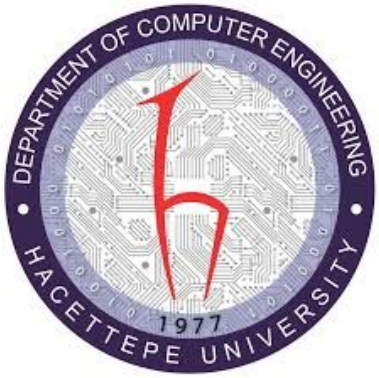


DEPARTMENT OF COMPUTER ENGINEERING

BBM 418 - COMPUTER VISION LABORATORY

PROGRAMMING ASSIGNMENT-2

AUTHOR: ATAKAN AYYILDIZ
(ID: 21526681)
PROJECT SUPERVISOR: DOÇ. DR. NAZLI İKIZLER CINBIŞ

APRIL 16, 2021

Abstract

In this assignment we were getting familiar with image recognition techniques. Such as tiny image and BoVW (with Sift implementation). After getting familiar with these feature extraction techniques, then we made classifications given data set.[1] The used classification techniques are the KNN classifier[2] and Linear SVM. [3] At last we compared the result and made inferences.

The Implementation Details

0.0.1 About data sets:

First of all we need to specify the Scene Data set path. After the path specification, we first get the tiny images. g̈et_tiny_image(img)f̈unction takes image array as parameter then resize it to 16x16, at last it normalize the image. It converges the mean to zero. We can see that the means are now near zero (and would be zero if it was not for rounding error when converting the values to integers), but the variance of each distribution has not changed. This shows how calculating the residuals is effectively shifting the mean of each distribution to align them for direct comparison. After that the function returns a 1D numpy array. We pass the 1D array, class type(Kitchen,Mountain, etc...) and filename to the tiny_data_set. We are passing filenames because we split them and we do not want them lost. They are required for correct and misclassification.

```
def get_tiny_image(image):
    image = cv2.resize(image, (16, 16))
    image = (image - np.mean(image)) / np.std(image)
    # Normalization work well because when we do this means can be near to zero but the variance will be the same

    image = image.flatten()  # to obtain 1d image array
    return image
```

After creating tiny_data_set we create the BoVW data set. But first we need to explain the Sift library.
About SIFT: There are mainly 4 steps in sift algorithm.

1.Scale space Extrema Detection: In order to detect larger corners scale-space filtering is used. In it, Laplacian of Gaussian (LoG) is found for the image with various values. Sigma() acts as a scaling parameter. Local maxima accross the scale and space which gives us a list of (x,y,) can be found.These are the potential keypoints at scale. Sift uses Difference of Gaussian's (DoG) which is an approximation of LoG. Once DoG are found, images are searched for local extrema over scale and space.

2.Keypoint Localization: In order to get more accurate results on potential keypoints Taylor series are used. Threshold values are used to get more accurate result, such as if a local extrema intensity is less than the contrast-Threshold value then it is rejected otherwise accepted. But DoG has higher response for edges and edges must be removed. There are some algorithms for edge detection. At the end low-contrast key-points and edge key-points are removed.

3.Orientation Assignment: Orientation is assigned to each keypoint to achieve invariance to image rotation.

4.Keypoint descriptor: Here keypoint descriptor is created. There are 128 bin

values are available. (16 sub-blocks x 8 bin). It is represented as a vector.

Keypoints between two images are matched by identifying their nearest neighbours. But sometimes the nearest two neighbour may be very near because of noise or etc. In order to avoid this case if ratio of closest distance to second closest distance is greater than 0.8, then they are rejected. This mostly eliminates correct ones.

Sift.detect is for detecting keypoints. After keypoints are found function compute the descriptors from the keypoints that have been found. Keypoints are the list of the keypoints and descriptors is a numpy array of size #of keypoint x 128(bin values)

```
# bow
kp, descriptor = sift.detectAndCompute(img, None)
bow_data_set.append([descriptor, folder, filename])
```

After obtaining the keypoints and descriptors we put the descriptors to the data set with its class name and filename.(Passing them to the data sets is the same for both feature extraction techniques.)

```
X_train, X_test, y_train, y_test, file_train, file_test = train_test_split([x[0] for x in data_set],
                                                                           [x[1] for x in data_set],
                                                                           [x[2] for x in data_set],
                                                                           test_size=0.2, random_state=42)
```

Splitting is the same for both data sets.

