M5 - Visual Recognition

Gerard Martí
Adriana Fernández
Eric López
Sergio Sancho

Autonomous University of Barcelona

1 Introduction

The objective of this project is to be able to estimate, detect and classify images. We will focus on two differennt tasks. First, we will tackle the scene recognition problem: given an image of a scene, the system should be able to label it. For example, given an image Fig. 1 our system should return the label "mountain".

The second task will be image retrieval: Given an image, the system should be able to retrieve from a database the x images more similar to the original (e.g. Fig. 2).

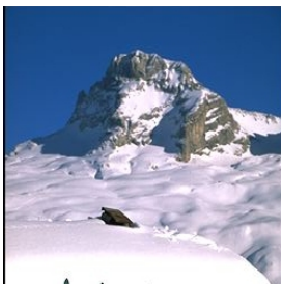

Fig. 1: Example of a mountain

We will try and experiment with several methods and compare their performance, for both problems. Finally, we will work on Convolutional Neural Networks and we will be able to get familiarized with how they work and their potential.

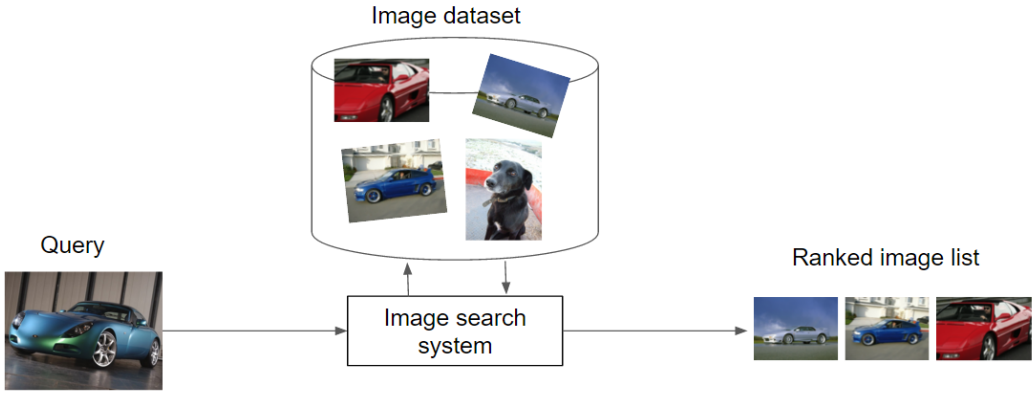

Fig. 2: Scheme image retrieval

2 Classification task

The classification process that we will follow during the project is shown in Fig. 3. There are several blocks in the process that have to be analysed using different techniques. In the following sections we are going to explain each block of the process.

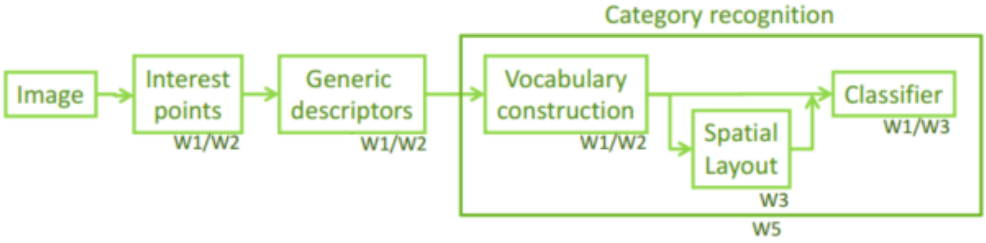

Fig. 3: Classification process

2.1 Interest points

To apply category recognition the first step is to find the relevant points that characterize the image. Those can be located in places such as corners, edges, etc. We have studied different detectors and experimented with them to decide which one works better by giving the other parameters a fixed value. The results are shown in Table 1.

