ECE 661 - Computer Vision
Homework - 7

Due on Tuesday, 30th October 2018

Naveen Madapana

Homework Description

This homework involves implementing Local Binary Pattern (LBP) algorithm which is prominently used to create texture-based features of images. Next, K - Nearest Neighborhood (KNN) classifier is implemented to identify the class labels of images. The LBP features serve as input features for KNN classifier.

1. LBP Feature Extraction

Local Binary Pattern (LBP) is a texture-based feature extraction technique which is used to compute translation and rotation invariant features in grayscale images. Dr. Avinash Kak's tutorial on *Modelling Texture and Color in Images* [1] is taken as a reference to implement the LBP algorithm. The main idea behind LBP lies in the following: 1. *Describe inter-pixel variations with a binary pattern*, 2. *Create a rotation-invariant representation for each pixel* and 3. *Encode an image through a histogram of binary patterns.*

1.1 Create binary patterns for inter-pixel variations

First step in creating the binary patterns involves defining the neighborhood of each pixel X. The neighborhood is defined by two parameters: radius R of the circle and the number of points P on the circle.

The equation to compute the relevant points (neighborhood points) on the circle are given in terms of perturbations in x direction (Δu) and y direction (Δv). The coordinates of the points on the circumference of the circle will be obtained by adding these perturbations to the pixel coordinates of the center pixel.

$$(\Delta u, \Delta v)_p = (Rcos(\frac{2\pi p}{P}), Rsin(\frac{2\pi p}{P})), \quad with\ p = 0, 1, 2, 3, \ldots, P-1$$

Let A be the center pixel with coordinates (x, y). Then, the neighboring points on the circle will be given by $x + \Delta v, y + \Delta u$. While $p = 0$ corresponds to the bottom pixel, $p = 4$ corresponds to the top pixel w.r.t the center pixel A. Since the points on the circle may not exactly correspond to the centers of the neighboring pixels, we need to use bi-linear interpolation to estimate the pixel intensities of the extracted neighboring points.

Let A be the intensity of center pixel, B be the intensity of neighboring pixel on x axis, C be the intensity of the neighboring pixel on y axis and D be the intensity of the that is on the diagonal. A, B, C and D are chosen in such a way that the point p lies in the rectangle formed by joining the pixel centers of these points. The estimated intensity value (I_p) of point p using bi-linear interpolation is given below.

$$I_p = (1 - \Delta u)(1 - \Delta v)A + (1 - \Delta u)\Delta vB + \Delta u(1 - \Delta v)C + \Delta u \Delta vD$$

For $P = 8$, we have eight neighbors. The intensity value is computed for each of these P neighbors. Next, these intensity values are thresholded w.r.t the intensity value of the center pixel to obtain a binary pattern, i.e. if $I_p \geq A$, indicates a value of one and zero otherwise. In this way, we create a binary vector for each pixel in the image.

In more words, we will do rastor scan on the image. At every pixel, we will consider the neighboring points on the circle with radius R and then compute the binary vector for each pixel.

1.2 Rotation invariance in LBP

Given the binary pattern (a sequence of zeros and ones), the goal is to create rotation invariant patterns. The intuition behind this lies in the fact that the perception of an image does not greatly depend on the orientation in which we look at the image.

In this regard, the binary sequence is circularly shifted to obtain minimum integer when the binary sequence is converted into a decimal value. This would lead to a binary sequence that has the greatest number of zeroes in the significant bits. For example, consider the initial binary pattern ([1 1 0 0] = 12) and the resulting pattern that gives a minimum integer value ([0 0 1 1] = 3).

In this homework, the BitVector module [1] and own implementations were used to perform the circular shifts to obtain the binary sequence corresponding to minimum integer values.

1.3 Image Encoding

Next, the rotation invariant binary pattern is encoded by a single integer ranging from 0 to $P+2$. The order of the binary pattern is used for encoding. The steps are given below.

