Batuhan Tosun

2017401141

EE 576 MACHINE VISION | HOMEWORK 4

Q2) Since I have an odd student id, I have done the Question 2. The goal of this project is to work with scene learning and recognition tasks. ”Scenes-5places” dataset consists of 428 images which are taken in 5 different places. The places/labels in this dataset are “Pl1”, “Pl2”, “Pl7”, “Pl9”, “Pl10”.

Since iterating over the files in C++ to read the images is rather complicated task and unrelated to the purpose of this project, I have extracted a text file, **“all_data.txt”**, in which the directories and the labels of the images are written. In the main program, I used this text file to reach and store the directories and labels of the images.

2.a) Since the dataset is not divided into train and test, the first step is to assign 75% of the samples in each class to train (learning) dataset and 25% to the test dataset. To achieve this, I have implemented and used my own **‘TrainTestSplitter’** function.

After splitting our dataset, the next thing to do is to obtain the Bag of Visual Words representation of the images. To do this, I have implemented and used my **‘SIFTfeatureExtractor’** and **‘BOWRepresentation’** functions. ‘SIFTfeatureExtractor’ function is used to get the descriptors of the images. To get the BOW representations, first, the descriptors of each image in the learning (train) set are given to a ‘BOWKMeansTrainer’ object. Then using the cluster() method of this object, we get the vocabulary of words with length equal to the number of clusters, which is given as a parameter to the ‘BOWKMeansTrainer’ object. The total number of clusters was 100 in my project. Then using a ‘BOWImgDescriptorExtractor’ object and the generated vocabulary, we compute the BOW representation of each image.

2.b) Now, it is time to train our SVM model to perform the classification task. The learning (training) images with BOW descriptors are used in this step. Since this is a multiclass classification task (not binary), we can choose different strategies on how to accomplish the learning of SVM model. The two most common strategies are one-against-all and one-against-one. In this project, I chose the **“one-against-all”** strategy due to the fact that it requires smaller number of SVM to train compared to “one-against-one”. In this strategy, I trained 5 different SVM, each for one class. Although this method is risky for unbalanced datasets, I used a method, which is described below, to prevent the risks.

The number of images in each class are very different; therefore, the learning process of SVM models get affected badly. For example, “Pl2” place has the most image and it has 4-5 times as many images as some places. This results in SVM models to be more inclined to predicting an image as “Pl2”. To prevent from the undesired effects of unbalanced dataset, either “upsampling” or “downsampling” methods should be implement to the training dataset (only in the learning phase). I chose **“upsampling”** method, which is to increment the number of samples in each class via copying its own samples until an (almost) equal distribution is obtained in the dataset.

2.c) Now, it is time to see the performance of our learned SVM models on the learning samples. Before, experimenting on different threshold values, I would like to share the confusion matrix of the training samples. To predict the class of a sample image, first, the confidence values from all the SVM models are gathered and then the one with the highest confidence determines the predicted label of that sample. With this prediction logic, the resulting confusion matrix becomes:

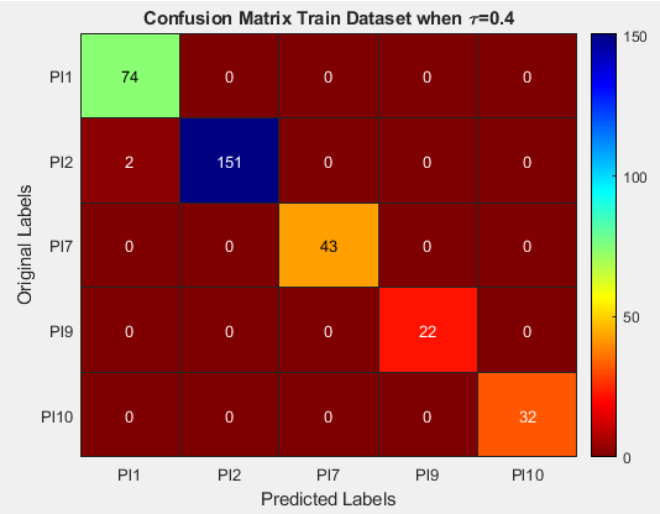


Fig1: Confusion Matrix of Training Samples

From the confusion matrix, it can be observed that our general model (with 5 different SVMs) performs pretty good on the training (learning) set since there are not many samples on the non-diagonal entries. However, this result might be due to overfitting. Therefore, to see the real performance of the model we should be looking at its performance on the test dataset, which will be done in the next section.

Where to apply the thresholding was a little bit unclear; therefore, I followed two different approaches.

