Final Report CV1 - Part 1

Macha Meijer (13136011)
Karim Abdel Sadek (15079015)
Kristiyan Hristov (12899437)

October 2023

1 Part 1 - Bag-of-Words image classification

For the first part, the goal was to implement image classification using Bag-of-Words (BoW). This implementation consists of five parts: extracting SIFT descriptors, building a visual vocabulary, encoding the features using the vocabulary, representing the images by frequencies of visual words, and classifying the data using a support vector machine (SVM). After implementing the BoW image classification, an evaluation of the results was done.

The classification was done using the CIFAR10 dataset. From this dataset, the classes airplane, bird, ship, horse, and automobile were used for the implementation. The train and test split defined by CIFAR were used.

1.1 Extracting SIFT descriptors

For building the BoW image classifier, the first step was to extract SIFT descriptors from the image dataset. This was done by using the OpenCV sift module. After extracting the keypoint descriptors, the keypoints were visualized. The result of this is shown in figure 1. One can see that for all images in the figure multiple keypoints were found, on various scales and in various directions.

However, in 17 of the 5000 training images, no descriptors were found. A visualization of a few of these images can be found in figure 2. It is likely that no keypoints were detected because the main object is too small or because there is not enough contrast between the main object and the background. Since no features were detected, it was decided to remove the images from the dataset.

