Assignment 3 - Local Descriptor Based Image Classification

Introduction

In this assignment, we worked as a group to classify a set of images using various methods and experimented with combinations of these methods. The image prediction pipeline consists of 4 steps:

- Feature Extraction
- Finding Dictionary Centers
- Feature Quantization
- Classification

We have implemented a number of methods to achieve these goals. We, then, combined these methods to create full pipelines that predict image tags. The "Caltech20" dataset given with the assignment is used for training and testing.

Preprocessing

As it has been suggested, we have resized the images to lower the computational cost.

Feature Extraction

ORB (OpenCV implementation)

Files: feature_extraction/orb.py
Author: Mahmut Karaca

I have written a wrapper for the ORB feature extraction method implemented in the OpenCV library. As the homework requires, I have used the Python bindings for the library. The wrapper is simply a class with a single static method which takes image path as parameter and returns the key points and descriptors as a tuple.

SIFT (OpenCV implementation)

Files: feature_extraction/sift.py
Author: İpek Erdoğan

I have used the OpenCV implementation of Scale-Invariant Feature Transform (SIFT) in my pipeline to extract the descriptors from the images. SIFT extracts different numbers of descriptors from each image, which are 128-dimensional feature vectors. I restricted the number of descriptors per image as 100, to lower the computational cost. It gives 1611904 descriptors from the Caltech20 training dataset, if I don't restrict the number.

HU Moments Descriptor (Own Implementation)

Files: hu.py
Author: M.Yasin Adıyaman

I have used Hu moments to describe image patches. There are 7 Hu moments for each image patch which is resized to (50,50). Since Hu moments are applicable to silhouette images, I generated mask images first and then calculated Hu moments for mask(binary) images. For each image patch, I firstly convert color space to HSV to extract hue information. Then I convolve upon hue dimension with different padding and stride values to generate 19 masks. Therefore, I could reach 7*19=126 dimension descriptor vector for a patch. I added 2 extra elements (0) to the vector and generated a 128-dimension descriptor vector. I repeated this process for different 135 patches which resulted from 2D convolution over image with different padding (this time stride and padding values are equal). Then, for each image the descriptor object generates a 135x128 descriptor.

Histogram Of Oriented Gradients (Own Implementation)

Files: feature_extraction/hog.py
Author: Ahmet Emin Yetkin

Histogram of gradients (HOG), is a popular feature description algorithm that basically considers orientation of gradients for describing structures in images. The main idea of HOG descriptor is local structures within an image can be described by the distribution of intensity gradients or edge directions.
In this project, I followed these steps to build HOG descriptor:
I converted variable type to float32 and normalized the input image values between 0 to 255. I calculated gradients along x and y directions by using sobel edge detector. I used 3x3 kernels. After finding gradients, next step was calculating magnitudes and orientations. Orientations were normalized between 0 to 180 degrees since gradients would be unsigned and direction info is redundant. After, I divided the image into many 8x8 pixel cells. In each cell, the magnitude values of these 64 cells were binned and cumulatively added into 9 buckets of unsigned direction.Then I slided a 2x2 cells (thus 16x16 pixels) block across the image. In each block region, 4 histograms of 4 cells are concatenated into one-dimensional vector of 36 values and then normalized to have an unit weight. The final HOG feature vector is the concatenation of all the block vectors.

Local Binary Pattern (Own Implementation)

Files: Linear-Binary-Pattern-Feature-extraction/lbp.py

Author: Ahmet Karagöz

It was firstly described in 1994 and has since been found to be a powerful feature for texture classification. It is commonly used for face detection, face recognition or object detection tasks. Local Binary Pattern algorithm uses four different parameters. First one is radius which is used to build the circular local binary pattern and represents the radius around the central pixel It is set to 1 here. Second one is neighbors, and it is the number of sample points to build the circular local binary pattern. It is set to eight here (higher number of

sample points need more computational cost). Others are the number of cells in the vertical and horizontal axis. Workflow of this implementation consists of converting images into grayscale, applying an lbp filter to each pixel and find lpb image, and finding the histogram of the image. There are some functions in the lbp.py file to do these works. Output descriptors are 256x1 for each image.

Finding Dictionary Centers

K-Means (Own Implementation)

Files: clustering/kmeans.py
Author: İpek Erdoğan

To determine the codewords (dictionary centers), we needed a clustering algorithm. By processing all the descriptors and extracting some center values (vectors) from them, we create some benchmarks for our feature vectors. So I implemented the K-Means algorithm. It starts with random initial K centers and iteratively updates the center values, with the average values of cluster members. "Cluster membership" determined according to the euclidean distance. A data vector belongs to the cluster whose center is closest (minimum euclidean distance) to the data vector. For this problem, the important output of K-Means function is the list of centroids. We give the centers list to Mahmut's Bag of Visual Words implementation, and this center values will be the vocabulary.

Here, while determining the "K", it's important to understand its meaning. This number should at least be equal to the class number. But this is also not enough. Feature vectors should represent their class' specific features if it's possible to be distinguished from other class' feature vectors. If we keep this number too low (i.e. 20), our feature vectors will be too generalized for being classified correctly and we can not expect a good accuracy from a classifier which we fed with these feature vectors.

Yet, if we keep this number too high (i.e. 1000), our feature vectors will be too specific for being classified correctly. This situation would cause overfitting. Also, we should consider the computational cost. In the light of these constraints, we did our experiments with different number of K's to see these effects on the results.

Feature Quantization

Bag of Visual Words (Own Implementation)

Files: feature_quantization/bag_of_visual_words.py
Author: Mahmut Karaca

Bag of Visual Words were implemented as a single function. It simply generates a histogram using the image features and dictionary centers as input. Euclidean distance metric is used for calculating the nearest center.

Classification

Random Forest (Sklearn implementation)

Author: İpek Erdoğan

As a classifier, I decided to use Random Forest. Random Forest classifier consists of different Decision Trees. It classifies inputs by taking the average of the results of these decision trees or deciding on the dominant choice. Normally, with a quick research, you can see there are more examples which use SIFT with SVM than the ones which use with Random Forest. But there are some works [1][2] on comparison between the Random Forest and SVM, not only about accuracy but also about computation cost, which took my attention. I wanted to see the results with my experiments. That's why I chose Random Forest.

Adaboost Classifier (Sklearn Implementation)

Author: Ahmet Karagöz

I choose an adaboost classifier, in other words adaptive boosting classifier. AdaBoost is one of the first boosting algorithms to be adapted in solving practices. Adaboost helps us combine multiple classifiers into a single classifier. I choose adaboost because it is fast and simple.

Evaluation

We have implemented different pipelines individually. In this part of the report, we will first look at the results of each pipeline separately and at the end, we will compare the average f1 scores of the models. We ran all of the pipelines for different number of clusters (K) which are 20,50,100.

