(a)

File name for the code: Project2.ipynb
File name for the result: output.txt

(b)

You can directly run the project2.ipynb file. In the section of training neural network, you can specify the number of neurons and input.

(c)

Result for layer size of 250 neurons
Accuracy = 40%

```
[25]
result = HOG_nn.pred(test_input.T)
#print probability result
print(result)
```

```
[[7.96258897e-13 5.07085538e-06 1.09504089e-07 1.74108319e-01
  7.73218200e-13 4.33168518e-34 1.47117814e-01 3.44989732e-15
  9.84174832e-01 2.82480584e-09]]
```

```
[26]
result[result >= 0.5] = 1
result[result < 0.5] = 0
result
```

```
matrix([[0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])
```

Result for layer size of 500 neurons
Accuracy = 40%

```
result = HOG_nn_500.pred(test_input.T)
print("probability result:")
print(result)
result[result >= 0.5] = 1
result[result < 0.5] = 0
print(result)
```

```
probability result:
[[9.99999997e-01 3.12256353e-14 1.00000000e+00 1.50151626e-23
  1.00000000e+00 1.10860741e-06 1.00000000e+00 4.46865026e-36
  1.00000000e+00 1.00000000e+00]]
[[1. 0. 1. 0. 1. 0. 1. 0. 1. 1.]]
```

Result for layer size of 1000 neurons
Accuracy = 40%

```
result = HOG_nn_1000.pred(test_input.T)
print("probability result:")
print(result)
result[result >= 0.5] = 1
result[result < 0.5] = 0
print(result)
```

```
probability result:
[[2.47667760e-09 9.99499761e-01 1.00000000e+00 7.38954563e-41
  9.89073402e-01 9.64289540e-01 6.64197595e-06 1.00000000e+00
  9.99997632e-01 1.00000000e+00]]
[[0. 1. 1. 0. 1. 1. 0. 1. 1. 1.]]
```

(d)Source Code

Import library

```
[2]
import cv2
import math
import numpy as np
from matplotlib import pyplot as plt
```

Define function for normalization and image display

```
[3]
#image normalization
def normalization(img, range):
  normed_img = img/(img.max()/range)
  return normed_img
```

```
[4]
def plotImage(image, title):
  plt.imshow(image, 'gray', vmin = 0, vmax = 255)
  plt.title(title)
  fig = plt.gcf()
  fig.set_size_inches(13,13)
  plt.show()
```

```
def convolve2d(image, kernel, stride = 1):
  kernel = np.flipud(np.fliplr(kernel))

  k_sizeX, k_sizeY = kernel.shape

  im_sizeX, im_sizeY = image.shape

  padding = int(np.floor((k_sizeX-1)/2)) # padding = ((k-1) / 2)

  #output image (convolved with image)
  new_image = np.zeros((im_sizeX + 2*padding, im_sizeY + 2*padding))
  new_image[padding: im_sizeX+padding, padding: im_sizeY + padding] = image[:,:]

  output = np.zeros(new_image.shape)

  new_im_sizeX, new_im_sizeY = new_image.shape
  for y in range(new_im_sizeY):
    if y > new_im_sizeY-k_sizeY:
      break

    for x in range(new_im_sizeX):
      if x > new_im_sizeX-k_sizeX:
        break

      if( y % stride == 0 and x%stride == 0):

        output[int(np.floor((2*x+k_sizeX)/2)),int(np.floor((2*y+k_sizeY)/2))] = (kernel * new_image[x:x+k_sizeX, y:y+k_sizeY]).sum()

  return output
```

```python
def grey_scale(img):
    R, G, B = img[:,:,0], img[:,:,1], img[:,:,2]
    imgGray = 0.2989 * R + 0.5870 * G + 0.1140 * B
    return imgGray
```

