#Gowtham Reddy Vanta Id: 916472365

# 2365 Saluki Basenji Norwich terrier Shetland sheepdog

# 2365 29(Test Set)

In [2]:
```
# Import all necessary libraries
import cv2 as cv
import pandas as pd
import glob
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import os
from os import listdir
from os.path import isfile, join
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from pathlib import Path
import xml.etree.ElementTree as ET
from skimage import filters, exposure, io, color
from skimage.feature import hog
from sklearn.metrics import pairwise_distances
```

In [3]:
```
dog_images = glob.glob('C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/GowthamImages/*/*')
annotations_dir = 'C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/GowthamAnnotations'
annotations = glob.glob('C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/GowthamAnnotations/*/*')
cropped = "./Cropped/"
img_size = 299  # For Xception input
train_dir = './Cropped'  # './Images'
batch_size_training = 256
batch_size_validation = 256
input_shape = (img_size, img_size, 3)
```

In [4]:
```
# The function to extract bounding boxes
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
```

In [5]:
```
# Create the output directory for cropped and resized images
cropped = "./Cropped/"
```

In [6]:
```
annotations=glob.glob('/C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/GowthamAnnotations/*/*')
```

In [7]:
```
######## Get image path from annotation path ########
def get_image(annot):
    img_path = '/C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/GowthamImages/'
    file = annot.split('/')
    img_filename = img_path + file[-2]+'/'+file[-1]+'.jpg'
    return img_filename
```

In [8]:
```
num_images = min(len(dog_images), len(annotations))

for i in range(num_images):
    bbox = get_bounding_boxes(annotations[i])
    dog = get_image(annotations[i])
    im = Image.open(dog)
    for j in range(len(bbox)):
        im2 = im.crop(bbox[j])
        im2 = im2.resize((128, 128), Image.ANTIALIAS)
        new_path = dog.replace('C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/', './Cropped/')
        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)
```

### Converting Color images to grayscale

In [9]:
```
# Load image and convert it to grayscale
def load_and_convert_to_grayscale(image_path):
    image = io.imread(image_path)
    grayscale_image = color.rgb2gray(image)
    return grayscale_image
```

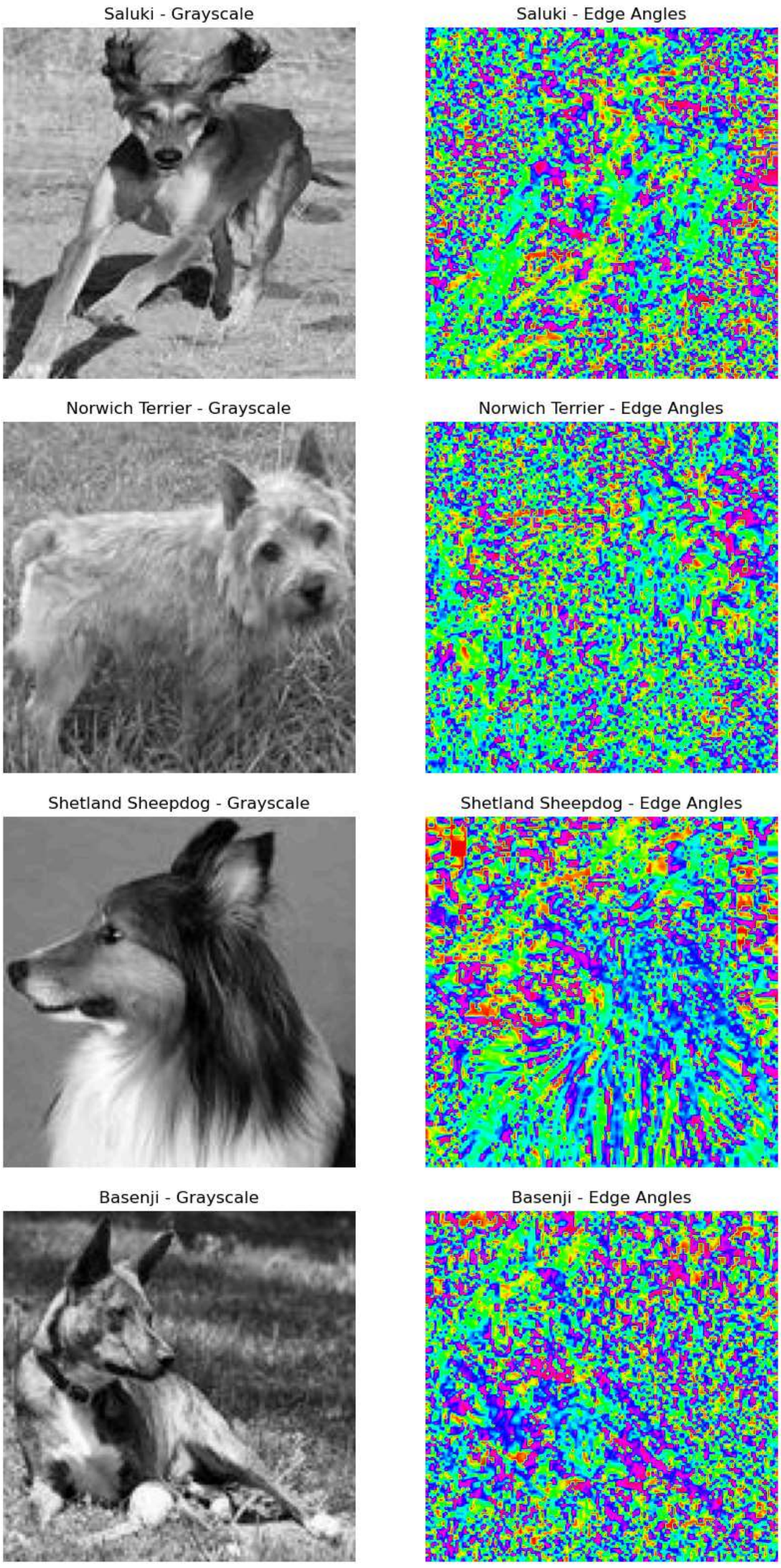
In [10]:
```
# Paths to images in each class
class_images = {
    "Saluki": "C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/Cropped/n02091831-Saluki/n02091831_1400-0.jpg",
    "Norwich Terrier": "C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/Cropped/n02094258-Norwich_terrier/n02094258_847-0.jpg",
    "Shetland Sheepdog": "C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/Cropped/n02105855-Shetland_sheepdog/n02105855_5719-0.jpg",
    "Basenji": "C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/Cropped/n02110806-basenji/n02110806_4395-0.jpg"
}
```