1.SIFT-K-Means (Ipek's Implementation) - Bag of Words(Mahmut's Implementation) - Random Forest

Author: İpek Erdoğan

K=20

Class	Precision	Recall	F1 Score
Faces	0.19	0.55	0.29
airplanes	0.11	0.75	0.20
anchor	0.46	0.30	0.36
background_class	0.00	1.00	0.00
barrel	1.00	0.00	0.00
camera	1.00	0.00	0.00
car_side	0.00	0.00	0.00

dalmatian	0.00	0.00	0.00
ferry	1.00	0.00	0.00
headphone	0.19	0.35	0.25
lamp	1.00	0.00	0.00
pizza	0.50	0.05	0.09
pyramid	1.00	0.00	0.00
snoopy	0.33	0.05	0.09
soccer_ball	0.00	1.00	0.00
stop_sign	1.00	0.00	0.00
strawberry	0.12	0.05	0.07
sunflower	0.56	0.25	0.34
water_lilly	0.43	0.15	0.22
windsor_chair	1.00	0.10	0.18
yin_yang	0.67	0.10	0.17
Accuracy			0.14
Macro Average	0.50	0.22	0.11
Weighted Average	0.56	0.14	0.12

K=50
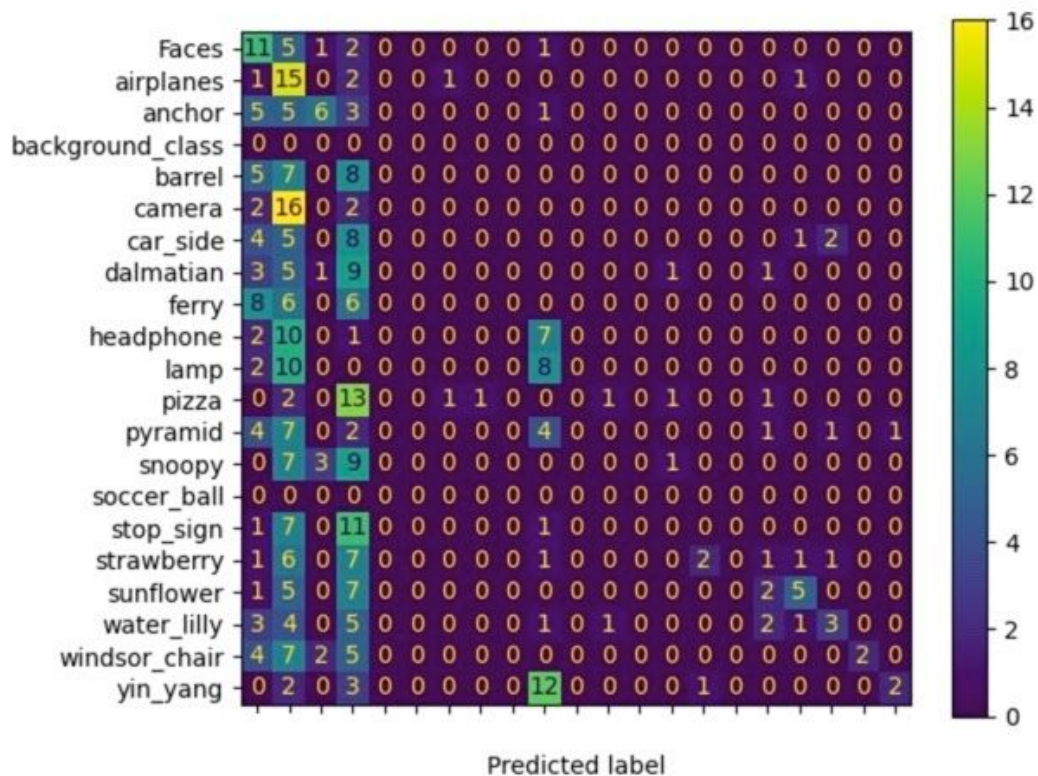
Class	Precision	Recall	F1 Score
Faces	0.15	0.35	0.21
airplanes	0.11	0.85	0.20
anchor	0.45	0.25	0.32
barrel	0.00	1.00	0.00
camera	1.00	0.00	0.00
car_side	1.00	0.00	0.00
dalmatian	0.33	0.05	0.09
ferry	0.50	0.05	0.09
headphone	1.00	0.00	0.00
lamp	0.22	0.30	0.26
pizza	1.00	0.00	0.00
pyramid	1.00	0.00	0.00
snoopy	1.00	0.00	0.00
soccer_ball	1.00	0.25	0.40
stop_sign	0.00	0.00	0.00
strawberry	0.36	0.20	0.26
sunflower	0.50	0.20	0.29
water_lilly	0.00	0.00	0.00
windsor_chair	1.00	0.35	0.52
yin_yang	0.91	0.50	0.65
Accuracy			0.18
Macro Average	0.58	0.22	0.16
Weighted Average	0.61	0.18	0.17

