Machine learning methods for image classification: a practical survey

Arantxa Casanova, Belén Luque, Anna Martí, Santi Puch

Universitat Autònoma de Barcelona
{arantxa.casanova, belen.luque, anna.marti, santiago.puch}@e-campus.uab.cat

Abstract. In this work we perform a thorough practical survey of the most well-known machine learning techniques used in image classification. We consider two frameworks for the image classification problem; the first is based on handcrafted methods, whereas the second is based on data-driven techniques. We assess and report the performance of all the surveyed methods in a scene recognition problem.

Keywords: Image processing, computer vision, machine learning, deep learning, bag of words, spatial pyramids, fisher vectors

1 Introduction

Image categorization is the pattern classification problem that consists in assigning one or multiple labels to an image based on its semantic content. This is a very challenging task as one has to cope with inherent object and scene variations, as well as changes in viewpoint, lighting and occlusion. However, it also has been and still is an active topic of research in the compute vision field.

For that reason, many approaches to image categorization have been proposed throughout years of research. One of the most famous models for image classification is the bag of visual words[1], or bag-of-visterms, in which local image features are treated as words. Several improvements have been proposed on top of this model, such as spatial pyramids[2] or Fisher vectors[3]. More recently, the advances in the field of deep learning have made possible to achieve classification accuracies close to that of humans[4], specifically by convolutional neural networks such as VGG[5] or GoogLeNet[6].

In this paper we present a practical survey of the aforementioned techniques and methods, and we report their performance over a scene-recognition problem.

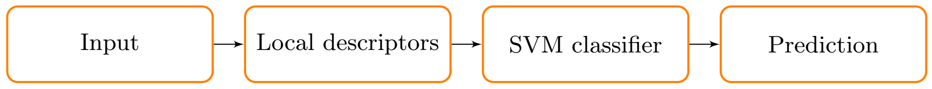


Fig. 1: Basic image classification pipeline

2 Dataset

The dataset used for this work is a subset of the Urban and Natural Scene Categories Dataset[7] that consists of 2713 images. All the images in the dataset can be categorized in one of the 8 available classes: *coast*, *forest*, *highway*, *inside city*, *mountain*, *open country*, *street* and *tall building*. The whole dataset is split in train, validation and test sets, which are composed of 1706, 200 and 807 images respectively.

3 Basic image classification

In a basic image classification approach, the image is first represented as several local features, referred as descriptors, in order to be classified.

As seen in Figure 1, these descriptors are fed into a classifier, which assigns them the probability of pertaining to the 8 different classes. With these probabilities, a prediction can be done by assigning a class to the input image.

To see what kind of descriptor is better suited for our dataset, we trained a Linear SVM classifier using different local descriptors with 100 features. As it is shown in Table 1, the best local descriptor for our dataset is SIFT[8].

At this point, we had an image represented by 100 features of 128 dimensions, which was very time consuming. To solve this problem, we applied PCA to reduce the dimensionality of the features. Figure 2 shows the evolution of the accuracy and time, depending on the dimensionality reduction, ranging from 0 o 100. We can see in the figure that the accuracy increases following approximately

Local descriptor	**Accuracy**	**Time**
SURF	36.22 %	400.06s
ORB	16.98 %	**214.897s**
SIFT	**38.41 %**	646.602s
BRISK	15.61 %	407.226s

Table 1: Performance of the system with four different descriptors of 100 features and a linear SVM classifier with parameter C=1

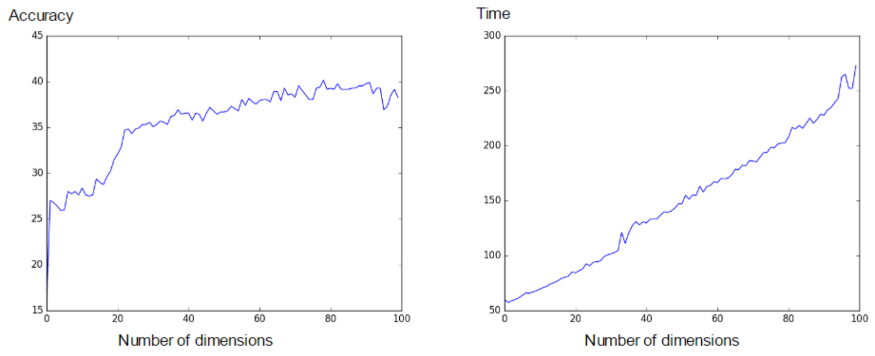


Fig. 2: From left to right: Accuracy vs. dimensionality reduction. Time vs. dimensionality reduction.

a logarithmic function, but the time increases linearly.

