CV1 Final Project Part1

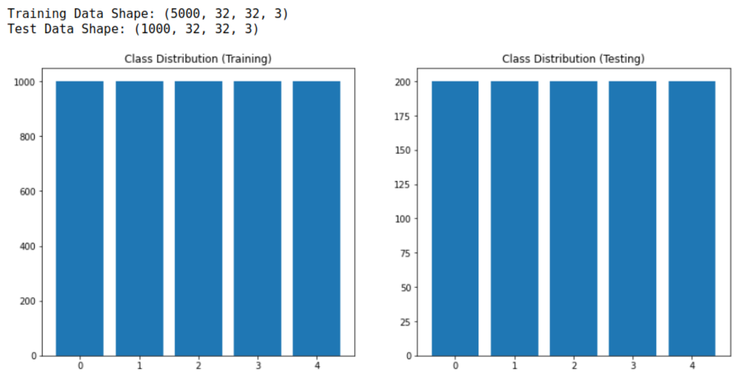
Luis Zerkowski, Tanya Kaintura, Jelle van der Lee

Introduction

This is the report on the first part of the final project of Computer Vision 1. In this paper, we carefully describe how each section was implemented and show all the intermediate results we got. After going through each segment, we conclude by evaluating and comparing the models we developed, pointing out their differences and discussing the residual flaws of the project. Even though this should be a self-contained paper, we also provide the implementations on a `Jupyter Notebook` as required to make sure the results are reproducible.

Data Check

Before working on the image classification pipeline, we have to make sure we understand the data we are working with. Thus we start by plotting checking the data shape and plotting the class distribution for both training and test data as shown on figure 1.



```
Training Data Shape: (5000, 32, 32, 3)
Test Data Shape: (1000, 32, 32, 3)
```

Figure 1: Data shape and class distribution

2.1 Feature Extraction and Description

SIFT

Our task for **2.1** is to extract the SIFT descriptors from our training images. This is the first out of five steps for creating a bag-of-words based image classification system. We approach this task by using the SIFT algorithm provided by `OpenCV` to extract descriptors on all grayscale versions of the training images. It's important to perform this color map conversion to the images because the SIFT algorithm works better on grayscale. Finally, for two images on each class, we also compute the key-points to be able to display these images with their blob circles. The final plot of our results is shown in figure 2 on the first and the second columns, where we display two separate images for each of the five classes.

HoG

For HoG we use a very similar implementation, computing the descriptors for each grayscale version of the training images. The main difference on implementing this technique is the hyperparameter optimization. To ensure good descriptors and nice visualizations we had to explore some combinations to finally get to a final setup with 8 gradient orientations, 2×2 pixels per cell and 1×1 cells per block. The results are shown in the last column of figure 2.

2.2 Building Visual Vocabulary

Implementation Approach

In **2.2** we want to create a visual vocabulary by using K-Means clustering, where each cluster center is a visual word. As required, we fix the number of clusters (i.e. vocabulary size) to 1000 and then start by choosing the amount of images we are going to use to compute the clusters (30%, 40% or 50%) and randomly sample them from the training data. To ensure we have balanced data, even if the number of descriptors is not exactly the same for each category, we sample the same amount of images from each class.

After fitting the K-Means model, we fit a PCA model with the images used to build the visual vocabulary and project them onto a 2-dimensional space to visualize the computed clusters. For clarity, only the 10 clusters (i.e. visual words) that appear the most are plotted. In figure 3 the results are displayed.

For better parameter exploration, we later test different vocabulary sizes (500 and 1500). Even though the plots for these values are not included, the model evaluation for these parameters can be found on the last section of this paper.

