Data Loading

Step 1: Setup Kaggle API in Colab

- Move the file to the appropriate directory:
- *Step 2: Download the Dataset**

-Now that you've set up Kaggle access, you can download the dataset directly.

Step 3: Extract the Dataset

-The downloaded file will be a zip file. You can unzip it in Colab with the following command:

```
# Install necessary packages and setup Kaggle API
!pip install kaggle

!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Download and extract the dataset
!kaggle datasets download -d gauravduttakiit/wheat-growth-stage-challenge

import zipfile
with zipfile.ZipFile('wheat-growth-stage-challenge.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/wheat-growth')

# Check extracted files
!ls /content/wheat-growth
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2024.12.14)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.3.0)
Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.11/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->kaggle) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->kaggle) (3.10)
mv: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
Dataset URL: https://www.kaggle.com/datasets/gauravduttakiit/wheat-growth-stage-challenge
License(s): unknown
Downloading wheat-growth-stage-challenge.zip to /content
 99% 1.06G/1.07G [00:14<00:00, 91.1MB/s]
100% 1.07G/1.07G [00:14<00:00, 79.5MB/s]
sol  submission.csv  test  train
```

```
BATCH_SIZE = 32 #@param {type:"slider",min:32,max:1024,step:32}
EPOCHS = 25
IMG_SIZE = 500
INPUT_SHAPE=(IMG_SIZE,IMG_SIZE,3)
OUTPUT_SHAPE = 7
NUM_IMAGES = 1000 #@param {type:"slider",min:1000,max:10695,step:10
```

BATCH_SIZE: [32

NUM_IMAGES: [1000

Data Understanding

 # 1. Basic Data Inspection

First, let's confirm the number of images and their dimensions in the training dataset. This will give us the number of images, a sample image's size (dimensions), and color mode (RGB or grayscale).link text

```
import os
from collections import defaultdict
from PIL import Image
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from collections import Counter
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
```

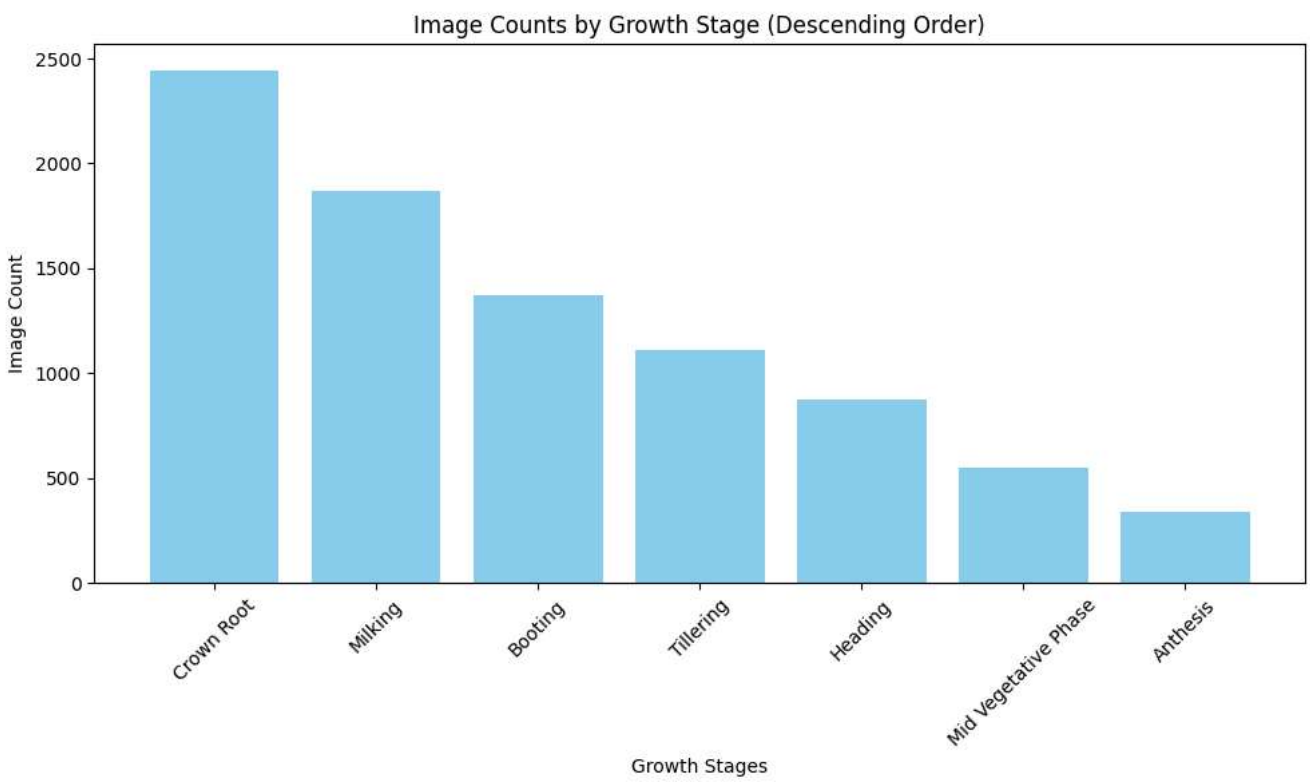
```
from sklearn.metrics import silhouette_score


# Define dataset path
base_train_dir = '/content/wheat-growth/train'


# Count images in each growth stage folder
growth_stages = os.listdir(base_train_dir)
image_counts = {stage: len(os.listdir(os.path.join(base_train_dir, stage))) for stage in growth_stages}
print("Image Counts by Growth Stage: ", image_counts)
```

```
Image Counts by Growth Stage:  {'Booting': 1369, 'Anthesis': 337, 'Tillering': 1111, 'Mid Vegetative Phase': 551, 'Crown Root': 244
```

1. Basic Data Inspection

First, let's confirm the number of images and their dimensions in the training dataset. This will give us the number of images, a sample image's size (dimensions), and color mode (RGB or grayscale).

```
# Visualize image counts
sorted_counts = sorted(image_counts.items(), key=lambda x: x[1], reverse=True)
stages, counts = zip(*sorted_counts)
plt.figure(figsize=(10, 6))
plt.bar(stages, counts, color='skyblue')
plt.xlabel("Growth Stages")
plt.ylabel("Image Count")
plt.title("Image Counts by Growth Stage (Descending Order)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
# Display image dimensions and channels
train_images = []
labels = []
dimensions = []
for stage in growth_stages:
    stage_dir = os.path.join(base_train_dir, stage)
    for filename in os.listdir(stage_dir):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(stage_dir, filename)
            train_images.append(img_path)
            labels.append(stage)
            with Image.open(img_path) as img:
                dimensions.append(img.size)

if train_images:
```

```
sample_image_path = train_images[0]
with Image.open(sample_image_path) as img:
    print(f"Sample Image Size: {img.size}, Mode: {img.mode}")
```

