Assignment1

October 6, 2023

[ ]:

```python
# IMPORT ALL DEPENDENCIES

import os
import cv2
from matplotlib import pyplot as plt
import xml.etree.ElementTree as ET
from PIL import Image
import numpy as np
import math
from scipy.spatial import distance as distance_module
```

[ ]:

```python
# DECLARING ALL PATHS AND VARIABLES.

DATA_DIR = "\\".join(os.getcwd().split("\\")[:-1]) + "\\" + "DataSet"
ANNOTATIONS_DIR = DATA_DIR + "\\" + "Annotations\\"
IMG_DIR = DATA_DIR + "\\" + "Images\\"
PROCESSESED_PATH = DATA_DIR + '\\' + 'ProcessedDatasets\\'
AVAILABLE_CLASSES = ['n02089078-black-and-tan_coonhound','n02091831-Saluki'
                     ,'n02092002-Scottish_deerhound',
                     'n02095314-wire-haired_fox_terrier']



CLASS_NAMES = []
CLASS_CODES = []
ANNOTATION_PATHS = []
IMAGE_PATHS = []
PROCESSED_IMAGE_PATHS = []



for i in range(4):
    CLASS_NAMES.append("-".join(AVAILABLE_CLASSES[i].split("-")[1:]))
    CLASS_CODES.append(AVAILABLE_CLASSES[i].split("-")[0])
    ANNOTATION_PATHS.append(ANNOTATIONS_DIR + AVAILABLE_CLASSES[i] + "\\" )
    IMAGE_PATHS.append(IMG_DIR + AVAILABLE_CLASSES[i] + "\\")
    PROCESSED_IMAGE_PATHS.append(PROCESSESED_PATH + CLASS_CODES[i]+"-"␣
 ↪+CLASS_NAMES[i]+ "\\")
```

[ ]:

```
# FUNCTION TO PROCESS THE IMAGE BASED ON THE CORRESPONDING ANNOTATIONS.

def get_bounding_boxes(annot):
  xml = annot
  tree = ET.parse(xml)
  root = tree.getroot()
  objects = root.findall('object')
  bbox = []
  for o in objects:
    bndbox = o.find('bndbox')
    xmin = int(bndbox.find('xmin').text)
    ymin = int(bndbox.find('ymin').text)
    xmax = int(bndbox.find('xmax').text)
    ymax = int(bndbox.find('ymax').text)
    bbox.append((xmin,ymin,xmax,ymax))
  return bbox

#FUNCTION TO CROP EVERY IMAGE IN EVERY CLASS AND SAVE IN A PROCESSED DIRECTORY.

def crop_image(image_path , annotation_path,save_path):
  img = cv2.imread(image_path)
  bb = get_bounding_boxes(annotation_path)
  bbox = bb[0]
  cropped_data = img[bbox[1]:bbox[3], bbox[0]:bbox[2]]    # cropping the image
  cropped_data = cv2.resize(cropped_data,dsize=(331 ,331),interpolation=cv2.
 ↪INTER_CUBIC) # rescaling it to a square image
  crop_img = Image.fromarray(cropped_data,'RGB')  # converting the numpy array␣
 ↪to an image
  crop_img.save(save_path)
```