First, the number of runs of the binary pattern is determined. The no. of runs is quantifies the transitions between zeros and ones in the rotation invariant binary sequence. In this homework, the BitVector module [1] was used to obtain the number of runs.

1. $Nr = get\_runs(binary\_sequence)$
2. If sequence has only zeros, encoding = 0
3. If sequence has only ones, encoding = P
4. If Nr = 2, encoding = No. of ones in the sequence
5. If Nr $>$ 2, encoding = P+1

For each pixel, an encoding value is computed using the procedure described above. Once we have the encoding values for all pixels, a normalized histogram consisting of $P+2$ bins $(0, 1, 2, \ldots P+1)$ was computed. This normalized histogram values will act as the feature vector of the image.

It is invaluable that the length of the feature vector is invariant to the size of the image.

2. K-Nearest Neighbours Algorithm (KNN)

KNN is a supervised classifier that require both the input images/ features and their corresponding class labels. KNN stores the input training information (features and their labels) to predict the class labels of unseen examples during the testing period. Hence the memory complexity is linear w.r.t the size of training data.

KNN requires a distance function to evaluate the closeness between two feature vectors. Prominently used distance functions include Euclidean distance, cosine distance and dot product. In this homework, all three distance metrics were implemented and the resulting confusion matrices are compared against each other.

In this homework, we have five categories of images: 1. building, 2. car, 3. beach, 4. tree and 5. mountain. Each category has 20 image examples in the training set. However, in testing set, there are five image examples corresponding to each class.

The LBP histogram computed in the previous section acts as the input feature for the KNN Classifier. The dimension of feature vector is $P+2$ i.e. 10 in our case. Note that the dimension of the feature is independent of the the size of the image.

There is no training as such for KNN. During the testing period, the new instance is compared with each of the training instances using the distance metrics. The class labels corresponding to top K training instances that have smallest distance were considered to determine the final class label. Next, among the K class labels, the class label that has the majority is chosen as the class label for the test instance.

Let $|X|$ be the norm the vector X. The Euclidean distance between the vectors X and Y is given by $(X-Y)^T(X-Y)$. Next section presents the KNN results for several distance functions and values of K.

3. Results

The LBP histogram of a randomly chosen example image from each training class is presented below in figures 1 and 2. For each case the value of $R = 1$ and $P = 8$. Figure 1 shows these histograms. The parameters used for LBP where: $P = 8$ and $R = 1$.

The following results (Figure 3) show the confusion matrices for $K = 5$ using the euclidean and cosine distance $(1 - cos(\theta))$ metrics.

It can be clearly seen that the histogram is clearly different each class. Further, the results obtained using Euclidean distance metric is superior in comparison to the Cosine distance metric. Also, there is no difference in the accuracy (refer to the confusion matrices 3, 4 and 5). The overall accuracy using Euclidean metric is 64% and by using cosine metric is 56%.

