Assignment 2

Mustafa Emre Başar, 21726999
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
b21726999@cs.hacettepe.edu.tr

April 15, 2021

1 Introduction

In this assignment we were supposed to implement scene classifier with using K-Nearest Neighbour and SVM, using Tiny Image feature and Bag of Visual Words for pre-processing the images.

Detailed explanation of steps are examined in further sections.

2 Test and Train Data Set Creation

Data set creation was made manually. I tried to pick images that have different kind of features to increase flexibility. I used 80 percent of the images for train and rest for the test data set.

Sample counts shown below;

	Train	Test
Bedroom	172	44
Highway	208	52
Kitchen	168	42
LivingRoom	232	57
Mountain	300	74
Office	173	42
Total	1253	311

3 Feature Extraction

3.1 Tiny Image Feature

After data set creation, for the first ore-processing, I used tiny image feature. I resize the image and normalize it to get a more appropriate result and return feature of every image as one-dimensional array each index corresponds to an image. Relevant code shown below;

```
def get_tiny_images(image_paths):
    height = 16
    width = 16

    tiny_images = np.zeros((len(image_paths), width * height))

    for i, image_data in enumerate(image_paths):
        image = Image.open(image_data)
        image_re = np.asarray(image.resize((width, height),
                        Image.ANTIALIAS), dtype='float32').flatten()
        image_nm = (image_re - np.mean(image_re)) / np.std(image_re)
        tiny_images[i, :] = image_nm

    return tiny_images
```

3.2 Bag of Words

Another pre-process feature is Bag of Words feature. For this feature I used SIFT to extract features of each image. After the key points and descriptor extraction by using MiniBatchedKmeans, clustered the data. Since there are six classes, I created clusters of 60 to be able to get get similar features together. Here, the parameter batch size, corresponds to samples used in each iteration in K means, is declared according to the size of train data. After clustering the the data, the corresponding histograms are created these histograms normalized according to the size of the key points. At the end, created histograms are returned to be classified later.

The function as shown below;

```
def build_vocabulary(image_paths, dico):
    dico = []
    sift = cv2.SIFT_create()

    for path in image_paths:
        img = cv2.imread(path)

        kp, des = sift.detectAndCompute(img, None)
        for d in des:
            dico.append(d)

    k = 60

    x = 3
    batch_size = int(1253 * x)
    vocab = MiniBatchKMeans(n_clusters=k, batch_size=batch_size,
                            verbose=1).fit(dico)

    vocab.verbose = False

    histo_list = []
    for path in image_paths:
        img = cv2.imread(path)
        kp, des = sift.detectAndCompute(img, None)

        histo = np.zeros(k)
        nkp = np.size(kp)

        for d in des:
            idx = vocab.predict([d])
            histo[idx] += 1 / nkp

        histo_list.append(histo)

    return histo_list
```

4 Classification

After processing the images for the classifications there are two methods. Both of these methods using an array of features of images. First train the model with train_image_features with the corresponding labels then try to predict the test images according to trained model. Here the train and test image features are the return values of Tiny Image or Bag of Words according to the choose.

4.1 K-Nearest Neighbour

First one is K neighbor classification. I used KNN classifier from sklearn library.

```
from sklearn.neighbors import KNeighborsClassifier
```

Usage;

```
    if (classify == "knn"):
        knn = KNeighborsClassifier(n_neighbors=5)
        c = knn.fit(train_image_feats, train_labels)
        pred_labels = knn.predict(test_image_feats)
```

My K parameter choosing here, is based on trials and errors ,by trying many number of k's , aimed to find the best one possible.

4.2 Linear SVM

The second classifying method was Linear SVM. I used svm of sklearn again.

```
from sklearn import svm
```

Usage;

```
if (classify == "svm"):
    C = 10.0
    G = 0.01
    K = 'linear'
    clf = svm.SVC(C=C, gamma=G, kernel=K)
    c = clf.fit(train_image_feats, train_labels)
    pred_labels = clf.predict(test_image_feats)
    print("SVM parameters : C = %.1f, Gamma : %.2f
            Kernel : %s " % (C, G, K))
```

Here you can see the parameters C, G and K they correspond to regularity, gamma and kernel parameters respectively. I mostly used G with low values when I used tiny image since data is relatively small and C ,which is how much it can tolerate miss classification, again used with small values when tiny image is used, since the data itself is already small, toleration should be small also. Here kernel parameter is linear since we were supposed to use LinearSVM.

5 Test Phase

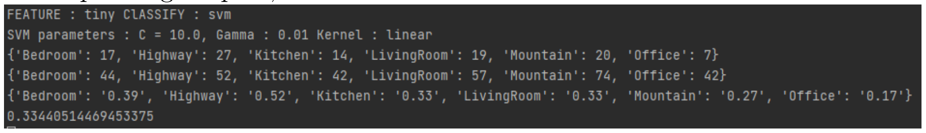
5.1 Accuracy Calculation

For accuracy calculation, I simply divide the amount of correctly found test images to total number of test images. Code Shown Below;

```
match = 0
    for i in range(len(pred_labels)):
        pred = pred_labels[i]
        act = test_labels[i]
        if (pred == act):
            match += 1
            cor_found_labels[act] = cor_found_labels[act] + 1

    accur = match / len(pred_labels)

    for i in cor_found_labels.keys():
        acc = cor_found_labels[i] / test_sample_counts[i]
        test_labels_accurs[i] = "%.2f" % acc

    print(cor_found_labels)
    print(test_sample_counts)
    print(test_labels_accurs)
    print(accur)
```

Here if the label predicted is same with corresponding test label then it is a match. Ratio of matches to total predictions is the accuracy. Also with second for loop, I found accuracy for each class individually. cor_found_labels and test_sample_counts are dictionaries that stores correctly predicted labels and test image counts for each class respectively.

Corresponding output ;



```
FEATURE : tiny ClASSIFY : svm
SVM parameters : C = 10.0, Gamma : 0.01 Kernel : linear
{'Bedroom': 17, 'Highway': 27, 'Kitchen': 14, 'LivingRoom': 19, 'Mountain': 20, 'Office': 7}
{'Bedroom': 44, 'Highway': 52, 'Kitchen': 42, 'LivingRoom': 57, 'Mountain': 74, 'Office': 42}
{'Bedroom': '0.39', 'Highway': '0.52', 'Kitchen': '0.33', 'LivingRoom': '0.33', 'Mountain': '0.27', 'Office': '0.17'}
0.33440514469453375
```

