Analysis of Exclusively Dark Images Dataset using OpenCV and Computer Vision

Project Description:

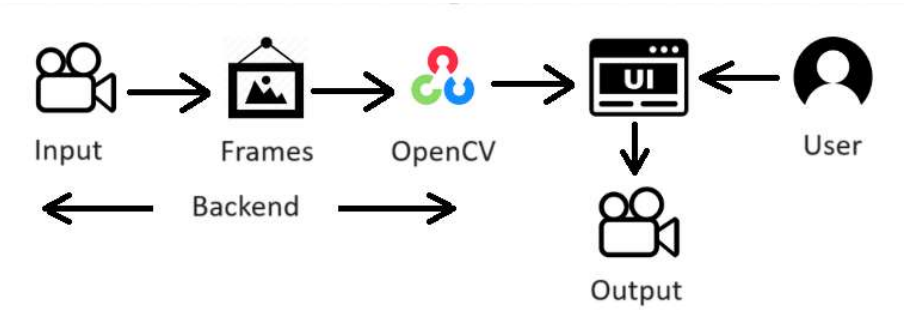
The objective of the project is to improve dark images by applying computer vision algorithms. Enhancing the visibility and quality of underexposed or poorly lit photographs is the aim in order to make them more useful for a range of applications.

Scenario 1: Security and Surveillance
Low light levels often generate footage that is too dark to see crucial information in the security and surveillance industries, like person identification or suspicious activity detection. Through the use of cutting-edge computer vision techniques, this study seeks to improve such dark images. This project makes it easier for security professionals to monitor and effectively respond to any threats by enhancing the visibility of crucial characteristics in surveillance footage through the application of techniques like gamma correction and histogram equalisation.

Scenario 2: Medical Imaging
Underexposure may result in images from medical imaging tests like CT, MRI, and X-rays that are too dark to read clearly. The goal of this project is to create image enhancement techniques that will improve the clarity and brightness of these medical photographs. The project uses tools such as OpenCV to apply contrast modifications and other upgrades to show hidden features.

Scenario 3: Night - Time Photography
Poor lighting is a common problem for nighttime photographers, leading to dark underexposed pictures that lack clarity and detail. This study investigates how to improve such photographs by adding more colour and detail using computer vision algorithms. The project gives photographers the skills to recover lost details in nighttime photos using techniques like contrast stretching and brightness correction with OpenCV, transforming useless images into aesthetically pleasing and high-quality shots.

Technical Architecture:



Pre-requisites:

To complete this project, you must require the following software, concepts, and packages.

1. **Python Packages**

   If you are using **anaconda navigator**, follow the below steps to download the required packages:

   Open the Anaconda prompt and create a virtual environment.

   - Type “pip install opencv-python==4.9.0.80” and click enter.
   - Type “pip install numpy ==1.26.3” and click enter
   - Type “pip install flask” and click enter.

2. Proficiency in Machine Learning and Deep Learning models in Jupyter Notebook and Google Colaboratory.

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- Python
- OpenCV - https://www.youtube.com/watch?v=WQeoO7MI0Bs
- **Experience with Jupyter Notebooks & Google Colaboratory:** Familiarity with using Jupyter or Colab notebooks for coding, visualization, and documenting work.
- **Linear Algebra:** Understanding of matrices and operations on them, which are fundamental for image processing tasks.
- **Probability and Statistics:** Basic knowledge is useful for understanding image noise, filtering techniques, and statistical image enhancements.
- Basic Understanding of Image Processing Techniques

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for computer vision.
- Gain knowledge of OpenCV.

Project Flow:

1. Project Planning and Setup

- Define Project Scope
- Data Collection
- Environment Setup

2. Data Loading and Preliminary Inspection

- Load Data
- Preliminary Data Inspection

3. Exploratory Data Analysis (EDA)

- Statistical Analysis of Image Data
- Histogram Analysis
- Metadata Analysis (if applicable)
- Visual Inspection

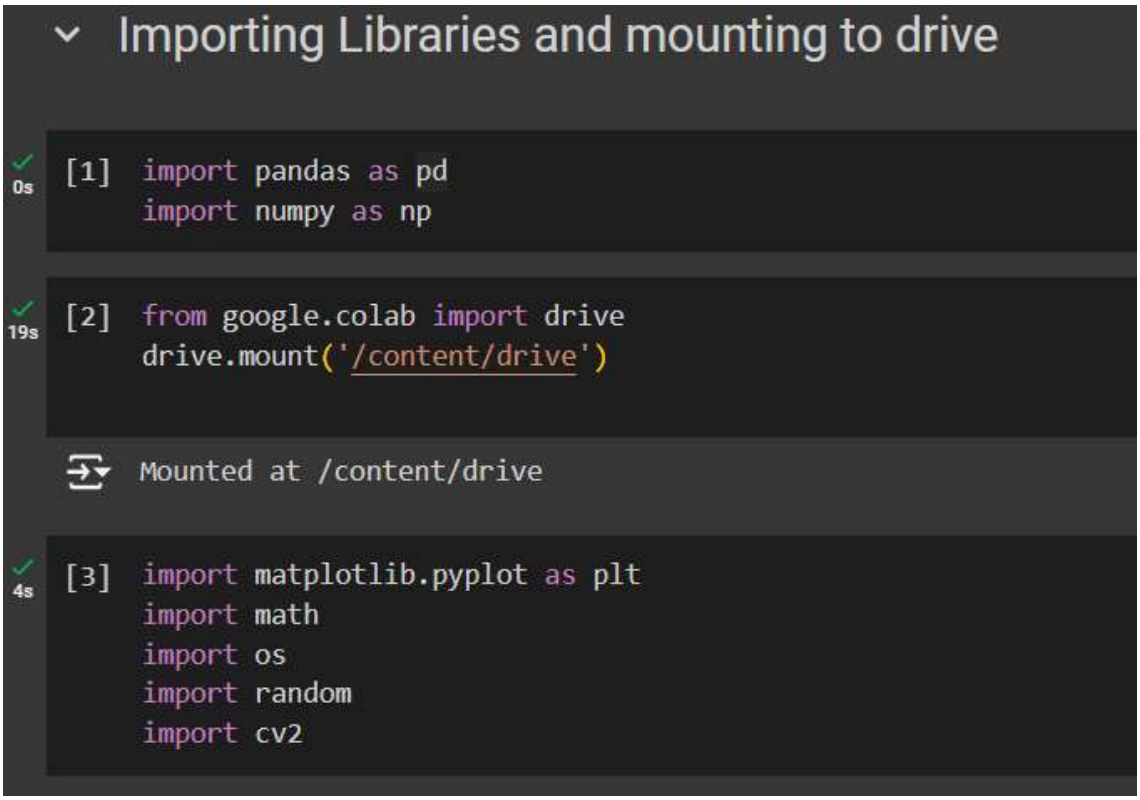
4. Image Enhancement Techniques

- Apply Basic Image Enhancement Techniques
- Visualise Results

Milestone 1: Analysis Dark Images Dataset

Activity 1: Importing libraries amd mounting to drive

- Import necessary modules (numpy, pandas, matplotlib, cv2, os).
- Code snippet for importing these libraries and setting up the environment.

