BOGAZICI UNIVERSITY

CMPE 537

COMPUTER VISION

Instructor: LALE AKARUN

HOMEWORK 3 - REPORT

IREM ZOG

M.FIKRET YALCINBAS

T.ONUR OZCELIK

MUSTAFA DAGDELEN

JANUARY 2021

1 Abstract

In this study, a basic image classification pipeline was implemented. The main purpose throughout the study was to classify images using local descriptor algorithms such as SIFT, HOG or SURF. The SIFT, HOG, ORB were the algorithms chosen for this study. Also for the other steps, to learn dictionary centers, KMeans, to create image features from learned dictionary centers, VLAD, BoVW and lastly, to classify images, Multi Layer Perceptron, random forest, AdaBoost were selected.

2 Irem Zog

2.1 Data Augmentation

Since the image data set is highly imbalanced, I have used Image Data Generator module of Keras API to generate more images for relatively small classes. The classes that I have worked with are barrel, camera, dalmatian, ferry, lamp, pizza, pyramid, snoopy, soccer ball, stop sign, strawberry, sunflower, water lilly, Windsor chair and yin yang. The summary of data augmentation can be found in Table 2.1.

I have created augmented images using random transformations of the images, including rotation up to the angle 45°, horizontal or vertical shifts, flips, zoom in range 0.9 to 1.0. After creating Image Data Generator using these settings, I have used flow_from_directory function to span each directory (class) separately and generate augmented images belonging to the class. I have used batch size of 32 and iterated 20 times for each class.

Unfortunately, this part is more like trial-and-error for me, rather than a systematic approach. Nonetheless, I have spent too much time generating new images, so I had not enough time to try more systematically.

2.2 Histogram of Oriented Gradients Implementation

To compute local descriptor of images, I have implemented Histogram of Oriented Gradients algorithm. This algorithm initially designed for detecting pedestrians from an image. However, it is useful for other class types.

The algorithm goes as follows:

i Resize the image to 64x128 pixels

ii Divide the image into non-overlapping 8x8 pixel cells

iii For each pixel in the cell:

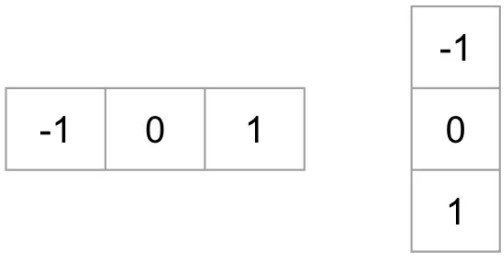


Figure 2.1: Kernels for computing x and y axis gradients.

- Compute gradients along both x and y axis using kernels in the Figure 2.1
- Use the x and y gradients to compute both direction and magnitude of the total gradient (Equation 1 & 2)

$$Magnitude = \sqrt{g_x^2 + g_y^2} \tag{1}$$

$$Orientation = \frac{180}{\pi} * (\arctan(\frac{g_y}{g_x}) \mod \pi) \tag{2}$$

iv Calculate the histogram of gradients. The bins are divided into 20°angles from 0 to 180. There are total of 9 bins. If the gradient angle is in between two bin center, bi-linear interpolation is applied to calculate the gradient contribution to both bins.

v Cells are grouped in 2x2 blocks, the histograms are concatenated and normalized using L2 norm.

vi Block histograms are group into a single histogram, and the image histogram is then normalized to get rid of illumination effects. Then, reshaped into local descriptors again.

Although algorithm suggest 1:2 image ratio (64x128 pixels), I have used 1:1, specifically 128x128 pixel images. The reason is we were trying to detect various types of object, rather than pedestrians. And, the scale change improved my results. For each block, histogram of 1x36 is created. There are 15x15 blocks for 128x128 pixel image. Therefore, 225x36 shaped local descriptors are computed for each image.

2.3 K-Means Dictionary

I used scikit-learn's K-Means clustering to generate visual dictionary. Then, I have generated labels for each 1x36 shaped local descriptor vectors coming from HOG.

I have experimented with cluster sizes of 40, 50 and 100. 50 cluster gave the best result.

2.4 Bag of Visual Words Implementation

After assigning each local descriptor to a cluster, I have computed a histogram for each image and count the number of local descriptor belonging to specific cluster. Then, to normalize, I have divided the histogram to descriptor number in each image, which is 225.