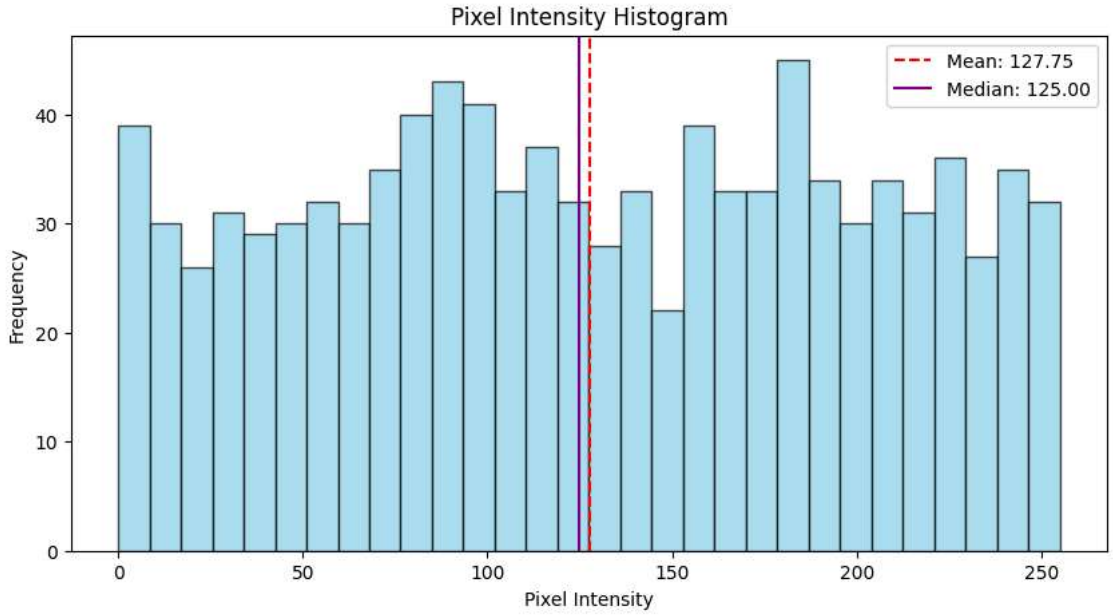
```
Sample Image Size: (512, 512), Mode: RGB
```

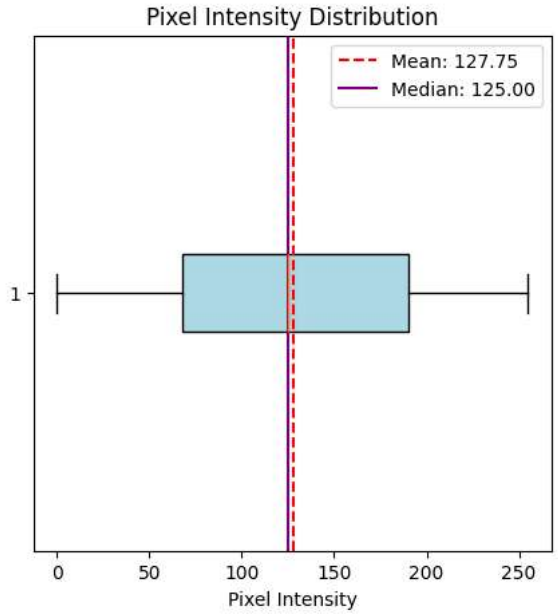
2. Statistical Analysis of Pixel Intensities

Now, let's compute mean, median, and standard deviation of pixel intensities. To handle potentially large data, we'll process a sample of images.

```
import numpy as np
import matplotlib.pyplot as plt

# Example pixel intensity array (replace with your actual image data)
pixel_intensities = np.random.randint(0, 256, size=(1000,))  # Random example data

# Calculate statistics
mean_intensity = np.mean(pixel_intensities)
median_intensity = np.median(pixel_intensities)
std_dev_intensity = np.std(pixel_intensities)
min_intensity = np.min(pixel_intensities)
max_intensity = np.max(pixel_intensities)

# Print statistics
print(f"Mean Pixel Intensity: {mean_intensity:.2f}")
print(f"Median Pixel Intensity: {median_intensity:.2f}")
print(f"Standard Deviation: {std_dev_intensity:.2f}")
print(f"Min Pixel Intensity: {min_intensity:.2f}")
print(f"Max Pixel Intensity: {max_intensity:.2f}")

# Plot histogram
plt.figure(figsize=(10, 5))
plt.hist(pixel_intensities, bins=30, color='skyblue', alpha=0.7, edgecolor='black')
plt.axvline(mean_intensity, color='red', linestyle='--', label=f"Mean: {mean_intensity:.2f}")
plt.axvline(median_intensity, color='purple', linestyle='-', label=f"Median: {median_intensity:.2f}")
plt.title("Pixel Intensity Histogram")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.legend()
plt.show()

# Plot box plot
plt.figure(figsize=(5, 5))
plt.boxplot(pixel_intensities, vert=False, patch_artist=True, boxprops=dict(facecolor="lightblue"))
plt.axvline(mean_intensity, color='red', linestyle='--', label=f"Mean: {mean_intensity:.2f}")
plt.axvline(median_intensity, color='purple', linestyle='-', label=f"Median: {median_intensity:.2f}")
plt.title("Pixel Intensity Distribution")
plt.xlabel("Pixel Intensity")
plt.legend()
plt.show()
```

```
Mean Pixel Intensity: 127.75
Median Pixel Intensity: 125.00
Standard Deviation: 72.72
Min Pixel Intensity: 0.00
Max Pixel Intensity: 255.00
```

