2.(a) Cropping and Resize Images in Your 4-class Images Dataset:

```
import os
from PIL import Image
import glob
import xml.etree.ElementTree as ET
from pathlib import Path
import warnings
warnings.filterwarnings("ignore")

img_dir = r'D:\Data Mining\Programming Assignment - 1\Data Files\
Images'
ann_dir = r'D:\Data Mining\Programming Assignment - 1\Data Files\
Annotation'

# Function to list items in a directory
def list_directory(dir):
    try:
        items = os.listdir(dir)
        return items
    except FileNotFoundError:
        print(f"Directory '{dir}' not found.")
        return []

dog_images = glob.glob(img_dir+'\\*\\*')
annotations = glob.glob(ann_dir+'\\*\\*')

# Listing of items in both of the directories
isub, asub = list_directory(img_dir), list_directory(ann_dir)
print("Classes in Image Directory:\n",isub)
print("The total Images in four classes are ",len(dog_images))
print("\nClasses in Annotation Directory:\n",asub)
print("The total Annotations in four classes are ",len(annotations))
print()

# From the annotations the range of X and Y coordinates are read
def get_bounding_boxes(ann):
    tree = ET.parse(ann)
    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
```

```
# The function to retrieve image path
def get_image(annot):
    file = annot.split('\\')
    img_filename = img_dir+'\\' + file[-2]+'\\'+file[-1]+'.jpg'
    return img_filename

# Cropping of the Images from all the four classes
for i in range(len(dog_images)):
    bbox = get_bounding_boxes(annotations[i])
    dog = get_image(annotations[i])
    im = Image.open(dog)

    # To process the first bounding box
    if bbox:
        im2 = im.crop(bbox[0])
        im2 = im2.resize((128, 128))
        new_path = dog.replace('D:\\Data Mining\\Programming
Assignment - 1\\Data Files\\Images\\', '.\\Cropped-1\\')
        new_path = new_path.replace('.jpg','-' + str(0) + '.jpg')
        im2 = im2.convert('RGB')
        head, tail = os.path.split(new_path)
        Path(head).mkdir(parents=True, exist_ok=True)
        im2.save(new_path)
        #print(f"Cropped image saved: {new_path}")  # Confirmation
message

    # To process all bounding boxes
    for j in range(len(bbox)):
        im2 = im.crop(bbox[j])
        im2 = im2.resize((128,128))
        new_path = dog.replace('D:\\Data Mining\\Programming
Assignment - 1\\Data Files\\Images\\','.\\Cropped-2\\')
        new_path = new_path.replace('.jpg','-' + str(j) + '.jpg')
        im2=im2.convert('RGB')
        head, tail = os.path.split(new_path)
        Path(head).mkdir(parents=True, exist_ok=True)
        im2.save(new_path)
       # print(f"Cropped image saved: {new_path}")  # Confirmation
message
print("Images are cropped")
```

```
Classes in Image Directory:
 ['n02092002-Scottish_deerhound', 'n02093428-
American_Staffordshire_terrier', 'n02094114-Norfolk_terrier',
'n02110958-pug']
The total Images in four classes are  768

Classes in Annotation Directory:
 ['n02092002-Scottish_deerhound', 'n02093428-
```

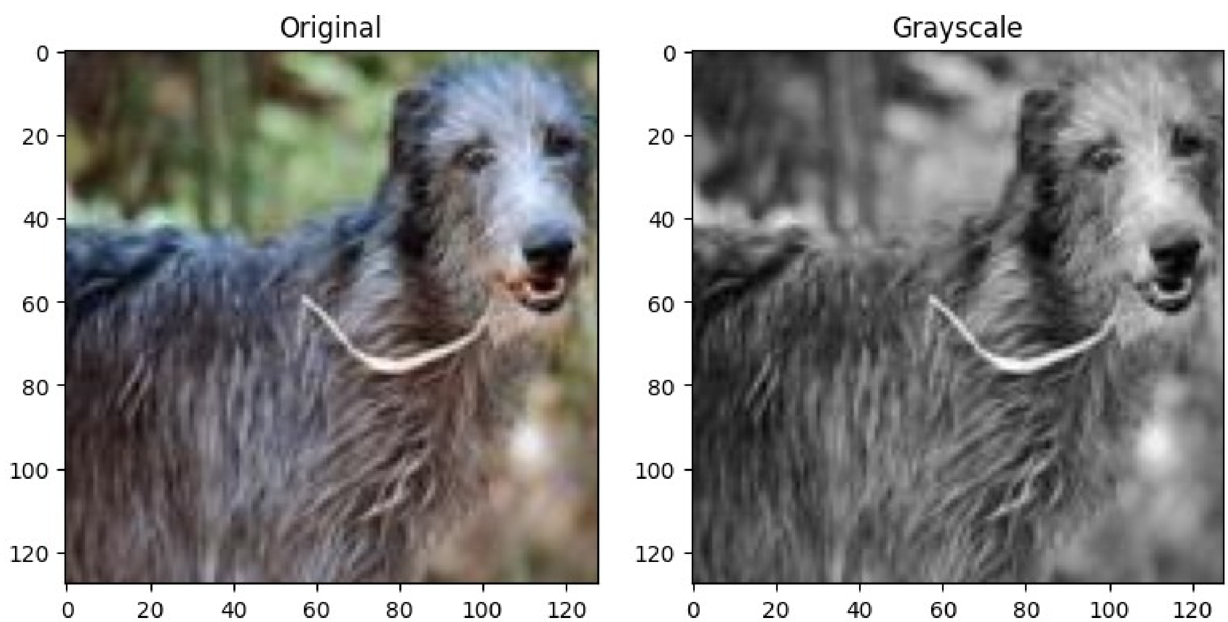
```
American_Staffordshire_terrier', 'n02094114-Norfolk_terrier',
'n02110958-pug']
The total Annotations in four classes are  768

Images are cropped
```

2.(b) Feature Extraction: Edge histogram AND Similarity Measurements

```
import os
import matplotlib.pyplot as plt
%matplotlib inline
import random
from skimage import data,io
from skimage.color import rgb2gray
import numpy as np
from skimage import filters
from skimage import exposure, img_as_float
from sklearn.metrics.pairwise import euclidean_distances,
manhattan_distances, cosine_distances
import warnings
warnings.filterwarnings("ignore")

main_dir = r'D:\Data Mining\Programming Assignment - 1\Codes\Cropped-
1'

# Gathering all class directories
class_dir = [os.path.join(main_dir, class_name)
              for class_name in os.listdir(main_dir)
              if os.path.isdir(os.path.join(main_dir, class_name))]

# Function to gather random images from each and every class
def get_random_img(images_per_class=1):
    image_files = []
    for c in class_dir:
        files = [os.path.join(c, file)
                 for file in os.listdir(c)
                 if file.endswith(('.jpg'))]
        selected_files = random.sample(files, images_per_class)
        image_files.extend(selected_files)
    return image_files

# Get random images from the classes
image_files = get_random_img()
edge_histograms = []

for file in image_files:
    # GrayScale
    original = io.imread(file)
    grayscale = rgb2gray(original)

    fig, axes = plt.subplots(1, 2, figsize=(8, 5))
    ax = axes.ravel()

    file_name = os.path.basename(file)
    class_name = os.path.basename(os.path.dirname(file))
```

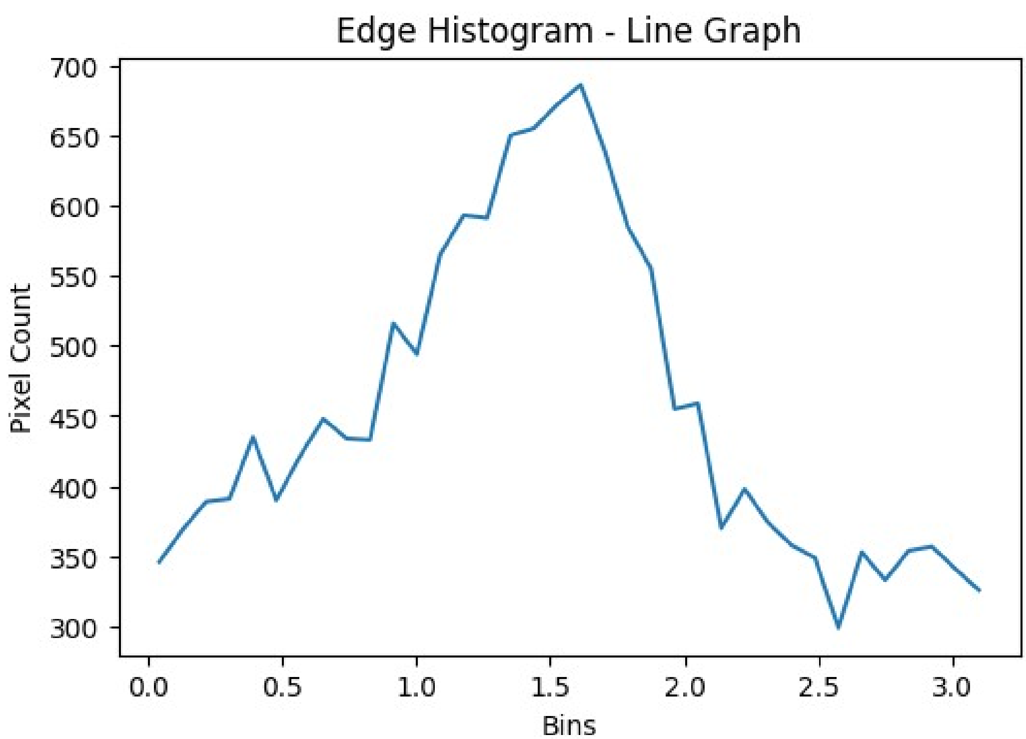
```
    print(f"File Name: {class_name} ({file_name})")
    ax[0].imshow(original)
    ax[0].set_title("Original")
    ax[1].imshow(grayscale, cmap=plt.cm.gray)
    ax[1].set_title("Grayscale")

    fig.tight_layout()
    plt.show()

    # Angle as the direction of edge gradient at the pixel
    def angle(dx, dy):
        """Calculate the angles between horizontal and vertical
operators."""
        return np.mod(np.arctan2(dy, dx), np.pi)
    angle_sobel = angle(filters.sobel_h(grayscale),
filters.sobel_v(grayscale))
    print(" Angle: \n",angle_sobel)

    #Histogram
    hist_data=exposure.histogram(angle_sobel, nbins=36)
    print("\nHistogram Data\n",hist_data)

    edge_histograms.append(hist_data)

    hist_counts, bin_edges = hist_data

    # Plotting the Edge Histogram

    # Line Graph
    plt.figure(figsize=(6, 4))
    plt.plot(bin_edges, hist_counts)
    plt.xlabel("Bins")
    plt.ylabel("Pixel Count")
    plt.title("Edge Histogram - Line Graph")
    plt.show()

    # Bar Graph
    plt.figure(figsize=(10, 5))
    plt.bar(bin_edges, hist_counts, width=0.05, align='center')
    plt.xlabel("Bins")
    plt.ylabel("Pixel Count")
    plt.title("Edge Histogram - Bar Graph")
    plt.show()

    # Pie Chart
    plt.figure(figsize=(9, 9))
    plt.pie(hist_counts, labels=[f'Bin {i+1}' for i in
range(len(hist_counts))], autopct='%1.1f%%', startangle=90)
    plt.title("Edge Histogram - Pie Chart")
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn
```