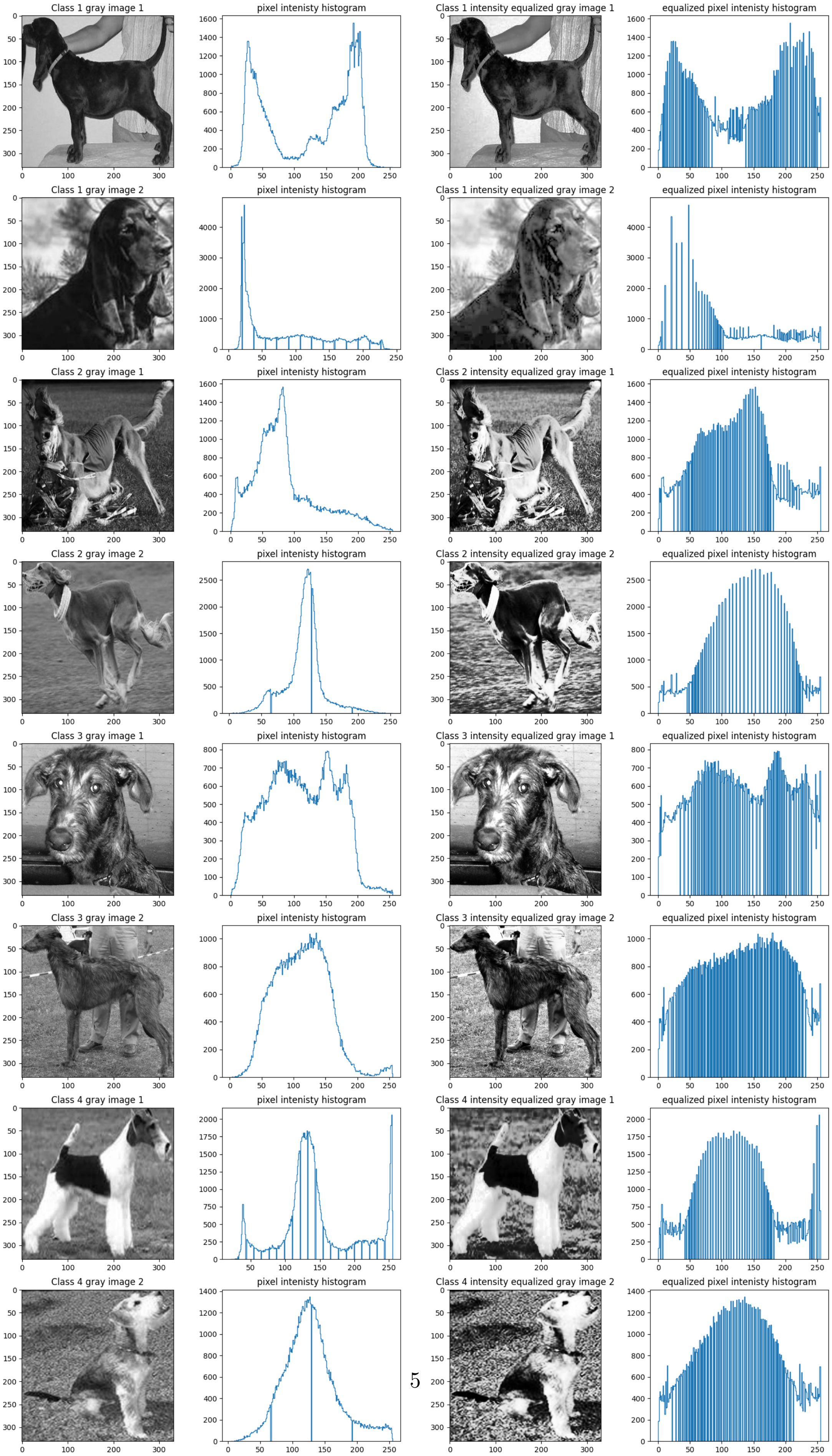
[ ]:

```
for i in range(4):
    for dog in os.listdir(IMAGE_PATHS[i]):
        image_path = dog
        annotation_path = dog.split(".")[0]
        if not os.path.exists(PROCESSED_IMAGE_PATHS[i]):
            os.mkdir(PROCESSED_IMAGE_PATHS[i])
        crop_image(IMAGE_PATHS[i]+  image_path, ANNOTATION_PATHS[i]
                   + annotation_path, PROCESSED_IMAGE_PATHS[i] + dog)
```

[ ]:

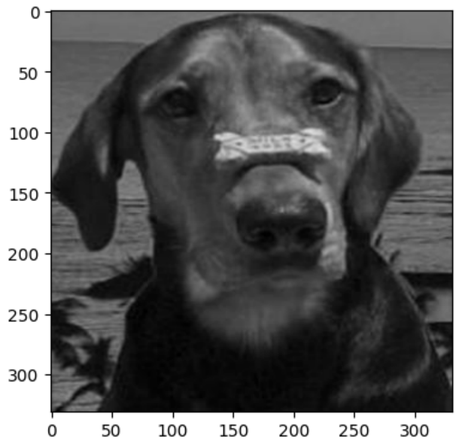
```
#Refered yagnaVNK/ Data-Mining-1/Assignment1 repository

import os
```