In [11]:
```
# Convert images
grayscale_images = {class_name: load_and_convert_to_grayscale(path) for class_name, path in class_images.items()}
```

In [12]:
```
# Define angle calculation function
def angle(dx, dy):
    """Calculate the angles between horizontal and vertical operators."""
    return np.mod(np.arctan2(dy, dx), np.pi)

# Apply Sobel filter to calculate edge angles
def compute_edge_angles(image):
    sobel_h = filters.sobel_h(image)  # Sobel horizontal
    sobel_v = filters.sobel_v(image)  # Sobel vertical
    angle_sobel = angle(sobel_h, sobel_v)
    return angle_sobel
```

```
# Calculate edge angles for each grayscale image
edge_angles = {class_name: compute_edge_angles(image) for class_name, image in grayscale_images.items()}
```

In [13]:
```
# Display grayscale image and edge angles
def plot_image_and_angles(class_name, grayscale_image, edge_angle_image):
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    axes[0].imshow(grayscale_image, cmap='gray')
    axes[0].set_title(f"{class_name} - Grayscale")
    axes[0].axis('off')

    axes[1].imshow(edge_angle_image, cmap='hsv')  # hsv colormap for better angle visualization
    axes[1].set_title(f"{class_name} - Edge Angles")
    axes[1].axis('off')

    plt.show()

# Plot images and angles for each class
for class_name in grayscale_images:
    plot_image_and_angles(class_name, grayscale_images[class_name], edge_angles[class_name])
```



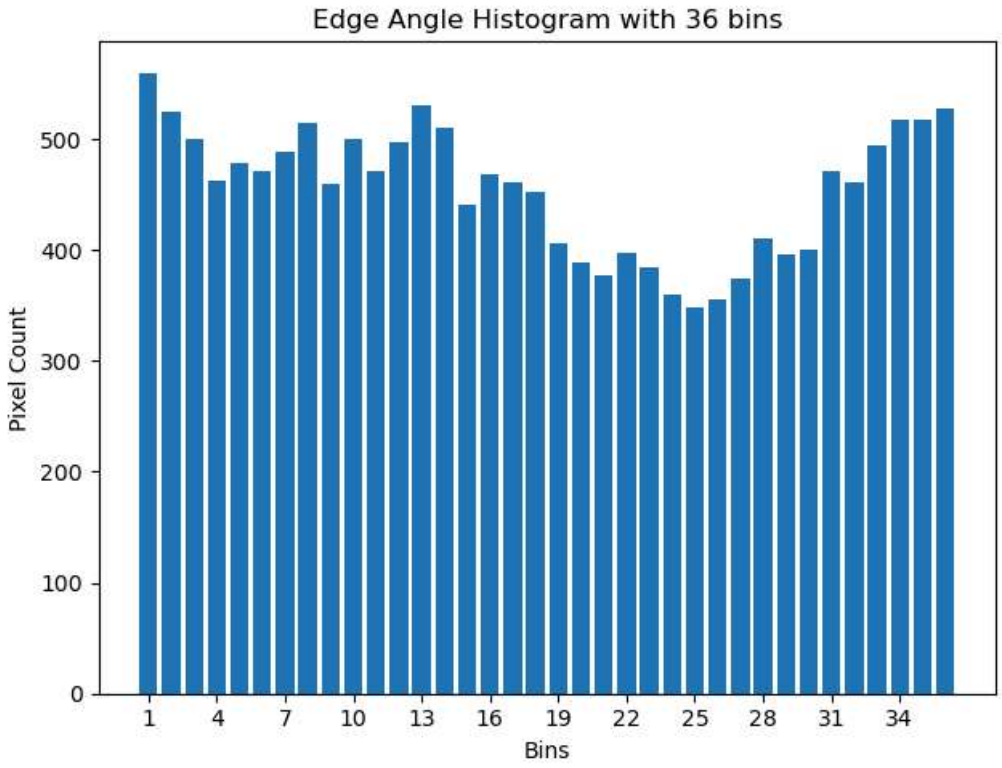

Edge Histograms(36 Bins)