Knowing this we decided to set the number of dimensions to 23 as a good trade-off between accuracy and time, so the image was represented as 100 features of 23 dimensions.

Using the computed features, an SVM classifier was trained. As there are several possible kernels functions for this kind of classifier, we trained several versions of the SVM using different kernels to know which was more appropriate for our problem. Furthermore, we did a parameter sweep to find the optimal parameters for each of the tested kernels. The parameters to optimize for each kernel are shown in Table 2 and the results obtained, in Table 3. Our final kernel function was the Radial Basis Function (RBF), as it performed the best.

When an image is fed to the classifier, it assigns a vector of 8 probabilities to each feature in the image. To decide the class of the image we tried two different approaches:

SVM kernel	**Gamma**	**D**	**R**	**C**
Linear	-	-	-	[1, 9]
Polynomial	1/23	[1, 9]	[1, 4]	[1, 9]
RBF	[0, 1]	-	-	[1, 9]
Sigmoid	[0, 0.5]	-	[-2, 2]	[1, 9]

Table 2: Parameters of the SVM kernels to optimize.

SVM kernel	**Optimal Parameters**	**Accuracy**
Linear	C=4	34.32 %
Polynomial	C=3, D=0.1, R=0.1	44.61 %
RBF	C=5, Gamma=0.1	**47.09 %**
Sigmoid	C=8, Gamma=0.1, R=-2	44,86 %

Table 3: Performance of different SVM kernels with their optimal parameters.

- **Hard assignment**: Each feature contributes with a unique vote for the class with a higher probability. The image is assigned the most frequent class.
- **Aggregation of probabilities**: Each feature contributes to the 8 classes with a probability. The contributions of all the features are aggregated and the image is assigned the most frequent class.

We decided to use Aggregation of probabilities as it does a soft classification taking into account the probabilities of all the classes instead of doing a hard classification, and in consequence, it performs better.
Considering the best parameters for every block in the pipeline (see Figure 1), the accuracy we obtained with this basic image classification framework was 54,28%.

4 Bag of Words framework

So far we describe images as a set of SIFT descriptors, but these features end up being classified individually to the 8 possible classes. In a last step, the class of the image is decided, based on the classification of its features. However, we wanted to improve the system by following another approach, where a global description of the image was used to decide its class, rather than classifying each feature independently and then aggregating these individual decisions. We accomplished this by using the Bag of Words (BoW) framework.

In order to represent an image with the BoW model, a codebook has to be defined in the first place. To do this, we used the training set to divide the feature space in different clusters of similar features, which were the visual words forming the codebook. Once we had a set of visual words, every feature extracted from an image at any point in the pipeline could be mapped to the closest word. The representation of an image using BoW is given by a histogram of the visual words in it, which is more compact and efficient than the one we previously had. In order for the SVM to compare more precisely two histograms, we used an intersection kernel instead of RBF.

The BoW framework attempts to describe images in a more global way, and that is why it works much better with dense SIFT, where a grid is defined over the

image and a SIFT descriptor is extracted on each of its locations. By doing this, we obtained features of all the objects in the image, which allowed for a global description rather than focusing on the most important objects.
To test this approach we defined several experiments combining the following options:

- SIFT: normal or dense.
- Normalization: none or L1.
- SVM kernel: RBF, intersection or pyramid kernel.

For each of the combinations, we used a 4-fold cross-validation with 100 samples per fold and a random search to find the optimal hyperparameters. The range of the codebook sizes tested was [2^7, ... , 2^{15}]. For the SVM's kernel, the parameter 'C' ranged from 10^{-4} to 10^3 and for the 'gamma' of the RBF kernel, from 10^{-3} to 10^1, both using logarithmic spacing.