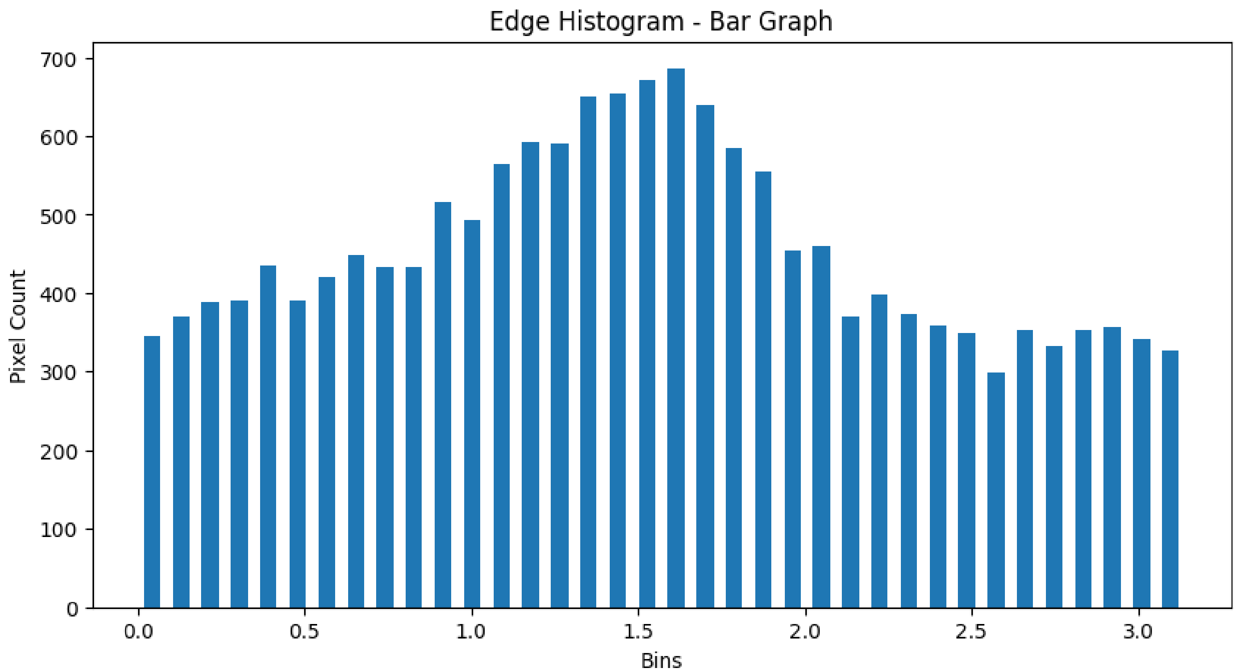
```
as a circle.
    plt.show()

# Calculation of Distances bestween Edge Histogram datasets
select_edge_hist = random.sample(edge_histograms,2)
hist_1 = np.array(select_edge_hist[0]).reshape(1, -1)
hist_2 = np.array(select_edge_hist[1]).reshape(1, -1)
eud_dist = euclidean_distances(hist_1, hist_2)[0][0]
man_dist = manhattan_distances(hist_1, hist_2)[0][0]
cos_dist = cosine_distances(hist_1, hist_2)[0][0]

print('The Distances for the edge histogram are between 1 and 3
datasets')
print('Euclidean Distance: ',eud_dist)
print('Manhattan Distance: ',man_dist)
print('Cosine Distance: ',cos_dist)

File Name: n02092002-Scottish_deerhound (n02092002_7791-0.jpg)
```



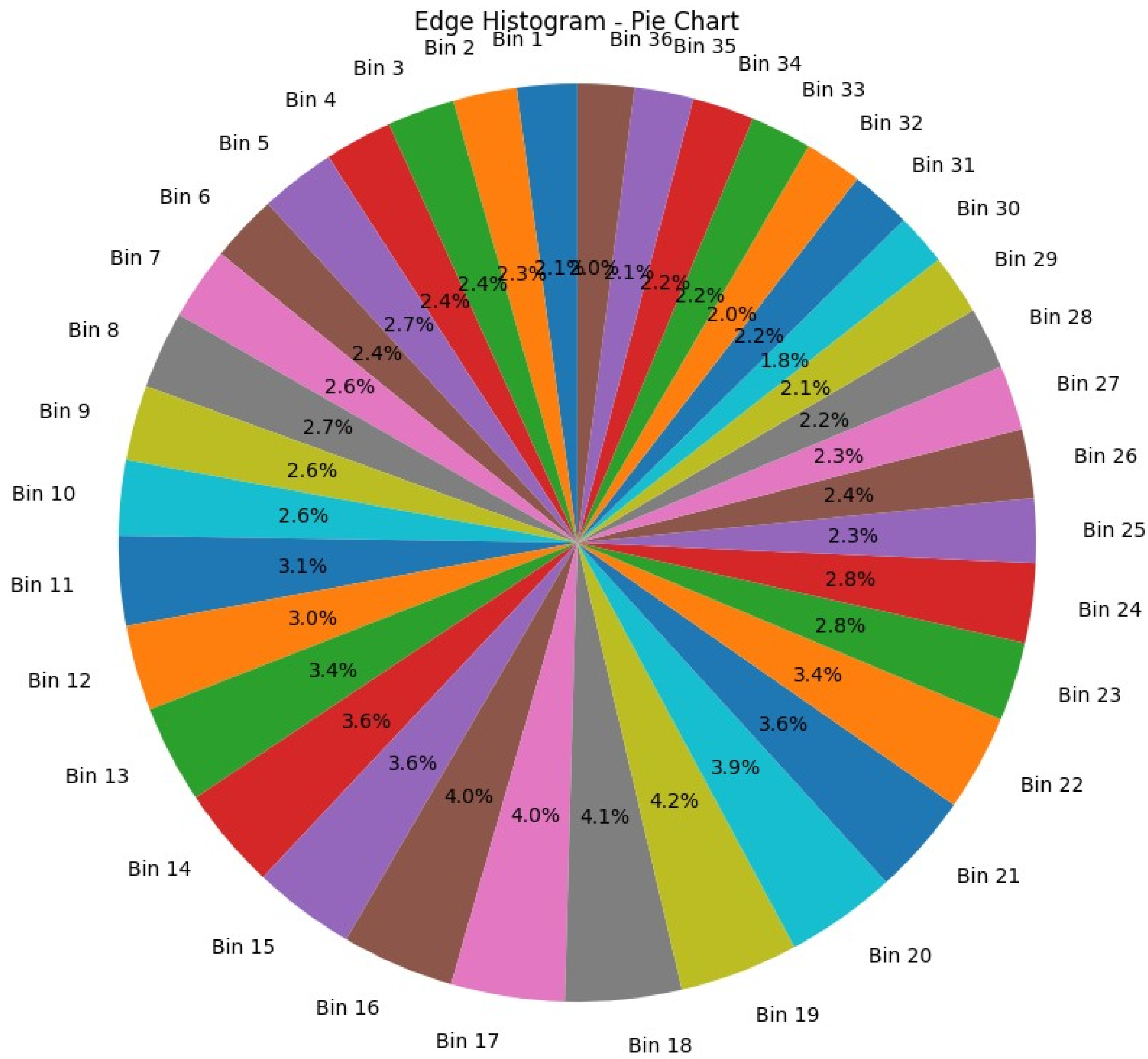

```
 Angle:
 [[2.91479381 0.9817665  1.40657798 ... 0.81029943 0.99935555
0.62463921]
 [0.09065989 1.00955245 1.44916894 ... 0.53872997 0.52632182
0.24348783]
 [0.78539816 1.07311647 1.46387339 ... 0.60547226 0.44252027
0.17016359]
 ...
 [2.11121583 1.88328789 0.78539816 ... 2.36336208 3.10108732
0.21115697]
 [0.11379201 2.39726727 1.99928255 ... 2.25770579 0.05127328
```

```
0.29611749]
 [0.42019671 2.11882024 1.77981927 ... 1.8626421  0.37848072
0.78539816]]

Histogram Data
 (array([346, 369, 389, 391, 435, 390, 421, 448, 434, 433, 516, 494,
565,
       593, 591, 650, 655, 672, 686, 640, 585, 555, 455, 459, 370,
398,
       374, 358, 349, 299, 353, 333, 354, 357, 341, 326],
dtype=int64), array([0.04363323, 0.13089969, 0.21816616, 0.30543262,
0.39269908,
       0.47996554, 0.56723201, 0.65449847, 0.74176493, 0.82903139,
       0.91629786, 1.00356432, 1.09083078, 1.17809725, 1.26536371,
       1.35263017, 1.43989663, 1.5271631 , 1.61442956, 1.70169602,
       1.78896248, 1.87622895, 1.96349541, 2.05076187, 2.13802833,
       2.2252948 , 2.31256126, 2.39982772, 2.48709418, 2.57436065,
       2.66162711, 2.74889357, 2.83616003, 2.9234265 , 3.01069296,
       3.09795942]))
```