0.0.2 Implementation of Bag of Visual Words:

After splitting the data set of BoVW we need a descriptor list that includes all the train descriptors in an unordered numpy list. We pass the descriptor list to KMeans with 100 clusters. K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. We take the k(n_clusters) as 100. I have tried it with different k values. Up to 100 it increases fast but when it is higher than 100 there is not much increases on accuracy. Also there is a theorem about choosing the appropriate k value is called Elbow Technique [4].
After defining KMenas and fitting with the descriptor list. We had to create the histograms as the BoVW.

```
def create_histograms(array, kmean_algo):  # histogram is the bag of visual words
    histogram_list = []
    for descriptor in array:
        if descriptor is not None:
            histogram = np.zeros(len(kmean_algo.cluster_centers_))
            cluster_result = kmean_algo.predict(descriptor)
            for i in cluster_result:
                histogram[i] += 1.0
            histogram_list.append(histogram)
    return histogram_list
```

.

kmean_algo is the our defined KMeans algorithm and it had to be predicted with both train and test descriptors. Kmeans.predict returns index of the cluster each sample belongs to. After that we make frequency histogram from the vocabularies and the frequency of the vocabularies in the image. Those histograms are our bag of visual words.

0.0.3 Evaluation of Data sets:

Evaluation of the tiny image:

```
def tiny_image_with_KNN_SVM(data_set):
    X_train, X_test, y_train, y_test, file_train, file_test = train_test_split([x[0] for x in data_set],
                                                                               [x[1] for x in data_set],
                                                                               [x[2] for x in data_set],
                                                                               test_size=0.2, random_state=42)
    # Linear SVM and tiny image
    clf = svm.LinearSVC(max_iter=10000, dual=False)  # In order to avoid convergence
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print("My Accuracy of tiny image with Linear SVM {0:.2f}".format(accuracy(y_pred, y_test)))
    print("Accuracy of accuracy_score function", accuracy_score(y_pred, y_test))
    print("Classification report:\n", classification_report(y_pred, y_test))
    print("Confusion matrix:\n", confusion_matrix(y_pred, y_test))
    print("-------------------------------------------------")

    # KNN and tiny image
    classifier = KNeighborsClassifier(n_neighbors=19)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    print("My Accuracy of tiny image with 19-NN {0:.2f}".format(accuracy(y_pred, y_test)))
    print("Accuracy of accuracy_score function", accuracy_score(y_pred, y_test))
    print("Classification report:\n", classification_report(y_pred, y_test))
    print("Confusion matrix:\n", confusion_matrix(y_pred, y_test))
```

After splitting the tiny_data_set we first create the Linear Support Vector Classifier with specified max_iter=10000 and dual=False. We give these parameters in order to avoid the convergence. When we use **LinearSVC()** there is a warning: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations. If the algorithm does not converge, then the current estimate of the SVM's parameters are not guaranteed to be any good, hence the predictions can also be complete garbage.Solving the linear SVM is just solving a quadratic optimization problem. The solver is typically an iterative algorithm that keeps a running estimate of the solution (i.e., the weight and bias for the SVM). It stops running when the solution corresponds to an objective value that is optimal for this convex optimization problem, or when it hits the maximum number of iterations set.

Then we fit the SVM model according to the given training data. After fitting the SVM we just pass the whole X_test to the predict function, it returns a list with predicted class labels(Kitchen, Mountain,etc...)

After completing the SVM,next is the **KNN classifier**. I had tried different n values to obtain best accuracy. But it always gave me around 40% accuracy. Therefore I just gave the highest accuracy (11).