When using normal SIFT, with 100 key points, the best accuracy obtained was of 54.3%, which means that we were not improving the system with respect to the basic image classification approach. However, the experiments with dense SIFT provided a maximum accuracy of 79.31% (gain of 32%). This result was obtained with an 8x8 grid on dense SIFT, a reduction of the features to 60 dimensions using PCA, a codebook of 512 visual words, no normalization of the descriptor and an intersection kernel with 'C' 0.063.

5 Beyond Bag of Words

At this point, our classifier was working at almost 80% of accuracy, but there were still some aspects that could be improved.

5.1 Spatial pyramids

In both the basic image classification system and the BoW approach we treated all the extracted features equally, independently of their location in the image. The spatial information was lost, as the location in which an object should be found in order for the image to belong to a certain class was not taken into account. With the use of spatial pyramids, we expected to solve this loss of information.

Implementing spatial pyramids consists on dividing the image in several grids of different resolutions 0,...,$\mathcal{L}$ and computing the histogram of visual words for each of the resulting regions, to finally concatenate them in a sole descriptor.

Furthermore, a new kernel based on the intersection kernel is defined. Being $\mathcal{I}^\ell$ the intersection kernel at the resolution level ℓ, equation 1 defines the kernel function for images X and Y.

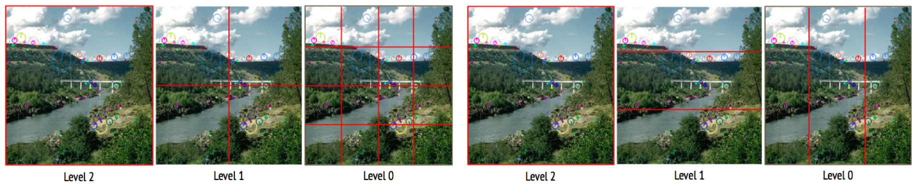


Fig. 3: Two different configurations of the spatial pyramid's grid. On the left, a total of 21 concatenated histograms would form the image descriptor. On the right, only 7 histograms would be used to create it.

$$k^L(X,Y) = \frac{1}{2^L} I^0 + \sum_{\ell=1}^{L} \frac{1}{2^{L-\ell+1}} I^\ell \tag{1}$$

We tested several configurations of the grid, like the ones shown in Figure 3. The uniform configuration with 21 regions yielded the best results, with a 6% improvement in accuracy when using normal SIFT, although it only provided a 1% gain with dense SIFT, as dense SIFT is already descriptive enough for the system.

5.2 Fisher vectors

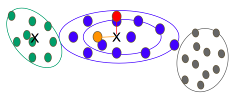


Fig. 4: The orange and red samples contribute equally to the blue visual word.

The next improvement we wanted to apply was to take into account the location of a sample within the probability distribution of a visual word. Up until now, the two samples in Figure 4 contributed equally to the blue visual word, as their distance to the center of the cluster is the same, although their location with respect to the probability distribution is clearly different. In order to incorporate this information, we used Fisher vectors, which provide knowledge about the mean and co-variance of the local descriptors.

In this case, the experiments were carried out using different combinations of the following:

- Dense SIFT grid sampling: 8x8 or 16x16.
- Normalization: L2 or power normalization.
- SVM kernel: intersection or pyramid kernel.

Using the same cross-validation configuration as in BoW and a random search, we obtained a best accuracy of 84.51%, almost a 5% more than using BoW with spatial pyramids.