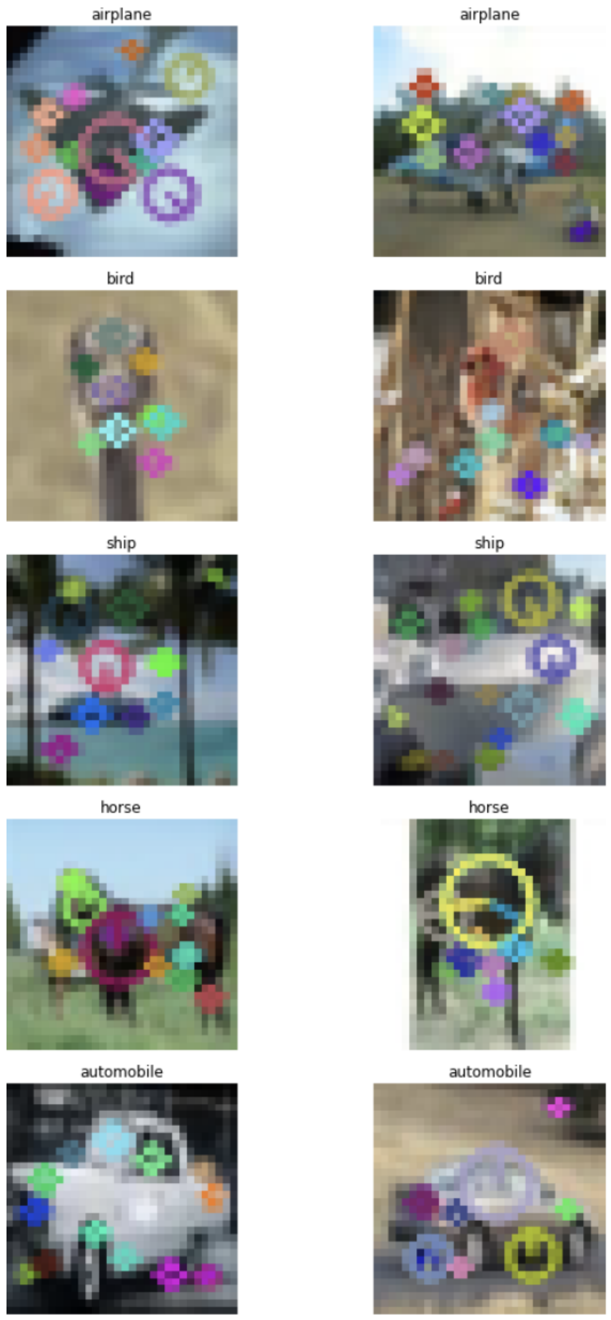

Figure 1: Visualization of keypoint descriptors for all five classes

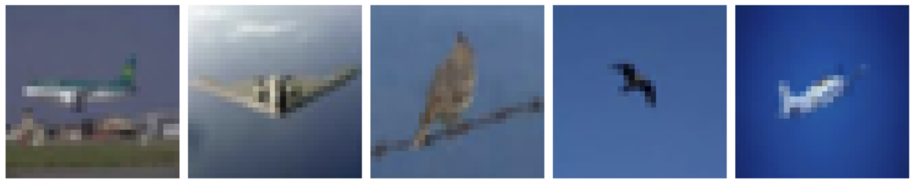

Figure 2: Visualization of images where no keypoints were detected

1.2 Building a visual vocabulary

After the descriptors were extracted, a visual vocabulary was built. This vocabulary was created by clustering the high-dimensional descriptors of a subset of the dataset into 1000 clusters. Every cluster represents one visual word. To cluster the data, the Scikit-Learn K-means function was used. A 30%, 40%, and 50% subset size was used. The results of the clustering are shown in figure 3. For the different subset percentages, the results are quite different. As one would expect, with a 0.5 subset, the clusters are less occluded than with a 0.3 subset.

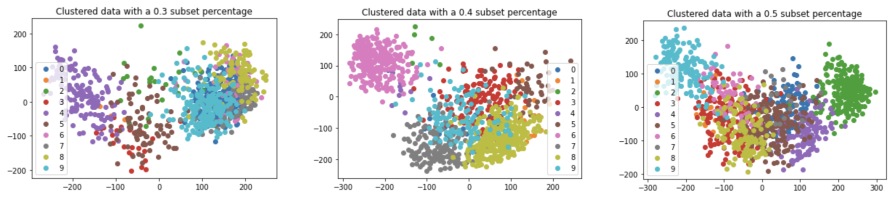

Figure 3: 10 out of 1000 clusters visualized when clusters are defined on respectively 30%, 40%, and 50% of the data.

1.3 Encoding image features using the visual vocabulary

After the visual vocabulary was defined, all image keypoint descriptors were encoded in terms of this vocabulary. This was done by using the clustering result to predict for each keypoint descriptor to which cluster it belongs. This resulted in a list with clusters for each image, where the length of the list was equal to the number of keypoint descriptors of that image.

1.4 Representing the images by frequencies of visual words

As input data for the final training, visual words frequencies were used. This means that, for training, every image was encoded in a histogram, which represented the frequency of each visual word in that specific image. To visualize the differences in features per class, a histogram for the frequencies of visual words per class was made. The histograms can be found in figure 4. There are a few differences visible between the classes. For example, for the classes automobile, horse, and bird, the first few visual words have a high frequency, in contrast to the classes airplane and ship. Also, the class airplane has a few very high frequencies around word 200, while this isn't the case for all other classes.

