PROGRAMMING ASSIGNMENT-1

1. http://vision.stanford.edu/aditya86/ImageNetDogs/
2. https://scikit-image.org/

a) CroppingandResizeImagesinYour4-classImagesDataset:

https://www.kaggle.com/code/espriella/stanford-dogs-transfer-crop-stack/notebook

In [1]:
```
import os,cv2
dir = os.getcwd()
images_folder = 'Images/'
os.listdir(os.path.join(dir,images_folder))
```

Out[1]:
```
['Great_Dane', 'papillon', 'Pomeranian', 'Rhodesian_ridgeback']
```

In [41]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#import required libraries

import os,cv2
import xml.etree.ElementTree as ET

dir = os.getcwd()
images_folder = 'Images/'
annotations_folder = 'Annotations/'
out_folder = "Cropped_Images"
os.makedirs(out_folder, exist_ok=True)
def is_xml_file(file_path):
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            return '<annotation>' in content
    except Exception as e:
        return False

# Loop through each specific folder
for folder in os.listdir(os.path.join(dir,images_folder)) :
    img_folder = os.path.join(images_folder, folder)
    annot_folder = os.path.join(annotations_folder, folder)
    for filename in os.listdir(annot_folder):
        file_path = os.path.join(annot_folder, filename)
        if is_xml_file(file_path):
            tree = ET.parse(file_path)
            root = tree.getroot()
            image_filename = root.find('filename').text
            image_path = os.path.join(dir, img_folder, image_filename)
```

```
            image = cv2.imread(image_path+".jpg")
            if image is not None:
                for obj in root.findall('object'):
                    bbox = obj.find('bndbox')
                    xmin = int(bbox.find('xmin').text)
                    ymin = int(bbox.find('ymin').text)
                    xmax = int(bbox.find('xmax').text)
                    ymax = int(bbox.find('ymax').text)
                    roi = image[ymin:ymax, xmin:xmax]
                    resized_roi = cv2.resize(roi,(128,128))
                    out_folder_path = os.path.join(out_folder, folder)
                    os.makedirs(out_folder_path, exist_ok=True)
                    out_path = os.path.join(out_folder_path, f'{image_filename}.jpg
                    cv2.imwrite(out_path,resized_roi)
print("-----------Cropping and ResizeImages completed----------")
```

