition.

Research focuses on various theories about requirements, architectures, forms of representation, and varieties of mechanisms relevant to integration and control of vision systems.

The team, composed of around a dozen of members: professors, researchers and students, is a dynamic and productive team involved in several applications of cognitive systems including tracking, surveillance, robotics or cognitive assistants.

2 Project description

The objective this project was to implement a state-of-the-art recognition method (training and recognition phase of the algorithm), available in the end as a web-service from the laboratory's page. I was alone on this project, supervised by my tutor Matej Kristan and other researchers from the laboratory.

A web-application performing object detection and categorization already existed online: ViCoS Eye.[8]. My long-term goal was then to extend it with my algorithm, using an infrastructure already in place, consisting of a web interface and a distributed processing backend.

The expected behaviour of the web-service is shown on Fig. 1: the user uploads a picture, selects the bounding box of the target on this image and launchs the search. The application is now supposed to find similar regions on all images of a particular database, and to return the best samples.

Similar web-services already exist, such as Google images[9], TinEye[10], Macroglossa[11]. But none of these web-services include locality, only using global image representation for the search. Thus, the desired application is more specific by including locality at all steps of the method, and by being able to retrieve particular regions of an image as well as entire images.

[7] http://www.vicos.si/
[8] http://eye.vicos.si/
[9] http://images.google.com/
[10] https://www.tineye.com/
[11] http://www.macroglossa.com/

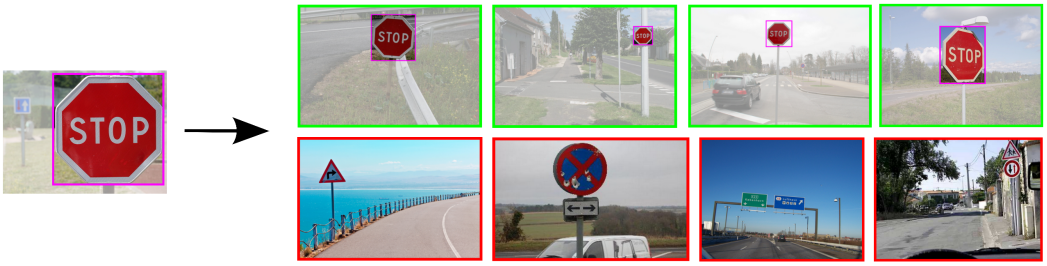

Fig. 1. Expected behaviour of the application on a particular database. The query instance in shown on the left, delineated by the bounding box. The results are shown on the right. The upper row is the positive results, with detected regions delineated by the bounding box. The lower row is an example of images without detected region.

Strategic issues

The objective of this service is to provide an online demonstration of state-of-the-art computer vision algorithms developed in the laboratory, as a proof of concept of current computer vision research and recent papers.

Secondarily, this service can be used to eventually find potential clients or investors. For this reason, the application has to result in a working demonstration available online from a web page and a mobile application.

No particular technology was imposed on me. The only constraint was to write at the end a wrapper implemented in Java, to be able to use Hadoop and MapReduce technologies to perform an efficient parallelization of the algorithm.

The time execution of the recognition algorithm should be adequate for an online application. We estimate that it is acceptable for a user to wait 2 to 5 seconds [1] [2] for this kind of application. This is consequently included in the objective of this project.

3

Project design and implementation

At the beginning of the internship, my tutor sent me two interesting papers to study, strongly related to this project: *Locality in Generic Instance Search from One Example* [3] and *Recognizing Locations with Google Glass: A Case Study* [4], recent comparison of practical implementations adapted to the Google Glass devices. The first paper was a good starting point, really similar to my objective. I also had the occasion to read several other interesting papers who helped me to implement the application and reuse existing code.

My work was divided into small steps, implementing small building blocks working together: explosion of the image into different regions, construction of global descriptors for all of these regions, compression, recognition... Each step requiring the previous ones to perform well.

All steps were implemented in Matlab, a useful and fast tool for mathematic research and image processing, using VLFeat[12] [5], an open source library written in C with interfaces in Matlab, implementing popular computer vision algorithms.

In this chapter, we will review the implementation details about each development step, evaluate the preliminary results obtained by the application and discuss the remaining work and possible future improvements.

3 Selective search

Our objective is to find similarities between our target and particular regions of images in our database. Thus, we need an efficient way to select and extract regions to compare, robust to scale. The naive way is to apply a sliding window at different scales on the whole image and to compare each resulting region. But this is an extremely costly method, not appropriate for practical application with time-demanding feature extraction and classification. Other methods were proposed to solve this problem, such as randomly sampling the regions, or by using visual words of a Bag-of-Words model combined to a generalized Hough-transform to predict the location of potential objects. [6]

We decided to use another recent method called Selective Search [7], performing really good. The idea is to use a bottom-up grouping procedure to generate good object locations. The starting regions are generated using the Felzenszwalb and Huttenlocher detection method [8], and all the regions are iteratively grouped together by calculating similarities between neighouring regions, until the whole image becomes a single region. This allows to capture all scales, is fast to compute and generates a small but relevant set of regions, basically between 100 and 1000 regions for each image. By using a diverse set of strategies such as different similarity measures and a variety of colour spaces with different invariance properties, this method is also able to detect potential objects of different nature, according to texture and colour at the same time.