Detector	SIFT	Dense	SURF	ORB	FAST	Harris-Laplace
keypoints per image	506	1849	551	400	2157	501
Total computation time(s)	185.3	167.9	776.9	503.9	551.9	146.1
Accuracy	0.747	0.795	0.801	0.713	0.862	0.749

Table 1: Detector comparison. The parameters where fixed to: Descriptor: SIFT, k = 5000, Classifier: SVM using 5 cross-validation, N samples = 50000

We can extract several conclusions from the results presented in Table 1:

- The computation time for Harris-laplace, SIFT and Dense is shorter, but they do not present the best results. Also, the methods that detect more keypoints per image seem to spent more time doing the extraction of visual word representations. Dense and Harris-laplace are really fast in computing the Local Descriptors.
- The best results are obtained for FAST and SURF. We will try to focus on those two detectors for experiments with early fusion and for other further experiments.

Fig. 4: SURF and FAST detection

2.2 Descriptor

The second step of the classification process was to describe the keypoints detected. We have analysed different descriptors as SIFT, LBP and HOG. Then, we have applied early fusion to obtain a more complex descriptor that take into account colour and shape. In Fig. 5 we can observe as SIFT and HOG descriptors are more accurate for this specific problem than LBP (which also has a lot higher computation time). SIFT seems to provide us with a better performance than the other two descriptors.

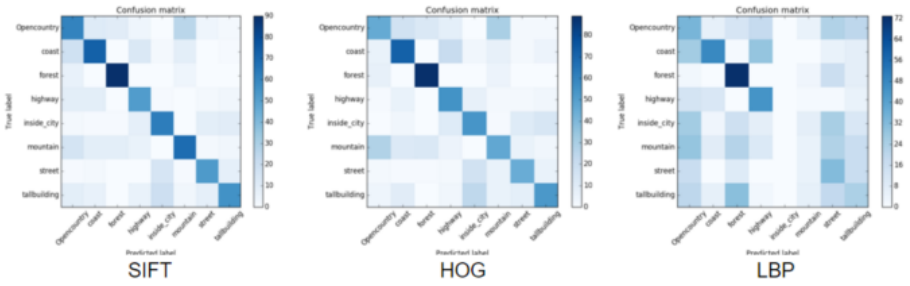

Fig. 5: Confusion matrices of SIFT, HOG, LBP descriptors

Descriptor	SIFT	HOG	LBP
Accuracy	0.630	0.571	0.386

Table 2: Detector comparison. The parameters where fixed to: Detector: SIFT, k = 5000, Classifier: SVM using 2 cross-validation, N samples = 50000

Early fusion was applied using color and shape descriptors, as you can see in Table 3. On the table, all the experiments that we performed are presented. We can observe several things:

1. In some of the fusion, we obtained a poorer performance than when using the single descriptor (mainly when using the Color histogram descriptor). Sometimes though, the fusion was beneficial (for example, for Dense-SIFT, with a great 80 % accuracy).

2. The PCA reduction did help us to improve our performance, but it also made us lose some accuracy. This may have been too due to a bad choice of number of components.
3. In general, Color naming tends to give us a better performance compared to using color histogram descriptor.

Detector - Descriptor	Color descriptor	Accuracy (D-C-F) (%)
Dense - SIFT	Color histogram	80 - 30.73 - 81.78
Dense - SIFT	Color naming	79.6 - 43 - 80.02
SURF - SURF	Color histogram	37 - 44 - 44
SURF - SURF	Color naming	36 - 52 - 42
FAST - SIFT	Color histogram	84 - 40 - 84
FAST - SIFT	Color naming	66 - 53 - 64
SIFT - SIFT	Color histogram	68 - 45 - 49

Table 3: Early fusion comparison. For Accuracy, (D-C-F) means Descriptor - Color - Fusion, and references the accuracy that a Linear SVM model achieves using said descriptors

2.3 Vocabulary construction

For almost all the experiments we have been using k-means to compute the vocabulary, but we also did experiments to try a soft-assigment approach, using Gaussian Mixture Models to compute it. In Table **??** one can see the results obtained.

