ECE 661: Homework 7

Xingguang Zhang

(Fall 2020)

Theory Question

1. The reading material for Lecture 15 presents three different approaches to characterizing the texture in an image: 1) using the Grayscale Co-Occurrence Matrix (GLCM); 2) with Local Binary Pattern (LBP) histograms; and 3) using a Gabor Filter Family. Explain succinctly the core ideas in each of these three methods for measuring texture in images.

GLCM: The basic idea of GLCM is to estimate the joint probability distribution $P[x_1, x_2]$ for the grayscale values in an image, where x_1 is the grayscale value at any randomly selected pixel in the image and x_2 the grayscale value at another pixel that is at a specific vector distance d from the first pixel.

LBP: LBP uses a local binary pattern to characterize the grayscale variations around a pixel through runs of 0s and 1s. The pattern is grayscale invariant and rotation invariant.

Gabor: Gabor filters aims to characterize the repetitively occurring micro-patterns by a highly localized Fourier transform. Applying the Fourier transform on different locations on an image, we can measure the periods and direction of the micro-pattern by analysing the associated frequence pattern.

2. With regard to representing color in images, answer Right or Wrong for the following questions:

(a). RGB and HSI are just linear variants of each other. – Wrong

(b). The color space L*a*b* is a nonlinear model of color perception. –Wrong

(c). Measuring the true color of the surface of an object is made difficult by the spectral composition of the illumination. –Right

Implementation steps

LBP Feature Extraction

Characterizing the Local Inter-Pixel Grayscale Variation

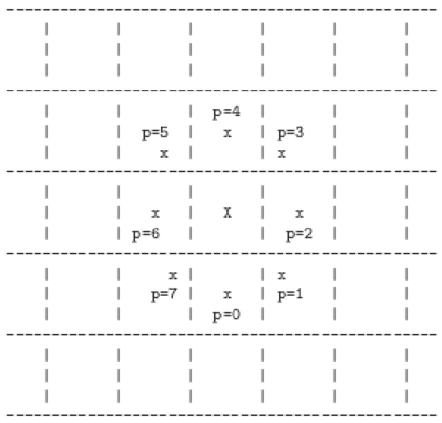
Consider the pixel at the location marked by uppercase 'X' in Fig. 1(a) below and its 8 neighboring points on a unit circle. The exact positions of the neighboring points vis-a-vis the pixel under consideration is given by

$$(\Delta u, \Delta v) = (R\,cos(\frac{2\pi p}{P}, R\,sin(\frac{2\pi p}{P}))) \qquad p = 0, 1, 2, ..., 7$$

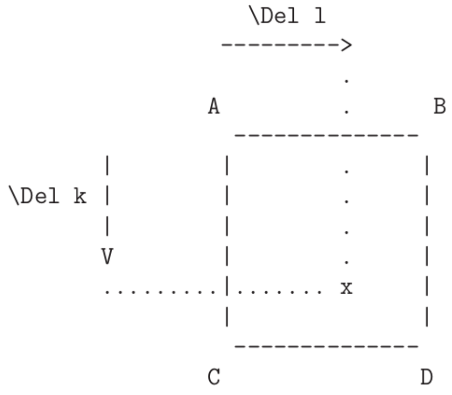
with the radius R = 1 and with P = 8. The point p = 0 gives us the neighboring point that is straight down from the pixel under consideration; the point p = 1, the neighboring point to the right of the one that is straight down; and so on.
We can denote the grayscale of the pixel on 'X' in the center of the circle as $g(0, 0)$, and it's 8 neighbors as $g(-1, -1)$, $g(-1, 0)$, $g(-1, 1)$, $g(0, -1)$, $g(0, 1)$, $g(1, -1)$, $g(1, 0)$, $g(1, 1)$. The grayscale

values at the circle points, of which we have 8 when $P = 8$, are denoted as $g(p_0) \sim g(p_7)$. We can estimate those gray scale by bilinear interpolation.
Let's say that the labels A, B, C, and D at the four corners of the rectangle shown at Fig.1(b) are the centers of four adjoining pixels and that we want to estimate through interpolation the gray level at the point marked 'x' inside the rectangle. Bilinear interpolation says that the gray level at point x can be approximated by

$$g(x) \approx (1-\Delta k)(1-\Delta l)A + (1-\Delta k)\Delta l B + \Delta k(1-\Delta l)C + \Delta k \Delta l D$$