Predicted label
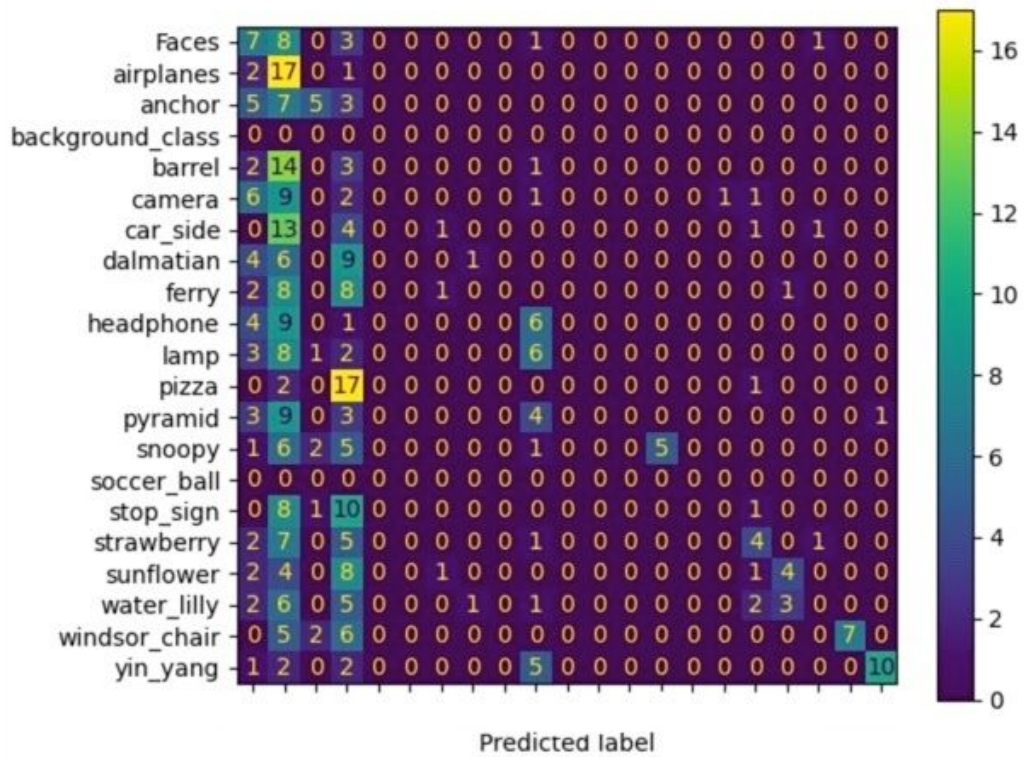
K=100

Class	Precision	Recall	F1 Score
Faces	0.11	0.20	0.14
airplanes	0.10	0.95	0.19
anchor	0.64	0.45	0.53
barrel	0.00	1.00	0.00
camera	1.00	0.00	0.00
car_side	1.00	0.00	0.00
dalmatian	1.00	0.05	0.10
ferry	1.00	0.00	0.00
headphone	1.00	0.00	0.00
lamp	0.29	0.25	0.27
pizza	1.00	0.00	0.00
pyramid	1.00	0.05	0.10
snoopy	1.00	0.00	0.00
soccer_ball	1.00	0.25	0.40
stop_sign	1.00	0.00	0.00

strawberry	0.00	0.00	0.00
sunflower	0.86	0.30	0.44
water_lilly	1.00	0.00	0.00
windsor_chair	1.00	0.35	0.52
yin_yang	0.93	0.70	0.80
Accuracy			0.19
Macro Average	0.75	0.23	0.17
Weighted Average	0.79	0.19	0.18
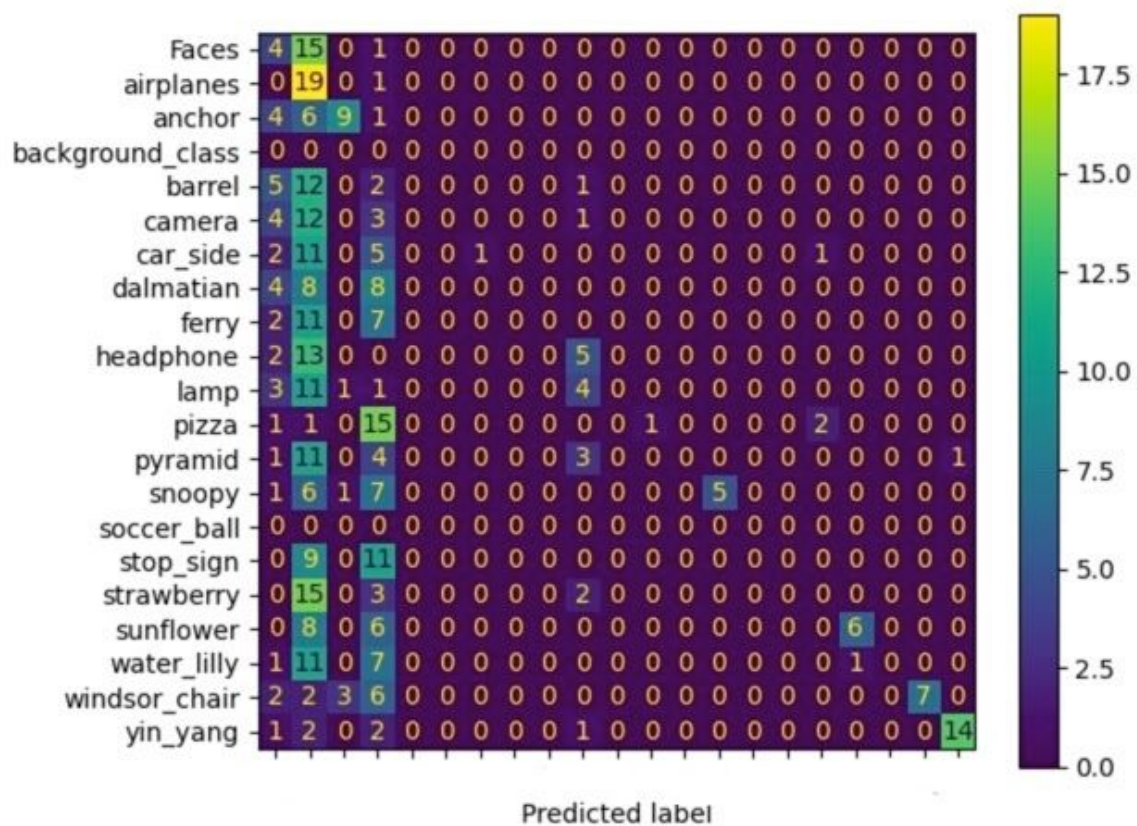
2.ORB-K-Means(Ipek's Implementation)-Bag of Words(Mahmut's Implementation)-Decision Tree

Author: Mahmut Karaca

By observing the results of classification, we can conclude that decision tree classification is practically useless in our use case. We speculate that a larger dataset might produce better results.

K=20

Class	Precision	Recall	F1 Score
Faces	0.04	0.1	0.06
airplanes	0.07	0.3	0.12
anchor	0.21	0.25	0.23
background_class	0	1	0
barrel	0	0	0
camera	0	0	0
car_side	0.19	0.15	0.17
dalmatian	0	0	0
ferry	0	0	0
headphone	0.07	0.1	0.08
lamp	0	0	0
pizza	0	0	0
pyramid	0.43	0.15	0.22
snoopy	0.25	0.15	0.19
soccer_ball	0	1	0
stop_sign	0.14	0.05	0.07
strawberry	0.24	0.25	0.24
sunflower	0	0	0
water_lilly	0.05	0.05	0.05
windsor_chair	0.25	0.05	0.08
yin_yang	0.2	0.05	0.08

	Precision	Recall	F1 Score
accuracy			0.09
macro_avg	0.1	0.17	0.08
weighted_avg	0.11	0.09	0.08

K=50

Class	Precision	Recall	F1 Score
Faces	0.02	0.05	0.02
airplanes	0.07	0.35	0.12
anchor	0.19	0.2	0.2
background_class	0	1	0
barrel	0	0	0
camera	0	0	0
car_side	0.11	0.05	0.07
dalmatian	0	0	0
ferry	0	0	0
headphone	0.04	0.05	0.04
lamp	0	0	0
pizza	0.2	0.05	0.08
pyramid	0.33	0.05	0.09
snoopy	0.17	0.1	0.12
soccer_ball	0	1	0
stop_sign	0	0	0
strawberry	0	0	0
sunflower	0	0	0
water_lilly	0.06	0.05	0.05
windsor_chair	0.4	0.1	0.16