The Matlab code for this Selective Search method is available on the web page of the authors Jasper Uijlings[13] and Koen van de Sande[14]. The Fig. 2 shows a selected subset of regions generated by the algorithm.

We execute this algorithm on all our images, and we store the coordinates of all the resulted regions.

[12] `http://www.vlfeat.org/`
[13] `http://disi.unitn.it/~uijlings/MyHomepage/index.php#page=projects1`
[14] `http://koen.me/research/`

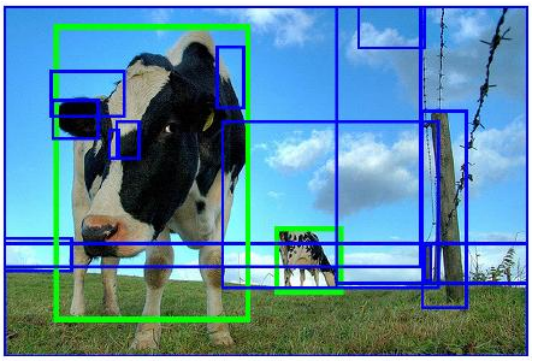

Fig. 2. Example of bounding boxes generated by the Selective Search [7] algorithm.

4 Fisher vectors

All my research algorithm is based on Fisher vector [9] [10] [11] as a global image descriptor for the regions. This image representation is one of the best-performing state-of-the-art global image descriptors and can be seen as a generalization of the Bag-of-Visual-Words model [12]. The Fisher vector is obtained by pooling local image features together, and describes how a set of descriptors deviates from an average distribution of descriptors modeled by a Gaussian Mixture Model.

Let $I = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_N}), \boldsymbol{x_i} \in \mathbb{R}^D$ be a set of N D-dimensional feature vectors (e.g. SIFT descriptors [13]) extracted from an image. Let $\Theta = (\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}, \pi_k : k = 1, \ldots, K)$ be the parameters of a Gaussian Mixture Model with K mixture components fitting the distribution of the descriptors, where $\boldsymbol{\mu_k}$, $\boldsymbol{\Sigma_k}$ and π_k are respectively the mean, covariance and prior probability for the mode k in the mixture model, assuming diagonal covariance matrices. The GMM associates each vector $\boldsymbol{x_i}$ to a mode k with a strength given by the posterior probability (soft alignment of $\boldsymbol{x_i}$ to the Gaussian k):

$$q_{ik} = \frac{\exp\left[-\frac{1}{2}(\boldsymbol{x_i} - \boldsymbol{\mu_k})^T \boldsymbol{\Sigma_k}^{-1}(\boldsymbol{x_i} - \boldsymbol{\mu_k})\right]}{\sum_{t=1}^{K} \exp\left[-\frac{1}{2}(\boldsymbol{x_i} - \boldsymbol{\mu_t})^T \boldsymbol{\Sigma_k}^{-1}(\boldsymbol{x_i} - \boldsymbol{\mu_t})\right]}.$$

In what follows, we denote by $\boldsymbol{\sigma_k^2}$ the variance vector, i.e. the diagonal of $\boldsymbol{\Sigma_k}$. For each mode k, consider the gradient with respect to the mean $\boldsymbol{\mu_k}$ and variance $\boldsymbol{\sigma_k}$:

$$\boldsymbol{\mathcal{G}}^{\boldsymbol{I}}_{\boldsymbol{\mu_k}} = \frac{1}{N\sqrt{\pi_k}} \sum_{i=1}^{N} q_{ik} \frac{\boldsymbol{x_i} - \boldsymbol{\mu_k}}{\boldsymbol{\sigma_k}},$$

$$\boldsymbol{\mathcal{G}}^{\boldsymbol{I}}_{\boldsymbol{\sigma_k}} = \frac{1}{N\sqrt{2\pi_k}} \sum_{i=1}^{N} q_{ik} \left[\left(\frac{\boldsymbol{x_i} - \boldsymbol{\mu_k}}{\boldsymbol{\sigma_k}} \right)^2 - 1 \right],$$

where the division between vectors is as a term-by-term operation. The Fisher vector of the image from where were extracted the descriptors I is the stacking of the vectors $\boldsymbol{\mathcal{G}}^{\boldsymbol{I}}_{\boldsymbol{\mu_k}}$ and then of the vectors $\boldsymbol{\mathcal{G}}^{\boldsymbol{I}}_{\boldsymbol{\sigma_k}}$ for each of the K modes in the Gaussian mixtures:

$$\boldsymbol{\Phi}(I) = \begin{bmatrix} \vdots \\ \boldsymbol{\mathcal{G}}^{\boldsymbol{I}}_{\boldsymbol{\mu_k}} \\ \vdots \\ \boldsymbol{\mathcal{G}}^{\boldsymbol{I}}_{\boldsymbol{\sigma_k}} \\ \vdots \end{bmatrix},$$

and is therefore $2KD$-dimensional.