(a) local pattern locations



(b) Bilinear interpolation

Figure 1: Illustration of local pattern

In the particular case when $R = 1$ and $P = 8$, we can calculate $g(p_0) \sim g(p_7)$ from the nine neighbors by transform matrix H, the formula is:

$$\begin{bmatrix} g(p_0) \\ g(p_1) \\ g(p_2) \\ g(p_3) \\ g(p_4) \\ g(p_5) \\ g(p_6) \\ g(p_7) \end{bmatrix} = H \begin{bmatrix} g(-1,-1) \\ g((-1,0)) \\ g(-1,1) \\ g(0,-1) \\ g(0,0) \\ g((0,1) \\ g((1,-1) \\ g(1,0) \\ g(1,1) \end{bmatrix}$$

Where H is:

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & (1-\frac{\sqrt{2}}{2})(1-\frac{\sqrt{2}}{2}) & \frac{\sqrt{2}}{2}(1-\frac{\sqrt{2}}{2}) & 0 & (1-\frac{\sqrt{2}}{2})\frac{\sqrt{2}}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2}(1-\frac{\sqrt{2}}{2}) & \frac{1}{2} & 0 & (1-\frac{\sqrt{2}}{2})(1-\frac{\sqrt{2}}{2}) & \frac{\sqrt{2}}{2}(1-\frac{\sqrt{2}}{2}) & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{\sqrt{2}}{2}(1-\frac{\sqrt{2}}{2}) & 0 & (1-\frac{\sqrt{2}}{2})\frac{\sqrt{2}}{2} & (1-\frac{\sqrt{2}}{2})(1-\frac{\sqrt{2}}{2}) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\sqrt{2}}{2}(1-\frac{\sqrt{2}}{2}) & (1-\frac{\sqrt{2}}{2})(1-\frac{\sqrt{2}}{2}) & 0 & \frac{1}{2} & (1-\frac{\sqrt{2}}{2})\frac{\sqrt{2}}{2} & 0 \end{bmatrix}$$

After we got $g(p_0) \sim g(p_7)$ centered at each pixel, we binarize the grayscales by comparing them with the local threshold $g(0,0)$, which is the grayscale of the central pixel. If the grayscale is greater then the threshold, we denote it as 1, otherwise it's 0. By this thresholding, we can convert the grayscales to a binary pattern.

Rotation-Invariant Representations from Local Binary Patterns

We want the LBP to be rotation-Invariant, but the in place rotation will make the binary pattern from the last step shift by the certain bits. A uniform representation for the binary pattern is desired. We will refer to this representation of a binary pattern as its **minIntVal** representation. Every P-digit binary pattern has its unique decimal representation. We lift shift the original binary pattern by 1 bit for P times and construct an associated decimal list. The **minIntVal** representation is got by taking the binary pattern with the minimal decimal value in the list.

Encoding the minIntVal Forms of the Local Binary Patterns

I encoded the **minIntVal** from last step following the creators of LBP's suggestions:

* If the **minIntVal** representation of a binary pattern has exactly two runs, that is, a run of 0s followed by a run of 1s, represent the pattern by the number of 1s in the second run. Such encodings would be integers between 1 and $P-1$, whose corresponding decimal representations are $2^0, 2^1, ..., 2^{P-1}$, both ends inclusive.

* Else, if the **minIntVal** representation consists of all 0's, represent it be the encoding 0.

* Else, if the **minIntVal** representation consists of all 1's, represent it by the encoding P.

* Else, if the **minIntVal** representation involves more than two runs, encode it by the integer $P + 1$.