In [14]:
```
# Compute the histogram with 36 bins for edge angles
def compute_histogram(edge_angle_image, bins=36):
    # Compute the histogram of the edge angles using skimage.exposure.histogram
    hist, _ = exposure.histogram(edge_angle_image, nbins=bins)
    return hist
```

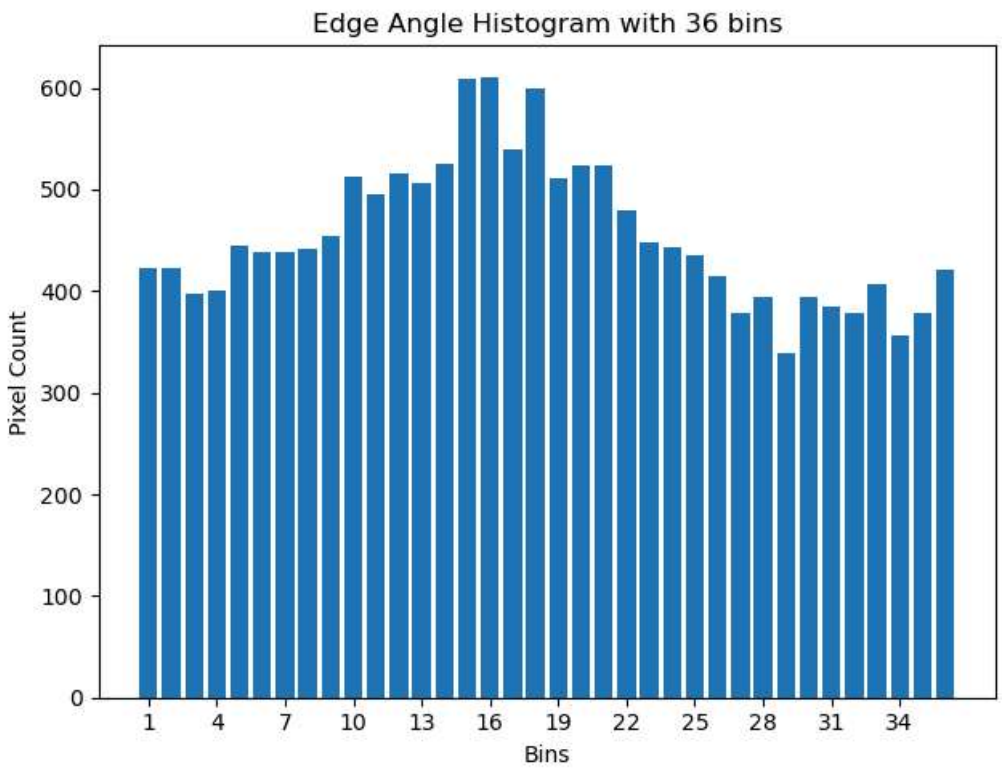
In [15]:
```
# Plot the grayscale image and its corresponding edge angle histogram side by side
def plot_image_and_histogram(image, edge_angle_image, hist, bins=36):
    # Create a figure with two subplots (1 row, 2 columns)
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))

    # Plot the grayscale image
    ax[0].imshow(image, cmap='gray')
    ax[0].set_title('Grayscale Image')
    ax[0].axis('off')  # Hide axis labels for the image

    # Plot the histogram with bin numbers on the x-axis
    ax[1].bar(range(1, bins+1), hist, align='center')
    ax[1].set_title('Edge Angle Histogram with 36 bins')
    ax[1].set_xlabel('Bins')
    ax[1].set_ylabel('Pixel Count')
    ax[1].set_xticks(range(1, bins+1, 3))  # Set x-ticks from 1 to 36, spaced by 3

    plt.tight_layout()  # Adjust layout so everything fits nicely
    plt.show()
```

In [16]:
```
for class_name, grayscale_image in grayscale_images.items():
    # Computing edge angles for the grayscale image
    print(f"Processing {class_name} image...")
    edge_angle_image = compute_edge_angles(grayscale_image)

    # Computing the histogram of edge angles
    hist = compute_histogram(edge_angle_image)

    # Plotting the grayscale image and corresponding histogram side by side
    plot_image_and_histogram(grayscale_image, edge_angle_image, hist)
```