5.2 Confussion Matrices

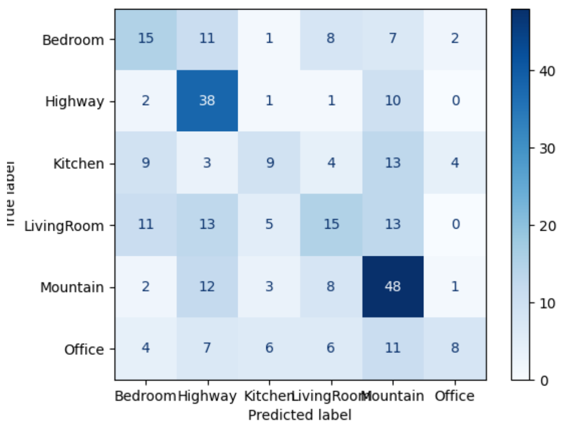


Figure 1: Tiny Image - KNN

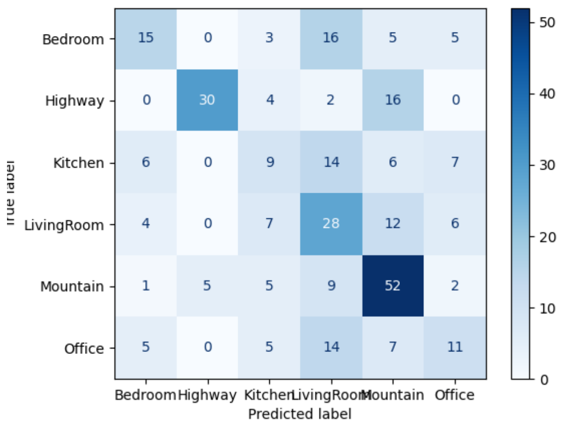


Figure 2: Tiny Image - SVM

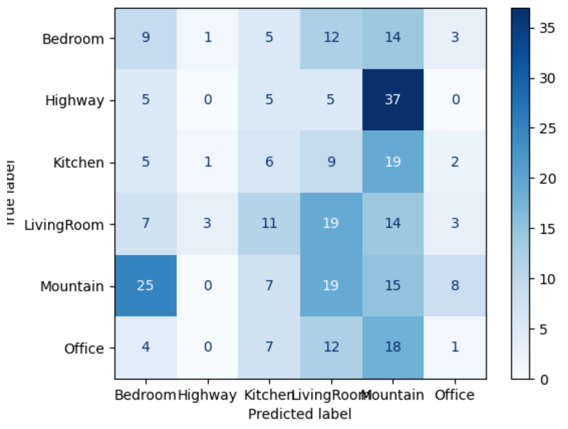


Figure 3: BoW - KNN

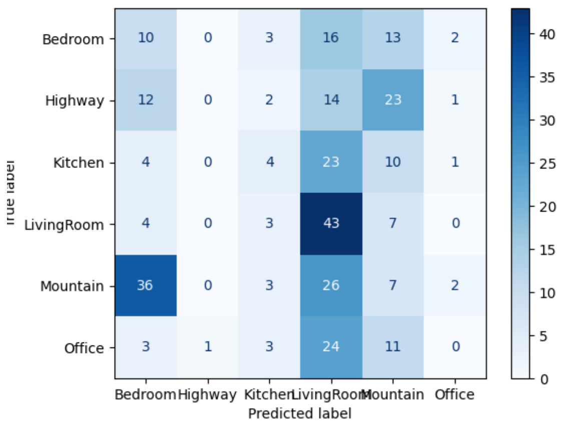


Figure 4: Bow - SVM

5.3 Test Images

GOOD EXAMPLES;
Bedroom;

(a) Bedroom 1

(b) Bedroom 2

(c) Bedroom 3

(d) Bedroom 4

(e) Bedroom 5

Highway;

(a) Highway 1

(b) Highway 2

(c) Highway 3

(d) Highway 4

(e) Highway 5

Kitchen;

(a) Kitchen 1

(b) Kitchen 2

(c) Kitchen 3

(d) Kitchen 4

(e) Kitchen 5

Living Room;

(a) Living Room 1

(b) Living Room 2

(c) Living Room 3

(d) Living Room 4

(e) Living Room 5

Mountain;

(a) Mountain 1

(b) Mountain 2

(c) Mountain 3

(d) Mountain 4

(e) Mountain 5

Office;

(a) Office 1

(b) Office 2

(c) Office 3

(d) Office 4

(e) Office 5

BAD EXAMPLES;
Bedroom;

(a) Bedroom 1

(b) Bedroom 2

(c) Bedroom 3

(d) Bedroom 4

(e) Bedroom 5

Highway;

(a) Highway 1

(b) Highway 2

(c) Highway 3

(d) Highway 4

(e) Highway 5

Kitchen;

(a) Kitchen 1

(b) Kitchen 2

(c) Kitchen 3

(d) Kitchen 4

(e) Kitchen 5

Living Room;

(a) Living Room 1

(b) Living Room 2

(c) Living Room 3

(d) Living Room 4

(e) Living Room 5

Mountain;

(a) Mountain 1

(b) Mountain 2

(c) Mountain 3

(d) Mountain 4

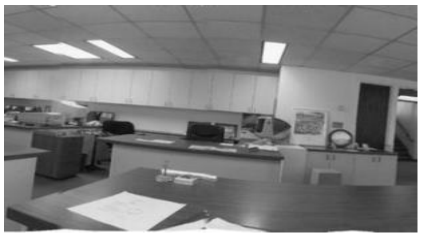
(e) Mountain 5

Office;

(a) Office 1

(b) Office 2

(c) Office 3


(d) Office 4

(e) Office 5

5.4 Comparison

Best results for ;

- Tiny Image - K-Nearest Neighbour : % 40
- Tiny Image - Linear SVM : % 46
- Bag of Words - K-Nearest Neighbour : % 17
- Bag of Words - Linear SVM : % 19

At the beginning, I was expecting to get better result with BoW but didn't end up like that. There might be several reasons causing that. Since the data set creation can effect the accuracy of the methods different type of train and test separation might lead to a better accuracy.

Another possible reason for the unexpected result might be the type of Kmeans method I used. MiniBatchedKMeans use smaller batches comparing to the regular KMeans method, so that might be the case.

On the other hand in classification side, I can say, I get what I expected. I was expecting to get a better result with SVM and according to the results, I can say that.

In general other possible misguiding cause might be categories that have similar to each other. For example Office and Living Room have some type of similarities in some images and it may have been a problem.

In conclusion, although according to the results Tiny Image can be seen over BoW but I think with a better implementation, BoW would give a better result and as I mentioned before LinearSVM is better choice for classification than K-Nearest Neighbour.

6 References

[1] https://www.kaggle.com/pierre54/bag-of-words-model-with-sift-descriptors
[2] https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python
[3] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
[4] https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html