{}

Define Prewitt Operator

```python
Px = np.array([[1, 0, -1],
               [1, 0, -1],
               [1, 0, -1]])

Py = np.array([[1, 1, 1],
               [0, 0, 0],
               [-1, -1, -1]])
```

```python
def get_hist_cell(theta, M):
    #get the histogram of the each cell
    width, height = M.shape[0], M.shape[1]
    bin_size = 9
    cell_size = 8
    step = 8
    hist_vector = np.zeros((int(width/cell_size), int(height/cell_size), bin_size))
    for i in range(hist_vector.shape[0]):
        for j in range(hist_vector.shape[1]):
            cell_magnitude = M[i * cell_size: (i+1) * cell_size, j*cell_size : (j+1) *cell_size]
            cell_theta = theta[i * cell_size: (i+1) * cell_size, j*cell_size : (j+1) *cell_size]
            cell_hist = get_bin(cell_theta, cell_magnitude)
            hist_vector[i][j] = cell_hist



# get the histogram of the whole image
    hog_vector = []
    for i in range(int(width/cell_size) - 1):
        for j in range(int(height/cell_size) -1):
            block_vec = []
            block_vec.extend(hist_vector[i][j])
            block_vec.extend(hist_vector[i][j+1])
            block_vec.extend(hist_vector[i+1][j])
            block_vec.extend(hist_vector[i+1][j+1])
            hog_vector.extend(block_normalization(block_vec))
    return hog_vector
```

HOG Feature

```python
def get_bin(cell_theta, cell_M):
    #calculate the histogram of the each cell
    bin_size = 9
    bin_degree = 180/bin_size
    hist = np.zeros(9)
    for i in range(cell_theta.shape[0]):
        for j in range(cell_theta.shape[1]):
            bin_index = int((cell_theta[i, j] + 10) // 20)

            v_1 = cell_M[i, j] * (bin_degree * (bin_index + 1) - 10 - cell_theta[i,j])/bin_degree
            v_2 = cell_M[i, j] * (cell_theta[i,j] - bin_degree * bin_index + 10)/bin_degree

            hist[bin_index] += v_1
            if bin_index +1 <= 8:
                hist[bin_index+1] += v_2
    return hist

def block_normalization(block_vec):
    temp = np.sqrt(np.sum(np.power((block_vec),2)))
    if temp == 0:
        return block_vec
    return block_vec/temp
```