In order to construct the Gaussian Mixture Model required for the Fisher vector computation, we need a large set of local descriptors from our database. Following the directions from [4], confirmed by our own tests, we chose to use PHOW descriptors [14], variation of dense SIFT descriptors taken at different scales. The idea is to randomly select a set of PHOW descriptors extracted from all the images of our database. Because these descriptors are high dimensional, we reduce them to 64 dimensions using the Principal Component Analysis (PCA) [15]. This step even has a positive impact on the results, the obtained decorrelated data being fitted more accurately by the GMM.

As the performance of the GMM construction algorithm is strongly dependant on the initial clusters, it is beneficial to perform a pre-clustering on the data, instead of starting from random initial clusters. For this reason, we performed a K-means clustering with all our descriptors and used the results as initial clusters for the GMM construction.

Finally everything is ready to compute the GMM using the Expectation Maximization (EM) algorithm [16] to optimize a Maximum Likelihood (ML) criterion. We obtain three matrices corresponding to the GMM parameters, and we store them in our database. With 256 clusters obtained from 64-dimensional vectors, we store a matrix of 256×64 dimensions for the means and covariances (the covariance matrix being diagonal, we just have to store the variances on the diagonal), and a 256-dimensional vector for the priors.

Once the GMM has been constructed, we can compute the Fisher vectors for all the regions stored in the database. In order to have a robust scale-invariant detection algorithm, all the regions are resized to 400×400 pixels. For each resized region, we can now extract a set of PHOW descriptors, reduced to 64 dimensions via PCA, and use our GMM parameters to construct the Fisher vector, global descriptor of the region.

5 Product quantization

The Fisher vector is a very good and reliable global visual descriptor for an image, but is really high-dimensional. With 256 clusters in the GMM and 64-dimensional local descriptors, the dimension of the Fisher vector will be $N = 2*256*64 = 32768$ dimensions. If we compute this descriptor for hundreds of regions on thousands of images, this will result in Terabytes of data, impossible to store. We obviously need a way to compress our data.

We used the Product Quantization method [17] [18] to compress the descriptors. Each Fisher vector is split into 1024 subvectors, and for each subvector we keep only the index of the closest cluster center in a given codebook. To construct this codebook, we need another training step, just after the GMM construction. One again, we use the descriptors randomly extracted from all the images during the GMM training phase. We select a random set of these descriptors to construct a random collection of Fisher vectors. As each of these Fisher vectors is constructed from local descriptors taken from different images from all the database, they are not linked to any region of image, which means that they are general and universal enough to be a good representation for general objects found in the database.

We split all these randomly constructed Fisher vectors into 1024 subvectors. We obtain a huge collection of 32-dimensional subvectors, from which we can perform a K-means clustering to construct the codebook. We used 1024 clusters for this codebook, which is then a 1024×32 matrix containing the cluster centers obtained from all our subvectors. The Fig. 3 summarizes the main steps of construction of this codebook.

Since we have the codebook, we can now perform the Product Quantization [18] method to compress each Fisher vector extracted from the regions in the data set. For a given Fisher vector, we split it into 1024 subvectors and compare each of these subvectors with each cluster center in the codebook, using cosine similarity as a comparison method. We keep only the index of the closest cluster. At the end, the compressed Fisher vector is a 1024-dimensional vector, instead of a 32768-dimensional vector for an uncompressed vector. We managed to reduce the dimension of the data by a factor of 32. Furthermore, since we can now store the descriptors as vectors of 16bits integers instead of 64bits doubles, we even managed to compressed the data memory by a factor of 128 (plus an extra small amount of memory to

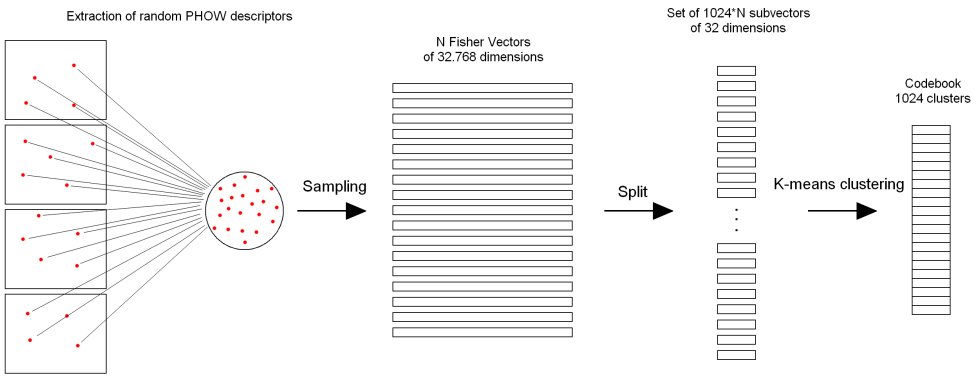

Fig. 3. Construction of the codebook from random local descriptors.

store the codebook).

This method has two advantages. First, it considerably reduces the size of descriptors in memory. But it also allows to speed up the recognition algorithm. Indeed, if we have a new Fisher vector to compare with all our database, we can pre-compute the distance between all its subvectors and all the cluster centers in the codebook, and store these precomputed values in a 1024×1024-dimensional look-up table (Fig. 5). This process is described in the next section.

The Fig. 4 resumes the main steps of construction of the compressed Fisher descriptors from all the regions.