(a) Building image

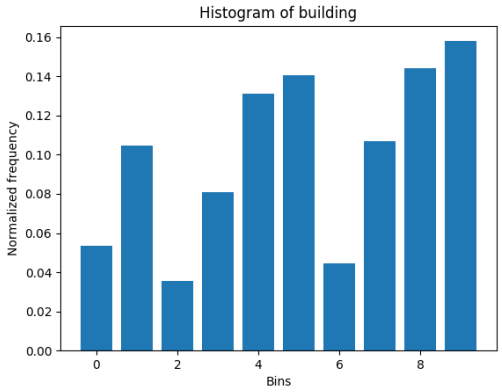
(b) Beach image

(c) Car image

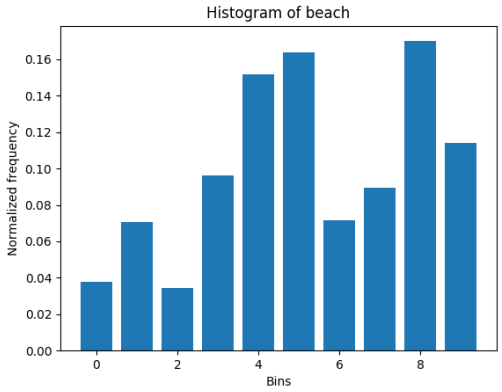
(d) Mountain image

(e) Tree image

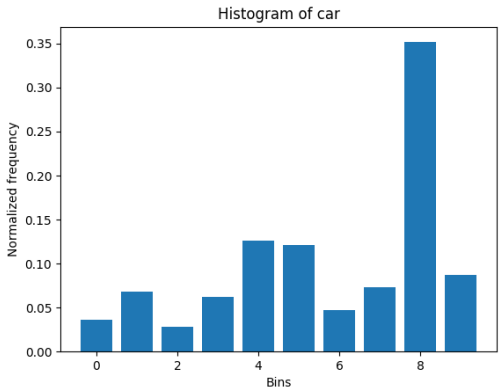
Figure 1: Training images for various classes