Method	k = 16	k = 32	k = 64	k = 80	k = 128	k-means
FAST+SIFT, Linear SVM	0.71	0.802	0.825	0.824	0.832	0.86
DENSE + SIFT, Linear SVM	0.645	0.695	0.732	0.734	0.738	0.81

Table 4: Results for GMM soft-assigment for computing the codebook, with nsamples = 50000 and cross-validation

GMM does not outperform k-means in any of the experiments, so we decided to keep experimenting using k-means. Also, the computation time is really high for bigger k, but the performance obtained do not justify its costs.

2.4 Spatial Layout and Classifier

Finally, after we have described each image and obtained its features, we need to a classifier to assign that label to an image. We have analysed different types of classifiers, such as KNN and SVM. Also we have studied different variants of SVM, such as Linear SVM, HISVM and SPM Kernel SVM. We also implemented spatial pyramids, to model the spatial information of each image and add them to the features, and intermediate fusion, to be able to combine different descriptors after they are computed.

The comparison among the different kernels and options is presented in Table 5

Detector	Descriptor	Classifier	No SPM	SPM2x2	SPM2x2 + IF	SPM3x1 + IF
FAST	SIFT	Linear SVM	0.8289	0.8327	0.8512	0.8587
FAST	SIFT	HISVM	0.8550	0.8674	**0.8909**	0.8599
FAST	SIFT	SPM Kernel SVM	-	0.8624	0.8674	0.8636

Table 5: Fast+SIFT+Colornaming, weights 0.35-0.65 for intermediate fusion. SVM kernels comparison. Comparison of applying or not spatial pyramids and intermediate fusion in our system with different SVM kernels (k = 1000 samples = 50000, no PCA).

We can see that the spatial pyramid had improved the results in every case, and that intermediate fusion has a really positive result in our system. SURF as a descriptor seems to be not adequate for this kind of application, it always show worse results, and that is why we discarded it and decided to use FAST+SIFT.The SVM kernel that offers the best performance is HI-SVM.

Applying different weights to the combination of the vocabularies different improvements are obtained, we cannot conclude any dependence. In this case, the best result was weighing more colour than shape, with the weights indicated (results of this experiments can be seen on Fig. 6).

This behaviour could be caused by the difference between the categories to classify. In some cases shape is more important than colour, and in other on the contrary. Therefore, by changing weights you there are a trade off between failures and successes over the different categories.

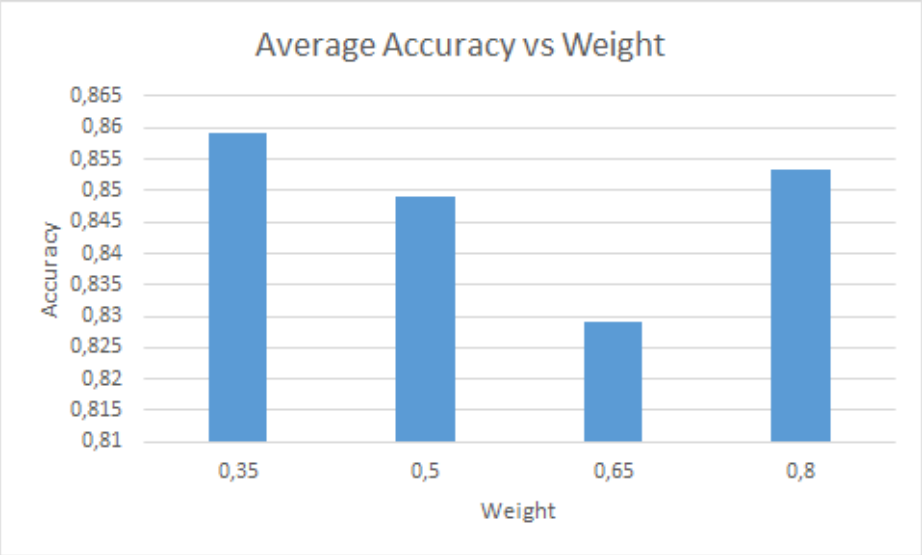

Fig. 6: Average score achieved with different weights. Horizontal axis represent the importance of the shape over the colour.