Finally, we use a histogram with bins from 0 to $P + 1$ to summary over the whole image. The following table shows the final histograms for the first images in each classes in the training set. I **resized** every images to 250×250 before extrating its LBP.

image	histogram
beach/1.jpg	0.1137, 0.0967, 0.0699, 0.1332, 0.1334, 0.1055, 0.0438, 0.0758, 0.0516, 0.1764
building/01.jpg	0.0798, 0.1075, 0.0597, 0.1886, 0.2005, 0.0908, 0.0352, 0.0870, 0.0333, 0.1176
beach/01.jpg	0.0608, 0.2939, 0.0510, 0.1325, 0.1674, 0.0781, 0.0281, 0.0499, 0.0230, 0.1154
beach/01.jpg	0.1149, 0.1534, 0.0632, 0.1151, 0.1173, 0.0853, 0.0424, 0.0594, 0.0361, 0.2129
beach/01.jpg	0.0666, 0.0813, 0.0867, 0.1340, 0.1786, 0.1132, 0.0641, 0.0864, 0.0649, 0.1243

Table 1: Histograms of the LBP

NN-Classifier

Using Euclidean distance metric find the k-nearest neighbors of the feature vector (histogram) of each testing image in the feature space of training images. Here I set $k = 5$. After the 5 nearest neighbors in the training set being extracted, we got 5 labels, then we can the label which appears the maximum number of times in the set of 5 labels as the prediction class. If two labels appear the same times, we compute the sum of the distances for the corresponding neighbors, the label

with the least sum of distances will be finally assigned to the test image.
Then the confusion matrix is computed, and the accuracy can be calculated by:

$$accuracy = \frac{\text{sum of the diagonal component of the confusion matrix}}{\text{sum of the whole confusion matrix}}$$

The confusion matrix is shown below. I finally got 64% accuracy.

