



---

# SCENE RECOGNITION

---

Computer Vision



**Sara El besomy    201601849**

**SPRING 2020**

**ZEWAIL UNIVERSITY OF SCIENCE & TECHNOLOGY**

## Objective:

Performing scene recognition with three different methods. I will classify scenes into one of 15 categories by training and testing on the 15 scene database.

Implement three methods:

- Tiny images representation (get\_tiny\_images()) and nearest neighbor classifier (nearest\_neighbor\_classify()).
- Bag of words representation (build\_vocabulary(), get\_bags\_of\_words()) and nearest neighbor classifier.
- Bag of words representation and linear SVM classifier (svm\_classify()).

## Implementation details:

### 1. Get\_tiny\_images ( ) Function:

```
def get_tiny_images(image_paths):
    tiny_images = []
    new_size = 16
    for img in image_paths:
        image = imread(img, as_gray=True)
        image = resize(image, (new_size, new_size))
        image = (image - np.mean(image)) / np.std(image)
        image = image.flatten()
        tiny_images.append(image)

    tiny_images = np.asarray(tiny_images)

    return tiny_images
```

#### Inputs:

Image\_paths : a 1-D Python list of strings. Each string is a complete path to an image on the file system.

#### Outputs:

An n x d numpy array where n is the number of images and d is the length of the tiny image representation vector. e.g. if the images are resized to 16x16, then d is 16 \* 16 = 256.



**Description:**

In this function, the images are rescaled into very small sizes (16x16) in order to represent the whole images with less amount of information. This is work because when an image is rescaled to be such small, we keep the low frequencies only and lose the high frequencies, which is useless information in our case (we only need to represent the image).

**Implementation in steps:**

1. Read the images and convert them to grayscale.
2. Resize the images to be 16x16 pixels.
3. Normalize the tiny images by subtracting the mean from each pixel and divide be the standard deviation.
4. Put all images in one list and return the images list.

**2. Build\_vocabulary () Function:**

```
def build_vocabulary(image_paths, vocab_size):
    All_features_vectors = []
    for img in image_paths:
        image = imread(img)
        image = resize(image, (128, 64))
        features_vector = hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=False,
                               feature_vector=True, multichannel=None)
        features_vector = features_vector.reshape(-1, 2*2*9)
        All_features_vectors.append(features_vector)

    All_features_vectors = np.vstack(All_features_vectors)
    print(All_features_vectors.shape)
    kmeans_clf = KMeans(n_clusters=vocab_size, max_iter=100).fit(All_features_vectors)
    vocab = kmeans_clf.cluster_centers_

    return vocab
```

**Inputs:**

Image\_paths : a Python list of image path strings.

Vocab\_size : an integer indicating the number of words desired for the bag of words vocab set.

**Outputs:**

A vocab\_size array, which contains the cluster, centers that result from the K Means clustering.



**Description:**

In this function, we build the vocabulary bag that will compare the feature vector with to build the histogram.

**Implementation in steps:**

1. Read the images.
2. Resize them to smaller size to be less time consuming. (I set the new size to be height = 2 x width)
3. Extract the features from the image using hog () function in skimage library.
4. Reshape the feature vector from one vector to an array with size (z\*z\*9)  
Where z is the size of cells\_per\_block and 9 is the number of bins in the histogram created by hog () function
5. Append all the feature vectors from the images in one list.
6. Concatenate the first two dimensions of the output feature vectors to make it 2D array.
7. Apply kmeans clustering on the output feature vectors to determine a specific number of clusters.

**3. Get\_bags\_of\_words( ) Function:**

```
def get_bags_of_words(image_paths):
    vocab = np.load('vocab.npy')
    print('Loaded vocab from file.')

    vocab_len = vocab.shape[0]
    histograms = np.zeros((len(image_paths), vocab_len))

    for i, img in enumerate(image_paths):
        image = imread(img)
        image = resize(image, (128, 64))
        features_vector = hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=False,
                              feature_vector=True, multichannel=None)
        features_vector = features_vector.reshape(-1, 2 * 2 * 9)
        histo = np.zeros(vocab_len)
        distances = cdist(features_vector, vocab)
        closest_vocab = np.argsort(distances, axis=1)[: , 0]
        index, count = np.unique(closest_vocab, return_counts=True)
        histo[index] += count
        histo = histo / np.linalg.norm(histo)
        histograms[i] = histo

    return histograms
```



**Inputs:**

Image\_paths : a Python list of image path strings.