Evaluation of the Bag of Visual Words:

As we had already discussed the KMeans implementation we now explain the KNN classification of the BoVW. 19-KNN gives the best result in itself them (61% accuracy). Linear SVM gives the best results (62% accuracy). Implementations of Bag of Words after Kmeans is the same as we did in the tiny images. So that i do not write it.

```python
def boVW_with_KNN_SVM(data_set):
    X_train, X_test, y_train, y_test, file_train, file_test = train_test_split([x[0] for x in data_set],
                                                                               [x[1] for x in data_set],
                                                                               [x[2] for x in data_set],
                                                                               test_size=0.2, random_state=42)
    descriptor_list = list()
    for x in X_train:
        descriptor_list.extend(x)
    descriptor_list = np.array(descriptor_list)
    kmeans = KMeans(n_clusters=100, n_init=10)
    kmeans.fit(descriptor_list)
    train_histograms = np.array(create_histograms(X_train, kmeans))
    test_histograms = np.array(create_histograms(X_test, kmeans))

    y_pred = list()
    k = 19
    classifier = KNeighborsClassifier(n_neighbors=k)
    classifier.fit(train_histograms, y_train)  # pickle.load(pickle_in5)
    for x in test_histograms:
        y_pred.append(classifier.predict([x]))
    print("My Accuracy of BoVW with 19-NN {0:.2f}".format(accuracy(y_pred, y_test)))
    print("Accuracy of accuracy_score function", accuracy_score(y_pred, y_test))
    print("Classification report:\n", classification_report(y_pred, y_test))
    print("Confusion matrix:\n", confusion_matrix(y_pred, y_test))
    print("-------------------------------------------------")

    y_pred = list()
    clf = svm.LinearSVC(max_iter=10000, dual=False)  # In order to avoid convergence (More on report)
    clf.fit(train_histograms, y_train)
    for x in test_histograms:
        y_pred.append(clf.predict([x]))
    print("My Accuracy of BoVW with Linear SVM {0:.2f}".format(accuracy(y_pred, y_test)))
    print("Accuracy of accuracy_score function", accuracy_score(y_pred, y_test))
    print("Classification report:\n", classification_report(y_pred, y_test))
    print("Confusion matrix\n", confusion_matrix(y_pred, y_test))
```

0.0.4 Accuracy metrics:

There were four types of accuracy metrics used. First one is my written code below.

```python
def accuracy(predicted, real):
    n = len(predicted)
    counter = 0
    for i in range(n):
        if predicted[i] == real[i]:
            counter += 1

    return counter / n
```

```
Classification report:
              precision    recall  f1-score   support

     Bedroom       0.23      0.39      0.29        23
     Highway       0.78      0.68      0.72        62
     Kitchen       0.17      0.32      0.23        22
  LivingRoom       0.70      0.51      0.59        73
    Mountain       0.89      0.89      0.89        85
      Office       0.54      0.46      0.49        48

    accuracy                           0.62       313
   macro avg       0.55      0.54      0.54       313
weighted avg       0.67      0.62      0.64       313
```

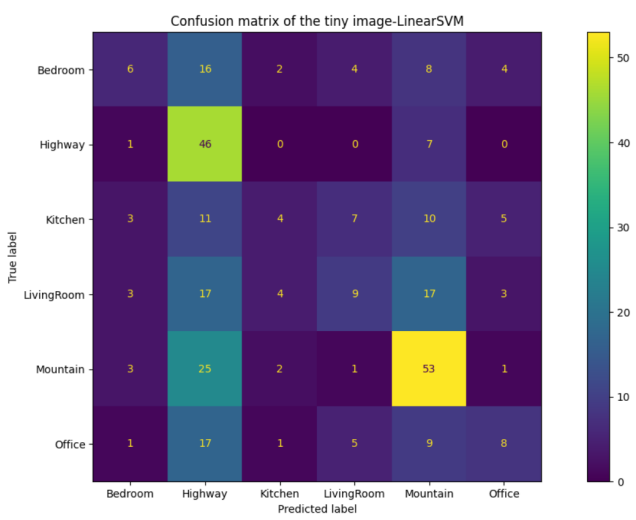
```
Confusion matrix
[[ 9  2  5  4  1  2]
 [ 5 42  6  0  7  2]
 [ 4  1  7  5  0  5]
 [14  2  9 37  1 10]
 [ 2  7  0  0 76  0]
 [ 6  0 13  7  0 22]]
```