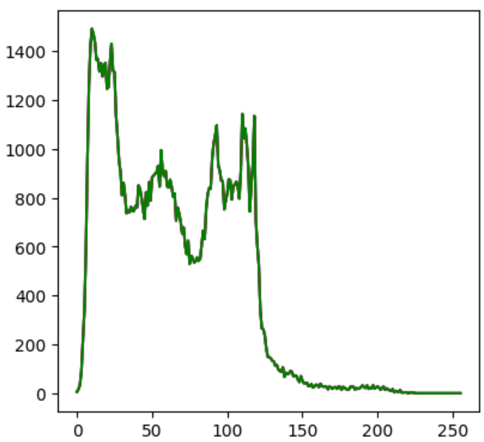
```
fig = plt.figure(figsize=(20, 35))

rows = 8
columns = 4


for i in range(4):

    img1 = cv2.imread(PROCESSED_IMAGE_PATHS[i]
                      +  os.listdir(PROCESSED_IMAGE_PATHS[i])[np.random.
 ↪randint(0,40)] )
    img2 = cv2.imread(PROCESSED_IMAGE_PATHS[i]
                      +  os.listdir(PROCESSED_IMAGE_PATHS[i])[np.random.
 ↪randint(0,40)] )
    img1_gray = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
    img2_gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
    arr_1 =img1_gray.flatten()
    arr_2 =img2_gray.flatten()

    fig.add_subplot(rows,columns,8*i+1)
    plt.imshow(img1_gray,cmap='gray')
    plt.title("Class "+str(i+1)+" gray image 1")


    fig.add_subplot(rows,columns,2*(4*i+1))
    plt.hist(arr_1,bins=255,histtype='step')
    plt.title("pixel intenisty histogram")

    fig.add_subplot(rows,columns,8*i+5)
    plt.imshow(img2_gray,cmap='gray')
    plt.title("Class "+str(i+1)+" gray image 2")

    fig.add_subplot(rows,columns,2*(4*i+3))
    plt.hist(arr_2,bins=255,histtype='step')
    plt.title("pixel intenisty histogram")




    img1_eq = cv2.equalizeHist(img1_gray)
    img2_eq = cv2.equalizeHist(img2_gray)
    arr_1_eq =img1_eq.flatten()
    arr_2_eq =img2_eq.flatten()

    fig.add_subplot(rows,columns,8*i+3)
    plt.imshow(img1_eq,cmap='gray')
    plt.title("Class "+str(i+1)+" intensity equalized gray image 1")
    fig.add_subplot(rows,columns,4*(2*i+1))
    plt.hist(arr_1_eq,bins=255,histtype='step')
    plt.title("equalized pixel intenisty histogram")

    fig.add_subplot(rows,columns,8*i+7)
    plt.imshow(img2_eq,cmap='gray')
    plt.title("Class "+str(i+1)+" intensity equalized gray image 2")

    fig.add_subplot(rows,columns,8*(i+1))
    plt.hist(arr_2_eq,bins=255,histtype='step')
    plt.title("equalized pixel intenisty histogram")
```

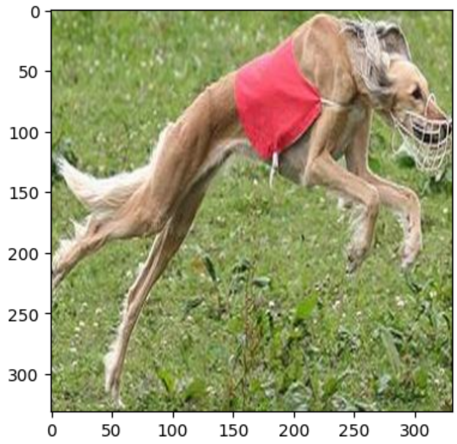

Class 1 gray image 1
pixel intenisty histogram
Class 1 intensity equalized gray image 1
equalized pixel intenisty histogram
Class 1 gray image 2
pixel intenisty histogram
Class 1 intensity equalized gray image 2
equalized pixel intenisty histogram
Class 2 gray image 1
pixel intenisty histogram
Class 2 intensity equalized gray image 1
equalized pixel intenisty histogram
Class 2 gray image 2
pixel intenisty histogram
Class 2 intensity equalized gray image 2
equalized pixel intenisty histogram
Class 3 gray image 1
pixel intenisty histogram
Class 3 intensity equalized gray image 1
equalized pixel intenisty histogram
Class 3 gray image 2
pixel intenisty histogram
Class 3 intensity equalized gray image 2
equalized pixel intenisty histogram
Class 4 gray image 1
pixel intenisty histogram
Class 4 intensity equalized gray image 1
equalized pixel intenisty histogram
Class 4 gray image 2
pixel intenisty histogram
Class 4 intensity equalized gray image 2
equalized pixel intenisty histogram
0
50
100
150
200
250
300
0
200
400
600
800
1000
1200
1400
1600
0
1000
2000
3000
4000
0
500
1000
1500
2000
2500
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750
2000
0
200
400
600
800
1000
1200
1400

