Project 2 Document

Weicheng Dai

# 1. The dependencies

The important dependencies are listed below:

* 1. [Jupyter notebook](https://jupyter.org/) version 6.4.4
  2. [Python](https://www.python.org/) version 3.9.7.
  3. [NumPy](https://numpy.org/) version 1.21.2
  4. [matplotlib](https://matplotlib.org/) version 3.4.3
  5. [PIL](https://pillow.readthedocs.io/en/stable/) version 8.3.2
  6. [sklearn](https://scikit-learn.org/stable/index.html) version 1.0.1

# 2. Instructions on using the script

There are 9 cells in the codes, which are marked by indexes, as can be seen in Figure 1. When user wants to use this script, simply click ‘run all’ and it will automatically execute each function in order.

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

Figure 1: Example of indexes

# 2.1 Cells from # 1 to # 8

The cells from # 1 to # 8 each defines a function which will be helpful in this process. Their functions are listed below:

cell#1: import packages

cell#2: the function that converts RGB images to gray level ones

cell#3: the function that computes gradients of both horizontal and vertical direction

cell#4: the function that computes gradient magnitude and gradient angle

cell#5: the function that computes the orientation gradient for each pixel

cell#6: the function that cumulates orientation gradient of pixels to form blocks

cell#7: the function that computes the distance between two feature maps, which is intersection.

cell#8: the function that processes the training data

# 2.2 Cell # 9

The cell that processes the testing data and print the results.

# 3. Results

The gradient magnitude images for the 10 test images:

图片包含 照片, 田地, 站, 男人

描述已自动生成图片包含 乐器, 照片, 女人, 站

描述已自动生成图片包含 照片, 看着, 站, 黑暗

描述已自动生成图片包含 游戏机, 站, 播放, 女人

描述已自动生成夜晚的街景

中度可信度描述已自动生成

图片包含 草, 建筑, 照片, 田地

描述已自动生成图片包含 水, 男人, 黑暗, 大

描述已自动生成建筑与房屋的城市空拍图黑白照

中度可信度描述已自动生成建筑旁的轨道上有一列火车的黑白照片

中度可信度描述已自动生成夜晚的自行车

低可信度描述已自动生成

The table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test image | Correct Classification | File name of 1st NN, distance & classification | File name of 2nd NN, distance & classification | File name of 3rd NN, distance & classification | Classification from 3-NN |
| crop001034b | Human | crop001275b  0.6533  Human | crop001028a  0.592  Human | 00000093a\_cut  0.5537  No-human | Human |
| crop001070a | Human | crop001275b  0.5405  Human | 00000093a\_cut  0.4954  No-human | crop001028a  0.481  Human | Human |
| crop001278a | Human | crop001275b  0.4878  Human | crop001028a  0.4124  Human | no\_person\_\_no\_bike\_247\_cut  0.4065  No-human | Human |
| crop001500b | Human | no\_person\_\_no\_bike\_247\_cut  0.342  No-human | crop001275b  0.3027  Human | crop001028a  0.2256  Human | Human |
| person\_and\_bike\_151a | Human | crop001275b  0.527  Human | no\_person\_\_no\_bike\_247\_cut  0.515  No-human | crop001028a  0.4836  Human | Human |
| 00000003a\_cut | No-human | no\_person\_\_no\_bike\_247\_cut  0.5156  No-human | crop001275b  0.51  Human | 00000093a\_cut  0.451  No-human | No-human |
| 00000090a\_cut | No-human | crop001275b  0.342  Human | 00000053a\_cut  0.3257  No-human | no\_person\_\_no\_bike\_247\_cut  0.2832  No-human | No-human |
| 00000118a\_cut | No-human | no\_person\_\_no\_bike\_219\_cut  0.4783  No-human | crop001275b  0.4722  Human | 00000093a\_cut  0.455  No-human | No-human |
| no\_person\_no\_bike\_258\_ cut | No-human | crop001275b  0.4377  Human | no\_person\_\_no\_bike\_247\_cut  0.3804  No-human | crop001028a  0.365  Human | Human |
| no\_person\_no\_bike\_264\_ cut | No-human | no\_person\_\_no\_bike\_247\_cut  0.4043  No-human | crop001275b  0.3838  Human | 00000093a\_cut  0.3677  No-human | No-human |