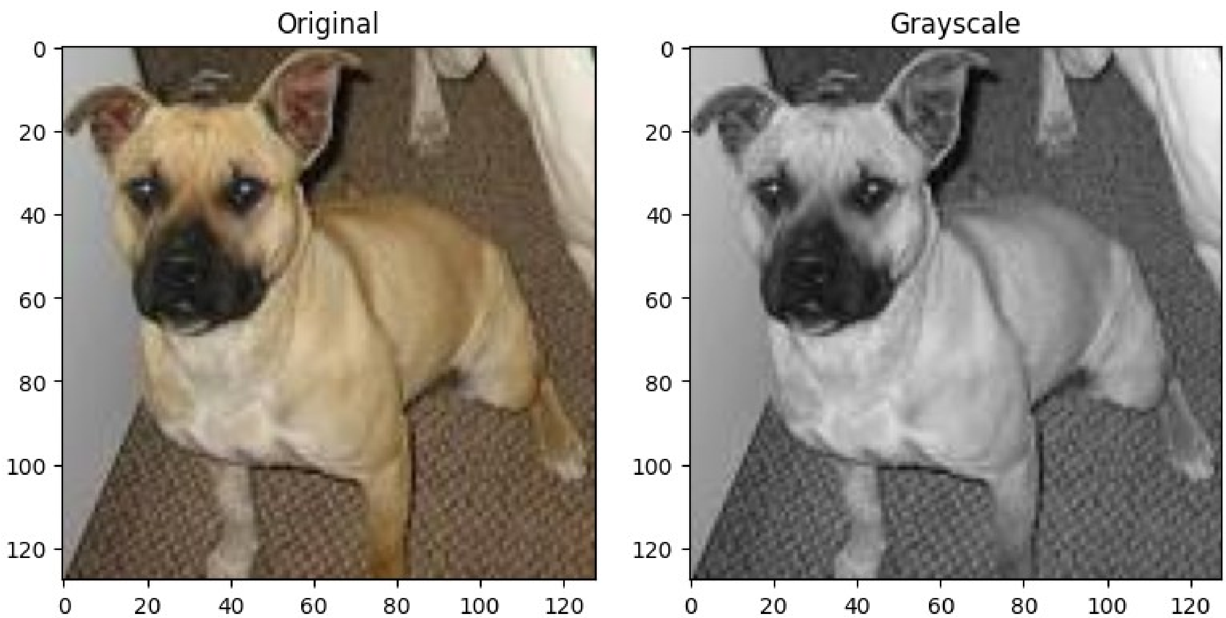




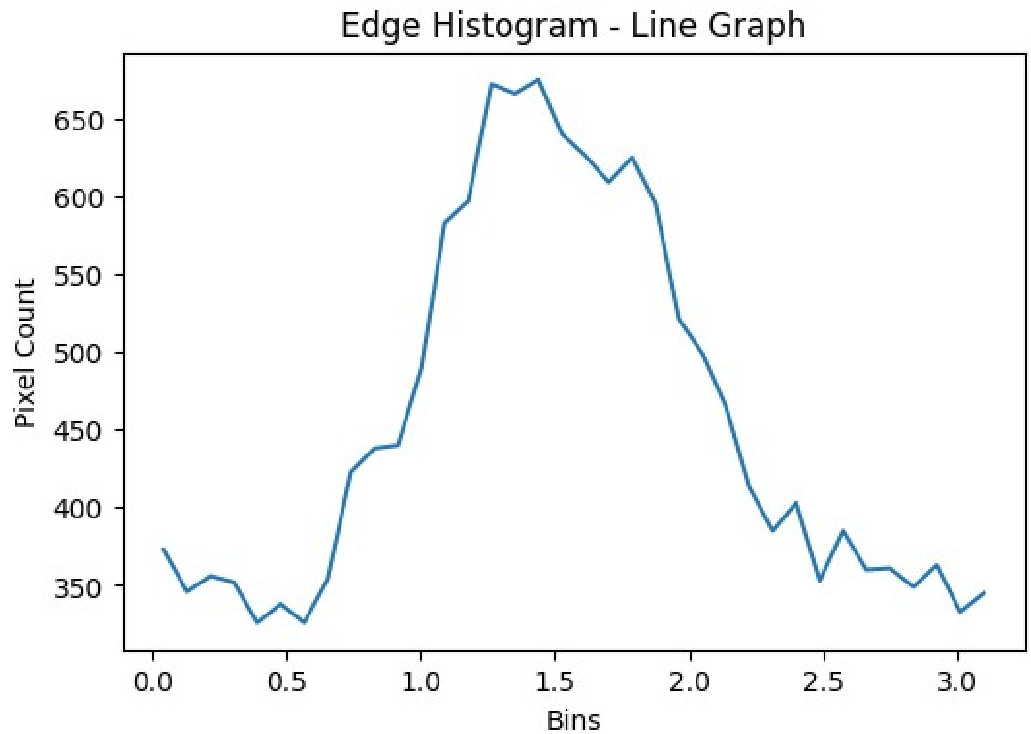
Edge Histogram - Bar Graph
Pixel Count
Bins
0
100
200
300
400
500
600
700
0.0
0.5
1.0
1.5
2.0
2.5
3.0

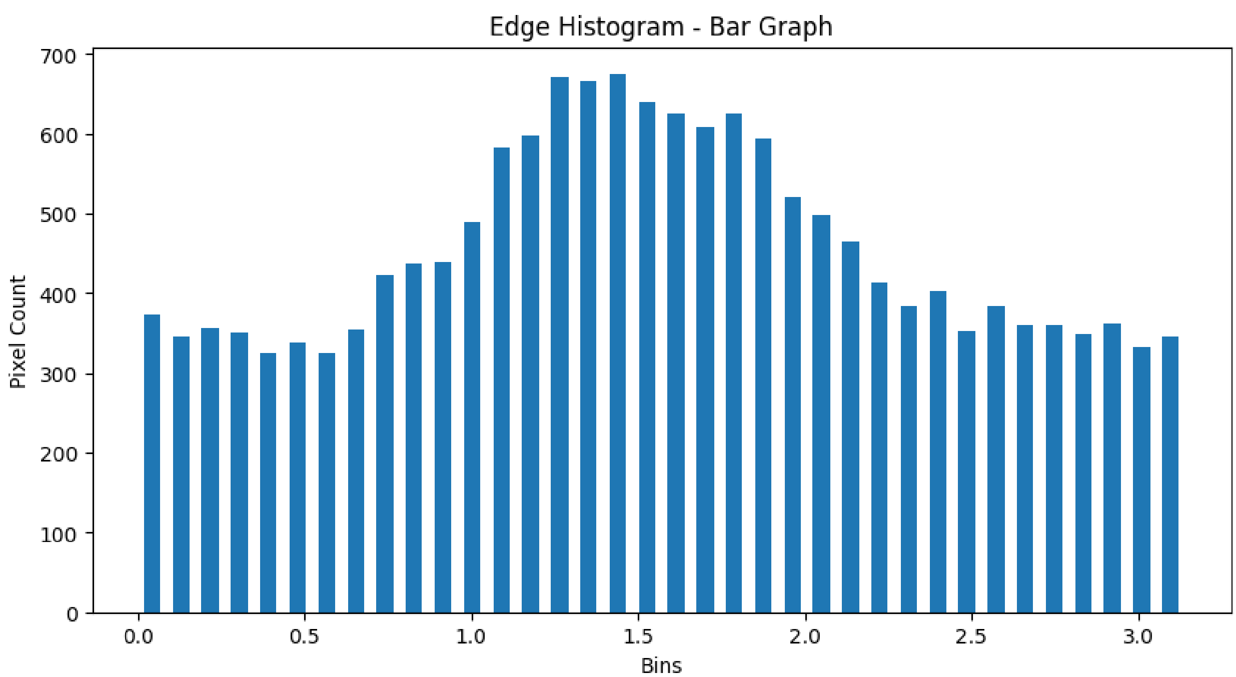


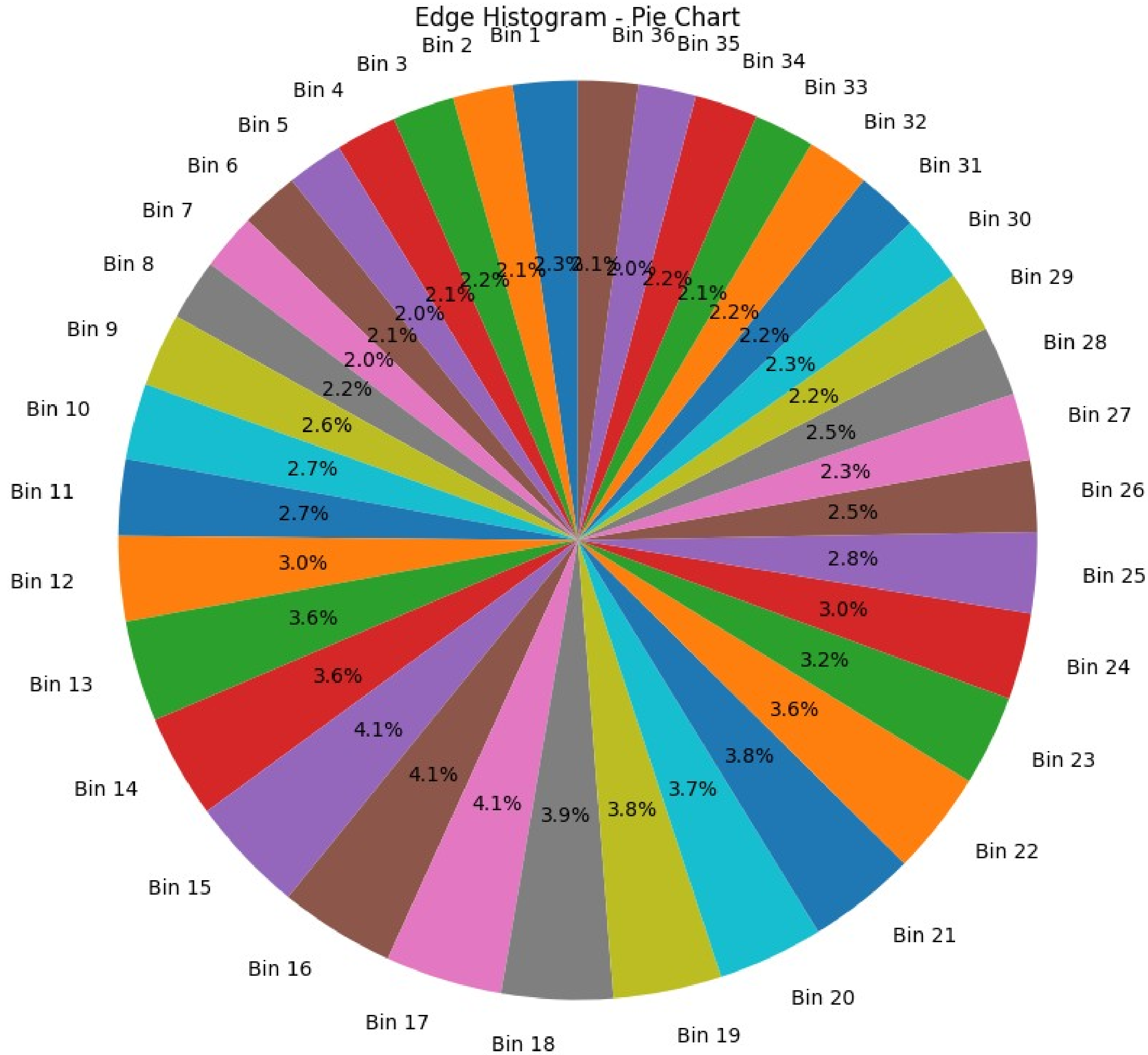
File Name: n02093428-American_Staffordshire_terrier (n02093428_3353-0.jpg)



