University of Amsterdam

Computer Vision Report

Keypoint Detection, Bag of Visual Words and Image Classification

18 October 2024

Students:
Emma Kasteleyn
Jasper van der Valk
Platon Karageorgis

Tutor:
Vlad, Matey & Antonios

Course:
Computer Vision 1

1 Introduction

In computer vision, effective image classification is an important item, with applications for autonomous vehicles to medical diagnostics. This report tries to implement an image classification system using the Bag-of-Visual-Words (BoVW) model. This model tries to represent images through 'visual words' to facilitate the categorization of visual content.

To study this type of image classification, we use the CIFAR-10 dataset, which contains 32x32 pixel RGB images spread across ten dinstinct classes, with 5000 training images and 1000 test images for each class [3]. For the scope of this report, we will focus on five of these classes: Frog, Automobile, Bird, Cat, and Deer. On this dataset, we will construct a visual vocabulary and will train classifiers on this vocabulary.

With this report, we hope to find effective classifiers, also evaluating their performance. Moreover, this report helps us understand the influence of different hyperparameters, like sample size, on the model accuracy. Through this exploration, we aim to highlight the effectiveness and limitations of the BoVW model in real-world image classification.

2 Theory and method

2.1 The Bag-of-Visual words

The BoVW model can be explained in three steps. We first need to extract the features of an image using keypoint detection. Subsequently, we need to constuct a vocabulary of visual words. Lastly, we need to represent the images in a histogram of the 'visual words'. We will go over these three steps in more detail in this section.

2.1.1 Keypoint detection and feature extraction

Keypoint detection can be defined as finding the most 'interesting' points in an image, like corners and edges. We call these points keypoints. After identifying these points, we extract the features, using the surrounding area to create a unique descriptor for each keypoint.

In our report, we focus on two feature extraction methods: the Scale-Invariant Feature Transform (SIFT) and the KAZE algorithms. The SIFT algorithm creates features which are

robust to changes in scale, rotation, and illumination. KAZE also focusses on preserving edge details and object boundaries.

The SIFT algorithm is a feature detection, description and matching model [4]. First, SIFT divides the image patch into an $n \times n$ grid. Each grid cell is then again divided into an $m \times m$ subgrid. To reduce noise and ensure scale invariance, it applies Gaussian blurring to the image patch. Subsequently, it computes the gradient estimates by averaging the gradient around the center of the subcell, weighted by their distance to the center. The gradient is computed using the Difference of Gaussians. Afterwards, SIFT creates a q-cell orientation histogram for each grid element. This histogram contains the weighted gradient estimate. Then, it forms descriptor vectors by concatenating the histograms from each grid element, creating a $n \times n \times q$ dimensional vector. Next, these vectors are normalized to unit length. Lastly, the vector values are thresholded to account for unstable large gradient magnitudes and SIFT renormalizes the descriptor vectors.

The KAZE algorithm is another feature detection, description and matching model[1]. KAZE can be explained in six steps. First, KAZE uses edge-preserving filters (i.e. bilateral filter) to construct nonlinear scale spaces (instead of Gaussian blurring). Then, it detects the keypoints by finding extrema in this space. For each keypoint, the gradient estimates around the keypoint's center are computed, and then weighted by their distance to the center. Subsequently, it makes the keypoints rotation-invariant by assigning an orientation to each one based on the local image gradients. Next, it divides the region around each keypoint into cells and creates a histogram of gradient orientations and magnitudes. Similar to SIFT, these histograms are concatenated to a descriptor vector. Lastly, it follows the same normalization and thresholding process as SIFT.

2.1.2 Constructing the Visual Vocabulary

Using the feature descriptors, we create a single dataset with all the extracted descriptors. This dataset can be used as input for the clustering process. We cluster our data using K-Means clustering. Each cluster center will represent a 'visual' word. We decide on an algorithm with 1000 clusters.

K-means clustering chooses k data points to act as cluster centers [2]. These can be randomly chosen. It then allocates each data point to the cluster with the nearest center, which can be done by computing the Euclidean distance between each data point and the cluster centers:

$$\Phi(\text{clusters, data}) = \sum_{i \in \text{clusters}} \sum_{j \in \text{ith cluster}} (x_j - c_i)^T (x_j - c_i)$$

, where x_j is a data point, and c_i is the center of the ith cluster. Then, the algorithm recalculates the center of each cluster as the mean of the points assigned to that cluster. This 'new' center is the average of all feature vectors in the cluster. This process is repeated until the cluster centers change very little.