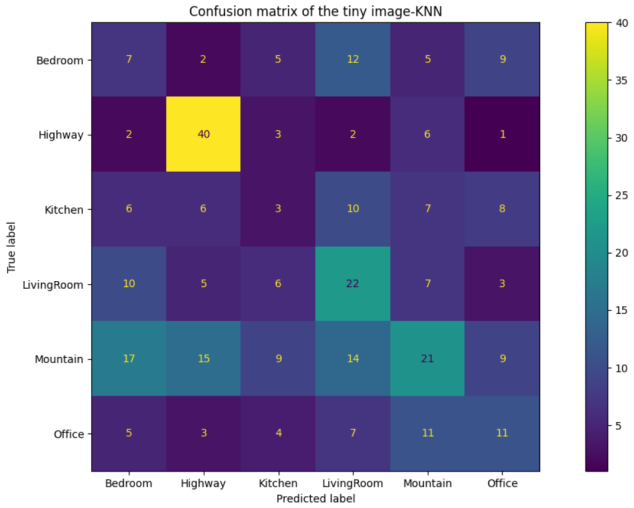
Second metric is the same as mine. Third one is the classification report. Confusion matrix is a great material because it helps us to make inference. But classification report gives better accuracy on weighted avg. In order to see different calculation metric result I used classification report. Also confusion metric gives a visual result. My created

accuracy metric and the accuracy_metric have the same result (that means I calculate true results).

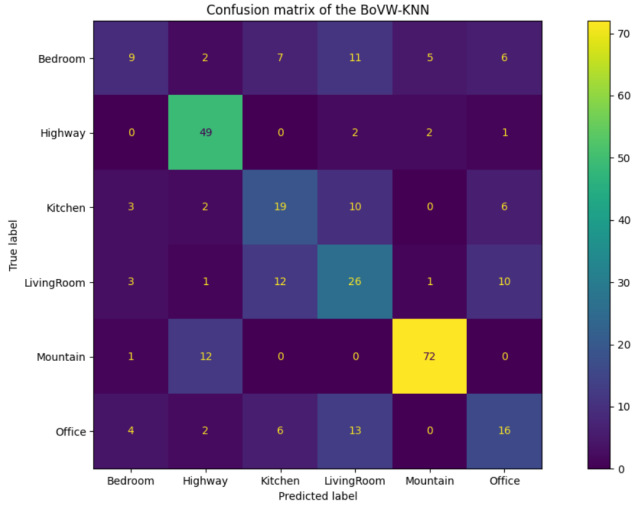
0.0.5 Confusion Matrix Plots:

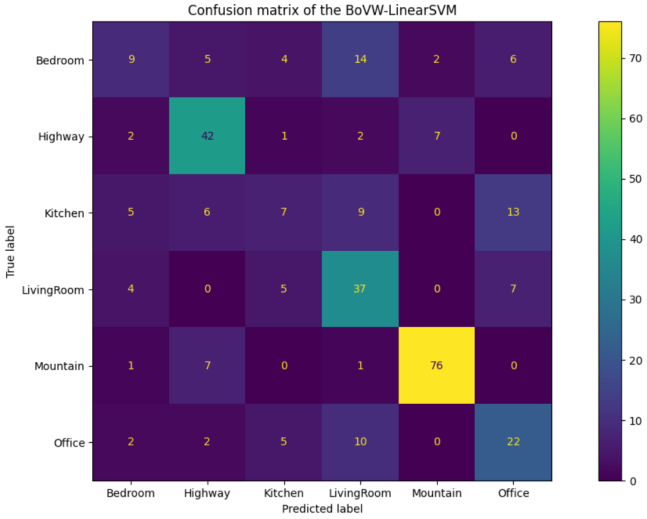
There are four confusion matrix. First two are created with the tiny images. The results are very bad. As you can see below.