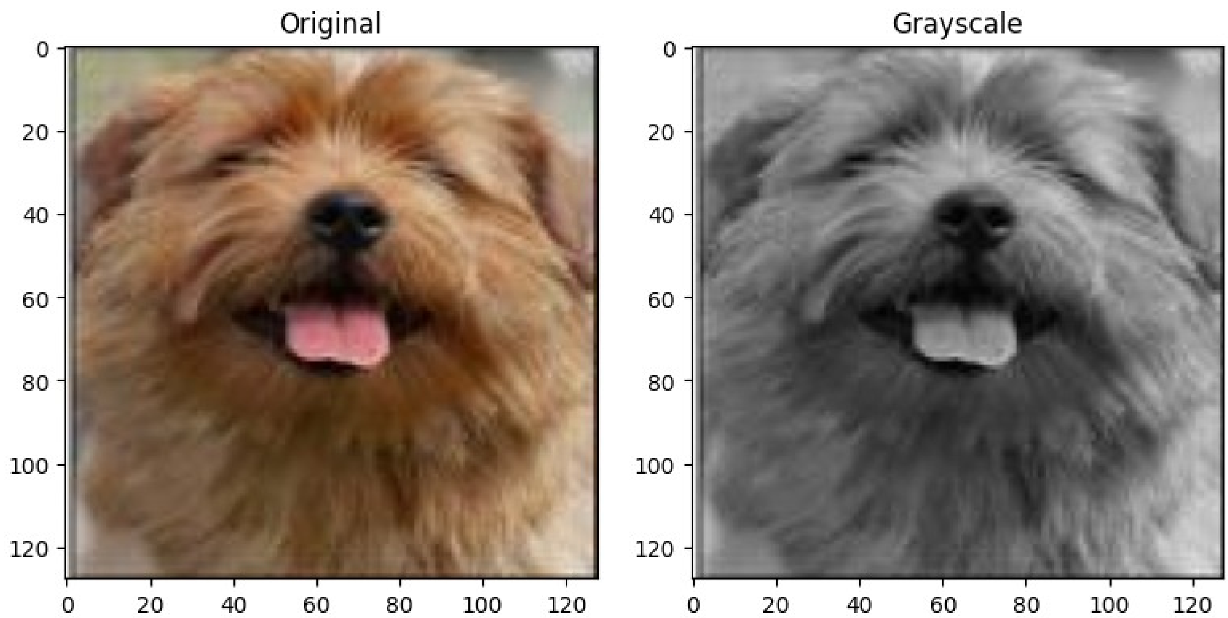

```
 Angle:
 [[0.         1.57079633 1.57079633 ... 1.86511466 1.66075981
1.68491157]
 [0.         1.42562482 1.49782612 ... 2.16901499 1.75545333
1.87498606]
 [0.         1.42562482 1.49782612 ... 2.26421682 1.80323176
2.10404769]
 ...
 [1.09782076 1.29615523 1.52023103 ... 1.00194847 2.41765545
2.35619449]
 [1.13819778 1.023918   0.48989386 ... 1.96648592 0.94405343
1.26274355]
 [1.41891387 1.12527283 1.78502484 ... 2.64978348 0.35144479
0.48689923]]

Histogram Data
 (array([373, 346, 356, 352, 326, 338, 326, 354, 423, 438, 440, 489,
583,
       597, 672, 666, 675, 640, 626, 609, 625, 595, 521, 499, 465,
413,
       385, 403, 353, 385, 360, 361, 349, 363, 333, 345],
dtype=int64), array([0.04363323, 0.13089969, 0.21816616, 0.30543262,
0.39269908,
       0.47996554, 0.56723201, 0.65449847, 0.74176493, 0.82903139,
       0.91629786, 1.00356432, 1.09083078, 1.17809725, 1.26536371,
       1.35263017, 1.43989663, 1.5271631 , 1.61442956, 1.70169602,
       1.78896248, 1.87622895, 1.96349541, 2.05076187, 2.13802833,
       2.2252948 , 2.31256126, 2.39982772, 2.48709418, 2.57436065,
       2.66162711, 2.74889357, 2.83616003, 2.9234265 , 3.01069296,
       3.09795942]))
```


Edge Histogram - Line Graph
Pixel Count
Bins
650
600
550
500
450
400
350
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Edge Histogram - Bar Graph
Pixel Count
Bins
700
600
500
400
300
200
100
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

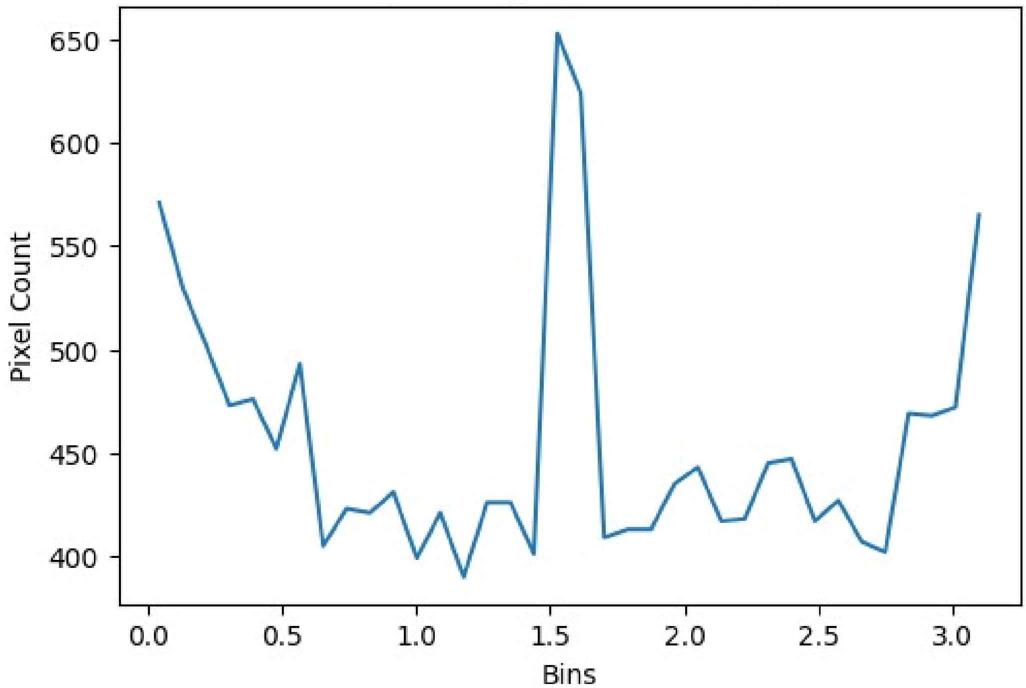



```
File Name: n02094114-Norfolk_terrier (n02094114_2573-0.jpg)
```