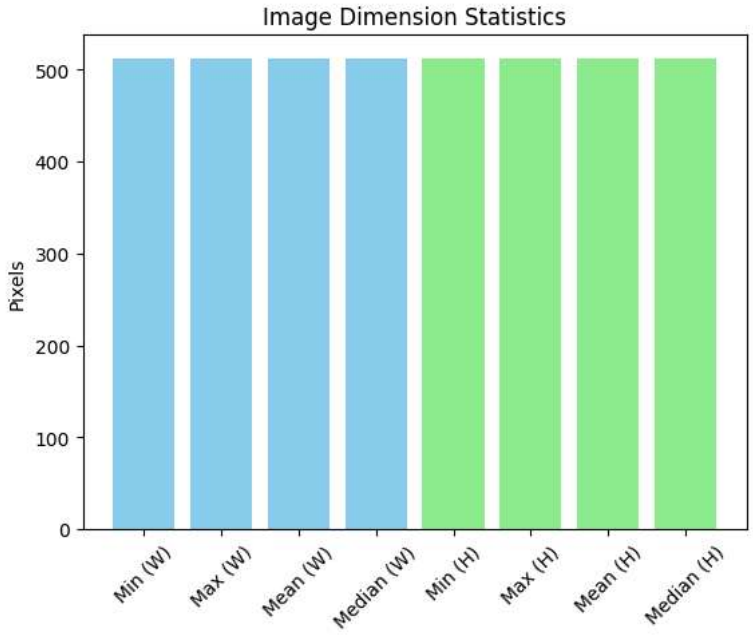




4. Dimension Analysis

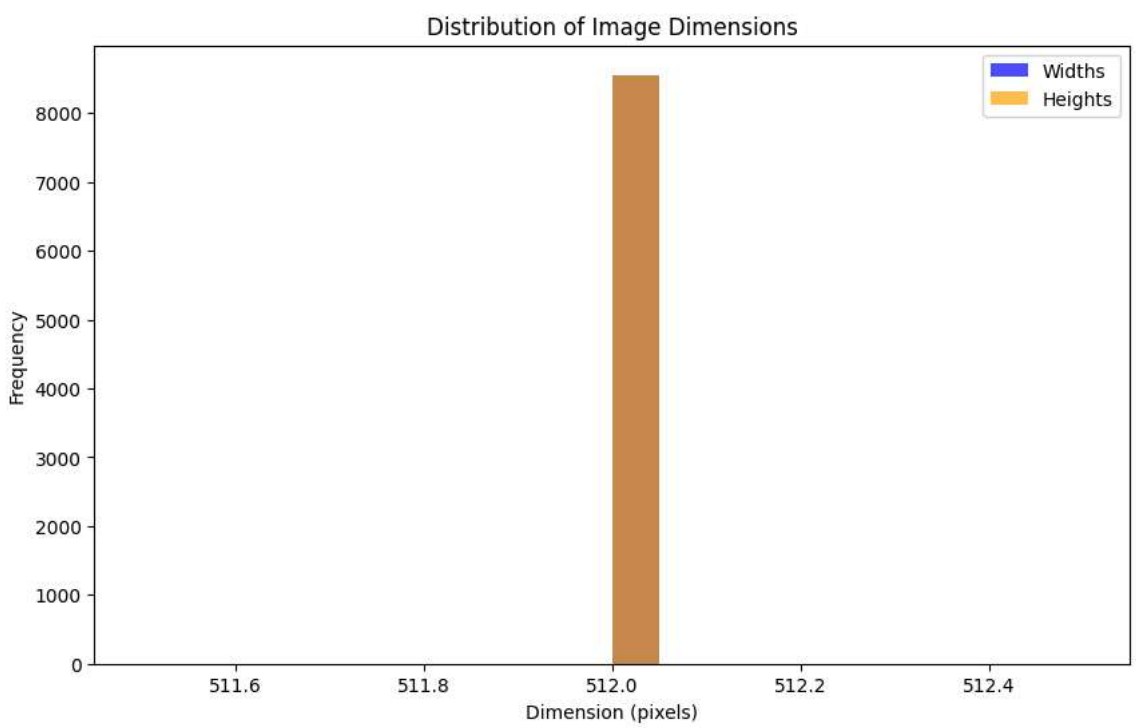
We'll collect the dimensions of each image and summarize them with statistics like minimum, maximum, mean, and mode. This way, we'll know if most images have similar dimensions or if resizing is necessary for model consistency.

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Analyze dimension statistics
dims = np.array(dimensions)
widths, heights = dims[:, 0], dims[:, 1]
plt.bar(["Min (W)", "Max (W)", "Mean (W)", "Median (W)", "Min (H)", "Max (H)", "Mean (H)", "Median (H)"],
        [widths.min(), widths.max(), widths.mean(), np.median(widths), heights.min(), heights.max(), heights.mean(), np.median(heights)],
        color=['skyblue'] * 4 + ['lightgreen'] * 4)
plt.title('Image Dimension Statistics')
plt.ylabel('Pixels')
plt.xticks(rotation=45)
plt.show()
print("Image Dimension Statistics:")
print(f" - Width: Min={widths.min()}, Max={widths.max()}, Mean={widths.mean():.2f}, Median={np.median(widths):.2f}")
print(f" - Height: Min={heights.min()}, Max={heights.max()}, Mean={heights.mean():.2f}, Median={np.median(heights):.2f}")
```




```
Image Dimension Statistics:
 - Width: Min=512, Max=512, Mean=512.00, Median=512.00
 - Height: Min=512, Max=512, Mean=512.00, Median=512.00
```

```
# Visualize image dimensions
plt.figure(figsize=(10, 6))
plt.hist(widths, bins=20, alpha=0.7, label='Widths', color='blue')
plt.hist(heights, bins=20, alpha=0.7, label='Heights', color='orange')
plt.xlabel("Dimension (pixels)")
plt.ylabel("Frequency")
plt.title("Distribution of Image Dimensions")
plt.legend()
plt.show()
```



Visualize sample images form all the stages

```
# Visualize sample images
def display_sample_images(dataset_path, num_images=5):
    sample_images = np.random.choice(train_images, num_images, replace=False)
    plt.figure(figsize=(15, 5))
    for i, img_path in enumerate(sample_images):
        img = Image.open(img_path)
        plt.subplot(1, num_images, i + 1)
        plt.imshow(img)
        plt.axis('off')
```

```
            plt.title(os.path.basename(os.path.dirname(img_path)))
        plt.show()