2.5 Under-sampling

When I first tried to train classifier with initial data set (before data augmentation), the classifier had a bias between some classes, which had more images than others. Even data augmentation is not a complete solution to this problem. Therefore, I have decided to combine over-sampling with under-sampling. After data augmentation, I have applied various under-sampling methods to training data, and selected the one which gave the best result.

The methods I have tried are (from imbalanced-learn):

- Near Miss
- Condensed Nearest Neighbour
- Tomek Links
- Random Under Sampler

I have selected to go with Random Under Sampler method. The training image data set before and after data manipulations are shown in Table 2.1.

Class	Initial Training Set	After Data Augmentation	After Under Sampling	Test Set
airplanes	780	780	400	20
anchor	165	165	165	20
background_class	467	467	400	0
barrel	27	270	270	20
camera	30	600	400	20
car_side	103	515	400	20
dalmatian	47	470	400	20
Faces	405	405	405	20
ferry	47	470	400	20
headphone	203	203	203	20
lamp	41	410	410	20
pizza	33	165	165	20
pyramid	37	370	370	20
snoopy	97	485	400	20
soccer_ball	64	640	400	0
stop_sign	44	440	440	20
strawberry	130	520	400	20
sunflower	65	454	400	20
water_lilly	111	555	400	20
windsor_chair	36	360	360	20
yin_yang	40	200	200	20

Table 2.1: Number of class examples in the data set after each operation.

2.6 Random Forest Classifier and Grid Search Cross Validation

After creating the data set using local descriptors and quantization, I have trained Random Forest Classifier. To experiment with hyper-parameters, GridSearchCV library of sci-kit learn is applied, with 5-fold cross validation and f1_macro score as optimization objective. The parameter list for experiments can be found below.

- criterion : [gini, entropy]
- bootstrap : [True, False]
- max_depth : [10, 20, 40, 50, 80, 100, None]

- max_features : [auto, sqrt]
- min_samples_leaf : [1, 2, 4]
- min_samples_split : [2, 5, 10]
- n_estimators : [100, 200, 400, 600, 800]

The best results are achieved with criterion = gini, bootstrap = True, max_depth = 50, max_features = auto, min_samples_leaf = 1, min_samples_split = 5, n_estimators = 800 and Random Forest Classifier then trained with these parameters.

2.7 Performance Evaluation

The accuracy is 0.41 and F_1 macro average is 0.37. The confusion matrix (Figure 2.2) and precision-recall and F_1 metrics (Table 2.2) are shown below.