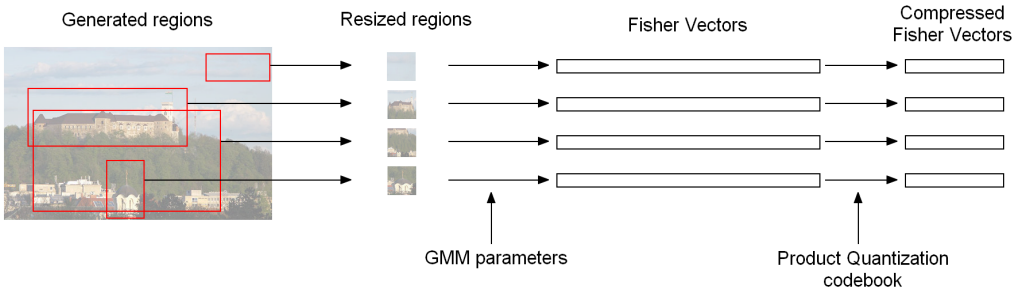

Fig. 4. Construction of the Compressed Fisher vectors.

6 Query

In this section we describe the search algorithm. A query is represented by an image uploaded by the user with a region of interest delineated by a bounding box. This region is resized to 400×400 pixels and its n-dimensional Fisher vector $\boldsymbol{u}$ is constructed using PHOW descriptors reduced to 64 dimensions using PCA. If we want to compare $\boldsymbol{u}$ with the descriptors of all the regions in our database, we will need to decompress all of them to calculate their similarity value (cosine similarity is the de facto similarity measure for Fisher vector [12]). This will require a lot of operations and a lot of time. We can significantly improve this step by using Product Quantization and pre-computing the distance between $\boldsymbol{u}$ and all the cluster centers in the codebook.

Let $\boldsymbol{v}$ be the n-dimensional decompressed Fisher vector describing a particular region in the database, obtained by pooling 1024 cluster centers from the codebook. By definition, the cosine similarity between $\boldsymbol{u}$ and $\boldsymbol{v}$ is represented using the dot product and magnitude:

$$\rho(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\| \, \|\boldsymbol{v}\|}$$

which is equivalent to:

$$\rho(\boldsymbol{u}, \boldsymbol{v}) = \frac{\sum_{i=1}^{n} u_i \times v_i}{\sqrt{\sum_{i=1}^{n} (u_i)^2} \times \sqrt{\sum_{i=1}^{n} (v_i)^2}}.$$

By splitting the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ into N subvectors (assuming that n is a multiple of N), this measure can be rewritten as:

$$\rho(\boldsymbol{u}, \boldsymbol{v}) = \frac{1}{\sqrt{\sum_{i=1}^{n} (u_i)^2}} \times \frac{1}{\sum_{j=0}^{N-1} \left(\sum_{i=1}^{\frac{n}{N}} (v_{Nj+i})^2 \right)} \times \sum_{j=0}^{N-1} \left(\sum_{i=1}^{\frac{n}{N}} u_{Nj+i} \times v_{Nj+i} \right).$$

We can notice that the first element of this product corresponds to the magnitude of the query Fisher vector $\boldsymbol{u}$, independant from the region Fisher vector $\boldsymbol{v}$. This value can be stored at the very beginning of the query and does not have to be recalculated for each region.

The second element of the product corresponds to the sum of the sum of each subvectors of the region Fisher vector $\boldsymbol{v}$ (actually corresponding to particular cluster centers of the codebook). Since they are independant to $\boldsymbol{u}$, all these values can even be precomputed during the training step and stored in the database just after the codebook construction.

The last element of the product corresponds to the sum of the dot product between the subvectors of $\boldsymbol{u}$ and the subvectors of $\boldsymbol{v}$ (particular cluster centers of the codebook). All these values can be precomputed just after the construction of $\boldsymbol{u}$. From all the cluster centers of the codebook and all the subvectors of $\boldsymbol{u}$, we can build a 1024×1024 matrix containing all possible combinations of dot products. As the compressed Fisher vectors contain only indices matching the closest cluster in the codebook, we can now use this matrix as a simple look-up table and we do not need to decompress the descriptors anymore. This method avoids a lot of redondant and time-consuming calculation. The Fig. 5 describes how to compute this look-up table.

Finally, with all these precomputed values:

m = magnitude of $\boldsymbol{u}$
$\boldsymbol{A}$ = look-up table containing precomputed sum of squared elements for each cluster center
$\boldsymbol{B}$ = look-up table containing precomputed dot products between cluster centers and subvectors from $\boldsymbol{v}$

and by writing $\boldsymbol{v'}$ as the 1024-dimensions compressed Fisher vector obtained by compressing $\boldsymbol{v}$, we can calculate the cosine similarity between $\boldsymbol{u}$ and $\boldsymbol{v}$ as follows:

$$\rho(\boldsymbol{u}, \boldsymbol{v}) = \frac{1}{m} \times \frac{1}{\sum_{i=1}^{N} \boldsymbol{A}(v'_i)} \times \sum_{i=1}^{N} \boldsymbol{B}(v'_i, i).$$

The cosine similarity is measuring the orientation between two vectors. The obtained value is bounded between -1 and 1, with a similarity equal to 1 for vectors with same orientation. Thus, we can apply a threshold to filter our results. Only the regions with a cosine similarity above this threshold will be considered as positive examples and returned by the application.