2.1.3 Histogram of visual words

To construct a histogram of visual words, we set the number of bins equal to the number of visual words, which is 1000 in our case. For each descriptor in the image, we assign it to the visual word, adding it to the bin of that word. This results in a histogram representing the frequency of each visual word in the image.

In order to get an idea of the visual word representation of a class, we take the mean over all histograms constructed for that class. This results in a histogram containing the average frequency of each visual word for a class.

2.2 Creating and training classifiers

After having obtained a BoVW, we create Support Vector Machine (SVM) classifiers. An SVM is a supervised machine learning model, which aims to find the optimal hyperplane that best

seperates the classes in the feature space. There are different hyperplanes which can be chosen (i.e. linear or radial basis function (Gaussian)).

This SVM classifier can be trained using a One-vs-Rest (OvR) strategy. OvR can handle multi-class classification problems, which is necessary for our five class example. It trains five classifiers in our case, where each classifier is i.e. class A vs. not class A. Thus, each classifier is trained to recognize one class as the positive class and all others as the negative class. Using these classifiers, we can make predictions on test data, where each classifier will provide a probability for its positive class.

2.3 Evaluating the classifiers

To evaluate our classifiers, we will compute the mean Average Precision (mAP). This is defined as

$$\text{mAP}c = \frac{1}{m_c} \sum i = 1^n \frac{f_c(x_i)}{i}$$

where n is the total number of images, m_c is the number of images of class c, x_i is the ith image in the ranked list, and $f_x(x_i)$ returns 1 if x_i is of class c and 0 otherwise. We rank the images based on their probability of being in the specific class.

These classifiers can be optimized by looking at specific settings of hyperparameters. We will look at nine different settings. First of all, we will vary the number of visual words, thus the number of clusters. We will also use different training subset sizes for the classifiers. Moreover, we change the SVM parameters used: the kernel types, the regularization parameter values, and gamma values. Moreover, we will vary the upscaling factor, so varying the size of the images. Lastly, we will adjust the settings of the feature extractors. For the SIFT extractor, we will change the number of keypoints and scale levels. For the KAZE extractor, we will change the threshold.

3 Results

In Figure 1, we visualize two images per class and show the keypoints found by the SIFT and KAZE algorithm. This plot has been generated by the FAISS library which is well-known for its faster results in comparison to other methods [5]. We notice that the SIFT algorithm tends to detect more keypoints than the KAZE algorithm. In Deer Image 2, the KAZE algorithm does not even detect any keypoints. This could be because the KAZE algorithm constructs a nonlinear scale space, which may make it more robust to detecting keypoints.