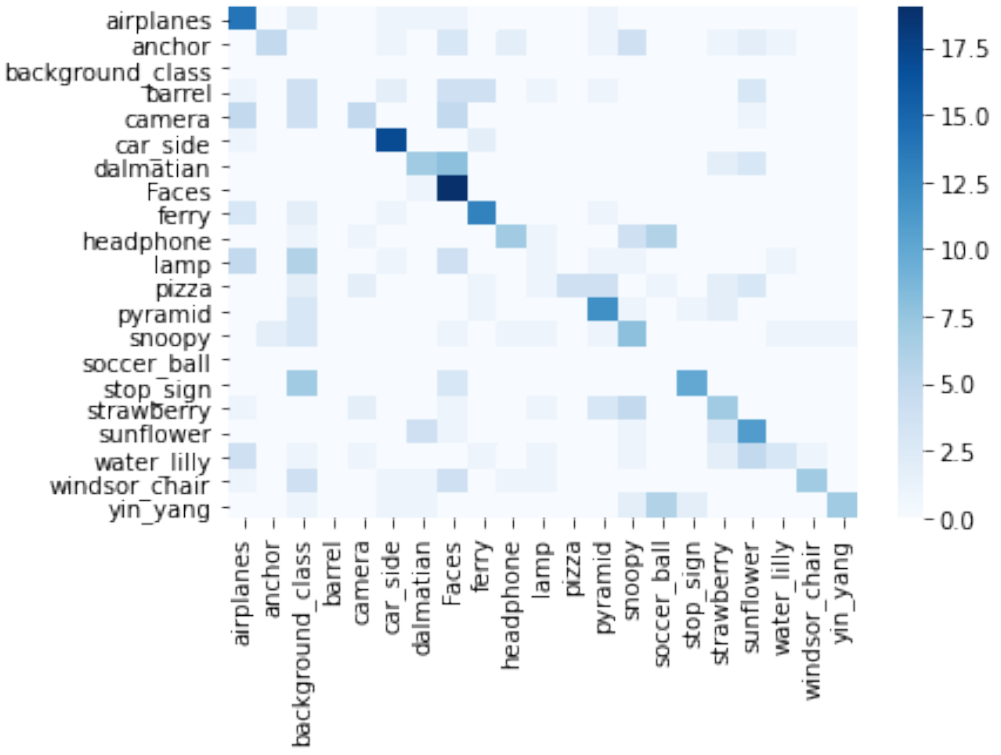


Figure 2.2: Confusion matrix of augmented and under sampled data with random forest classifier.

	precision	recall	f1-score	support
airplanes	0.40	0.70	0.51	20
anchor	0.71	0.25	0.37	20
background_class	0.00	0.00	0.00	0
barrel	0.00	0.00	0.00	20
camera	0.45	0.25	0.32	20
car_side	0.68	0.85	0.76	20
dalmatian	0.47	0.35	0.40	20
Faces	0.35	0.95	0.51	20
ferry	0.59	0.65	0.62	20
headphone	0.64	0.35	0.45	20
lamp	0.12	0.05	0.07	20
pizza	1.00	0.20	0.33	20
pyramid	0.48	0.60	0.53	20
snoopy	0.30	0.40	0.34	20
soccer_ball	0.00	0.00	0.00	0
stop_sign	0.77	0.50	0.61	20
strawberry	0.37	0.35	0.36	20
sunflower	0.39	0.55	0.46	20
water_lilly	0.50	0.15	0.23	20
windsor_chair	0.78	0.35	0.48	20
yin_yang	0.88	0.35	0.50	20
accuracy			0.41	380
macro avg	0.47	0.37	0.37	380
weighted avg	0.52	0.41	0.41	380

Table 2.2: Precision, recall, F_1 score and support for each class and final accuracy and macro averages.

3 M. Fikret Yalcinbas

I chose to use ORB (Oriented Fast and Rotated BRIEF) as the local descriptor for my section. The OpenCV implementation of ORB was used to generate keypoints and descriptors for the training and image sets. Then OpenCV's k-means function was used to cluster the descriptors in order to form the dictionary. I wrote my own implementation of the bag of visual words technique to generate quantized feature vectors, which were then fed into a Random Forest classifier using GridSearchCV (sklearn). After tweaking parameters, the resulting test accuracy score was 17%.

Classifier test score: 0.173

Macro F1 score: 0.143

Class	F1	Precision	Recall
1	0.167	0.0943	0.75
2	0.526	0.555	0.5
3	0.0	0.0	0.0
4	0.0	0.0	0.0
5	0.0	0.0	0.0
6	0.320	0.8	0.2
7	0.0	0.0	0.0
8	0.285	0.187	0.6
9	0.0	0.0	0.0
10	0.266	0.4	0.2
11	0.0	0.0	0.0
12	0.0	0.0	0.0
13	0.0	0.0	0.0
14	0.413	0.666	0.3
15	0.0	0.0	0.0
16	0.249	0.75	0.15
17	0.181	0.666	0.105
18	0.0	0.0	0.0
19	0.0	0.0	0.0
20	0.0	0.0	0.0
21	0.6	0.9	0.45

Table 3.1: Other scores

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	15	0	4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	4	10	4	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	14	0	4	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
5	7	0	6	0	0	0	0	4	0	1	0	0	0	0	0	1	1	0	0	0
6	11	0	5	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	12	0	2	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0
8	6	0	2	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0
9	16	0	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
10	5	1	4	0	0	0	0	4	0	4	0	0	0	1	1	0	0	0	0	0
11	12	0	2	0	0	0	0	4	0	1	0	0	0	1	0	0	0	0	0	0
12	10	0	9	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
13	9	0	7	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
14	1	1	9	0	0	0	0	3	0	0	0	0	0	6	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	9	1	3	0	0	1	0	2	0	0	0	0	0	0	0	3	0	0	0	0
17	9	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
18	10	1	6	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
19	6	0	6	0	0	0	0	5	0	2	0	0	0	0	0	0	0	0	0	0
20	2	0	8	0	0	0	0	9	0	0	0	0	0	1	0	0	0	0	0	0

Table 3.2: Confusion matrix

4 Timoteos Onur Özçelik

4.1 Introduction

In this section, to classify Caltech 21 images, firstly feature selection operation was done with k-means clustering algorithm over descriptors obtained from SIFT interest point detector. Then, **implemented** Bag-of-Visual-Words (BoVW) was applied to construct features for each image through quantization over cluster centers discovered by k-means.