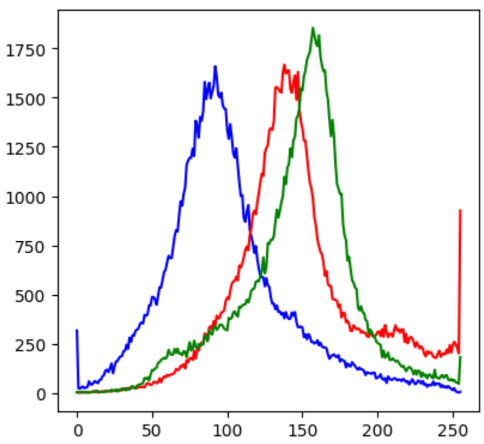
RGB Histogram

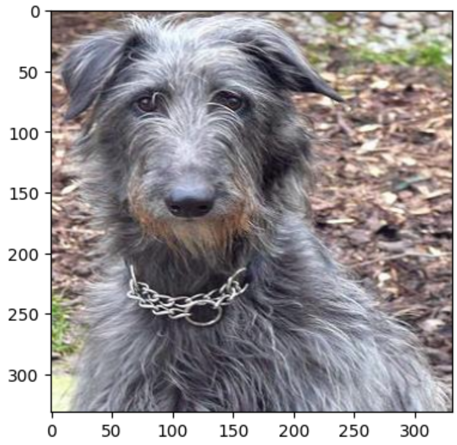
```
[ ]: fig = plt.figure(figsize=(10, 20))

     rows = 4
     columns = 2



     for i in range(4):

         img = cv2.imread(PROCESSED_IMAGE_PATHS[i] +  os.
      ↪listdir(PROCESSED_IMAGE_PATHS[i])[np.random.randint(0,40)] ,cv2.IMREAD_COLOR)
         fig.add_subplot(rows,columns,2*(i)+1)
         plt.imshow(img)
         img_hist_blue = cv2.calcHist([img],[2],None,[256],[0,256])
         img_hist_red = cv2.calcHist([img],[0],None,[256],[0,256])
         img_hist_green = cv2.calcHist([img],[1],None,[256],[0,256])


         fig.add_subplot(rows,columns,2*(i+1))
         plt.plot(img_hist_blue,color='blue')
         plt.plot(img_hist_red,color='red')
         plt.plot(img_hist_green,color='green')
```