Results and Analysis

There are a few different aspects that can be analyzed about this section. First the very fact that euclidean distances lose meaning in high-dimensions, even converging to a constant value on limit, which makes K-Means a not so good clustering technique for data with lots of features. Not only that, but also the fact that two latent variables are definitely not enough to capture

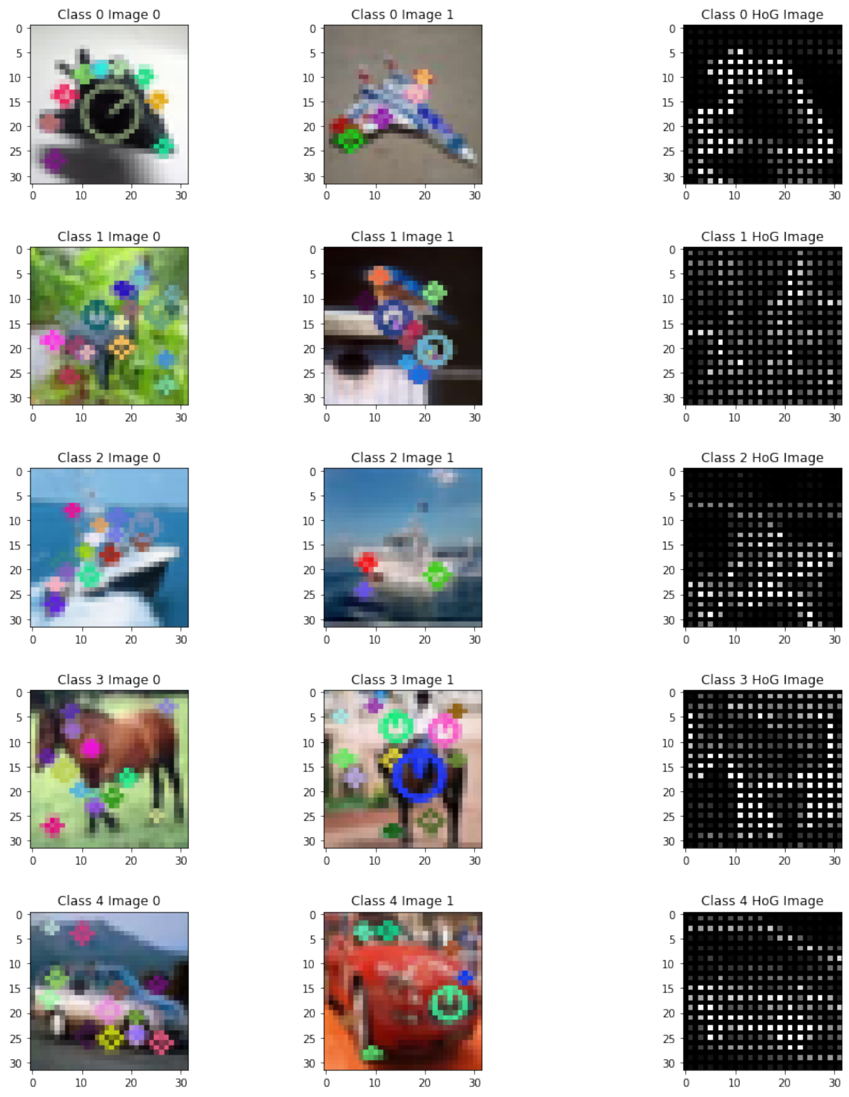


Figure 2: SIFT blobs and HoG images.

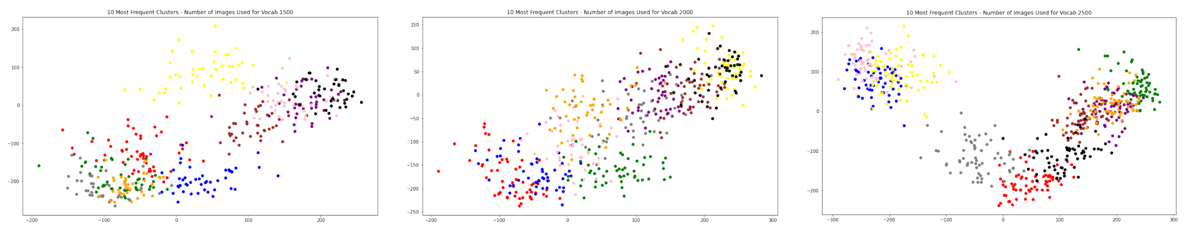


Figure 3: 10 clusters from visual vocabulary using K-Means for 30%, 40% and 50% of images.

the variance of our data (only around 18% is explained). As a combination of these two factors, we observe overlapping clusters with very spread data on the plots of figure 3.