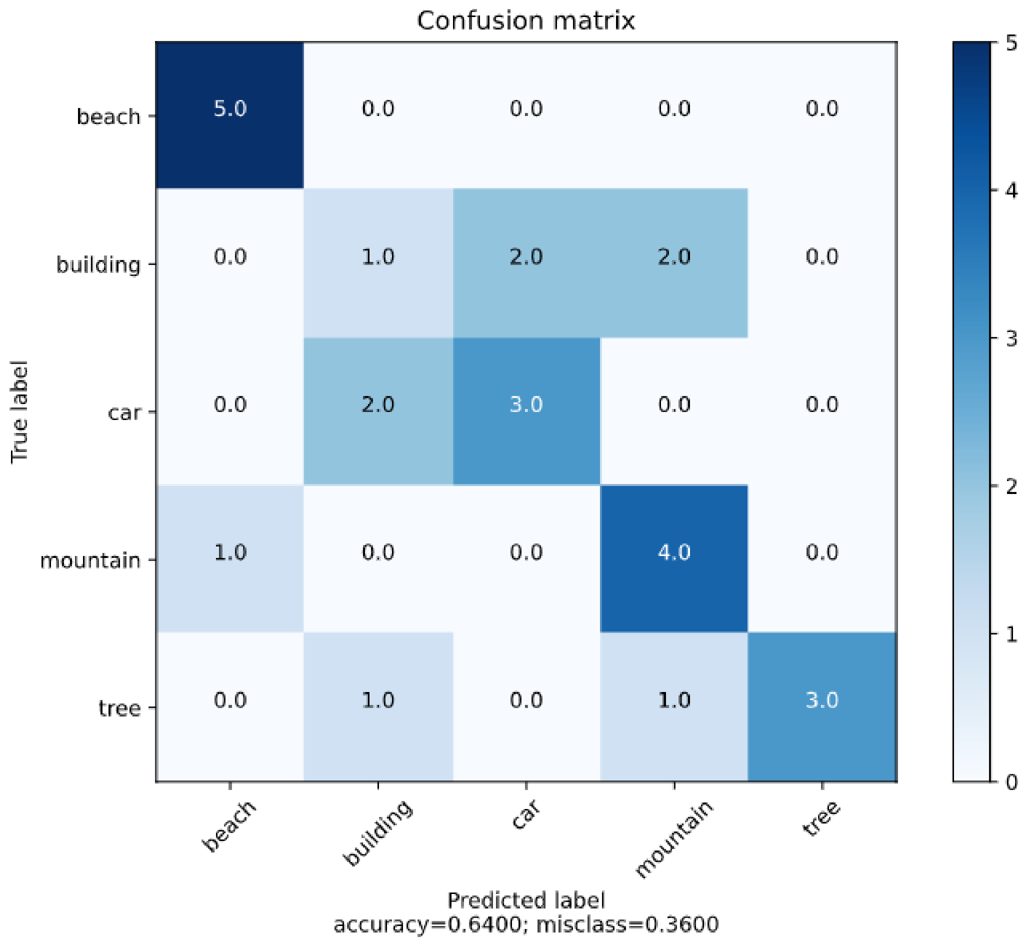


Figure 2: confusion matrix when $K = 5$

Observation

1. The building class is misclassified the most. It may because the variance inside this class is high.
2.The LBP representation is sensitive to the scale of the image, as we resize the input image to different scale, the result can be severely affected.
3.The LBP is a pretty effective representation. Applying PCA to the feature doesn't affect the result in my experiment.