# Call the function to display images
display_sample_images(base_train_dir)
```

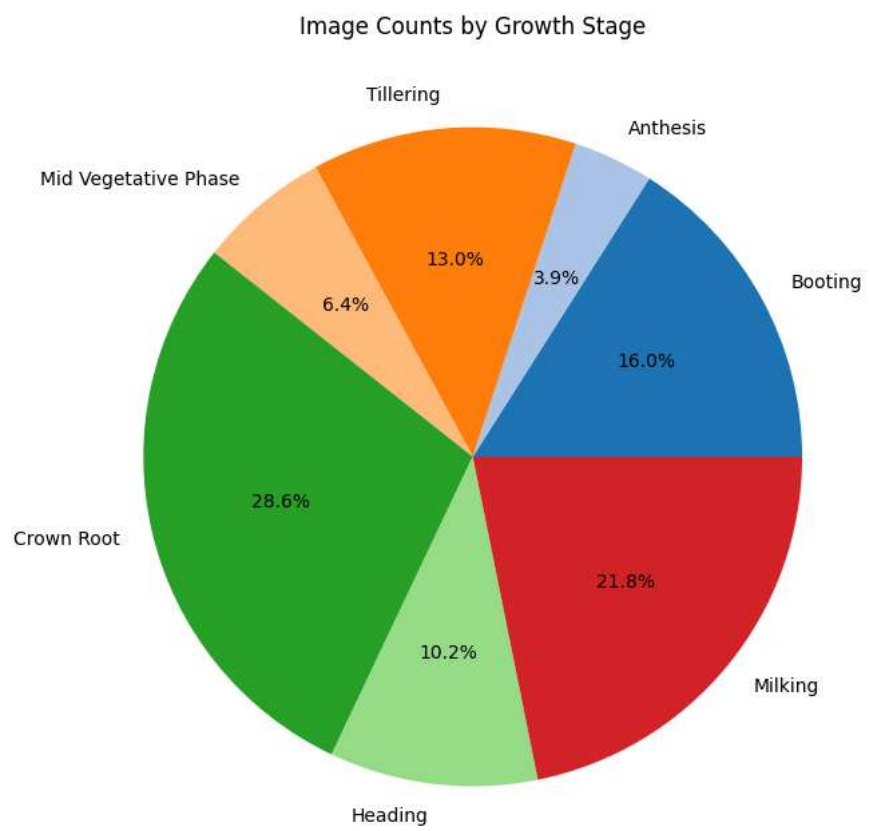
3. Class Distribution

If you have labels (e.g., in a CSV file), we can load and examine the distribution across growth stages.

```
# Label distribution
labels_df = pd.read_csv('/content/wheat-growth/submission.csv')
print("Label Distribution:")
print(labels_df['UID'].value_counts())
```

```
Label Distribution:
UID
F3LbWkZq    1
bUkIXag3    1
TibrOIR4    1
Y7PKHbyS    1
FTWghSZe    1
           ..
zlcCwhf4    1
3Xzl1hKm    1
U9tvELuz    1
EW3J2bh8    1
Olx7gipM    1
Name: count, Length: 3558, dtype: int64
```

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import os

# Image counts by growth stage
growth_stages = {stage: len(os.listdir(os.path.join(base_train_dir, stage))) for stage in os.listdir(base_train_dir)}

# Increase pie chart size and display
plt.figure(figsize=(10, 8))  # Set the size of the chart
plt.pie(growth_stages.values(), labels=growth_stages.keys(), autopct='%1.1f%%', colors=plt.cm.tab20.colors)
plt.title('Image Counts by Growth Stage')
plt.show()

# Print image counts by growth stage
print("Image Counts by Growth Stage:", growth_stages)
```




```
Image Counts by Growth Stage: {'Booting': 1369, 'Anthesis': 337, 'Tillering': 1111, 'Mid Vegetative Phase': 551, 'Crown Root': 2446
```