Having said that, it is also possible to notice that there is some kind of separation between the 10 most frequent clusters, indicating that even with these limitations, the model is able to capture some core features about these groups of data (i.e. visual words). This fact will also be later ratified in another section since we can differentiate between frequency of visual words for each class, showing that the generation of clusters was indeed somewhat meaningful.

2.3 Encoding Features Using Visual Vocabulary

For this part, the goal was to represent images with the built visual vocabulary. The implementation thus is quite straightforward: for each image, we use our trained K-Means model to predict a class for all its descriptors (single for HoG, but potentially multiple for SIFT). In figure 4 we show the frequency of visual words for one random image and also the frequency of visual words for all the images in the dataset.

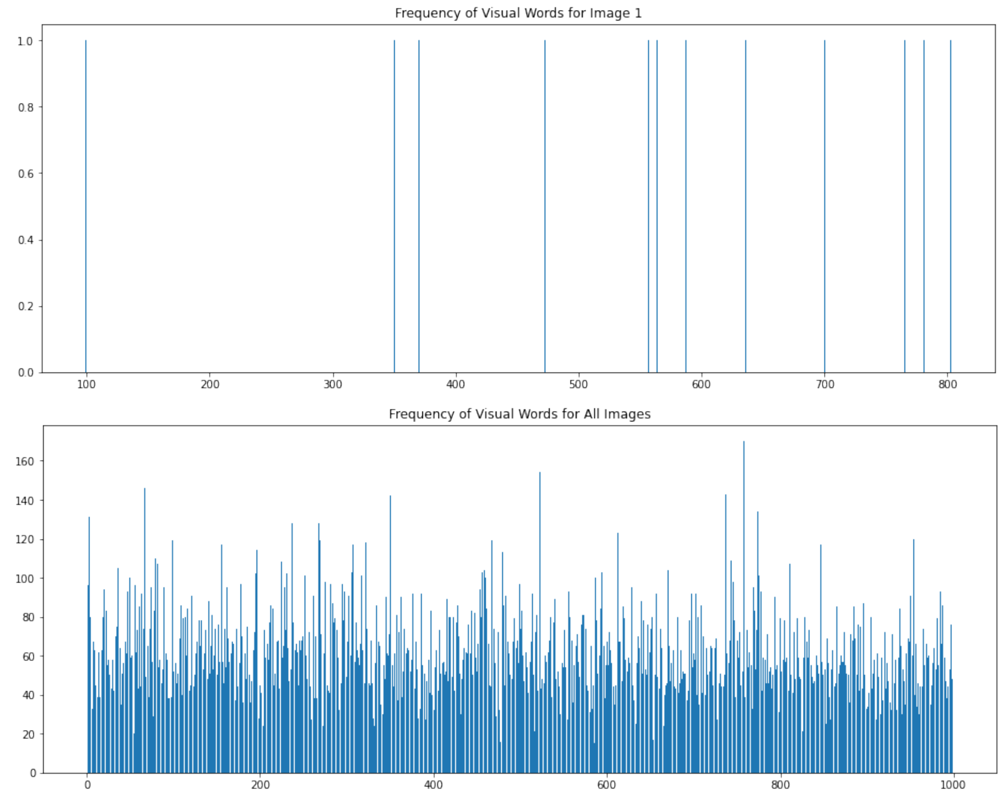


Figure 4: Frequencies of visual words for single image and for the entire dataset.

2.4 Representing Images by Frequencies of Visual Words

Implementation Approach

In this step, we make use of a procedure very similar to the one used in section **2.3**. We also use the trained K-Means model to make predictions on the descriptors of all images, but this time we use quantization for a more stable representation. The idea is to represent images by the frequency of visual words encoded by their descriptors, but divided by the amount of descriptors the image has. With this procedure we ensure a more controlled discrepancy between array lengths.

With the pipeline for representing images with visual words in hands, we plot the normalized frequency of visual words for each class, as shown in image 5.