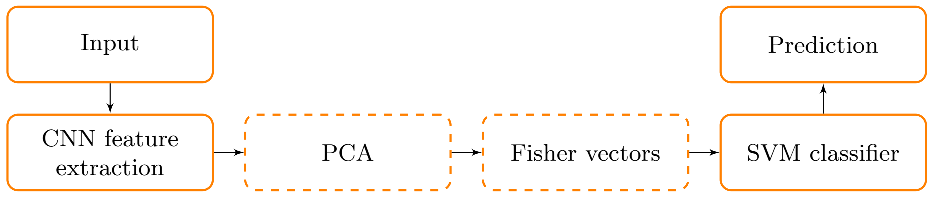


Fig. 5: CNN and SVM pipeline

6 From hand-crafted to learnt features

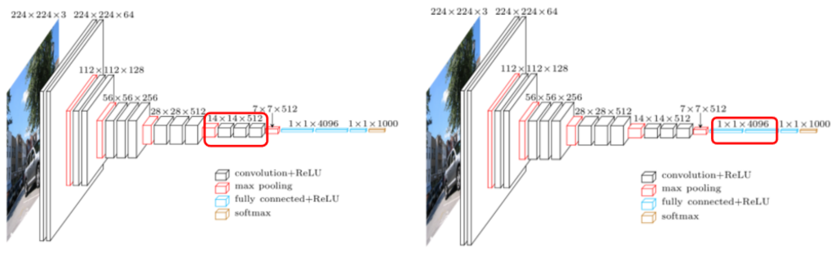
Instead of constructing a descriptor with some criteria defined by hand, such as SIFT, ORB or SURF, we wanted to use already learnt features by a pre-trained Convolutional Neural Network: VGG-16.

VGG-16 is one of the most well-known Convolutional Neural Networks (CNN) architectures, mainly due to the fact that this network achieved one of the best accuracies in the ImageNet Large-Scale Visual Recognition Challenge (ILSVR) on 2014[5]. Its architecture, trained with more than a million images, is based on a stack of convolutional layers with 3x3 receptive field kernels and 2x2 max-pooling layers with stride 2. These convolutional layers are followed by 3 fully connected (FC) layers of 4096, 4096 and 1000 units respectively, having the last one a softmax activation that outputs the probability of the image belonging to each of the 1000 classes in the ImageNet dataset.

As seen in Figure 5, we were only replacing the feature extraction block, where we were obtaining dense SIFT descriptors, by the CNN. The CNN feature extraction block was built with two different approaches:

1. Using the last convolutional block, marked in red in Figure 6a, which outputs 512 different 14x14 feature maps per image. Then the number of feature maps is reduced using PCA to only keep 52. Finally, the fisher vectors of these features maps are computed to classify them with an SVM.
2. Using the last fully connected layer, marked in red in Figure 6b to directly obtain a feature vector of 4096 dimensions per image and feed it directly to the SVM.

The final parameters of the two approaches were selected with a 3-fold cross-validation with random search, getting 40 sampled elements per fold. We chose an intersection kernel for the SVM, based on the previous experiments with Bag of Words. The parameter C was chosen between 10^{-5} and 10^{3}. In the case of the first approach, which also uses PCA and Fisher vectors, the latter had been tested with GMMs (Gaussian Mixture Models) of 16, 32 and 64 gaussians. The PCA dimensionality reductions tested are 256, 205, 154, 102 and 52.



(a) From last convolutional block (b) From last fully connected layer

Fig. 6: Feature extraction from VGG-16 model

Using the last convolutional block as features, 32 gaussians GMM, a PCA reduction up to 52 dimensions and a parameter C of 0.04283 yielded the best accuracy, according to the cross-validation results presented in Table 4. Using the last fully connected layer as features, the best parameter C is 0.000801, reaching a mean accuracy of 92.5% with a standard deviation of 1.63%.

GMM gaussians	Accuracy (mean)	Accuracy (std)	C
16	92.45%	1.47%	0.003107
32	**93.09%**	**1.76%**	**0.04283**
64	93.01%	1.57%	0.000680

Table 4: Cross-validation results using the output from the last convolutional block as features

7 Fine tuning of pre-trained CNNs

Until this point all the presented methods rely on a pipeline primarily based on a feature extraction block and a classification block. In Section 6 we leveraged CNNs as feature-extractor; however, CNNs are known for being powerful end-to-end systems. In this section we aimed at using an end-to-end CNN, but instead of creating and training one from scratch, we fine-tuned an existing one: VGG-16. Figure 8 shows the detailed architecture of VGG-16.

