# Final Project: Image Classification

**Davide Belli**
11887532
davide.belli@student.uva.nl

**Gabriele Cesa**
11887524
gabriele.cesa@student.uva.nl

## Introduction

In these two projects, we are going to tackle the Image Classification problem using two radically different approaches. The first approach is Bag-of-Words based. This means that it aims to describe every image as a bag of visual items, or features, and then discriminate between classes based on the presence and count of those features for every image. There are various ways to define features in images, and our implementation is based on SIFT descriptors of points in the image, which are then matched with the closest element in a visual vocabulary. In the end, SVM classifiers are used to decide whether an image belongs or not to a certain class given its visual features. The second approach we are going to use is based on Convolutional Neural Networks (CNN). This approach is much more recent in literature, and aims to use learnable filters to find smaller (first) and larger (towards the last layers) patterns in the input image. Once the CNN is trained, its filters will hopefully identify the most relevant features, and once again an SVM or logistic regression will use those features to classify the input image into classes.

## 1 Bag-of-Words based Image Classification

In this first part of our approach, we are going to implement and experiment different ways to solve the image classification task under the Bag-of-Words approach. In particular, our dataset consists of 4 image categories (airplanes, motorbikes, faces, cars) with up to 500 training images and 50 test images each. The rationale behind BoW approach is to describe images as a composition (*bag*) of features (*visual words*). If we consider the larger level of granularity, for example, we can describe a face as being composed by two eyes, a nose and a mouth. In practice, we are using many more features per images, resulting in the capability to represent finer details image characteristic that do not appear so obvious at a first glance (e.g. some particular edge/shape in cars that can not be described as a particular component such as tires or windows, but may be still relevant to distinguish it from a face). By representing every image with this bag-of-(visual-)words, we can train a classifier to learn mapping the presence and quantity of such features to a particular category. Precisely, the classifiers we are going to use are Support Vector Machines. All the steps we just discussed, in reality, can be tackled and accomplished in different ways or using different models. In the successive sections, we are going to discuss which options we experimented with, reporting their effect in changing performances of the model and analyzing why they improve or worsen the results. Since SVM can only perform binary classification, we trained 4 different SVM to perform "one-vs-all" classifications. As a result, each image is classified with respect of each of the classes and its prediction is made of 4 binary labels. Given the image real labels and the binary assignment of images to every class, we can compute the *Accuracy* measure for each classifier. Each of the 4 SVM classifiers also returns a confidence level for every image to belong to that particular class. With this confidence levels computed for every test image, we can build a ranking for each of the four classes. *Average Precision* is the second measure we are going to use to evaluate our models. Finally, Accuracies and Average Precisions over the classes can be joint to represent the performances of the four classifier as parts of a whole model. *Mean Average Precision* and *Average Accuracy* are the result of this.

### 1.1 Implementation

#### 1.1.1 Feature Extraction and Description

The first task to be completed under the Bag-of-Words approach, is to define what will build up the visual-words Vocabulary. Indeed, the vocabulary must depend on the content of our dataset. For example, we don't care about having the feature descriptor of parts of the human body if we are classifying cars. In our case, we are going to extract features from the dataset using SIFT descriptors. We have already seen and discussed this method in the previous assignment. As a brief recap, the SIFT descriptor is a 128-dimensional vector representing the shape (in particular, direction of gradients) in the neighborhood of a point. This 128 dimensions are obtained by considering the intensity of gradients with respect to 8 orientation for every patch in a $4 \times 4$ matrix of patches around one point. The descriptor values are also normalized, clipped and rotated towards the main orientation (to make the descriptor invariant and avoid other numerical problems). There are two main ways to determine which points to consider when generating SIFT descriptors. The approach most commonly used consists in detecting interest points such as corners, blobs and other 2D shapes. A threshold can also be set to filter the (possibly large) number of feature points. Dense SIFT, on the other way, considers every point on the image independently by what it is represented in that region. To avoid an extremely large number of descriptors, a stride can be set to compute SIFT only every $n$ pixels horizontally and vertically. Moreover, Dense SIFT requires the size of the bins to be fixed. To compute this steps, we used MATLAB APIs from VLFeat [1].

SIFT and Dense SIFT, usually, are performed over a gray-scale version of the image. However, we notice that, as in every kind of image compression, converting 3 channels to 1 may result in losing or altering some information. For this reason, we implemented both SIFT versions to work over images in RGB format but also with representations in opponent color space and normalized rgb color space. In order to take the different channels into account we staked the 128-dimensional descriptors for a point computed on the 3 channels, leading to a $3 \cdot 128 = 384$ dimensional descriptors. However, a problem can rise: applying SIFT algorithm over different channels can yield different interest points in each of them, which makes not possible to stack descriptors for a point. In order to solve this issue, we decided to, first, detect interesting points on the gray-scale version of the image and, then, compute SIFT descriptors on the three channels only on those points.

#### 1.1.2 Building the Visual Vocabulary

After generating a set of many descriptors from a subset of the training images, we had to build a Vocabulary of descriptors in order to describe the images as bags of words. Obviously, we had to choose beforehand the number of terms in the Vocabulary. Too few visual words wouldn't allow a meaningful and rich enough description of images, while a Vocabulary with infinite size would make every image a set of unique features. For this step, we fixed a different number of vocabulary sizes and experimented with them to empirically discover which was the best parameter. After defining the size of our vocabulary as $K$, we used $K$-MEANS algorithm to cluster our set of descriptors and we took the centroids of the resulting clusters to generate the vocabulary terms.

