HOG

March 19, 2022

1 Lab Assignment 3

Computer Vision - Term 5, 2022

Instructor: Dr. Saumya Jetly TA: Ribhu Lahiri

Deadline: Friday, 18 March 2022 11:59 pm

Submission form link: https://forms.gle/aLT9AqtasQemhWMN9

Total points: 5 (with possible extra credit)

1.0.1 Before Starting

Please download the FER-2013 dataset here: https://drive.google.com/file/d/1GTX0XYKtyOo9VADHL_AjLcp59

You will need it for the tasks that follow.

[1]:
```
from google.colab import drive
drive.mount ('/content/drive' )
```

```
Mounted at /content/drive
```

[2]:
```
! unzip /content/drive/MyDrive/Colab\ Notebooks/HOG/fer2013.zip
```

```
Archive:  /content/drive/MyDrive/Colab Notebooks/HOG/fer2013.zip
  inflating: fer2013.csv
  inflating: __MACOSX/._fer2013.csv
```

[3]:
```
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from skimage import io
from skimage import color
from skimage.transform import resize
from skimage.feature import hog
```

[4]:
```
# Reading in the dataset
df = pd.read_csv('./fer2013.csv')
df.head()
```

[4]:

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121…	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15…	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38…	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1…	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84…	Training

[5]:
```
# Extract relevant data
x_data = pd.Series(df.pixels)
y_data = pd.Series(df.emotion)
x_data = np.array(list(map(str.split, x_data)), np.float32)
x_data/=255 # Normalizing
```

[6]:
```
# Reshape into 48x48 images
x_data = x_data.reshape(-1, 48, 48)
x_data.shape
```

[6]:
```
(35887, 48, 48)
```

1.0.2 Task 1: Creating histogram of Oriented Gradients (4 points)

The first task is to create the histogram of oriented gradients feature descriptor. There are two steps to it, the first is to create the oriented gradients from a given image which returns the magnitude and angle matrices. The next step then, is to use those matrices to create the histogram features.

You can refer to this blog as discussed in the lab session: https://iq.opengenus.org/object-detection-with-histogram-of-oriented-gradients-hog/

Implement the `get_oriented_gradients` function *(2 points)*

Use any sample image from the dataset to test out your function

[7]:
```
# Read in any image from the dataset
img = x_data[2]
plt.figure(figsize=(15, 8))
plt.imshow(img, cmap="gray")
plt.axis("off")
plt.show()
```

[8]:
```
img = np.array(img)
def get_oriented_gradients(img):
    '''
    Returns the magnitude and angles of the oriented gradients
    from the given image

    Parameters
    ----------
    img: np.ndarray
        The given image

    Returns
    -------
```