Class	Precision	Recall	F1 Score
yin_yang	0.43	0.15	0.22
accuracy			0.06
macro_avg	0.1	0.15	0.06
weighted_avg	0.11	0.06	0.06

K=100

Class	Precision	Recall	F1 Score
Faces	0.11	0.3	0.16
airplanes	0.06	0.25	0.09
anchor	0.12	0.15	0.14
background_class	0	1	0
barrel	0	0	0
camera	0	0	0
car_side	0.22	0.1	0.14
dalmatian	0	0	0
ferry	0.14	0.05	0.07
headphone	0.03	0.05	0.04
lamp	0	0	0
pizza	0	0	0
pyramid	0	0	0
snoopy	0.57	0.2	0.3
soccer_ball	0	1	0
stop_sign	0.29	0.1	0.15
strawberry	0.08	0.05	0.06
sunflower	0	0	0
water_lilly	0.14	0.15	0.15

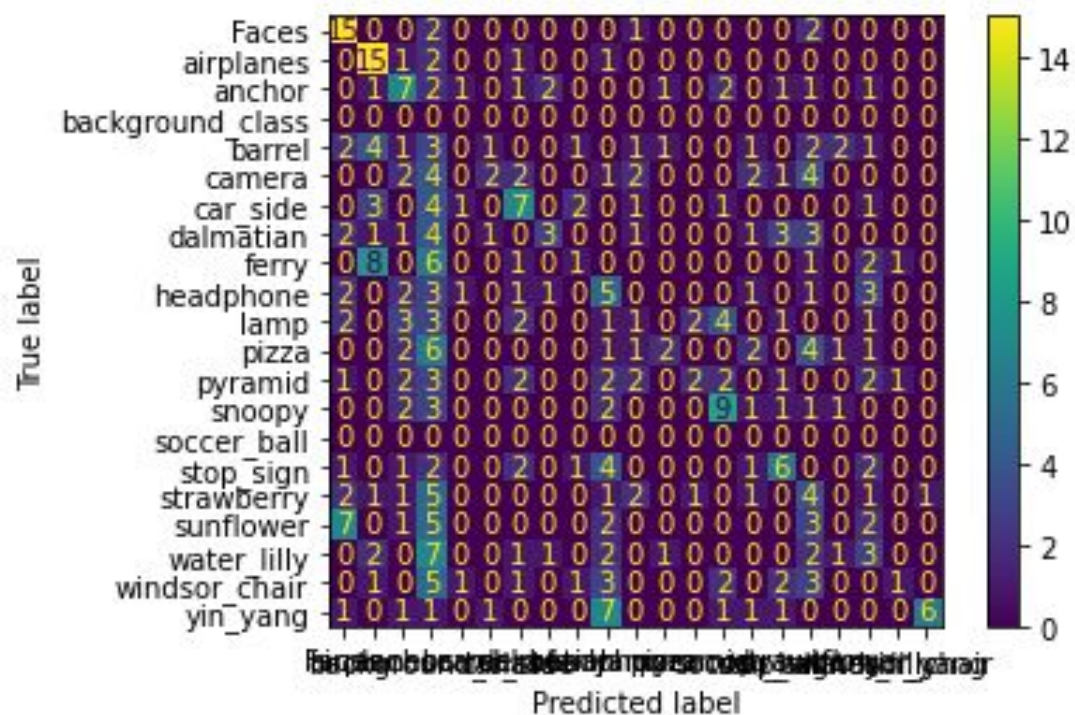|	precision	recall	f1-score
windsor_chair	0	0	0
yin_yang	0	0	0
accuracy			0.07
macro_avg	0.08	0.16	0.06
weighted_avg	0.09	0.07	0.07

3.HOG(Ahmet Emin's Implementation)-K-Means(Ipek's Implementation)-Bag of Words(Mahmut's Implementation)-SVM

Author: Ahmet Emin Yetkin

K=20

	precision	recall	f1-score	support
Faces	0.43	0.75	0.55	20
airplanes	0.42	0.75	0.54	20
anchor	0.26	0.35	0.3	20
background	0	1	0	0
barrel	0	0	0	20
camera	0.4	0.1	0.16	20
car_side	0.33	0.35	0.34	20
dalmatian	0.43	0.15	0.22	20
ferry	0.17	0.05	0.08	20
headphone	0.16	0.25	0.19	20
lamp	0.08	0.05	0.06	20
pizza	0.4	0.1	0.16	20
pyramid	0.4	0.1	0.16	20
snoopy	0.43	0.45	0.44	20

	precision	recall	f1-score	support
soccer_ball	0	1	0	0
stop_sign	0.35	0.3	0.32	20
strawberry	0.13	0.2	0.16	20
sunflower	0	0	0	20
water_lilly	0.15	0.15	0.15	20
windsor_chair	0.33	0.05	0.09	20
yin_yang	0.86	0.3	0.44	20
accuracy			**0.23**	380
macro avg	0.27	0.31	0.21	380
weighted avg	0.3	0.23	0.23	380

K=50

	precision	recall	f1-score	support

Faces	0.33	0.7	0.45	20
airplanes	0.36	0.8	0.5	20
anchor	0.41	0.45	0.43	20
background	0	1	0	0
barrel	0.17	0.05	0.08	20
camera	0	0	0	20
car_side	0.43	0.3	0.35	20
dalmatian	0.25	0.1	0.14	20
ferry	0.33	0.15	0.21	20
headphone	0.2	0.3	0.24	20
lamp	0.11	0.05	0.07	20
pizza	0.38	0.15	0.21	20
pyramid	0.18	0.1	0.13	20
snoopy	0.23	0.25	0.24	20
soccer_ball	0	1	0	0
stop_sign	0.62	0.4	0.48	20
strawberry	0.08	0.1	0.09	20
sunflower	0.09	0.05	0.06	20
water_lilly	0.19	0.2	0.2	20
windsor_chair	0.25	0.05	0.08	20
yin_yang	0.71	0.25	0.37	20
accuracy			**0.23**	380
macro avg	0.25	0.31	0.21	380
weighted avg	0.28	0.23	0.23	380
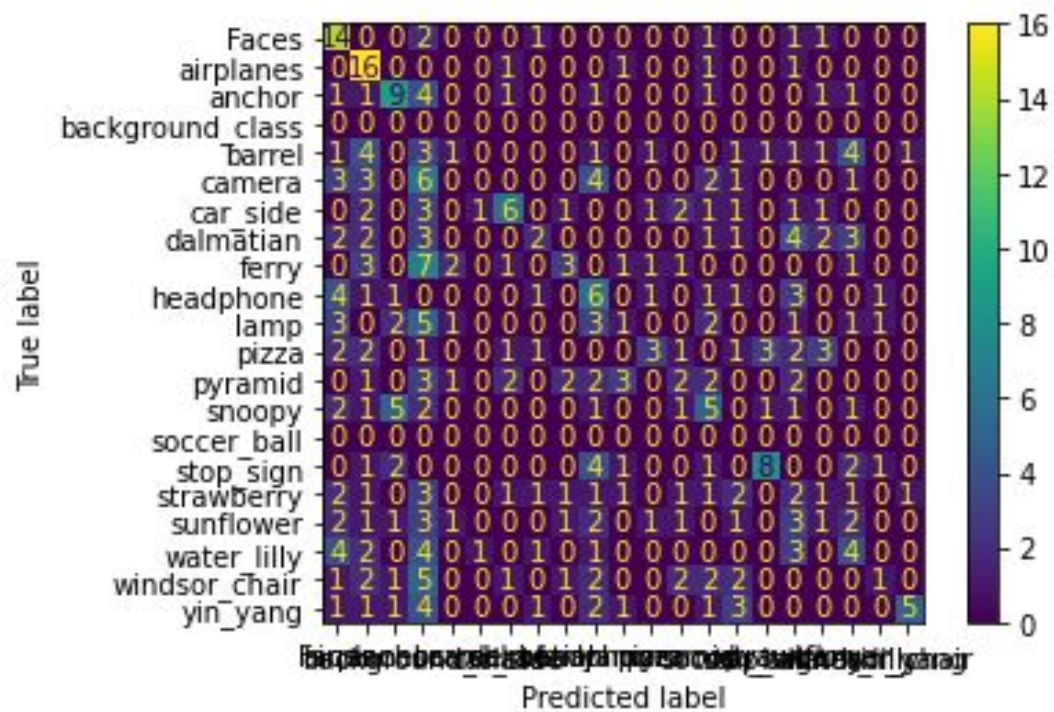
K=100