```
----------Cropping and ResizeImages completed----------
```

(b) Feature Extraction: Edge histogram AND Similarity Measurements

i)Choose 1 image from each class.

In [6]:
```
images=['Images/papillon/n02086910_10147.jpg','Images/Great_Dane/n02109047_1005.jpg
```

ii. Convert the color images to grayscale images https://scikit-image.org/docs/stable/auto_examples/color_exposure/plot_rgb_to_gray.html

In [8]:
```
from skimage import io, color, filters, exposure
from skimage.color import rgb2gray
def convert_gray(image):
    img = io.imread(image)
    gray_img = color.rgb2gray(img)
    return gray_img
```

iii. For each image I, use the following import numpy as np from skimage import filters 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(I), filters.sobel_v(I)) to obtain an "angle" for each pixel in the images (Intuitively, one can think of the "angle" as the direction of edge gradient at the pixel).

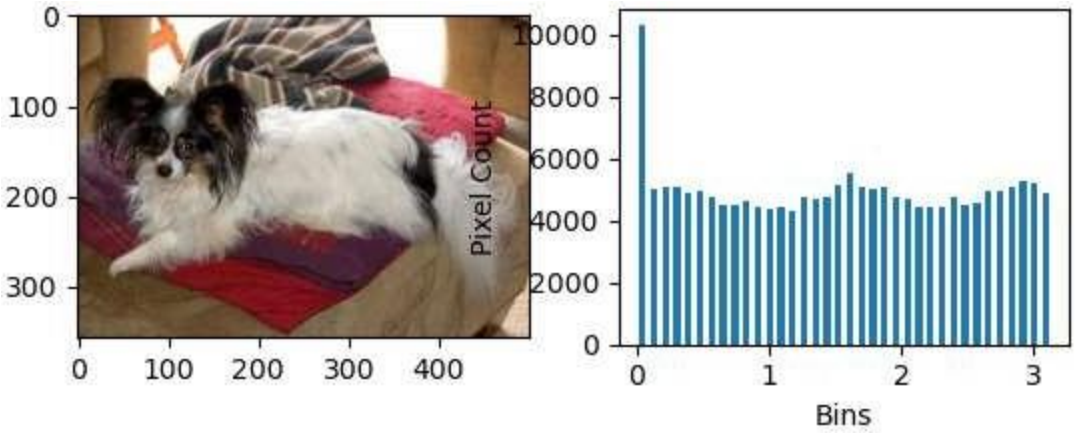
In [10]:
```
import numpy as np
def angle(dx, dy):
    return np.mod(np.arctan2(dy, dx), np.pi)
#angle_sobel = angle(filters.sobel_h(I),filters.sobel_v(I))
```

iv. Useskimage.exposure.histogram(seehttps://scikit-image.org/docs/stable/api/skimage.exposure.html#skimage.exposure.histogram)toobtainahistogramwith36 bins.

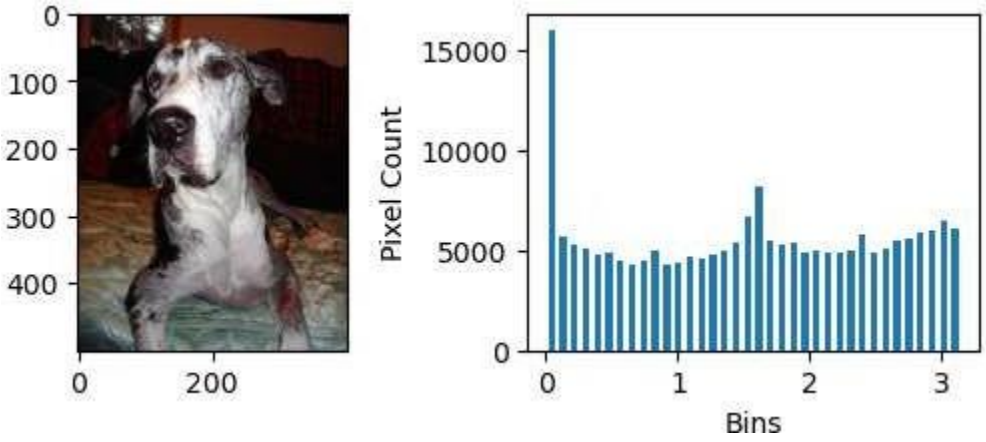
Plot the images with their corresponding edge histogram values (add x-axis label "Bins" and y-axis label "Pixel Count").

In [13]:
```
import matplotlib.pyplot as plt
edge_histograms=[]
for img in images:
    gray_image = convert_gray(img)
    angle_sobel = angle(filters.sobel_h(gray_image), filters.sobel_v(gray_image))
    hist,hist_centers =exposure.histogram(angle_sobel,nbins=36)
    edge_histograms.append(hist)
    print(img)
    plt.subplot(221),plt.imshow(io.imread(img))
    plt.subplot(222),plt.bar(hist_centers, hist, width=0.05, align='center')
    plt.xlabel("Bins")
    plt.ylabel("Pixel Count")
    plt.show()
```

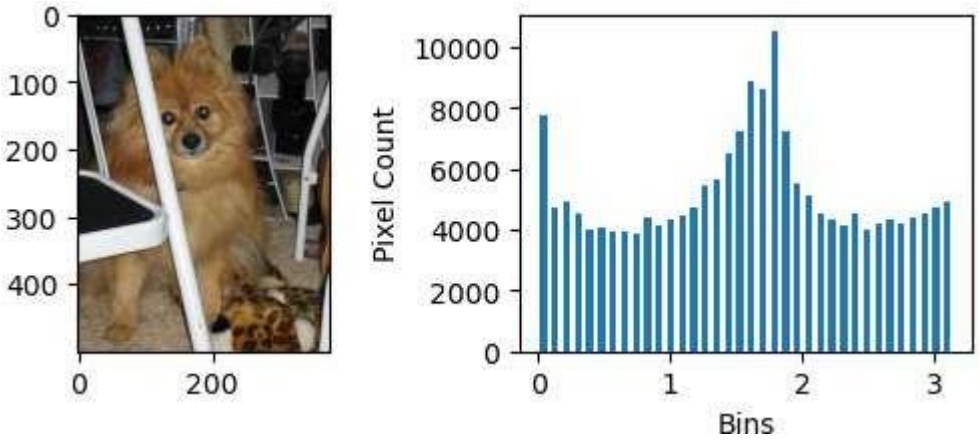
```
Images/papillon/n02086910_10147.jpg
```



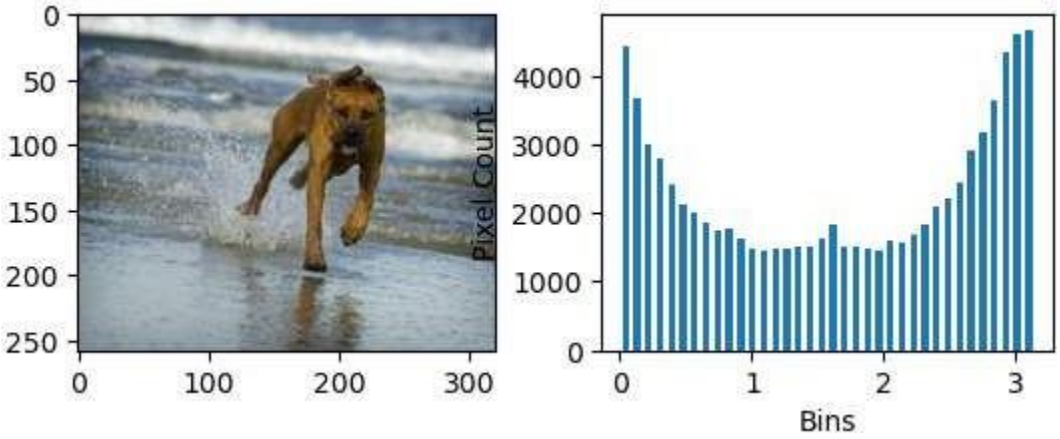