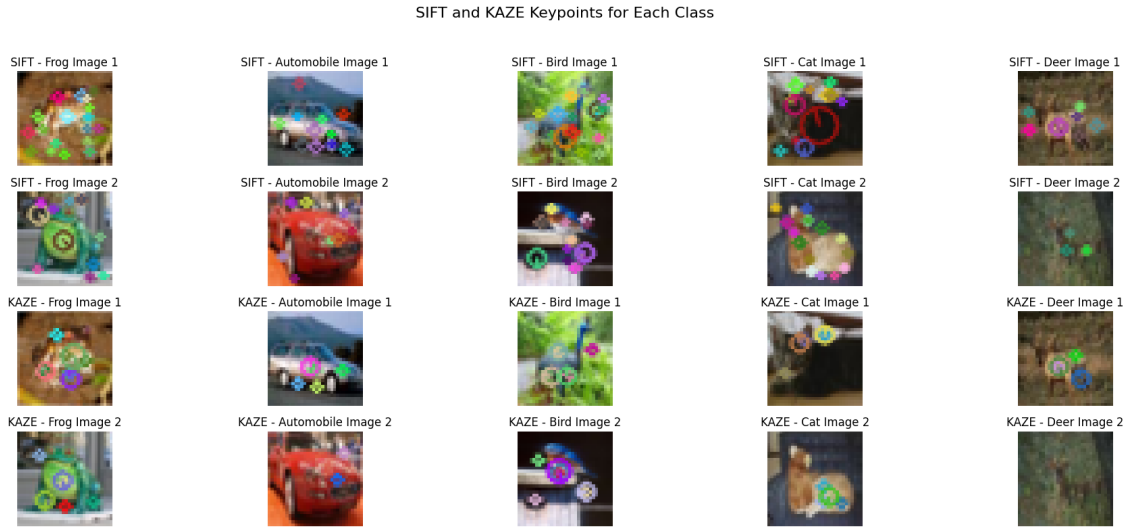
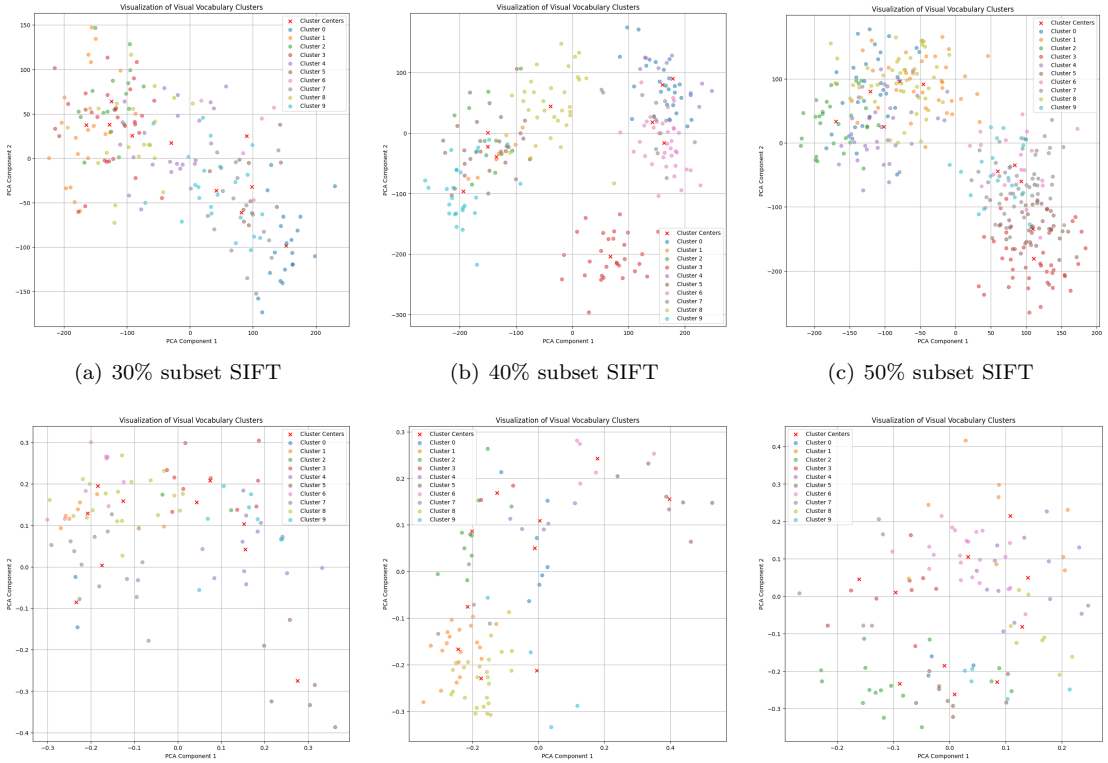


Figure 1: Two images per class type with their corresponding descriptors obtained by SIFT and KAZE. The plot was generated using FAISS (Facebook AI Similarity Search).

After performing the keypoint detection, we use the feature descriptors to create a visual vocabulary using K-means. We perform this construction for three different subset sizes: 30%, 40%, and 50%. The first ten cluster centers and their corresponding cluster points are visualized in Figure 2. We notice that the clusters seem to get bigger as we increase the subset size. This can be explained by the detection of more feature descriptors if the image dataset is larger. Furthermore, the clusters are less clear for smaller subsets. That can be due to that a smaller subset has less variability, meaning that PCA might find it difficult to distinguish well-defined clusters, resulting in scattered/overlapping clusters. Moreover, again it becomes clear that KAZE detects less keypoints, and thus descriptors, as there are significantly less points in the plots of the KAZE algorithm.



(a) 30% subset SIFT (b) 40% subset SIFT (c) 50% subset SIFT

(d) 30% subset KAZE (e) 40% subset KAZE (f) 50% subset KAZE

Figure 2: The SIFT and KAZE descriptor clusters using FAISS

Moreover, to get a better understanding of the data we repeat the same procedure using the sk-learn library instead of Faiss. The result is depicted in Figure 3. Overall, it supports the arguments we made for the previous case using Faiss, however in this case the clusters are much more dense and clearer.