#### 1.1.3 Encoding and Quantization

This step consisted in representing images as list of features with fixed size. At first, we had to convert the "bag-of-descriptor" representation to a Bag-of-Words. This is done by replacing every descriptor with the index of the closest descriptor in our $K$-sized dictionary. Now, one problem still remains: images may be represented by a different amount of indexes, depending on the number of feature points found by SIFT. To avoid this inconsistence, we convert the Bag-of-Words representation to an histogram of size $K$, containing the normalized frequency of every dictionary word in that image.

#### 1.1.4 Classification

Finally, we need a model to be able to use the features we computed with the goal of classifying images in different categories. For this task we are using a Support Vector Machine implementation provided in LIBLINEAR library [2]. In particular, we are going to create and train 4 different binary

[1] http://www.vlfeat.org/matlab/matlab.html

[2] https://www.csie.ntu.edu.tw/ cjlin/liblinear/

SVM classifiers (one per each class). To train the classifier, we take care of choosing different images from the ones we already used to build the dictionary terms. One important hyper-parameter to be tuned when using SVMs with soft margin is the Cost value for misclassified points. The implementation provided by LIBLINEAR includes a parameter to make it tune this parameter using Cross-Validation in order to find the optimal value. Also, different kernels can be applied to SVM to improve them. We did not add them to our experiments since LIBLINEAR implementation doesn't allow it and we found that linear SVM was enough to obtain good results in this task. Finally, rankings are created for every classifier as explained previously and measures are computed.

### 1.2 Results and Analysis

In this section we will discuss and interpret results obtained when experimenting with different configurations of hyper-parameters and features. For most of the experiments we fix our basic configuration and change only one variable. The default configuration uses SIFT for key points, GrayScale input images, a vocabulary size of 400, 150 images per class as SVM training sample (for a total of 600 training samples), and linear kernels for SVM. The values and configuration used for each hyper-parameter are:

- SIFT point extraction: [dense, key-point]
- SIFT color space = [grayscale, RGB, rgb, opponent]
- Vocabulary size ($K$) = [20, 50, 100, 200, 400, 800]
- Image samples to generate Vocabulary= [20, 50, 100, 200]
- Image samples to train SVM = [20, 50, 100, 150, 200]

#### 1.2.1 SIFT and Color spaces

Table 1: mAP for SIFT and DSIFT changing color spaces

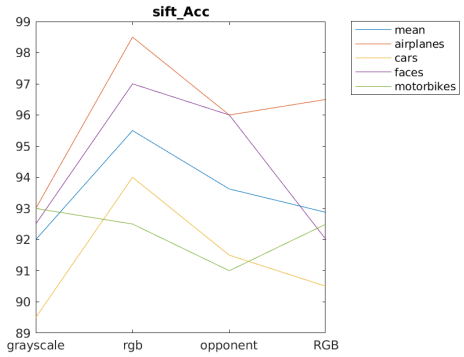
| | SIFT | DSIFT |
|---|---|---|
| grayscale | 0.935 | 0.935 |
| rgb | 0.964 | 0.909 |
| opponent | 0.941 | 0.956 |
| RGB | 0.941 | 0.931 |

Table 2: Mean Accuracies for SIFT and DSIFT changing color spaces

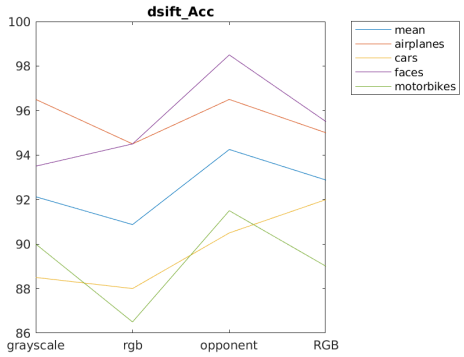
| | SIFT | DSIFT |
|---|---|---|
| grayscale | 0.920 | 0.921 |
| rgb | 0.955 | 0.909 |
| opponent | 0.936 | 0.942 |
| RGB | 0.929 | 0.929 |

The first tunable option we discussed is the choice between using SIFT for key-point sampling or DSIFT for dense sampling while varying the color space used (among Grayscale, RGB, rgb and opponent). In this first experiment we are trying all possible combinations to see if the performances of the two sampling approaches change in relation with the color space we are using. In Fig. 1 and 2 we show respectively Accuracy and Average Precision values obtained for each class and in average by every model. Numerical results for Mean Accuracy and Mean Average Precision can also be found in Tables 2 and 1.