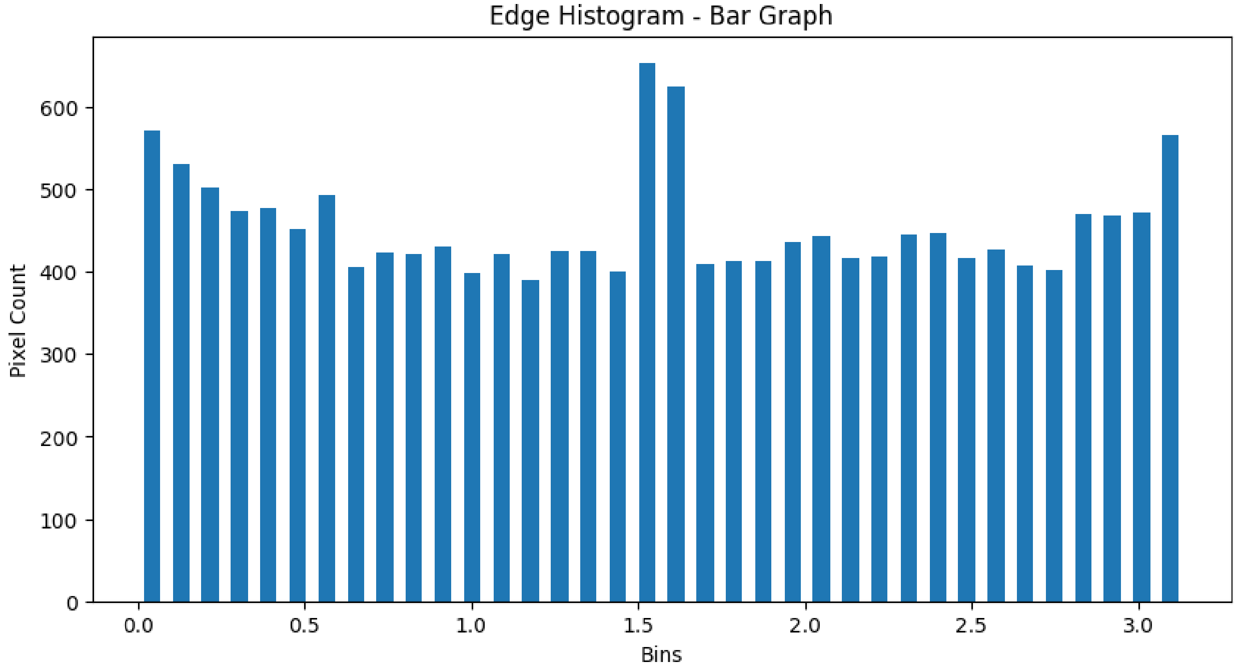


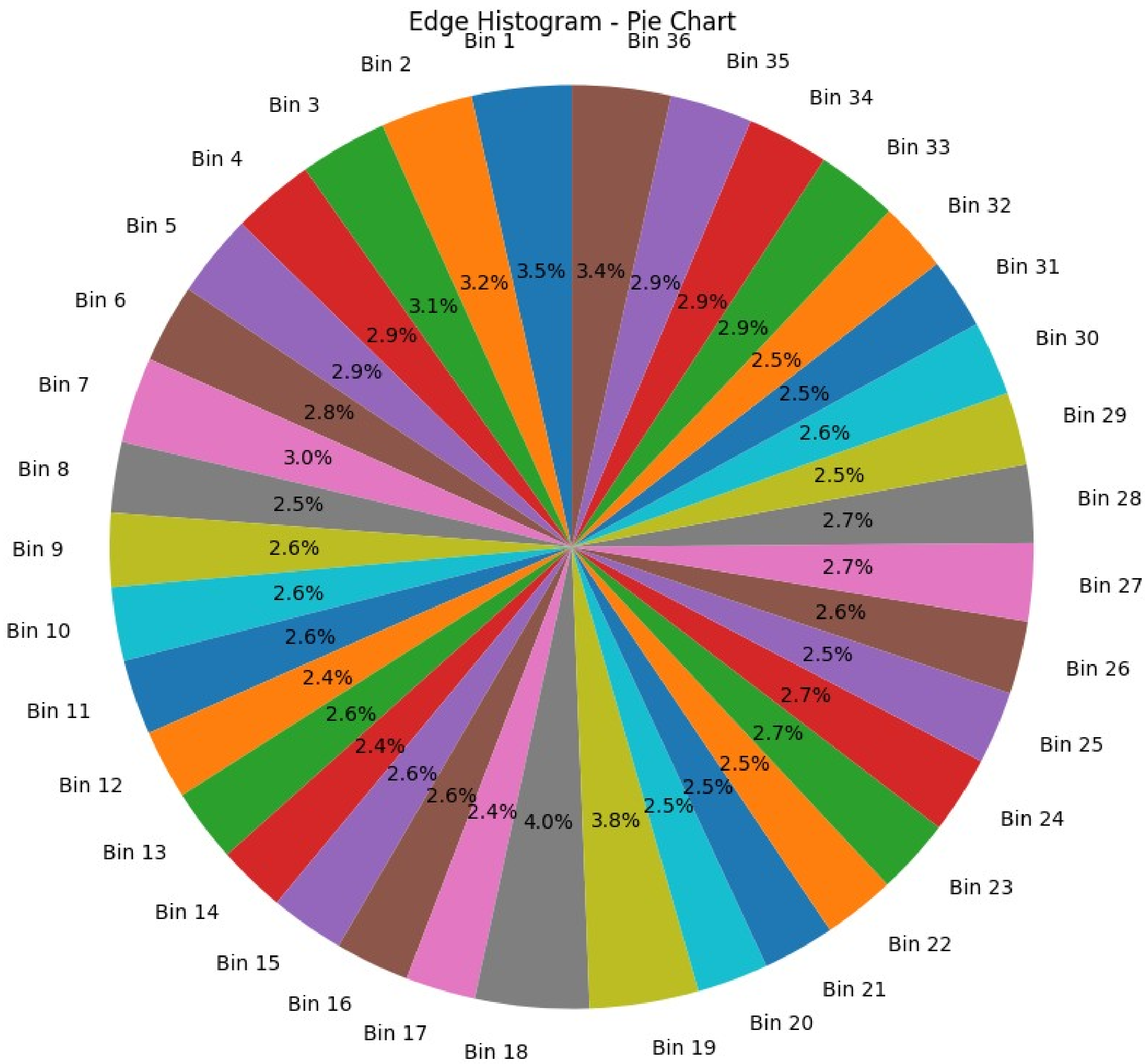

```
 Angle:
 [[1.50164038 1.51575603 1.76709633 ... 0.14388428 2.46102972
1.73217144]
 [1.51247795 1.4947629  3.13898044 ... 3.11080341 2.15284323
1.59040166]
 [1.57954322 1.56038725 1.27217856 ... 2.94786115 1.73094778
1.59136956]
 ...
 [1.57277475 1.57800648 1.50449006 ... 2.04150272 1.62497633
1.64262386]
 [1.57684052 1.52459368 2.2163707  ... 0.63775332 1.39390385
1.43691833]
 [1.55182963 1.50182829 2.62630875 ... 0.28666032 1.2129616
1.23736114]]

Histogram Data
 (array([571, 530, 502, 473, 476, 452, 493, 405, 423, 421, 431, 399,
421,
       390, 426, 426, 401, 653, 624, 409, 413, 413, 435, 443, 417,
418,
       445, 447, 417, 427, 407, 402, 469, 468, 472, 565],
dtype=int64), array([0.04363323, 0.13089969, 0.21816616, 0.30543262,
0.39269908,
       0.47996554, 0.56723201, 0.65449847, 0.74176493, 0.82903139,
       0.91629786, 1.00356432, 1.09083078, 1.17809725, 1.26536371,
       1.35263017, 1.43989663, 1.5271631 , 1.61442956, 1.70169602,
       1.78896248, 1.87622895, 1.96349541, 2.05076187, 2.13802833,
       2.2252948 , 2.31256126, 2.39982772, 2.48709418, 2.57436065,
       2.66162711, 2.74889357, 2.83616003, 2.9234265 , 3.01069296,
       3.09795942]))
```