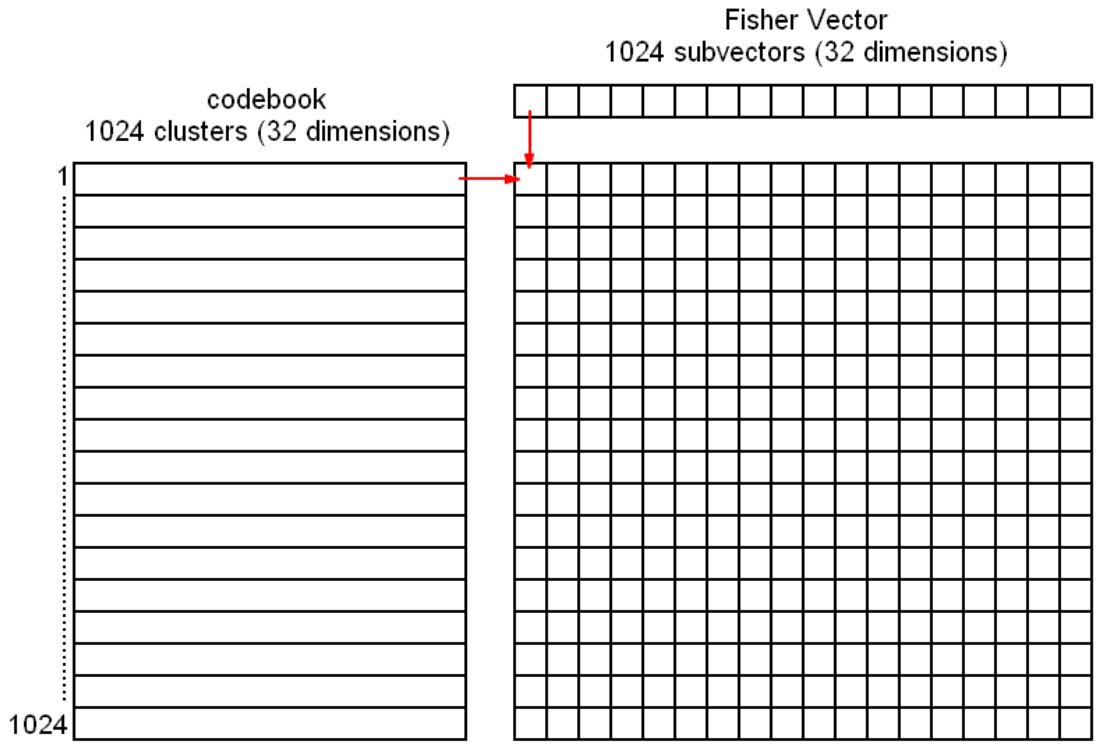

Fig. 5. Construction of the look-up table. Each cell of the matrix contains a dot product between a cluster center from the codebook and a subvector from the target's Fisher vector.

7 Preliminary results

In this section we discuss the results obtained by the algorithm. We test our application on a small database of 32 images containing the Ljubljana castle, the Ljubljana Franciscan church and the Pisa Tower, as shown in Fig. 6. In total, the Selective Search generated 22580 regions from these 32 images. We chose a value of 0.2 for the cosine similarity threshold.

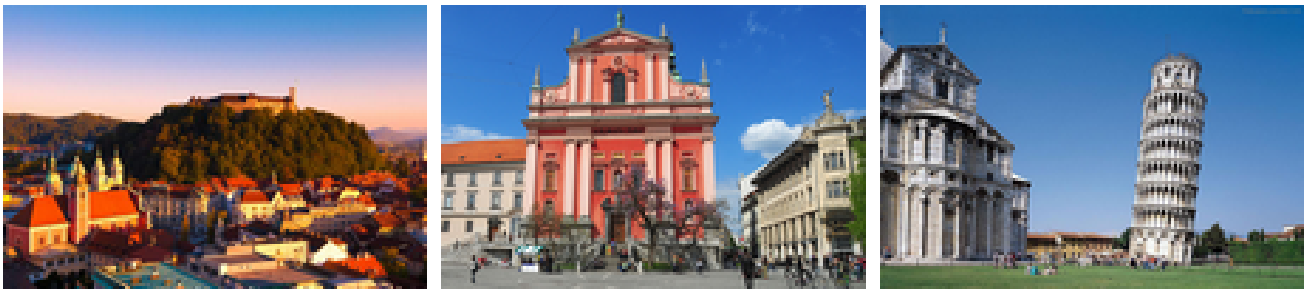

Fig. 6. Examples of images in the database. On the left: Ljubljana castle. In the middle: Ljubljana Franciscan church. On the right: Pisa Tower.

The Fig. 7 shows our test image, another image of Ljubljana castle hill, with the region focused on the castle. The program is supposed to retrieve all the regions containing the same castle from all our images in database.

Fig. 7. Training image and bounding box

The Fig. 8 shows the top 40 regions from the 75 regions retrieved by the algorithm.

These results seem visually really good and really encouraging, even though they contain sometimes a few imprecision. The algorithm managed to find regions containing the castle on most of the images in the database, and almost none of the pictures from Pisa Tower or Franciscan Church appears on the 75 returned samples.