Regarding soft assignment with GMM do not alter the results significantly, more testing could be needed to have a final answer.

We tried both 3x1 and 2x2 spatial pyramid shapes, but the best results were obtained with 2x2 pyramids. PCA was discarded because although the computational cost was lower, we could not find a good number of components to represent our data. The results obtained applying PCA were always worse.

The system works pretty well, offering an accuracy of **89.09%** in our best configuration.

3 Image Retrieval

Done the classification task of the lab we move to another related objective: Image retrieval. The idea is to input an image query and the system

should be capable of retrieve from its database the most similar images to the query.

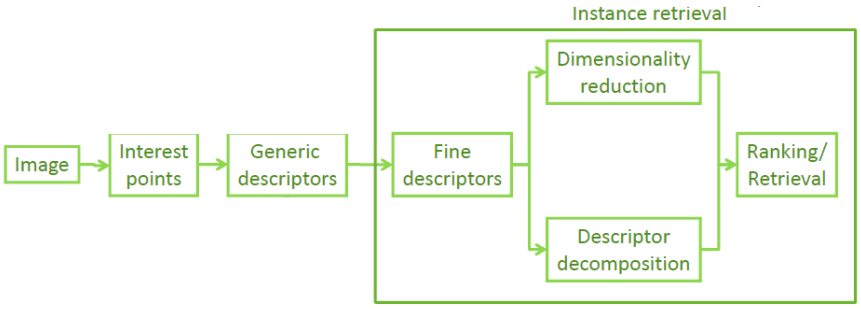

Fig. 7: Image Retrieval process.

3.1 Global descriptors

The first test we have done were using types of global descriptors after the ordinary descriptors applied over the interest points in the previous step. The descriptors we have used were Vlad, Fisher and BOW (as a descriptor).

To compute similarities we have used as first approach Fast NN and Product Quantization. We have done the testing using every possible combination.

The accuracy performances of each descriptor can be seen on Table 6. The bests results were achieved using Fisher descriptor and Fast NN as similarity method, outperforming the others in every measure (Top 1, Top 5, Top 15, Top 20 and Top 25).

We have use FAST+SIFT as a detector+descriptor combination because is the one that provided us with better results in the previous weeks. For product Quantization, we use adc as the distance function, as in our testing it has shown on par results with sdc distance but much better performance.

3.2 Applying LSH over the global descriptors.

Locality-sensitive hashing (LSH) is a method to reduce the dimensionality of high-dimensional data, so it is perfect to improve the quality of our

Performance	Fast NN			Product Q.		
(Accuracy)	Vlad	Fisher	BoW	Vlad	Fisher	BoW
Top 1	0.69	0.78	0.67	0.66	0.51	0.52
Top 5	0.74	0.82	0.74	0.67	0.55	0.56
Top 10	0.72	0.80	0.72	0.66	0.55	0.57
Top 15	0.70	0.79	0.71	0.61	0.56	0.57
Top 20	0.69	0.78	0.70	0.60	0.56	0.56
Top 25	0.69	0.77	0.69	0.60	0.56	0.56

Table 6: Results obteined on image retrieval. Always using FAST+SIFT and a vocabulary size of K = 1000 (except for fisher, which uses GMM and we use K = 64).

system if we apply them over the global descriptors, which have a very large dimensionality.

If we take a look to the accuracy scores we have obtained (Table 7) we can realise that the results we have obtained (using Product Quantization in both cases) are pretty equal in both cases, with a bit lower results when we use LSH. Nevertheless, the performance time was not affected to much by the applying this method in our case.