**Outputs:**

An nxd numpy matrix, where n is the number of images in image\_paths and d is size of the histogram built for each image.

**Description:**

In this function, we extract a feature vector from every image in our set and compare it with our vocabularies to calculate the histogram of the image features.

**Implementation in steps:**

1. Load the vocabs file.
2. Read the image.
3. Resize the image to smaller size.
4. Use hog () function to extract features from the image.
5. Calculate the distance between each feature vector for each image and all the vocabs words.
6. Get the closest vocab word to the feature vector.
7. Make the histogram of the features vectors for all images.

#### 4. Svm\_classify () Function:

```
def svm_classify(train_image_feats, train_labels, test_image_feats):  
    clf = LinearSVC(random_state=0, tol=1e-5)  
    clf.fit(train_image_feats, train_labels)  
    predictions = clf.predict(test_image_feats)  
  
    return predictions
```

**Inputs:**

train\_image\_feats: An nxd numpy array, where n is the number of training examples, and d is the image descriptor vector size.

train\_labels: An nx1 Python list containing the corresponding ground truth labels for the training data.

test\_image\_feats: An mxd numpy array, where m is the number of test images and d is the image descriptor vector size.



**Outputs:**

An mx1 numpy array of strings, where each string is the predicted label for the corresponding image in test\_image\_feats

**Description:**

In this function, we apply linear support vector machine using scikit learn module on the training and test data to classify the images according to their feature vectors.

**5. Nearest\_neighbor\_classify () Function:**

```
def nearest_neighbor_classify(train_image_feats, train_labels, test_image_feats):
    categories = np.unique(train_labels)
    predicts = []

    k = 1
    distances = cdist(test_image_feats, train_image_feats, 'euclidean')
    for d in distances:
        labels = []
        index = np.argsort(d)
        for i in range(k):
            labels.append(train_labels[index[i]])

        amount = 0
        for item in categories:
            if labels.count(item) > amount:
                label_final = item

        predicts.append(label_final)

    return predicts
```

**Inputs:**

train\_image\_feats: An nxd numpy array, where n is the number of training examples, and d is the image descriptor vector size.

train\_labels: An nx1 Python list containing the corresponding ground truth labels for the training data.

test\_image\_feats: An mxd numpy array, where m is the number of test images and d is the image descriptor vector size.



**Outputs:**

An mx1 numpy list of strings, where each string is the predicted label for the corresponding image in test\_image\_feats.

**Description:**

In this function, we use KNN as a classifier for the feature vectors of both training and testing data.

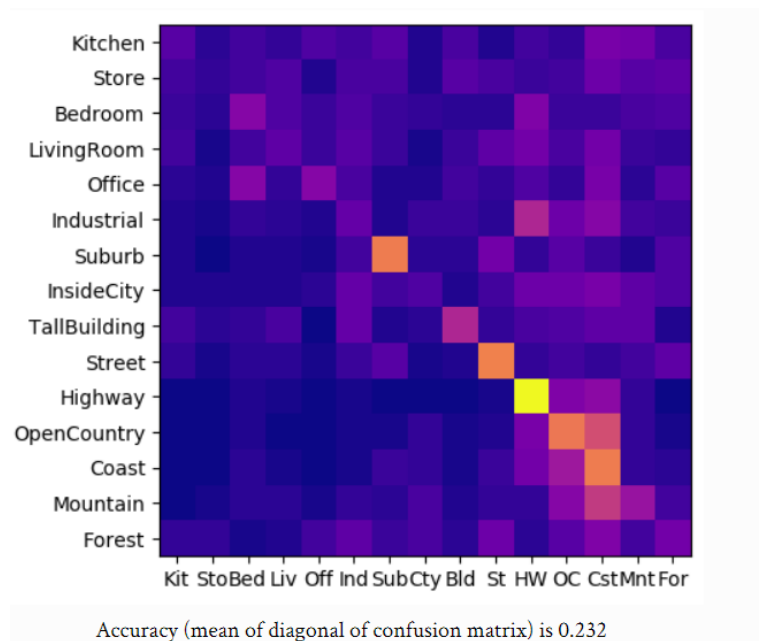
**Results:**

- Tiny images and KNN:**

We use the tiny image function as a feature extraction method and KNN as a classifier.