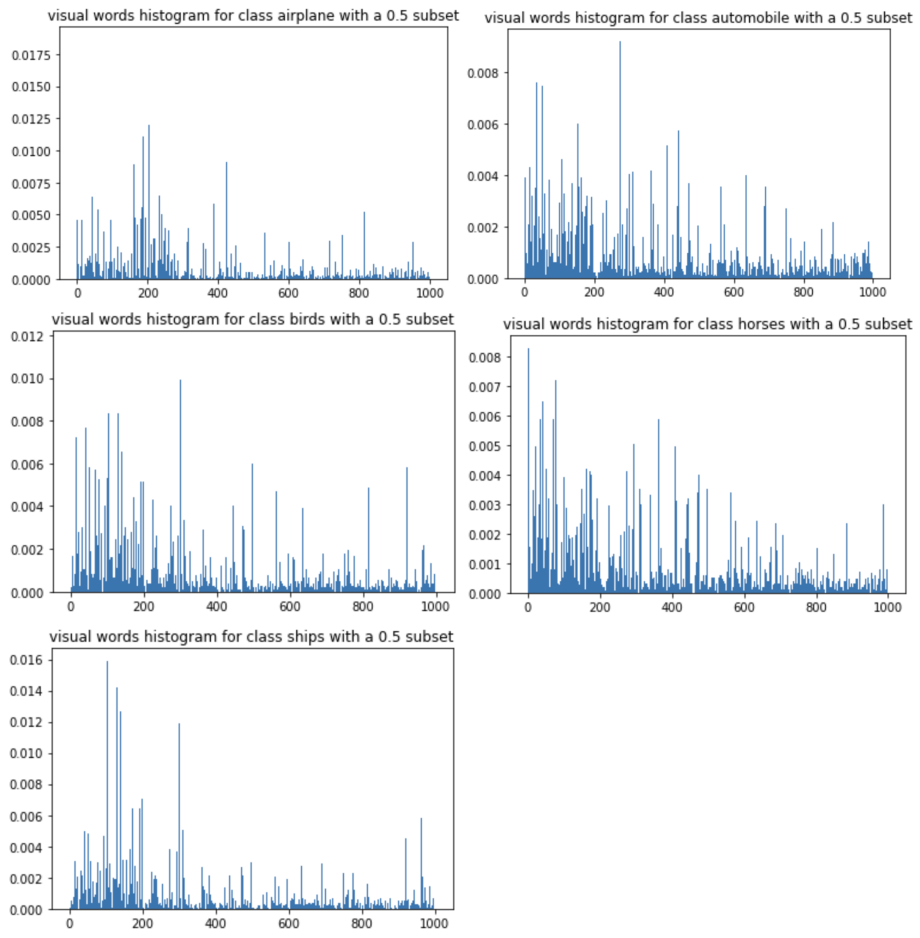

Figure 4: Histograms with frequencies of each visual word for the classes airplane, automobile, bird, horse, and ship

1.5 Classification

After the histograms were created, the feature extraction was done. To actually classify the images based on the features extracted, support vector machines (SVMs) were used. For every class, a binary classifier was trained. This means that to classify the images, 5 classifiers were trained, each specialized in predicting one class.

Every classifier was trained on 200 positive images and 400 negative images. This means that 200 images for the class the classifier was designed to predict were included and 100 per class the classifier had to predict as 0 were included. This was a modeling choice of ours to make sure that the class imbalance was not too big, which would be the case if an equal amount of images of every class were taken. To ensure that different images would be used for training and for determining the vocabulary with k-means clustering, the images were randomly selected from the second half of the dataset, while the images used for clustering all were in the first half of the dataset.

1.6 Results

To evaluate the result, the MAP metric was computed for each class individually and an average was taken over all classes. To find the best settings for the classification, different experiments were done. All results reported are obtained from testing on the test dataset. Since k-means clustering is used, and this algorithm gives different solutions every time, the results may differ a bit per run. A seed could be fixed to alleviate this problem.

1.6.1 Different subset sizes for clustering

The first step in obtaining the optimal settings was experimenting with the subset size used for determining the visual words via k-means clustering, as described in section 1.2. A 30%, 40%, and 50% subset were used, all in combination with a 1000-word vocabulary and SIFT features. The results are stated in table 1. In general, a 40% subset gives the best results, though the differences are very small. When looking at the class scores, one can see that in general the horse class has the highest MAP and the bird class has the lowest MAP. This could be due to the fact that birds are smaller, which can cause more background to appear on the image, which causes fewer bird features to be found by SIFT.

Class	30% subset MAP	40% subset MAP	50% subset MAP
Airplane	**64.8**	60.7	60.6
Bird	11.6	**13.4**	11.0
Ship	41.5	38.8	**45.8**
Horse	**67.3**	**67.3**	69.2
Automobile	40.0	**49.4**	35.6
Average	45.0	**45.9**	44.4

Table 1: MAP scores for different subset sizes in %. The best result per class is in bold.