Input
CNN
Prediction

Fig. 7: CNN pipeline

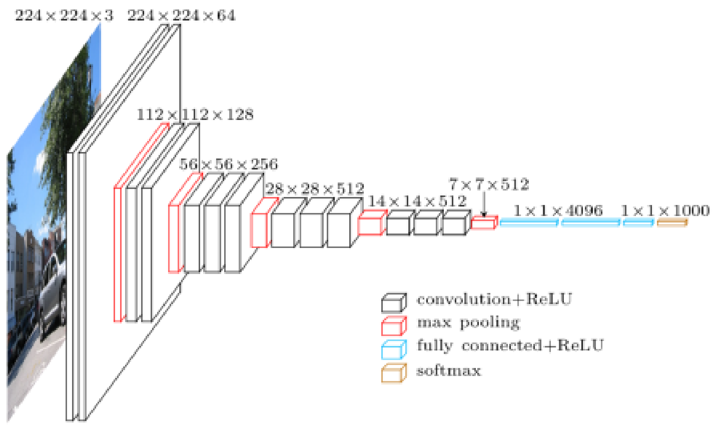
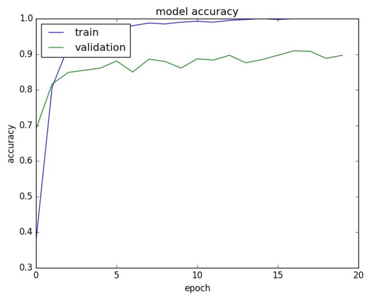


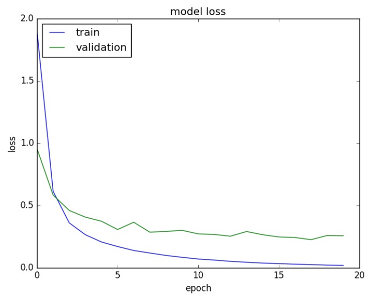
Fig. 8: VGG-16 architecture

The first experiment we conducted was to load the VGG-16 model with the ImageNet pre-trained weights and to replace the last FC layer for one that had 8 output units, so that the architecture could be adapted to our problem: classify images in one of the 8 scene categories. All the layers except the last one were frozen, that is, their weights were not updated as a result of backpropagation, hence we only trained the last FC layer. As a result, we obtained a validation accuracy and loss of almost 90% and less than 0.5, respectively (see Figure 9).

Given the almost unsurpassable accuracy of this model, we decided to cut the VGG-16 model earlier, so that we did not have all the generalization power of the architecture trained on a dataset as big as ImageNet. That way, we had to train the last layers of the architecture —the ones that represent more abstact



(a) Training and validation accuracy



(b) Training and validation loss

Fig. 9: Training and validation metrics captured during training when fine-tuning a VGG-16 network with a replaced softmax layer.

features— to obtain again this generalization power and therefore have a good performance. Concretely, we cut the network after the last convolutional layer in block 3, and we added a 4 by 4 max-pooling layer, two FC layers of 2048 units each, and a softmax layer with 8 units afterwards. The training scheme consisted on training first the weights of the added FC layers by freezing the layers from the original architecture, and then unfreezing the layers and updating all the weights in the network.

In order to assess the influence of data augmentation on the performance of the CNN, we created two new training sets with only 400 and 40 images that preserved the priors from the original dataset. Table 5 shows the results of fine-tuning the aforementioned architecture with several combinations of datasets and data augmentation. The degrees of freedom considered in the data augmentation procedure were rotation and horizontal flip.

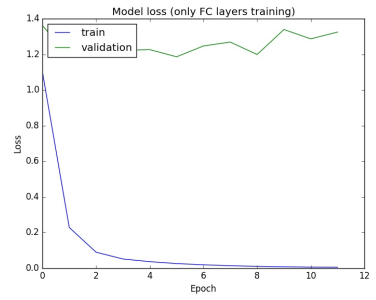
We achieved an accuracy near 73% in the validation set. However, there was a clear evidence that the system was overfitting, as the validation loss was not decreasing alongside the training loss, but instead it was stabilized (Figure 10a). We decided to include L2 regularization of $\lambda = 0.01$ in all the weights included in the new layers, and a dropout layer after each of the FC layers with dropout probability $\rho = 0.5$. The training loss for this architecture was the one in Figure 10b, which is clearly better, given that the validation loss keeps decreasing as long as the training loss decreases. As a result, we obtained 76% of accuracy in the validation set.