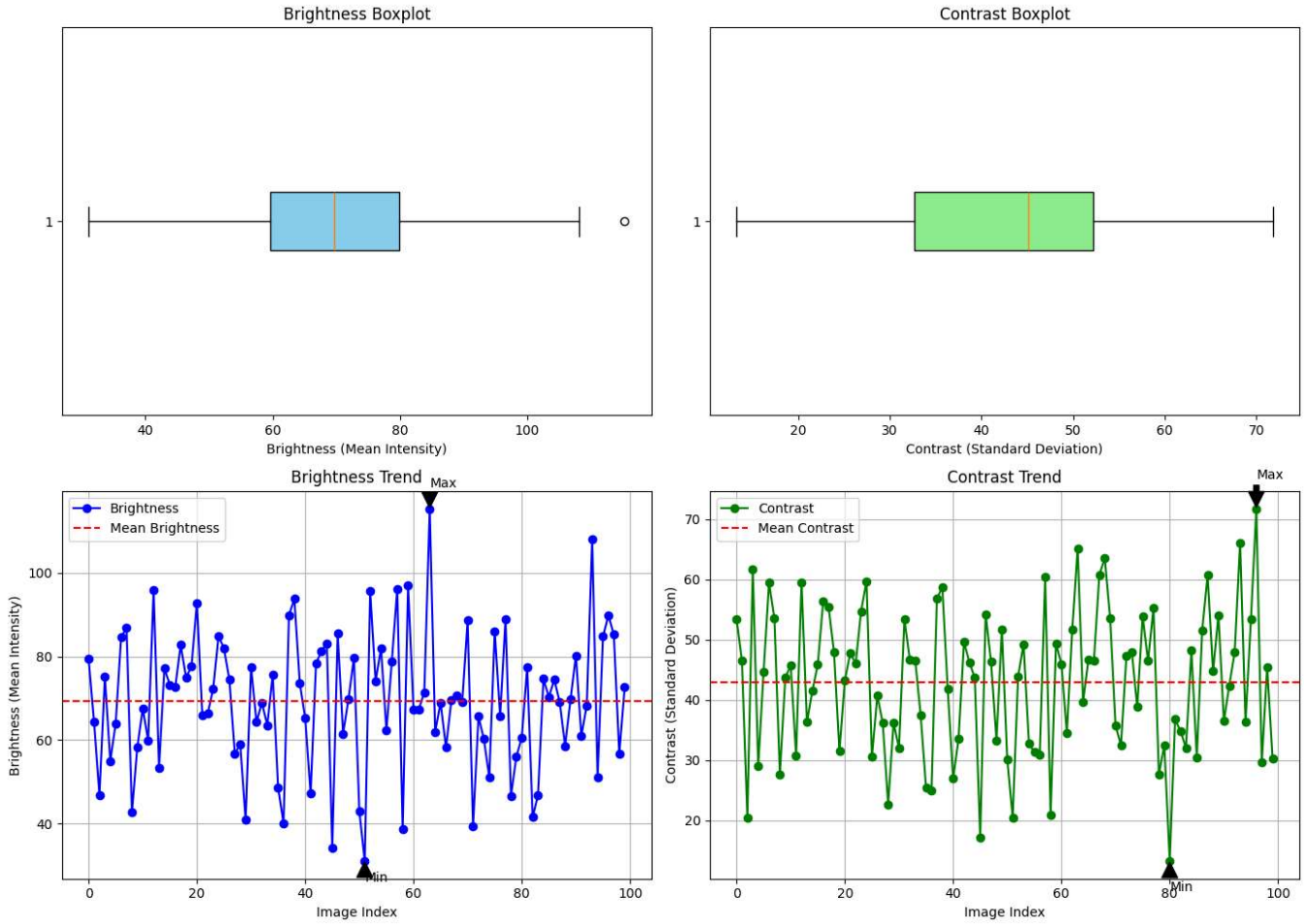
 # 4. Contrast and Brightness analysis

We'll collect Contrast and Brightness of each image and summarize them with statistics like minimum, maximum, mean, and mode. This way, we'll know if most images have similar Contrast and Brightness or if resizing is necessary for model consistency.

```
import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
# Assuming train_images and base_train_dir are already defined
# Collect dimensions of all images
dimensions = []
for img_name in train_images:
    img_path = os.path.join(base_train_dir, img_name)
    with Image.open(img_path) as img:
        dimensions.append(img.size)  # (width, height)

# Convert dimensions list to numpy array for statistical analysis
dimensions_array = np.array(dimensions)
widths = dimensions_array[:, 0]
heights = dimensions_array[:, 1]
# Assuming train_images and base_train_dir are already defined
# Brightness (mean intensity) for a sample of images
sample_images = train_images[:100]  # Use a subset for performance
brightness = [
    np.array(Image.open(os.path.join(base_train_dir, img))).mean() for img in sample_images
]

# Contrast (standard deviation of intensity) for the sample images
contrast = [
    np.array(Image.open(os.path.join(base_train_dir, img))).std() for img in sample_images
]

# Brightness and Contrast Statistics
print("Brightness Statistics:")
print(f" - Min brightness: {np.min(brightness):.2f}")
print(f" - Max brightness: {np.max(brightness):.2f}")
print(f" - Mean brightness: {np.mean(brightness):.2f}")
print(f" - Median brightness: {np.median(brightness):.2f}")
print()

print("Contrast Statistics:")
print(f" - Min contrast: {np.min(contrast):.2f}")
print(f" - Max contrast: {np.max(contrast):.2f}")
```

```
print(f" - Mean contrast: {np.mean(contrast):.2f}")
print(f" - Median contrast: {np.median(contrast):.2f}")

# Graphical Representation
fig, ax = plt.subplots(2, 2, figsize=(14, 10))

# Brightness Boxplot
ax[0, 0].boxplot(brightness, vert=False, patch_artist=True, boxprops=dict(facecolor='skyblue'))
ax[0, 0].set_title("Brightness Boxplot")
ax[0, 0].set_xlabel("Brightness (Mean Intensity)")

# Contrast Boxplot
ax[0, 1].boxplot(contrast, vert=False, patch_artist=True, boxprops=dict(facecolor='lightgreen'))
ax[0, 1].set_title("Contrast Boxplot")
ax[0, 1].set_xlabel("Contrast (Standard Deviation)")

# Brightness Line Graph
ax[1, 0].plot(range(len(brightness)), brightness, color='blue', marker='o', label='Brightness')
ax[1, 0].axhline(np.mean(brightness), color='red', linestyle='--', label='Mean Brightness')
ax[1, 0].set_title("Brightness Trend")
ax[1, 0].set_xlabel("Image Index")
ax[1, 0].set_ylabel("Brightness (Mean Intensity)")
ax[1, 0].legend()
ax[1, 0].grid(True)
# Annotating peaks
max_brightness_idx = np.argmax(brightness)
min_brightness_idx = np.argmin(brightness)
ax[1, 0].annotate('Max', xy=(max_brightness_idx, brightness[max_brightness_idx]),
                  xytext=(max_brightness_idx, brightness[max_brightness_idx] + 5),
                  arrowprops=dict(facecolor='black', shrink=0.05))
ax[1, 0].annotate('Min', xy=(min_brightness_idx, brightness[min_brightness_idx]),
                  xytext=(min_brightness_idx, brightness[min_brightness_idx] - 5),
                  arrowprops=dict(facecolor='black', shrink=0.05))

# Contrast Line Graph
ax[1, 1].plot(range(len(contrast)), contrast, color='green', marker='o', label='Contrast')
ax[1, 1].axhline(np.mean(contrast), color='red', linestyle='--', label='Mean Contrast')
ax[1, 1].set_title("Contrast Trend")
ax[1, 1].set_xlabel("Image Index")
ax[1, 1].set_ylabel("Contrast (Standard Deviation)")
ax[1, 1].legend()
ax[1, 1].grid(True)
# Annotating peaks
max_contrast_idx = np.argmax(contrast)
min_contrast_idx = np.argmin(contrast)
ax[1, 1].annotate('Max', xy=(max_contrast_idx, contrast[max_contrast_idx]),
                  xytext=(max_contrast_idx, contrast[max_contrast_idx] + 5),
                  arrowprops=dict(facecolor='black', shrink=0.05))
ax[1, 1].annotate('Min', xy=(min_contrast_idx, contrast[min_contrast_idx]),
                  xytext=(min_contrast_idx, contrast[min_contrast_idx] - 5),
                  arrowprops=dict(facecolor='black', shrink=0.05))

# Adjust layout for better readability
plt.tight_layout()
plt.show()
```

```
Brightness Statistics:
 - Min brightness: 31.11
 - Max brightness: 115.28
 - Mean brightness: 69.37
 - Median brightness: 69.61

Contrast Statistics:
 - Min contrast: 13.27
 - Max contrast: 71.77
 - Mean contrast: 42.99
 - Median contrast: 45.12
```



t-SNE (t-distributed Stochastic Neighbor Embedding)

It is an unsupervised non-linear dimensionality reduction technique for data exploration and visualizing high-dimensional data.

Above demonstration specify least and maximum t-SNE

Tried applying t-SNE for all the labels in training dataset code is given below

```
# t-SNE Analysis with Grouped Labels
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
from sklearn.manifold import TSNE
import numpy as np
```

```
# Load pre-trained VGG16 model for feature extraction
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False
```