Feed into the Neuro Network

```python
class nn():
    def __init__(self, X, layer_size, test_X):
        np.random.seed(1)
        self.X = X.T
        self.test_X = test_X.T
        # self.y = y
        self.ground_truth = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 ,0 ,0 ,0 ,0]])
        self.num = 0
        self.test_ground_truth = np.array([[1, 1, 1, 1, 1, 0, 0, 0, 0, 0]])
        self.y_pred = np.zeros((self.ground_truth.shape[1],1))
        self.sam = self.ground_truth.shape[1]
        self.learning_rate = 0.00003
        # self.layer_num = 2
        self.layer_size = layer_size
        self.weights_1 = 2 * np.random.rand(self.layer_size, X.shape[1]) - 1
        self.weights_2 = 2 * np.random.rand(1, self.layer_size) - 1
        self.bias_1 = np.random.rand(self.layer_size,1)
        self.bias_2 = np.random.rand(1,1)
        self.test_loss = 0

        self.loss = 0
        # self.layer_output = feed_forward(X, weights)

    def dSigmoid(self,x):
        s = 1/(1+np.exp(-x))
        return np.multiply(s,(1-s))

    def dRelu(self,x):
        x[ x <= 0] = 0
        x[x > 0] = 1
        return x
```

```python
    def sigmoid_function(self,x):
        return 1/(1+np.exp(-x))

    def relu_function(self,x):
        return np.maximum(0,x)

    def loss_function(self, y_true, y_pred):
        y_pred[y_pred == 1] = 1 - (1e-8)
        loss = (1./self.sam) * (-np.dot(y_true,np.log(y_pred).T) - np.dot(1-y_true, np.log(1-y_pred).T))
        return loss

    def train(self):
        test_brek = False
        for iteration in range(10000):
            #feed forward
            if iteration % 10 == 0:
                test_loss = self.loss_function(self.test_ground_truth, self.pred(self.test_X))
                # if the loss for test increase, we stop the iteration
                # print(self.y_pred)
                if test_loss > self.test_loss:
                    self.num += 1
                else:
                    self.num = 0
                if self.num == 3:
                    break
                self.test_loss = test_loss

                print("iteration" + str(iteration) + ":" + str(self.y_pred) + "Loss" + str(self.loss) + "test loss:" + str
(self.test_loss))
            before_1 = np.dot(self.weights_1, self.X) + self.bias_1
            layer1_output = self.relu_function(before_1)
            before_2 = np.dot(self.weights_2, layer1_output) + self.bias_2
            layer2_output = self.sigmoid_function(before_2)
            layer2_output[layer2_output == 1] = 1 - (1e-8)
            self.y_pred = layer2_output
            self.loss = self.loss_function(self.ground_truth, self.y_pred)
```

```
        d_y_pred = - np.divide(self.ground_truth, self.y_pred) + np.divide(1-self.ground_truth, 1-self.y_pred)
        d_before_2 = np.multiply(d_y_pred,self.dSigmoid(before_2))
        d_layer1_output = np.dot(self.weights_2.T, d_before_2)
        d_weight_2 = 1./layer1_output.shape[1] * np.dot(d_before_2, layer1_output.T)
        d_bais_2 = 1./layer1_output.shape[1] * np.dot(d_before_2, np.ones([d_before_2.shape[1], 1]))
        d_before_1 = np.multiply(d_layer1_output,self.dRelu(before_1))
        d_input = np.dot(self.weights_1.T, d_before_1)
        d_weight_1 = 1./self.X.shape[1] * np.dot(d_before_1, self.X.T)
        d_bais_1 = 1./self.X.shape[1] * np.dot(d_before_1, np.ones([d_before_1.shape[1],1]))
        self.weights_1 = self.weights_1 - self.learning_rate * d_weight_1
        self.bias_1 = self.bias_1 - self.learning_rate * d_bais_1
        self.weights_2 = self.weights_2 - self.learning_rate * d_weight_2
        self.bias_2 = self.bias_2 - self.learning_rate * d_bais_2



    def pred(self, input):
        before_1 = np.dot(self.weights_1, input) + self.bias_1
        layer1_output = self.relu_function(before_1)
        before_2 = np.dot(self.weights_2, layer1_output) + self.bias_2
        layer2_output = self.sigmoid_function(before_2)
        return layer2_output
```

Prepare the training image

```
def prepare_image(img_name):
    img = cv2.imread(img_name)
    img = grey_scale(img)
    Gx = convolve2d(img,Px)
    Gy = convolve2d(img,Py)
    M = np.sqrt(Gx*Gx + Gy*Gy)
    M = np.round(normalization(M, 255))
    theta = np.zeros(Gx.shape)
    for i in range(theta.shape[0]):
        for j in range(theta.shape[1]):
            if Gy[i,j] == 0 and Gx[i,j] == 0:
                theta[i,j] = 0
            elif Gx[i,j] == 0:
                theta[i,j] = 90
            else:
                theta[i,j] = np.arctan2(Gy[i,j], Gx[i,j]) * 180 / np.pi
                if theta[i,j] < -10:
                    theta[i,j] += 180
                elif theta [i,j] >= 170:
                    theta[i,j] -= 180
    for i in range(theta.shape[0]):
        for j in range(theta.shape[1]):
            if str(theta[i,j]) == "nan":
                print(true)
    hog_vector= get_hist_cell(theta, M)
    return hog_vector
```