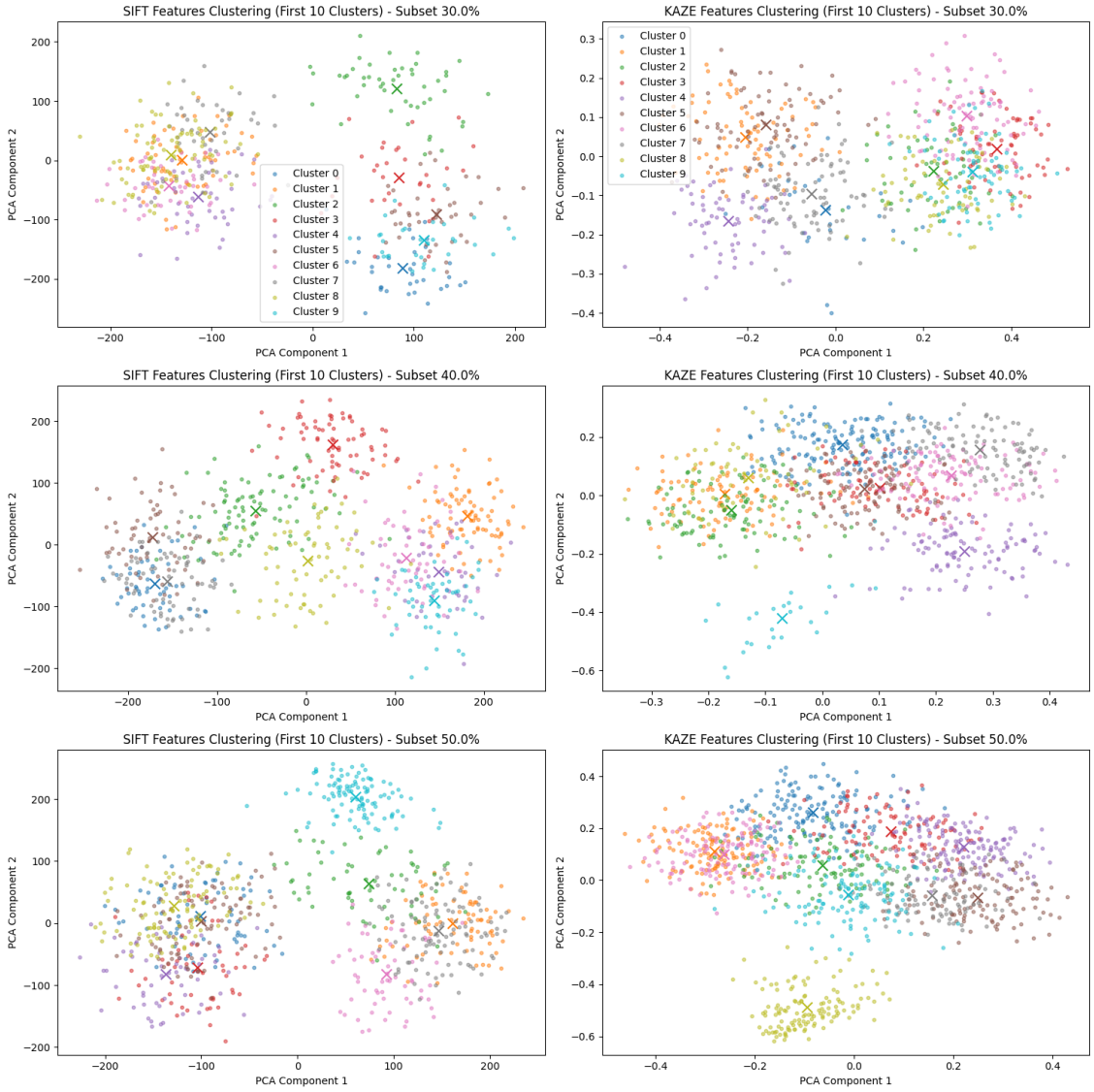
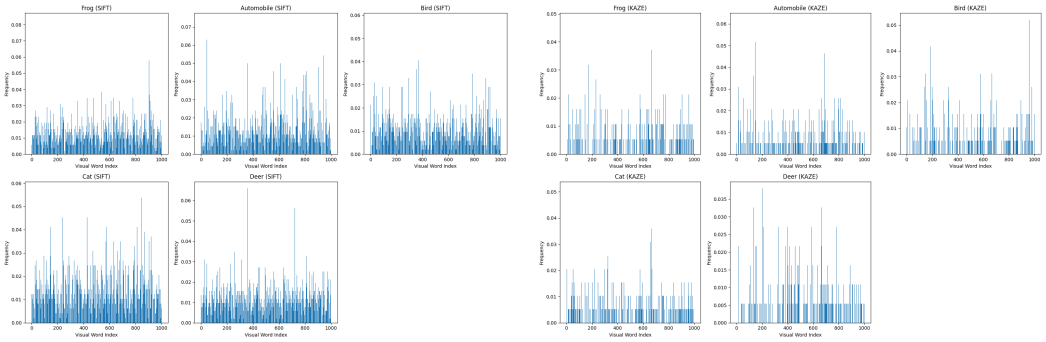