Code

```python
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt

def neighborVector(img):
    tl = img[:-2, :-2].copy()
    t = img[:-2, 1:-1].copy()
    tr = img[:-2, 2:].copy()
    l = img[1:-1, :-2].copy()
    c = img[1:-1, 1:-1].copy()
    r = img[1:-1, 2:].copy()
    bl = img[2:, :-2].copy()
    b = img[2:, 1:-1].copy()
    br = img[2:, 2:].copy()
    neighbormap = np.stack([tl, t, tr, l, c, r, bl, b, br], axis=-1)
    nm = neighbormap
    return nm.reshape((-1, nm.shape[-1]))


def lbp_matrix():
    du, dv = np.sqrt(2)/2, np.sqrt(2)/2
    lbp_matrix = np.asarray([[0, 0, 0, 0, 0, 0, 0, 1, 0],
                             [0, 0, 0, 0, (1-du)*(1-dv), du*(1-dv), 0, dv*(1-du),
   dv*du],
                             [0, 0, 0, 0, 0, 1, 0, 0, 0],
                             [0, du*(1-dv), dv*du, 0, (1-du)*(1-dv), dv*(1-du), 0,
   0, 0],
                             [0, 1, 0, 0, 0, 0, 0, 0, 0],
                             [dv*du, du*(1-dv), 0, dv*(1-du), (1-du)*(1-dv), 0, 0,
   0, 0],
                             [0, 0, 0, 1, 0, 0, 0, 0, 0],
                             [0, 0, 0, du*(1-dv), (1-du)*(1-dv), 0, dv*du, dv*(1-du
   ), 0]])
    return lbp_matrix.T


def makeminIntVal(intval):
    P = intval.shape[-1]
    idices = [_ for _ in range(P)] * 2
    IntVal = np.zeros((intval.shape[0], P))
    weight = 2 ** np.arange(P-1, -1, -1)
    for i in range(P):
        IntVal[:, i] = np.dot(intval[:, idices[i:i+P]], weight)
    minIntVal = np.min(IntVal, axis=1)
    return minIntVal


def encode(bv):
    N = bv.shape[0]
    P = 8
    hist = np.zeros(P+2)
    for i in range(P+1):
        hist[i] = np.sum(bv == (2**i-1)) / N
    hist[-1] = 1 - np.sum(hist[:-1])
    return hist


def LBP(img):
    neighborVec = neighborVector(img)
    threshold_vec = img[1:-1, 1:-1].reshape((-1,1))
    H = lbp_matrix()
    lbp_map = np.dot(neighborVec, H)
    lbp = lbp_map > threshold_vec
    pattern = lbp.astype(np.int)
    minbv = makeminIntVal(pattern)
    hist = encode(minbv)
    return hist

def KNN(train, test, train_label, test_label, K=5, conf_score=100):
    ntest = np.sum(test_set**2, axis=1).reshape((-1, 1))
    ntrain = np.sum(train_set**2, axis=1).reshape((1, -1))
    tt = np.dot(test_set, train_set.T)
    edistance = np.sqrt(ntest - 2 * tt + ntrain)
    idx = edistance.argsort(axis=1)
    neighbors = train_label[idx[:, 0:5]]
    scores = 1 - np.sort(edistance)
    prediction = []
    confusion_m = np.zeros((5, 5))
    for i in range(ntest.shape[0]):
        p = np.zeros(5)
        for j in range(K):
            p[neighbors[i, j]] += (conf_score + scores[i, j])
        pred = np.argmax(p)
        prediction.append(pred)
        confusion_m[test_label[i], pred] += 1
    acc = np.sum(np.array(prediction)==test_label) / ntest.shape[0]
    return confusion_m, acc

def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    """
    given a sklearn confusion matrix (cm), make a nice plot

    Arguments
    ---------
    cm:           confusion matrix from sklearn.metrics.confusion_matrix

    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']

    title:        the text to display at the top of the matrix

    cmap:         the gradient of the values displayed from matplotlib.pyplot.cm
                  see http://matplotlib.org/examples/color/colormaps_reference.
   html
                  plt.get_cmap('jet') or plt.cm.Blues

    normalize:    If False, plot the raw numbers
                  If True, plot the proportions

    Usage
    -----
    plot_confusion_matrix(cm           = cm,                  # confusion matrix
   created by
```

```
                                                          # sklearn.metrics.
   confusion_matrix
                          normalize    = True,                # show proportions
                          target_names = y_labels_vals,       # list of names of
   the classes
                          title        = best_estimator_name) # title of graph

    Citiation
    ---------
    http://scikit-learn.org/stable/auto_examples/model_selection/
   plot_confusion_matrix.html

    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(
   accuracy, misclass))
    plt.show()

# -------------------------------load images ----------------------------
folder_train = '/home/xingguang/Documents/ECE661/hw7/images/training'
folder_test = '/home/xingguang/Documents/ECE661/hw7/images/testing'
classes = ['beach', 'building', 'car', 'mountain', 'tree']
train_data = {'beach':[], 'building':[], 'car':[], 'mountain':[], 'tree':[]}
```

```
test_data = {'beach':[], 'building':[], 'car':[], 'mountain':[], 'tree':[]}
for c in list(train_data.keys()):
    for i in range(1, 21):
        img_path = os.path.join(folder_train, c, '{0:02d}.jpg'.format(i))
        train_data[c].append(cv2.imread(img_path, cv2.IMREAD_GRAYSCALE))
    for i in range(1, 6):
        img_path = os.path.join(folder_test, c + '_' + str(i) +'.jpg')
        test_data[c].append(cv2.imread(img_path, cv2.IMREAD_GRAYSCALE))

# -------------------------------LBP steps -------------------------------
train_set = np.zeros((100, 10))
train_label = np.zeros(100, dtype=np.int)
test_set = np.zeros((25, 10))
test_label = np.zeros(25, dtype=np.int)
train_idx = 0
test_idx = 0
for i, c in enumerate(classes):
    for j, img in enumerate(train_data[c]):
        train_set[train_idx, :] = LBP(cv2.resize(img, (250,250)))
        if j == 0: print("LBP histogram of the first image in "+c+':', train_set[
    train_idx, :])
        train_label[train_idx] = i
        train_idx += 1
    for img in test_data[c]:
        test_set[test_idx, :] = LBP(cv2.resize(img, (250,250)))
        test_label[test_idx] = i
        test_idx += 1

# -------------------------------Measuring results --------------------------
confusion_matrix, accuracy = KNN(train_set, test_set, train_label, test_label, K
    =5)
plot_confusion_matrix(confusion_matrix,
                          target_names=classes,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=False)
```