Finally, we wanted to optimize some of the hyperparameters associated with the fine-tuning of the network, namely, the optimizer, the learning rate, the regularization value and the batch size for all the optimizers, and also the momentum value and whether or not Nesterov momentum was used for the case of SGD optimizer.

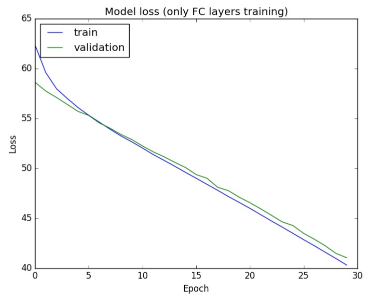
To accomplish that, we performed a random search of 5 iterations for each of the 3 selected optimizers (Adam, SGD and RMSProp), where the ranges of values considered for each hyperparameter were:

Dataset	**Data augmentation**	**Training samples**	**Accuracy**
40	x10	400	38.8%
40	x50	2000	39.38%
400	No	400	65.25%
400	x5	2000	72.88%

Table 5: Performance of the fine-tuned network for different combinations of datasets and data augmentation.



(a) Training and validation loss without dropout nor L2 regularization



(b) Training and validation loss with dropout and L2 regularization

Fig. 10: Training and validation loss captured during training when fine-tuning the modified VGG-16.

- **Batch size**: $5, 10, \cdots, 45$
- **Learning rate**: $[10^{-7}, 10^{-4}]$
- **Regularizer value (λ)**: $[10^{-4}, 10^{0}]$
- **Momentum**: Uniform pdf [0, 1]
- **Nesterov momentum**: True, False

Due to out-of-memory related problems the search was not as exhaustive as it should have been, and as a result we could not find an appropriate set of hyper-parameters that could improve the performance of the system. For that reason, we fine-tuned the final system using the default parameters: Adam optimizer, learning rate of 1e-5, regularizer of 0.01, dropout of 0.5, and batch size of 20. We used the whole dataset and x2 data augmentation with rotation and horizontal flip, and we obtained an accuracy of 82.9% in the test set.

8 Training a CNN from scratch

VGG-16 has more than 100 million parameters (most of them located at the FC layers) and was trained with more than 1 million images from 1000 different classes. However, we only had a dataset with near 2000 images, which is far from the million images used to train the network, so it made sense to create new CNN architectures that were suitable for the training dataset (in terms of parameters) and that could be tailored to solve our scene-recognition problem. That is why we created and trained 2 new CNNs from scratch.

The first architecture, baNETas (Figure 11), consisted of 3 blocks of convolutional layer with 3x3 kernels + batch normalization before the Rectified Linear Unit (ReLU) activation + max-pooling layer of 2x2 pooling size and stride 1. After these 3 blocks, 2 FC layers of 2048 neurons were added, and at the end of the network a softmax layer with 8 output units was incorporated. It can be easily noticed that this network drew inspiration from VGG-16, as it was organized in blocks and it included 2 FC and a softmax layer. However, by having only 3 blocks, hence only 3 convolutional layers, we could reduce the number of parameters while maintaining the feature extraction power of convolutional layers. We also reduced drastically the number of parameters of the network by reducing the number of neurons in the FC layers from 4096 to 2048. Furthermore, we also added batch normalization layers after each convolutional layer but before the ReLU activation, which was motivated by slight increase of performance of the network.

The second architecture, abasNET (Figure 12), was a modification of the provided baseline network. We achieved a 80% reduction with respect to baNETas and almost a 90% reduction of the number of parameters with respect to the baseline network by increasing the pool size of max-pooling layers from 2x2 to 4x4, and by replacing the 4096 neurons FC layer by 2 FC layers with 512 neurons each. Despite this aggressive reduction of parameters, the network demonstrated to still have the capacity to perform well on the scene-recognition problem.