	precision	recall	f1-score	support
Faces	0.37	0.65	0.47	20
airplanes	0.36	0.7	0.47	20
anchor	0.32	0.5	0.39	20
background	0	1	0	0
barrel	0	0	0	20
camera	0	0	0	20
car_side	0.38	0.5	0.43	20
dalmatian	0.29	0.1	0.15	20
ferry	0.14	0.05	0.07	20
headphone	0.23	0.3	0.26	20
lamp	0.22	0.1	0.14	20

pizza	0.29	0.1	0.15	20
pyramid	0	0	0	20
snoopy	0.31	0.2	0.24	20
soccer_ball	0	1	0	0
stop_sign	0.46	0.3	0.36	20
strawberry	0.05	0.05	0.05	20
sunflower	0.21	0.15	0.18	20
water_lilly	0.11	0.1	0.1	20
windsor_chair	0.33	0.05	0.09	20
yin_yang	0.71	0.25	0.37	20
accuracy			**0.22**	380
macro avg	0.23	0.29	0.19	380
weighted avg	0.25	0.22	0.21	380
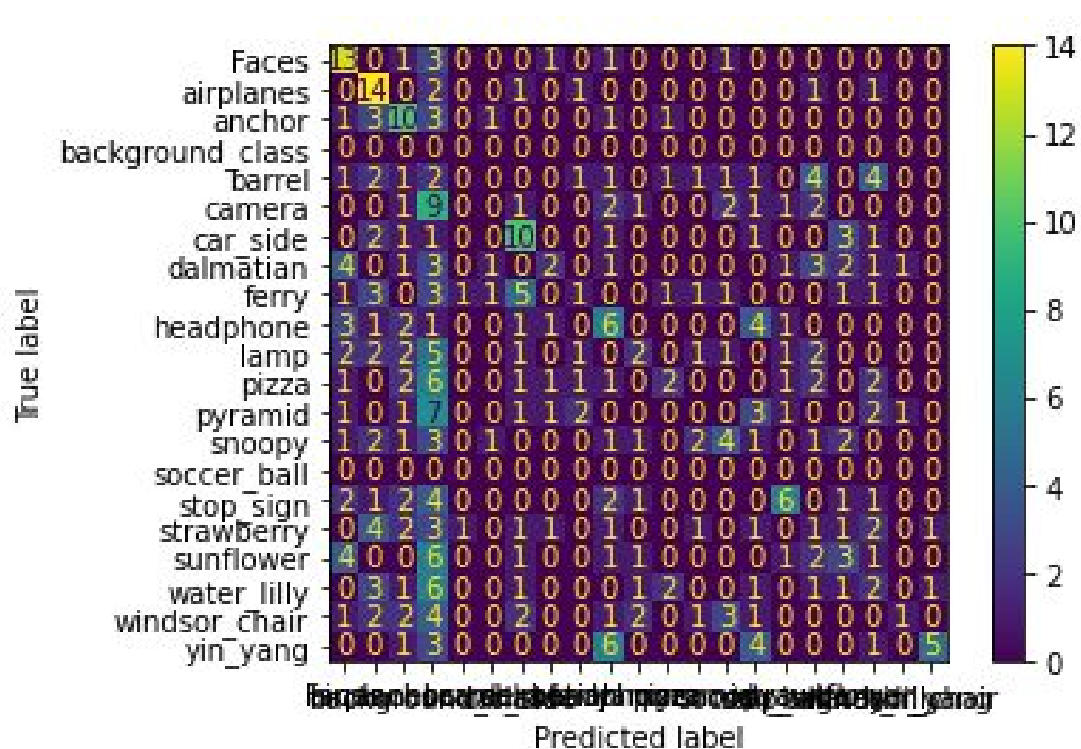
4.LBP(Ahmet's Implementation)-K-Means(Ipek's Implementation)-Bag of Words(Mahmut's Implementation)-Adaboost
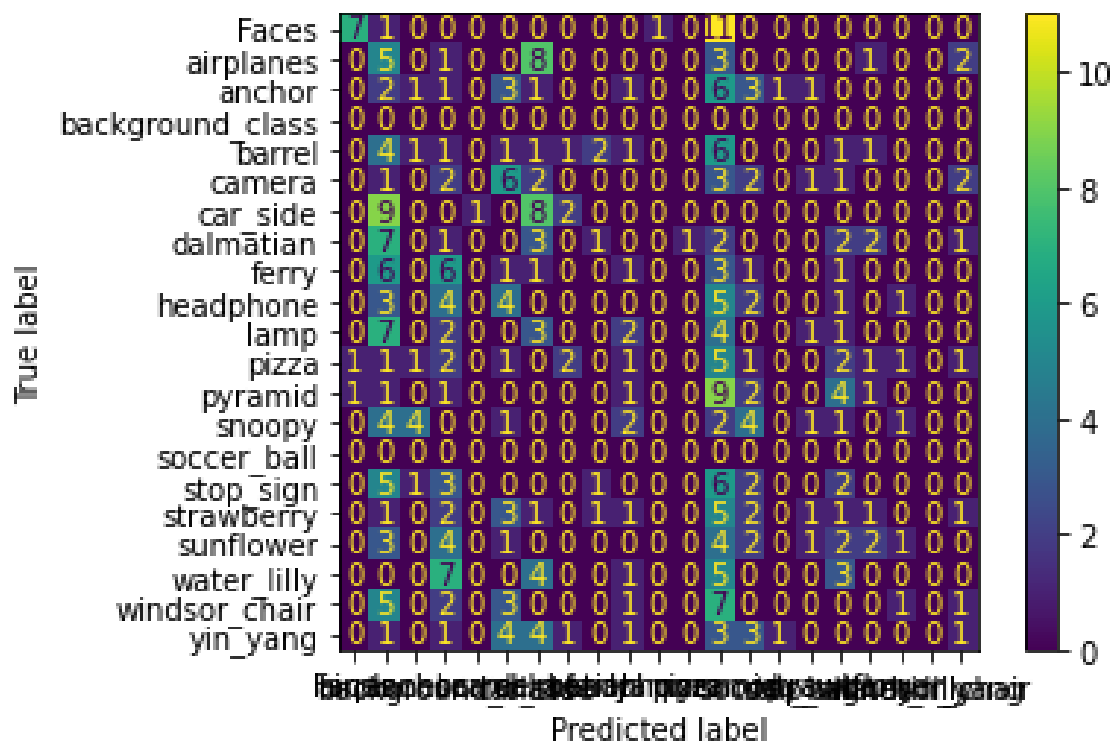
Author: Ahmet Karagöz