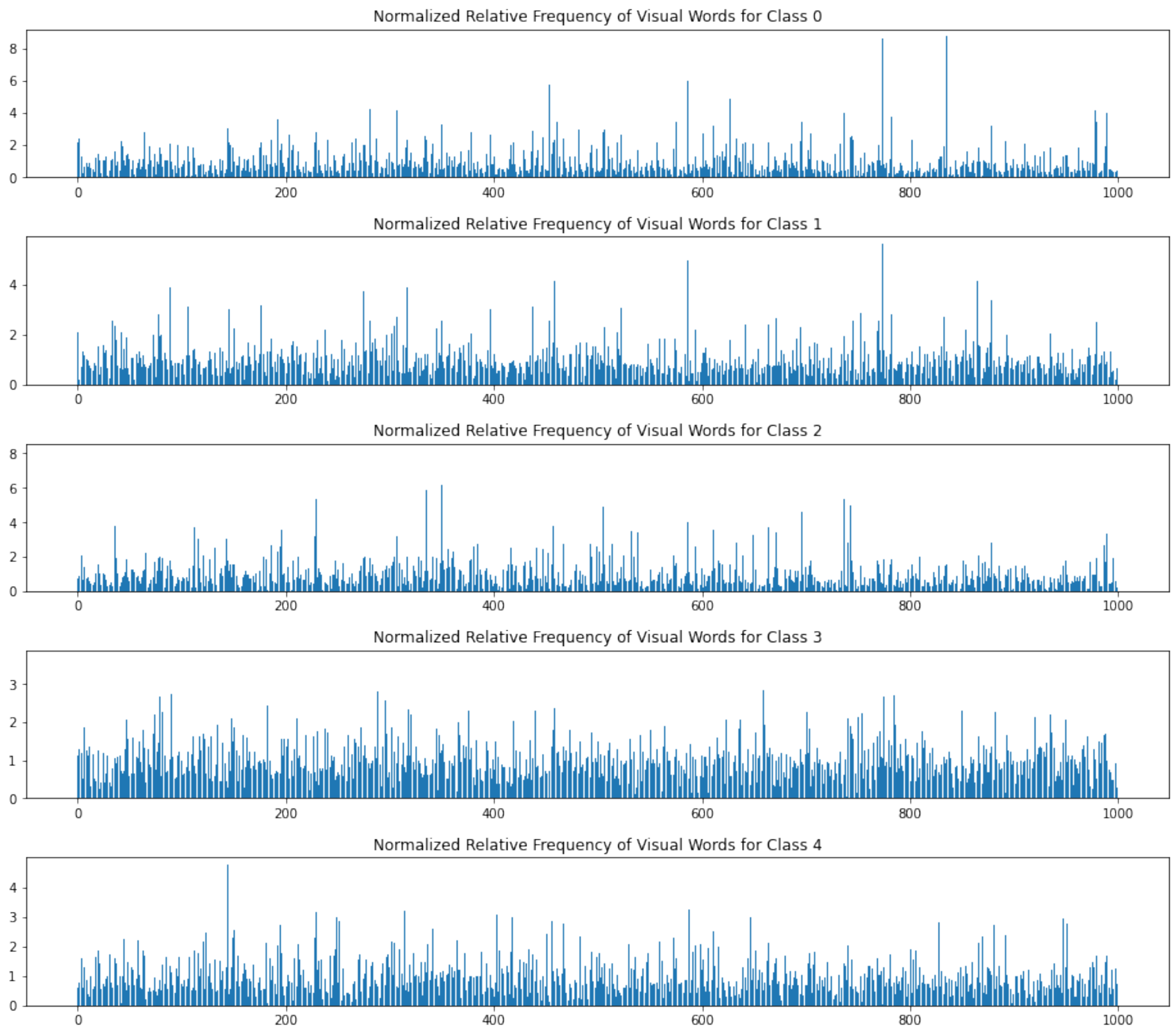


Figure 5: Normalized frequencies of visual words per class.

Results and Analysis

As seen in figure 5, the normalized relative frequency of visual words for each class are quite similar, mainly differing in the spikes of the histograms. We notice however that class 3 and class 4 have more different frequency of visual words distributions, which likely makes them easier to detect later on. As seen in our final results from section 2.6, we see that the images from class 3 and 4 indeed have a significantly higher mAP values than the rest. The most important point to get from the histograms is that this method does not represent a good way of classifying images since the distributions appear quite similar, thus leading to not very good mAP scores.