We tried adding Gaussian noise at the input, a dropout layer after each fully connected layer with a probability of 0.5, and a L2 norm regularizer for all the weights in both architectures.

BaNETas presented a clear overfitting, as it can be seen in the trends of the train and validation loss functions in Figure 13, when none of the aforementioned layers and regularization methods were used. We believe that the reason for this overfitting was caused, among other things, due to the huge number of trainable parameters, almost 38 millions. We tried then different combinations of Gaussian noise, dropout layer and L2 norm regularization, and we achieved 84.5% accuracy in validation when using L2 with $\lambda = 0.1$ and dropout $\rho = 0.5$. In spite of that, we considered changing the architecture to reduce the number of parameters and achieve a better accuracy, which motivated the creation of abasNET.

Regarding the abasNET model, we opted to use L2 regularization right from the beginning (using the same value as in baNETas, $\lambda = 0.1$), as it yielded better

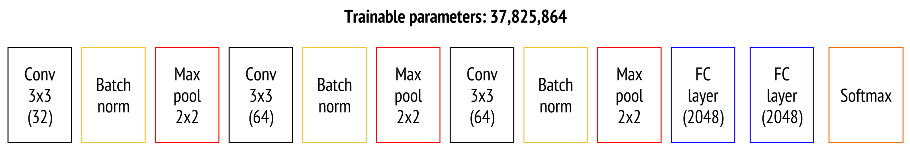


Fig. 11: baNETas architecture (first CNN model)

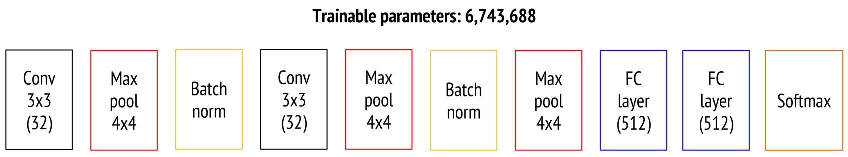


Fig. 12: abasNET architecture (second CNN model)

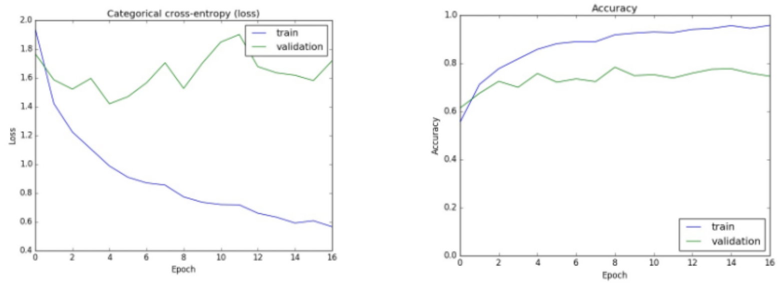


Fig. 13: Baseline baNETas loss and accuracy plots

performance in the previous experiments with baNETas, and that in conjunction with the reduced number of parameters of the network improved the accuracy by 1 point, that is, we obtained 85.5% accuracy in validation using the baseline abasNET model (see Figure 14).

We also tried combinations of Gaussian noise and dropout layers in this second model, but they did not provide any significant improvement whatsoever. Actually, the performance was slightly worse when considering any of these layers, which can be interpreted as a reduction of the capacity of the network.

In all cases, the training scheme was the following:

- **Image dimensions (width x height)**: 128 x 128
- **Training samples per epoch**: 10000
- **Validation samples per epoch**: Full validation set (200 samples)
- **Number of epochs**: 100
- **Model checkpoint**: Yes, only the model with the best accuracy in the validation set is saved
- **Early stopping**: Yes, training stops whenever the validation loss does not decrease after 10 epochs.
- **Degrees of freedom for data augmentation**: 10º rotation, horizontal flip, ±20% zoom.
- **Batch size**: 150
- **Optimizer**: Adam
- **Learning rate**: 0.0001

System	**Accuracy**
Dense SIFT + PCA + Fisher vectors + SVM	84.51%
Conv. Layer + PCA + Fisher vectors + SVM	**96.78%**
FC layer + SVM	94.92%
Fine tuned CNN (VGG-16)	82.90%
End-to-end trained CNN (abasNET)	88.72%

Table 6: Summary of the surveyed systems and their accuracy

9 Summary and discussion

We can find a major increase in performance when changing the hand-crafted features by learnt features. With a 12% increase, as we can see in Table 6, Deep Learning consolidates itself as the best candidate to extract features that best fit the training database. Using the last fully connected layer as feature vectors to train an SVM is just 2% accuracy behind. This decrease in performance could be due to the decreased number of features per image in this former system. When the features are extracted from the convolutional layers, we have 14x14x52 = 10192 features per image instead of the 4096 features per image we have in the fully connected layer.