1st Approach

In the first approach, I applied thresholding to the best (highest) confidence values of the samples. If a sample has a confidence value, which is generated by the general model, lower than the threshold, it is classified as "uncategorized" and it will not be taken into consideration. If it has a confidence value higher than the threshold, it is classified as the corresponding class of the SVM model which gives the highest confidence value. In this approach, we are actually measuring the performance of the general model on each class with different threshold values.

For three different τ threshold values are used. The threshold values are τ = [0.0, 0.5, 0.7]. To achieve this step, I have implemented and used my 'OneSVMPredict' and 'AllSVMPredict' functions. After obtaining the predictions, using my own 'ConfMatrix' function, I obtained the recall and precision rates for each **"class"** and for each τ.

τ=0.0	"PI1"	"PI2"	"PI7"	"PI9"	"PI10"
Precision	0.974	1.000	1.000	1.000	1.000
Recall	1.000	0.987	1.000	1.000	1.000

Tab1: Precision and Recall values of each class when τ=0.0

τ=0.5	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.986	1.000	1.000	1.000	1.000
Recall	1.000	0.993	1.000	1.000	1.000

Tab2: Precision and Recall values of each class when τ=0.5

τ=0.8	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.000	0.000	0.000	0.000	0.000
Recall	0.000	0.000	0.000	0.000	0.000

Tab3: Precision and Recall values of each class when τ=0.8

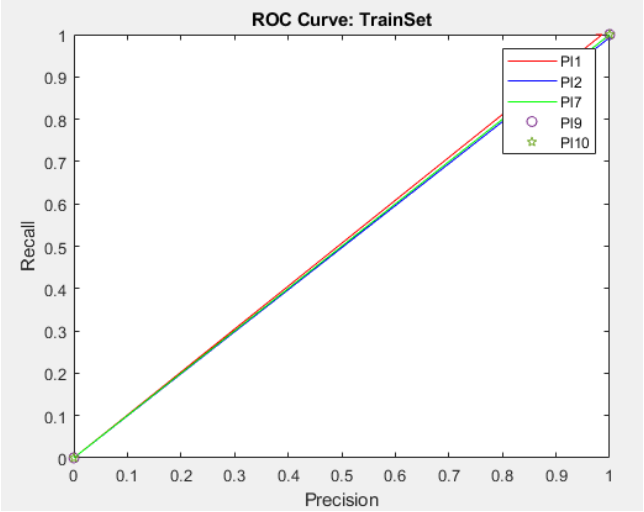


Fig2: ROC Curve(recall vs precision) of TrainSet wrt 1st Approach

2nd Approach

In the second approach, instead of measuring the performance of the general model, I tried to measure the performance of each individual SVM model on their own binary classification task. For this approach, the resulting confidence values of all SVM models for a given sample are all thresholded and those ones higher than the threshold are labeled as “1” and others “-1”.

For three different τ threshold values are used. The threshold values are τ = [0.0, 0.5, 0.7]. To achieve this step, I have implemented and used my ‘OneSVMPredict’ and ‘AllSVMPredict’ functions. After obtaining the predictions, using my own ’ConfMatrix’ function, I obtained the recall and precision rates for each **“model”** and for each τ.

τ=0.0	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.228	0.472	0.133	0.068	0.099
Recall	1.000	1.000	1.000	1.000	1.000

Tab4: Precision and Recall values of each model when τ=0.0

τ=0.5	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.961	0.993	1.000	1.000	1.000
Recall	1.000	0.948	0.977	1.000	0.937

Tab5: Precision and Recall values of each model when τ=0.5

τ=0.8	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.000	0.000	0.000	0.000	0.000
Recall	0.000	0.000	0.000	0.000	0.000

Tab6: Precision and Recall values of each model when τ=0.8

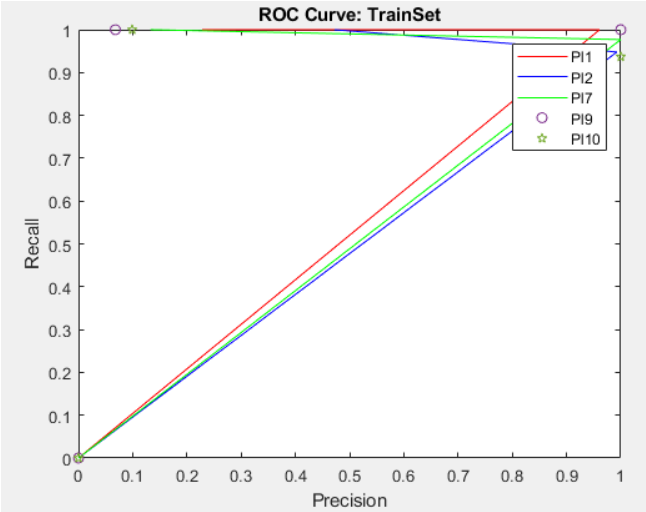


Fig3: ROC Curve(recall vs precision) of TrainSet wrt 2nd Approach

2.d) Now, it is time to see the performance of our learned SVM models on the test samples. Before, experimenting on different threshold values, I would like to share the confusion matrix of the test samples. To predict the class of a sample image, first, the confidence values from all the SVM models are gathered and then the one with the highest confidence determines the predicted label of that sample. With this prediction logic, the resulting confusion matrix becomes:

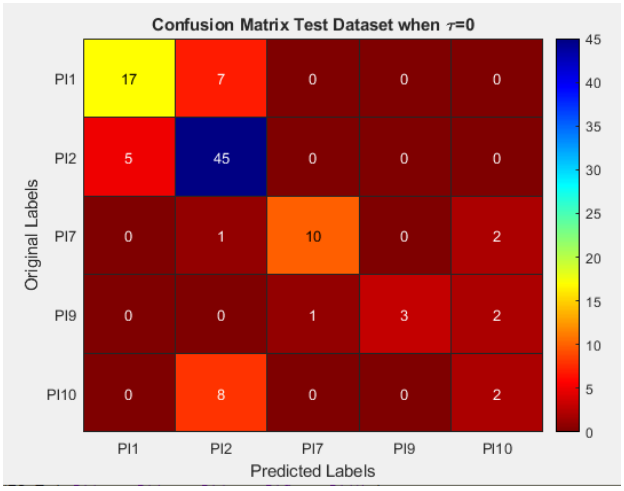


Fig4: Confusion Matrix of Test Samples

1st Approach

τ=0.0	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.772	0.737	0.909	1.000	0.333
Recall	0.708	0.899	0.769	0.500	0.200

Tab7: Precision and Recall values of each class when τ=0.0

τ=0.5	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.849	0.744	1.000	1.000	1.000
Recall	0.708	0.921	1.000	1.000	0.166

Tab8: Precision and Recall values of each class when τ=0.5

τ=0.8	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.000	0.000	0.000	0.000	0.000
Recall	0.000	0.000	0.000	0.000	0.000

Tab9: Precision and Recall values of each class when τ=0.8

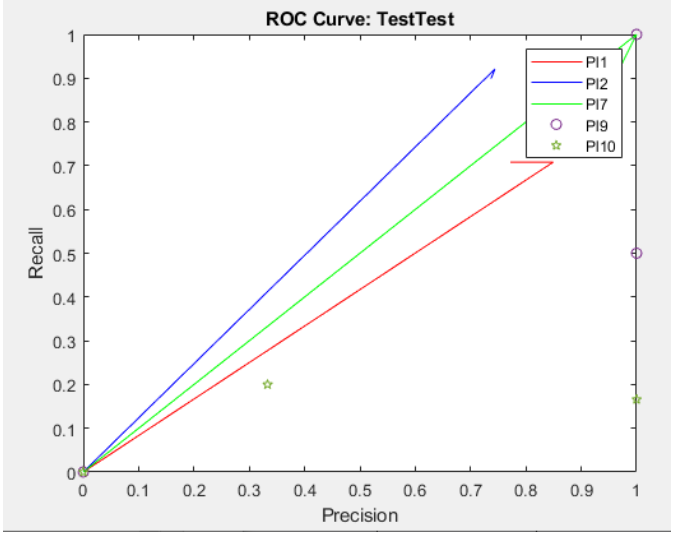


Fig5: ROC Curve(recall vs precision) of TestSet wrt 1st Approach

It is desired to have both high precision and high recall values. Therefore, considering the precision and recall values of each class, our general model has the best performance on class “Pl2” and has the worst performance on class “Pl10”. When we compare the resulting recall-precision rates for each class with those obtained in the previous part, we could observe that the recall and precision rates are lower for each class when the model is performed on the test set. This could mean that the model overfits in the learning stage therefore its performance is not so good on the new samples.