2.5 Classification

Once we have our normalized representation of visual words for each image, we can finally proceed to image classification. In this section, we are suggested to train a SVM model and for each class compute an one-vs-all classifier. To implement it thus we start by selecting the images we are going to use from a subset of images that were not used to build the visual words vocabulary. We loop through the selected image 5 times, one for each classifier. For each time, we mark images from one class as positive examples and the images from all the other classes as negative examples and then we fit our classifier for the positive class.

For this task, we have a wider selection of parameters to play around and optimize. We tried different number of images used as classifier data (50, 200 and 300), different regularization parameters ($C = 0.2$, $C = 0.4$ and $C = 0.8$) and finally different tolerance for stopping ($tol = 1e^{-2}$, $tol = 1e^{-3}$ and $tol = 1e^{-4}$). We describe the results for all these different tests, along with the other hyperparameter tests we did on the next section.

2.6 Evaluation and Discussion

Implementation Approach

For this last part, we basically go through the entire pipeline with the test data. So we compute the descriptors for the grayscale test images, we build the normalized frequency of visual words for each one of them and finally we predict the probabilities for each image with each classifier. In the end, we are going to have a list of five scores for each image, indicating its probabilities of being from each class.

Once we have these lists for all the images, we can rank them so that for each class the images with highest probability of being from that class come first and the images with lowest probability of being from that class come last. This step is important for the mAP computation, since an unsorted list of images could lead to a much lower score because it could end up accounting for the correctly classified images in later iterations (with large denominator). With the ranked list, visualize the top 5 and bottom 5 predictions for each class as shown in figures 6 and 7, and also compute the mAP scores for each class.

