The Hong Kong Polytechnic University

Department of Electronic and Information Engineering

Feature Extraction and Image Classification

EIE4100

Laboratory 1

Seksembayev Kairat

18078689d

I used python to complete the lab, but it is quite similar to the MATLAB code provided. All code used in the lab can be also found in the attachment.

Image Processing

def mat2gray(A):

    A = np.double(A)

    out = np.zeros(A.shape, np.double)

    normalized = cv2.normalize(A, out, 1.0, 0.0, cv2.NORM\_MINMAX)

    return normalized

def rgb2gray(rgb):

    return np.dot(rgb[..., :3], [0.299, 0.587, 0.144])

A = plt.imread('test\_color.jpg')

A\_gray = rgb2gray(A)

nc, nr = A\_gray.shape

Sx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) # Sobel operator

Sy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]]) # Sobel operator

Ax = signal.correlate2d(A\_gray, Sx, mode='same')

Ay = signal.correlate2d(A\_gray, Sy, mode='same')

plt.figure(num="Image")

a = plt.subplot(2,2,1); plt.imshow(A); plt.axis('off'); a.title.set\_text("Original")

b = plt.subplot(2,2,2); plt.imshow(A\_gray, cmap='gray'); plt.axis('off'); b.title.set\_text("Gray")

c = plt.subplot(2,2,3); plt.imshow(np.uint8(mat2gray(Ax)\*255), cmap='gray');plt.axis('off')

c.title.set\_text("Result of horizontal gradient")

d = plt.subplot(2,2,4); plt.imshow(np.uint8(mat2gray(Ay)\*255), cmap='gray')

d.title.set\_text("Result of vertical gradient")

plt.axis('off')

plt.show()

print(str(A.shape )+ " size of original image\n"+str(A\_gray.shape)+" size of the gray image")

Text

Description automatically generated

Text

Description automatically generated

Size of A = 256(h) x 384(w)x3

Size of B = 256(h) x 384(w)x1

For Python code:

rgb2gray(rgb): Convert a RGB colour image to a gray-scale image.

correlate2d() applies filter to filter an image. Used to compute the gradient of a given image

For MATLAB code:

rgb2gray: Convert a RGB colour image to a gray-scale image.

imfilter: Use a filter to filter an image.

mat2gray: Convert matrix to gray-scale image.

Extracting the HoG Feature Vector

def GradientMandO(self,I):

        Sx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

        Sy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

        Ix = signal.correlate2d(I, Sx, mode='same')

        Iy = signal.correlate2d(I, Sy, mode='same')

        # I\_mag: gradient magnitude

        I\_mag=np.sqrt(Ix\*\*2 + Iy\*\*2)

        # Gradient orientation

        nr, nc = I.shape

        Ipr = np.zeros(shape=(nr, nc))

        I\_angle = np.zeros(shape=(nr, nc))

        for j in range(nr):

            for i in range(nc):

                if abs(Ix[j, i]) <= 0.0001 and abs(Iy[j, i]) <= 0.0001: #When both Ix and Iy are close to zero

                    I\_angle[j, i] = 0.00

                else:

                    Ipr[j, i] = math.atan(Iy[j,i]/(Ix[j,i]+np.finfo(float).eps)) #Compute the angle in radians

                    I\_angle[j, i] = Ipr[j, i]\*180/math.pi # Compute the angle in degrees

                    if Ix[j, i] < 0: # If Ix is negative, 180 degrees added

                        I\_angle[j, i] = 180+I\_angle[j, i]

                    if I\_angle[j, i] < 0: # If the angle is negative,360 degrees added

                        I\_angle[j, i] = 360+I\_angle[j, i]

        return (I\_mag, I\_angle)

def getFeatureVec(self,I, I\_mag, I\_angle, nr\_b, nc\_b):

        nr, nc = I.shape

        nbin = 9

        nr\_size = int(nr/nr\_b)

        nc\_size = int(nc/nc\_b)

        Image\_HoG = np.zeros(shape=(1, nbin\*nr\_b\*nc\_b))

        for i in range(nr\_b):

            for j in range(nc\_b):