We can notice that we obtain several regions from the same image. This can be easily improved by a non-maxima suppression, comparing the overlap between all detected regions, and keeping only the region with the best overlap.

Evaulation of the results

For reasons of objectivity, we need during the development phase a way to evaluate our results, not only visually. Thus, we wrote a function returning the sensitivity (also known as recall), specificity and precision of the algorithm. These values are calculated as follows:

$$sensitivity = \frac{\#TP}{\#TP + \#FN},$$

$$specificity = \frac{\#TN}{\#TN + \#FP},$$

$$precision = \frac{\#TP}{\#TP + \#FP},$$

with the notation:

$\#TP$ = number of True Positives,
$\#FP$ = number of False Positives,
$\#TN$ = number of True Negatives,
$\#FN$ = number of False Negatives.

The problem is now how to determine if a region represents a positive or a negative example. Sometimes it can be a really difficult decision even for a human being. We decided to manually annotate each image in the database with a class and ground truth (bounding box containing the object supposed to be retrieved). Then, in a similar way as in [19], for each generated region, we can compare the overlap

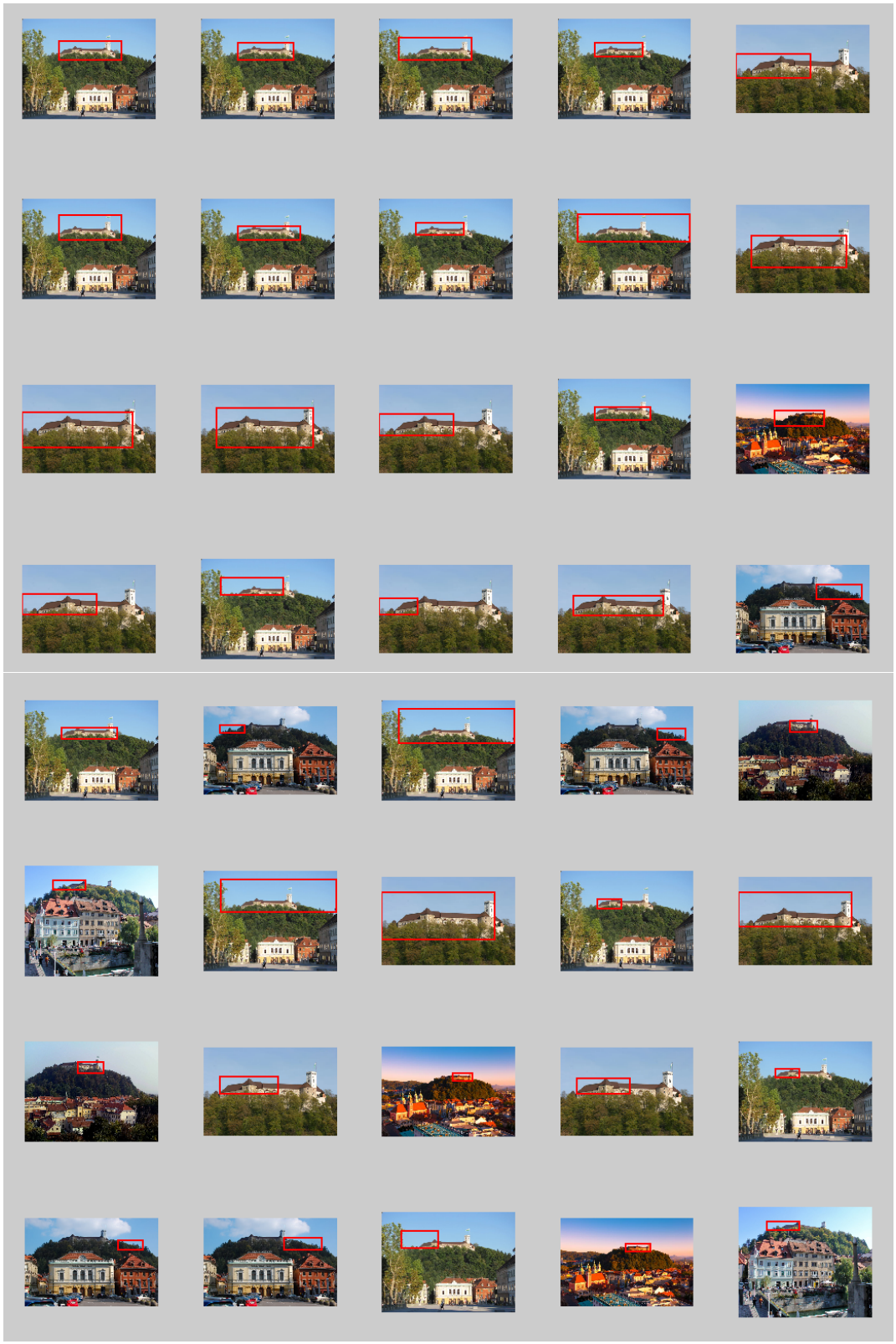

Fig. 8. Top 40 regions retrieved by the algorithm.

between the region and the ground truth. The overlap is calculated as the quotient between the intersection and the union of the two regions A_r (retrieved region) and A_g (ground truth):

$$overlap = \frac{A_r \cap A_g}{A_r \cup A_g}$$

See also the Fig. 9 for a visualization of this method. This overlap value is used to decide if the example is a positive or a negative sample, considered positive if the overlap if higher than a given threshold. We chose a value of 0.3 for this threshold.