1.6.2 Different vocabulary sizes

After the best subset size was determined, an experiment was conducted to decide on the best vocabulary size, which also means the best amount of clusters in which the data is clustered. Three different vocabulary sizes were tested, namely 500, 1000, and 1500 words. The vocabulary sizes were all combined with a 40% subset size and SIFT features. The results are displayed in table 2. A vocabulary size of 500 words gives the best result for each class individually and for the average over the classes. A possible explanation for this is that a larger vocabulary size causes the features to be too image-specific, which can make it harder to generalize the features per class. In general, the classifier with

Class	500 words	1000 words	1500 words
Airplane	61.8	60.0	predictions, true
Bird	**14.3**	13.4	8.5
Ship	**41.8**	41.4	3.1
Horse	**69.0**	67.4	67.4
Automobile	**51.6**	44.2	30.0
Average	**47.7**	45.3	39.6

Table 2: MAP scores for different vocabulary sizes in %. The best result per class is in bold.

500 words and a 40% subset performs the best. Therefore, this classifier was used to be compared for the experiments described in the following sections.

1.6.3 Features extraction and classification based on HOG descriptor

We start this section by reporting the results obtained based on HOG descriptors. In contrast to the sections before, where we experimented with different methods to obtain features based on SIFT, in this section we obtained the image features via HOG, which means that per image, a histogram of oriented gradients was used for training. We select 200 images for our 'positive' class for each classifier. On the other hand, we selected 100 images for each of the remaining classes. This is to keep coherence with the previous section. We investigated 3 different settings based on HoG, which included varying the arguments of our HOG extractor function. The settings are the following:

- Setting 1: orientations=9, pixels-per-cell=(8, 8), cells-per block=(3, 3), transform-sqrt=True
- Setting 2: orientations=7, pixels-per-cell=(10, 10), cells-per block=(2, 2), transform-sqrt=True, block-norm="L2-Hys"
- Setting 2: orientations=8, pixels-per-cell=(10, 10), cells-per block=(2, 2), transform-sqrt=True, block-norm="L2-Hys"

In Table **??**, the results for every setting are presented.

Class	Setting 1	Setting 2	Setting 3
Airplane	39.4	35.5	**44.3**
Bird	44.8	**49.5**	36.9
Ship	44.3	**54.4**	46.2
Horse	**69.2**	61.8	60.1
Automobile	**58.5**	49.8	57.4
Average	**51.1**	50.1	48.9

Table 3: MAP scores for different feature extraction with HOG. The best result per class is in bold.

We discuss three of the arguments we modified from the default setting. This will highlight why we decided to modify some of the parameters and why.

- **Orientation Bins** We tried to reduce the number of orientation bins. The idea is that sometimes fewer bins can capture the essential features without focusing too much on finer details that could lead to overfitting. Increasing it, however, helps to identify finer details.
- **Cell Size** Increasing the cell size can be beneficial in cases where the dataset varies significantly, and the focus needs to be on broader distinguishing characteristics rather than finer details. It then helps in capturing more general features. Diminishing it allows us to extract more local features
- **Block Size** Reducing the block size also helps us to detect more localized features. Increasing it allows us to capture more general features.

To conclude, the best setting between the SIFT setting and the HOG setting is the latter. Precisely, what we defined as Setting 1 for HOG give the best MAP score.

1.7 Ranked plots

We now plot the top and bottom 5 ranked test images for each classifier. The plot is showed in figure 5.

We can observe and make a consideration on the results. Our classifier for horses actually predicts a higher score for images of horses, since they are at the top of the list. The classifier for cars assigns and higher score to cars, and so on. In general, for all the classifiers, we can see that they give a high-score to images from the class that we used as the positive class for the training. Obviously, some small imperfections are present, due to the noisy nature of the images (a bird could look like a plane from a specific angle). In general, Figure 5 is coherent with what we expected and it means that our training process proceeded smoothly and correctly!