                I\_mag\_block = I\_mag[i\*nr\_size: (i+1)\*nr\_size, j\*nc\_size: (j+1)\*nc\_size]

                I\_angle\_block = I\_angle[i\*nr\_size: (i+1)\*nr\_size, j\*nc\_size: (j+1)\*nc\_size]

                # HoG1 creates HoG histogram

                gh = self.HoG1(I\_mag\_block, I\_angle\_block, nbin)

                # Histogram\_Normalization normalizes the input histogram gh

                ngh = self.Histogram\_Normalization(gh)

                pos = j\*nbin+i\*nc\_b\*nbin

                Image\_HoG[:, pos:pos+nbin] = ngh

        return Image\_HoG

def HoG1(self,Im, Ip, nbin):

        ghist = np.zeros(shape=(1,nbin))

        [nr1, nc1] = Im.shape

        interval = np.round(180/nbin, 0)

        for i in range(nr1):

            for j in range(nc1):

                if Ip[i, j] > 180:

                    Ip[i, j] = abs(Ip[i, j] - 360)

                index = int(np.int(Ip[i, j]/interval))

                if index >= nbin:

                    index = index - 1

                ghist[0, index] += np.square(Im[i,j])

        return ghist

interval is 180/9, where 9 is the number of bins used

def Histogram\_Normalization(self,ihist):

        total\_sum = np.sum(ihist)

        nhist = ihist / total\_sum

        return nhist

nhist is the normalized histogram and hence its is the original histogram/sum(original histogram)

Matching of HoG histogram. Image Classification and Evaluation

The procedure is done in ClassifyTrainAndTestImages method.

def ClassifyTrainAndTestImages(self):

        if(os.path.exists("h1.npy") and os.path.exists("h2.npy")):

            h1 = np.load("h1.npy")

            h2 = np.load("h2.npy")

            print("h1 and h2 are retrieved")

        else:

            h1 = np.zeros(shape=(25, self.nbin\*self.nr\_b\*self.nc\_b)) # training data

            h2 = np.zeros(shape=(25, self.nbin\*self.nr\_b\*self.nc\_b)) # test data

            for i in range(1,6):

                for j in range(1,6):

                    I = self.rgb2gray(self.extractImg(i,j,"training"))

                    h1[(j-1)+(i-1)\*5] = self.getFeatureVec(I,self.GradientMandO(I)[0],self.GradientMandO(I)[1],self.nr\_b, self.nc\_b)

                    I = self.rgb2gray(self.extractImg(i,j,"test"))

                    h2[(j-1)+(i-1)\*5] = self.getFeatureVec(I,self.GradientMandO(I)[0],self.GradientMandO(I)[1],self.nr\_b, self.nc\_b)

            np.save("h1.npy",h1)

            np.save("h2.npy",h2)

            print("h1 and h2 are saved")

        if(os.path.exists("d1.npy") and os.path.exists("d2.npy") and os.path.exists("chi.npy")):

            d1 = np.load("d1.npy")

            d2 = np.load("d2.npy")

            chi = np.load("chi.npy")

            print("d1, d2 and chi are retrieved")

        else:

            d1 = np.zeros (shape=(25,25))

            d2 = d1.copy()

            chi = d2.copy()

            for i in range(25):

                for j in range(25):

                    d1[i, j] = np.around(np.sum(np.abs(h2[i, :]-h1[j, :])),4)

                    d2[i, j] = np.around(np.sum(np.square(np.abs(h2[i, :]-h1[j, :]))), 4)

                    chi[i, j] = np.around(np.sum(np.square(np.abs(h2[i, :]-h1[j, :])) / (h2[i,:]+h1[j, :]+np.finfo(float).eps)), 4)

            np.save("d1.npy",d1)

            np.save("d2.npy",d2)

            np.save("chi.npy",chi)

        d1\_min = np.argmin(d1,axis=1)

        d2\_min = np.argmin(d2,axis=1)

        chi\_min = np.argmin(chi,axis=1)

        acc = np.zeros(shape=(25))

        for i in range(5):

            for j in range(5):