Performance	Product Q.			Product Q. + LSH		
(Accuracy)	Vlad	Fisher	BoW	Vlad	Fisher	BoW
Top 1	0.66	0.51	0.52	-	0.51	0.49
Top 5	0.67	0.55	0.56	-	0.53	0.52
Top 10	0.66	0.55	0.57	-	0.53	0.54
Top 15	0.61	0.56	0.57	-	0.54	0.55
Top 20	0.60	0.56	0.56	-	0.54	0.55
Top 25	0.60	0.56	0.56	-	0.54	0.54

Table 7: Comparision of results obteined using or not LSH combined with Product Quantization.

3.3 Inverted files on product quantization.

Inverted files is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a

document or a set of documents. Our objective by using this method was to improve the performance time again.

Nevertheless, the results were not as expected and the performance was not improved. Regarding the accuracy results, as we can observe on Table 8, the scores we have achieved were similar, a in some cases a bit better than the previous case.

Performance	Product Q.			Product Q. + Inv. Files		
(Accuracy)	Vlad	Fisher	BoW	Vlad	Fisher	BoW
Top 1	0.66	0.51	0.52	-	0.48	0.55
Top 5	0.67	0.55	0.56	-	0.56	0.59
Top 10	0.66	0.55	0.57	-	0.56	0.59
Top 15	0.61	0.56	0.57	-	0.56	0.59
Top 20	0.60	0.56	0.56	-	0.56	0.59
Top 25	0.60	0.56	0.56	-	0.55	0.58

Table 8: Comparision of results obteined using or not Inverted Files combined with Product Quantization.

3.4 Visual results and conclusions

Doing an overall view of this task, we found out that the best configuration in our case for the image retrieval system was Fisher + FastNN, achieving an accuracy score of 0.82 on Top 5 evaluation.

In Figures 8, 9 and 10 we can see examples of the output given with an aleatory image query.

4 Convolutional Neural Networks

In the last weeks of the project we tried using Deep Convolutional Neural Networks to the problem of scene recognition and classification. Using the same dataset as in section 1, we will use first a pretrained network and check the performance we obtain, and then we will try to design our own network and try to achieve the best possible results. We will use MatConvNet, a MATLAB toolbox that implements Convolutional Neural Networks.

Fig. 8: This query is really interesting: there are several dogs, and one of the dogs has really fluffy hair. The system give us dogs which has a similar fluffy hair, but also dogs near patches of grass that have a similar texture.

Fig. 9: This query is a closeup of a face of a dog. Most of the images obtained are also of a closeup of a dog face, where the nose, eyes and ears are visible.

4.1 Pretrained Network - AlexNet

Working with Deep Convolutional Neural Networks have some major drawbacks: First, we need a really huge dataset for the network to be

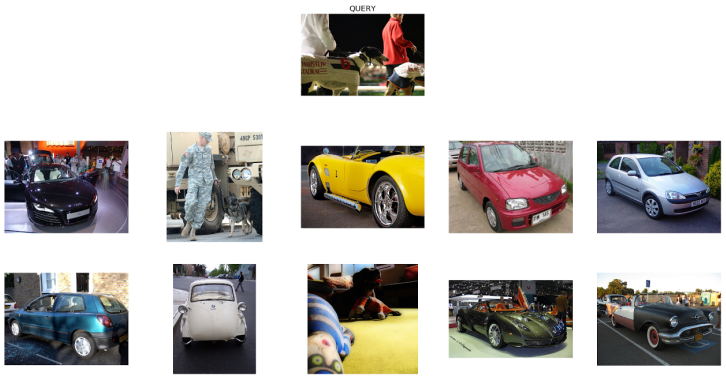

Fig. 10: An example of bad behaviour of the system. The piece of cloth that the dog is wearing modifies the dog shape, and it seems that the classifier is confused by this and gives us cars in return. Also, the dog's perspective in that picture is not common.

trained properly (as it can have milions of different parameters) and, secondly, it is very computationally expensive to fully train a neural network, needing a high-end graphical card to handle all the computations in a reasonable time. For this reason, our first contact with Convolutional Neural Networks was using a pre-trained network. We used AlexNet (Figure **??**, pretrained with over one million images from the dataset ImageNet.