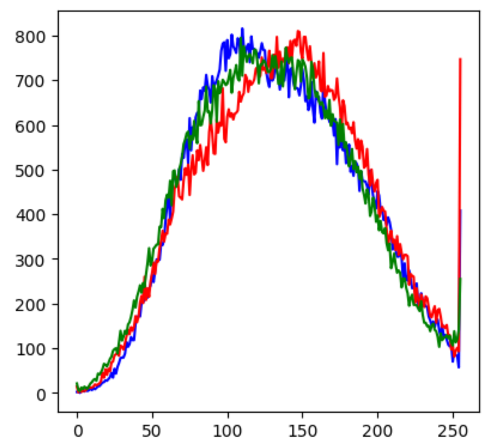

0
50
100
150
200
250
300
0
50
100
150
200
250
300

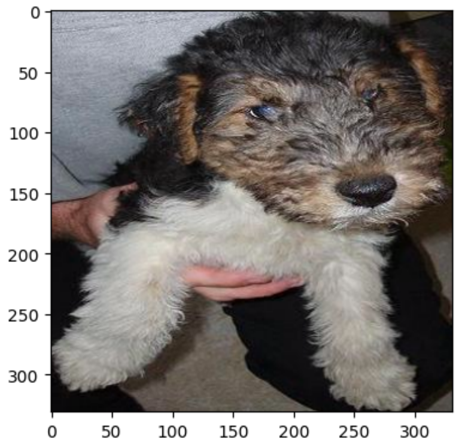

1400
1200
1000
800
600
400
200
0
0
50
100
150
200
250

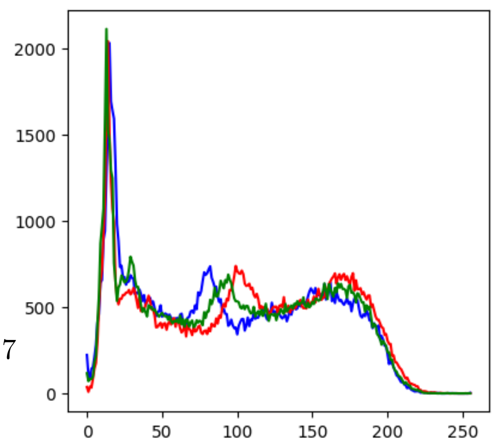

0
50
100
150
200
250
300
0
50
100
150
200
250
300


1750
1500
1250
1000
750
500
250
0
0
50
100
150
200
250


0
50
100
150
200
250
300
0
50
100
150
200
250
300


800
700
600
500
400
300
200
100
0
0
50
100
150
200
250


0
50
100
150
200
250
300
0
50
100
150
200
250
300


2000
1500
1000
500
0
0
50
100
150
200
250

Histogram Comparison

[ ]:

```
class_1 = np.random.randint(0,1)
class_2 = np.random.randint(2,3)
img1_i = np.random.randint(0,40)
img2_i = np.random.randint(41,80)
img3_i = np.random.randint(0,50)


img1_class_1 = cv2.imread(PROCESSED_IMAGE_PATHS[class_1]
                          +  os.listdir(PROCESSED_IMAGE_PATHS[class_1])[img1_i]␣
 ↪)
img2_class_1 = cv2.imread(PROCESSED_IMAGE_PATHS[class_1]
                          +  os.listdir(PROCESSED_IMAGE_PATHS[class_1])[img2_i]␣
 ↪)
img3_class_2 = cv2.imread(PROCESSED_IMAGE_PATHS[class_2]
                          +  os.listdir(PROCESSED_IMAGE_PATHS[class_2])[img3_i]␣
 ↪)

img1_gray_class_1 = cv2.cvtColor(img1_class_1,cv2.COLOR_BGR2GRAY)
img2_gray_class_1 = cv2.cvtColor(img2_class_1,cv2.COLOR_BGR2GRAY)
img3_gray_class_2 = cv2.cvtColor(img3_class_2,cv2.COLOR_BGR2GRAY)



img1_eq = cv2.equalizeHist(img1_gray_class_1)
img2_eq = cv2.equalizeHist(img2_gray_class_1)
img3_eq = cv2.equalizeHist(img3_gray_class_2)


hist1 = cv2.calcHist([img1_eq], [0], None, [256], [0, 256])
hist2 = cv2.calcHist([img2_eq], [0], None, [256], [0, 256])
hist3 = cv2.calcHist([img3_eq], [0], None, [256], [0, 256])


arr1= []
for i in hist1:
    arr1.append(i[0])

arr2= []
for i in hist2:
    arr2.append(i[0])

arr3= []
for i in hist3:
    arr3.append(i[0])
```

Euclidian Distance

Same Class 2 images

[ ]:

```
sum = 0
for i in range (0,256):
    sum = sum + (hist1[i][0]-hist2[i][0])**2
dist = math.sqrt(sum)
print('euclidean distance of same classes:', dist)


sum = 0
for i in range (0,256):
    sum = sum + (hist1[i][0]-hist3[i][0])**2
dist = math.sqrt(sum)
print('euclidean distance of different classes:', dist)


arr1= []
for i in hist1:
    arr1.append(i[0])

arr2= []
for i in hist2:
    arr2.append(i[0])

arr3= []
for i in hist3:
    arr3.append(i[0])


print(distance_module.euclidean(arr1, arr2))
print(distance_module.euclidean(arr1, arr3))
```

```
euclidean distance of same classes: 6921.279505987314
euclidean distance of different classes: 5782.536813544727
6921.279296875
5782.53662109375
```

[ ]:

```
from scipy.spatial.distance import cityblock

sum = 0
for i in range (0,256):
    sum = sum + abs(hist1[i][0]-hist2[i][0])
print('manhattan distance of same classes:', sum)
```

```
sum = 0
for i in range (0,256):
    sum = sum + abs(hist1[i][0]-hist3[i][0])
print('manhattan distance of different classes:', sum)



print(cityblock(arr1,arr2))
print(cityblock(arr1,arr3))
```

```
manhattan distance of same classes: 88070.0
manhattan distance of different classes: 70388.0
88070.0
70388.0
```

