CMPE 537 Assignment 3
Local Descriptor-based Image Classification

Burak Yıldızöz, Ümit Develer and Oğuzhan Sevim

January 20, 2021

1 Introduction

In this project, we design and implement different local descriptor-based image classifiers.

The dataset used in this project is called Caltech20. Each image in this dataset is labelled by one of 21 classes including background class. Furthermore, training and test sets consist of 2972 and 380 images.

For the sake of simplicity, only the parts that are our own implementations will be discussed in detail. The other implementations that are implemented by a library will only be shortly mentioned.

2 Local Descriptors

As the local descriptors, we have used HOG, ORB, and SIFT. Among these, only HOG is implemented by one of the group members. For the other descriptors, OpenCV library is used. Therefore, the details of SIFT and ORB will not be discussed in the report.

2.1 Histogram of Oriented Gradients (HOG)

The first local descriptor used is HOG. In the regular usage of HOG, a given image is first partitioned into $n \times m$ grid. Then, we find the histogram of gradient orientations within each of the windows of the resulting grid. Consequently, we tread the centers of each window as a keypoint where the calculated HOG is considered to be the descriptor of that keypoint.

Since the centers of these windows do not have any particular information, considering the centers of windows as keypoints is against the fundamental idea of keypoints. Therefore, we have used another approach where we first find the keypoints (without descriptors) of a given image by using OpenCV's SIFT class. Then, around each of these keypoints, we draw a square window where we calculate the histogram of gradient orientations. The size of the square window is taken as a design hyperparameter and tried to be optimized by a grid search. As a result this window size is taken as 25×25.

For a single image, the general pipeline for finding the HOG descriptors is as follows:

- Find the SIFT keypoints (not the descriptors) on the RGB image. This step is implemented by using OpenCV's SIFT function.
- Convert image into grayscale format.

- Calculate the vertical and horizontal gradient vectors of each pixel of the grayscale image by using the Sobel filters shown below:

$$\begin{bmatrix} 1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{1}$$

- For each pixel, calculate the angle of the gradient. This is calculated by finding the arctan of horizontal and vertical gradient directions.
- Around each keypoint, draw a square with size of 25×25 pixels. For this region calculated the histogram of the gradient angles by using a bin size of 30.
- If the number of keypoints are higher than a threshold (taken as 500 after several trials), randomly select 500 of these descriptor.
- Return the descriptors of each of these keypoints.

3 Dictionary and Feature Quantization

As the clustering algorithm, we have used *k*-Means. It is implemented by one of the group members.

3.1 *k*-Means Clustering

The descriptors generated in the local descriptor part are converted into the codewords. In this step, *k*-Means clustering algorithm is used to learn dictionary centers (codewords). The outline of the *k*-Means clustering algorithm can be summarized as:

1) Select initial *k* centroids ar randomç
2) Assign each descriptor to the cluster with the nearest centroid.
3) Compute each centroid as the mean of the decriptors assigned to it.
4) Repeat steps 2 and 3 until nol change or maximum number of iterations reached.

Each cluster center (centroids) produced by *k*-Means becomes a codeword. The number of clusters is the codebook size. The generated codebook is used for quantizing features. Quantization of features means that the feature vector maps it to the index of the nearest codeword in a codebook.

For feature quantization, histogram is used. Each image can be represented into a histogram of codewords. The bins in the histogram correspond to the codewords. The count of every bin corresponds to the number of times the corresponding codeword matches a keypoint in the given image. In this way, an image can be represented by a histogram of codewords. The histograms of the training images can be used to learn a classification model.

4 Classifiers

By quantization, each a constant number of features are extracted from each image in the training and data sets. For the classification part, we have experimented with following 3 classification methods:

- Support Vector Machine (SVM)
- Multilayer Perceptron (MLP)
- k-Nearest Neighbor (kNN)

Withing these classifiers, only kNN is implemented by one of our group members. The implementation of the remaining classifiers are done by using scikit learn library. The details of kNN implementation is given under Section 4.1.

Each of these classifiers have their own specific parameters to be tuned. Hyperparameters of the MLP are number of hidden layers and the size of each layer. For SVM, we have inverse regularization parameter C. Finally, for kNN, k is the hyperparameter to be tuned. Tuning of the classifiers are done by using grid search and the details are shown in Section 5.