2nd Approach

τ=0.0	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.233	0.485	0.126	0.0582	0.097
Recall	1.000	1.000	1.000	1.000	1.000

Tab10: Precision and Recall values of each model when τ=0.0

τ=0.5	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.792	0.700	1.000	1.000	1.000
Recall	0.792	0.700	0.385	0.333	0.100

Tab11: Precision and Recall values of each model when τ=0.5

τ=0.8	“Pl1”	“Pl2”	“Pl7”	“Pl9”	“Pl10”
Precision	0.000	0.000	0.000	0.000	0.000
Recall	0.000	0.000	0.000	0.000	0.000

Tab12: Precision and Recall values of each model when τ=0.8

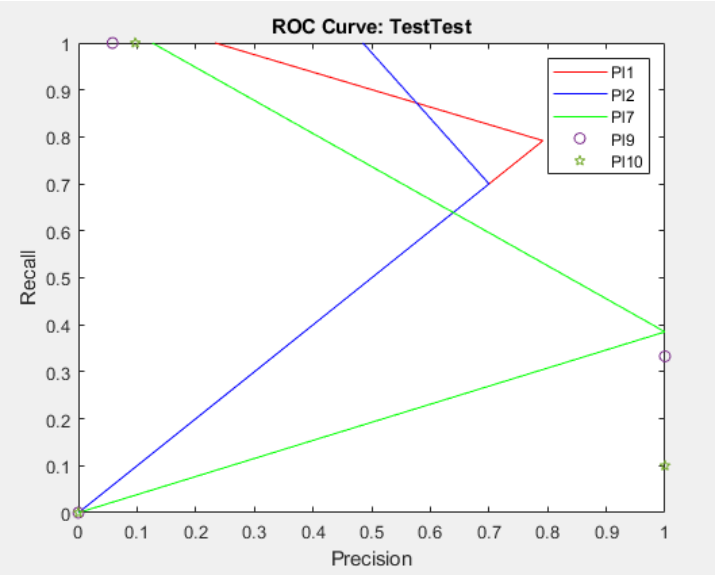
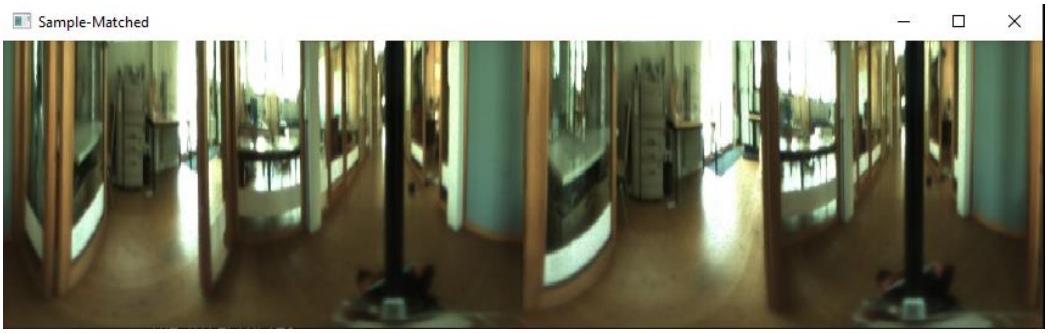


Fig2: ROC Curve (recall vs precision) of TestSet wrt 2nd Approach

It is desired to have both high precision and high recall values. Therefore, considering the precision and recall values of each class models, the “Pl1” class model has the best performance and the “Pl10” has the worst performance in general.

2.e) In this part of the project, the aim is to find the best matching sample via matching the SIFT features of the given test image with those of the matched class. To achieve this, I have implemented and used my own **‘imgMatcher’** function. This function first finds the corresponding test sample from the test dataset with respect to the given ‘user_number’ variable. Using the learned SVM models, it predicts the class of the test sample and ‘matched class’ becomes the predicted class in this case. The next thing to do is to iterate over the samples of the matched class and to find the similarities between the given test sample and each sample in the matched class using their SIFT descriptors. To find the similarities, first, Brute Force matcher is used to match the descriptor vectors of the two samples. To filter the matches and only consider the good ones Lowe’s ratio test is used. Therefore, the distance ratio between the two nearest matches of a considered keypoint is computed and it is considered as a good match when this value is below a threshold. The image that gives the highest number of good matches with our given image is considered as the best matching image. The performance of this function is almost perfect when the predicted class is correct. Even though the predicted class is wrong, the best matching image and the given test image have a lot of similarities. Below, I put there are some examples.



Sample: “t1152873113.101280_x3.196403_y-0.315628_a-0.019693.jpeg” – Class: “Pl1”

Match: “t1152873115.141308_x3.616271_y-0.327267_a-0.021332.jpeg” – Class: “Pl2”

Sample: “t1152873149.867481_x5.211422_y-1.303718_a1.471692.jpeg” – Class: “Pl2”

Match: “t1152873149.867481_x5.211422_y-1.303718_a1.471692.jpeg” – Class: “Pl2”

Sample: “t1152873148.031363_x5.182961_y-1.653757_a1.411156.jpeg” – Class: “Pl2”

Match: “t1152873148.031363_x5.182961_y-1.653757_a1.411156.jpeg” – Class: “Pl2“