In general, we see the same trends when considering Accuracy or Average Precision to evaluate the same model, thus most of the considerations we are going to make when analyzing one measure are valid also for the other one and vice versa. At a first glance, we notice that mean Accuracy and mAP values in both SIFT and DSIFT are in the same range, between 91% and 96%. Looking at the accuracy levels for different classes, however, we see how the two methods behave and perform differently. In particular, for SIFT using key points, best accuracies are obtained when classifying airplanes, while DSIFT yields better results for faces. An intuition behind this difference is that dense SIFT uses more background points that may be thought as irrelevant but that actually can bring some useful

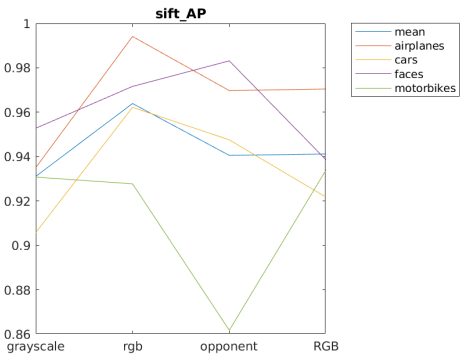



(a) Accuracy levels using key point SIFT

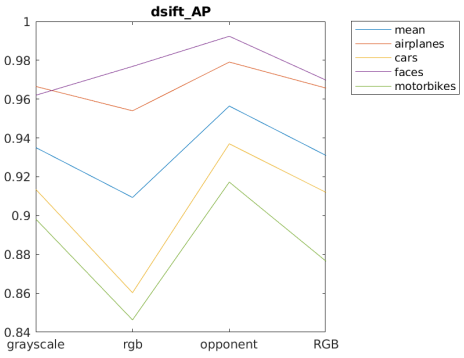



(b) Accuracy levels using dense SIFT sampling

Figure 1: Comparing Accuracy between SIFT and DSIFT




(a) Average Precision levels using key point SIFT




(b) Average Precision levels using dense SIFT sampling

Figure 2: Comparing Average Precision between SIFT and DSIFT

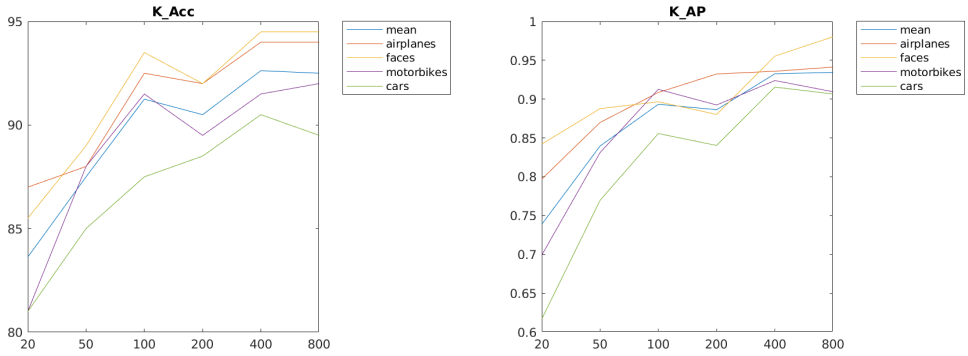
context information. Specifically, the background in face images very often depicts frames, doors and shelves with their characteristic neat horizontal and vertical edges, while images from other classes frequently present similar backgrounds. Landscapes and streets are common contexts for those three classes. In addition, motorbikes and airplanes are very often shown on an empty background (the sky for airplanes and white canvas for motorbikes): this may be an additionally misleading information for our classifiers. On the other hand, we can explain key points SIFT improvement in airplanes by considering that, in our dataset, they have much simpler shapes with less features than other classes. They are characterized by an elongated horizontally straight shape which allows a clear distinction at a first glance also for the human viewer. This intuition can be interpreted thinking at the feature representation of the object using the visual vocabulary: airplane images will probably be represented by many occurrences of few visual words (few words in the dictionary are very frequent, while most other words do not appear). As a consequence, we expect airplane images to be easily distinguishable from other classes which are composed by a more heterogeneous set of features, given the complexity of shapes in faces, motorbikes and cars. Finally, it is evident that motorbikes and cars classifiers are never as accurate as the ones for airplanes and faces. This is most likely because, frequently, those objects are pictured in noisy context appearing in other classes with many similar surrounding details, while also the object themselves are composed by many complex visual features. In the end, the classifiers may be tricked more easily by this similarly heterogeneous information in the feature representation.

Let's now consider the differences found when changing color spaces, again referring to the plots in Fig. 1 and Fig. 2. As expected, we find that Grayscale achieves low scores in comparison with other color spaces, especially when using SIFT. Indeed, Grayscale only brings one channel of information

while every other color space has 3 channels. The color space leading to the best results when using key-point SIFT sampling is rgb (normalized), while the best performer with dense SIFT is opponent color space. The intuition we had to explain this behavior is that some color spaces can better represent and "grasp" changes of gradients in feature points (rgb), while other are able to be more discriminative in a general context (opponent). Although there are differences (up to 3%) in the results achieved by different spaces when looking at mean scores, we see larger fluctuations (up to 8%) in performances if considering single classes (for example, consider motorbikes in Average Precision plots). This behavior is possibly caused by the fact that different color spaces are better than others in representing color in some kind of object. For example, we find that opponent color space in SIFT is not very effective to represent the features that compose motorbikes, while it improves performances in face classifiers. Indeed, in some objects tonalities change more smoothly (faces, with shades of pink) between neighbor patches than in other objects where changes are more abrupt (cars, motorbikes, with drastic color changes in different components of the vehicle).

Overall, we think that to gather more stable results and to draw complete conclusions we should consider a more complex scenario. In fact, having to discriminate between only 4 classes and considering that different instances of objects in the same class always have the same position, size and rotation, it is expected that even worse models achieve good results (90% Accuracy and AP or more).