4.1 k-Nearest Neighbor Classifier (kNN)

For the implementation, the ground truth values are saved first. This is all that is done for training part. For the prediction of a given histogram, the distance between that histogram and all ground truth histograms are calculated. Then, the minimum k of them are selected. If there is only one maximum occurence in the corresponding labels, then the prediction is that label. If two or more labels are equally occured in these k distances which are occured the most, then the label of the minimum distance is chosen.

5 Results of Hyperparameter Tuning

In this section, we present the results of hyperparameter tuning processes of each group members' pipelines are discussed under following 3 subsections.

5.1 HOG-SVM Tuning

The hyperparameters of HOG-SVM pipeline can be listed as follows:

- **w** : window size where HOG is calculated around each keypoint.
- **k** : number of centroids considered in the k-means clustering. Please notice that **k** corresponds to the number of words in the dictionary.
- **c** : Inverse regularization parameter for the SVM. As we increase **c**, the model tends to fit better to the training data. However, after a certain point, it starts overfitting.

With the definitions above, the $5-$fold cross-validation accuracies for different **k** and **c** values are shown in the Table 1. Since we can't show a $3D$ table here, the affects of window size is not illustrated. However, the highest cross-validation accuracies are obtained when the square window size is taken as 25×25.

Three highest mean of 5−fold validation accuracies are shown in red colors in Table 1. By using the highest accuracy, final model parameters are taken as $\mathbf{w} = 25$, $\mathbf{k} = 275$, and $\mathbf{c} = 300$.

Table 1: HOG-SVM tuning validation accuracies

k\c	100	150	200	250	300	350	400	450	500
100	64.14	64.75	65.52	65.32	65.45	65.35	65.42	65.59	65.89
125	64.68	65.22	65.62	65.76	66.09	66.23	66.50	66.40	66.30
150	66.36	66.53	66.60	66.50	66.36	66.46	66.43	66.33	66.43
175	65.66	66.13	66.46	66.67	66.94	66.80	67.00	67.31	67.41
200	67.91	67.85	68.38	68.35	68.35	68.48	68.28	68.28	68.35
225	67.47	67.61	67.14	67.00	67.27	67.40	66.97	66.70	66.73
250	66.36	67.47	67.64	67.85	67.71	67.41	67.47	67.27	67.47
275	68.22	68.35	68.05	68.32	68.72	68.52	68.45	68.25	68.41

5.2 ORB-MLP and ORB-SVM Tuning

The feature size of each descriptor obtained from ORB is fixed and 32. The parameters to be adjusted for the best MLP classification is listed below:

- **k**: the number of centroids in the *k*-Means clustering. **k** centroids equal the number of words in the codebook.
- **hidden_layer_size**: The numbers of neuron and hidden layer are the most crucial paramters in the MLP. These parameters ara chosen from 5-fold cross validation (using *GridSearchCV*).
- **max_iter**: As the maximum number of iterations increases, the chance of convergence also increase. However, execution time becomes longer. Please select this value according to your computer specifications.
- **learning_rate**: Learning rate of the MLP can be chosen as *constant* or *adaptive*. *Adaptive* learning rate keeps the learning rate constant. If there exists a fail in decrease of total loss, the current learning rate is divided by 5. *Adaptive* learning rate is used for whole pipeline.

From the parameters defined above, the 5-fold cross validation accuracies are shown in Table YY for different **k** and **hidden_layer_sizes**.

To compare MLP with SVM, ORB-SVM tuning also is provided. As in the HOG-SVM section, the parameters to be adjusted for the best SVM classification is listed below:

- **k**: the number of centroids in the *k*-Means clustering. **k** centroids equal the number of words in the codebook.
- **c**: inverse regularization parameter for the SVM. As we increase **c**, the model tends to fit better to the training data.

The 5-fold cross-validation accuracies for differetent **k** - **hidden_layer_size** and **k** - **c** parameters are shown in Table 2. T

Table 2: ORB-MLP and ORB-SVM tuning validation accuracies

k	Hidden Layer Size for MLP			C for SVM		
	(100)	(100, 100)	(100, 100, 100)	200	300	400
100	29.47	27.89	28.68	33.15	36.57	37.11
300	31.31	28.42	29.63	43.15	37.63	42.10
500	33.15	29.31	30.14	42.71	42.63	41.73

5.3 SIFT Tuning

The feature size of each descriptor obtained from SIFT is fixed and 128. The parameters to be adjusted for classifications are listed below:

- **num_cluster**: Cluster size of each histogram.
- **k**: Number of neighbors to use for k-nearest neighbors vote.
- **hls**: Number of neurons in hidden layer. (Hidden Layer Size)
- **C**: Regularization parameter. The strength of the regularization is inversely proportional to *C*.