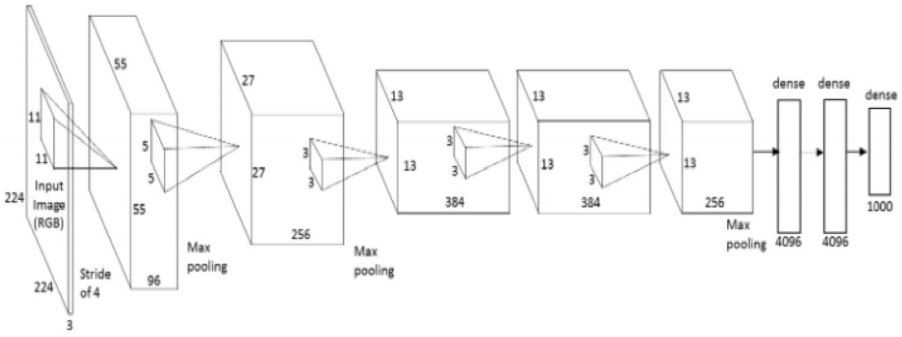

Fig. 11: AlexNet

In the figure we can see the different type of layers that are present in a convolutional network:

- **Convolutional Layers**, which performs convolutions over its input with learned kernels through training.
- **RelU layers**, which applies a non-linearity after a convolutional layer, which is necessary for a neural network (otherwise we could just use 1 layer, as we could combine every layer in a linear combination)
- **pooling layer**, which is convolution with a max filtering, useful to gain robustness to the spatial location of features.
- **Fully Connected Layers**, which are layers where the input and the output neurons are all connected to each other.

There are other types of layers, like Dropout or batch normalization, but we will not use them until later, so they will be mentioned there.

As we said before, we cannot train the full network because it will take too much time and we do not have enough data. What we can do is train only some of the layers in the network. This is called **fine-tuning**. We will try to fine-tune only the last two fully connected layers of the network. We will configure two networks, which we will call AlexNet and AlexNe_Small, which use input images of 227x227x3 and 37x37x3, respectively. After training for 10 epochs or iterations, this the error function we obtained:

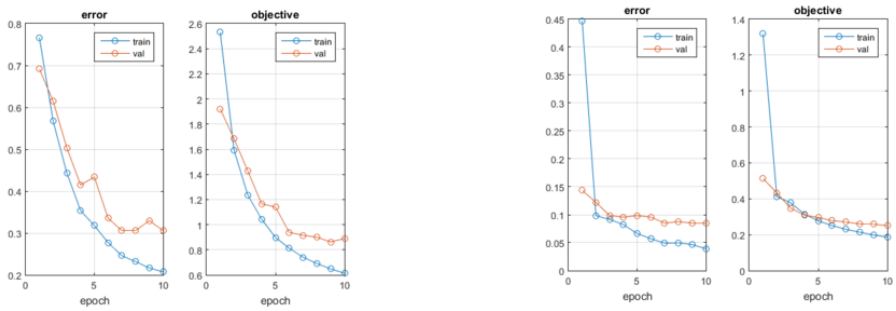

Fig. 12: Error and objective plots for both AlexNets, (left smaller, right larger) for 10 epochs

We can see that the bigger network reduces the error much more faster than the other one, and that in both network we have a small amount of overfitting (as the results in the validation set are worse than the

obtained with the training set). The accuracy was **0.71** for AlexNet_small and **0.9281** for normal AlexNet. We can say that with bigger images we obtain better results. It is easy to see why: with higher resolution, it is easier for the network to extract better features and be able to distinguish the images better. Of course this comes with a price: much bigger computation effort. It take almost 4 times more time to train the 10 epochs for the AlexNet network (More than one hour) than for the AlexNet_small network (about 15 minutes).