By just fine-tuning VGG to adapt it to our reduced database we only obtain 82.9%. This drop in performance can be explained by a possible overfitting, due to the relation between the huge number of parameters in the VGG model and our dataset. This performance is increased to 88.72% using a new model (abasNET), with less parameters, so it can generalize better and adapt to new data in our tests.

When using Convolutional Layers, PCA, Fisher vectors and SVM the system achieves the best performance. In Figure 15 we present its confusion matrix and

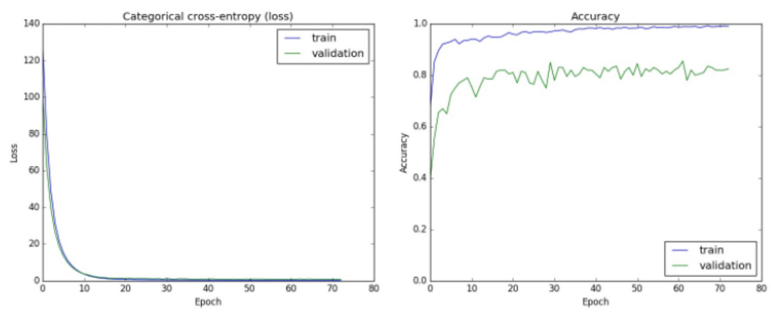
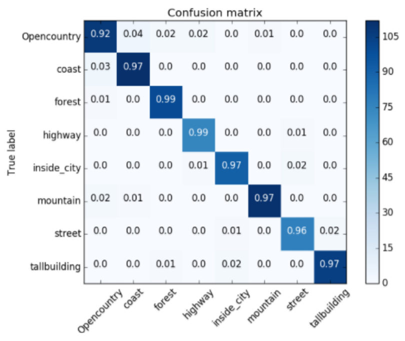
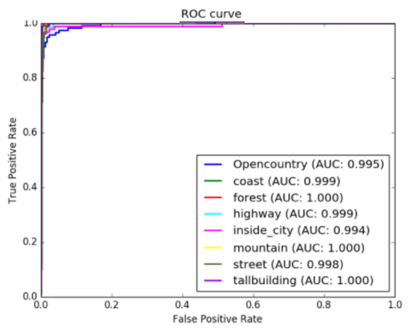


Fig. 14: Baseline abasNET with L2 regularization loss and accuracy plots



(a) Confusion matrix



(b) ROC curves

Fig. 15: Performance metrics of the best system: Conv. Layer + PCA + Fisher vectors + SVM

ROC curves. The most confused classes are Opencountry and Coast since they are quite similar in composition and colors.

10 Conclusions and future work

Using Deep Learning to learn the best features that can represent a dataset yields a better performance than using hand-crafted features. Although it is a conclusion firmly proved in this report, it is important to be aware of the complexity of Deep Learning models. The training process can be extensive and it can be difficult to find the best parameters without a fully exhaustive search, which proves impossible given the vast number of parameters and configurations available. A future study could be conducted to optimize more hyper-parameters in the abasNET architecture or even invent some new one to achieve better performance.

References

1. Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray, "Visual categorization with bags of keypoints," .
2. Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Computer vision and pattern recognition, 2006 IEEE computer society conference on*. IEEE, 2006, vol. 2, pp. 2169–2178.
3. F. Perronnin and C. Dance, "Fisher kernels on visual vocabularies for image categorization," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, June 2007, pp. 1–8.
4. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014.
5. Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.
6. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.
7. Aude Oliva and Antonio Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.
8. David G Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.