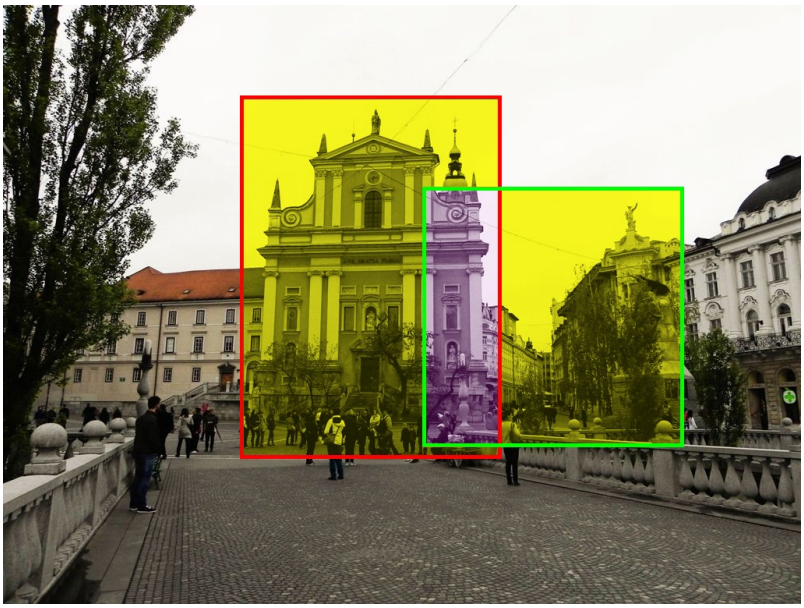

Fig. 9. Classification of an example as positive or negative. In red: the groundtruth manually annotated. In green: a detected region. In purple: the intersection between the regions. In yellow+purple: the union between the regions. This example is obviously detected as a Negative example.

Let us now evaluate our results with this method:

$$\begin{aligned} sensitivity &= 0.2235 \\ specificity &= 0.9971 \\ precision &= 0.3689 \end{aligned}$$

The sensitivity means that we managed to retrieve 22.3% of the regions considered as positive. The specificity means that 99.7% of the negative samples were indeed not selected. The precision means that 36.9% of the returned examples were actually positive examples.

The specificity value is a really good result, we managed to eliminate most of the uninteresting regions. However, we still have a significant proportion of false positives among the returned regions (given by the precision) and false negatives, not detected similar regions (given the sensitivity).

This means that we can still find ways to improve the recognition algorithm. The next section lists some possible approaches worth to try to eventually perform better results.

Nevertheless, we have to take care about not jumping to conclusions. Some other reasons can explain these low performance results. First of all, the automatic way to detect positive examples is not 100% accurate. This method is supposed to find all the positive samples, but is also subject to generate false

positive examples, which can significantly bias the results. Another point to consider is that our program only performs detection of local similarities, which should not be confused with classification. A human being will sometimes consider two images really similar because containing the same object, even if these images do not have close similarities in a mathematical perspective. This is the difference between human perception, complex combination between recognition and classification, and machine perception.

8 Remaining work

8.1 Possible improvements

Even if we already obtain really interesting results, we can imagine a lot of possible improvements.

Since we often get several similar regions from the same image, actually corresponding to the same region but slightly shifted or scaled, we can apply some post-processing methods such as non-maxima suppression to remove the duplicates and keep the most accurate region. We can perhaps also take this extra information into account to increase the accuracy of the returned coordinates.

Another idea is to try to optimize the target bounding-box selected by the user by performing the Selective Search on the input image as well and keeping only the result with best overlap with the user's bounding-box. This has the advantage to compare regions obtained by the same algorithm and can perhaps improve the results, without certitude.

Some images are more challenging than the other ones, with for example huge variations of intensity. Some preprocessing methods can be performed to solve or at least improve this probem, such as histogram equalization on images.

Another huge improvement can be done concerning the color information. While the Google Image search is mostly based on color histogram, our application does not use at all any color information, being only based on shape and distribution of visual words. This can be the subject of further researches. The simpliest idea would be to use the color version of PHOW descriptors, extracted on the three HSV image channels and stacked into a single descriptor. More advanced methods such as using color names [20] [21], combined with our method, can probably perform even better.

Finally, implementing the same algorithm in C can probably speed-up significantly the training and recognition phases of our application.

8.2 Deployment on a large database

Testing this algorithm on a small database is obviously not enough, the final goal being to perform the search on a large database. However, even with only 32 images in the database, the full training algorithm demanded more than 24h on a 2.30 GHz Quad-Core processor, which is impossible to extend on a larger database. At this step, we need to deploy it on a powerful parallelized system.

The ViCoS laboratory can provide three servers with 30 CPU units per machine. By adapting our code to make it compatible with this architecture, we can considerably speed up the training and recognition phases and test the application on a larger database.