K=20

	precision	recall	f1-score	support
Faces	0.72	0.90	0.80	20
airplanes	0.00	0.00	0.00	20
anchor	0.00	0.00	0.00	20
background_class	0.00	1.00	0.00	0
barrel	0.14	0.30	0.19	20
camera	1.00	0.00	0.00	20
car_side	0.19	0.25	0.21	20
dalmatian	0.05	0.05	0.05	20
ferry	0.00	0.00	0.00	20
headphone	0.00	0.00	0.00	20
lamp	0.05	0.10	0.07	20
pizza	0.27	0.20	0.23	20
pyramid	0.00	0.00	0.00	20
snoopy	0.15	0.55	0.23	20
soccer_ball	0.00	1.00	0.00	0
stop_sign	0.06	0.15	0.08	20
strawberry	0.00	0.00	0.00	20

	precision	recall	f1-score	support
sunflower	0.00	0.00	0.00	20
water_lilly	0.00	0.00	0.00	20
windsor_chair	0.25	0.05	0.08	20
yin_yang	0.00	0.00	0.00	20
accuracy			0.13	380
macro avg	0.14	0.22	0.09	380
weighted avg	0.15	0.13	0.10	380

K=50

	precision	recall	f1-score	support
Faces	0.78	0.35	0.48	20
airplanes	0.08	0.25	0.12	20
anchor	0.12	0.05	0.07	20
background_class	0.00	1.00	0.00	0
barrel	0.00	0.00	0.00	20
camera	0.21	0.30	0.25	20
car_side	0.22	0.40	0.29	20
dalmatian	0.00	0.00	0.00	20
ferry	0.00	0.00	0.00	20
headphone	0.00	0.00	0.00	20
lamp	0.00	0.00	0.00	20
pizza	0.00	0.00	0.00	20

pyramid	0.10	0.45	0.17	20
snoopy	0.17	0.20	0.18	20
soccer_ball	0.00	1.00	0.00	0
stop_sign	0.00	0.00	0.00	20
strawberry	0.05	0.05	0.05	20
sunflower	0.22	0.10	0.14	20
water_lilly	0.00	0.00	0.00	20
windsor_chair	1.00	0.00	0.00	20
yin_yang	0.11	0.05	0.07	20
accuracy			0.12	380
macro avg	0.15	0.20	0.09	380
weighted avg	0.16	0.12	0.10	380

K=100

	precision	recall	f1-score	support
Faces	0.31	0.85	0.45	20
airplanes	0.11	0.75	0.19	20
anchor	0.00	0.00	0.00	20
background_class	0.00	1.00	0.00	0
barrel	0.00	0.00	0.00	20
camera	0.10	0.05	0.07	20
car_side	0.14	0.05	0.07	20
dalmatian	1.00	0.00	0.00	20
ferry	0.00	0.00	0.00	20
headphone	0.00	0.00	0.00	20
lamp	1.00	0.00	0.00	20
pizza	0.08	0.05	0.06	20
pyramid	0.20	0.05	0.08	20
snoopy	0.00	0.00	0.00	20
soccer_ball	0.00	1.00	0.00	0
stop_sign	1.00	0.00	0.00	20
strawberry	0.00	0.00	0.00	20
sunflower	1.00	0.05	0.10	20
water_lilly	1.00	0.00	0.00	20

|				
windsor_chair	1.00	0.00	0.00	20
yin_yang	0.14	0.10	0.12	20
accuracy			0.10	380
macro avg	0.34	0.19	0.05	380
weighted avg	0.37	0.10	0.06	380
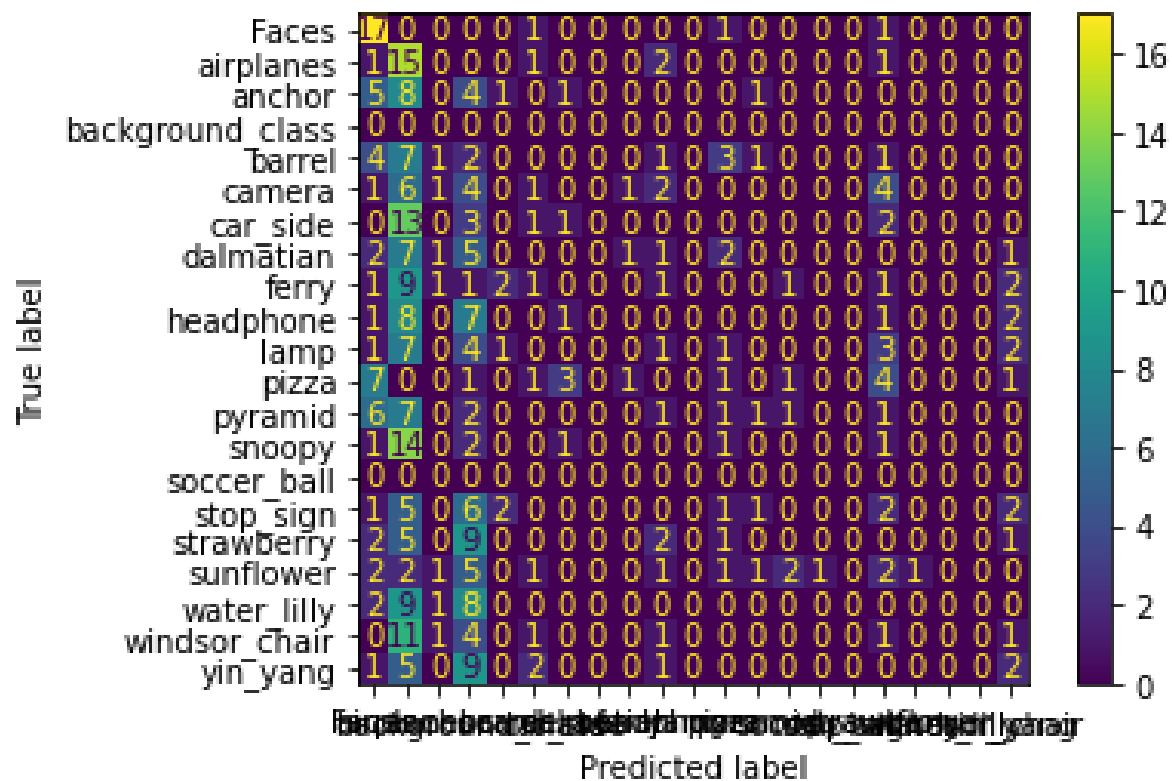
5.Hu Moment(Yasin's Implementation)-K-Means(Ipek's Implementation)-Bag of Words(Mahmut's Implementation)-MLP