Figure 3: The SIFT and KAZE descriptor clusters using Scikit-Learn

Subsequently, now that we have created a visual vocabulary, we can visualize this for each class using the mean histogram of visual words for each class for the 50% subset (Figure 4). We notice that the y-axes of the SIFT algorithm have a bigger range (up to 0.07) than the KAZE algorithm (up to 0.05). The y-axes present the frequency of a visual word. The KAZE algorithm detects less descriptors, meaning that it likely also creates less visual words. Some classes have high bin values for specific word indices. Especially the bird and deer classes seem to have high frequencies for some indices in both algorithms.



(a) SIFT Histograms (b) KAZE Histograms

Figure 4: Mean histograms of visual words for each class

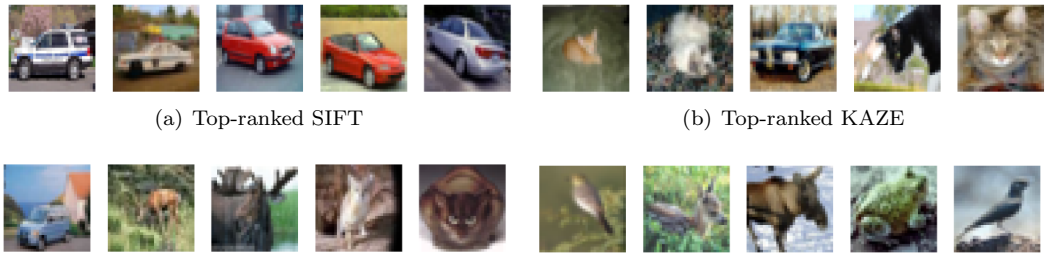
We will now create one-vs-rest SVM classifiers for each feature extraction method. We use the part of the training data which was not used for the visual dictionary to get a more robust evaluation of the visual vocabulary's effectiveness. We test the performance of different vocabulary subset sizes, meaning we use 70%, 60%, and 50% of the training data for the SVM classifiers to pick the optimal size. This way, we ensure a more robust evaluation of the visual vocabularies.

In Table 1, the precision of each specific class and the mean average precision (mAP) is shown. We notice that the mAP is maximized for a subset size of 50%. Therefore, we will continue with this subset size in our further analysis. We also notice that the detection of automobiles has a very high precision in the 50% subset, while the bird and deer precision is average. Therefore, our earlier observations of high frequency peaks in these class types do not seem to have much effect on the precision. Moreover, the SIFT and KAZE algorithm seem to have about the same performance based on precision. If we look at the non-rounded numbers of the mAP of 50%, we notice that the KAZE extractor just outperforms the SIFT with a mAP of 0.3101 and 0.3068 respectively.