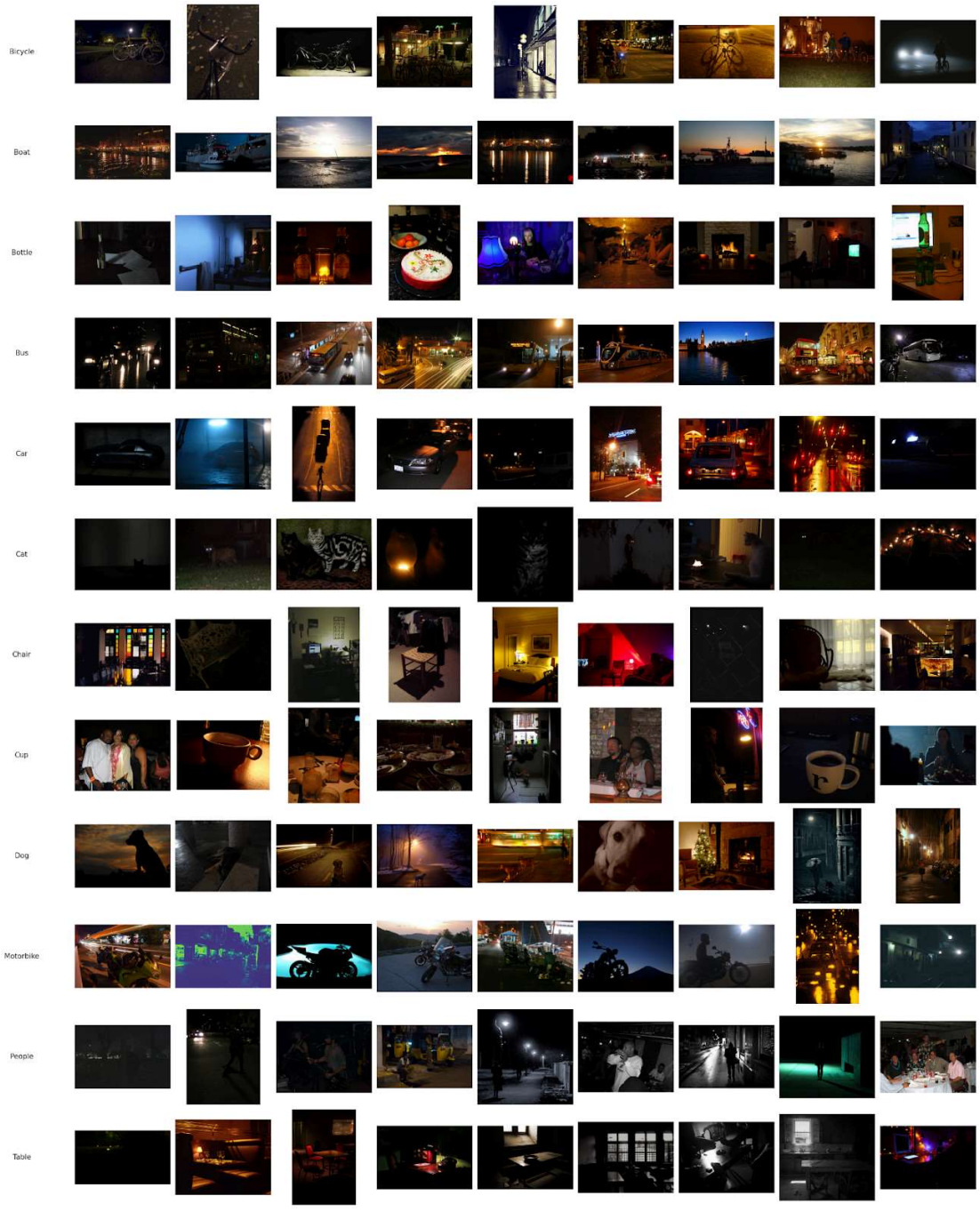


Activity 2: Loading and displaying Extremely Dark Data

- Load the entire Dataset via drive
- Copy the paths of all the dataset folders
- Explanation of how the data (dark images) was loaded from a directory.
- Visualisation of a subset of images using a custom function, visualize_images, to display a grid of sample images.

```python
import os
import matplotlib.pyplot as plt
from matplotlib.image import imread

def visualize_images(folders, num_images):
    total_images = num_images * len(folders)
    rows = len(folders)
    cols = num_images + 1  # +1 for the heading in each row

    fig = plt.figure(figsize=(cols * 3, rows * 3))

    for row, folder in enumerate(folders):
        image_files = os.listdir(folder)[:num_images]
        folder_name = os.path.basename(folder)

        # Adding folder name as the heading
        ax = fig.add_subplot(rows, cols, row * cols + 1)
        ax.text(0.5, 0.5, folder_name, horizontalalignment='center', verticalalignment='center', fontsize=15)
        ax.axis('off')

        for col, image_file in enumerate(image_files):
            img = imread(os.path.join(folder, image_file))
            ax = fig.add_subplot(rows, cols, row * cols + col + 2)
            ax.imshow(img)
            ax.axis('off')

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

```python
# List of folders to visualize images from
folders = [
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Bicycle',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Boat',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Bottle',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Bus',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Car',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Cat',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Chair',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Cup',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Dog',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Motorbike',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/People',
    '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Table'
]

# Number of images to display from each folder
num_images = 9

visualize_images(folders, num_images)
```



Activity 3: Data Inspection

- Define functions to count dark images in each folder.
- Preprocessing images and splitting the dataset into training and testing sets
- Checking the structure of the dataset
- Overview of inspecting the images to understand the extent of darkness and the types of images in the dataset.
- Key observations made during this step.

```python
# Function to calculate the average pixel intensity of an image
def is_dark_image(image, threshold=50):
    # Convert the image to grayscale
    grayscale_image = image.convert('L')
    # Calculate the average pixel intensity
    avg_intensity = np.mean(grayscale_image)
    # Check if the average intensity is below the threshold
    return avg_intensity < threshold

# Function to count dark images in each folder
def count_dark_images_in_folders(folders, target_size, threshold=50):
    dark_image_counts = {}
    for folder in folders:
        dark_count = 0
        image_files = os.listdir(folder)
        for file in image_files:
            img_path = os.path.join(folder, file)
            img = Image.open(img_path).convert('RGB')
            img = img.resize(target_size)  # Resize image to target size
            if is_dark_image(img, threshold):
                dark_count += 1
        dark_image_counts[os.path.basename(folder)] = dark_count
    return dark_image_counts

# Define the target size for resizing images
target_size = (64, 64)

# Count dark images in each folder
dark_image_counts = count_dark_images_in_folders(folders, target_size)

# Display the results
for folder, count in dark_image_counts.items():
    print(f'{folder}: {count} dark images')
```

```
Bicycle: 457 dark images
Boat: 373 dark images
Bottle: 432 dark images
Bus: 391 dark images
Car: 532 dark images
Cat: 615 dark images
Chair: 472 dark images
Cup: 374 dark images
Dog: 626 dark images
Motorbike: 365 dark images
People: 528 dark images
Table: 387 dark images
```

```python
# Define the target size for resizing images
target_size = (128, 128)

# Function to preprocess images
def preprocess_images(images):
    preprocessed_images = []
    for img in images:
        img = np.array(img) / 255.0  # Normalize pixel values
        preprocessed_images.append(img)
    return np.array(preprocessed_images)

# Process images in batches
batch_size = 100
all_images = []
all_labels = []
for images_batch, labels_batch in load_images_in_batches(folders, batch_size, target_size):
    preprocessed_images = preprocess_images(images_batch)
    all_images.append(preprocessed_images)
    all_labels.append(labels_batch)

all_images = np.vstack(all_images)
all_labels = np.hstack(all_labels)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(all_images, all_labels, test_size=0.2, random_state=42)

print(f'Training set size: {X_train.shape[0]}')
print(f'Testing set size: {X_test.shape[0]}')
```

```
Training set size: 5908
Testing set size: 1478
```

```python
[18] # 1. Check Dataset Structure
def check_dataset_structure(folders):
    dataset_info = {}
    for folder in folders:
        num_images = len(os.listdir(folder))
        dataset_info[folder] = num_images
    return dataset_info

dataset_structure = check_dataset_structure(folders)
print("Dataset Structure:", dataset_structure)
```

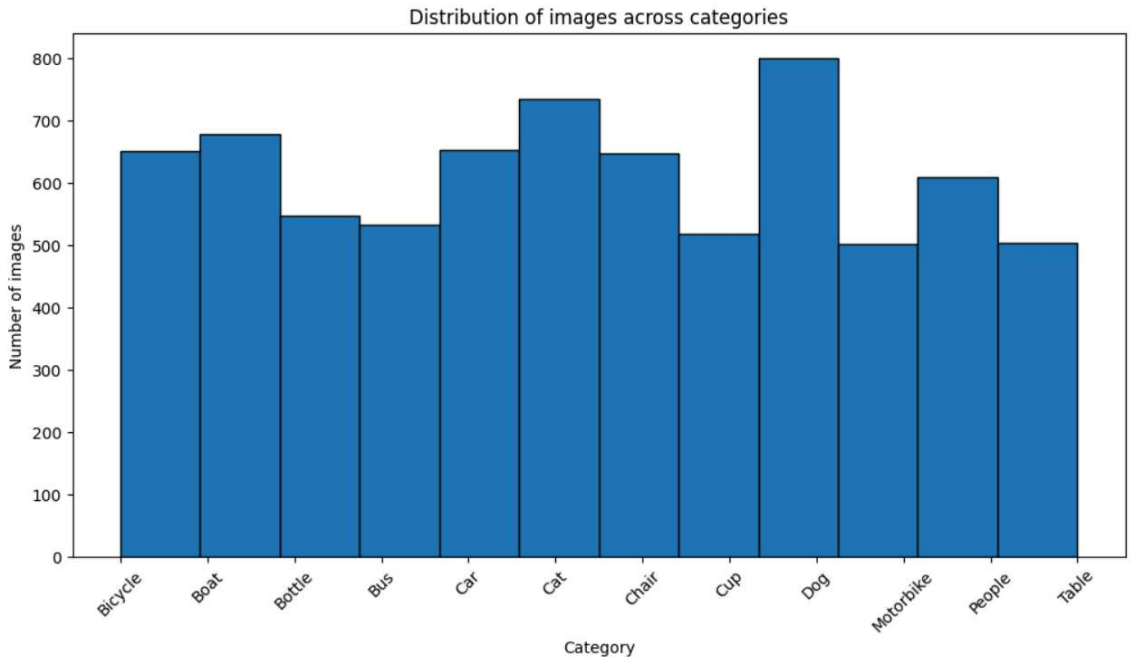
```
Dataset Structure: {'/content/drive/MyDrive/Exclusively Dark Data/ExDark/Bicycle': 652, '/content/drive/MyDrive/Exclusively Dark Data/ExDark/Boat
```

Activity 4: Analysing data by basic visualisations

- Plotting Distribution of images across categories.
- Checking if there are any missing or corrupted images.
- Checking the dimensions of each image of the dataset.

Data Visualization