4.2 SIFT

Besides being rotation-invariant, SIFT is also a scale-invariant feature detector. It discovers key points in images with their descriptor vectors length of 128.

Class	#Images	#Descriptor
Faces	405	275715
airplane	781	222254
anchor	165	104953
background	467	502365
barrel	27	11598
camera	30	10864
car _side	103	40779
dalmatian	47	28227
ferry	47	19196
headphone	203	40329
lamp	41	10627
pizza	33	29065
pyramid	37	10270
snoopy	97	32710
soccer_ball	64	19183
stop_sign	44	13893
strawberry	130	149736
sunflower	65	30699
water_lilly	111	37812
windsor _chair	36	12514
yin_yang	40	9115

Table 4.1: Number of Images and Descriptors per Class in train set

It must be stated that there is an extreme imbalance in the number of images and descriptors between classes before and after SIFT.

4.3 Clustering with k-means

Over sampled descriptors on train images, k-means clustering was realized to select features will be fed to the classifier. The number of clusters was determined as 100, assuming approximately 5 features per class. However, the imbalance in the data is likely to be reflected here.

4.4 Quantization with BoVW

BoVW technique simply computes histogram vector of images according to the cluster centers provided with k-means. In other word, each image will have a feature vector length of 100, and each feature will be the number of assigned descriptors to the respective cluster center. Nevertheless, since each image can have different number of descriptors, normalization was inevitable. Therefore, each image feature vector was normalized with division by the number of descriptor computed.

4.5 AdaBoost Classifier

AdaBoost is an classifier which use ensemble learning approach with boosting. It begins with a "weak" (or base) learner and then each following classifier trees aim to reduce the bias (increase the accuracy) of the previous models by paying more attention to the misclassified inputs by the previous trees. It has some hyper-parameters relevant to the success and the simplicity of the model. To establish these purposes, number of estimators and learning rate values were chosen 500 and 0.05 respectively.

4.6 Results

AdaBoost classifier was trained and tested on Caltech 21 Dataset. The results are shown in Table 5.1 .

4.6.1 Performance Metrics

	Train			Test		
	precision	recall	f1-score	precision	recall	f1-score
accuracy			0.48			0.12
macro avg	0.43	0.19	0.20	0.19	0.10	0.09
weighted avg	0.48	0.48	0.42	0.21	0.12	0.10

Table 4.2: Overall Results for Train and Test

It is immediately obvious that there is a huge difference (variance) in accuracy between train and test results (underfitting). The imbalance between classes might be the reason behind high variance. Also, another reason might be due to background class which is dominant in train set but doesn't have any example in test set. Hence, AdaBoost classifier was trained with upsampling with replacement over inputs to reduce the variance between train and test sets. At the end of upsampling, the model had same number of inputs for each class.

	Train			Test		
	precision	recall	f1-score	precision	recall	f1-score
accuracy			0.42			0.21
macro avg	0.54	0.42	0.41	0.35	0.19	0.19
weighted avg	0.54	0.42	0.41	0.38	0.21	0.21

Table 4.3: Overall Results with Upsampling Strategy for Train and Test

4.6.2 Confusion Matrix

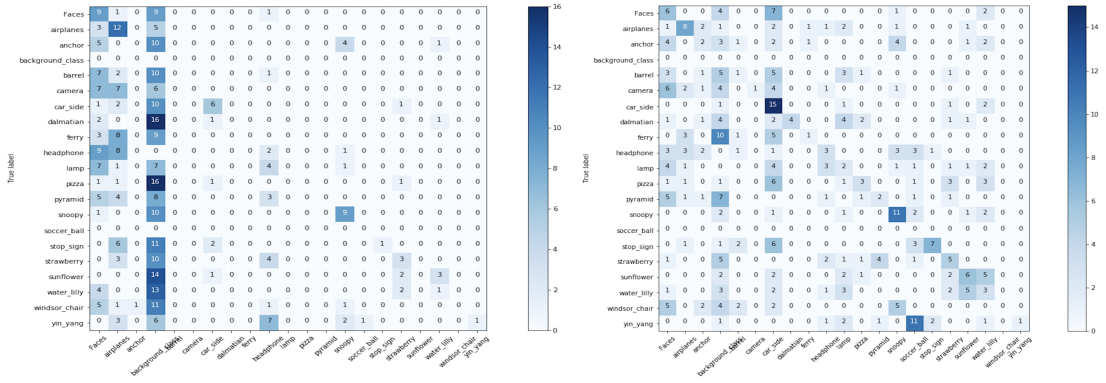


Figure 4.1: Without upsampling (left) and with upsampling (right) AdaBoost Results