Read in the training data, First positive, then negative

```python
# from skimage import io
import os
data_input = []
path = "/Users/shenmengjie/Desktop/Computer Vision/project2/data/Train_Positive/"
path_2 = "/Users/shenmengjie/Desktop/Computer Vision/project2/data/Train_Negative/"

file_dir = os.listdir(path)
for file in file_dir:
    if not os.path.isdir(file):
        file_name = path + file
        # img = cv2.imread(file_name)
        # plotImage(img, "test")
        # print(file_name)
        hog_vector = prepare_image(file_name)
        data_input.append(hog_vector)
        print("img done!")
    else:
        print("cannot open the file!")

# print("positive done!")
file_dir_2 = os.listdir(path_2)
for file in file_dir_2:
    if not os.path.isdir(file):
        file_name = path_2 + file
        hog_vector = prepare_image(file_name)
        data_input.append(hog_vector)
    else:
        print("cannot open the file!")
data_input = np.matrix(data_input)
print("done")
```

Read in the test data

```python
import os
test_input = []
path = "/Users/shenmengjie/Desktop/Computer Vision/project2/data/Test_Positive/"
path_2 = "/Users/shenmengjie/Desktop/Computer Vision/project2/data/Test_Negative/"

file_dir = os.listdir(path)
for file in file_dir:
    if not os.path.isdir(file):
        file_name = path + file
        # img = cv2.imread(file_name)
        # plotImage(img, "test")
        # print(file_name)
        hog_vector = prepare_image(file_name)
        test_input.append(hog_vector)
    else:
        print("cannot open the file!")

# print("positive done!")
file_dir_2 = os.listdir(path_2)
for file in file_dir_2:
    if not os.path.isdir(file):
        file_name = path_2 + file
        # print(file_name)
        # img = cv2.imread(file_name)
        # plotImage(img, "test")
        # print(file_name)
        hog_vector = prepare_image(file_name)
        test_input.append(hog_vector)
    else:
        print("cannot open the file!")
test_input = np.matrix(test_input)
print("done")
```

Train a neural network with layer size of 250 neurons

```python
from numpy import random
HOG_nn = nn(data_input, 250, test_input)
HOG_nn.train()
```

```
  3.36926875e-38 4.69615752e-34 2.83720352e-01 2.55776324e-01]]Loss[[0.22942169]]test loss:[[4.62003373]]
iteration3840:[[9.99996514e-01 1.00000000e+00 1.00000000e+00 9.99999990e-01
  9.99999990e-01 9.99409380e-01 9.98290872e-01 3.23901650e-02
  9.99999997e-01 9.99995268e-01 1.50928894e-13 1.73160495e-15
  2.94447827e-01 7.28142574e-02 4.76652465e-27 5.58483262e-09
  3.44802095e-38 4.80188811e-34 2.82418233e-01 2.55476621e-01]]Loss[[0.22417327]]test loss:[[4.61503073]]
iteration3850:[[9.99996629e-01 1.00000000e+00 1.00000000e+00 9.99999990e-01
  9.99999990e-01 9.99426454e-01 9.98344733e-01 3.58527129e-02
  9.99999997e-01 9.99995408e-01 1.54594465e-13 1.78358253e-15
  2.94078442e-01 7.30110105e-02 4.88790010e-27 5.70044656e-09
  3.52777036e-38 4.90879150e-34 2.81090971e-01 2.55133239e-01]]Loss[[0.21896047]]test loss:[[4.61009612]]
iteration3860:[[9.99996739e-01 1.00000000e+00 1.00000000e+00 9.99999990e-01
  9.99999990e-01 9.99442810e-01 9.98396232e-01 3.96441337e-02
  9.99999997e-01 9.99995542e-01 1.58303364e-13 1.83658229e-15
  2.93626872e-01 7.31833855e-02 5.01038971e-27 5.81675457e-09
  3.60797440e-38 5.01616369e-34 2.79734152e-01 2.54719236e-01]]Loss[[0.21378614]]test loss:[[4.60522736]]
iteration3870:[[9.99996844e-01 1.00000000e+00 1.00000000e+00 9.99999990e-01
  9.99999990e-01 9.99458502e-01 9.98445530e-01 4.37866823e-02
  9.99999997e-01 9.99995670e-01 1.62050596e-13 1.89055085e-15
  2.93101282e-01 7.33336512e-02 5.13408018e-27 5.93355021e-09
  3.68870062e-38 5.12406857e-34 2.78343104e-01 2.54242154e-01]]Loss[[0.20865599]]test loss:[[4.60042261]]
iteration3880:[[9.99996945e-01 1.00000000e+00 1.00000000e+00 9.99999990e-01
  9.99999990e-01 9.99473577e-01 9.98492771e-01 4.83073181e-02
  9.99999998e-01 9.99995793e-01 1.65830363e-13 1.94542292e-15
  2.92509075e-01 7.34638903e-02 5.25905072e-27 6.05059838e-09
  3.77001200e-38 5.23256109e-34 2.76912732e-01 2.53708861e-01]]Loss[[0.2035706]]test loss:[[4.59570081]]
iteration3890:[[9.99997042e-01 1.00000000e+00 1.00000000e+00 9.99999990e-01
  9.99999990e-01 9.99487978e-01 9.98537825e-01 5.32232457e-02
  9.99999998e-01 9.99995911e-01 1.69636314e-13 2.00112420e-15
  2.91820036e-01 7.35645888e-02 5.38443459e-27 6.16764669e-09
```

```python
result = HOG_nn.pred(test_input.T)
#print probability result
print(result)
```