**Accuracy:** 23.2 %

**Confusion matrix:**



### Samples of the results:

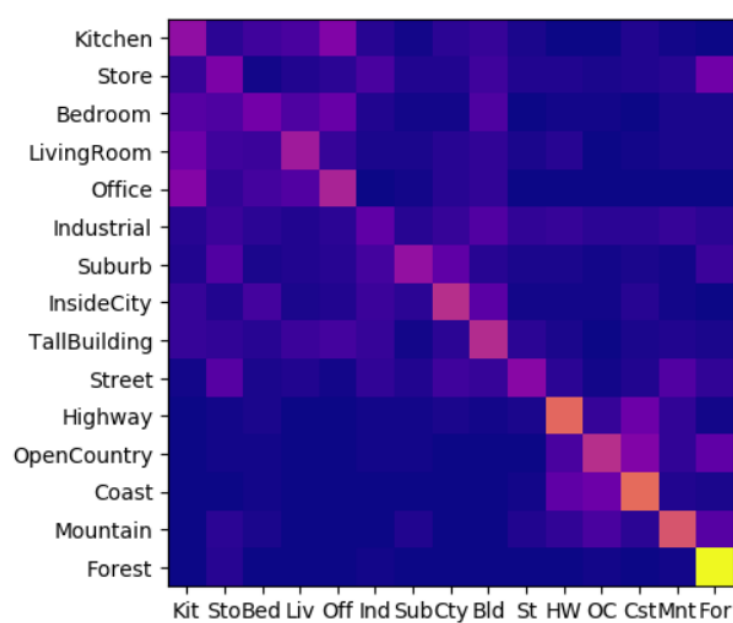
Category name	Accuracy	Sample training images	Sample true positives	False positives with true label	False negatives with wrong predicted label
Kitchen	0.090			 LivingRoom TallBuilding	 Office Industrial
Store	0.040			 Office TallBuilding	 Coast TallBuilding
Bedroom	0.160			 Industrial Office	 Kitchen Forest

- Bag of words and KNN:**

We here use bag of words as a feature extraction method and KNN as a classifier.

**Accuracy: 39 %**

**Confusion matrix:**




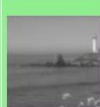














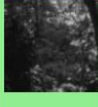







Accuracy (mean of diagonal of confusion matrix) is 0.390





### Sample of the results:

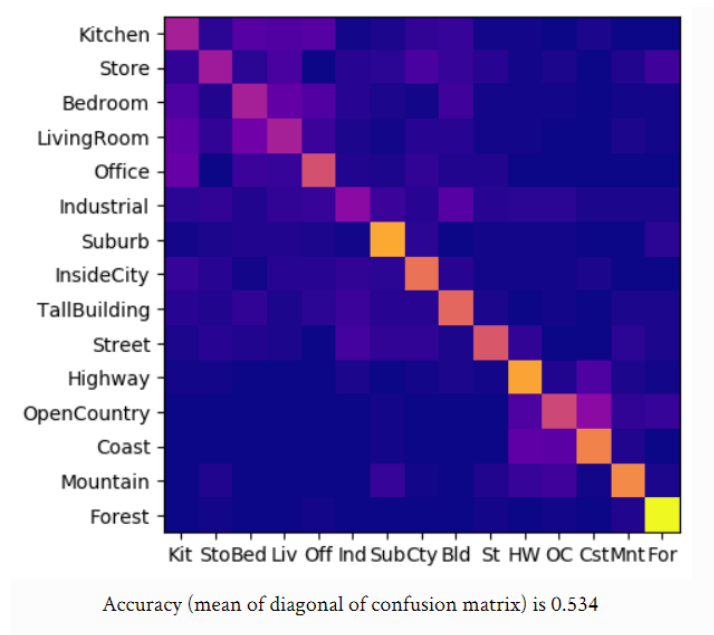
Coast	0.580								
						OpenCountry	InsideCity	Street	Highway
Mountain	0.510								
						Highway	Store	Street	Street
Forest	0.930								
						Store	Mountain	Store	Mountain
Category name	Accuracy	Sample training images		Sample true positives		False positives with true label		False negatives with wrong predicted label	

- Bag of words and SVM:**

























We use bag of methods as a features extraction method and Support vector machine as a classifier.

**Accuracy: 53.4 %**

**Confusion matrix:**



### Sample of results:

Category name	Accuracy	Sample training images	Sample true positives	False positives with true label	False negatives with wrong predicted label
Street	0.520	 	 	 Mountain  Industrial	 LivingRoom  Mountain
Highway	0.740	 	 	 Kitchen  Coast	 Mountain  Coast
OpenCountry	0.470	 	 	 Mountain  Coast	 Coast  Coast

### Reference:

<https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/>