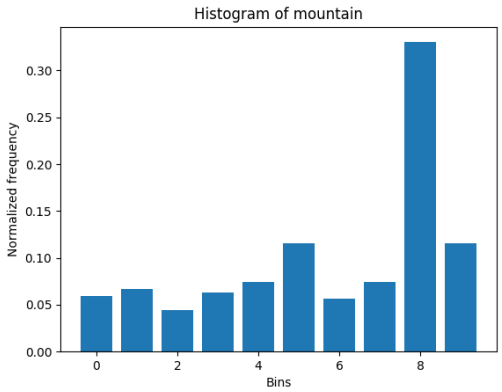
(a) Building image



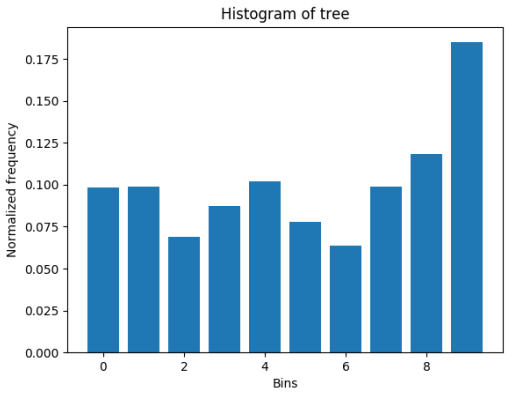
(b) Beach image



(c) Car image

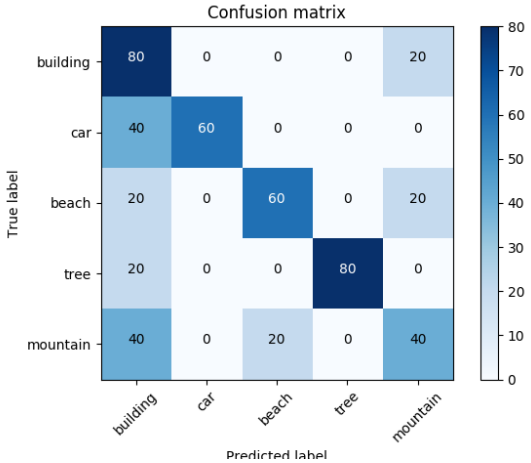


(d) Mountain image

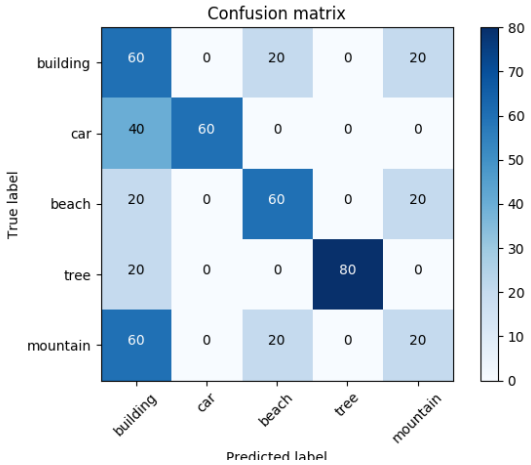


(e) Tree image

Figure 2: LBP histograms of training images for various classes

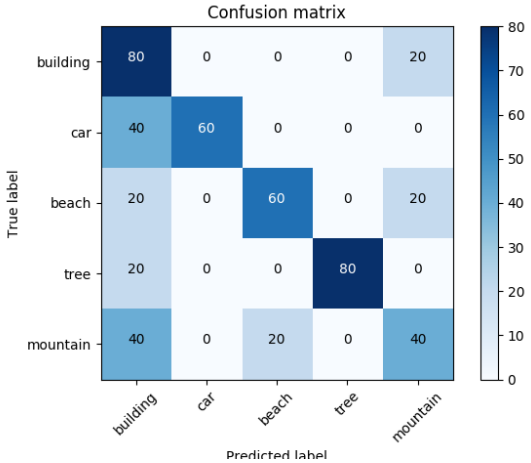


(a) Euclidean distance metric

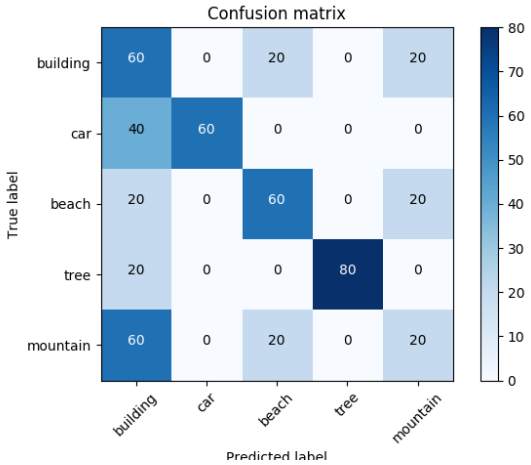


(b) Cosine distance metric

Figure 3: Confusion matrices using two distance metrics for K = 1.

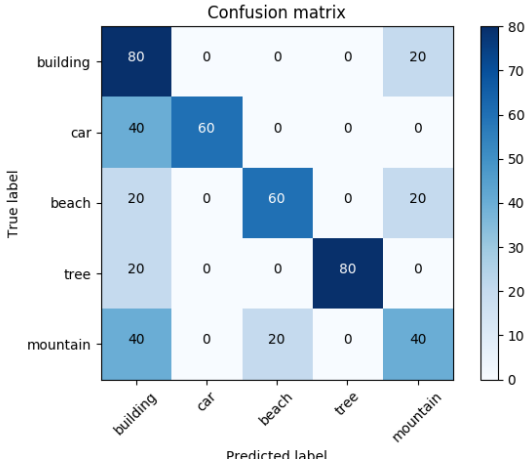


(a) Euclidean distance metric

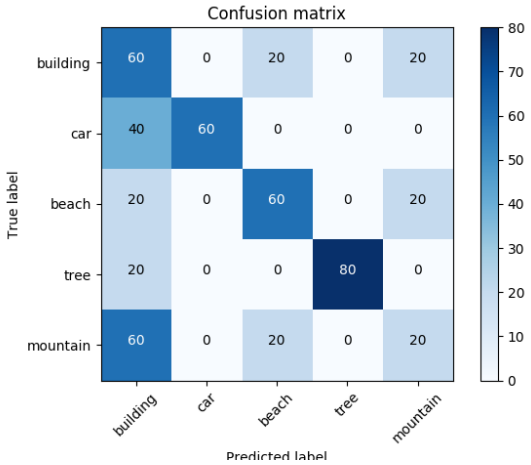


(b) Cosine distance metric

Figure 4: Confusion matrices using two distance metrics for K = 3.



(a) Euclidean distance metric



(b) Cosine distance metric

Figure 5: Confusion matrices using two distance metrics for K = 5.