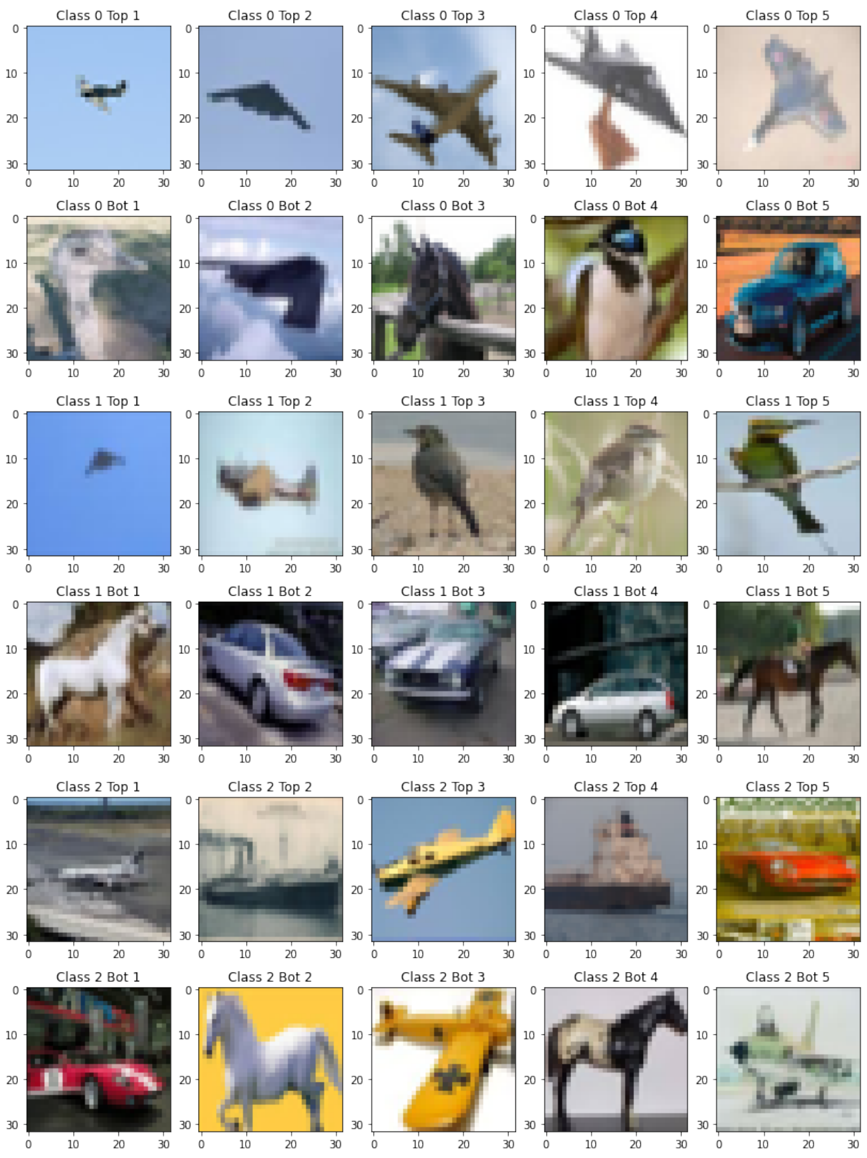


Figure 6: Final results for classes 0, 1 and 2.

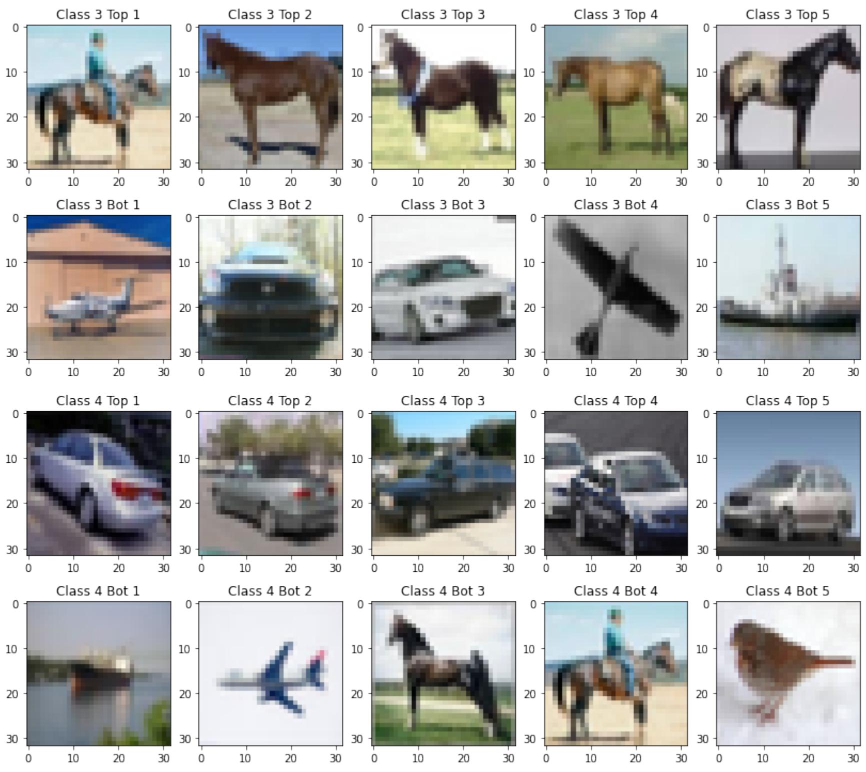


Figure 7: Final results for classes 3 and 4.

Results and Analysis

After implementing our method, we want to test out different settings for the model in order to possibly increase resulting mAP and classified images. Firstly, we test and record the mAP results on different subset ratios for our vocabulary list. These are the ratios of 0.3, 0.4 and 0.5 under the fixed vocabulary size of 1000 and the results are shown in table 1.

mAP						
Vocabulary subset ratio	Class 0	Class 1	Class 2	Class 3	Class 4	Average
30%	0.1774	0.1316	0.1687	0.3584	0.2957	0.2264
40%	0.2039	**0.1813**	0.1895	**0.3635**	**0.3601**	**0.2597**
50%	**0.2217**	0.1056	**0.2072**	0.3614	0.3106	0.2413

Table 1: mAP on different subsets under fixed vocabulary size 1000

As we can see in table 1, the vocabulary subset of 40% results in the highest values of mAP, with a slight improvement over using a subset of 50%. However, the lower subset of 30% results in significantly worse values of mAP. The low ratio causes the vocabulary subset to not contain enough data to properly train our model on, resulting in a lower value for mAP. In this case the higher ratio of 0.4 or 0.5 is preferred. A possible reason for the slightly better results from the 0.4 ratio might be that with a higher vocabulary subset, the model tends to over-fit causing the evaluation results to decrease.