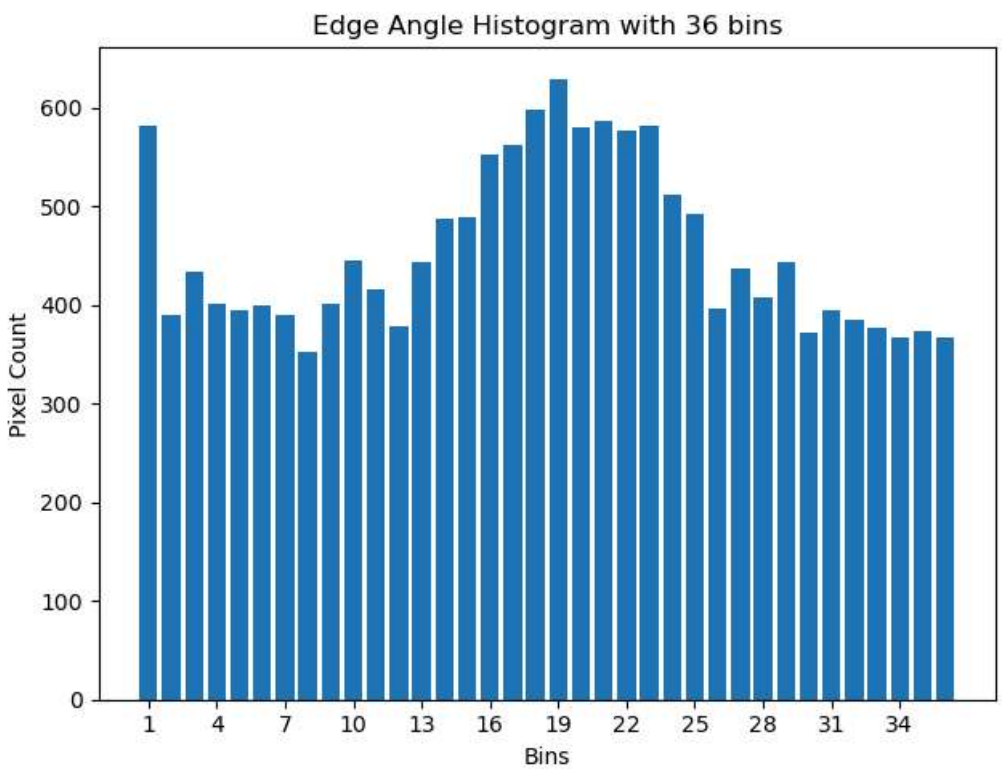
```
Processing Saluki image...
```

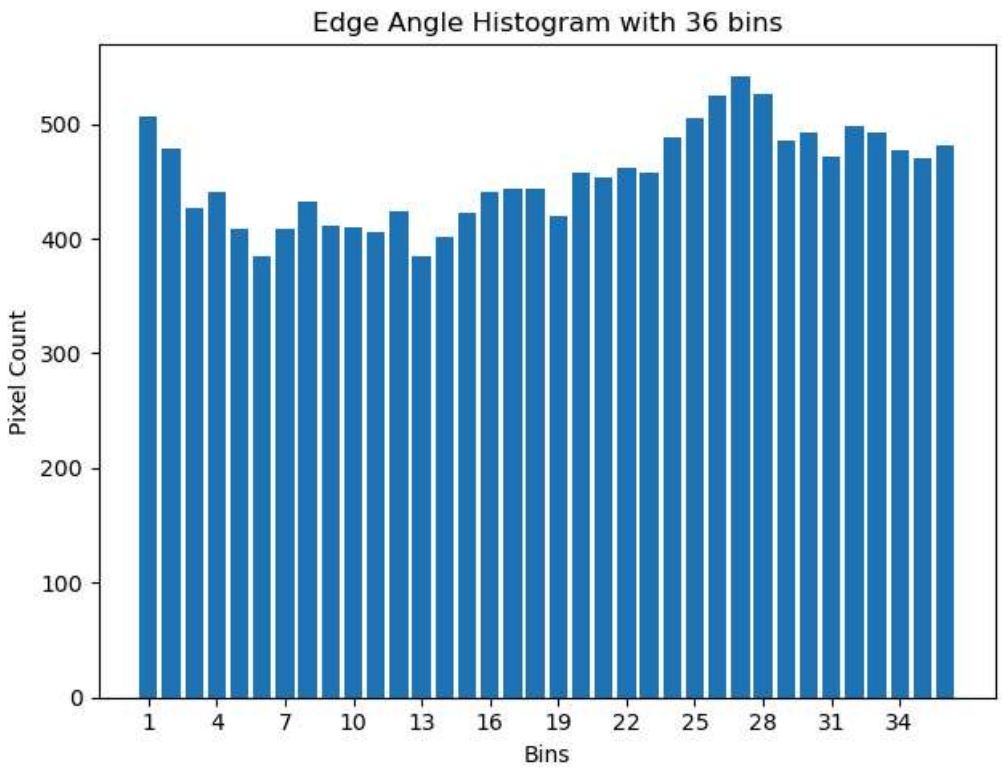

```
Processing Norwich Terrier image...
```


```
Processing Shetland Sheepdog image...
```


```
Processing Basenji image...
```