                acc[j+5\*i] = 5\*i

        d1\_res = d1\_min - acc

        d2\_res = d2\_min - acc

        chi\_res = chi\_min - acc

        d1\_acc=d2\_acc=chi\_acc = 0

        for i in d1\_res:

            if(i>=0 and i<=4):

                d1\_acc+=1

        d1\_acc = d1\_acc/d1\_res.size

        for i in d2\_res:

            if(i>=0 and i<=4):

                d2\_acc+=1

        d2\_acc = d2\_acc/d2\_res.size

        for i in chi\_res:

            if(i>=0 and i<=4):

                chi\_acc+=1

        chi\_acc = chi\_acc/chi\_res.size

        print(d1\_acc,d2\_acc,chi\_acc)

def extractImg(self,class\_no=1, image\_no=1, type="training"):

        type = "training" if (type == "training") else "test"

        path = str(class\_no) + "/" + str(class\_no) + str(image\_no) + '\_'+type.capitalize()+".bmp"

        return plt.imread(path)

def rgb2gray(self,rgb):

        if rgb.ndim != 3:

            return rgb

        return np.dot(rgb[..., :3], [0.299, 0.587, 0.144])

def \_\_init\_\_(self, nr\_b = 2, nc\_b = 2):

        self.nc\_b = nc\_b

        self.nr\_b = nr\_b

Explanation of Image classification and accuracy evaluation procedure:

Since the computation of HOG features take a lot of time, the final results and temporary results are stored in npy files.In this sense, the process the program executes faster and also makes the debug process easier. Before the code is executed, it checks whether the features vectors and results of matching are already computed. If so, the results are loaded from corresponding npy files, otherwise, the “heavy” computation is done. In order to re-compute for different input parameters, all npy files have to be removed.

Firstly, images for training and testing are extracted from according folders. In order for the code for image extraction to work properly, python script has to be placed inside “Images” folder, like it is done in the attached folder. Then, HOG feature vectors are computed for training and testing images. The HOG feature vectors for training images are stored in h1 numpy array and feature vectors for testing images are stored in h2 numpy array. Then, for each testing image the 3 different types of distances between it and all 25 training images are computed.

In order to evaluate the accuracy, the index of a minimum distance of a testing image for each of training images is found (so the matrix is reduced from 25x25 to 1x25). This is repeated for all types of distances used. Lastly, in order to identify the correct match we make use of a way we stored our feature vectors and related distances. All HOG feature vectors are stored in h1 and h2 in order they were extracted, so this order is also same for numpy arrays for 3 types of distances calculated. We can say the match is correct if the index of a minimum distance value among 25 training images belongs to the same class as the test image.

Example, if test image belongs to class 1 and the distance for this test image is computed for all 25 training images. Among those 25 training images` feature vectors first 5 belong to class 1 and hence if an index of a min distance is between 0 and 4 this is a correct guess. The same rule applies to all other classes, however the comparison should take into an account the fact that indexes of the next class, say class 2, would be from 5 to 9 and so on.

In this sense, we calculate the number of times our matching is correct and divide it by the total number of training images used.

Results:

(i) Set nr\_b = 2 and nc\_b = 2, evaluate the accuracy of classifying the 25 testing images using the distance metrics: L1-metric, L1-metric, and Chi-square distance.

Text

Description automatically generated

(ii) Repeat (i) by setting nr\_b = 3 and nc\_b = 3.

Text

Description automatically generated

Discussion

As we can see the overall accuracy decreases when the size of the HOG block is increased.

When 2x2 the average accuracy is ~78%

When 3x3 the average accuracy is ~71%

Also, we can see how L2 metrics always shows lowest accuracy among all methods to measure the distance between feature vectors. L1 metric shows more stable results whereas Chi-square distance was the most accurate when HOG block is 2x2 and the second most accurate when HOG block is 3x3. In this sense, it is better to use 2x2 HOG block.

Summary

In this lab I was able to implement HOG feature extraction and Image classification based on 3 different distance metrics from scratch. It was also a great practice for Image processing techniques.

Pythod codes attached are:

Text

Description automatically generated