Based on our conclusion that a subset of 40% leads to the best performance, we set this optimal ratio as fixed and compare our results for different vocabulary sizes of 500, 1000 and 1500. The results on the mAP are shown in table 2. As seen in table 2, we report the best

mAP						
Vocabulary size	Class 0	Class 1	Class 2	Class 3	Class 4	Average
500	0.1543	0.0905	**0.1987**	**0.3794**	0.3209	0.2288
1000	**0.2039**	**0.1813**	0.1895	0.3635	**0.3601**	**0.2597**
1500	0.1461	0.0991	0.1649	0.2882	0.3358	0.2068

Table 2: mAP on different vocabulary sizes under fixed subset ratio of 40%

results for a vocabulary size of 1000. This optimal vocabulary size reports only a slight improvement over the vocabulary size of 500, but is significantly better than the vocabulary size of 1500. We believe this is also caused by the over-fitting of the model for large vocabulary sizes.

Based on our optimal settings as discussed above, we want to report the mAP results based on the SIFT and HoG Descriptor. These results are displayed in table 3 and we can see a clear preference of the model for the SIFT-Descriptor compared to the HoG Descriptor when looking at the results for mAP. For the last optimization step we wish to find the optimal hyper-

mAP						
	Class 0	Class 1	Class 2	Class 3	Class 4	Average
SIFT-Descriptor	**0.2039**	**0.1813**	**0.1895**	**0.3635**	**0.3601**	**0.2597**
HoG Descriptor	0.0636	0.0449	0.0956	0.035	0.0615	0.0601

Table 3: mAP comparisons for SIFT and HoG descriptors

parameters of SVM that will results in the highest mAP values for each class. As discussed in the implementation of *Q2.5*, we have tried multiple parameters such as number of images used as classifier data, different kernels, different regularization parameters and different 'tolerance for stopping' values. The hyper-parameters that resulted in the highest values of mAP are displayed in table 4 and table 5 displays the impact of the hyper-parameters of SVM on the resulting mAP values.

Apart from the results on the final Test data, we reported the results on the Training data in table 5 as well. The idea is to show that even though the final results on the Test data were sub-optimal, the model indeed learnt something, not randomly guessing classes.
Apart from changing the hyper-parameters mentioned in table 4 and testing for improved results of mAP, we changed the values of the other hyper-paramters of SVM as well. However, the change in these parameters only resulted in significantly worse results.

Parameter	**Setting**
Images	600
Regularization parameter C	0.2
Tolerance for stopping	$1e^{-2}$

Table 4: The optimal Hyper-parameters of SVM

mAP						
	Class 0	Class 1	Class 2	Class 3	Class 4	Average
Without Hyperparameterization	**0.2039**	0.1813	0.1895	0.3635	0.3601	0.2597
With optimal SVM parameters	0.1739	**0.1948**	**0.2219**	**0.3644**	**0.3642**	**0.2638**
Optimal parameters on Training Data	0.75	0.72	0.79	0.83	0.8	0.778

Table 5: The impact of the hyper-parameters of SVM on the mAP

Under the best setting of the parameters above, we report the final images in figures 6 and 7. When looking at the top 5 images for each class in these figures, we see that for classes 0, 3 and 4 we receive optimal results: all images are correctly classified. For the classes 1 and 2 however, we see that two images are classified wrongly. We believe that this is caused by the images of these specific classes to be harder to differentiate between one another, as argued on the frequency of visual words part. The resulting images follow correctly from our final mAP values in table 5: Class 1 and Class 2 have the lowest mAP values.
When looking at the bottom 5 images for each class, we see that this is much more precise: from all classes, only one image is wrongly put at the bottom even though it belongs in that specific class (class 0 in our example).