Train a neural network with layer size of 500 neurons

```python
# random.seed(1)
HOG_nn_500 = nn(data_input, 500, test_input)
HOG_nn_500.train()
```

```
  9.99999990e-01 9.99999990e-01 2.51232512e-24 1.00000000e+00
  9.99999990e-01 1.00000000e+00 9.99999990e-01 4.40080746e-04
  4.28632562e-07 5.63272443e-17 9.99999990e-01 9.99999990e-01]]Loss[[7.48660127]]test loss:[[7.01657636]]
iteration210:[[9.99654980e-01 9.99999990e-01 1.00000000e+00 1.00000000e+00
  9.97615565e-01 4.37097829e-03 1.36018126e-07 9.99999990e-01
  9.99999990e-01 9.99999990e-01 2.61731701e-24 1.00000000e+00
  9.99999990e-01 1.00000000e+00 9.99999990e-01 4.65211102e-04
  4.60648068e-07 6.30139707e-17 9.99999990e-01 9.99999990e-01]]Loss[[7.43901032]]test loss:[[7.01081253]]
iteration220:[[9.99677450e-01 9.99999990e-01 1.00000000e+00 1.00000000e+00
  9.97865997e-01 5.76620115e-03 1.83499583e-07 9.99999990e-01
  9.99999990e-01 9.99999990e-01 2.72565864e-24 1.00000000e+00
  9.99999990e-01 1.00000000e+00 9.99999990e-01 4.91553385e-04
  4.94832720e-07 7.04600966e-17 9.99999990e-01 9.99999990e-01]]Loss[[7.39145285]]test loss:[[7.00508956]]
iteration230:[[9.99698289e-01 9.99999990e-01 1.00000000e+00 1.00000000e+00
  9.98088872e-01 7.59667972e-03 2.47408831e-07 9.99999990e-01
  9.99999990e-01 9.99999990e-01 2.83716347e-24 1.00000000e+00
  9.99999990e-01 1.00000000e+00 9.99999990e-01 5.19097607e-04
  5.31262031e-07 7.87386874e-17 9.99999990e-01 9.99999990e-01]]Loss[[7.34403389]]test loss:[[6.99941661]]
iteration240:[[9.99717381e-01 9.99999990e-01 1.00000000e+00 1.00000000e+00
  9.98287087e-01 9.99127107e-03 3.33329963e-07 9.99999990e-01
  9.99999990e-01 9.99999990e-01 2.95151991e-24 1.00000000e+00
  9.99999990e-01 1.00000000e+00 9.99999990e-01 5.47803535e-04
  5.69982766e-07 8.79236357e-17 9.99999990e-01 9.99999990e-01]]Loss[[7.29653657]]test loss:[[6.99380588]]
iteration250:[[9.99735034e-01 9.99999990e-01 1.00000000e+00 1.00000000e+00
  9.98463192e-01 1.31121440e-02 4.48672930e-07 9.99999990e-01
  9.99999990e-01 9.99999990e-01 3.06824936e-24 1.00000000e+00
  9.99999990e-01 1.00000000e+00 9.99999990e-01 5.77589049e-04
  6.10998344e-07 9.80864637e-17 9.99999990e-01 9.99999990e-01]]Loss[[7.24916083]]test loss:[[6.98827347]]
iteration260:[[9.99751311e-01 9.99999990e-01 1.00000000e+00 1.00000000e+00
  9.98619436e-01 1.71600313e-02 6.03218047e-07 9.99999990e-01
```

```python
result = HOG_nn_500.pred(test_input.T)
print("probability result:")
print(result)
result[result >= 0.5] = 1
result[result < 0.5] = 0
print(result)
```

```
probability result:
[[9.99999997e-01 3.12256353e-14 1.00000000e+00 1.50151626e-23
  1.00000000e+00 1.10860741e-06 1.00000000e+00 4.46865026e-36
  1.00000000e+00 1.00000000e+00]]
[[1. 0. 1. 0. 1. 0. 1. 0. 1. 1.]]
```