```python
# Plot the distribution of images across categories
plt.figure(figsize=(12, 6))
plt.hist(all_labels, bins=len(folders), edgecolor='black')
plt.xticks(range(len(folders)), [os.path.basename(folder) for folder in folders], rotation=45)
plt.xlabel('Category')
plt.ylabel('Number of images')
plt.title('Distribution of images across categories')
plt.show()
```




```
[20] def check_missing_or_corrupted_images(folders):
         missing_or_corrupted = []
         for folder in folders:
             for file in os.listdir(folder):
                 img_path = os.path.join(folder, file)
                 try:
                     img = cv2.imread(img_path)
                     if img is None:
                         missing_or_corrupted.append(img_path)
                 except:
                     missing_or_corrupted.append(img_path)
         return missing_or_corrupted

     missing_or_corrupted_images = check_missing_or_corrupted_images(folders)
     print("Missing or Corrupted Images:", missing_or_corrupted_images)

     Missing or Corrupted Images: []

     def check_image_dimensions(folders):
         dimensions = []
         for folder in folders:
             for file in os.listdir(folder):
                 img_path = os.path.join(folder, file)
                 img = cv2.imread(img_path)
                 dimensions.append(img.shape[:2])  # Height, Width
         return dimensions

     image_dimensions = check_image_dimensions(folders)
     print("Image Dimensions:", image_dimensions)

     Image Dimensions: [(683, 1024), (500, 375), (465, 900), (480, 640), (1087, 734), (730, 1095), (360, 640), (375, 500), (338, 507), (428, 640), (7
```

Milestone 2: Exploratory Data Analysis and Computer Vision techniques

Exploratory Data Analysis (EDA) is a crucial step in data-driven projects, particularly in computer vision. It helps identify patterns, spot anomalies, test hypotheses, and check assumptions. EDA helps in understanding image data characteristics before applying advanced processing techniques. It includes image size and aspect ratio distribution, color channel analysis, histogram analysis, data visualisation, and class distribution for assessing quality and identifying imbalances.

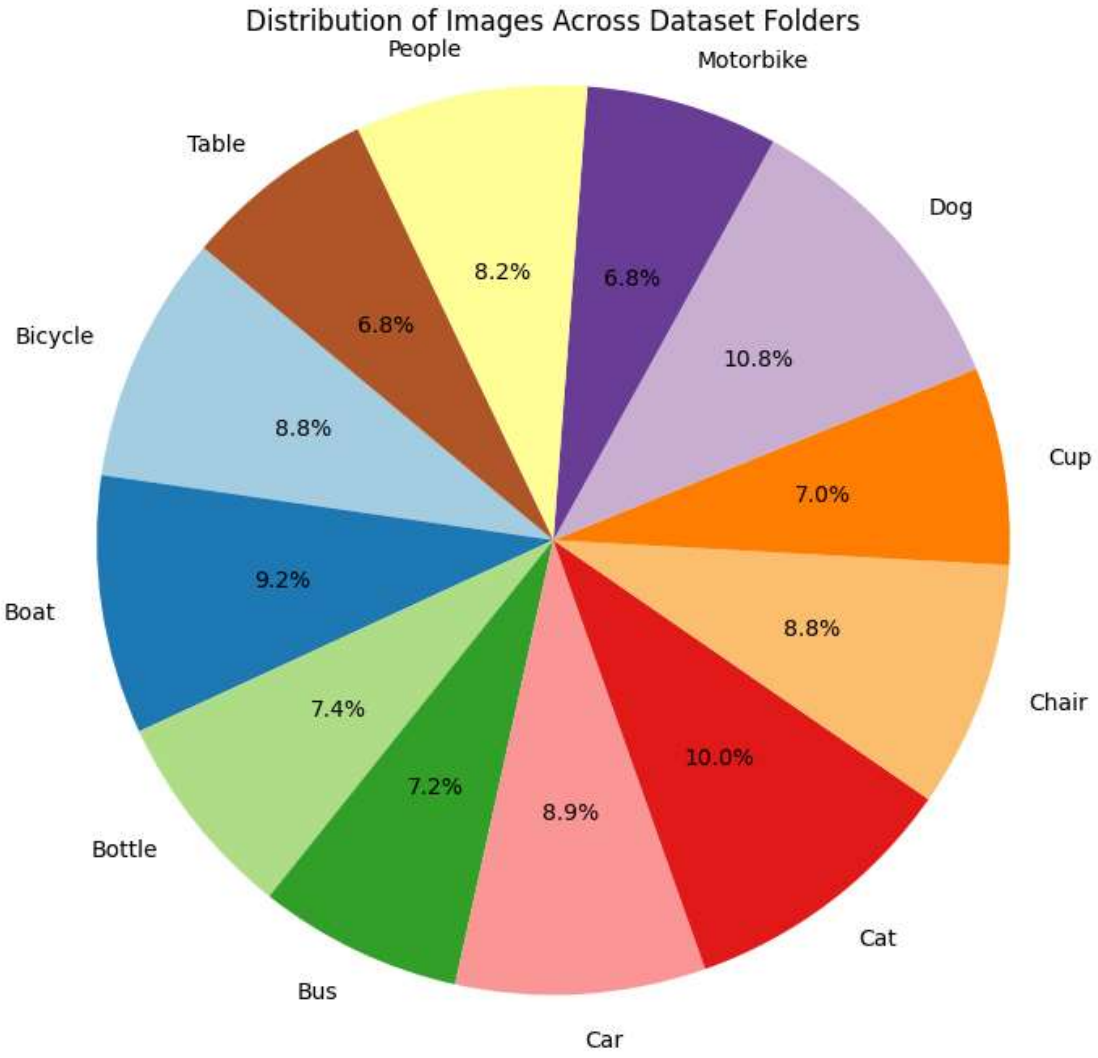
EDA in Computer Vision aids in data quality assessment, understanding variability, and feature engineering, identifying low-quality images, guiding preprocessing steps, and enhancing model performance.

Activity 1: Exploratory Data Analysis

EXPLORATORY DATA ANALYSIS

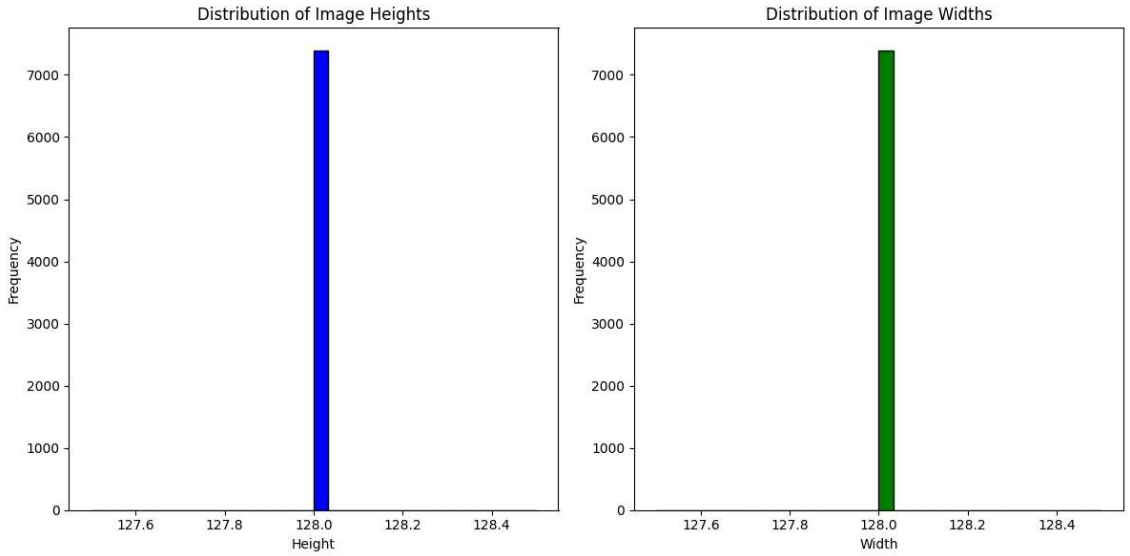
```
[22] def count_images_in_folders(folders):
         folder_names = []
         image_counts = []
         for folder in folders:
             folder_names.append(os.path.basename(folder))  # Get the folder name
             image_counts.append(len(os.listdir(folder)))  # Count images in the folder
         return folder_names, image_counts