# 4. Codes

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image

# import cv2

import os

from sklearn.preprocessing import normalize

def toGray(img):

""" convert RGB img to gray level img

Parameters

----------

img: {ndarray} shape = [row \* col]

the input image which is RGB

Returns

-------

ans: {ndarray} shape = [row \* col]

the output image which is gray level

"""

ans = np.zeros([img.shape[0],img.shape[1]], dtype = np.float16)

# The sequence is R, G, B

for i in range(img.shape[0]):

for j in range(img.shape[1]):

ans[i,j] += round(0.299 \* img[i, j, 0] + 0.587 \* img[i, j, 1] + 0.114 \* img[i, j, 2])

return ans

def gradient\_operation(img):

""" compute the horizontal and vertical gradient

Parameters

----------

img: {ndarray} shape = [row \* col]

the input image that we want to compute the gradient

Returns

-------

grad\_hori: {ndarray} shape = [row \* col]

the horizontal gradient, same shape as the input

grad\_vert: {ndarray} shape = [row \* col]

the vertical gradient, same shape as the input

"""

# The Prewitt operator with vertical and horizontal orientation

Prewitt\_X = np.array([[-1, 0, 1],

[-1, 0, 1],

[-1, 0, 1]], dtype=np.float16)

Prewitt\_Y = np.array([[1, 1, 1],

[0, 0, 0],

[-1, -1, -1]], dtype=np.float16)

# The answers initialized with all 0s, same shape as the input image

grad\_hori = np.zeros([img.shape[0],img.shape[1]], dtype = np.float16)

grad\_vert = np.zeros([img.shape[0],img.shape[1]], dtype = np.float16)

# The procedure of doing the convolution

# Since the two operators are of the same shape, we can do it with one iteration

for i in range(img.shape[0]):

for j in range(img.shape[1]):

for m in range(Prewitt\_X.shape[0]):

for n in range(Prewitt\_X.shape[1]):

if(i - Prewitt\_X.shape[0] // 2 < 0 or i + Prewitt\_X.shape[0] // 2 >= img.shape[0] or

j - Prewitt\_X.shape[1] // 2 < 0 or j + Prewitt\_X.shape[1] // 2 >= img.shape[1]):

continue

else:

grad\_hori[i, j] += Prewitt\_X[m, n] \* img[i - 1 + m, j - 1 + n]

grad\_vert[i, j] += Prewitt\_Y[m, n] \* img[i - 1 + m, j - 1 + n]

return grad\_hori, grad\_vert

def generate\_magnitude\_direction(grad\_hori, grad\_vert):

""" generates gradient magnitude and gradient angle

Parameters

----------

grad\_hori: {ndarray} shape = [row \* col]

the horizontal gradient of an image

grad\_vert: {ndarray} shape = [row \* col]

the vertical gradient of an image

Returns

-------

gradient: {ndarray} shape = [row \* col]

the gradient magnitude of an image at each pixel

direction: {ndarray} shape = [row \* col]

the gradient angle of an image at each pixel

"""

# np.hypot does (x^2 + y^2)^(0.5) at each pixel

gradient = np.hypot(grad\_hori, grad\_vert)

# np.arctan2 generates the answer within the range [-pi, pi], and we convert it into [0, 180]

direction = (np.arctan2(grad\_vert, grad\_hori) \* 180 / np.pi) % 180

return gradient, direction

def OG(gradient, direction):

""" finds the corresponding orientation gradient for each pixel

Parameters

----------

gradient: {ndarray} shape = [row \* col]

the gradient magnitude of an image at each pixel

direction: {ndarray} shape = [row \* col]

the gradient angle of an image at each pixel

Returns

-------

orientation\_gradient: {ndarray} shape = [row \* col \* 9]

the orientation gradient of an image

"""

orientation\_gradient = np.zeros([gradient.shape[0], gradient.shape[1], 9], dtype = np.float16)

for i in range(gradient.shape[0]):

for j in range(gradient.shape[1]):

cur\_class = int(direction[i, j] // 20) # where the current class is, should be 0~8

if(cur\_class == 9):

cur\_class-=1

pivot = direction[i, j] % 20 # use pivot to find another class

if(pivot<10):