Imbalance problem becomes more clear in confusion matrix (see Figure 5.1). AdaBoost classifier without upsampling tends to classify each input as background class in test even though there is no background class image in test set. On the other hand, although the problem couldn't be solved completely with the upsampling, the distribution over classes is relatively more balanced (particularly compare results for first 4 classes). Besides, upsampling strategy caused some correct classifications to be misclassified contrary to what was expected (see camera class). The results produced by the classifiers demonstrates the low capacity of the presented model(s).

5 Mustafa Dağdelen

5.1 Introduction

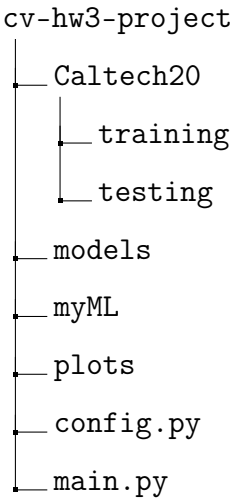
In this image classification pipeline you can find both personally implemented modules such as KMeans, GMM, MLP or VLAD and third party implemented modules. Third party modules were used at some points due to excessive execution time. Also you can easily switch between implemented and third party modules with command line arguments. And lastly, the pretrained KMeans model has been uploaded to Google Drive so that you can easily run the whole system from end to end.

5.2 Code Explanation

To achieve the metric in this report simply run the script with default parameters *python3 dagdelen-mustafa-cv-hw3.py*.

5.2.1 Folder Structure

- The project folder structure can be seen at the right hand side. Our training and test data is in *Caltech20* folder. In *models* folder we kept our pretrained models and in *myML* folder personally implemented algorithms can be found.
- In *config.py* we set command line parameters.
- Lastly, in *main.py* file we run all pipeline end to end.



5.2.2 Command Line Parameters

- `train_path` Defines training data path.
- `test_path` Defines test data path.

- *subsampling* Set if you want to sub-sample train data.
- *cluster_option* Defines the method which will create our dictionary or simply clustering method.
- *cluster_size* Defines our feature vector size.
- *n_epoch* MLP training epoch number.
- *learning_rate* MLP learning rate.
- *batchsize* Training batch size.

5.3 Methodology

To classify images, the created pipeline includes SIFT, KMeans, VLAD and MLP, respectively.

5.3.1 SIFT step

- *"In our group SIFT was selected by Timoteos Özçelik and I selected SURF but according to the OpenCV documentation SURF is a patented algorithm and the only way to use it compiling OpenCV from source code. So that I started to implement my image classification with SIFT. I thought if I had some time left I would compile SURF but I don't have enough time left and I shared with you the SIFT implemented pipeline. Thank you for your understanding."*
- At this step CV2 implemented builtin function was used.
- For each image we got descriptors with shape (x, 128).
- After getting the descriptors of all the images, it became the shape of our training data (882562, 128) and for the test it was (112093, 128).
- You can see how many descriptors each class has on class basis in the table 6.1. So, the imbalance in the data set is clearly seen when looking at the table 6.1. To tackle with this issue we sample randomly examples from the data.