     # Function to plot the pie chart
     def plot_image_distribution(folder_names, image_counts):
         plt.figure(figsize=(8, 8))
         plt.pie(image_counts, labels=folder_names, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
         plt.title('Distribution of Images Across Dataset Folders')
         plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
         plt.show()
```

```
[23] folder_names, image_counts = count_images_in_folders(folders)

     # Plot the pie chart
     plot_image_distribution(folder_names, image_counts)
```

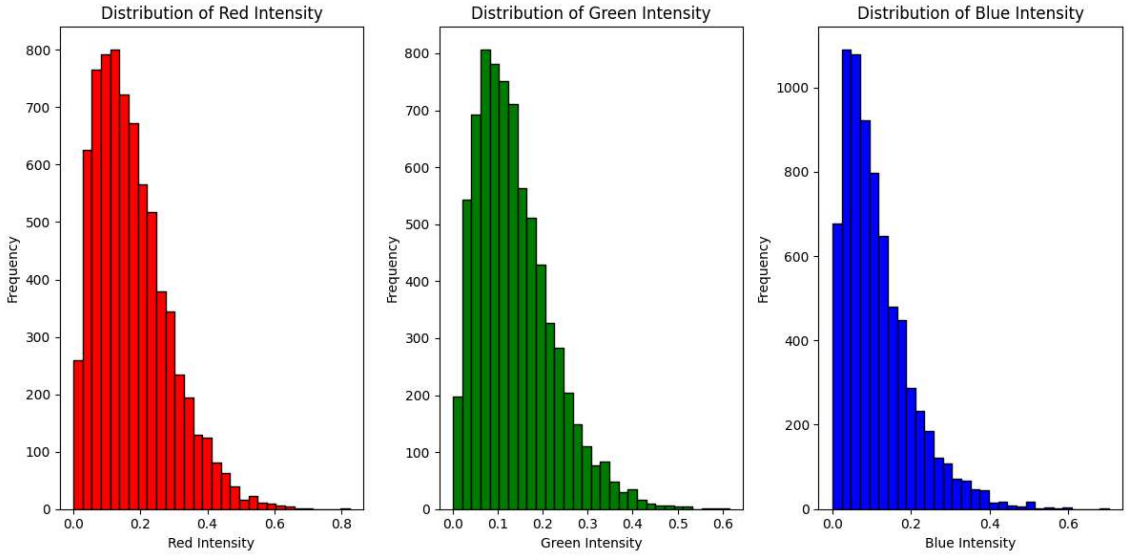

Distribution of Images Across Dataset Folders
People
Motorbike
Table
Dog
Bicycle
Cup
Boat
Chair
Bottle
Cat
Bus
Car
8.2%
6.8%
6.8%
10.8%
8.8%
7.0%
9.2%
8.8%
7.4%
10.0%
7.2%
8.9%

```python
# Analyze image sizes
image_shapes = [img.shape for img in all_images]
image_shapes_df = pd.DataFrame(image_shapes, columns=['Height', 'Width', 'Channels'])

print(image_shapes_df.describe())

# Plot the distribution of image heights and widths
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(image_shapes_df['Height'], bins=30, color='blue', edgecolor='black')
plt.xlabel('Height')
plt.ylabel('Frequency')
plt.title('Distribution of Image Heights')

plt.subplot(1, 2, 2)
plt.hist(image_shapes_df['Width'], bins=30, color='green', edgecolor='black')
plt.xlabel('Width')
plt.ylabel('Frequency')
plt.title('Distribution of Image Widths')

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

```
       Height   Width  Channels
count  7386.0  7386.0    7386.0
mean    128.0   128.0       3.0
std       0.0     0.0       0.0
min     128.0   128.0       3.0
25%     128.0   128.0       3.0
50%     128.0   128.0       3.0
75%     128.0   128.0       3.0
max     128.0   128.0       3.0
```



```python
# Analyze color distribution
def calculate_color_distribution(images):
    color_distribution = {'R': [], 'G': [], 'B': []}
    for img in images:
        R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
        color_distribution['R'].append(np.mean(R))
        color_distribution['G'].append(np.mean(G))
        color_distribution['B'].append(np.mean(B))
    return color_distribution

color_distribution = calculate_color_distribution(all_images)

plt.figure(figsize=(12, 6))
plt.subplot(1, 3, 1)
plt.hist(color_distribution['R'], bins=30, color='red', edgecolor='black')
plt.xlabel('Red Intensity')
plt.ylabel('Frequency')
plt.title('Distribution of Red Intensity')

plt.subplot(1, 3, 2)
plt.hist(color_distribution['G'], bins=30, color='green', edgecolor='black')
plt.xlabel('Green Intensity')
plt.ylabel('Frequency')
plt.title('Distribution of Green Intensity')

plt.subplot(1, 3, 3)
plt.hist(color_distribution['B'], bins=30, color='blue', edgecolor='black')
plt.xlabel('Blue Intensity')
plt.ylabel('Frequency')
plt.title('Distribution of Blue Intensity')

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

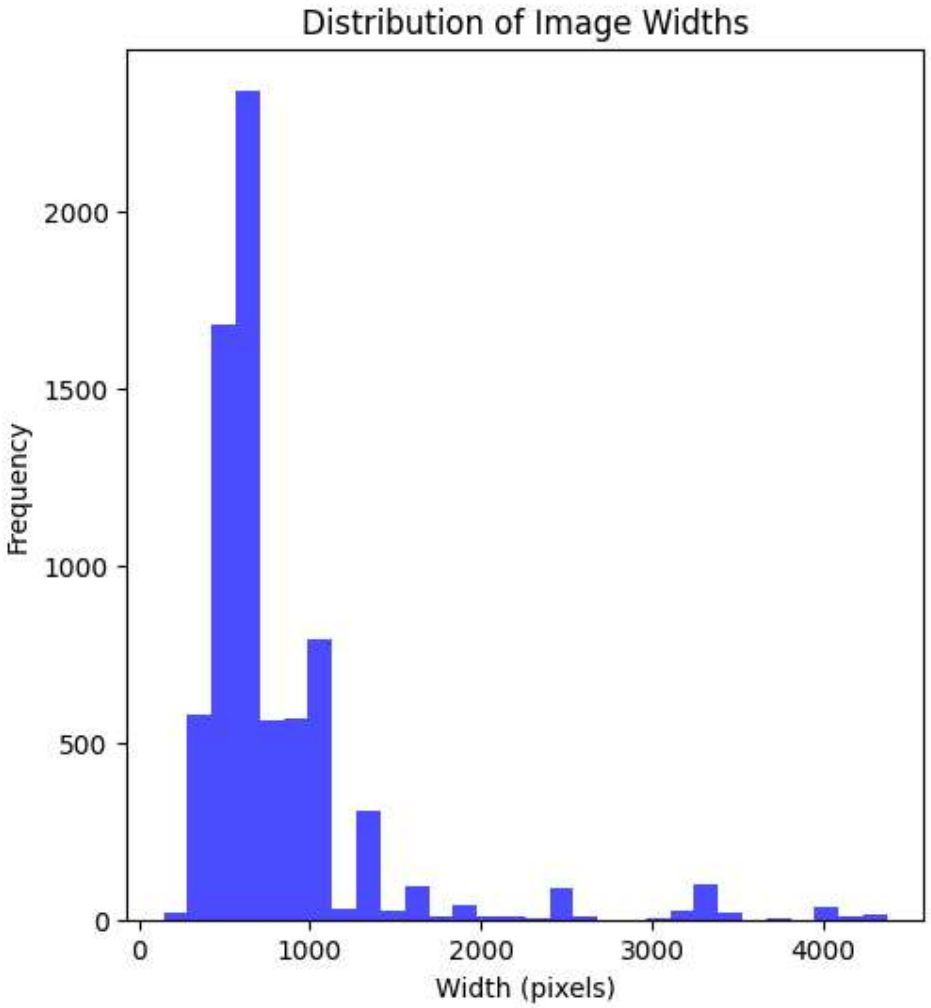