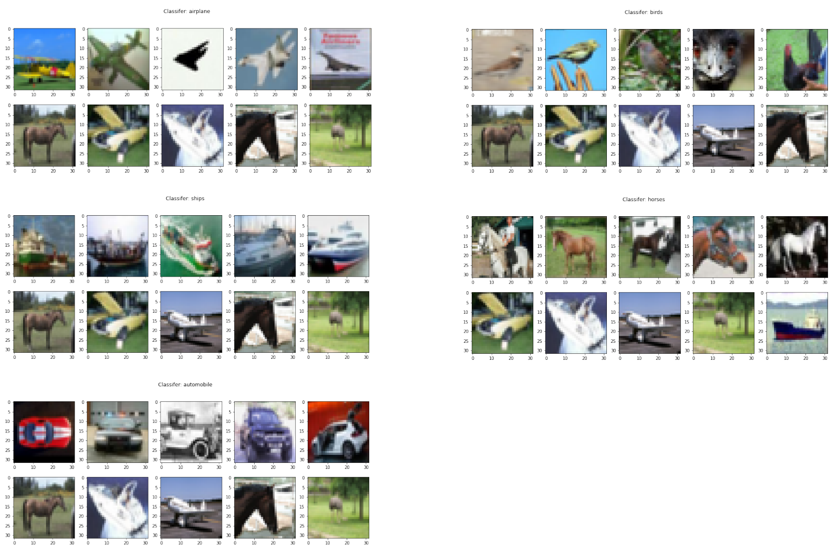

Figure 5: Top-5 and the bottom-5 ranked test images for the classifiers for airplane, automobile, bird, horse, and ship

1.8 Hyperparameter Tuning

We now conclude the report for part 1 by performing hyperparameter tuning. We will do it by tuning the most relevant parameters for a Support Vector Machine. Here is a description of the parameters we decided to tune.

- Kernel: The kernel function determines the mapping of data into a higher-dimensional space. We could use linear, radial basis, and polynomial kernels. This should be tuned depending on the type of data that we have. For example, we expect a linear kernel to not behave well on our dataset!
- Regularization Parameter (C): The regularization parameter, controls the trade-off between maximizing the margin between classes and minimizing the classification error on the training data. Theoretically, with a $C \to \infty$ we would classify each point correctly, i.e. an SVM with hard margins. However, this would lead to overfitting. Employing a smaller C, we permit our classifier to misclassify some points during training, allowing for a better generalization. On the counter side, if C is too low we may incur into underfitting.
- Class Weights: For imbalanced datasets, assigning different weights to classes can help the model focus on correctly classifying the minority class. This can be done by setting the argument class_weight to 'balanced'.

We performed hyperparameter tuning by the use of GridSearch. In this way, we automize the run of our program and we get to try all the combinations of parameters. Our suitable choices for the parameters are the following:

- Kernel: ['linear', 'rbf', 'poly']
- Regularization Parameter (C): 'C': [0.1, 0.8 1, 2, 10]
- Class Weights: Balanced or None
- gamma: If our kernel is 'rbf', we experiment with either 'scale' or 'auto'. This parameter is not tunable for the other types of kernels.

We report in table 4 the best result we obtained with the best set of parameters:

Class	Best setting
Kernel and Gamma	**RBF, 'scale'**
Regularization C	**2**
Class Weights	**Balanced**
Airplane	**67.7**
Bird	**83.9**
Ship	**79.1**
Horse	**86.3**
Automobile	**83.6**
Average	**80.1**

Table 4: Best Parameters MAP scores after hyperparameter tuning using feature extraction with HOG.

We see that via hyperparameter tuning we managed to drastically increase our results. Setting the class weights to 'balanced' helped the most in reaching a high MAP score.