[ ]:

```
print(cv2.compareHist(hist1,hist2,cv2.HISTCMP_BHATTACHARYYA))
print(cv2.compareHist(hist1,hist3,cv2.HISTCMP_BHATTACHARYYA))

#bhattacharya distance of same classes: 0.34421483
#bhattacharya distance of different classes: 0.46494624

#0.5396511225453978
#0.6097795550313342
```

```
0.551864116438768
0.4743432080591224
```

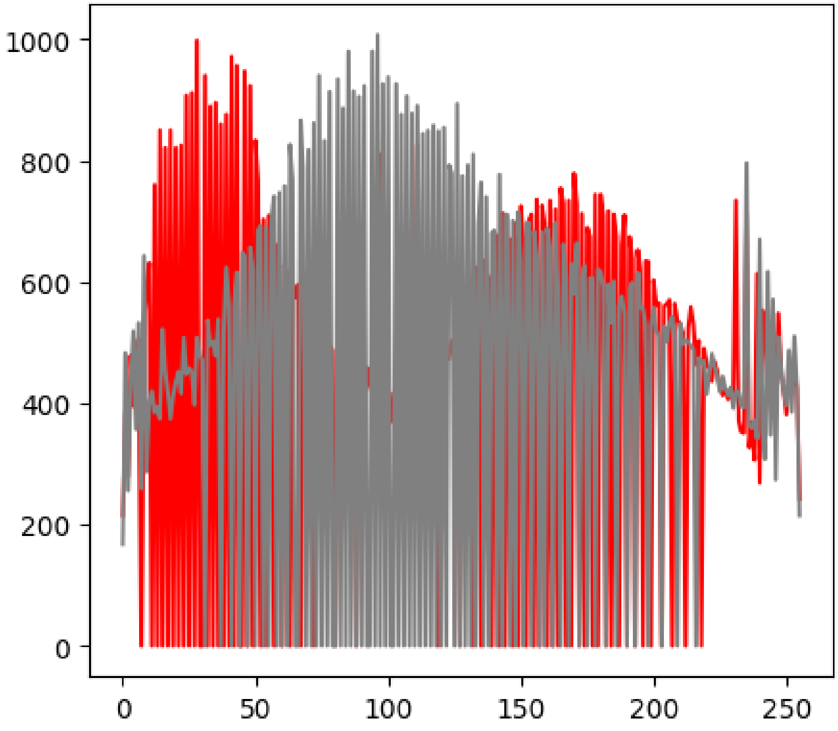
[ ]:

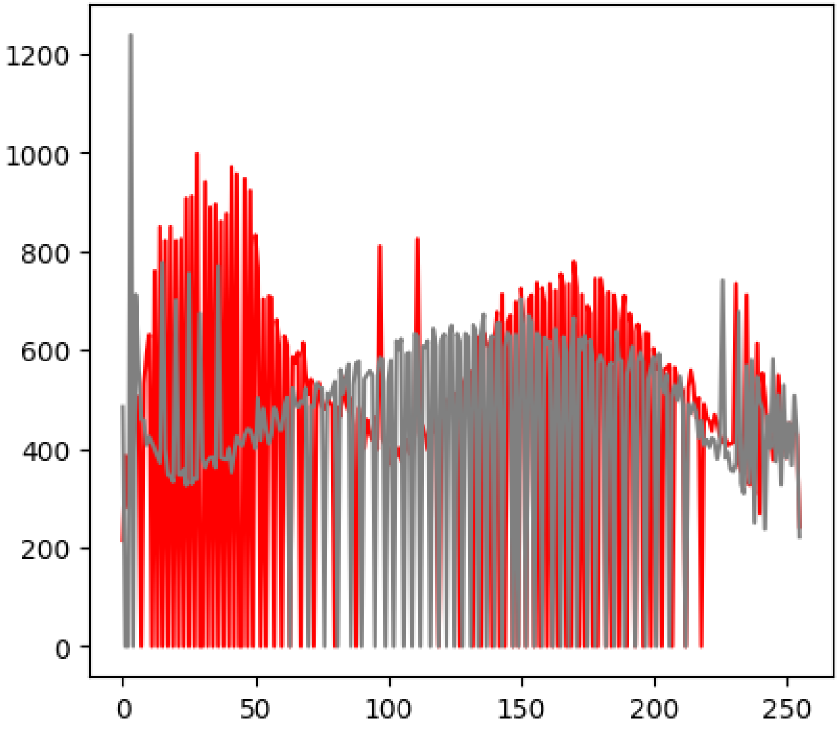
```
fig = plt.figure(figsize=(5, 10))

rows = 2
columns = 1

print(cv2.compareHist(hist1,hist2,cv2.HISTCMP_INTERSECT))
print(cv2.compareHist(hist1,hist3,cv2.HISTCMP_INTERSECT))

fig.add_subplot(rows,columns,1)
plt.plot(hist1,color = 'red')
plt.plot(hist2,color = 'grey')

fig.add_subplot(rows,columns,2)
plt.plot(hist1,color = 'red')
plt.plot(hist3,color = 'grey')
```

```
65526.0
74367.0
```