```
# Function to calculate pixel intensity distribution
def calculate_pixel_intensity_distribution(folders, target_size):
    all_pixel_values = []
    for folder in folders:
        image_files = os.listdir(folder)
        for file in image_files:
            img_path = os.path.join(folder, file)
            img = Image.open(img_path).convert('L')  # Convert to grayscale
            img = img.resize(target_size)  # Resize image to target size
            pixel_values = np.array(img).flatten()
            all_pixel_values.extend(pixel_values)
    return np.array(all_pixel_values)
```

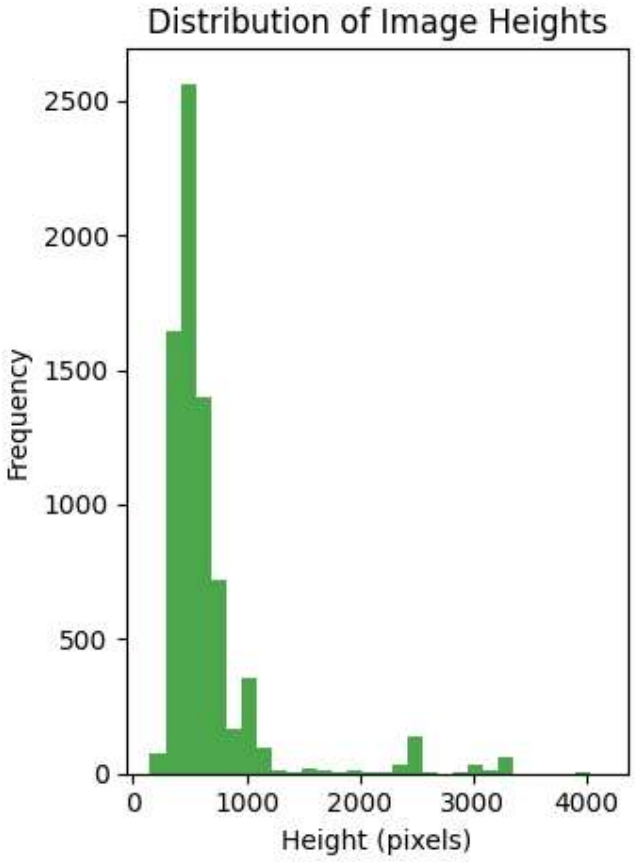
```
[28] def load_images_and_dimensions(folders):
        image_dimensions = []
        class_distribution = {os.path.basename(folder): 0 for folder in folders}
        for folder in folders:
            image_files = os.listdir(folder)
            class_distribution[os.path.basename(folder)] += len(image_files)
            for file in image_files:
                img_path = os.path.join(folder, file)
                img = Image.open(img_path).convert('RGB')
                image_dimensions.append(img.size)
        return image_dimensions, class_distribution

    # Define the target size for resizing images
    target_size = (64, 64)

    # Load image dimensions and class distribution
    image_dimensions, class_distribution = load_images_and_dimensions(folders)

    # Calculate pixel intensity distribution
    pixel_intensity_distribution = calculate_pixel_intensity_distribution(folders, target_size)
```

0s

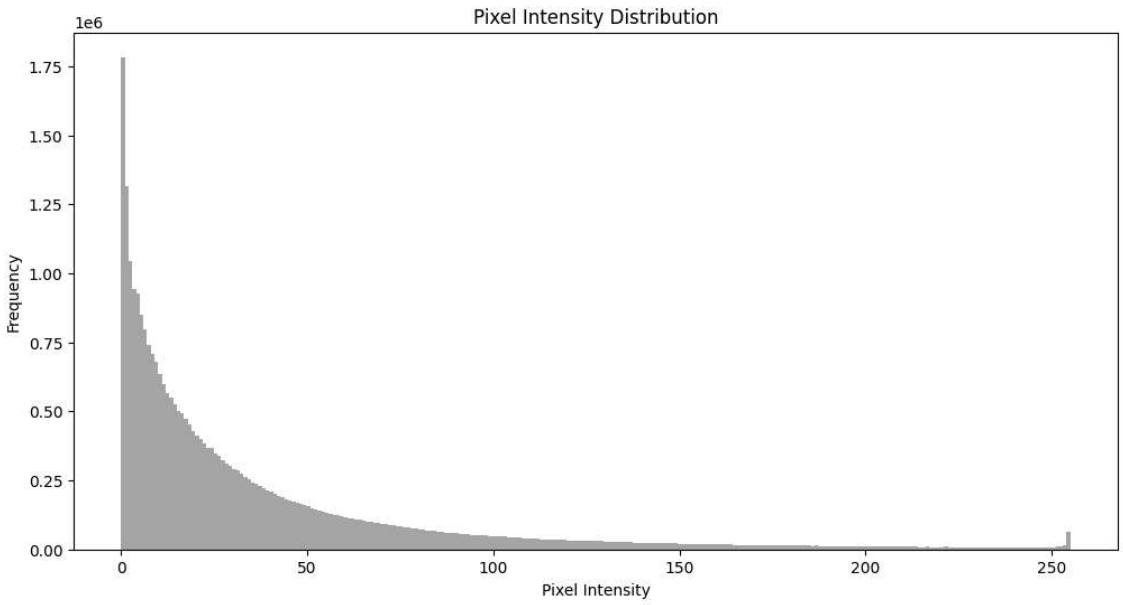
```
# Plot distribution of image dimensions
widths, heights = zip(*image_dimensions)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(widths, bins=30, color='blue', alpha=0.7)
plt.title('Distribution of Image Widths')
plt.xlabel('Width (pixels)')
plt.ylabel('Frequency')
```



```
plt.subplot(1, 2, 2)
plt.hist(heights, bins=30, color='green', alpha=0.7)
plt.title('Distribution of Image Heights')
plt.xlabel('Height (pixels)')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```




```
# Plot pixel intensity distribution
plt.figure(figsize=(12, 6))
plt.hist(pixel_intensity_distribution, bins=256, range=(0, 255), color='gray', alpha=0.7)
plt.title('Pixel Intensity Distribution')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.show()
```



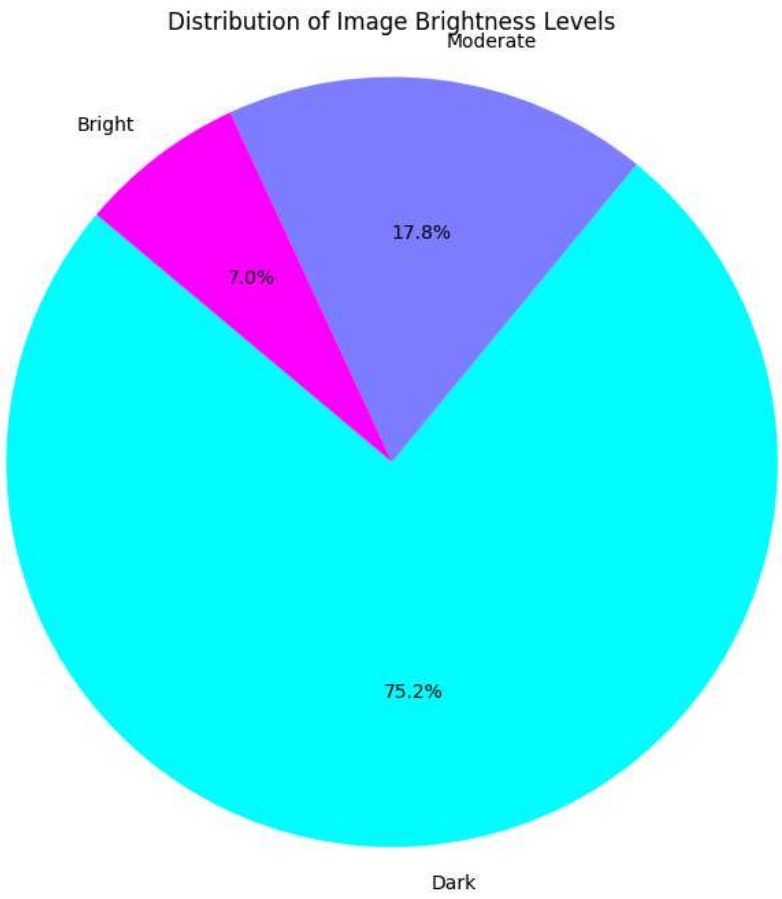

```
[32] # Function to load images and calculate average brightness
def calculate_average_brightness(folders):
    class_brightness = {os.path.basename(folder): [] for folder in folders}
    for folder in folders:
        image_files = os.listdir(folder)
        for file in image_files:
            img_path = os.path.join(folder, file)
            img = Image.open(img_path).convert('L')  # Convert to grayscale
            brightness = np.array(img).mean()
            class_brightness[os.path.basename(folder)].append(brightness)
    return class_brightness
```

```
[33] # Function to calculate aspect ratios
def calculate_aspect_ratios(image_dimensions):
    aspect_ratios = [w / h for w, h in image_dimensions]
    return aspect_ratios
```