4. Code

Listing 1: HW7 code

```
import os, sys, time
import numpy as np
import cv2
from os.path import join, basename, dirname, splitext
from copy import deepcopy
from BitVector import BitVector
from glob import glob

import pickle

def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix',cmap=plt.cm.Blu
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = 100 * (cm.astype('float') / cm.sum(axis=1)[:, np.newaxis])
        cm = cm.astype('int')
    else:
        print('Confusion matrix, without normalization')

    # print(cm)
    plt.figure()

    np.set_printoptions(precision=0)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = 'd' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center", \
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

def create_circ_points(radius = 1, num_neighbors = 8):
    '''
    Find out the points on the circle
    This function returns the x and y coordinates of points on a circle.
```

```
    '''
    R = radius
    P = num_neighbors

    x_lst = []
    y_lst = []
    for p in range(P):
        du = R*np.cos(2*np.pi*p/P)
        dv = R*np.sin(2*np.pi*p/P)
        if(abs(du) < 1e-4): du = 0.0
        if(abs(dv) < 1e-4): dv = 0.0
        y_lst.append(du)
        x_lst.append(dv)
    return x_lst, y_lst

def bilin_interp(A, B, C, D, dx, dy):
    # print A, B, C, D
    # dx is x-axis distance from A to the point
    # dy is y-axis distance from A to the point
    return (1-dy)*(1-dx)*A + (1-dy)*dx*B + dy*(1-dx)*C + dy*dx*D

def lbp_binvec(A):
    '''
    Input:
        * A: np.ndarray of size 3 x 3
    '''
    v1 = A[2, 1]
    v2 = bilin_interp(A[1,1], A[1,2], A[2,1], A[2,2], 0.707, 0.707)
    v3 = A[1, 2]
    v4 = bilin_interp(A[1,1], A[1,2], A[0,1], A[0,2], 0.707, 0.707)
    v5 = A[0, 1]
    v6 = bilin_interp(A[1,1], A[1,0], A[0,1], A[0,0], 0.707, 0.707)
    v7 = A[1,0]
    v8 = bilin_interp(A[1,1], A[1,0], A[2,1], A[2,0], 0.707, 0.707)
    ret = np.array([v1, v2, v3, v4, v5, v6, v7, v8])

    ret = ret >= A[1,1]

    return ret.astype(int).tolist()

def lbp_value(binvec, P = 8):
    '''
    Return the encoding the LBP pattern given in binvec
    '''
    bv = BitVector(bitlist = binvec)
    intvals =  [int(bv<<1) for _ in range(len(binvec))]
    minbv = BitVector(intVal = min(intvals), size = len(binvec))
    bvruns = minbv.runs()
    if(len(bvruns) > 2): return P + 1
    elif(len(bvruns) == 1 and bvruns[0][0] == '1'): return P
    elif(len(bvruns) == 1 and bvruns[0][0] == '0'): return 0
    else: return len(bvruns[1])