Edge Histogram - Line Graph

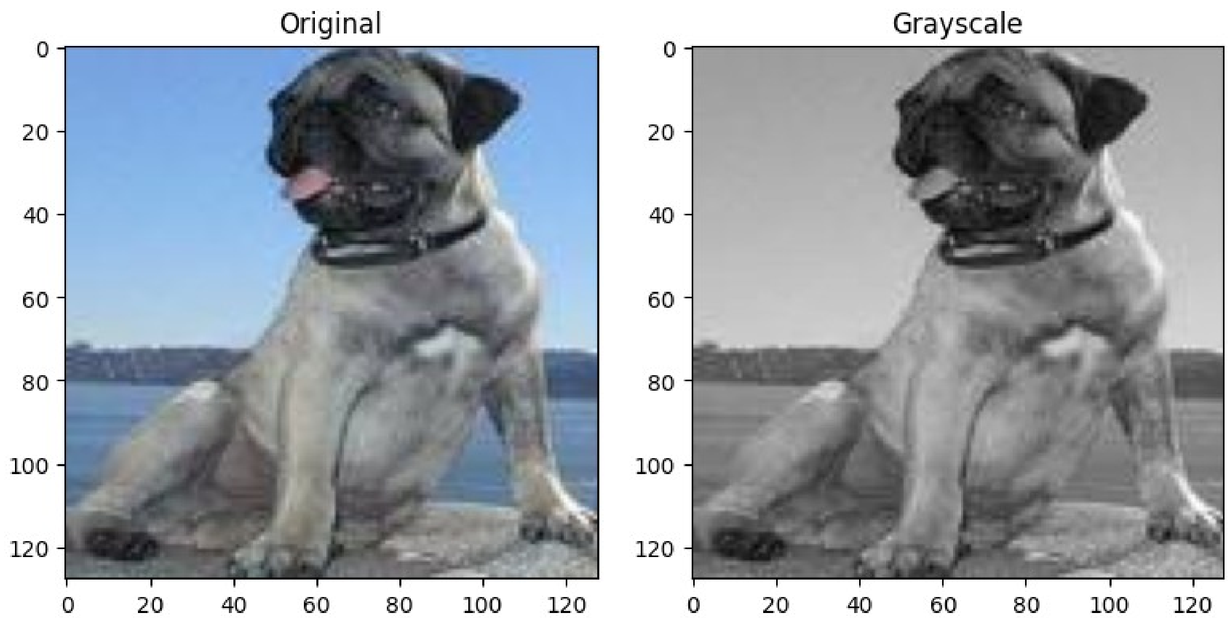


Edge Histogram - Bar Graph



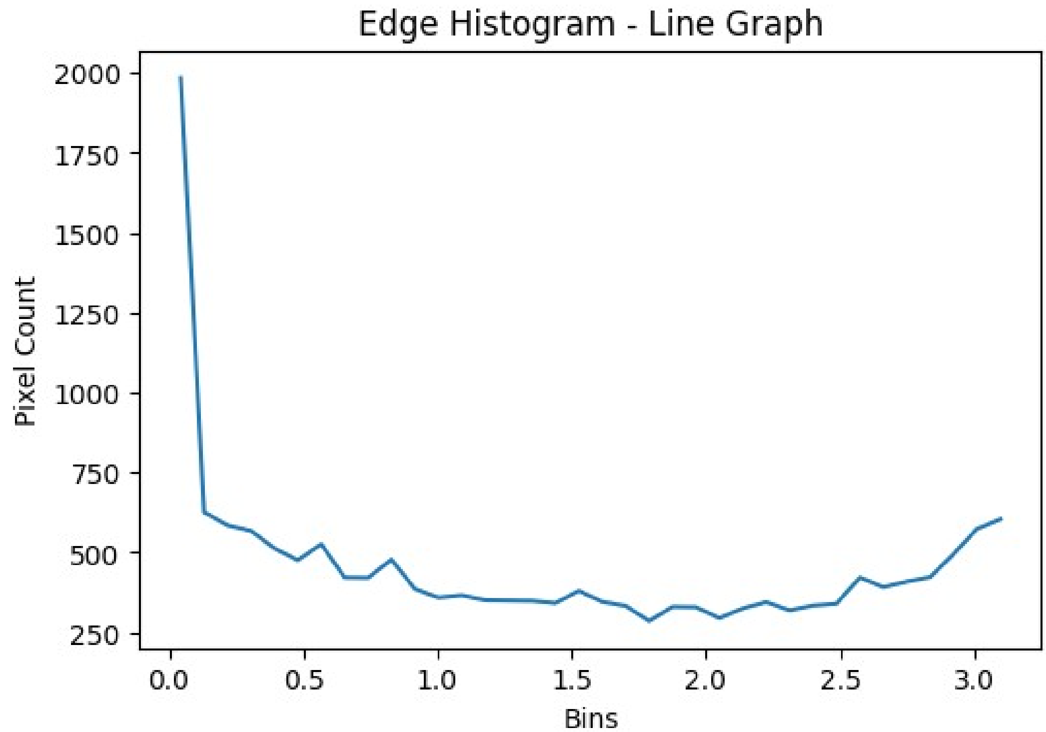


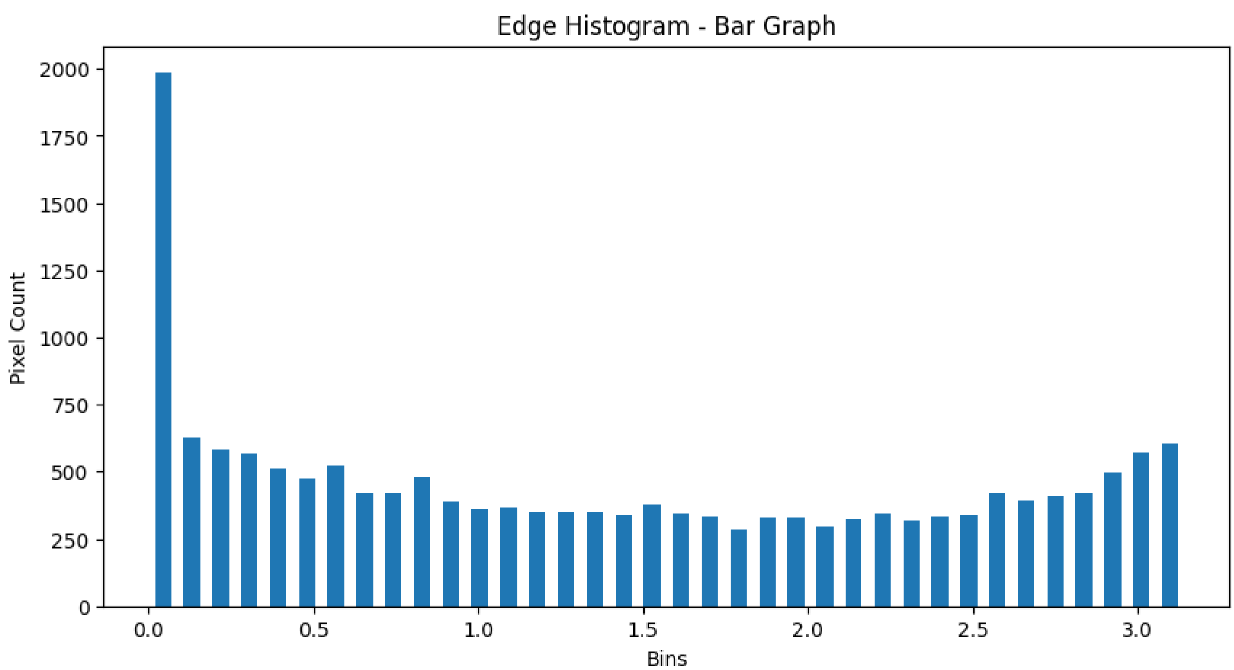
File Name: n02110958-pug (n02110958_11239-0.jpg)

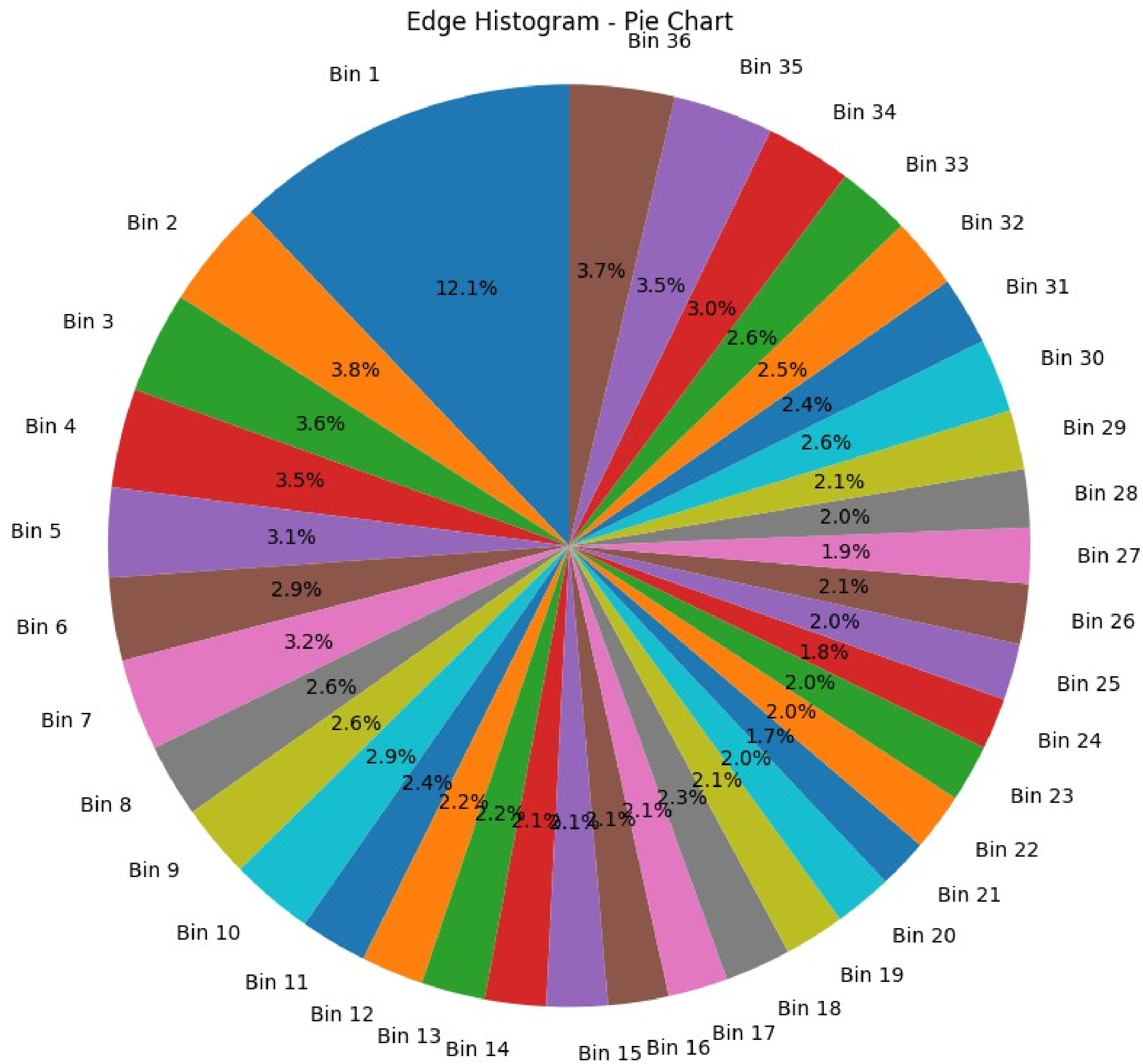



```
 Angle:
 [[0.         0.         0.         ... 0.         0.         0.
]
 [0.         0.         0.         ... 0.         0.         0.
]
 [0.         0.         0.         ... 0.         0.         0.
]
 ...
 [3.03000302 0.0525618  0.28039797 ... 0.37798159 2.69024277
0.01340914]
 [2.61646558 2.44489569 0.76738077 ... 1.70057188 2.92614714
0.39922921]
 [1.9204785  1.77268519 1.22906642 ... 2.8504611  1.70820959
1.19075338]]

Histogram Data
 (array([1985,  626,  585,  568,  513,  476,  526,  422,  421,  478,
386,
        359,  366,  351,  350,  349,  342,  380,  346,  333,  286,
330,
        329,  295,  325,  345,  319,  334,  340,  422,  392,  409,
423,
        495,  574,  604], dtype=int64), array([0.04363323, 0.13089969,
0.21816616, 0.30543262, 0.39269908,
       0.47996554, 0.56723201, 0.65449847, 0.74176493, 0.82903139,
       0.91629786, 1.00356432, 1.09083078, 1.17809725, 1.26536371,
       1.35263017, 1.43989663, 1.5271631 , 1.61442956, 1.70169602,
       1.78896248, 1.87622895, 1.96349541, 2.05076187, 2.13802833,
       2.2252948 , 2.31256126, 2.39982772, 2.48709418, 2.57436065,
```

```
       2.66162711, 2.74889357, 2.83616003, 2.9234265 , 3.01069296,
       3.09795942]))
```





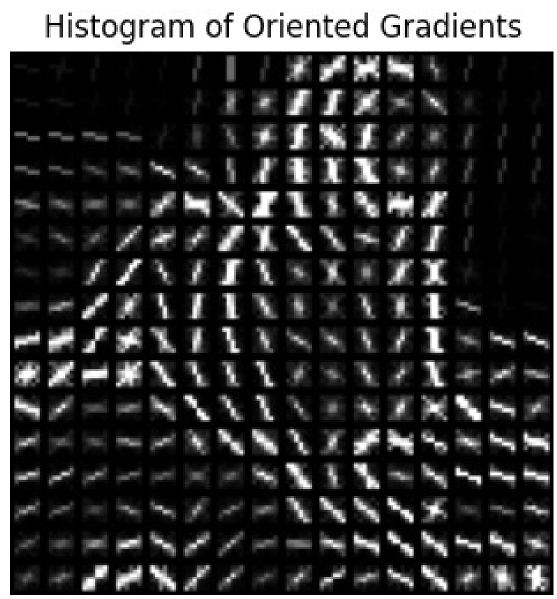



```
The Distances for the edge histogram are between 1 and 3 datasets
Euclidean Distance:  827.8236527183772
Manhattan Distance:  4062.0
Cosine Distance:  0.0437776245474909
```