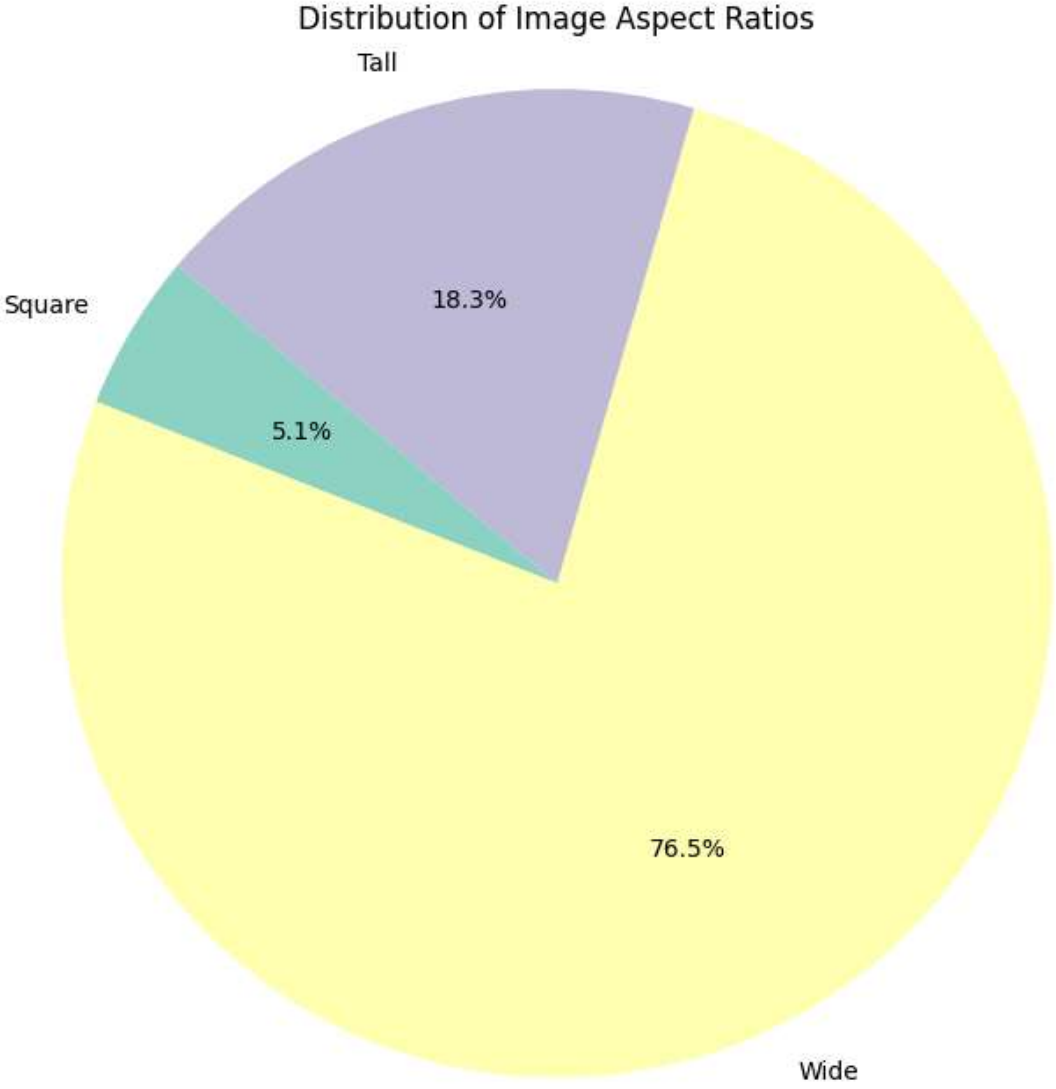
```
# Function to calculate color channel distributions
def calculate_color_channel_distributions(folders, target_size):
    all_red_values, all_green_values, all_blue_values = [], [], []
    for folder in folders:
        image_files = os.listdir(folder)
        for file in image_files:
            img_path = os.path.join(folder, file)
            img = Image.open(img_path).convert('RGB')
            img = img.resize(target_size)
            r, g, b = img.split()
            all_red_values.extend(np.array(r).flatten())
            all_green_values.extend(np.array(g).flatten())
            all_blue_values.extend(np.array(b).flatten())
    return np.array(all_red_values), np.array(all_green_values), np.array(all_blue_values)
```

```python
# Function to calculate image sizes
def calculate_image_sizes(image_dimensions):
    image_sizes = [w * h for w, h in image_dimensions]
    return image_sizes

# Function to find the most frequent images
def find_most_frequent_images(folders, target_size, top_n=5):
    image_hashes = {}
    for folder in folders:
        image_files = os.listdir(folder)
        for file in image_files:
            img_path = os.path.join(folder, file)
            img = Image.open(img_path).convert('RGB')
            img = img.resize(target_size)
            img_hash = hash(img.tobytes())
            if img_hash in image_hashes:
                image_hashes[img_hash] += 1
            else:
                image_hashes[img_hash] = 1
    most_frequent_images = sorted(image_hashes.items(), key=lambda x: x[1], reverse=True)[:top_n]
    return most_frequent_images
```

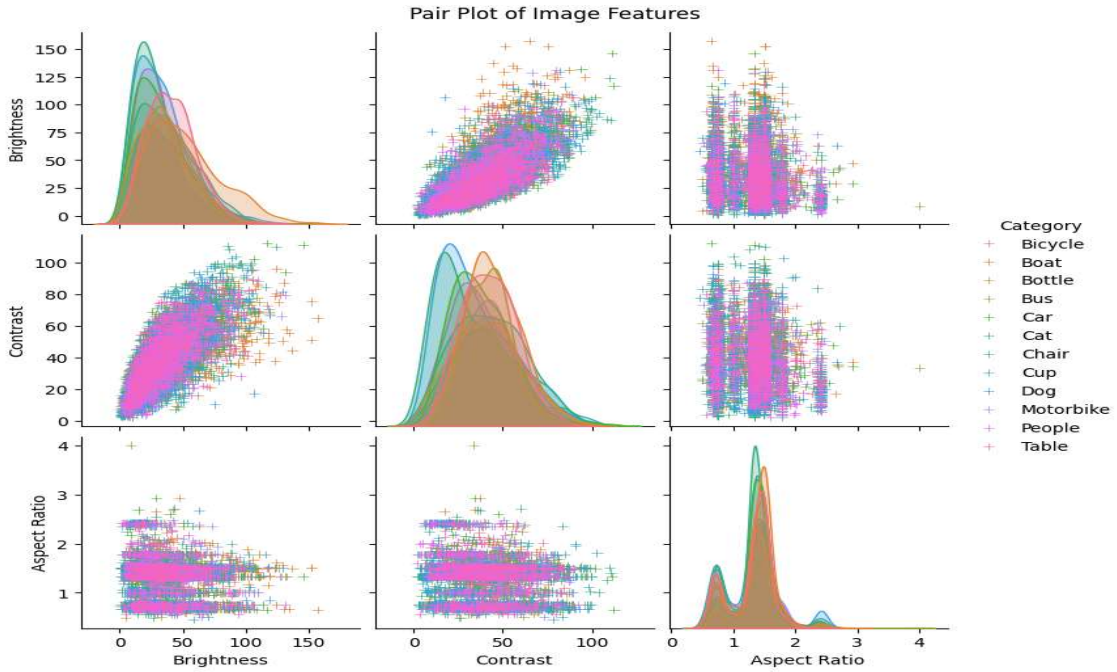
```python
    return np.mean(cv2.cvtColor(image, cv2.COLOR_BGR2GRAY))

# Function to classify brightness
def classify_brightness(brightness, thresholds=(85, 170)):
    if brightness < thresholds[0]:
        return 'Dark'
    elif brightness < thresholds[1]:
        return 'Medium'
    else:
        return 'Bright'

# Classify all images in a folder
def classify_images_by_brightness(folders):
    brightness_classes = {'Dark': 0, 'Medium': 0, 'Bright': 0}
    for folder in folders:
        for file in os.listdir(folder):
            img_path = os.path.join(folder, file)
            img = cv2.imread(img_path)
            brightness = calculate_brightness(img)
            category = classify_brightness(brightness)
            brightness_classes[category] += 1
    return brightness_classes

# Function to plot brightness distribution
def plot_brightness_distribution(brightness_classes):
    plt.figure(figsize=(8, 8))

    # Get the colors from the colormap
    colors = plt.cm.cool(np.linspace(0, 1, len(brightness_classes)))

    plt.pie(brightness_classes.values(), labels=brightness_classes.keys(), autopct='%1.1f%%', startangle=140, colors=colors)
    plt.title('Distribution of Image Brightness Levels')
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    plt.show()
# Call the function with the defined data
plot_brightness_distribution(brightness_classes)
```




```python
def classify_images_by_aspect_ratio(folders):
    aspect_ratios = {'Square': 0, 'Wide': 0, 'Tall': 0}
    for folder in folders:
        for file in os.listdir(folder):
            img_path = os.path.join(folder, file)
            img = cv2.imread(img_path)
            h, w, _ = img.shape
            ratio = w / h
            if 0.9 < ratio < 1.1:
                aspect_ratios['Square'] += 1
            elif ratio >= 1.1:
                aspect_ratios['Wide'] += 1
            else:
                aspect_ratios['Tall'] += 1
    return aspect_ratios

# Plot the pie chart
def plot_aspect_ratio_distribution(aspect_ratios):
    plt.figure(figsize=(8, 8))
    plt.pie(aspect_ratios.values(), labels=aspect_ratios.keys(), autopct='%1.1f%%', startangle=140, colors=plt.cm.Set3.colors)
    plt.title('Distribution of Image Aspect Ratios')
    plt.axis('equal')
    plt.show()

aspect_ratios = classify_images_by_aspect_ratio(folders)
plot_aspect_ratio_distribution(aspect_ratios)
```