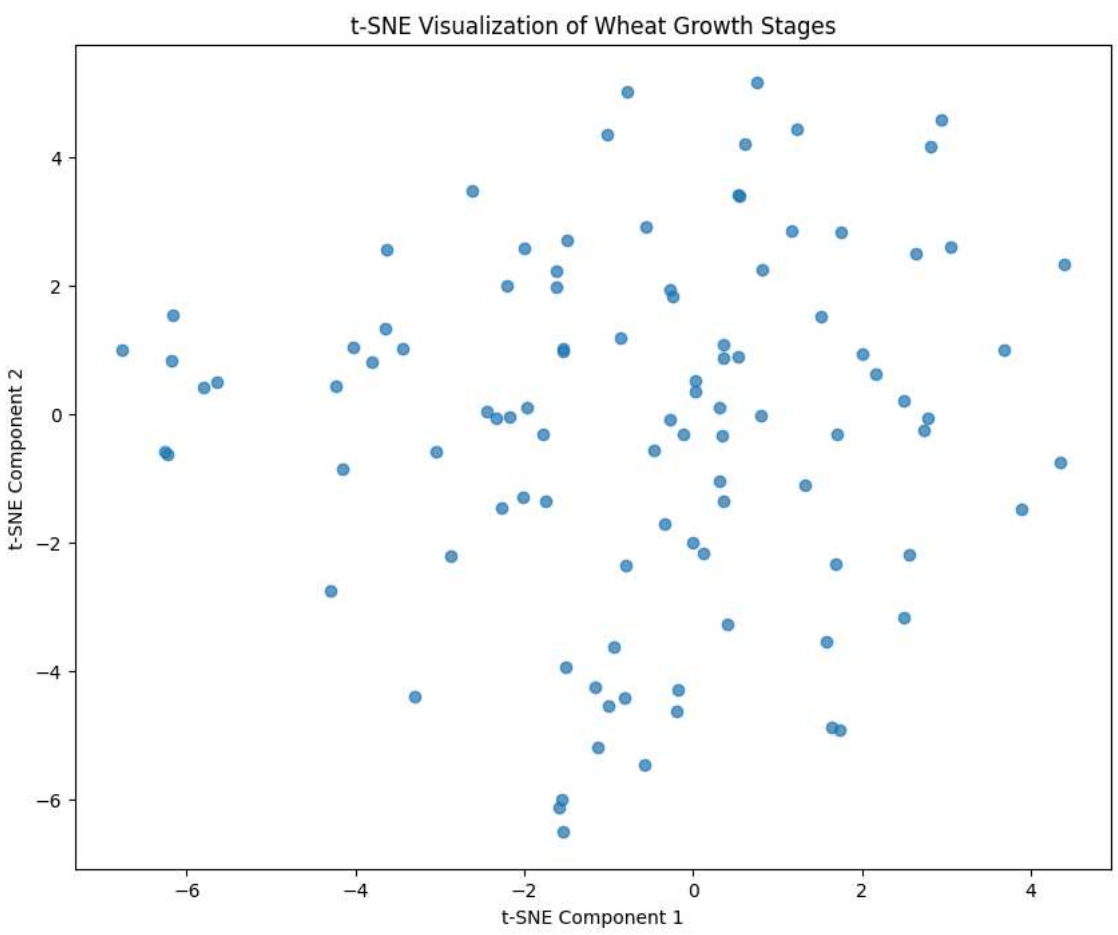
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_no
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
def extract_features(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)
    return base_model.predict(img_array).flatten()


# Extract features for t-SNE
features = np.array([extract_features(img) for img in train_images[:100]])  # Use a subset for performance
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
```

```
tsne = TSNE(n_components=2, random_state=42)
tsne_features = tsne.fit_transform(features)


# Plot t-SNE results
plt.figure(figsize=(10, 8))
plt.scatter(tsne_features[:, 0], tsne_features[:, 1], alpha=0.7)
```

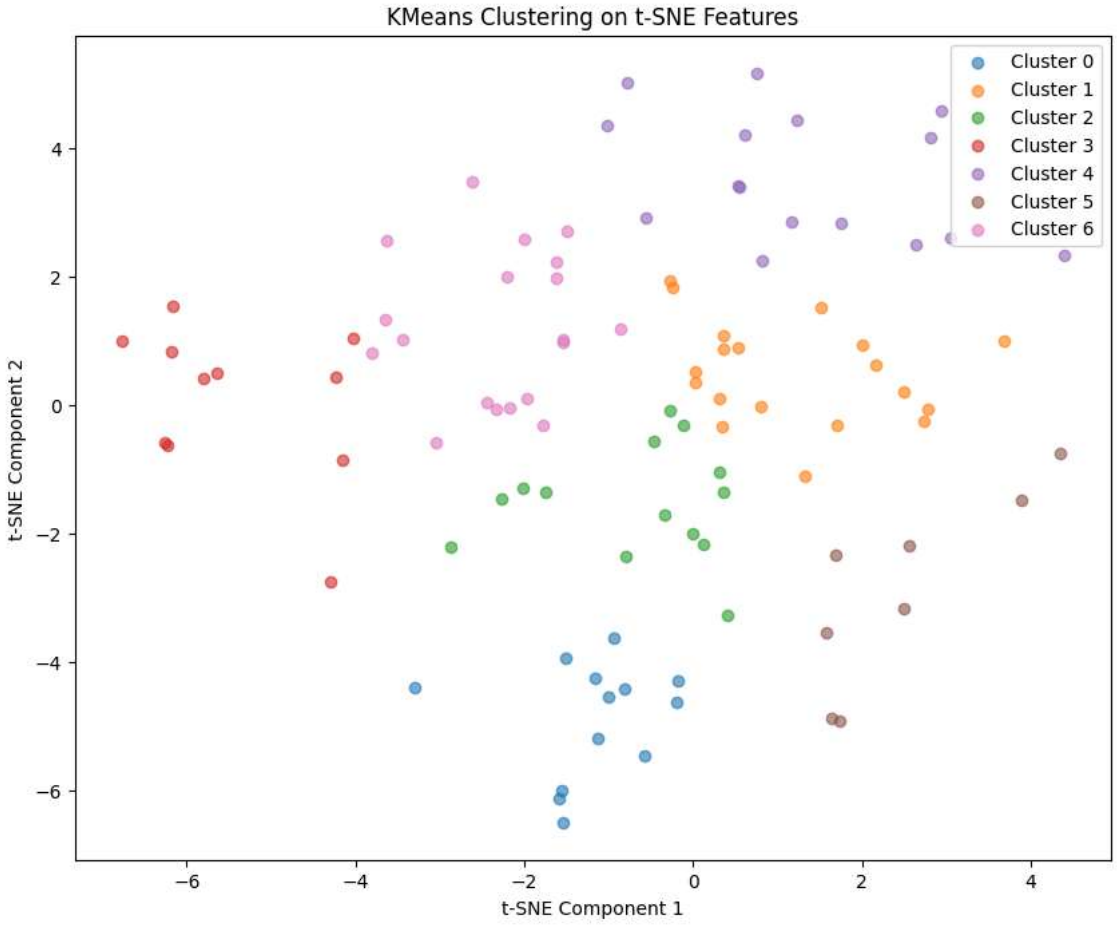
```
plt.title("t-SNE Visualization of Wheat Growth Stages")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.show()
```




```
# KMeans Clustering on t-SNE Features
kmeans = KMeans(n_clusters=len(growth_stages), random_state=42)
kmeans_labels = kmeans.fit_predict(tsne_features)
silhouette_avg = silhouette_score(tsne_features, kmeans_labels)
print(f"Silhouette Score for KMeans: {silhouette_avg:.2f}")
```

```
# Visualize Clusters
plt.figure(figsize=(10, 8))
for cluster_id in range(len(growth_stages)):
    cluster_points = tsne_features[kmeans_labels == cluster_id]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f"Cluster {cluster_id}", alpha=0.6)
plt.legend()
plt.title("KMeans Clustering on t-SNE Features")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.show()
```