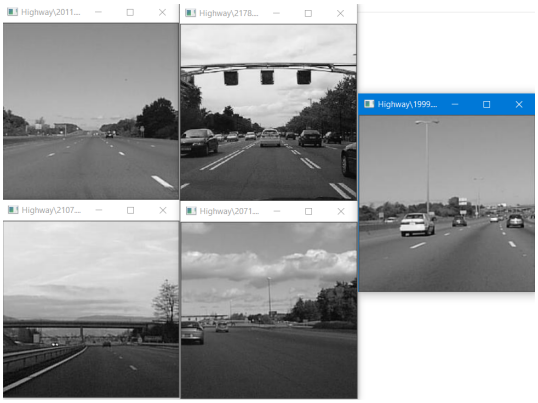
And now these below are BoVW implementations. These are much better so we can say BoVW implementation gives better results than tiny image

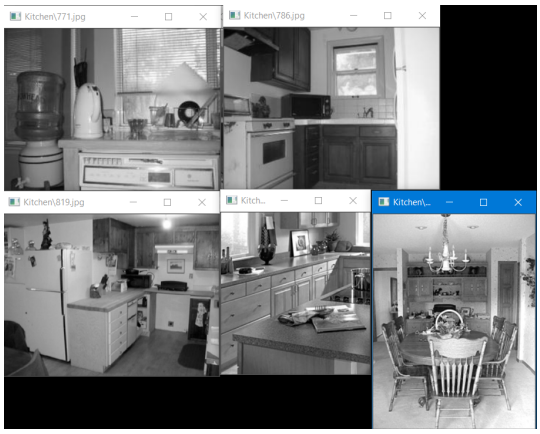


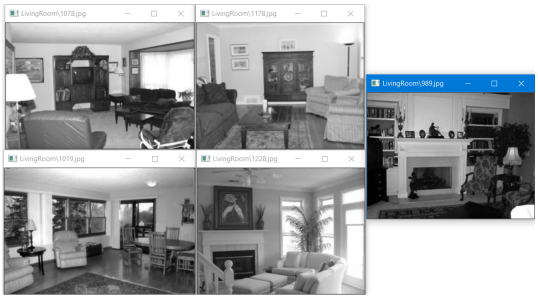


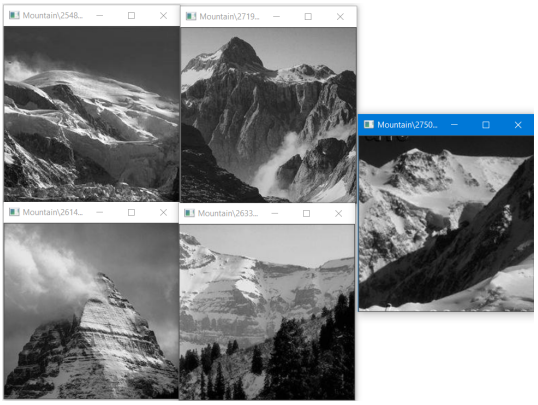
0.0.6 Correct Samples:

Correct bedroom samples 

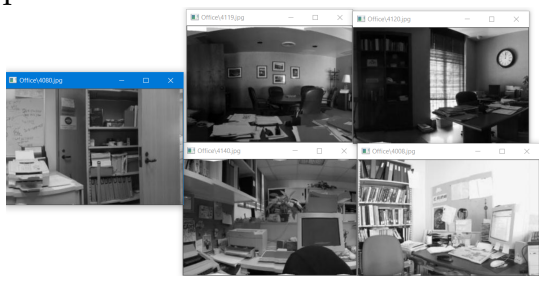
Correct highway samples 

Correct kitchen samples 

Correct living room samples 



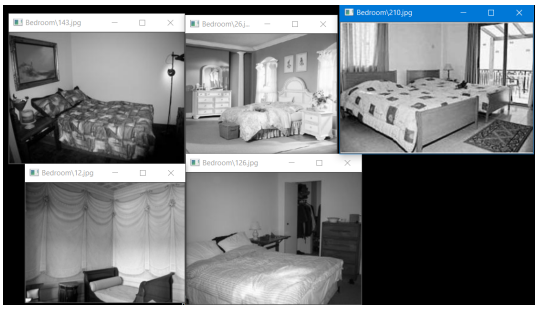
Correct mountain samples



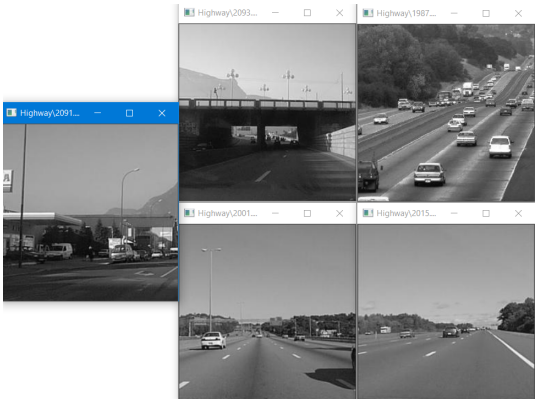
Correct office samples

0.0.7 False Samples:

These are failed images. The main reason: Bag of visual words of Linear SVM is poor at localizing objects within an image. By looking at the kitchen images it is hard to predict true label. It has many edges to predict wrong. But Linear SVM mostly it predict its own class

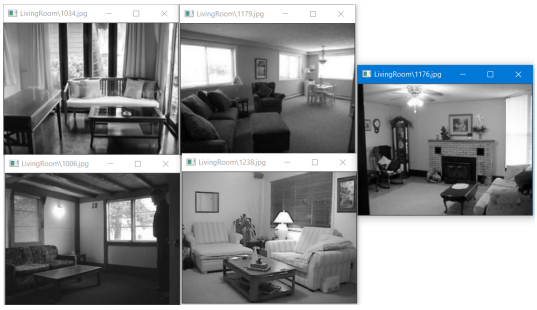


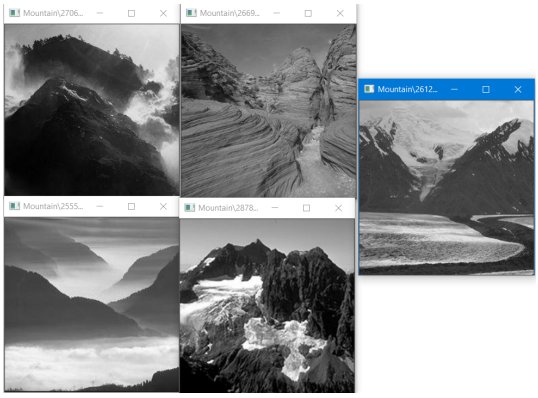
False bedroom samples

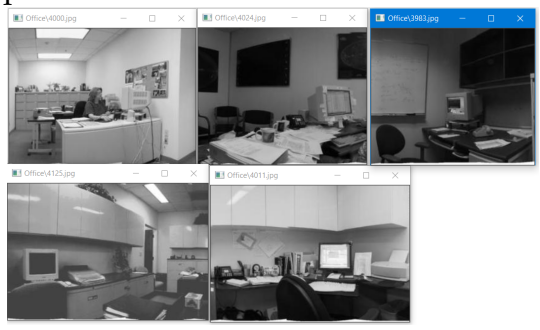


False highway samples

False kitchen samples 

False living room samples 

False mountain samples 

False office samples 

0.0.8 Conclusion:

First of all we can obviously say that Bag of Visual Words gets better results than tiny image. But tiny image much faster than BoVW. This is about img size and not using Kmeans. Tiny image with SVM get 33% accuracy at most.This is the worst of four that we implement. Next is again tiny image with KNN. It gets at most 19-NN gets 40% accuracy. But using N too high limits the performance when we use much higher data sets.

On the other hand when we implement BoVW it gets more higher accuracy. Taking

Kmeans n_cluster=100 and using KNN (K=19) gives 59% accuracy. Using SVM on BoVW gets the best results. It gives 62% accuracy. According to other same researches gets at most 72% accuracy. I can say that if I take normalization on the data set, then results gives me near to that 72% accuracy.

Bibliography

[1] [Online]. Available: https://drive.google.com/file/d/1c_kTQNm-SXN6-ZmVcC8ZMOiLssRhy-1I/view?usp=sharing

[2] [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[3] [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[4] [Online]. Available: https://en.wikipedia.org/wiki/Elbow_method_(clustering)