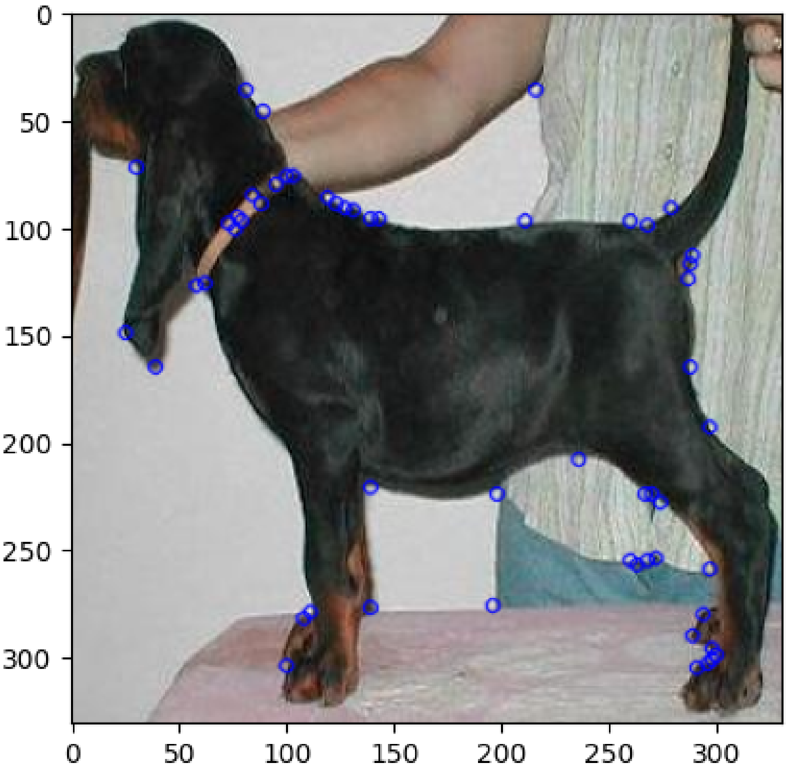
[ ]:

```
[<matplotlib.lines.Line2D at 0x198b9ac53d0>]
```


1000
800
600
400
200
0
0
50
100
150
200
250


1200
1000
800
600
400
200
0
0
50
100
150
200
250

```
[ ]: edge_threshold = 25
     patch_size = 20
     keypoints = 55

     img = cv2.imread(PROCESSED_IMAGE_PATHS[class_1] +
                      os.listdir(PROCESSED_IMAGE_PATHS[class_1])[7] )
     orb = cv2.ORB_create(edgeThreshold= edge_threshold,
                          patchSize=patch_size, nlevels=1,
                          fastThreshold=20,scaleFactor=2,
                          WTA_K=4,scoreType=cv2.ORB_HARRIS_SCORE
                          ,firstLevel=0, nfeatures=keypoints)
     # find the keypoints with ORB
     kp = orb.detect(img,None)
     # compute the descriptors with ORB
     kp, des = orb.compute(img, kp)
     # draw only keypoints location,not size and orientation
     img2 = cv2.drawKeypoints(img, kp, None, color=(0,0,255), flags=0)
     plt.imshow(img2), plt.show()
```



```
[ ]: (<matplotlib.image.AxesImage at 0x198b99a6390>, None)
```