Classes	Training Im.	Test Im.	Training Desc.	Test Desc.	S. Training Im.	S. Test Desc.	S. Train Desc.	S. Test Desc.
headphone	203	20	34578	4770	203	20	34578	4770
lamp	41	20	9515	5064	41	20	9515	5064
strawberry	130	20	50438	4748	130	20	50438	4748
waterlilly	111	20	32283	5948	111	20	32283	5948
dalmatian	47	20	17603	7346	47	20	17603	7346
airplanes	780	20	202712	5646	271	20	67998	5646
soccerball	64	0	15666	0	64	0	15666	0
pyramid	37	20	8776	5172	37	20	8776	5172
camera	30	20	9611	5646	30	20	9611	5646
barrel	27	20	8933	7375	27	20	8933	7375
yinyang	40	20	6801	3195	40	20	6801	3195
carside	103	20	35739	7288	103	20	35739	7288
ferry	47	20	16239	6153	47	20	16239	6153
snoopy	97	20	27891	4957	97	20	27891	4957
anchor	165	20	44486	5658	165	20	44486	5658
windsorchair	36	20	11150	5626	36	20	11150	5626
pizza	33	20	13040	7822	33	20	13040	7822
stopsign	44	20	10570	4952	44	20	10570	4952
backgroundclass	466	-	150342	-	206	-	67995	-
Faces	405	20	152891	7614	179	20	67912	7614
sunflower	65	20	23298	7113	65	20	23298	7113

Table 5.1: Quantitative information about the data set

5.3.2 Random sampling

- Our sampling method was based on a rule based approach. First, we got whole descriptors from training set. Then, we select the class which has minimum number of descriptors. In our data set, it was *yinyang* class with 6801 descriptors. We set each class descriptors to not exceed 10 times the class which has minimum number of descriptors.
- You can find quantitative information about sampled data set in table 6.1.
- Also in experiments section we made our experiments with both sampled and original data set.

5.3.3 Learning dictionary centers

- To learn dictionary centers, KMeans was used.

- Both personally implemented and sci-kit modules can be found in the code.
- Also, to decrease calculation time, pretrained KMeans models were putted under the models folder.

5.3.4 VLAD

- VLAD is a feature quantization method. In this method, firstly we calculate the nearest cluster center of the given descriptors. Then we simply collect these result according to the assigned classes. Lastly we normalize our vector with L2 Norm.
- In this process the only third party function was scipy's cdist function which calculates euclidean distance between each pair of the two collections of inputs.

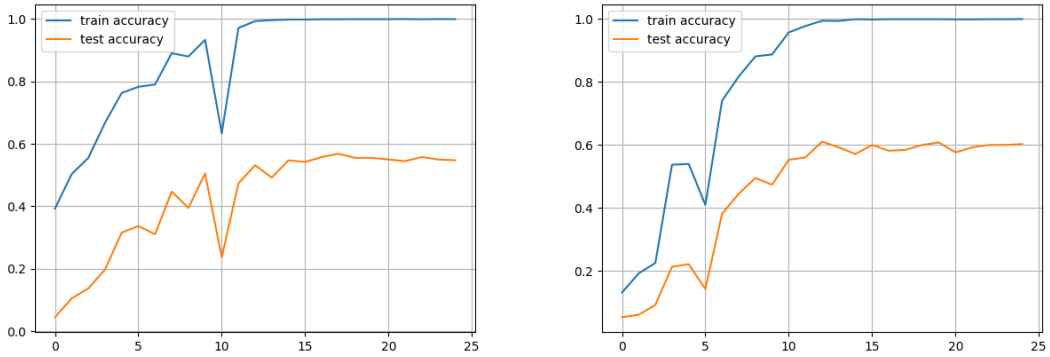
5.3.5 Multi Layer Perceptron

- For the classifier, the MLP was selected. And to be more simple and understandable, the implemented methods has been likened to the PyTorch's *nn.Module*

5.4 Experiments

- Two different experimental results which come from sub-sampled and normal data set, are shown in this section.
- The same metrics were used in both experiments.
- The classifier run 25 epoch with 0.4 learning rate.