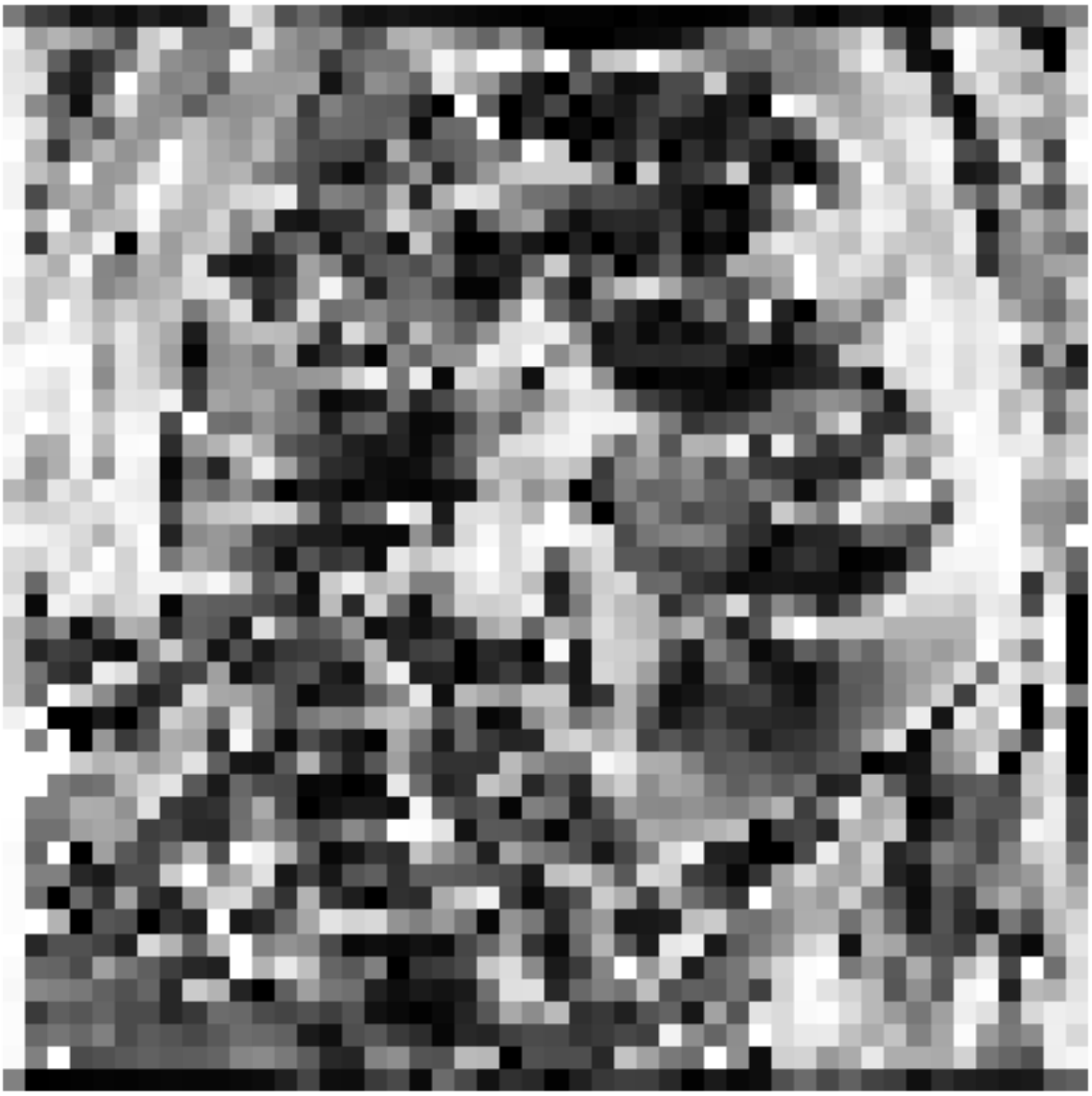
```
    mag: np.ndarray
        2-D Numpy array that contains the magnitudes of the oriented gradients

    theta: np.ndarray
        2-D Numpy array that contains the angles of the oriented gradients
    '''
    # Zero padding the image to make sure we can calculate gradients to the␣
↪corner pixels
    # Zero padding along rows
    import math

    # Here we can either ignore the top most pixels or do padding
    # so that all pixels are considered. In this case, we are doing zero padding
    # to make sure the input and output are of the same size.
    img = np.hstack ((img, np.zeros((img.shape[0],1))))
    img = np.hstack (( np.zeros((img.shape[0],1)), img ))
    # Zero padding along columns
    img = np.vstack((img, np.zeros((1,img.shape[1]))))
    img = np.vstack((np.zeros((1,img.shape[1])),img ))
    image_G = []
    image_theta = []

    for i in range (1 , img.shape[0]-1):
      G_rows = []
      theta_rows = []
      for j in range ( 1, img.shape[1]-1):
        dx = abs (img[i-1][j] - img [i+1][j] )
        dy = abs( img[i][j-1]  - img[i][j+1] )

        G = math.sqrt ((dx**2)+(dy**2))
        if dx ==0:
          theta = np.pi/2  # As x approaches infinity, of arctan(x) is Pi/2 .
        else:
          theta = np.arctan(dy/dx)

        G_rows.append(G)
        theta_rows.append(theta)

      image_G.append(G_rows)
      image_theta.append(theta_rows)

    return np.array(image_G), np.array(image_theta)
```

[9]:
```
mag, theta = get_oriented_gradients(img)
```

[10]:
```
plt.figure(figsize=(15, 8))
plt.imshow(mag, cmap="gray")
```

```
plt.axis("off")
plt.show()
```

[11]:
```
plt.figure(figsize=(15, 8))
plt.imshow(theta, cmap="gray")
plt.axis("off")
plt.show()
```



[12]:
```
m_viz = (mag > 0.5).astype(float)
plt.figure(figsize=(15, 8))
plt.imshow(m_viz, cmap="gray")
plt.axis("off")
plt.show()
```

Implement the `hog_features` function *(2 points)*

Use the image, magnitudes, and angles of oriented gradients to bin them and create the HoG features to be used by our classifier

NOTE: While I have set the default value of number of bins as 9. Feel free to experiment and observe the effects of changing it on the downstream task

1.0.3 Reference : Histogram of Oriented Gradients Research Paper by Carlo Tomasi

https://courses.cs.duke.edu/compsci527/fall15/notes/hog.pdf

[61]:

```
def cell_orientation_histograms(mag,theta,nbins = 9):
  assert theta.shape==mag.shape, "The mag and theta should be of same␣
 ↪dimensions"

  features = np.zeros (nbins)

  for i in range(theta.shape[0]):
    for j in range(theta.shape[1]):
      magnitude = mag[i][j]
      angle = math.degrees(theta[i][j])
      # Finding the bin to which it should be allocated
      bin_width = 180/nbins
      j_bin = math.floor(((angle /bin_width)-0.5))


      cj =  (bin_width/2)*((2*j_bin)+1)
      cj1 = (bin_width/2)*((2*(j_bin+1))+1)

      vj1 = magnitude * ( (cj1 - angle ) / bin_width )

      vj2 = magnitude *( (angle - cj )/bin_width )

      features[j_bin] += np.round(vj1,4)
      features[j_bin+1] += np.round(vj2,4)
  return features
```