### 1.2.2 Vocabulary Size




(a) Accuracy varying K

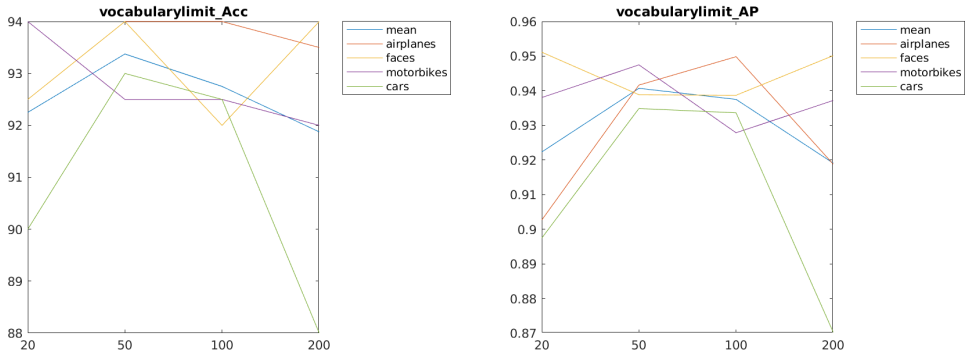
(b) Average Precision varying K

Figure 3: Improvement in Accuracy and AP when changing vocabulary size.

In the next experiment, we want to find out how the size of the vocabulary *K* influences the behavior of the classifiers. We expected to see an improvement in performances when increasing this value. Indeed, having a larger vocabulary allows us to represent a larger amount of features, thus increasing the discriminative power of our BOW representation. As show in Fig. 3, this idea is reflected in the behavior of our classifiers when varying this parameter. In details, we see a sharper increase in performances when going from $K = 20$ to $K = 50$, while the change is minor when considering higher vocabulary sizes. A further consideration analyzing the plots is that some classifiers tend to always be better or worse than other ones, but with larger gaps for lower vocabulary sizes. This may be the consequence of the fact that, if few clusters are used to build the vocabulary, features that are not that different will be clustered together. This results in some features that are very different from the other (in this case, especially features from faces), will be put in different clusters, while similar features (from example for cars and motorbikes) will be clustered together resulting indistinguishable from each other.

### 1.2.3 Number of Training Samples

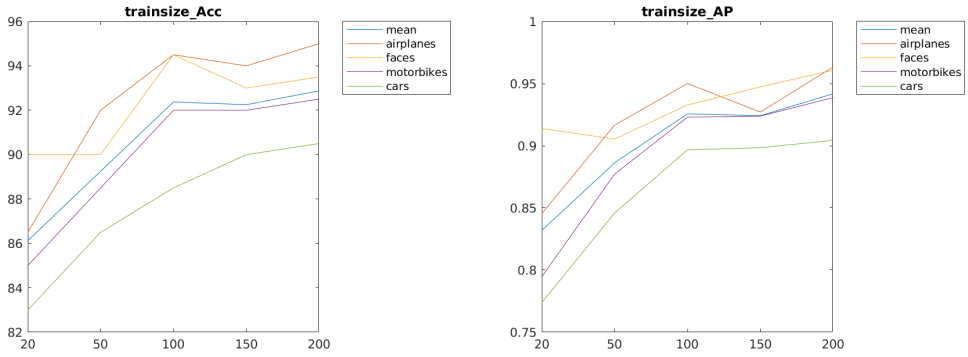
Finally, we consider what happens when we increase or decrease the amount of training data used to build the vocabulary or to train the SVM. In Fig. 4 we plot Accuracy and Average Precision obtained when changing the number of images used to learn the vocabulary. It is interesting to see that, against what we would expect, increasing the amount of images used after a certain point decreases




(a) Accuracy (b) Average Precision

Figure 4: Improvement in Accuracy and AP when changing the number of samples for vocabulary creation

performances. This behavior is confirmed by both measures and also considering the different classes separately.




(a) Accuracy (b) Average Precision

Figure 5: Improvement in Accuracy and AP when changing the number of samples for SVM training

Similarly, we experimented with changing the number of images used to train the SVM, as shown in Fig. 5 The behavior of the classifiers in this case are not surprising: just like with the vocabulary size, when increasing the number of images used to train the SVM, also the Accuracy and Average Precision improve. The rationale behind this is obvious. The more data we use to train our model, the more accurate the support vectors will be. It is also useful to remind that the method in LIBLINEAR library the we used to train our model includes iteration steps to find the optimal cost parameter.

### 1.3 Bonus Tasks

For the bonus part, we implemented 2 additional features: Spatial Information as additional features and Gaussian Mixture Model for clustering.

In order to preserve some spatial information, we split an image in a number cells (given the number of rows and columns) and compute an histogram for each of them. The feature vector describing an image will, then, be the concatenation of those histograms. The rationale is that the spatial position of the interesting points can be a very discriminative information: for instance, the relative position of eyes, mouth and nose is important to determine whether the content of an image is a face or not. After trying a few splits, we chose to split the images in 3 horizontal rows. Indeed, cars, motorcycles and planes usually are represented horizontally and centered, so their main points are in the central row. Conversely, faces usually span on the 3 cells, with hairs usually on the top one, and the chin on the bottom one. For these reasons, performances increases a lot with this trick.

The second feature we added is employing GMM for clustering. GMM can be interpreted as a generalization of *K-MEANS* that takes also into account the width (i.e. the variance) of the clusters (while KMEANS assumes all the clusters have the same size). We, then, used *Fisher encoding* to build the features of an image given its bag of descriptors.