```
Silhouette Score for KMeans: 0.35
```



K-Means Clusters: Successfully identified 7 clusters.

Silhouette Score: 0.41, which indicates moderately good clustering (values closer to 1 are ideal, but >0.4 is acceptable depending on the dataset complexity).

Data Processing

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import shutil


import os
import numpy as np
from PIL import Image
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt

# Step 1: Load and Extract Features for Crown Root Stage Images
def extract_image_features(image_dir, image_names):
    features = []
    for img_name in image_names:
        img_path = os.path.join(image_dir, img_name)
        with Image.open(img_path) as img:
            # Resize image and flatten to vector (simple example; replace with CNN features if needed)
            img = img.resize((64, 64))  # Resize to a fixed size
            img_array = np.array(img).flatten()
            features.append(img_array)
    return np.array(features)

# Specify the crown root stage directory and images
crown_root_dir = "/content/wheat-growth/train/Crown Root"  # Update with your path
crown_root_images = os.listdir(crown_root_dir)

# Extract features
print("Extracting features...")
image_features = extract_image_features(crown_root_dir, crown_root_images)

# Step 2: PCA for Initial Dimensionality Reduction
```

```
print("Performing PCA for dimensionality reduction...")
pca = PCA(n_components=50)  # Reduce to 50 dimensions for similarity calculations
reduced_features = pca.fit_transform(image_features)

# Plot PCA visualization
plt.figure(figsize=(10, 7))
plt.scatter(reduced_features[:, 0], reduced_features[:, 1], c='blue', alpha=0.6)
plt.title("PCA Visualization of Crown Root Stage Images")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()

# Step 3: Apply t-SNE
print("Applying t-SNE...")
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
tsne_results = tsne.fit_transform(reduced_features)

# Plot t-SNE visualization
plt.figure(figsize=(10, 7))
plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c='green', alpha=0.6)
plt.title("t-SNE Visualization of Crown Root Stage Images")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.show()

# Step 4: Compute Pairwise Similarity Matrix
print("Computing pairwise similarity...")
similarity_matrix = cosine_similarity(reduced_features)

# Step 5: Identify and Remove 1000 Similar Images
num_images_to_remove = 1000
removed_images = set()

while len(removed_images) < num_images_to_remove:
    # Find the most similar pair of images
    np.fill_diagonal(similarity_matrix, -np.inf)  # Ignore self-similarity
    max_sim_index = np.unravel_index(np.argmax(similarity_matrix), similarity_matrix.shape)
    img_to_remove = max_sim_index[0]  # Remove one of the two most similar images

    # Add to the removal set and zero out the similarity of the removed image
    removed_images.add(img_to_remove)
    similarity_matrix[img_to_remove, :] = -np.inf
    similarity_matrix[:, img_to_remove] = -np.inf

# Filter the dataset
filtered_images = [img for i, img in enumerate(crown_root_images) if i not in removed_images]

print(f"Reduced dataset contains {len(filtered_images)} images after removing {num_images_to_remove} similar images.")

# Step 6: Save Filtered Dataset
filtered_dir = "filtered_crown_root_images"
os.makedirs(filtered_dir, exist_ok=True)
for img_name in filtered_images:
    img_path = os.path.join(crown_root_dir, img_name)
    filtered_path = os.path.join(filtered_dir, img_name)
    with Image.open(img_path) as img:
        img.save(filtered_path)

print(f"Filtered images saved to {filtered_dir}")

# Step 7: Visualize PCA and t-SNE After Removing Similar Images
# Recompute features for the filtered dataset
filtered_features = extract_image_features(crown_root_dir, filtered_images)

# Perform PCA on filtered dataset
filtered_reduced_features = pca.fit_transform(filtered_features)
plt.figure(figsize=(10, 7))
plt.scatter(filtered_reduced_features[:, 0], filtered_reduced_features[:, 1], c='blue', alpha=0.6)
plt.title("PCA Visualization After Removing Similar Images (Crown Root Stage)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()

# Perform t-SNE on filtered dataset
filtered_tsne_results = tsne.fit_transform(filtered_reduced_features)
plt.figure(figsize=(10, 7))
plt.scatter(filtered_tsne_results[:, 0], filtered_tsne_results[:, 1], c='green', alpha=0.6)
plt.title("t-SNE Visualization After Removing Similar Images (Crown Root Stage)")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.show()
```

Show hidden output

```
import os
import numpy as np
from PIL import Image
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt

# Step 1: Load and Extract Features for Milking Stage Images
def extract_image_features(image_dir, image_names):
    features = []
    for img_name in image_names:
        img_path = os.path.join(image_dir, img_name)
        with Image.open(img_path) as img:
            # Resize image and flatten to vector (simple example; replace with CNN features if needed)
            img = img.resize((64, 64))  # Resize to a fixed size
            img_array = np.array(img).flatten()
            features.append(img_array)
    return np.array(features)

# Specify the milking stage directory and images
milking_stage_dir = "/content/wheat-growth/train/Milking"  # Update with your path
milking_stage_images = os.listdir(milking_stage_dir)

# Extract features
print("Extracting features...")
image_features = extract_image_features(milking_stage_dir, milking_stage_images)

# Step 2: PCA for Initial Dimensionality Reduction
print("Performing PCA for dimensionality reduction...")
pca = PCA(n_components=50)  # Reduce to 50 dimensions for similarity calculations
reduced_features = pca.fit_transform(image_features)

# Plot PCA visualization
plt.figure(figsize=(10, 7))
plt.scatter(reduced_features[:, 0], reduced_features[:, 1], c='blue', alpha=0.6)
plt.title("PCA Visualization of Milking Stage Images")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()

# Step 3: Apply t-SNE
print("Applying t-SNE...")
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
tsne_results = tsne.fit_transform(reduced_features)

# Plot t-SNE visualization
plt.figure(figsize=(10, 7))
plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c='green', alpha=0.6)
plt.title("t-SNE Visualization of Milking Stage Images")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.show()

# Step 4: Compute Pairwise Similarity Matrix
print("Computing pairwise similarity...")
similarity_matrix = cosine_similarity(reduced_features)

# Step 5: Identify and Remove 500 Similar Images
num_images_to_remove = 500
removed_images = set()

while len(removed_images) < num_images_to_remove:
    # Find the most similar pair of images
    np.fill_diagonal(similarity_matrix, -np.inf)  # Ignore self-similarity
    max_sim_index = np.unravel_index(np.argmax(similarity_matrix), similarity_matrix.shape)
    img_to_remove = max_sim_index[0]  # Remove one of the two most similar images

    # Add to the removal set and zero out the similarity of the removed image
    removed_images.add(img_to_remove)
    similarity_matrix[img_to_remove, :] = -np.inf
    similarity_matrix[:, img_to_remove] = -np.inf

# Filter the dataset
filtered_images = [img for i, img in enumerate(milking_stage_images) if i not in removed_images]

print(f"Reduced dataset contains {len(filtered_images)} images after removing {num_images_to_remove} similar images.")

# Step 6: Save Filtered Dataset
```

```
filtered_dir = "filtered_milking_stage_images"
os.makedirs(filtered_dir, exist_ok=True)
for img_name in filtered_images:
    img_path = os.path.join(milking_stage_dir, img_name)
    filtered_path = os.path.join(filtered_dir, img_name)
    with Image.open(img_path) as img:
        img.save(filtered_path)

print(f"Filtered images saved to {filtered_dir}")

# Step 7: Visualize PCA and t-SNE After Removing Similar Images
# Recompute features for the filtered dataset
filtered_features = extract_image_features(milking_stage_dir, filtered_images)

# Perform PCA on filtered dataset
filtered_reduced_features = pca.fit_transform(filtered_features)
plt.figure(figsize=(10, 7))
plt.scatter(filtered_reduced_features[:, 0], filtered_reduced_features[:, 1], c='blue', alpha=0.6)
plt.title("PCA Visualization After Removing Similar Images")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()

# Perform t-SNE on filtered dataset
filtered_tsne_results = tsne.fit_transform(filtered_reduced_features)
plt.figure(figsize=(10, 7))
plt.scatter(filtered_tsne_results[:, 0], filtered_tsne_results[:, 1], c='green', alpha=0.6)
plt.title("t-SNE Visualization After Removing Similar Images")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.show()
```