Author: Muhammed Yasin Adıyaman

K=20 for Hu Descriptor:

Class	PR	RI	F1	#Item
0	0.20	0.40	0.27	20
1	0.07	0.70	0.13	20
2	1.00	0.00	0.00	20
3	0.00	1.00	0.00	0
4	1.00	0.00	0.00	20
5	1.00	0.00	0.00	20
6	0.21	1.00	0.34	20
7	1.00	0.00	0.00	20
8	1.00	0.00	0.00	20
9	0.05	0.05	0.05	20
10	1.00	0.00	0.00	20
11	1.00	0.00	0.00	20
12	1.00	0.00	0.00	20
13	1.00	0.00	0.00	20
15	1.00	0.00	0.00	20
16	0.00	0.00	0.00	20
17	1.00	0.00	0.00	20
18	1.00	0.00	0.00	20
19	1.00	0.00	0.00	20
20	1.00	0.00	0.00	20

Accuracy = 11%
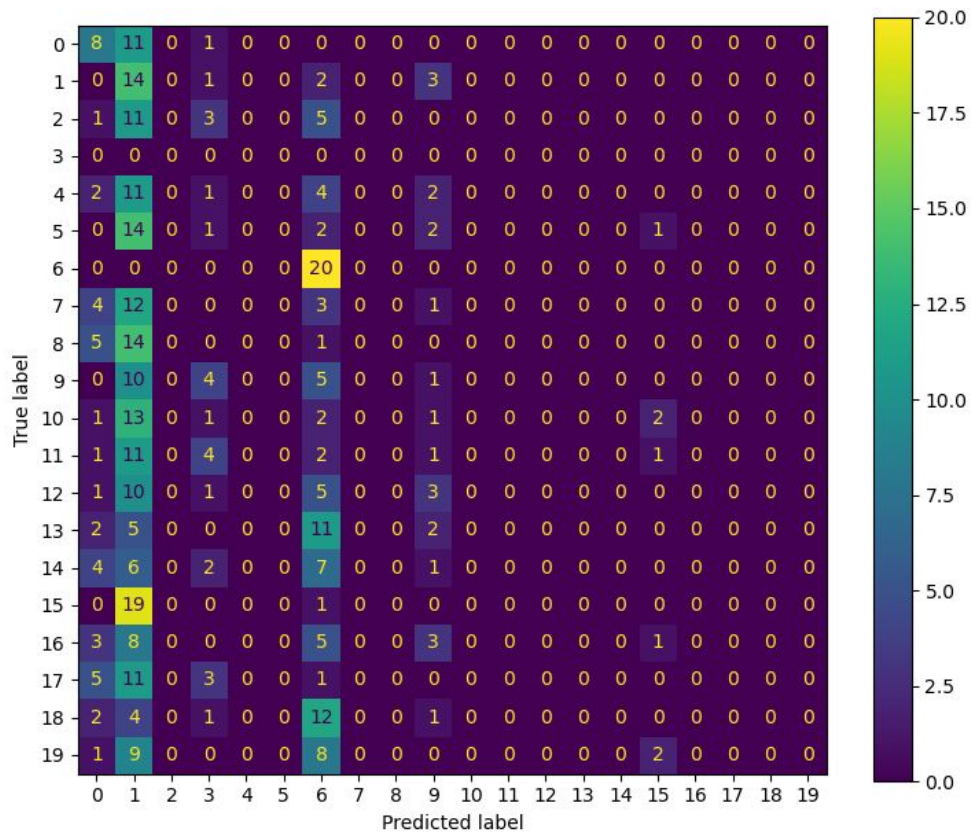
Confusion Matrix for Hu descriptor when K=20

K=50 for Hu Descriptor:

Classes	PR	R	F1	#Items
0	0.27	0.90	0.41	20
1	0.12	0.75	0.20	20
2	0.12	0.05	0.07	20
3	0.00	1.00	0.00	0
4	1.00	0.00	0.00	20
5	1.00	0.00	0.00	20
6	0.43	1.00	0.61	20
7	1.00	0.00	0.00	20
8	0.00	0.00	0.00	20
9	0.05	0.10	0.06	20

10	1.00	0.00	0.00	20
11	1.00	0.00	0.00	20
12	0.50	0.05	0.09	20
13	1.00	0.00	0.00	20
15	0.20	0.05	0.08	20
16	0.24	0.45	0.31	20
17	1.00	0.00	0.00	20
18	1.00	0.10	0.18	20
19	1.00	0.00	0.00	20
20	1.00	0.00	0.00	20
accuracy			0.18	380
macro avg	0.60	0.22	0.10	380
weighted avg	0.63	0.18	0.11	380