Distribution of Image Aspect Ratios
Tall
Square
18.3%
5.1%
76.5%
Wide

```python
import cv2
import numpy as np
import pandas as pd
import os
import seaborn as sns

# Function to calculate contrast of an image
def calculate_contrast(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return gray.std()
def calculate_aspect_ratio(image):
    height, width = image.shape[:2]
    return width / height
# Function to extract features from images in folders
def extract_image_features(folders):
    features = []
    labels = []
    for folder in folders:
        for file in os.listdir(folder):
            img_path = os.path.join(folder, file)
            img = cv2.imread(img_path)
            brightness = calculate_brightness(img)
            contrast = calculate_contrast(img)
            aspect_ratio = calculate_aspect_ratio(img)
            features.append([brightness, contrast, aspect_ratio])
            labels.append(os.path.basename(folder))  # Use folder name as label
    return np.array(features), np.array(labels)

# Example usage:
features, labels = extract_image_features(folders)
df = pd.DataFrame(features, columns=['Brightness', 'Contrast', 'Aspect Ratio'])
df['Category'] = labels

sns.pairplot(df, hue='Category', diag_kind='kde', markers='+')
plt.suptitle('Pair Plot of Image Features', y=1.02)
plt.show()
```

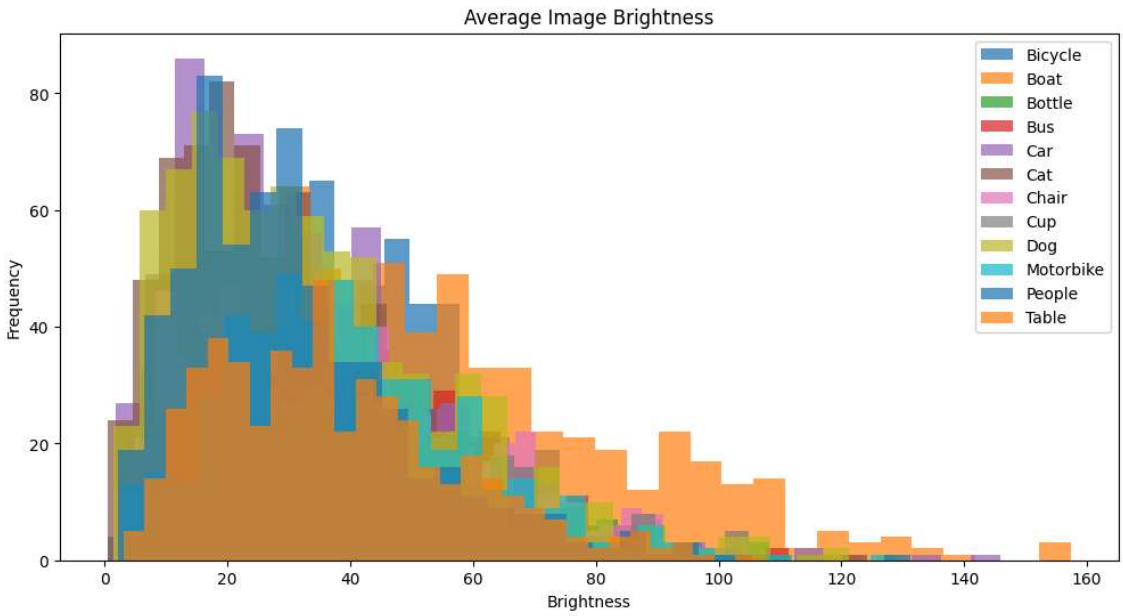


```python
# Function to load images and calculate average brightness
def calculate_average_brightness(folders):
    class_brightness = {os.path.basename(folder): [] for folder in folders}
    for folder in folders:
        image_files = os.listdir(folder)
        for file in image_files:
            img_path = os.path.join(folder, file)
            img = Image.open(img_path).convert('L')  # Convert to grayscale
            brightness = np.array(img).mean()
            class_brightness[os.path.basename(folder)].append(brightness)
    return class_brightness
```

```python
# Function to calculate aspect ratios
def calculate_aspect_ratios(image_dimensions):
    aspect_ratios = [w / h for w, h in image_dimensions]
    return aspect_ratios
```

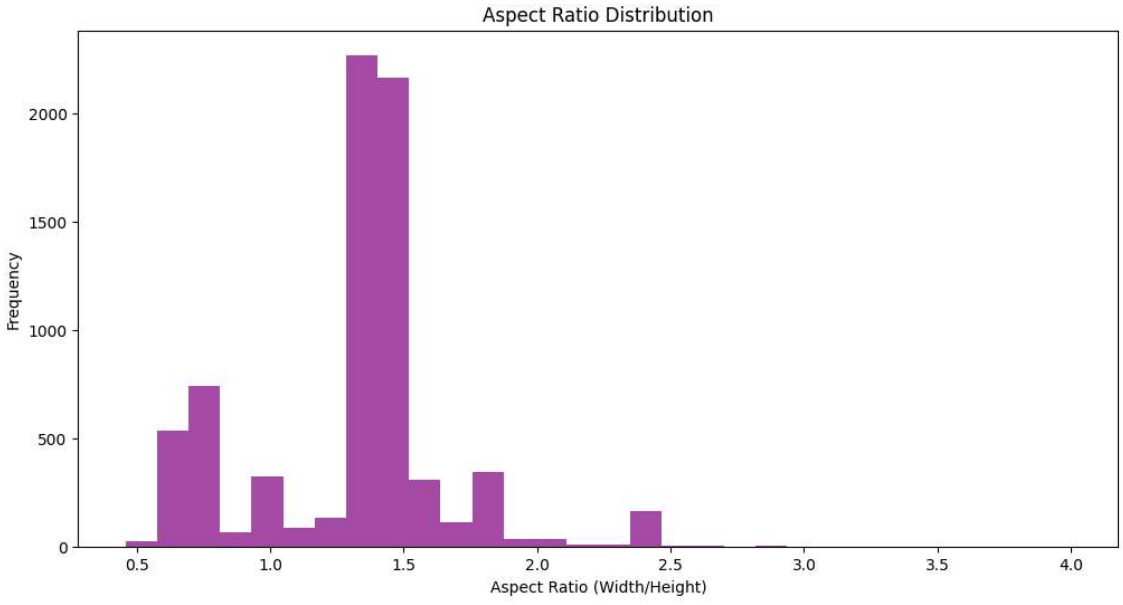
```python
# Function to calculate color channel distributions
def calculate_color_channel_distributions(folders, target_size):
    all_red_values, all_green_values, all_blue_values = [], [], []
    for folder in folders:
        image_files = os.listdir(folder)
        for file in image_files:
            img_path = os.path.join(folder, file)
            img = Image.open(img_path).convert('RGB')
            img = img.resize(target_size)
            r, g, b = img.split()
            all_red_values.extend(np.array(r).flatten())
            all_green_values.extend(np.array(g).flatten())
            all_blue_values.extend(np.array(b).flatten())
    return np.array(all_red_values), np.array(all_green_values), np.array(all_blue_values)
```

```python
# Calculate average brightness
class_brightness = calculate_average_brightness(folders)

# Average Brightness
plt.figure(figsize=(12, 6))
for folder, brightness in class_brightness.items():
    plt.hist(brightness, bins=30, alpha=0.7, label=folder)
plt.title('Average Image Brightness')
plt.xlabel('Brightness')
plt.ylabel('Frequency')
plt.legend(loc='upper right')
plt.show()
```