```

```
def get_lbp_hist(img_path):
    '''
    Return the LBP histogram given the path to an image (RGB or Grayscale)
    '''
    img = cv2.imread(img_path, 0)
    hist = [0]*(P+2)
    for x_idx in range(1, img.shape[1]-1):
        for y_idx in range(1, img.shape[0]-1):
            frame = img[y_idx-1:y_idx+2, x_idx-1:x_idx+2]
            binvec = lbp_binvec(frame)
            pvalue = lbp_value(binvec)
            hist[pvalue] += 1

    hist = np.array(hist).astype(float) / np.sum(hist)
    return hist

R = 1
P = 8

## Creating training data
print '================ Creating Training Data ================='
training_dir_path = 'Images\\training'
classnames = os.listdir(training_dir_path)
features = {cname: [] for cname in classnames}

for cname in classnames:
    print '---' + cname + '---'
    img_dir = os.path.join(training_dir_path, cname)
    img_paths = glob(os.path.join(img_dir, '*.jpg'))
    for img_path in img_paths:
        print os.path.basename(img_path),
        hist = get_lbp_hist(img_path)
        features[cname].append(hist)
    print ''
    features[cname] = np.array(features[cname])

with open('train_features.pickle', 'wb') as fp:
    pickle.dump(features, fp)

with open('train_features.pickle', 'rb') as fp:
    features = pickle.load(fp)

## Creating testing data
print '\n============= Creating Testing Data =============='
testing_dir_path = 'Images\\testing'
test_img_paths = glob(os.path.join(testing_dir_path, '*.jpg'))
out_features = {os.path.basename(test_img_path): None for test_img_path in test_img_paths}

for test_img_path in test_img_paths:
    print os.path.basename(test_img_path),
    out_feat_vec = get_lbp_hist(test_img_path)
    out_features[os.path.basename(test_img_path)] = out_feat_vec
```

```
with open('test_features.pickle', 'wb') as fp:
    pickle.dump(out_features, fp)

with open('test_features.pickle', 'rb') as fp:
    out_features = pickle.load(fp)


################
### MAIN #######
################

import pickle
import numpy as np
import os, sys, time
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

with open('train_features.pickle', 'rb') as fp:
    train_features = pickle.load(fp)

with open('test_features.pickle', 'rb') as fp:
    test_features = pickle.load(fp)

num_classes = len(train_features.keys())

cnames_to_ids = {cname: idx+1 for idx, cname in enumerate(train_features.keys())}
classnames = [cname for cname in train_features.keys()]

## Creating train data

train_input = None
train_output = None

for cname, data in train_features.items():
    # plt.bar(range(10), data[0,:])
    # plt.title('Histogram of ' + cname)
    # plt.xlabel('Bins')
    # plt.ylabel('Normalized frequency')
    # plt.show()
    cid = cnames_to_ids[cname]
    if(train_input is None): train_input = data
    else: train_input = np.append(train_input, data, axis = 0)
    out = cid * np.ones(data.shape[0])
    if(train_output is None): train_output = out
    else: train_output = np.append(train_output, out).astype(int)

def knn(train_input, train_output, test_inst, metric = 'euclidean', K = 5):
    # train_input: 2D np.ndarray
    # train_output: 1D np.ndarray. train instance labels.
    # test_inst: 1D np.ndarray. Size of vec is equal to no. of columns in M
    if(metric == 'euclidean'):
        dist = np.linalg.norm(train_input - test_inst, axis = 1)
    elif(metric == 'dot'):
```

```
        dist = -1 * np.sum(train_input * test_inst, axis = 1)
    elif(metric == 'cosine'):
        norm_train_input = train_input.transpose() / np.linalg.norm(train_input, axis = 1)
        norm_train_input = norm_train_input.transpose()
        norm_test_inst = test_inst / np.linalg.norm(test_inst)
        dist = 1 - np.sum(norm_train_input * norm_test_inst, axis = 1)

    argmin_ids = np.argsort(dist)[:5]
    c_argmin_ids = train_output[argmin_ids]

    all_class_ids = np.unique(train_output).astype(int)
    freqs = [0]*len(all_class_ids)
    for cid in c_argmin_ids:
        freqs[cid-1] += 1
    return np.argmax(freqs) + 1

test_pred_label = []
test_true_label = []

## Creating test data
for fname, test_inst in test_features.items():
    cname = os.path.splitext(fname)[0].split('_')[0]
    cid = cnames_to_ids[cname]

    ## Euclidean
    pred_label = knn(train_input, train_output, test_inst, metric = 'euclidean', K = 1)

    test_pred_label.append(pred_label)
    test_true_label.append(cid)

# print zip(test_true_label, test_pred_label)

conf_mat = confusion_matrix(test_true_label, test_pred_label)
print conf_mat
print 'Accuracy: %.02f'%(np.mean(np.diag(conf_mat/5.0)))

plot_confusion_matrix(conf_mat, classnames, normalize = True)
```

References

[1] Avinash Kak. *Measuring Texture and Color in Images*, 2016 (accessed November 08, 2016).