4

Conclusion

During these two months in the laboratory, following several papers and advice from my tutor and researchers, I developed an algorithm to retrieve local similarity in images from a small database. I tried to apply a rigorous scientific approach in every step of development, and obtained really interesting and workable results. My tasks during the next month and until the end of the internship will be to integrate it to the existing distributed system, to make the application available as a web-service and to explore possible ways to improve this service.

This project fitted perfectly into my educational background and professional perspective. I could put in practice all the knowledge acquired during my study in IMAC and in the University of Ljubljana. It even allowed me to learn into more details particular Computer Vision algorithms and methods, and how to transform a research project into a working web-service available for a multitude of users. I also discovered the research profession and all the difficulties related to this position.

The end of this internship does not mean the end of the project. This application will be used in an advertising purpose, which means that my code will potentially serve as a starting point for future commercial implementations or partnerships between investors and the laboratory. This would be a great reward for me. As my code is divided into small building blocks working together, it can also be easily reused for other purposes, like classification or tracking. An eventual project of trafic sign recognition was already mentioned by my tutor.

In conclusion, I was really happy to work on this project, in a dyamic and stimulating research laboratory, where I could learn a lot. I did my best to provide a clean and documented code, and I hope this will be used and helpful for future projects and research topics. In any case, it was a great professional experience for me.

5

Appendix

References

1. F. F.-H. Nah. *A study on tolerable waiting time: how long are web users willing to wait?*, Behaviour IT,23(3):153163, 2004.
2. D. Tabernik, L. Čehovin, M. Kristan, M. Boben, A. Leonardis, *ViCoS Eye - a webservice for visual object categorization*, The 18th Computer Vision Winter Workshop, 2013.
3. R. Tao, E. Gavves, C. G.M. Snoek, A. W.M. Smeulders, *Locality in Generic Instance Search from One Example*, Conference on Computer Vision and Pattern Recognition, 2014.
4. H. Altwaijry, M. Moghimi, S. Belongie, *Recognizing Locations with Google Glass: A Case Study*, IEEE Winter Conference on Applications of Computer Vision (WACV), 2014.
5. A. Vedaldi and B. Fulkerson, *Vlfeat: an open and portable library of Computer Vision algorithms*, International Conference on ACM Multimedia 2010, pp. 14691472.
6. S. Maji and J. Malik, *Object detection using a max-margin hough transform*, Conference on Computer Vision and Pattern Recognition, 2009.
7. Jasper R. R. Uijlings, Koen E. A. van de Sande, Theo Gevers, Arnold W. M. Smeulders, *Selective Search for Object Recognition*, International Journal of Computer Vision, Volume 104 (2), page 154-171, 2013.
8. P. F. Felzenszwalb and D. P. Huttenlocher, *Efficient Graph-Based Image Segmentation*, International Journal of Computer Vision, 59:167-181, 2004.
9. F. Perronnin, J. Sánchez, T. Mensink, *Improving the Fisher Kernel for Large-Scale Image Classification*, European Conference on Computer Vision, 2010.
10. F; Perronnin and C. Dance, *Fisher Kernels on Visual Vocabularies for Image Categorization*, Conference on Computer Vision and Pattern Recognition, 2006.
11. J. Sánchez, F. Perronnin, T. Mensink, J. Verbeek, *Image Classification with the Fisher Vector: Theory and Practice*, International Journal of Computer Vision, 2013.
12. H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, C. Schmid, *Aggregating local image descriptors into compact codes*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2012.
13. D. G. Lowe, *Object recognition from local scale-invariant features*, International Conference on Computer Vision 2. pp. 11501157. doi:10.1109/ICCV.1999.790410, 1999.
14. A. Bosch, A. Zisserman, and X. Munoz. *Image classification using random forests and ferns*, International Conference on Computer Vision, 2007.
15. H. Abdi and L. J. Williams, *Principal component analysis*, Wiley Interdisciplinary Reviews: Computational Statistics, 2: 433459, 2010.
16. A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum Likelihood from Incomplete Qata via the EM Algorithm*, Journal Of The Royal Statistical Society, 39(1):138, 1977.
17. R. M. Gray and D. L. Neuhoff, *Quantization*, IEEE Transactions on Information Theory, Vol. 44, No 6, 1998.
18. H. Jégou, M. Douze, C. Schmid, *Product quantization for nearest neighbor search*, IEEE Transactions on Pattern Analysis and Machine Intelligence 33, 2011.
19. Matej Kristan, Roman Pflugfelder, Aleš Leonardis, Jiri Matas, Fatih Porikli, Luka Čehovin, Georg Nebehay, Gustavo Fernandez, Tomas Vojir et al., *The Visual Object Tracking VOT2013 challenge results*, ICCV2013 Workshops, Workshop on Visual Object Tracking Challenge, 2013.
20. J. van de Weijer, C. Schmid, J. Verbeek, *Learning Color Names from Real-World Images*, Conference on Computer Vision and Pattern Recognition, 2007.
21. J. van de Weijer, C. Schmid, *Applying Color Names to Image Recognition*, IEEE International Conference on Image Processing, 2007.
22. J. Dean, S. Ghemawat, and G. Inc, *Mapreduce: simplified data processing on large clusters*, In OSDI'04: Proceedings of the 6th conference on Symposium on Oprating Systems Design Implementation, USENIX Association, 2004.
23. T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, Inc., 1st edition, 2009.