Subset	**Method**	**Frog**	**Automobile**	**Bird**	**Cat**	**Deer**	**mAP**
30%	SIFT	0.22	0.18	0.21	0.22	0.17	0.20
	KAZE	0.18	0.24	0.18	0.18	0.21	0.20
40%	SIFT	0.19	0.23	0.25	0.18	0.25	0.22
	KAZE	0.20	0.15	0.24	0.23	0.18	0.20
50%	SIFT	0.27	0.46	0.31	0.28	0.32	0.31
	KAZE	0.30	0.42	0.29	0.27	0.29	0.31

Table 1: Precision per class and mean average precision for SIFT and KAZE at different subsets

In order to get a better understanding of what these precisions mean, we can visualize the top 5 ranked and bottom 5 ranked images for each class type based on the confidence that it is classified as the correct class type. We will first look at the best performing class type, which is automobiles (Figure 5). We notice that the top 5 images of the SIFT algorithm include only cars, which explains the high precision. However, in the bottom-ranked images, we also observe an automobile. This can explain why the precision is not higher than 46%. Looking at the images of the KAZE extractor, we only observe one automobile in the top-ranked images. However, there appear no cars in the bottom-ranked images, which can explain why the precision is not much lower (0.42) than the one from the SIFT algorithm.

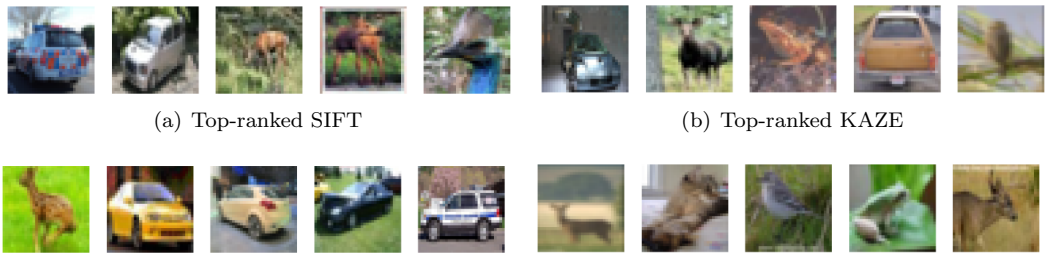


(a) Top-ranked SIFT (b) Top-ranked KAZE

(c) Bottom-ranked SIFT (d) Bottom-ranked KAZE

Figure 5: Ranked images of the Automobile based on their confidence scores

In Figure 6, we show the images of the lowest scoring class type based on precision, Cat. We indeed observe that in both algorithms, the top scoring images are never cats. Nevertheless, the bottom-ranked images for SIFT do not include any cats either. That could be why the precision is not lower than 28%. The KAZE bottom-ranked images do include one cat, but the precision of KAZE is not much lower than that of SIFT (27%), possibly due to the large dataset.



(a) Top-ranked SIFT (b) Top-ranked KAZE

(c) Bottom-ranked SIFT (d) Bottom-ranked KAZE

Figure 6: Ranked images of the Cat based on their confidence scores

Lastly, so far we have focused on which method performs the best. However, we can also tune the parameters to different settings and see how the mAP changes. This is done in Table 2. We have ranked the parameter values from best to worst mAP for the SIFT extractor. All other parameters are not varied during this process, with the basic settings being highlighted as bold in the table. Therefore, we notice that the basic settings for the number of visual words, kernel, octave layers, the kaze threshold, and the upscaling factor are non-optimal.

The most significant shift was caused by upscaling the images. Since our images are small and only 32x32, we attempted to upscale them 4 and 8 times. This yields a mAP of 35% for the KAZE algorithm. For the SIFT algorithm, switching the kernel to Radial Basis Function (rbf) has the most effect on the mAP.

We also notice that the KAZE and SIFT parameter almost always agree on the best parameter, except for the number of visual words, with KAZE preferring 1000 visual words instead of 500. Nevertheless, they did disagree on the worst value for the number of visual words.

Changing the number of keypoints only has an improving effect for 10 keypoints. If there are that many keypoints found in the SIFT extractor. Probably, in an image of 32x32 there are not 50 or 100 keypoints found, which means that the extractor just returns the maximum amount of keypoints found.

Lastly, varying the gamma does not yield different mAPs. This is because our basic settings have a linear kernel. The gamma parameters only have an effect on non-linear kernels. Therefore, we see not difference in mAP.