```
Images/Great_Dane/n02109047_1005.jpg
```




```
Images/Pomeranian/n02112018_10129.jpg
```




```
Images/Rhodesian_ridgeback/n02087394_10014.jpg
```



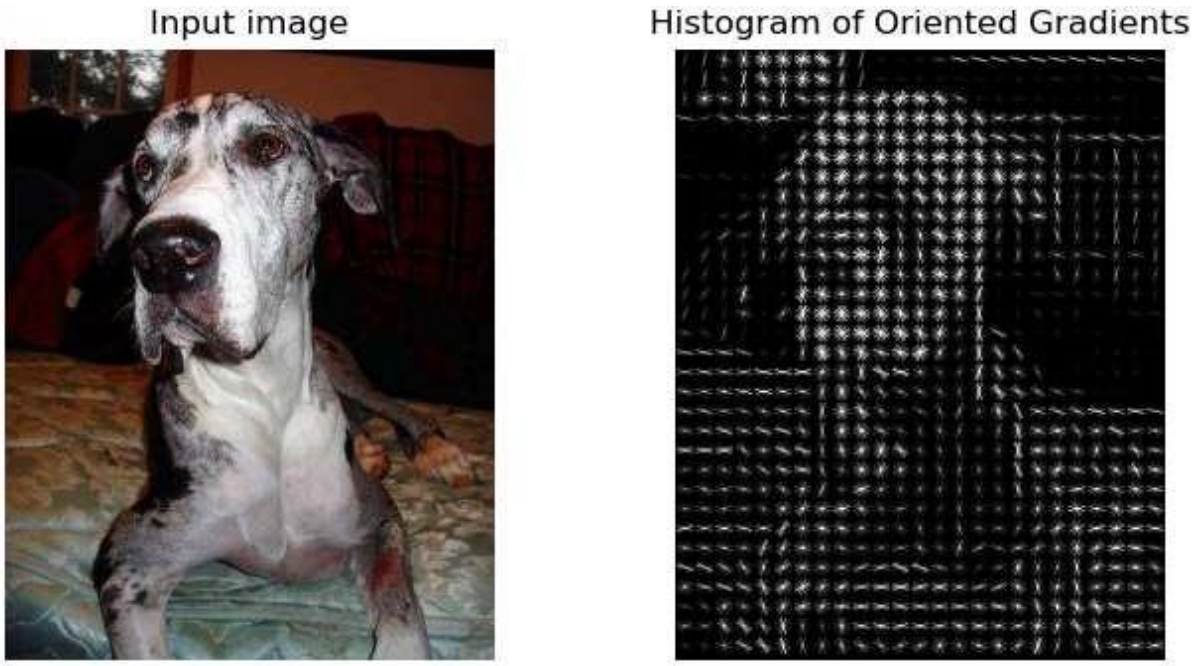
vi. Pick 2 edge histograms from the 4 you have constructed (These are the vector representations of the images) • Perform histogram comparison between the 2 edge histograms using the following met rics/measures. (seehttps://scikit-learn.org/stable/modules/generated/sklearn. metrics.pairwise.distance_metrics.html#sklearn.metrics.pairwise.distance_ metrics)– Euclidean distance– Manhattan distance– Cosine distance

In [15]:
```
from sklearn.metrics.pairwise import euclidean_distances, manhattan_distances, cosi