[62]:
```
def sliding_window(img,window_size = 8):
  ''' Function to apply sliding window over the image
  Default window size is 8 '''

  length_of_a_box = 8
  number_of_boxes = 2
  n =length_of_a_box

  # Divide the picture into boxes of size 8x8
  features= []
  count =0
  for i in range(0,img.shape[0],n):
    for j in range(0,img.shape[1],n):
        sub_image = img[i:i+n,j:j+n]
        sub_feature = []
        count +=1
        for i_ in range(0,sub_image.shape[0],n//2):
          for j_ in range(0,sub_image.shape[0],n//2):
            magnitude , angle = get_oriented_gradients(sub_image[i_:
 ↪i_+length_of_a_box,j_:j_+length_of_a_box])
```

```
            sub_feature += list(cell_orientation_histograms(np.
↪array(magnitude), np.array(angle)))
        features+= sub_feature

    # Normalising
    features = [value/math.sqrt(sum(np.square(features))) for value in features]
    return np.array(features)
```

[15]:
```
def hog_features(img, mag = None, theta = None,nbins=9,):
    '''
      Create the histogram of oriented gradients feature vector

      Parameters
      ----------
      img: np.ndarray
          The given image

      nbins: int
          The number of buckets (bins) in which to organise the oriented gradients

      mag: np.ndarray
          2-D Numpy array that contains the magnitudes of the oriented gradients

      theta: np.ndarray
          2-D Numpy array that contains the angles of the oriented gradients

      Returns
      -------
      np.ndarray
          The HoG features
          [given input of 48x48 images and 9 bins should be 1296]

    '''
    return sliding_window(img)
```

1.0.4 Task 2: Using HoG with AdaBoost Classifier (1 points)

The second task is to create an AdaBoost classifier that uses the HoG features to classify the given images into one of 7 classes representing different emotions.

Extra Credit: Implement the AdaBoost Classifier from scratch, i.e. without using scikit-learn (1 points)

[93]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x_data, y_data,
                                                    test_size=0.3,
                                                    random_state=42)
X_train.shape
```

[93]:
```
(25120, 48, 48)
```

[94]:
```
# Training was done only on a smaller portin of data due to the execution time

from tqdm import tqdm
X_train = np.array([hog_features(im) for im in X_train[0:476]])
# np.save('x_train.npy',X_train)

X_test = np.array([hog_features(im) for im in X_test[0:100]])
# np.save('/content/drive/MyDrive/Colab Notebooks/HOG/x_test.npy',X_test)
```

[215]:
```
# X_train = np.load('/content/drive/MyDrive/Colab Notebooks/HOG/x_train.npy')
# X_test = np.load('/content/drive/MyDrive/Colab Notebooks/HOG/x_test.npy')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:24: RuntimeWarning:
invalid value encountered in double_scalars
```

[ ]:
```
# In case you are attempting the extra-credit,
# use this class as reference for the bare-minimum functions
# you need to build

class AdaBoost:

  '''
    AdaBoost Classifier
  '''
  def __init__(self):
    pass

  def fit(self):
    pass

  def predict(self):
    pass
```

[97]:
```
from sklearn.ensemble import AdaBoostClassifier
n  = 476

clf = AdaBoostClassifier()
clf.fit(X_train[0:n], y_train[0:n])
```

[97]:
```
AdaBoostClassifier()
```

[98]:
```
from sklearn.metrics import classification_report

y_pred_train = clf.predict(X_train[0:n])
print(classification_report(y_train[0:n], y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.27      0.23      0.25        57
           1       1.00      0.17      0.29        12
           2       0.32      0.31      0.31        72
           3       0.37      0.52      0.43       120
           4       0.19      0.14      0.16        66
           5       0.34      0.28      0.31        57
           6       0.34      0.35      0.34        92

    accuracy                           0.33       476
   macro avg       0.40      0.28      0.30       476
weighted avg       0.33      0.33      0.32       476
```

[129]:
```
y_pred_test = clf.predict(X_test)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.30      0.23      0.26        13
           2       0.12      0.12      0.12        16
           3       0.32      0.33      0.33        27
           4       0.10      0.13      0.11        15
           5       0.33      0.18      0.24        11
           6       0.16      0.17      0.16        18

    accuracy                           0.21       100
   macro avg       0.22      0.20      0.20       100
weighted avg       0.22      0.21      0.21       100
```