Commenting on the accuracy, we obtain a **0.9281** on the finetuned AlexNet, which is better than any of the previous results, and taking into account that this was done with a network that was pretrained with a different dataset, it shows that CNN are a really powerful tool for image and scene recognition.

4.2 Extracting features from a CNN

After having fine-tuned the AlexNet CNN, we wanted to try a different approach. We could put an image into the network, and extract the output of some of the layers in the convolutional neural network. This output could be use as features for that specific image. We will then feed that data to our Bag-of-Words framework.

For the two last fully-connected layers, we will just take the 4096 features as a feature vector and use every one of them to train the classifier. So, we feed each image to the CNN, get the output of the fully connected layers and feed those ouput to the SVM. For the Convolutional layers, the procedure is a bit more complicated. For a convolutional layer of size (m*n*d) we consider as a feature vector each one with a length equal to the depth of the layer (d), and we have m*n vectors, where m and n are the sizes of the vector. Extracting those vectors from each image, we can build a codebook and proceed as before.

We will use the configuration that gave us the best results in section one. This is, using a Linear SVM, with cross-validation to get the best parameters, and, for the codebooks, we have a number of samples of 50000 and k = 5000. By using the same parameters as previous experiments, we can directly compare the performance of our best descriptor+detector combination with those CNN features.

The results obtained can be seen in Table 9

From this results we can make two observations: the fully connected layers are the ones which give us the best performance, and the deeper the features the better they are (although the accuracy using the layer Conv5 is low, it is probably a bug. We have not been able to find its source, but

Layer	Conv1	Conv2	Conv3	Conv4	Conv5	Fc6	Fc7
Accuracy	0.8327	0.8859	0.9083	0.9058	0.6282	**0.9504**	0.9368

Table 9: My caption

it should not be that low). The performance observed is better than any of our experiments in Section 1, and also better than just letting the CNN classify the network. We conclude then, that this procedure is an excellent way to implement a solution for this problem.

4.3 Designing and training our own network

During the assignments of this module, we have created our own deep convolutional neural network. We improved from the standard deep network architecture by means of implementing problem specific configurations from online competitions such as *Kaggle*.

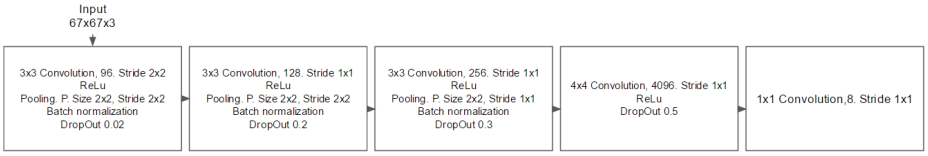

Fig. 13: Designing our own network. The global parameters are: batchSize = 128, numEpochs = 40, LearningRate = 0.0001 and after 25 steps 0.01, weightDecay = 0.00002 and momentum = 0.9.

Figure 13 depicts the architecture deployed for this assignment. It consist of five layers: the first one receives an input of 67x67x3 and it applies a 3x3 convolution to the input with a stride of 2x2, and the number of output channels are 96. Then it uses ReLu, applies a 2x2 Pooling with a stride of 2x2, batch normalization and a little bit of dropout in order to avoid the early stage overfitting. The next layer structure is basically the same, but the stride is 1x1, the number of the output channels are 128 and the dropout is set to 0.2. The third layer has the same structure, but the number of channels are 256 and the dropout is higher. The following layer is simpler: it applies a 4x4 convolution with a stride of 1x1 and the number of output channels are 4096. Therefore, it applies a ReLu and the

highest dropout of the system. Finally, the number of output channels are 8.