[ ]:

```
dataset_1 = []

dataset_2 = []

for dog in os.listdir(PROCESSED_IMAGE_PATHS[1]):
    img1_eq = cv2.imread(PROCESSED_IMAGE_PATHS[1] + dog,cv2.IMREAD_GRAYSCALE)
    hist1 = cv2.calcHist([img1_eq], [0], None, [256], [0, 256])
    dataset_1.append(hist1)

c1 = len(dataset_1)

for dog in os.listdir(PROCESSED_IMAGE_PATHS[3]):
    img2_eq = cv2.imread(PROCESSED_IMAGE_PATHS[3] + dog,cv2.IMREAD_GRAYSCALE)
    #img2_eq = cv2.equalizeHist(img1_gray_class_2)
    hist2 = cv2.calcHist([img2_eq], [0], None, [256], [0, 256])

    dataset_1.append(hist2)

dataset_1 = np.array(dataset_1)[:,:,0]

final_dataset_1 = dataset_1
```

[ ]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler


data = StandardScaler().fit_transform(final_dataset_1)
#data = final_dataset_1
print(data.shape)
pca= PCA(n_components=2)
principalComponents_dog = pca.fit_transform(data)

principalComponents_dog.shape
```

```
(357, 256)
```

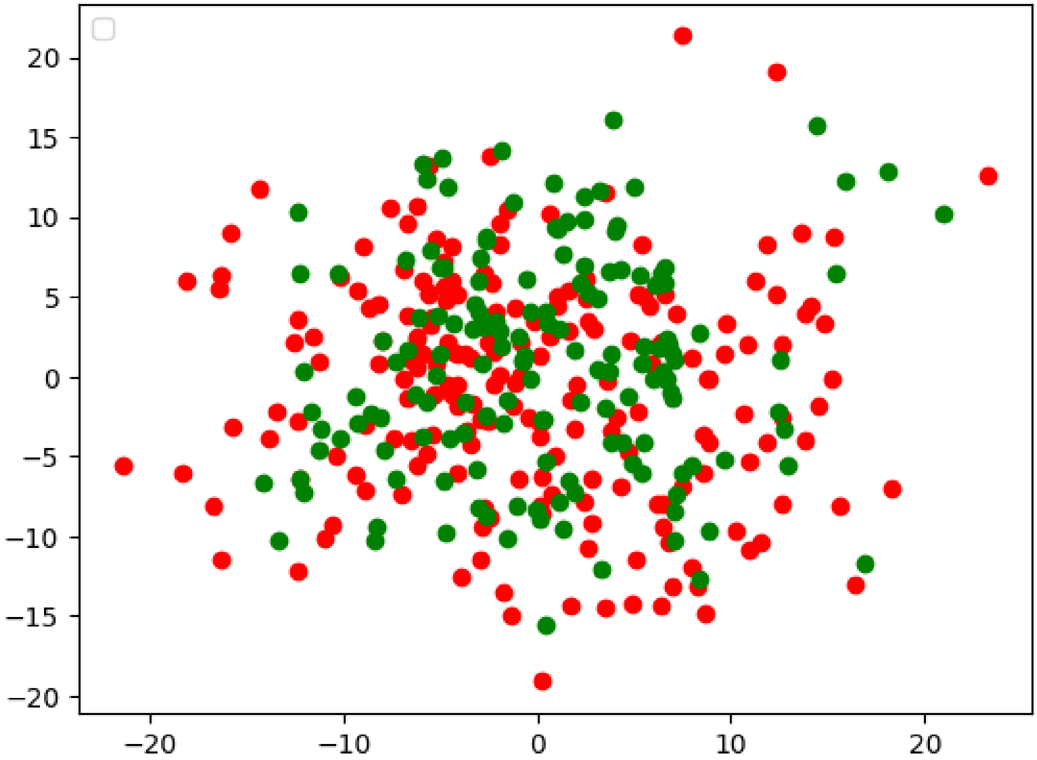
[ ]:

```
(357, 2)
```

[ ]:

```
fig = plt.figure()
ax1 = fig.add_subplot(111)

arr = np.linspace(0, 255, num=256)
plt.scatter(principalComponents_dog[:201,0],principalComponents_dog[:
 ↪201,1],c='r')
plt.scatter(principalComponents_dog[201:,0],principalComponents_dog[201:
 ↪,1],c='g')
plt.legend(loc='upper left')
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```


20
15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20