Accuracy is 18%.
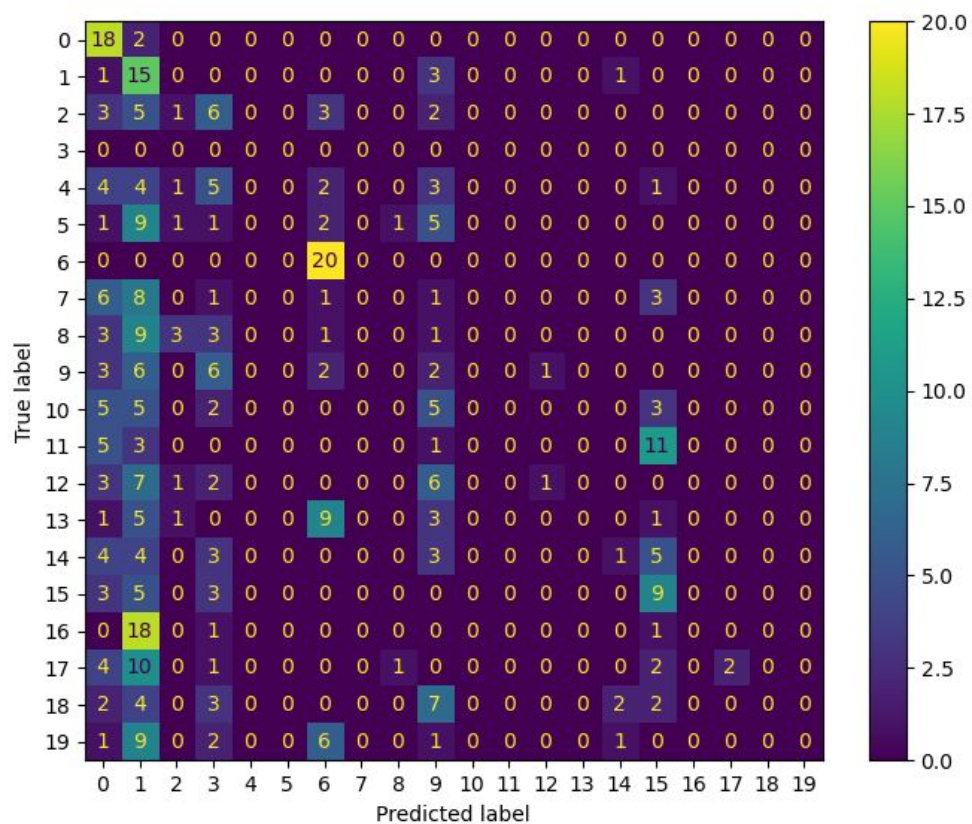
Confusion Matrix for Hu descriptor when K=50

K=100 for Hu Descriptor:

Classes	PR	R	F1	#ofItems
0	0.31	0.80	0.44	20
1	0.15	0.75	0.25	20
2	0.33	0.10	0.15	20
3	0.00	1.00	0.00	0
4	1.00	0.00	0.00	20
5	1.00	0.00	0.00	20
6	0.43	1.00	0.61	20
7	1.00	0.00	0.00	20
8	0.25	0.05	0.08	20
9	0.09	0.15	0.11	20
10	1.00	0.00	0.00	20
11	1.00	0.00	0.00	20
12	1.00	0.00	0.00	20
13	1.00	0.00	0.00	20
15	0.60	0.15	0.24	20
16	0.32	0.45	0.38	20
17	0.00	0.00	0.00	20
18	0.67	0.20	0.31	20
19	1.00	0.00	0.00	20
20	1.00	0.00	0.00	20
accuracy			0.19	380
macro avg	0.61	0.23	0.13	380
weighted avg	0.64	0.19	0.14	380

Accuracy is 19%.
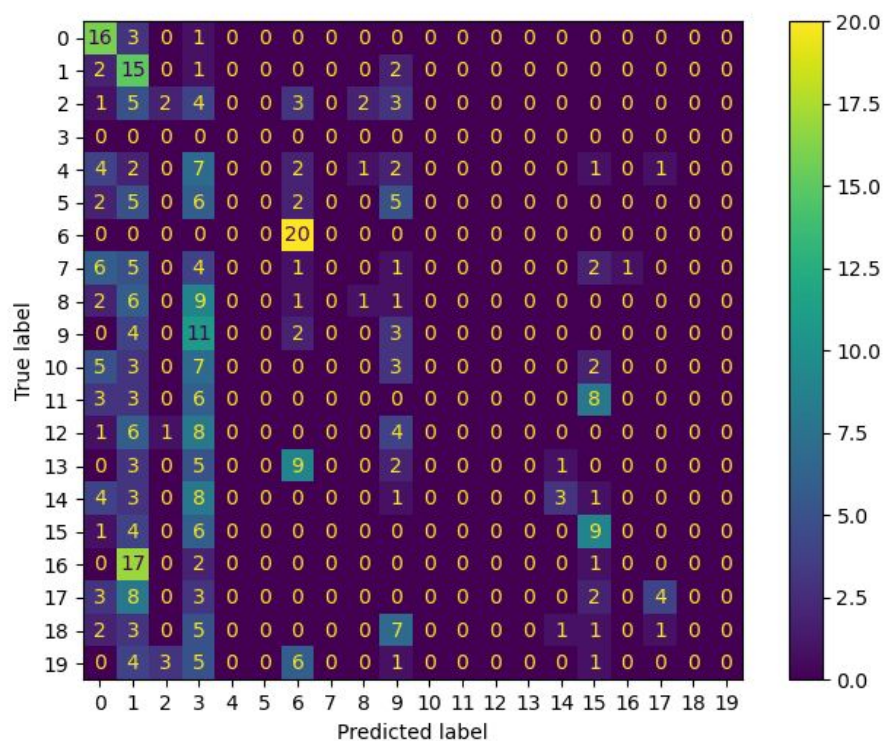
Confusion Matrix for Hu descriptor when K=100

6.Overall Results

Pipeline	K	Macro Average F1 Score
SIFT-K-Means (Ipek) - Bag of Words(Mahmut) - Random Forest	20	0.11
	50	0.16
	100	0.17

ORB-K-Means(Ipek)-Bag of Words(Mahmut)-Decision Tree	20	0.08
	50	0.06
	100	0.06
HOG(Ahmet Emin)-K-Means(Ipek)-Bag of Words(Mahmut)-SVM	20	0.23
	50	0.23
	100	0.21
LBP(Ahmet)-K-Means(Ipek)-Bag of Words(Mahmut)-Adaboost	20	0.13
	50	0.12
	100	0.10
Hu Moment(Yasin)-K-Means(Ipek)-Bag of Words(Mahmut)-MLP	20	0.11
	50	*0.18
	100	*0.19

*Since simulations had not completed by the last submission date, I (Yasin) uploaded the results after that to : https://docs.google.com/document/d/193kI7wIy4eNKoK9Xi64Mxd2r4drA bDCGdU5lwvRTqFw/edit#

References

[1]   A. Bosch, A. Zisserman and X. Munoz, "Image Classification using Random Forests and Ferns," *2007 IEEE 11th International Conference on Computer Vision*, Rio de Janeiro, 2007, pp. 1-8, doi: 10.1109/ICCV.2007.4409066.
[2]   M. Sheykhmousa, M. Mahdianpari, H. Ghanbari, F. Mohammadimanesh, P. Ghamisi and S. Homayouni, "Support Vector Machine Versus Random Forest for Remote Sensing Image Classification: A Meta-Analysis and Systematic Review," in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 13, pp. 6308-6325, 2020, doi: 10.1109/JSTARS.2020.3026724.