# use mod to prevent edge situation

# cur\_weight is computed by finding the distance with current pivot

# but the true current weight is actually another\_weight, because we have to take the inverse value

# another\_weight + cur\_weight == 20

another\_class = (cur\_class - 1) % 9

cur\_weight = 10 - pivot

another\_weight = 10 + pivot

else:

another\_class = (cur\_class + 1) % 9

cur\_weight = pivot - 10

another\_weight = 30 - pivot

orientation\_gradient[i, j, cur\_class] += gradient[i, j] /20 \* another\_weight

orientation\_gradient[i, j, another\_class] += gradient[i, j] /20 \* cur\_weight

return orientation\_gradient

def feature(orientation\_gradient):

""" finds the cumulated orientation gradient for block

Parameters

----------

orientation\_gradient: {ndarray} shape = [row \* col \* 9]

the orientation gradient of an image

Returns

-------

feature\_map : {ndarray} shape = [36 \* n], n is the number of overlapping blocks

the feature map of a given img

"""

cell\_size = 8

block\_size = 16

# print(orientation\_gradient.shape[0]) # 160

# print(orientation\_gradient.shape[1]) # 96

# first we compute the feature map per cell

# num\_rows and num\_cols is the size of feature per cell

num\_rows = int(orientation\_gradient.shape[0] / cell\_size) # 20

num\_cols = int(orientation\_gradient.shape[1] / cell\_size) # 12

# this is the num of cols in the whole feature map

num\_blks = (num\_rows - 1) \* (num\_cols - 1)

feature\_cell = np.zeros([num\_rows, num\_cols, 9], dtype = np.float16)

# print(feature\_cell.shape[0]) # 20

# print(feature\_cell.shape[1]) # 12

# accumulate the orientation gradient of each cell

for i in range(0, orientation\_gradient.shape[0]-cell\_size + 1, cell\_size):

for j in range(0, orientation\_gradient.shape[1]-cell\_size + 1, cell\_size):

for k in range(cell\_size):

for m in range(cell\_size):

for d in range(9):

feature\_cell[int(i/cell\_size), int(j/cell\_size), d] += orientation\_gradient[(i+k), (j+m), d]

# use the orientation gradient of each cell to form the blks'

feature\_map = np.zeros([36, num\_blks], dtype = np.float16)

for i in range(0, num\_rows-1, 1):

for j in range(0, num\_cols-1, 1):

for k in range(9):

feature\_map[k, i\*(num\_cols-1)+j] = feature\_cell[i, j, k]

feature\_map[k+9, i\*(num\_cols-1)+j] = feature\_cell[i+1, j, k]

feature\_map[k+18, i\*(num\_cols-1)+j] = feature\_cell[i, j+1, k]

feature\_map[k+27, i\*(num\_cols-1)+j] = feature\_cell[i+1, j+1, k]

# use l2 norm

feature\_map = normalize(feature\_map, axis=0, norm='l2')

return feature\_map

def distance(map1, map2):

"""computes the distance between two feature maps

Parameters

----------

map1: {ndarray}, shape = [36 \* n], n is the number of overlapping blocks

the test img

map2: {ndarray}, shape = [36 \* n], n is the number of overlapping blocks

the train img

returns

-------

ans: float, the IOU of two maps

"""

numerator = np.sum(np.minimum(map1, map2))

denominator = map2.sum()

return numerator/denominator

def process\_data():

"""Deals with the training images, save files and export features

Returns

-------

training\_Pos: list, shape = [m1 \* 36 \* n], n is the number of overlapping blocks, m1 is the number of positive training img

training\_Neg: list, shape = [m2 \* 36 \* n], n is the number of overlapping blocks, m2 is the number of negative training img

"""

training\_Pos = []

# for each file in Positive training file, execute the functions above in order.

for filename in os.listdir("./Training images (Pos)"):

img = plt.imread("./Training images (Pos)" + "/" + filename)

img = toGray(img)

grad\_hori, grad\_vert = gradient\_operation(img)

gradient, direction = generate\_magnitude\_direction(grad\_hori, grad\_vert)

orientation\_gradient = OG(gradient, direction)

feature\_map = feature(orientation\_gradient)

training\_Pos.append(feature\_map)

# for those whose HOG should be saved, execute this separately.

if(filename[:-4] == 'crop001028a' or filename[:-4] == 'crop001030c'):

fo = open('pos\_{}\_lines.txt'.format(filename[:-4]), "w")

for i in range(feature\_map.shape[0]):

for j in range(feature\_map.shape[1]):

fo.write(str(feature\_map[i, j])+"\n")

fo.close()