Parameter	Value	mAP SIFT	mAP KAZE
Varying Number of Visual words	500	0.287323	0.280079
	1000	0.287166	0.283483
	1500	0.280332	0.280023
Varying Subset Size	**0.5**	0.287166	0.283483
	0.3	0.263850	0.270391
	0.4	0.256327	0.243683
Varying Kernel	*rbf*	0.343207	0.313739
	linear	0.287166	0.283483
Varying Regularization Parameter	0.1	0.324890	0.302868
	1.0	0.287166	0.283483
	10.0	0.284132	0.274618
Varying Gamma	***scale***	0.287166	0.283483
	0.1	0.287166	0.283483
	0.01	0.287166	0.283483
Varying Number of Keypoints	10	0.296341	0.283483
	50	0.287166	0.283483
	100	0.287166	0.283483
	None	0.287166	0.283483
Varying Octave Layers	7	0.313696	0.283483
	5	0.301577	0.283483
	3	0.287166	0.283483
	None	0.287166	0.283483
Varying Kaze Threshold	0.001	0.287166	0.283483
	None	0.287166	0.283483
	0.002	0.287166	0.264780
	0.005	0.287166	0.214202
Upscaling factor	4	0.331641	0.354515
	8	0.322619	0.353950
	1	0.287166	0.283483

Note: During the hyperparameter tuning, all parameters were kept constant except for the indicated parameter. The bold parameter values were used as the basic settings during this process.

Table 2: mAP Results for Varying Parameters

In Table 3, we have used the individually best performing parameter settings and combined them to see how much the mAP would improve. We notice that the mAP improves significantly, yielding an mAP of 42.6% for SIFT and 42.1% for KAZE. This is the highest mAP observed during our hyperparameter tuning, which shows the effectiveness of tuning parameters.

Parameter	Setting
Number of Visual words	1000
Subset Size	0.5
Kernel	*rbf*
Regularization Parameters	0.1
Gamma	*scale*
Number of Keypoints	100
Octave Layers	7
Kaze Threshold	0.001
Upscaling factor	4
mAP SIFT	0.4295
mAP KAZE	0.4291

Table 3: The best parameter setting and the resulting mAP for each method

4 Discussion

In conclusion, we were able to get a clear understanding of how different parameters affect image classification. Moreover, we have found that the SIFT and KAZE algorithms perform pretty similar based on precision. Although our report tries to improve the effectiveness of the BoVW model, some limitations also need to be discussed.

First of all, it is important to note that our images are 32x32 pixels. Therefore, it is hard to detect valuable visual words for each class type. We already saw a significant improvement of precision when we upscaled the image, which leads us to believe that a higher resolution image could possibly lead to higher precision.

Moreover, we notice different cluster plots for the Faiss and Scikit-learn algorithm in Figures 2 and 3. This can be due to the different goals of the algorithms. While Faiss is known to be fast, Scikit-learn takes longer, but has a higher precision. Therefore, they have different optimization objectives, also yielding different results in the cluster graphs.

We note that KAZE is a more robust algorithm when compared to SIFT. Although the keypoint extractors now perform pretty similar based on their mAP, the robustness of the KAZE algorithm can be helpful for datasets with higher resolution images. This can be interesting for future research.

We also discuss whether the histogram peaks tell in Figure 4 tell us anything about the precision. We do not observe a higher precision in the class types for which the histogram peaks were the highest. There is no direct relationship between precision and histograms as histograms present the characteristics and richness of the feature space, while precision quantifies the accuracy of positive predictions in a classification model, reflecting how well the model distinguishes between classes based on those features.

At the moment, we only vary one parameter at a time during hyperparameter tuning. This already significantly increases our mAP. However, it would be interesting to vary multiple parameters at one time and seeing if this results in an even higher mAP. We already noted this with the gamma parameter, as this parameter only has an effect on the mAP if a non-linear kernel is used, which is not the case in our basic settings. Therefore, we expect the mAP to be able to improve even more if all parameters are allowed to vary at once. This would be computationally intensive, so this is why we leave this as a suggestion for future research.

References

[1] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J. Davison. "KAZE Features". In: *Computer Vision – ECCV 2012*. Springer, 2012, pp. 214–227. DOI: 10.1007/978-3-642-33783-3_16. URL: https://link.springer.com/chapter/10.1007/978-3-642-33783-3_16.

[2] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. 2nd. Upper Saddle River, NJ, USA: Pearson, 2011.

[3] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009. URL: `https://www.cs.toronto.edu/~kriz/cifar.html`.

[4] David G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. 1999, pp. 1150–1157. URL: `https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf`.

[5] Cun Mu, Binwei Yang, and Zheng Yan. "An Empirical Comparison of FAISS and FENSHSES for Nearest Neighbor Search in Hamming Space". In: *CoRR* abs/1906.10095 (2019). arXiv: `1906.10095`. URL: `http://arxiv.org/abs/1906.10095`.