We run some more experiments using this two new features. More precisely, we used SIFT and grayscale colorspace, 200 images per class to build the vocabulary and 150 per class for the SVM training set. In all experiments we set $K$ to 400 (the number of clusters for K-MEANS and the number of mixture components for GMM). Table 3 shows in each experiment the corresponding Mean Average Precision and the Average Precisions for all the classes. The 2 values in the 'Spatial Information' column specify the number of rows and the number of columns in which the image is split (therefore, $(1, 1)$ means no split).

Table 3: Average Precision on test set in the different experiments

| Clustering | Spatial Info | Mean (mAP) | Airplanes | Cars | Faces | Motorbikes |
|---|---|---|---|---|---|---|
| KMEANS | (1, 1) | 0.928577 | 0.933496 | 0.912410 | 0.943960 | 0.924441 |
| KMEANS | (3, 1) | 0.967360 | 0.950300 | 0.976409 | 0.973221 | 0.969510 |
| GMM | (1, 1) | 0.961356 | 0.958343 | 0.942597 | 0.983581 | 0.960904 |
| GMM | (3, 1) | 0.970414 | 0.979490 | 0.941039 | 0.991880 | 0.969246 |

The values in red are the best performances for their column. We point out that adding both Spatial Information and GMM leads to the best average performances (mAP). Moreover, this combination has also the best Average Precisions on almost all the classes (both on airplanes and faces, while on motorbikes it is the second best with a difference from the best smaller than $0.0003$).

### 1.4 Final Experiment

Finally, we tried to run an experiment with the best configuration, i.e. setting the parameters to the values that showed the best performances in the previous experiments: 50 images per class to create the vocabulary, 200 images per class for SVM training set (however, since we are using less images for the vocabulary, we could use an even larger training set here with up to 250 images per class), key-point SIFT, rgb colorspace, GMM clustering and a (3,1)-split for spatial information. However, this setting would require us to use a very large training set, using spatial information and rgb color space. Unfortunately, each of these settings makes the training set larger (either increasing the number of data points or increasing the number of features of a single data point) and, so, we have not been able to fit such a large amount of data in our RAM. Therefore, we have been able only to run a smaller experiment, not employing the spatial information and using a smaller number of images per class in the training set (only 150). With this experiment we reached a mean Average Precision of 0.979, the best one we have found.

## 2 Convolutional Neural Networks for Image Classification

### 2.1 Understanding the Network Architecture

The structure of the pre-trained network is: $input \rightarrow conv \rightarrow mpool \rightarrow relu \rightarrow conv \rightarrow relu \rightarrow apool \rightarrow conv \rightarrow relu \rightarrow apool \rightarrow fc \rightarrow relu \rightarrow fc \rightarrow softmax$.

The repeated pattern in the architecture is the $conv \rightarrow relu \rightarrow pool$. In the first block it appears as $conv \rightarrow mpool \rightarrow relu$ but, since *max pooling* is used, the result is equivalent to $conv \rightarrow relu \rightarrow mpool$, though it is cheaper to compute (as the pooling is done earlier and, so, the size of the tensor on which *relu* is applied is smaller). In the end, 2 fully-connected layers map the features computed to the output classes.

The first of the 2 fully connected layers is the block with most parameters. Indeed, it connects all the neurons in the feature maps of the previous layer (64 $4 \times 4$ feature maps) to all the 64 neurons in the following one, with $4 \times 4 \times 64 \times 64 = 65536$ connections plus 64 biases. Accordingly, the memory required to store its parameters is 256KB (since parameters are encoded with *single precision*, i.e. with 4B).

### 2.2 Setting up the Hyperparameters

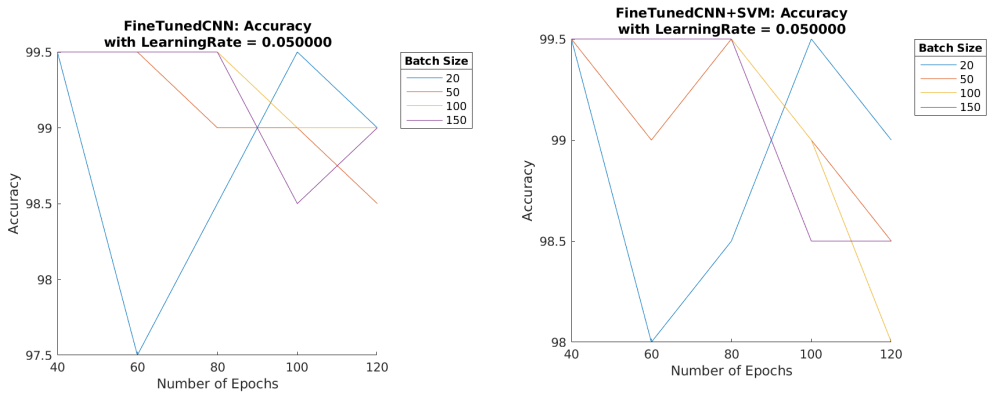
To perform hyper-parameters tuning we used Grid-Search: given a list of potential values for each hyper-parameter, we ran an experiment for each combination of assignments of those values. The values used for each hyper-parameter are:

- Number of Epochs: [40, 60, 80, 100, 120, 140]
- Batch Size = [20, 50, 100, 150]
- Learning Rate = [0.05, 0.02, 0.01]

Afterwards, we choose as best combination of hyper-parameters the one that has led to be highest accuracy. However, many combinations led to an accuracy of 99.5% (i.e. 199/200 images classified correctly) but none reached 100%. For this reason, we chose to take the simplest combination among the best ones, i.e. with smaller number of epochs and batch size. Therefore, our final setting is:

- Number of Epochs: 40
- Batch Size = 20
- Learning Rate = 0.05

Figures 6 and 7 show some of the accuracies we found. In these plots, we fixed once the learning rate and then the batch size and we only show how performances change varying the 2 parameters not fixed. The fixed parameters have been set to their values in the best hyper-parameters configuration.