**Metrics**

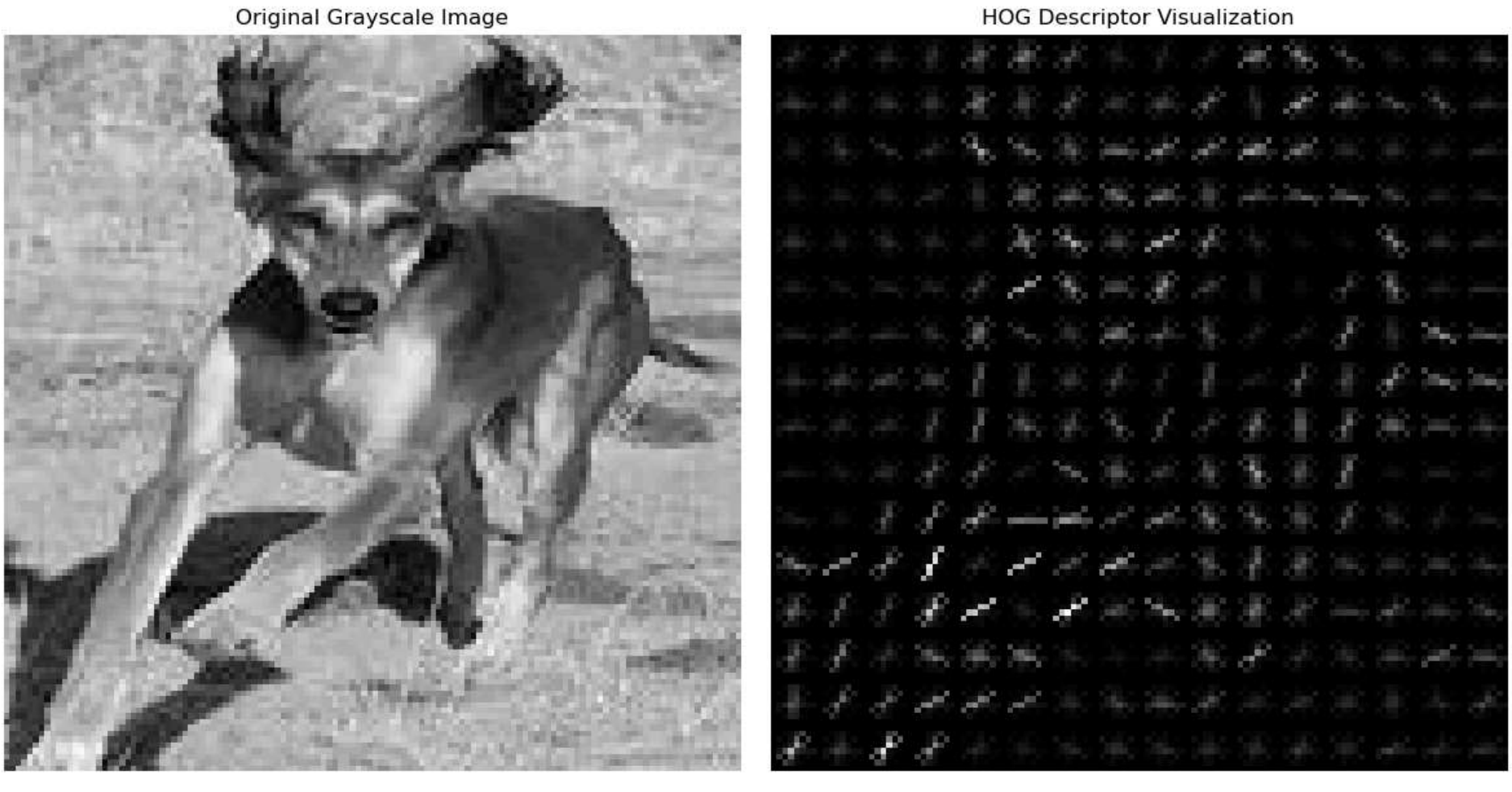
In [17]:
```python
# Function to compare two histograms using different distance metrics
def compare_histograms(hist1, hist2):
    # Reshape the histograms to be 2D arrays (required by pairwise_distances)
    hist1 = np.array(hist1).reshape(1, -1)
    hist2 = np.array(hist2).reshape(1, -1)

    # Compute Euclidean, Manhattan, and Cosine distances
    euclidean_dist = pairwise_distances(hist1, hist2, metric='euclidean')[0][0]
    manhattan_dist = pairwise_distances(hist1, hist2, metric='manhattan')[0][0]
    cosine_dist = pairwise_distances(hist1, hist2, metric='cosine')[0][0]

    # Print the results
    print(f"Euclidean Distance: {euclidean_dist}")
    print(f"Manhattan Distance: {manhattan_dist}")
    print(f"Cosine Distance: {cosine_dist}")

# Example usage: Pick two histograms from your constructed histograms
# For example, let's compare the histograms for "Saluki" and "Norwich Terrier"
hist1 = compute_histogram(compute_edge_angles(grayscale_images['Saluki']))
hist2 = compute_histogram(compute_edge_angles(grayscale_images['Norwich Terrier']))

# Compare the histograms
compare_histograms(hist1, hist2)
```

```
Euclidean Distance: 542.6932835405281
Manhattan Distance: 2744.0
Cosine Distance: 0.01936306298953805
```

**HOG**

In [18]:
```python
# Pick one image (e.g., "Saluki")
image = grayscale_images["Saluki"]

# Compute HOG descriptors and the HOG image for visualization
hog_descriptors, hog_image = hog(image, orientations=9, pixels_per_cell=(8, 8),
                                 cells_per_block=(2, 2), block_norm='L2-Hys',
                                 visualize=True, feature_vector=True)

# Visualize the original image and the HOG image side by side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), sharex=True, sharey=True)

# Original grayscale image
ax1.imshow(image, cmap='gray')
ax1.set_title('Original Grayscale Image')
ax1.axis('off')

# Rescale HOG image for better visualization
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

# HOG image
ax2.imshow(hog_image_rescaled, cmap='gray')
ax2.set_title('HOG Descriptor Visualization')
ax2.axis('off')
```