The results for all classifiers are in Table 3.

Table 3: Mean F1 scores of SIFT for all classifiers

num_cluster	k for kNN			hls for MLP			C for SVM		
	1	3	7	(50,)	(100,)	(150,)	200	300	400
100	24.98	24.38	23.09	29.7	32.63	33.85	44.52	42.66	41.64
300	21.42	20.94	17.57	33.41	38.52	41.43	50.94	49.86	48.1

6 Results

In this section, we present the results obtained from following 3 main models: HOG-SVM, ORB-MLP, and SIFT-kNN. The results of each model are discussed in the following sections. The classes are labeled as follows:

- 0: airplanes, 1: anchor, 2: background_class, 3: barrel, 4: camera,
- 5: car_side, 6: dalmatian, 7: Faces, 8: ferry, 9: headphone,
- 10: lamp, 11: pizza, 12: pyramid, 13: snoopy, 14: soccer_ball,
- 15: stop_sign, 16: strawberry, 17: sunflower, 18: water_lilly, 19: windsor_chair,
- 20: ying_yang

6.1 HOG-SVM

The evaluation metrics for HOG-SVM pair are given under Table 6. Mean F1 test score is found 46.13, where the test accuracy is 49.21%. The confusion matrix for the HOG-SVM is shown in Figure 1. The most confused class in the training is (4:camera). It is misclassified 9 times as the background image.

Table 4: Metrics for the best HOG-SVM tuning

	Per-class F1-score	Per-class Precision	Per-class Recall
0	85.11	74.07	100
1	36.36	28.57	50.00
2	0.00	0.00	0.00
3	37.04	71.43	25.00
4	24.00	60.00	15.00
5	88.37	82.61	95.00
6	51.43	60.00	45.00
7	70.59	58.06	90.00
8	70.97	100	55.00
9	30.00	30.00	30.00
10	6.90	11.11	5.00
11	55.56	62.50	50.00
12	46.67	70.00	35.00
13	56.52	50.00	65.00
14	0.00	0.00	0
15	75.00	100	60.00
16	28.57	33.33	25.00
17	58.82	71.43	50.00
18	42.55	37.04	50.00
19	81.08	88.23	75.00
20	23.08	50.00	15.00