5.4.1 Accuracy

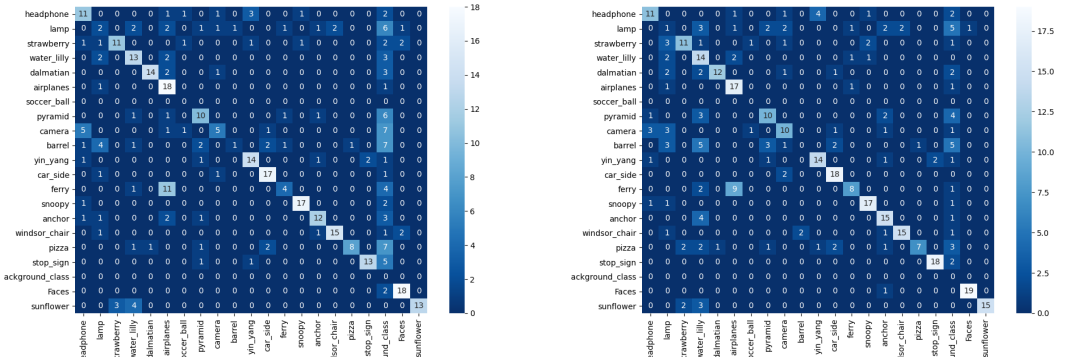


(a) Model with original data (b) Model with sub-sampled data

Figure 5.1: Comparison of two experimental results in terms of accuracy

- Here, we can easily see that the accuracy of the model was increased with our sub-sampling method.
- While the accuracy of the model was 0.568, this accuracy increased to 0.611 with the sub-sampling process.

5.4.2 Confusion Matrices



(a) Confusion matrix with original data (b) Confusion matrix with sub-sampled data

Figure 5.2: Confusion matrices of the experiments

- The incorrect classifications, especially the background class, were decreased dramatically with the sub-sampling process.

5.4.3 Other metrics

	Model with original data			Model with sub-sampled data		
	precision	recall	f1-score	precision	recall	f1-score
accuracy			0.568			0.611
macro avg	0.629	0.514	0.542	0.604	0.552	0.562
weighted avg	0.695	0.568	0.599	0.668	0.611	0.621

Table 5.2: Overall model metrics

	Model with original data			Model with sub-sampled data		
	precision	recall	f1-score	precision	recall	f1-score
headphone	0.524	0.550	0.537	0.647	0.550	0.595
lamp	0.154	0.100	0.121	0.059	0.050	0.054
strawberry	0.786	0.550	0.647	0.733	0.550	0.629
waterlilly	0.565	0.650	0.605	0.359	0.700	0.475
dalmatian	0.933	0.700	0.800	0.923	0.600	0.727
airplanes	0.450	0.900	0.600	0.567	0.850	0.680
soccerball	0.000	0.000	0.000	0.000	0.000	0.000
pyramid	0.588	0.500	0.541	0.588	0.500	0.541
camera	0.556	0.250	0.345	0.556	0.500	0.526
barrel	0.500	0.050	0.091	0.000	0.000	0.000
yinyang	0.737	0.700	0.718	0.737	0.700	0.718
carside	0.773	0.850	0.810	0.750	0.900	0.818
ferry	0.571	0.200	0.296	0.727	0.400	0.516
snoopy	0.895	0.850	0.872	0.810	0.850	0.829
anchor	0.750	0.600	0.667	0.625	0.750	0.682
windsorchair	0.882	0.750	0.811	0.882	0.750	0.811
pizza	0.889	0.400	0.552	0.875	0.350	0.500
stopsign	0.867	0.650	0.743	0.900	0.900	0.900
backgroundclass	0.000	0.000	0.000	0.000	0.000	0.000
Faces	0.783	0.900	0.837	0.950	0.950	0.950
sunflower	1.000	0.650	0.788	1.000	0.750	0.857

Table 5.3: Per-class model metrics

6 Comparison

Name	Descriptor	Quantizer	Classifier	F1 Score
Mustafa	SIFT	VLAD	MLP	0.61
Irem	HOG	BoVW	Random Forest	0.37
Timoteos	SIFT	BoVW	AdaBoost	0.21
Fikret	ORB	BoVW	Random Forest	0.17

Table 6.1: Score comparison

We find that ORB is inferior to HOG as a descriptor in this context.

We also find that VLAD performs better than BoVW as a quantizer. As VLAD is an extension of BoVW, this result is reasonable. Because VLAD keeps the sum of the residuals of the descriptors assigned to the cluster and the centroid of the cluster instead of counting the number of descriptors associated with clusters as in BoVW.

Pred: lamp

Pred: waterlilly

Pred: carside

Pred: background

Pred: windsor_chair

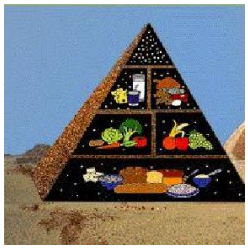
Pred: background

Pred: water_lilly

Pred: airplanes

Pred: background

Pred: background



Pred: background

Pred: headphone

Pred: background

Pred: headphone

Pred: water_lilly

Pred: background

Pred: Faces

Pred: headphone

Figure 6.1: Some Misclassified Examples