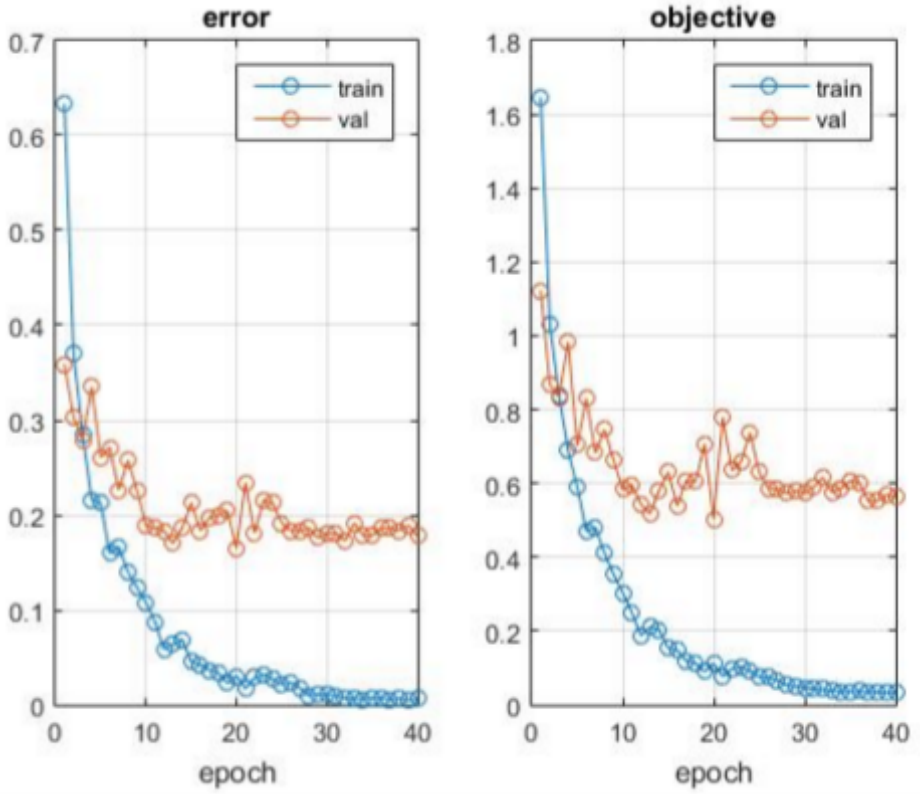

Fig. 14: Error and objective plots (train/val) with Dropout + Batch normalization settings.

With this configuration we have achieved an accuracy of **0.86**. As shown in Figure 14 the optimum number of epoch is around 20.

As Table 10 shows, we tried different configurations for our system. Dropout and Batch normalization techniques indeed improve the accuracy. Finally, we have included the last fully-connected layer of the network to the Bag of Words framework. Using this technique, our design achieved an accuracy of **0.8488**.

Technique	Test accuracy
Vanilla	0.7794
Dropout	0.7819
Batch normalization	0.8327
Dropout + Batch normalization	0.8600
Bag of words with last fc layer	0.8488

Table 10: Shows the difference in terms of accuracy between Dropout, Batch normalization or combining both. The momentum is set to 0.9, it increases the system performance.

5 Conclusions

During our work in this project we have been able to study and apply many different possibles approaches and solutions to the classification and retrieval problems. We have performed a fair number of experiments to compare between the methods and find the best parameters. This has helped us to get familiarized and comfortable in this kind of projects, where the structure and order of the experiments is as important as programming the algorithms, or more. This is the main skill we got from this project.

Regarding results, we found that Using the spatial characteristics of the image through spatial pyramids and the color information through interemdiate fusion gives us the best results in classification. For retrieval, Fisher is the method that gives us the better accuracy, with some very promising results. And finally, we have seen that CNN are a very powerful method for image classification, as the three different ways we have used them (finetuning, feature extracting and programming our own) gives really good and competitive results, and also a promising margin of improvement, with a better architecture and with more data for training.

During the project we had some problems with some of the provided files and had to spend a great amount of time debugging the code instead of experimenting, but, looking at it in a positive light, that helped us in understanding the code and the algorithms better.