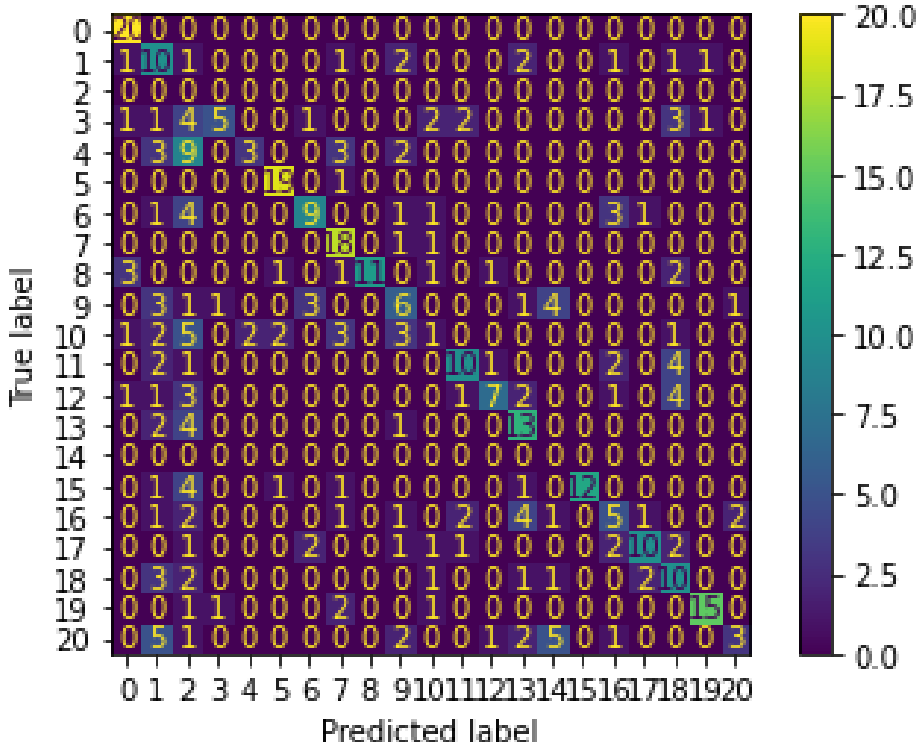

Figure 1: HOG-SVM for **k**=275 and **C**=300

6.2 ORB-MLP and ORB-SVM

Mean F1 test score is found 40, where the test accuracy is 43.15%. As can be seen from Table 2, ORB-SVM tuning method is better than ORB-MLP. There is no testing data for 2 : background_class and 14 : soccer_ball. From Table 5, we have no metrics for these classes. The highest precision is obtained for 20 : ying_yang class and the highest recall is obtained for 5 : car_side and 13 : snoopy classes. The highest F1 score is obtained for 13 : snoopy class. The most confused class in the training is 3:barrel. It is misclassified 7 times as the 16: strawberry image.

Table 5: Metrics for the best ORB-SVM tuning

Class	Per-class F1-score	Per-class Precision	Per-class Recall
0	51.16	47.83	55
1	50	50	50
2	0	0	0
3	12.5	16.67	10
4	27.59	44.44	20
5	65.31	55.17	80
6	41.18	50	35
7	39.29	30.56	55
8	27.78	31.25	25
9	40.91	37.5	45
10	6.45	9.09	5
11	51.28	52.63	50
12	38.71	54.54	30
13	76.19	72.72	80
14	0	0	0
15	60.61	76.92	50
16	21.05	16.22	30
17	60	60	60
18	27.78	31.25	25
19	56.25	75	45
20	80	93.33	70

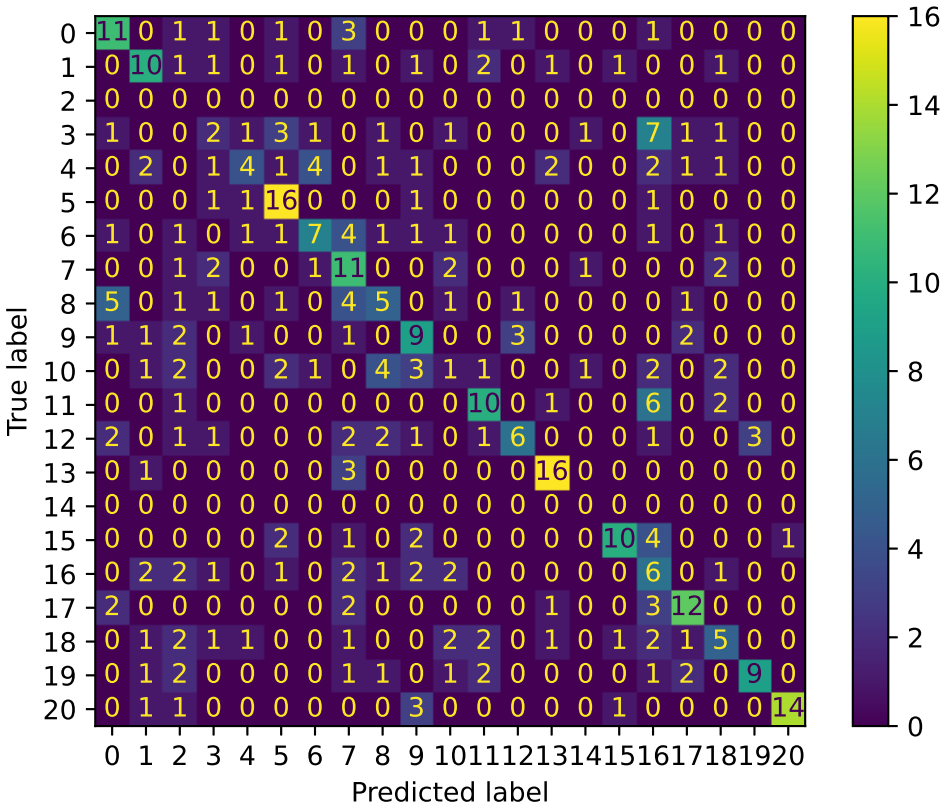

Figure 2: ORB-SVM for **k**=300 and **C**=200

6.3 SIFT (Best Model)

According to the results in Table 3, the best classification method for SIFT is SVM with $C = 200$. SVM shows stable results around 45-50 mean F1 score. kNN seems to be not a good choice for classification of a big dataset. As hidden layer size of MLP icreases, the success rate increases, along with runtime though. But since it is worse than SVM for all cases, and SVM is faster, the best choice for classification is clearly SVM. Increasing dictionary size does not affect the results significantly.