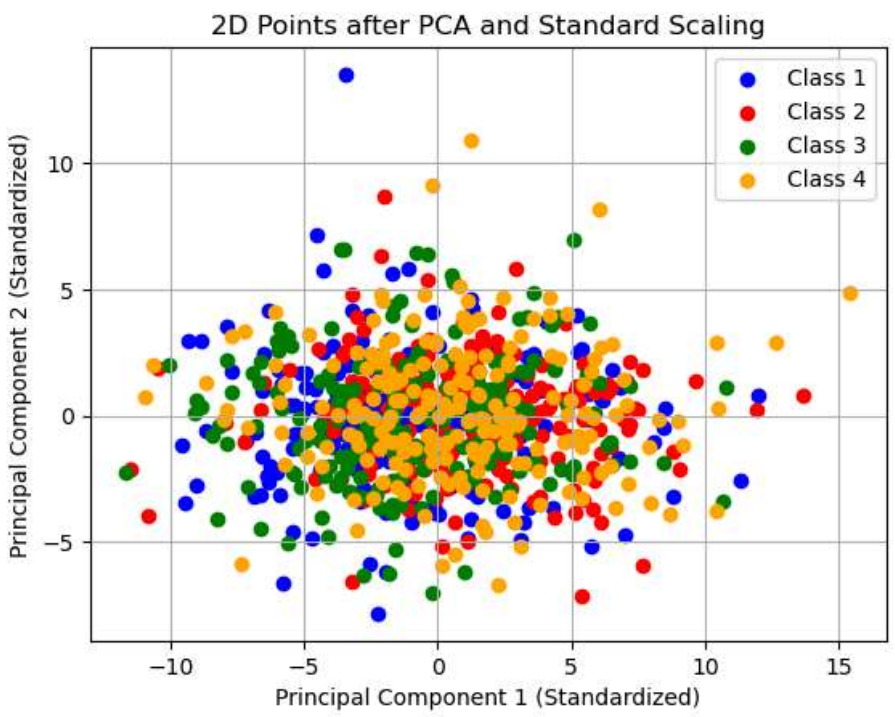
```
plt.tight_layout()
plt.show()
```




In [20]:

```
import cv2

def compute_edge_histogram(img):
    """Compute the edge histogram of the image."""
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    angle_sobel = np.mod(np.arctan2(filters.sobel_v(gray_img), filters.sobel_h(gray_img)), np.pi)
    hist, _ = exposure.histogram(angle_sobel.flatten(), nbins=36)
    return hist

def process_images(folder_path):
    """Process all images in a folder and return their edge histograms."""
    histograms = []
    for filename in os.listdir(folder_path):
        image_path = os.path.join(folder_path, filename)
        if image_path.lower().endswith(('.jpg', '.png')):  # Handle both jpg and png formats
            img = cv2.imread(image_path)
            hist = compute_edge_histogram(img)
            histograms.append(hist)
    return histograms

def plot_2d_points(reduced_data, num_classes):
    """Plot 2D points after PCA with different colors for different classes."""
    colors = ['blue', 'red', 'green', 'orange']  # Different colors for the four classes
    points_per_class = reduced_data.shape[0] // num_classes  # Calculate how many points per class
    for i in range(num_classes):
        start_index = i * points_per_class
        end_index = (i + 1) * points_per_class
        plt.scatter(reduced_data[start_index:end_index, 0],
                    reduced_data[start_index:end_index, 1],
                    color=colors[i], label=f'Class {i + 1}')

    plt.title('2D Points after PCA and Standard Scaling')
    plt.xlabel('Principal Component 1 (Standardized)')
    plt.ylabel('Principal Component 2 (Standardized)')
    plt.grid()
    plt.legend()
    plt.show()
```

```
# Folder paths for images from four classes
class_dirs = {
    "Saluki": "C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/Cropped/n02091831-Saluki",
    "Norwich Terrier": "C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/Cropped/n02094258-Norwich_terrier",
    "Shetland Sheepdog": "C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/Cropped/n02105855-Shetland_sheepdog",
    "Basenji": "C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/Cropped/n02110806-basenji"
}

# Initialize a list to store all histograms
all_histograms = []

# Process images and compute edge histograms for each class
for class_dir in class_dirs.values():
    class_histograms = process_images(class_dir)
    all_histograms.extend(class_histograms)  # Add histograms to the list

# Convert histograms to a NumPy array
all_histograms = np.array(all_histograms)

# Perform PCA for dimensionality reduction
pca = PCA(n_components=2)

# Use StandardScaler to normalize the data before applying PCA
scaler = StandardScaler()
reduced_data = pca.fit_transform(scaler.fit_transform(all_histograms))

# Plot the 2D points with different colors for each class without using class labels
plot_2d_points(reduced_data, num_classes=4)
```