2.(c) Histogram of Oriented Gradient (HOG) feature descriptor

```
import matplotlib.pyplot as plt
%matplotlib inline
from skimage.feature import hog
from skimage import data, exposure
warnings.filterwarnings("ignore")

main_dir = r'D:\Data Mining\Programming Assignment - 1\Codes\Cropped-
```

```
1'
class_dir = [os.path.join(main_dir, class_name)
              for class_name in os.listdir(main_dir)
              if os.path.isdir(os.path.join(main_dir, class_name))]

# Function for gathering only one random image from all classes
def get_random_img():
    all_images = []
    for c in class_dir:
        files = [os.path.join(c, file)
                 for file in os.listdir(c)
                 if file.endswith(('.jpg'))]
        all_images.extend(files)
    if all_images:
        return random.choice(all_images)
    else:
        return None

image = io.imread(get_random_img())

fd, hog_image = hog(
    image,
    orientations=10,
    pixels_per_cell=(8, 8),
    cells_per_block=(2, 2),
    visualize=True,
    channel_axis=-1,
)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9, 4), sharex=True,
sharey=True)

ax1.axis('off')
ax1.imshow(image, cmap=plt.cm.gray)
ax1.set_title('Input image')

# Rescaling histogram for better display
hog_image_rescaled = exposure.rescale_intensity(hog_image,
in_range=(0, 10))

ax2.axis('off')
ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
print("HOG Descriptors: ", fd)
```




```
HOG Descriptors:  [0.         0.         0.         ... 0.12831129
0.27310934 0.09519539]
```

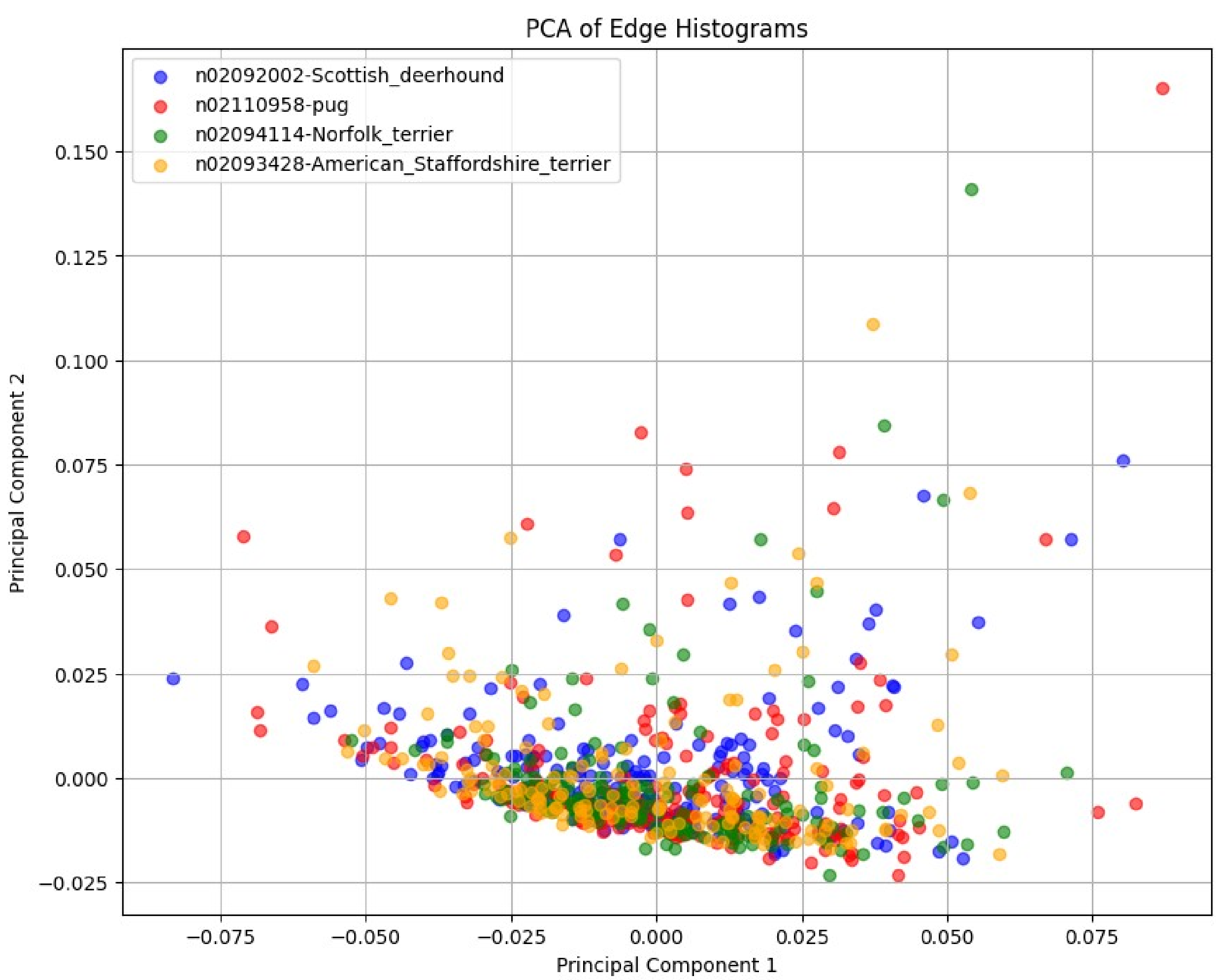
2.(d) Dimensionality reduction (using Principal Component Analysis, PCA)

```
import os
import numpy as np
from PIL import Image
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
%matplotlib inline
import json
warnings.filterwarnings("ignore")

main_dir = r'D:\Data Mining\Programming Assignment - 1\Codes\Cropped-
1'
class_dir = [os.path.join(main_dir, class_name)
              for class_name in os.listdir(main_dir)
              if os.path.isdir(os.path.join(main_dir, class_name))]
print("The four classes in my dataset are below\n")
def get_images():
    all_images = []
    for c in class_dir:
        files = [os.path.join(c, file)
                 for file in os.listdir(c)
                 if file.endswith(('.jpg'))]
        print(f"The {os.path.basename(c)} has {len(files)} images.")
        all_images.extend(files)
    return all_images
```

```python
image_files = get_images()
print(f"\nThe total number of images are {len(image_files)}.")
i=0
edge_histograms = []
reduced_edge_histograms = []

for file in image_files:
    # GrayScale
    original = io.imread(file)
    grayscale = rgb2gray(original)
    # Angle as the direction of edge gradient at the pixel
    def angle(dx, dy):
        return np.mod(np.arctan2(dy, dx), np.pi)
    angle_sobel = angle(filters.sobel_h(grayscale),
filters.sobel_v(grayscale))
    # Histogram
    hist_data=exposure.histogram(angle_sobel, nbins=36)
    edge_histograms.append(hist_data)
    hist_counts, bin_edges = hist_data
    # Normalized Histogram
    hist_counts_norm = hist_counts / hist_counts.sum()
    #print(f"\nHistogram Data {i+1}\n {hist_counts_norm}")
    i+=1
    reduced_edge_histograms.append(hist_counts_norm)

# Perform PCA dimensionality reduction
pca = PCA(n_components=2)
reduced_Histogram = pca.fit_transform(reduced_edge_histograms)
print(f"Histogram Data\n{reduced_Histogram[:5]}......")

# Plot the 2D points using four different colors for data from the
four classes
class_labels = [os.path.basename(os.path.dirname(file)) for file in
image_files]

# Create a color map for the classes
unique_classes = list(set(class_labels))
colors = ['blue', 'red', 'green', 'orange']

# Create a scatter plot
plt.figure(figsize=(10, 8))
for idx, class_name in enumerate(unique_classes):
    class_points = reduced_Histogram[np.array(class_labels) ==
class_name]
    plt.scatter(class_points[:, 0], class_points[:, 1],
                label=class_name, color=colors[idx], alpha=0.6)

plt.title('PCA of Edge Histograms')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

```
plt.legend()
plt.grid()
plt.show()
```

```
The four classes in my dataset are below

The n02092002-Scottish_deerhound has 232 images.
The n02093428-American_Staffordshire_terrier has 164 images.
The n02094114-Norfolk_terrier has 172 images.
The n02110958-pug has 200 images.

The total number of images are 768.
Histogram Data
[[-0.03790986  0.00332977]
 [-0.01191378 -0.0098194 ]
 [ 0.00432497 -0.00968445]
 [ 0.02126993 -0.00011832]
 [-0.00439424  0.00918957]]......
```



In the presented scatter plot it can be observed that there exist four classes which are distinguished by four colors. Blue (Pug): There is some seperation but still overlaps with other Red (American Staffordshire Terrier): Overlaps with other classes also Green (Norfolk Terrier): Also overlaps Orange (Scottish Deerhound): In this case, there can be seen some divisions but they still intersect. Based on this analysis there are no classes that are completely non-overlapping. However the blue class may contain some sites that appear visually different from others but all cannot be said to be very different from other classes. Therefore existing classes are not visually separable at all in this plot.

```
import pandas as pd
import json
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
import warnings
warnings.filterwarnings("ignore")

path= r'D:\Data Mining\Programming Assignment - 1\Data Files\
student_5\train.json'
# Loading the training dataset from the JSON file
data = []
with open(path, 'r') as f:
    for line in f:
        data.append(json.loads(line))

df = pd.DataFrame(data)
print(df.head())
tweets = df['Tweet'].astype(str)

vectorizer_count = CountVectorizer()
t_count = vectorizer_count.fit_transform(tweets)
print("Dimensionality of token count matrix:", t_count.shape)

vectorizer_tfidf = TfidfVectorizer()
x_tfidf = vectorizer_tfidf.fit_transform(tweets)
print("Dimensionality of TF-IDF feature count matrix:", x_tfidf.shape)
```