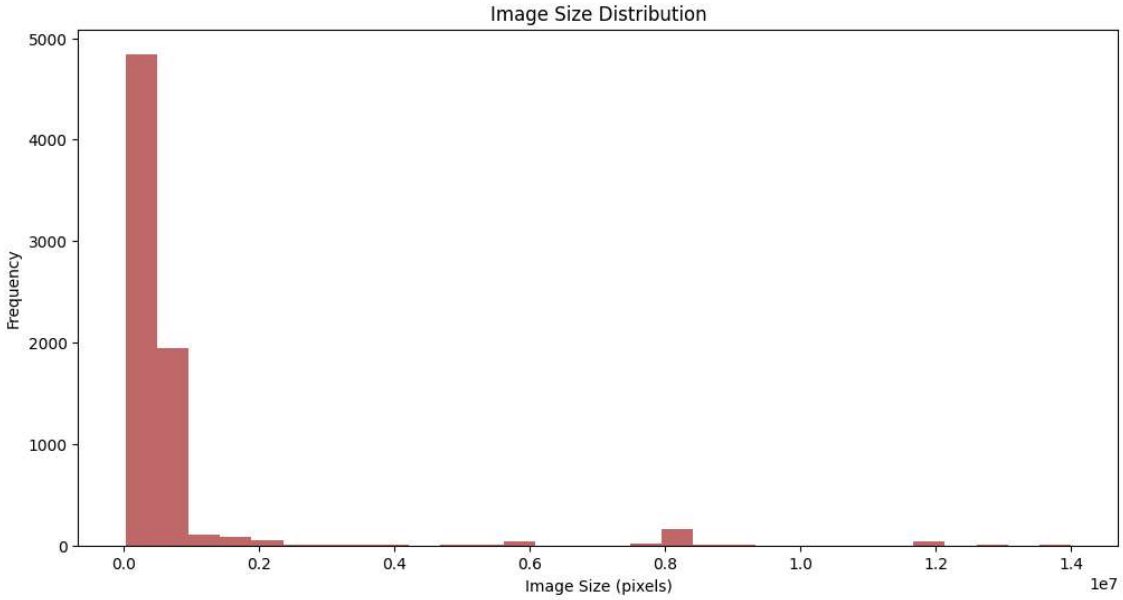


0s

```python
# Calculate aspect ratios
aspect_ratios = calculate_aspect_ratios(image_dimensions)
# Aspect Ratio Distribution
plt.figure(figsize=(12, 6))
plt.hist(aspect_ratios, bins=30, color='purple', alpha=0.7)
plt.title('Aspect Ratio Distribution')
plt.xlabel('Aspect Ratio (Width/Height)')
plt.ylabel('Frequency')
plt.show()
```



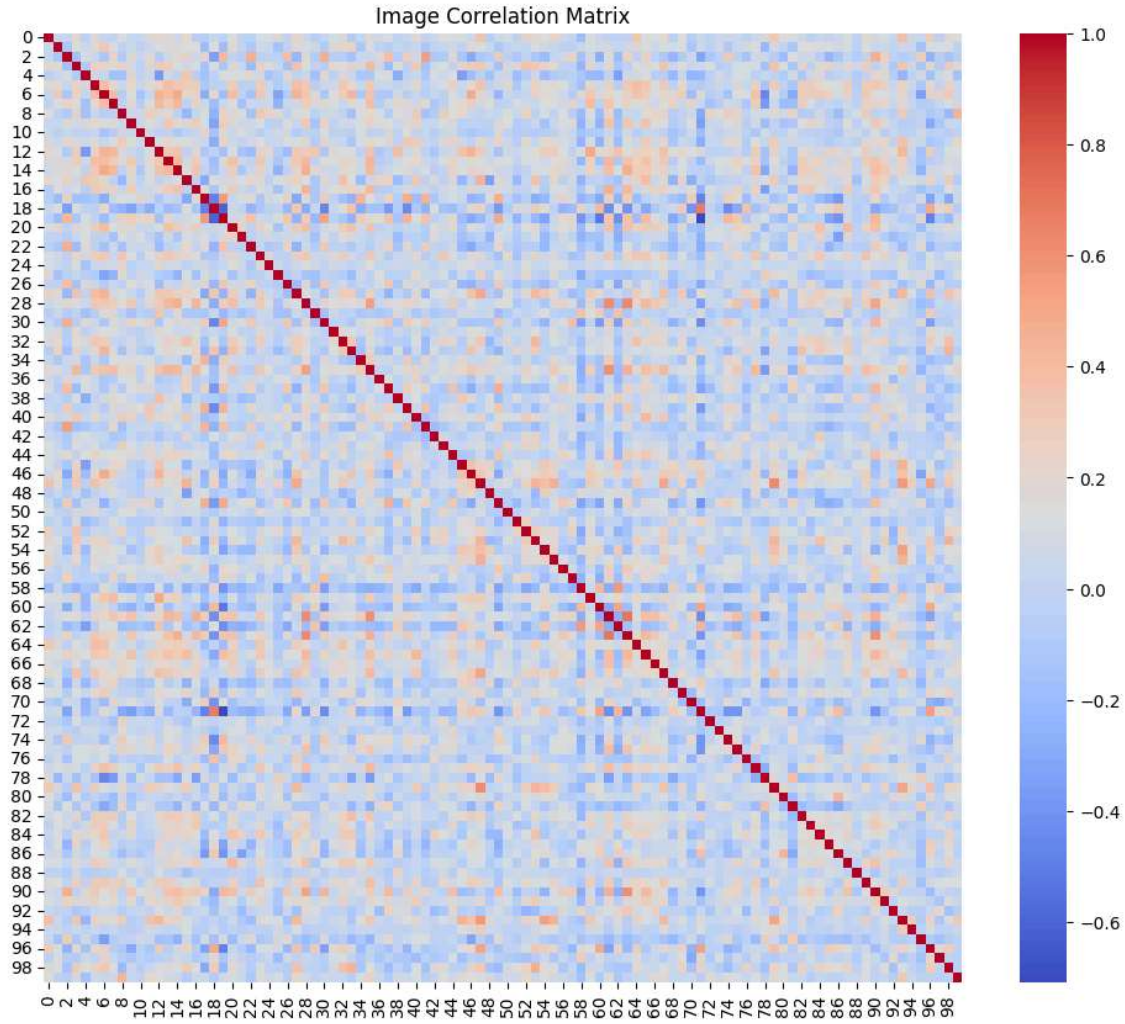
```python
# Calculate color channel distributions
red_distribution, green_distribution, blue_distribution = calculate_color_channel_distributions(folders, target_size)
# Color Channel Distribution
plt.figure(figsize=(12, 6))
plt.hist(red_distribution, bins=256, range=(0, 255), color='red', alpha=0.5, label='Red Channel')
plt.hist(green_distribution, bins=256, range=(0, 255), color='green', alpha=0.5, label='Green Channel')
plt.hist(blue_distribution, bins=256, range=(0, 255), color='blue', alpha=0.5, label='Blue Channel')
plt.title('Color Channel Distribution')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.legend(loc='upper right')
plt.show()
```



```
most_frequent_images = find_most_frequent_images(folders, target_size)

plt.figure(figsize=(12, 6))
for i, (img_hash, freq) in enumerate(most_frequent_images):
    plt.subplot(1, len(most_frequent_images), i + 1)
    for folder in folders:
        image_files = os.listdir(folder)
        for file in image_files:
            img_path = os.path.join(folder, file)
            img = Image.open(img_path).convert('RGB')
            img = img.resize(target_size)
            if hash(img.tobytes()) == img_hash:
                plt.imshow(img)
                plt.title(f'Freq: {freq}')
                plt.axis('off')
                break
plt.suptitle('Most Frequent Images')
plt.show()
```

Most Frequent Images

```
import matplotlib.pyplot as plt
import seaborn as sns
# Image Correlation Matrix
subset_images = all_images[:100]
flattened_images = subset_images.reshape(subset_images.shape[0], -1)
correlation_matrix = np.corrcoef(flattened_images)
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, cmap='coolwarm')
plt.title('Image Correlation Matrix')
plt.show()
```



Activity 2: Computer Vision Techniques for Image Enhancement

Key Techniques for Image Enhancement:

- Histogram Equalization
- Gamma Correction
- Filtering Techniques
- Image Transformation

Integrating EDA with Computer Vision:

- The insights gained from EDA inform the selection and tuning of computer vision techniques. For instance, if EDA reveals a consistent lack of contrast in images, histogram equalisation may be prioritised. Conversely, if noise is identified as a major issue, smoothing filters or noise reduction algorithms would be emphasised.

```python
import cv2
import matplotlib.pyplot as plt

# Function to apply histogram equalization
def histogram_equalization(img):
    img_yuv = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img_yuv[:,:,0] = cv2.equalizeHist(img_yuv[:,:,0])
    img_output = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2RGB)
    return img_output

# Enhance some test images
enhanced_images = np.array([histogram_equalization((img * 255).astype(np.uint8)) for img in X_test])

# Visualize original vs enhanced images
def visualize_enhanced_images(original, enhanced, num_images=5):
    plt.figure(figsize=(15, 10))
    for i in range(num_images):
        plt.subplot(2, num_images, i + 1)
        plt.imshow(original[i])
        plt.title('Original')
        plt.axis('off')

        plt.subplot(2, num_images, i + 1 + num_images)
        plt.imshow(enhanced[i])
        plt.title('Enhanced')
        plt.axis('off')
    plt.show()

visualize_enhanced_images(X_test, enhanced_images)
```

Final Output:

Conclusion

The project aimed to improve dark images using Exploratory Data Analysis (EDA) and computer vision techniques. The researchers used histograms and inspections to understand the dataset, revealing darkness, contrast issues, and variations in image quality. They used basic techniques like histogram equalisation and gamma correction to enhance visibility and contrast. The project also highlighted challenges, such as noise amplification in overly dark images. Future work could focus on noise reduction techniques or machine learning-based approaches for automated image enhancement. Experimenting with deep learning models like GANs could offer more advanced solutions for enhancing images with complex degradation patterns. The project's success demonstrates how EDA and computer vision techniques can significantly enhance image quality, particularly in low-light conditions.