No Classes are visually separable

In [21]:
```
# Using pandas to load the JSON file
train_df = pd.read_json('C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/train.json', lines=True)
test_df = pd.read_json('C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/test.json', lines=True)
val_df = pd.read_json('C:/Users/Gowtham reddy/Downloads/Gowtham_DM_Assignment1/Gowtham_DM_Assignment1/validation.json', lines=True)
```

In [22]:
```
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_tweet(tweet):
    # Convert to lowercase
    tweet = tweet.lower()

    # Remove URLs, mentions, and hashtags
    tweet = re.sub(r'http\S+|www\S+|@\S+|#\S+', '', tweet)

    # Remove punctuation
    tweet = tweet.translate(str.maketrans('', '', string.punctuation))

    # Tokenize the tweet
    words = word_tokenize(tweet)

    # Remove stopwords and lemmatize
    words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]

    # Join the words back into a single string
    return ' '.join(words)

# Apply preprocessing to the 'text' column in each dataset
train_df['processed_text'] = train_df['Tweet'].apply(preprocess_tweet)
test_df['processed_text'] = test_df['Tweet'].apply(preprocess_tweet)
val_df['processed_text'] = val_df['Tweet'].apply(preprocess_tweet)
```

In [23]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# Sample text data
X_train = train_df['processed_text']

# Initialize the CountVectorizer
count_vectorizer = CountVectorizer()
X_train_counts = count_vectorizer.fit_transform(X_train)

# Initialize the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

# Get the dimensionality of the count vector representation
count_dim = X_train_counts.shape
print(f'Count Vectorizer Dimensionality: {count_dim}')

# Get the dimensionality of the TF-IDF vector representation
tfidf_dim = X_train_tfidf.shape
print(f'TF-IDF Vectorizer Dimensionality: {tfidf_dim}')
```

```
Count Vectorizer Dimensionality: (3000, 6527)
TF-IDF Vectorizer Dimensionality: (3000, 6527)
```

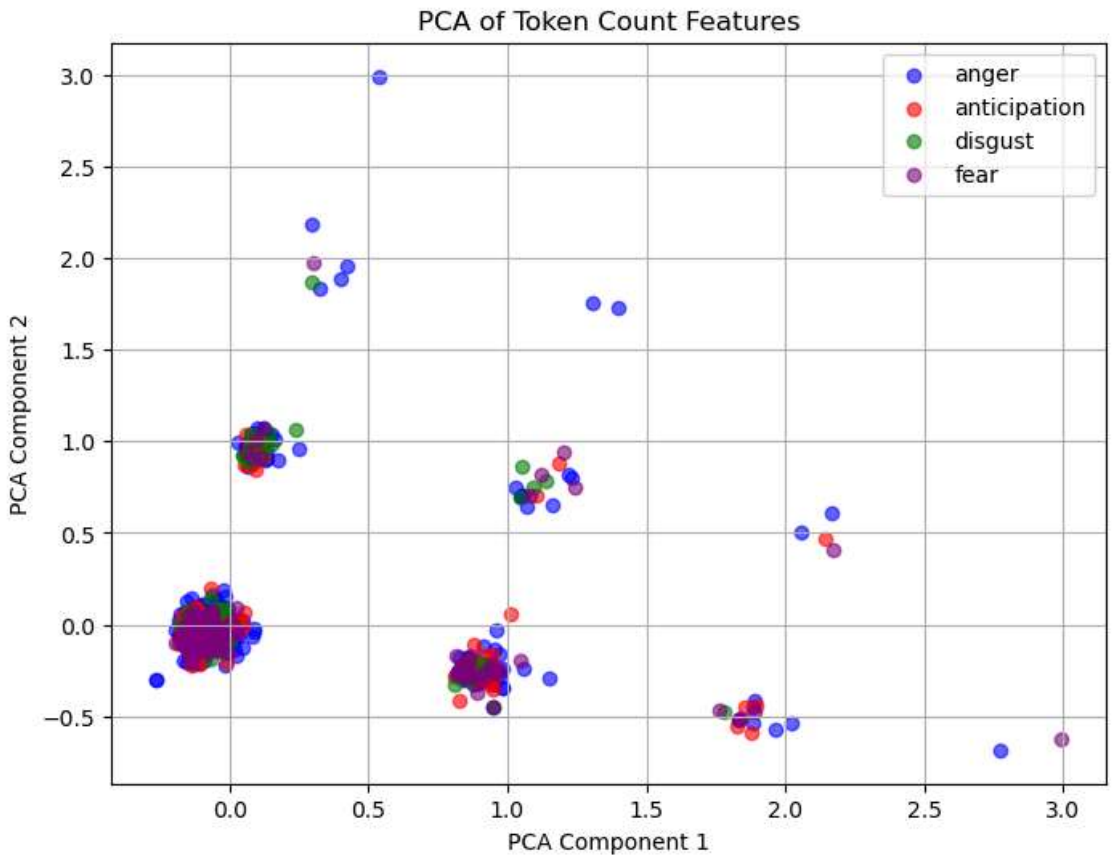
In [24]:
```
# Define the emotion columns
emotion_columns = ['anger', 'anticipation', 'disgust', 'fear', 'joy', 'love',
                   'optimism', 'pessimism', 'sadness', 'surprise', 'trust']