```
              ID                                              Tweet
anger  \
0  2017-En-30344  Live simply. Dream big. Be grateful. Give love...
False
1  2017-En-31519  Come to the @BullSkitComedy FUNdraiser this Fr...
False
2  2017-En-41145                       @bldmovs sadly beautiful photo.
False
3  2017-En-30651  @eachus At least he's willing to discuss, bett...
False
4  2017-En-30133  @PriiiincesssE thanks for distracting me from ...
False

   anticipation  disgust   fear    joy   love  optimism  pessimism
sadness  \
0         False    False  False   True  False      True      False
False
1         False    False  False  False  False     False      False
False
2         False    False  False   True   True      True      False
True
3         False    False  False  False  False      True       True
False
```

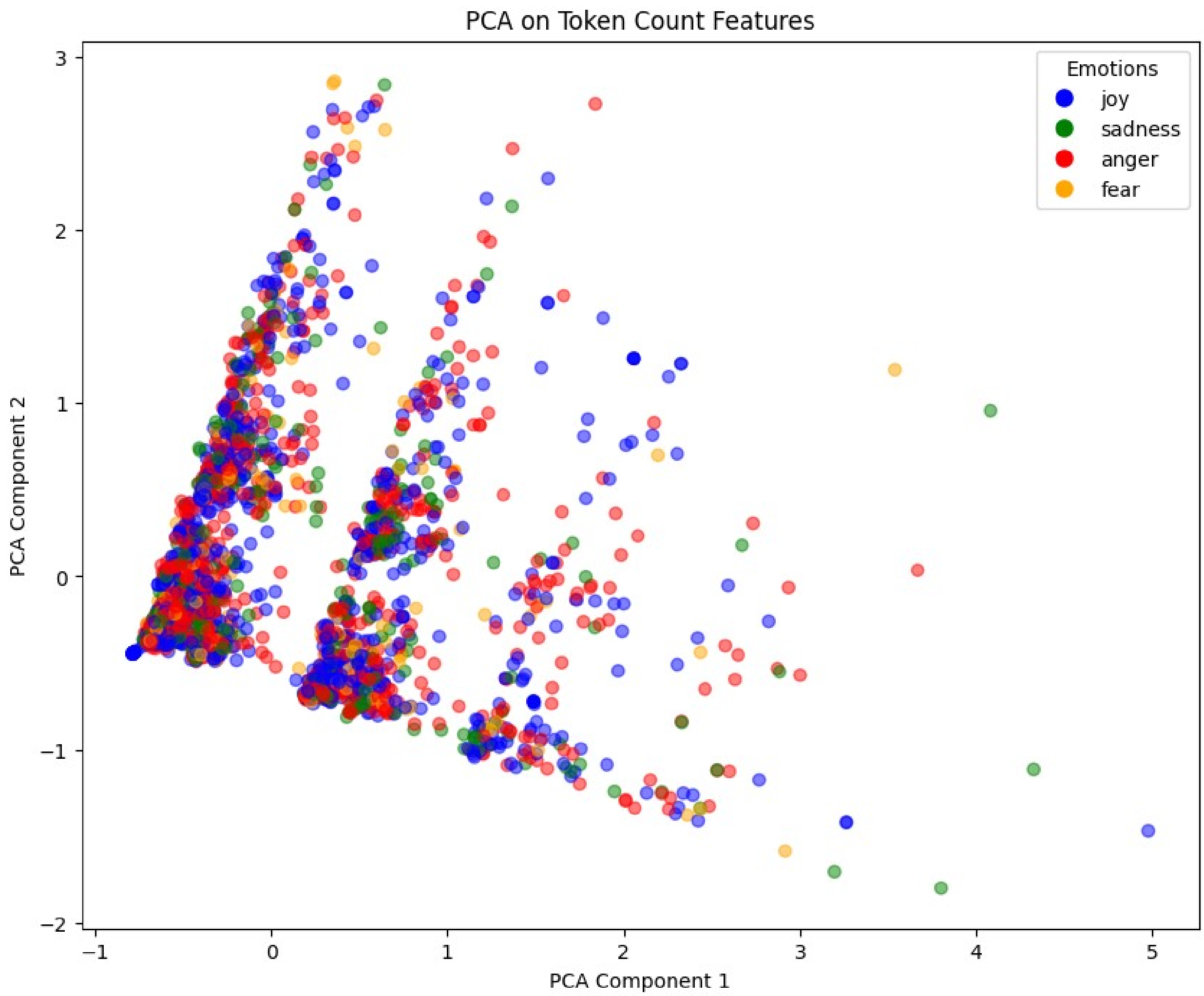
```
4          False    False  False   True   True      True      False
False

   surprise  trust
0     False   True
1     False  False
2     False  False
3     False  False
4     False  False
Dimensionality of token count matrix: (3000, 9524)
Dimensionality of TF-IDF feature count matrix: (3000, 9524)
```

```
from sklearn.decomposition import PCA

selected_classes = ['joy', 'anger', 'sadness', 'fear']
df_filtered = df[(df['joy'] == True) | (df['anger'] == True) |
(df['sadness'] == True) | (df['fear'] == True)]

tweets_filtered = df_filtered['Tweet']

vectorizer_count = CountVectorizer()
t_count = vectorizer_count.fit_transform(tweets_filtered)
pca_count = PCA(n_components=2)
t_count_reduced = pca_count.fit_transform(t_count.toarray())

vectorizer_tfidf = TfidfVectorizer()
X_tfidf = vectorizer_tfidf.fit_transform(tweets_filtered)
pca_tfidf = PCA(n_components=2)
X_tfidf_reduced = pca_tfidf.fit_transform(X_tfidf.toarray())

print("Dimensionality of reduced token count matrix:",
t_count_reduced.shape)
print("Dimensionality of reduced TF-IDF feature count matrix:",
X_tfidf_reduced.shape)
```

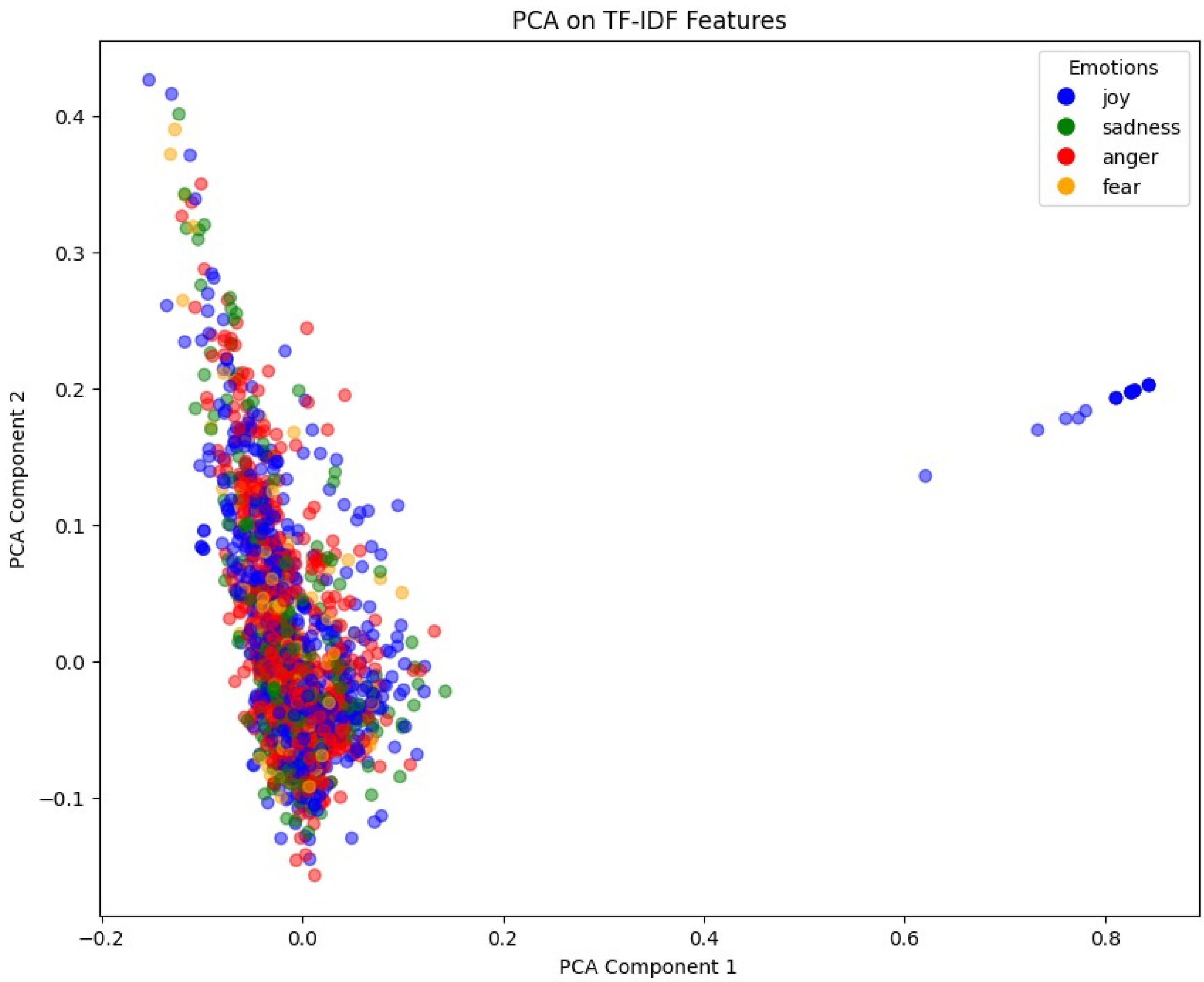
```
Dimensionality of reduced token count matrix: (2701, 2)
Dimensionality of reduced TF-IDF feature count matrix: (2701, 2)
```

```
import matplotlib.pyplot as plt
%matplotlib inline

class_colors = {
    'joy': 'blue',
    'anger': 'red',
    'sadness': 'green',
    'fear': 'orange'
}

df_filtered['label'] = df_filtered.apply(lambda row: 'joy' if
row['joy'] else
                                                     'anger' if
```

```
row['anger'] else
                                                     'sadness' if
row['sadness'] else
                                                     'fear', axis=1)
colors = df_filtered['label'].map(class_colors)

# Plotting the 2d plot
def plot_2d(data, title, colors, labels):
    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(data[:, 0], data[:, 1], c=colors, alpha=0.5)
    plt.title(title)
    plt.xlabel("PCA Component 1")
    plt.ylabel("PCA Component 2")

    legend = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=class_colors[label], markersize=10) for label in
labels]
    plt.legend(legend, labels, title="Emotions", loc="upper right")
    plt.show()

unique_labels = df_filtered['label'].unique()
plot_2d(t_count_reduced, "PCA on Token Count Features", colors,
unique_labels)
plot_2d(X_tfidf_reduced, "PCA on TF-IDF Features", colors,
unique_labels)
```



PCA on Token Count Features
Emotions
joy
sadness
anger
fear
PCA Component 2
PCA Component 1
3
2
1
0
−1
−2
−1
0
1
2
3
4
5



First Plot: There is a huge amount of overlap between emotions in the plot which seems to overlap which makes it difficult to distinguish any emotions are separable

Second Plot: Similar to the first plot there is also a overlap between emotions but there is a slight separation for the joy emotion as they are isolated from the remaining points

In summary there might be a one visually separable class which can be joy which has minute data points, but for the first plot there is no visually separable class.