def histogram_comparison(hist1,hist2):
    print("Manhattan Distance : {}".format(manhattan_distances(hist1.reshape(1,-1),
    print("Euclidean Distance : {}".format(euclidean_distances(hist1.reshape(1,-1),
    print("Cosine Distance : {}".format(cosine_distances(hist1.reshape(1,-1),hist2.
histogram_comparison(edge_histograms[0],edge_histograms[1])
```

```
Manhattan Distance : [[23344.]]
Euclidean Distance : [[7115.52303067]]
Cosine Distance : [[0.01206799]]
```

(c) Histogram of Oriented Gradient (HOG) feature descriptor Pick 1 image and compute its HOG descriptors. Visualise the image and the HOG descriptors fortheimage(seehttps://scikit-image.org/docs/stable/auto_examples/features detection/plot_hog.html#sphx-glr-auto-examples-features-detection-plot-hog-py)

In [17]:
```
from skimage.feature import hog

image_path = images[1]
image = io.imread(image_path)
fd, hog_image = hog(
    image,
    orientations=8,
    pixels_per_cell=(16, 16),
    cells_per_block=(1, 1),
    visualize=True,
    channel_axis=-1,
)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)

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

# Rescale 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()
```



(d) Dimensionalityreduction(usingPrincipalComponentAnalysis,PCA) i. Use images from all four classes. ii. Convert all the images from the four classes to edge histograms.(0.5 points) iii. Perform Principal Component Analysis (PCA) dimensionality reduction on the set of his tograms to reduce from 36 to 2 dimensions. (Note: You should not use the class labels) (1 point) iv. Plot the 2D points using four different colors for data from the four classes (see Figure 1). Howmany classes are visually separable (i.e., non-overlapping) ? (1 point)

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

edge_histogram = []
classes = []

for index,name in enumerate(os.listdir(os.path.join(dir,images_folder))):
    path_folder = os.path.join('Cropped_Images',name)
    for file_path in os.listdir(path_folder):
        if file_path.endswith('.jpg') or file_path.endswith('.png'):
            image_path = os.path.join(path_folder, file_path)
            gray_img = convert_gray(image_path)
            angle_sobel = angle(filters.sobel_h(gray_img), filters.sobel_v(gray_img
            hist,hist_centers =exposure.histogram(angle_sobel,nbins=36)
            edge_histogram.append(hist)
            classes.append(index)

edge_histogram = np.array(edge_histogram)
classes = np.array(classes)

pca = PCA(n_components=2)
h_pca = pca.fit_transform(edge_histogram)
names=os.listdir(os.path.join(dir,images_folder))
plt.scatter(h_pca[:, 0], h_pca[:, 1], c=classes)
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('2D PCA ')
plt.colorbar()
plt.show()
```