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

Table 6: Metrics for the best SIFT-SVM tuning

	Per-class F1-score	Per-class Precision	Per-class Recall
0	51.85	41.18	70
1	50	66.67	40
2	0	0	0
3	21.43	37.5	15
4	48.28	77.78	35
5	78.26	69.23	90
6	82.05	84.21	80
7	62.75	51.61	80
8	41.38	66.67	30
9	52.38	50	55
10	6.897	11.11	5
11	52.63	55.56	50
12	42.42	53.85	35
13	71.79	73.68	70
14	0	0	0
15	75	100	60
16	48.78	47.62	50
17	78.95	83.33	75
18	51.16	47.83	55
19	73.68	77.78	70
20	80	80	80

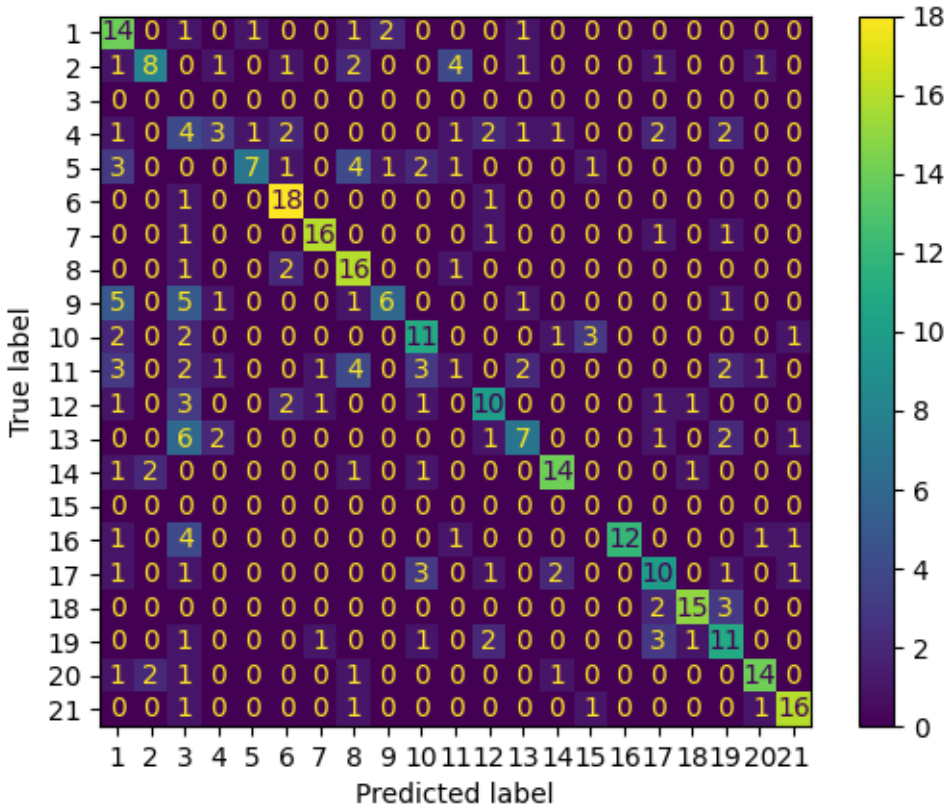

Figure 3: SIFT-SVM for **C**=200

7 Contributions of Group Members

7.1 Oğuzhan Sevim

My contributions to this assignment can be listed as follows:

- Implementation of HOG discussed in Section 2.1.
- Tuning the HOG-SVM model as discussed in Section 5.1.
- Writing *classification.py* (other than kNN implementation) and *evaluation.py*
- Preparing the general structure of the report on Overleaf.
- Helping the general pipeline in the GitHub.

7.2 Ümit Develer

My contributions to this assignment can be listed as follows:

- Implementation of k-Means clustering algorithm in Section 3.

- Implementation of quantization with histogram in Section 3.
- Writing *dictionary_and_quantization.py*
- Testing ORB-MLP and ORB-SVM tunnings.
- Helping the general pipeline in the Github.

7.3 Burak Yıldızöz

My contributions to this assignment can be listed as follows:

- OpenCV descriptors and BOW implementation
- Own implementation for KNN classifier
- Seperate test scripts for *descriptors*, *dictionary*, *quantization*, *classification*
- *main.py* for the general pipeline
- Lots of supplementary functions
- *README.md*