# Function to get the emotion label
def get_label(row):
    for emotion in emotion_columns:
        if row[emotion] == True:
            return emotion
    return None  # In case no emotion is labeled True

# Create the 'label' column
train_df['label'] = train_df.apply(get_label, axis=1)
test_df['label'] = test_df.apply(get_label, axis=1)
val_df['label'] = val_df.apply(get_label, axis=1)
```

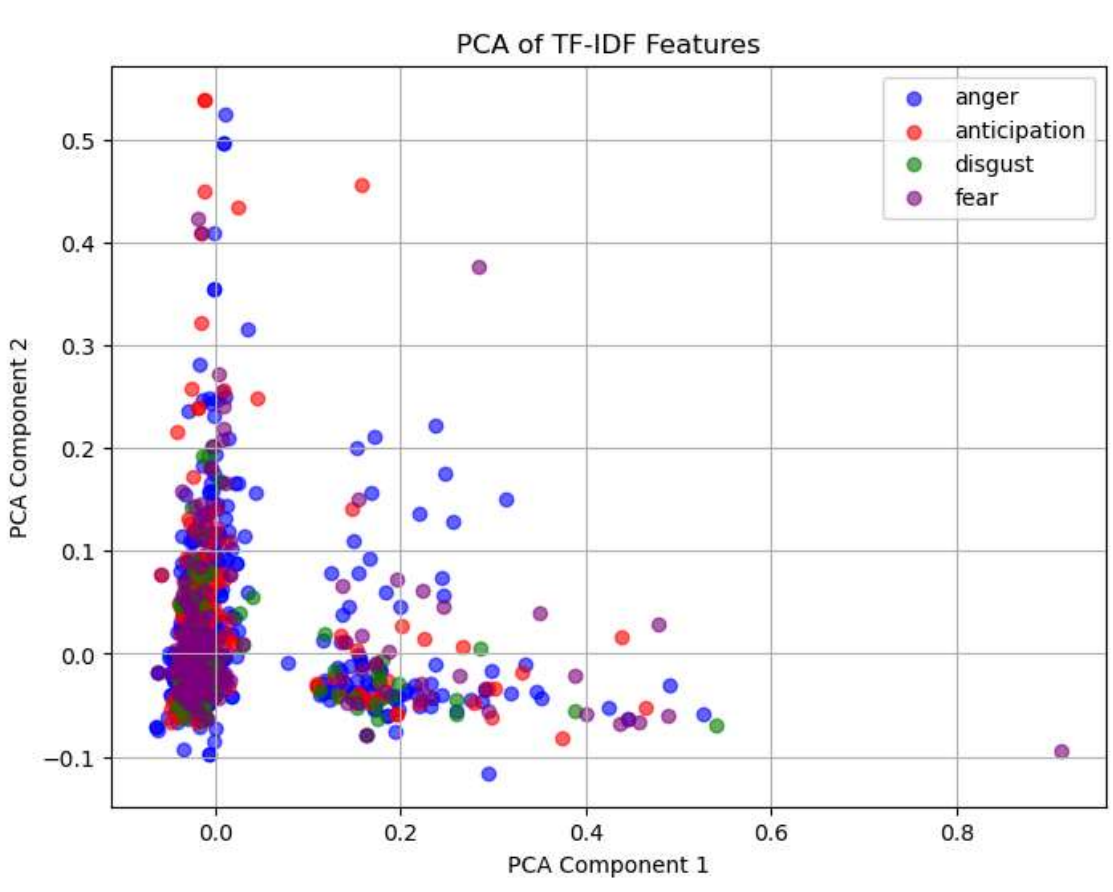
Four Classes: 'anger', 'anticipation', 'disgust', 'fear'

In [25]:
```
# Define the four classes
selected_classes = ['anger', 'anticipation', 'disgust', 'fear']

# Filter the training data for only the selected classes
```

```
filtered_train_df = train_df[train_df['label'].isin(selected_classes)]

# Prepare feature representations for token counts and tf-idf
X_filtered_count = count_vectorizer.transform(filtered_train_df['processed_text'])
X_filtered_tfidf = tfidf_vectorizer.transform(filtered_train_df['processed_text'])
```

In [26]:
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np

# Get the corresponding labels
y_filtered = filtered_train_df['label']

# Perform PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
X_count_pca = pca.fit_transform(X_filtered_count.toarray())
X_tfidf_pca = pca.fit_transform(X_filtered_tfidf.toarray())

# Assign colors for each class
colors = {
    'anger': 'blue',
    'anticipation': 'red',
    'disgust': 'green',
    'fear': 'purple'
}

# Plot function for 2D points
def plot_2d(X, y, title):
    plt.figure(figsize=(8, 6))
    for label in selected_classes:
        indices = np.where(y == label)
        plt.scatter(X[indices, 0], X[indices, 1], c=colors[label], label=label, alpha=0.6)
    plt.title(title)
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.legend(loc='best')
    plt.grid(True)
    plt.show()

# Plot for token count features
plot_2d(X_count_pca, y_filtered, 'PCA of Token Count Features')

# Plot for tf-idf features
plot_2d(X_tfidf_pca, y_filtered, 'PCA of TF-IDF Features')
```




No classes are visually separable

In [ ]: