Mini Project 3 Report

Made By: Omar Yasser and Mohamad Saleh

A) Tiny image Representation and the nearest neighbor classifier:

In this approach each image is cropped to have the same dimension value for both rows and columns. Then each image is resized to a much smaller size. The size used in our case was 16*16. To compute a vector representing each image the array of each image is flattened.

```
images_vectors = []
for image_path in image_paths:
    image = imread(image_path, as_gray=True)
    # print(image.shape)
    r, c = image.shape
    if r != c:  # cropping
        if r > c:
            image = image[r - c:r, :]
            # print(image.shape)
        else:
            image = image[:, c - r:c]
            # print(c-r-1)
            # print(image.shape)
    # npimage = np.array(image)
    resized_image = resize(image, (16, 16))
    # print (resized_image.ravel().shape)
    images_vectors.append(resized_image.ravel())
#print(np.array(images_vectors).shape)
return np.array(images_vectors)
```

This approach was used along with a simple nearest neighbor classifier in which the difference between each test image vector and all the train images vector is computed. The smallest difference train image’s label will be the test image label.

```
distances = cdist(test_image_feats, train_image_feats, 'euclidean')

# TODO:
# 1) Find the k closest features to each test image feature in euclidean space
# 2) Determine the labels of those k features
# 3) Pick the most common label from the k
# 4) Store that label in a list
n, d1 = test_image_feats.shape
m, d2 = train_image_feats.shape
l = len(train_labels)
test_labels = []
intervals_value = m / l
for i in range(n):
    minDistance = 1000
    minDistanceIndex = -1
    for j in range(m):
        distance = distances[i, j]
        if (distance < minDistance):
            minDistance = distance
            minDistanceIndex = j
    test_labels.append(train_labels[round(minDistanceIndex / intervals_value)])
```

The accuracy of this approach is low ranging from 15-25%

Our accuracy is about 10%

```
C:\ProgramData\Anaconda3\envs\code\python.exe C:/Users/asus/Desktop/Spring_2020/ComputerVision/miniProject3/MiniProject3/code/main.py -f tiny_image -c nearest_neighbor
Getting paths and labels for all train and test data.
Using tiny_image representation for images.
Loading tiny images...
Tiny images loaded.
Using nearest_neighbor classifier to predict test set categories.
Creating results_webpage/index.html, thumbnails, and confusion matrix.
Accuracy (mean of diagonal of confusion matrix) is 19.867%
Wrote results page to results_webpage/index.html.
```

B) Bag Of Words Representation and the LinearSVM Classifier:

In this approach the code runs through all the Train images to obtain the vocab that could represent the images.
This was done using a K means function. The function divided the data to the number of vocab desired. We used another function named minibatches K mean as it is much faster.

```
cellsPerBlock = 4
pixelsPerCell = 4
imageSize = 250
the_short_hogs = []
unClusteredVocab = []
# vocab_of_all_images=[]
for image_path in image_paths:
    image = imread(image_path, as_gray=True)
    # print(image.shape)
    r, c = image.shape
    if r != c:  # cropping
        if r > c:
            image = image[r - c:r, :]
            # print(image.shape)
        else:
            image = image[:, c - r:c]
            # print(c-r-1)
            # print(image.shape)
    # npimage = np.array(image)
    resized_image = resize(image, (imageSize, imageSize))
```

```
        the_long_hog = hog(resized_image, cells_per_block=(cellsPerBlock, cellsPerBlock),
                           pixels_per_cell=(pixelsPerCell, pixelsPerCell))
        the_short_hog = the_long_hog.reshape(-1, cellsPerBlock * cellsPerBlock * 9)
        for item in the_short_hog:
            unClusteredVocab.append(item)

        #   print(the_short_hog.shape)

        print(len(unClusteredVocab))
    # print("A")

    # vocab_of_image = KMeans(n_clusters=vocab_size,max_iter=10,tol=1).fit(the_short_hog)
    # print("The clusters center =")
    # print(vocab_of_image.cluster_centers_.shape)
    # print("End")
    # vocab_of_all_images.append(vocab_of_image.cluster_centers_)
    # unClusteredVocab = np.array(unClusteredVocab)
    # unClusteredVocab = unClusteredVocab.reshape(-1, cellsPerBlock * cellsPerBlock * 9)
    npUnClusteredVocab = np.array(unClusteredVocab)
    vocabOfAllImages = MiniBatchKMeans(n_clusters=vocab_size, max_iter=30000,tol=0.000001).fit(npUnClusteredVocab)
    print(vocabOfAllImages.cluster_centers_.shape)
    return np.array(vocabOfAllImages.cluster_centers_)
```

After obtaining the vocab, a histogram is computed for all the training and test images. To have even images with the same size each image is cropped and resized to the desired shape.

```
vocab = np.load('vocab.npy')
print('Loaded vocab from file.')

# TODO: Implement this function!
hist = []
cellsPerBlock = 4
pixelsPerCell = 4
imageSize = 250
for image_path in image_paths:
    image = imread(image_path, as_gray=True)
    # print(image.shape)
    r, c = image.shape
    if r != c:  # cropping
        if r > c:
            image = image[r - c:r, :]
            # print(image.shape)
        else:
            image = image[:, c - r:c]
            # print(c-r-1)
            # print(image.shape)
    # npimage = np.array(image)
    resized_image = resize(image, (imageSize, imageSize))
```

```
        resized_image = resize(image, (imageSize, imageSize))
        # print (resized_image.ravel().shape)
        the_long_hog = hog(resized_image, cells_per_block=(cellsPerBlock, cellsPerBlock),
                           pixels_per_cell=(pixelsPerCell, pixelsPerCell))
        the_short_hog = the_long_hog.reshape(-1, cellsPerBlock * cellsPerBlock * 9)
        distances = cdist(the_short_hog, vocab, 'euclidean')
        vocabRow, vocabCol = vocab.shape
        image_hist = np.zeros(vocabRow)
        for i in range(len(the_short_hog)):
            minDistance = 1000
            minDistanceIndex = -1
            for j in range(vocabRow):
                distance = distances[i, j]
                if (distance < minDistance):
                    minDistance = distance
                    minDistanceIndex = j
            image_hist[minDistanceIndex] += 1
        hist.append(image_hist)

    return np.array(hist)
```

A Linear support vector machine classifier is used later to determine at which region does the test image falls into and depending on that label of the test image is determined.

```
    # TODO: Implement this function!
    linearSVM = LinearSVC(tol=1e-8).fit(train_image_feats,train_labels)
    return np.array(linearSVM.predict(test_image_feats))
```

To obtain satisfactory results with high percentage a lot of variable could change such as the number of pixels in each cell, the numbers of cells in each block could be decreased increasing the number of blocks the image is cut into. Also, the vocab number could be increased. The K means iterations could be increased and its tolerance could be decreased. Most of these options were put in consideration during testing the code but to achieve better result we will a duration much grater than 10 minutes.

a Result of 56% was obtained by setting the image size to 250, the Cells per block to 4, number of pixels per cell 4, iterations to 1000, and tolerance to 0.000001:

```
C:\ProgramData\Anaconda3\envs\code\python.exe C:/Users/asus/Desktop/Spring_2020/ComputerVision/miniProject3/MiniProject3/code/main.py -f bag_of_words -c support_vector_machine
Getting paths and labels for all train and test data.
Using bag_of_words representation for images.
Loaded vocab from file.
Loaded vocab from file.
Using support_vector_machine classifier to predict test set categories.
Creating results_webpage/index.html, thumbnails, and confusion matrix.
Accuracy (mean of diagonal of confusion matrix) is 56.200%
Wrote results page to results_webpage/index.html.

Process finished with exit code 0
```

To obtain this result it took lots of time to compute the vocabulary and to compute the histogram of the images. The only parameters which we did not change that could make a desired effect is the vocab since it is defined in the main file.

A result of 47% is obtained by setting the image size to 200, the Cells per block to 8, Number of pixels per cell to 8, iteration to 1000 and tolerance to 0.000001 it was obtained in less than 15 minutes.

```
(200, 576)
Loaded vocab from file.
Loaded vocab from file.
Using support_vector_machine classifier to predict test set categories.
Creating results_webpage/index.html, thumbnails, and confusion matrix.
Accuracy (mean of diagonal of confusion matrix) is 47.467%
Wrote results page to results_webpage/index.html.
```

A result of 39% is obtained by setting size to 200, the Cells per block to 9, the number of pixels per cell to 9, iteration to 30,000 and tolerance to 0.000001. It was obtained in less than 10 minutes.

```
(200, 729)
Loaded vocab from file.
Loaded vocab from file.
Using support_vector_machine classifier to predict test set categories.
Creating results_webpage/index.html, thumbnails, and confusion matrix.
Accuracy (mean of diagonal of confusion matrix) is 39.867%
Wrote results page to results_webpage/index.html.
```