Assignment 2

Berk Karaimer, 21827541
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
b21827541@cs.hacettepe.edu.tr

April 13, 2021

1 Introduction

In this assignment we are given different scene image datasets and asked to classify them using Tiny image feature with K-NN, Tiny image feature with SVM and we are given choise of bag of sift/surf/orb which i chose to use sift so, Bag of Sift with K-NN, Bag of Sift with SVM.

2 Experiment

2.1 Dataset Division

For dividing the dataset into train and test, i set a parameter as 100 so that the train and test would have 100 per each category. As can be seen from the code snippet in the following part, the parameter is sent to getimagepaths function and in there its used in a for loop

```
NUM_TRAIN_PER_CAT = 100
train_image_paths, test_image_paths, train_labels, test_labels =
get_image_paths(DATA_PATH, CATEGORIES, NUM_TRAIN_PER_CAT)
```

```
for category in categories:

        image_paths = glob(os.path.join(data_path, 'train', category, '*.jpg'))

        for i in range(num_train_per_cat):
            train_image_paths.append(image_paths[i])
            train_labels.append(category)

        image_paths = glob(os.path.join(data_path, 'test', category, '*.jpg'))
        for i in range(num_train_per_cat):
            test_image_paths.append(image_paths[i])
            test_labels.append(category)
```

2.2 Tiny Image and Bag of Words implementation

Tiny images is extracted from gettinyimages function which takes data path and simply resizes them. I choose to resize them to 16x16 and to improve the performance of code, i normalized the pictures.

```
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
```

To extract features with Bag of Visual Words i chose to use SIFT algortihm for keypoints and to implement the SIFT, i used cyvlfeat library for DSIFT. Firstly, i used buildvocabulary function with parameters of imagepaths and vocabulary size of 400. I chose the size as 400 since i wanted it to work accurately and quickly. Within the function, i set the step size to make the program work faster and fast parameter as true. After sampling SIFT descriptors, it uses kmeans to cluster and returns cluster centres. The following snippet is for vocabulary code part.

```
 def build_vocabulary(image_paths, vocab_size):

    bag_of_features = []

    for path in image_paths:
        img = np.asarray(Image.open(path),dtype='float32')
        frames, descriptors = dsift(img, step=[5,5], fast=True)
        bag_of_features.append(descriptors)
    bag_of_features = np.concatenate(bag_of_features, axis=0).astype('float32')
    vocab = kmeans(bag_of_features, vocab_size, initialization="PLUSPLUS")

    return vocab
```

Nextly, to acquire train and test image features, i used getbagofsifts function with parameters of image path and vocabulary which was returned from previous function. Within the function, i constructed the sift features here with different sampling rate in dsift function and set fast as true again which helps the performance again. After that local features are assigned to closest cluster centre using scipy.spatial.distance library. While computing the distance i've set Euclidean distance as the distance metric between the points. From here, the function builds a histogram which shows how many times the clusters were used and normalizes the histogram as well.

```
def get_bags_of_sifts(image_paths, vocab):

    image_feats = np.zeros((len(train_image_paths),len(vocab)))

    for i, path in enumerate(image_paths):

        image = np.asarray(Image.open(path), dtype = 'float32')
        frames, descriptors = dsift(image, step=[9,9], fast=True)

        dist = distance.cdist(vocab, descriptors, 'euclidean')
        mdist = np.argmin(dist, axis = 0)
        histo, bins = np.histogram(mdist, range(len(vocab)+1))
        if np.linalg.norm(histo) == 0:
            image_feats[i, :] = histo
        else:
            image_feats[i, :] = histo / np.linalg.norm(histo)
    return image_feats
```

2.3 K-NN and SVM implementation

For K-NN, Firstly, the K-NN function is called as can be seen in the following snippet. The K-NN function takes train image features, labels and test image features which are generated from Tiny image function or Bag of SIFT function.

```
predicted_categories = nearest_neighbor_classify(train_image_feats,
train_labels, test_image_feats)
```

Secondly, if the K-NN function is called, it will predict the category of the given image by finding most similar training image. I have set the k parameter as 13 in K-NN to increase the performance of program. I have used distance.cdist function to find the distance between two list of features. At the end of K-NN it gives result of images predicted category list.

```
def nearest_neighbor_classify(train_image_feats, train_labels, test_image_feats)
    k = 13
    test_predicts = []
    dist = distance.cdist(train_image_feats, test_image_feats, 'euclidean')
    for i in range(dist.shape[1]):
        ans = np.argsort(dist[:,i])
        nn = dict()
        #print(ans)
        for j in range(k):
            if train_labels[ans[j]] in nn.keys():
                nn[train_labels[ans[j]]] += 1
            else :
                nn[train_labels[ans[j]]] = 1

        snn = sorted(nn.items(), key = operator.itemgetter(1), reverse=True)
        test_predicts.append(snn[0][0])

    return test_predicts
```

For SVM, Firstly, SVM function is called as seen from the following snippet. SVM function takes same inputs as K-NN which are again generated by Tiny image or Bag of SIFT.

```
predicted_categories = svm_classify(train_image_feats,
train_labels, test_image_feats)
```

Secondly, Within the SVM function, it train set of linear SVMs for classification and use the learned linear classifiers to predict the category of test images. C parameter is the penalty of svm classifier, decreasing c helps with regularization and higher c helps with correct classification. Kernel's default is rbf, Gamma sets how much influence a single training example has so if the gamma is large then the other images will be affected more. estimator is for scoring. GridSearchCV function takes param grid as dictionary to try these values given to it. Scoring is to evaluate the performance of the cross-validated model on the test set. With these done, the fit function is used to train the classifier. Best estimator is the estimator that gave highest score and with that done the classifier is trained again. Lastly, the predicted label is returned with predict function with parameter of test image features.

```
def svm_classify(train_image_feats, train_labels, test_image_feats):
    svc = SVC(random_state=0)
    param_C = [0.001 , 0.01 , 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]
    param_gamma = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0]
    param_grid = [{'C': param_C,
                   'gamma': param_gamma,
                   'kernel': ['rbf']}]

    gs = GridSearchCV(estimator = svc,
                      param_grid= param_grid,
                      scoring='accuracy',
                     )
    gs = gs.fit(train_image_feats, train_labels)

    classifier = gs.best_estimator_
    classifier.fit(train_image_feats, train_labels)


    pred_label = classifier.predict(test_image_feats)
    return pred_label
```

2.4 Accuracy

Accuracy of Tiny Image Feature and K-Nearest Neighbor, Bag of Visual Words and K-Nearest Neighbor, Tiny Image Feature and Linear SVM, Bag of Visual Words and Linear SVM are all calculated as the following code snippet. It takes the tested image labels and compares with predicted category of it and then divides the length of accurate ones with all test images length. This metric is called Classification Accuracy. I've used this metric since there were only 100 train images and 100 test images for every category i have used.

```
accuracy = float(len([x for x in zip(test_labels,predicted_categories)
if x[0]== x[1]]))/float(len(test_labels))
print("Accuracy of the nearest neighbour classifier using Tiny image features is
,accuracy)
test_labels_ids = [CATE2ID[x] for x in test_labels]
predicted_categories_ids = [CATE2ID[x] for x in predicted_categories]
train_labels_ids = [CATE2ID[x] for x in train_labels]
```

2.5 Plot confusion matrices

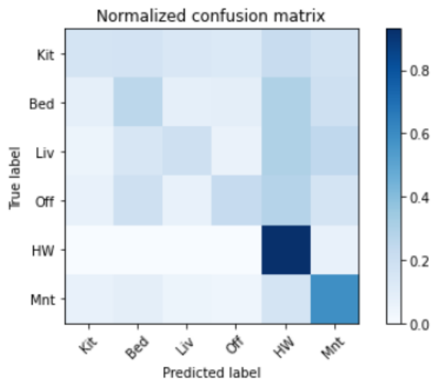

Figure 1: K-NN TINY

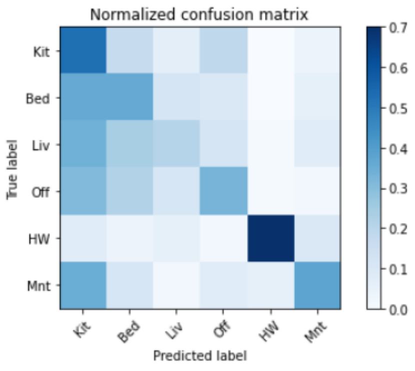

Figure 2: SVM TINY

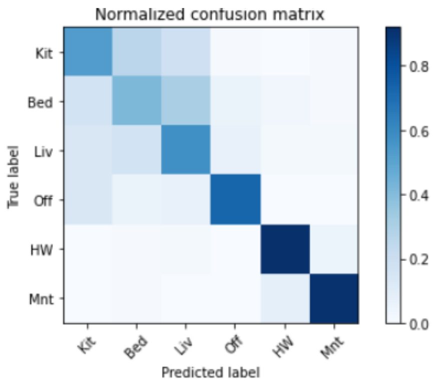

Figure 3: SVM SIFT

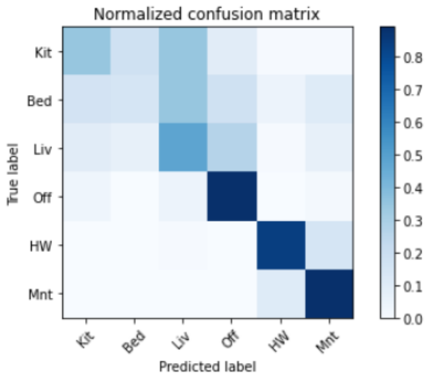

Figure 4: K-NN SIFT

2.6 Correct Samples

True positive results of each category as examples using k-nn with tiny, svm with tiny, SIFT with svm and SIFT with knn:
in kitchen category: 963, 968, 967, 967
in bedroom category: 197, 198, 196, 196
in living room category: 1177, 1176, 1177, 1177
in office category:4152, 4154, 4154, 4155
in highway category:2155, 2153, 2155, 2154
in mountain category:2721, 2721, 2723, 2723

2.7 Incorrect Samples

Here are some false negative and false positive examples for all four methods. They might be failing for several reasons. The parameters given for Feature detection and classification functions might not work for all the images and could end up failing. The images might not be good enough for the given functions to classify them, lighting in the image, saturation, etc.
in kitchen category: 968, 966, 968, 968
in bedroom category: 200, 200, 200, 200
in living room category: 1178, 1178, 1157, 1147
in office category:4155, 4155, 4155, 4131
in highway category:2144, 2155, 2124, 2155
in mountain category:2722, 2723, 2669, 2696

2.8 Comparisons

Accuracy of the nearest neighbour classifier using Tiny image features is 0.3933333333333333
Accuracy of the SVM classifier using Tiny image features is 0.41833333333333333
Accuracy of the SVM classifier using SIFT features is 0.6816666666666666
Accuracy of the KNN classifier using SIFT features is 0.6

Overall, from confusion matrices, highway seem accurate for all on the diagonal but Tiny image seems not to work as good as SIFT method. SIFT with SVM seems like it performs better from accuracy and the diagonal from the confusion matrix for the train set and test set i have given for the program.