(a) Using the Fine-Tuned CNN

(b) Using the "SVM" trained over the Fine-Tuned CNN features

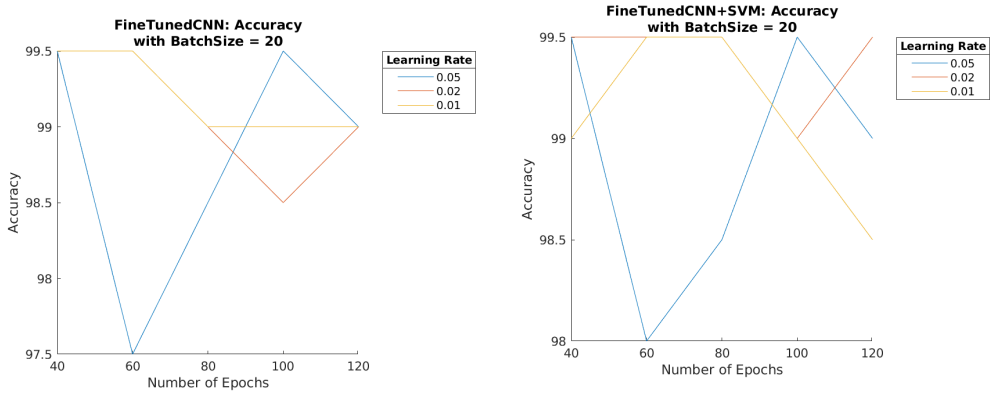
Figure 6: Accuracies using different Number of Epochs and different Batch Sizes

However, due to the small size of the test set (only 200 images) and the small number of experiments, it is hard to recognize any significant trend in these plots. In general we see that performance tends to increase with the number of epochs until a point from which the model starts to overfit. This point tends to be higher using smaller learning rates or batch sizes but the trends are very noisy and the differences in accuracy are very small (usually just 1 or 2 test images).

### 2.3 Feature Space Visualization

In order to visualize and compare the feature spaces of the pretrained and the finetuned networks, we took the activation values of the second last layer of the networks (the first fully connected layer) for each of the images. Afterwards, we ran TSNE over the 2 resulting datasets to "project" them in a 2-dimensional space and plot their distribution. Figure 8 shows the results we obtained: Figure 8a shows the distribution of the features from the pre-trained model while Figure 8b shows the ones from the fine-tuned model. The colors of the dots in the plots represent the classes the corresponding datapoints belong to.

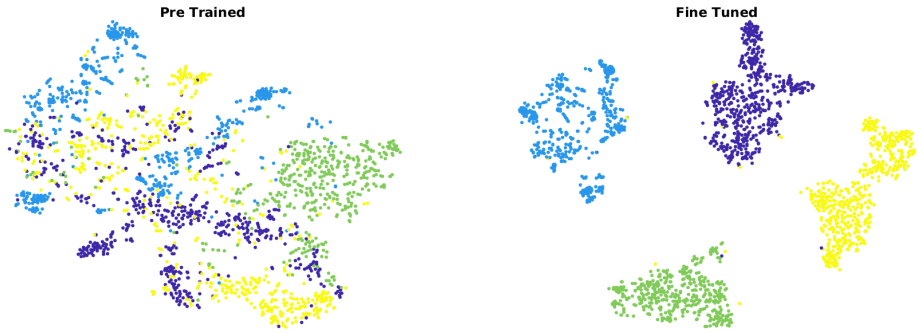
The difference between the 2 plots is remarkable: the features from the fine-tuned are almost perfectly linearly separable in the projected 2-dimensional space (except for a few outliers). Conversely, using




(a) Using the Fine-Tuned CNN

(b) Using the "SVM" trained over the Fine-Tuned CNN features

Figure 7: Accuracies using different Number of Epochs and different Learning rates




(a) Features from Pre-Trained CNN

(b) Features from Fine-Tuned CNN

Figure 8: Visualization of features using TSNE

features from the pre-trained model, different classes are partially overlapping and tend to be less clustered.

The reason is that the pre-trained model has not been trained for our task. Nevertheless, notice that a feature space for a larger number of classes that makes them separable should still be good for a smaller number of classes: therefore, one can expect the pre-trained feature space to work pretty well even with our few classes. However, since our task is way easier than the original one of the pre-trained, building a feature space where classes are separable is easier. Indeed, the pre-trained feature space contains information useful to recognize many other classes and it has been built to facilitate the separation of its original set of classes. Fine-tuning the model to recognize only our smaller set of classes enable the network to better exploit this feature space to separate only our classes (removing useless information for the other ones). As a result, the fine-tuned model produces better results.

### 2.4 Evaluating Accuracy

In each of our experiments, three types of models were built and evaluated:

- *Fine-Tuned CNN*: the fine-tuned model at the end of the training process
- *SVM trained over the fine-tuned features*: we took the features computed by the fine-tuned model and fed them to an SVM for classification
- *SVM trained over the pre-trained features*: we took the features computed by the pre-trained model and fed them to an SVM for classification

Indeed, the last layer of a neural network can be seen as a *logistic regression* (or *softmax regression* in case of multiple classes) over features computed by the rest of the network. It can make sense to replace this last layer with a stronger machine learning model as an SVM to boost performances. However, the SVM implementation we used is *LIBLINEAR* which implements only *linear* models. Moreover, SVMs don't support classification with more than 2 classes, therefore, *LIBLINEAR* actually implements *softmax regression* in that case. Unfortunately, this is also our case: indeed, the code provided to us used the '-s 0' option which, according to *LIBLINEAR* documentation[3], corresponds to 'L2-regularized logistic regression'. As a result, the CNN+SVM models we used are actually normal deep neural networks with a fully connected layer on the top. Therefore, training the "SVM" is equivalent to "forgetting" the last fully connected layer, freezing all the previous layers and train the network again.

Using the parameters provided in the assignment, we got the following accuracies:

- *Fine-Tuned CNN*: 99.5 %
- *Fine-Tuned + SVM*: 98.5 %
- *Pre-Trained + SVM*: 89.5 %

The best result is obtained with the Fine-Tuned CNN. Indeed, we can expect the *Pre-Trained+SVM* model to be the worse as the it tries to linearly separate the classes in the feature space of the pre-trained model. However, as we have seen in Fig 8a, classes in this feature space are overlapping and hardly separable. Conversely, both *Fine-Tuned + SVM* and *Fine-Tuned CNN* produces good results. It might surprise that not using the "SVM" leads to a better accuracy; however, as explained before, the 2 models are very similar and the only difference is in the training procedure. It is possible that the "SVM" training slightly overfitted or underfitted, as the features were tuned on the former fully connected layer.

Finally, setting the hyper-parameters to the best combination found during tuning, we got:

- *Fine-Tuned CNN*: 99.5 %
- *Fine-Tuned + SVM*: 99.5 %

Even tuning the hyper-parameters we haven't been able to classify correctly all 200 images and the performances of the fine-tuned CNN remain the same. However, we have been able to increase the performances of the hybrid model to the one of the CNN (199/200).

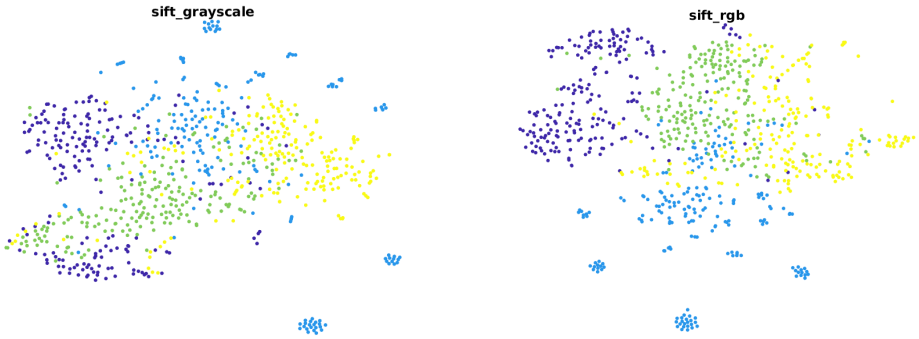
### 2.4.1 Comparison with Bag-of-Words Approach

In our experiments, the Neural Network approach has outperformed the Bag-of-Words one. In most of the experiments with Neural Network, accuracies ranged between 98% and 99.5% (which means only one image in the test set was misclassified). Bag-of-Words method has still produced some good results, but never as good as Neural Network's ones (*average accuracy* has never exceeded 96%). Moreover, both building and training a good Bag-of-Words have been more time consuming. Fine-tuning a pre-trained CNN on our small dataset has been rather fast, but running clustering algorithms as K-Means or GMM on high dimensional data or computing DSIFT descriptors for many images can take really long.

For comparison with features computed in the Bag-of-Words model, in Fig. 9 we plot the TSNE representation of features that are taken as input in the SVM for the experiments using SIFT with grayscale (default) and rgb color space (which is one of the best settings we found in this approach). It is evident as in the first case, different classes are more mixed together in the central part, or split in different parts of the graph (dark blue and light blue dots). On the other hand, in the second plot with rgb color space, different classes are more clustered and distinguishable, although they are not even close to being linearly separable as the features computed by the fine tuned CNN.

We can also compare the feature spaces built using the fine-tuned CNN and the ones computed from the Bag-of-Words features. Again, in Bag-of-Words feature spaces projected on the 2-dimensional space, classes overlap and, sometimes, tend to spread more uniformly across the space.

[3] https://www.csie.ntu.edu.tw/c̃jlin/liblinear/#download




(a) Features from SIFT + grayscale

(b) Features from SIFT + rgb color space

Figure 9: Visualization of features using TSNE

### 2.5 Bonus Tasks

#### 2.5.1 Data Augmentation

For the bonus part we implemented *Data Augmentation* and we tried other pre-trained models.