Train a neural network with layer size of 1000 neurons

```python
HOG_nn_1000 = nn(data_input, 1000, test_input)
HOG_nn_1000.train()
```

```
iteration90:[[1.12923625e-47 9.99999990e-01 9.99999990e-01 9.99999990e-01
  9.99999990e-01 9.09985621e-01 9.99999990e-01 9.99999990e-01
  9.99999990e-01 9.99999990e-01 4.04783769e-11 9.99999990e-01
  9.99999990e-01 1.00000000e+00 9.99999990e-01 9.99999990e-01
  1.00000000e+00 7.45002898e-02 1.53443415e-22 9.99999990e-01]]Loss[[13.01132954]]test loss:[[9.00862934]]
iteration100:[[1.25042741e-47 9.99999990e-01 9.99999990e-01 9.99999990e-01
  9.99999990e-01 8.58724075e-01 9.99999990e-01 9.99999990e-01
  9.99999990e-01 9.99999990e-01 2.41943477e-11 9.99999990e-01
  9.99999990e-01 1.00000000e+00 9.99999990e-01 9.99999990e-01
  1.00000000e+00 4.61244945e-02 8.50609192e-23 9.99999990e-01]]Loss[[12.90373193]]test loss:[[8.97836778]]
iteration110:[[1.45166486e-47 9.99999990e-01 9.99999990e-01 1.00000000e+00
  9.99999990e-01 7.97150056e-01 9.99999990e-01 9.99999990e-01
  9.99999990e-01 9.99999990e-01 1.50245981e-11 9.99999990e-01
  9.99999990e-01 1.00000000e+00 9.99999990e-01 9.99999990e-01
  1.00000000e+00 2.99010628e-02 4.93413927e-23 9.99999990e-01]]Loss[[12.79894133]]test loss:[[8.95317263]]
iteration120:[[1.76559054e-47 9.99999990e-01 9.99999990e-01 1.00000000e+00
  9.99999990e-01 7.32359982e-01 9.99999990e-01 9.99999990e-01
```

```python
result = HOG_nn_1000.pred(test_input.T)
print("probability result:")
print(result)
result[result >= 0.5] = 1
result[result < 0.5] = 0
print(result)
```

```
probability result:
[[2.47667760e-09 9.99499761e-01 1.00000000e+00 7.38954563e-41
  9.89073402e-01 9.64289540e-01 6.64197595e-06 1.00000000e+00
  9.99997632e-01 1.00000000e+00]]
[[0. 1. 1. 0. 1. 1. 0. 1. 1. 1.]]
```