# for each file in Negative training file, execute the functions above in order.

training\_Neg = []

for filename in os.listdir("./Training images (Neg)"):

img = plt.imread("./Training images (Neg)" + "/" + filename)

img = toGray(img)

grad\_hori, grad\_vert = gradient\_operation(img)

gradient, direction = generate\_magnitude\_direction(grad\_hori, grad\_vert)

orientation\_gradient = OG(gradient, direction)

feature\_map = feature(orientation\_gradient)

training\_Neg.append(feature\_map)

# for those whose HOG should be saved, execute this separately.

if(filename[:-4] == '00000091a\_cut'):

fo = open('neg\_{}\_lines.txt'.format(filename[:-4]), "w")

for i in range(feature\_map.shape[0]):

for j in range(feature\_map.shape[1]):

fo.write(str(feature\_map[i, j])+"\n")

fo.close()

return training\_Pos, training\_Neg

# first retrieve the training dataset with the function above

training\_Pos, training\_Neg = process\_data()

training\_Pos = np.array(training\_Pos) # shape = [m1 \* 36 \* n]

training\_Neg = np.array(training\_Neg) # shape = [m2 \* 36 \* n]

# then concatenate them, in order to sort more conveniently.

# remember the index between 0 to 9 is positive, index between 10 to 19 is negative

training = np.concatenate((training\_Pos, training\_Neg), axis=0)

# class value has the shape of [10\*20], 10 means 10 test imgs while 20 means 20 training imgs

class\_value = []

# same order as above

# except for computing the distance (IOU) between test imgs and training imgs

for filename in os.listdir("./Test images (Pos)"):

single\_test = []

img = plt.imread("./Test images (Pos)" + "/" + filename)

img = toGray(img)

grad\_hori, grad\_vert = gradient\_operation(img)

gradient, direction = generate\_magnitude\_direction(grad\_hori, grad\_vert)

plt.imsave("test\_gradient\_{}.png".format(filename[:-4]),

(gradient.astype(np.int16))/np.max(gradient.astype(np.int16)) \*255, cmap = 'gray')

orientation\_gradient = OG(gradient, direction)

feature\_map = feature(orientation\_gradient)

for i in range(training.shape[0]):

single\_test.append(distance(feature\_map, training[i]))

class\_value.append(single\_test)

if(filename[:-4] == 'crop001278a' or filename[:-4] == 'crop001500b'):

fo = open('test\_{}\_lines.txt'.format(filename[:-4]), "w")

for i in range(feature\_map.shape[0]):

for j in range(feature\_map.shape[1]):

fo.write(str(feature\_map[i, j])+"\n")

fo.close()

for filename in os.listdir("./Test images (Neg)"):

single\_test = []

img = plt.imread("./Test images (Neg)" + "/" + filename)

img = toGray(img)

grad\_hori, grad\_vert = gradient\_operation(img)

gradient, direction = generate\_magnitude\_direction(grad\_hori, grad\_vert)

plt.imsave("test\_gradient\_{}.png".format(filename[:-4]),

(gradient.astype(np.int16))/np.max(gradient.astype(np.int16)) \*255, cmap = 'gray')

orientation\_gradient = OG(gradient, direction)

feature\_map = feature(orientation\_gradient)

for i in range(training.shape[0]):

single\_test.append(distance(feature\_map, training[i]))

class\_value.append(single\_test)

if(filename[:-4] == '00000090a\_cut'):

fo = open('test\_{}\_lines.txt'.format(filename[:-4]), "w")

for i in range(feature\_map.shape[0]):

for j in range(feature\_map.shape[1]):

fo.write(str(feature\_map[i, j])+"\n")

fo.close()

# convert to ndarray for sorting

# 3-NN so find the largest 3 results, then print them

# remember the first 5 are positive test imgs, second 5 are negative test imgs

# and the value from 0 to 9 means positive sample, from 10 to 19 means negative sample

for i in range(len(class\_value)):

print(class\_value[i])

class\_value = np.array(class\_value)

class\_result = []

for i in range(class\_value.shape[0]):

idx = np.argsort(class\_value[i])[-3:]

class\_result.append(idx)

print(class\_result)