For *Data Augmentation*, we enriched our training set with each of the images in the training set rotated of $45°$, $90°$, $135°$, $180°$, $225°$, $270°$ and $315°$. We run an experiment using the best hyper-parameters found in 2.2 with the augmented training set. The resulting accuracies were:

- *Fine-Tuned CNN*: 99.0 %
- *Fine-Tuned + SVM*: 98.5 %
- *Pre-Trained + SVM*: 70.0 %

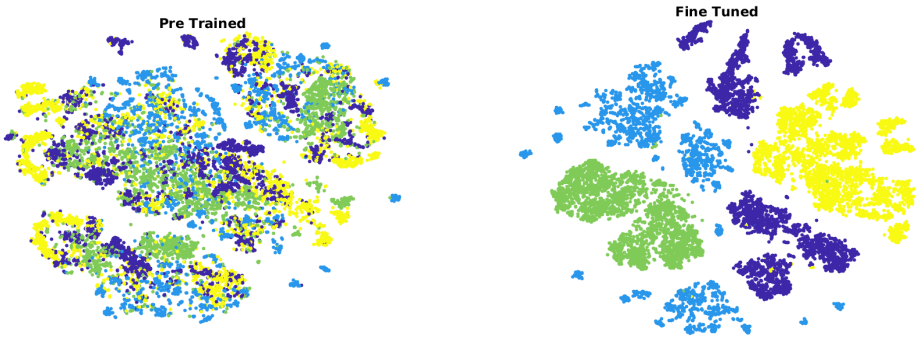
It is evident that performances got worse. For the pre-trained CNN + "SVM", this performance deterioration can be explained considering that the pre-trained model has not been trained on rotated images, therefore the features it produces are not very meaningful for the rotated images. Original images still have the same features but the dataset now contains many noisy data-points (the ones corresponding to the rotated images) which negatively affect the "SVM" training (since the test set does not contain rotated images). We can also see a (very) small decrease in performances for the fine-tuned model. Thought the difference is just a misclassified test image, we suppose the reason is that the data set does not contain, for instance, faces rotated at any angle. While it is possible that airplanes are pictured at more different angles, the dataset does not contain $180°$ rotated objects. We can, therefore, explain these results as an excessive data augmentation. In general, however, data augmentation may drastically improve results of a model, especially on smaller datasets. We have already experimented with employing this technique for other purposes using different datasets. If objects are depicted in different orientation, position, scaling, it could be useful to include more occurrences of those images with some image transformation, so that the model learns to recognize different classes in different conditions.

For the sake of completeness, we show the feature visualization of the augmented dataset using both the pre-trained and the fine-tuned networks in Fig 10.

As expected, features from the pre-trained model overlap a lot and make "SVM"'s task harder. Conversely, features produced by the fine-tuned model seem to be more clustered, though classes are not linearly separable (in the 2-dimensional projected space) anymore (with respect to the one visualized in Fig 8).

#### 2.5.2 Going Deeper

We also experimented with deeper networks as pre-trained models. Precisely, we used VGG Net and AlexNet. These networks are way larger than the one provided in the assignment and were much




(a) Features from Pre-Trained CNN

(b) Features from Fine-Tuned CNN

Figure 10: Visualization of features using TSNE on augmented dataset

Table 4: Comparing Accuracy in Deeper Networks

| | LeNet | AlexNet | VGGNet |
|---|---|---|---|
| {Fine-Tuned CNN} | 99.5 | 25.0 | 25.0 |
| {Pre-Trained + SVM} | 89.5 | 100.0 | 100.0 |
| {Fine-Tuned + SVM} | 98.5 | 25.0 | 25.0 |

slower to train. For this reason, we have not been able to tune the hyper-parameters for them and we had to use a small batch size (of 20) to make data fits in memory during training.

In table 4 we show the Accuracy obtained by employing deeper networks like AlexNet and VGGNet with respect to the original LeNet architecture. Both networks are already trained for classification on the ImageNet dataset. Indeed, they achieve perfect performances of *100%* just out-of-the-box, without any fine-tuning. Paradoxically, when running those networks on our dataset to fine-tune them to our task, Accuracy levels drastically decrease. We suppose that the hyper-parameter setting we use was far from optimal (maybe the learning rate was too large and led to underfitting or the number of epochs was not enough). Because of the very long time (more than 2 days, due to our hardware limitations) that takes to run training with these larger nets, we didn't have time to tune hyper-parameters in order to obtain convergence in our model. Nevertheless, the wider potential of those networks is evident considering how they are able to nail every prediction even when trained on a different dataset.

## Conclusion

In these final projects, we have included the knowledge we gained in the Computer Vision 1 course to create two working systems able to perform Image Classification. After implementing both Bag-of-Words and CNN models as described in this report, we spent some time performing grid search on their parameters with the purpose of understanding how and why different configurations affect performances. In the end, it was evident that newer models like Convolutional Neural Networks are much more powerful than Bag-of-Words. In particular, we have seen as transfer learning can be used to have a better starting point in the training instead of initializing a neural network from scratch. In general, with this approach, state-of-the-art models such VGGNet and AlexNet can be employed to boost performances, though, in our case, we have not been able to observe significant improvements due to the simple nature of the problem. Although Deep Learning methods are widely used nowadays, we believe it was very informative to build a Bag-of-Words approach from scratch to experiment with low-level concepts and details in order to better grasp the intuitions behind Image Classification tasks.