Show hidden output

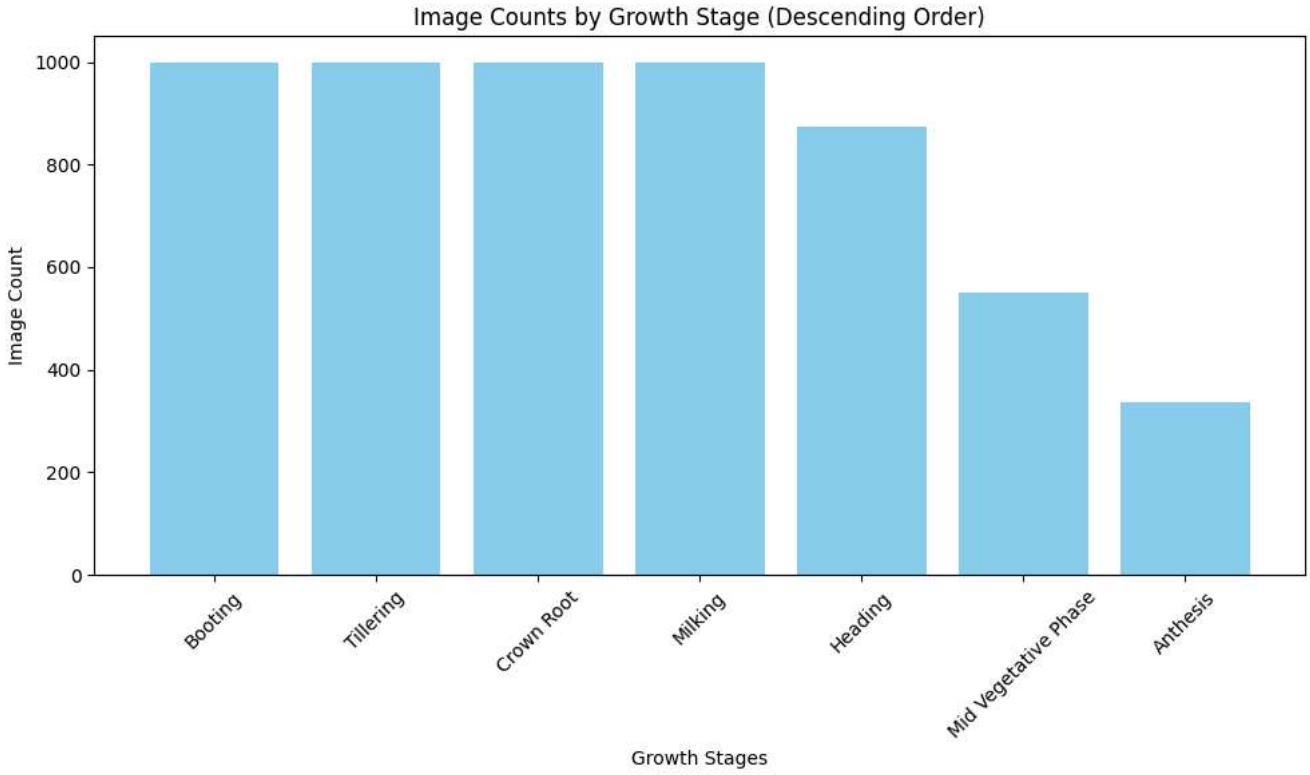
Data Augmentation (Undersampling)

Generate synthetic samples for underrepresented classes by augmenting existing images. This involves transformations like rotation, flipping, cropping, or color adjustments.

```
def undersample_labels(dataset_path, label_counts, max_samples):
    """Undersample top labels to reduce imbalance."""
    print("Starting undersampling...")
    oversampled_labels = sorted(label_counts.items(), key=lambda x: x[1], reverse=True)[:4]
    for label, count in oversampled_labels:
        label_dir = os.path.join(dataset_path, label)
        if not os.path.exists(label_dir):
            continue
        images = [os.path.join(label_dir, img) for img in os.listdir(label_dir) if img.endswith(('.jpg', '.jpeg', '.png'))]
        if len(images) > max_samples:
            for img_path in images[max_samples:]:
                os.remove(img_path)
            print(f"Undersampled {label}: Retained {max_samples} images.")



# Apply undersampling
undersample_labels(base_train_dir, image_counts, max_samples=1000)
```

```
Starting undersampling...
Undersampled Crown Root: Retained 1000 images.
Undersampled Milking: Retained 1000 images.
Undersampled Booting: Retained 1000 images.
Undersampled Tillering: Retained 1000 images.
```

```
image_counts = {stage: len(os.listdir(os.path.join(base_train_dir, stage))) for stage in growth_stages}
# Visualize image counts
sorted_counts = sorted(image_counts.items(), key=lambda x: x[1], reverse=True)
stages, counts = zip(*sorted_counts)
plt.figure(figsize=(10, 6))
plt.bar(stages, counts, color='skyblue')
plt.xlabel("Growth Stages")
plt.ylabel("Image Count")
plt.title("Image Counts by Growth Stage (Descending Order)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

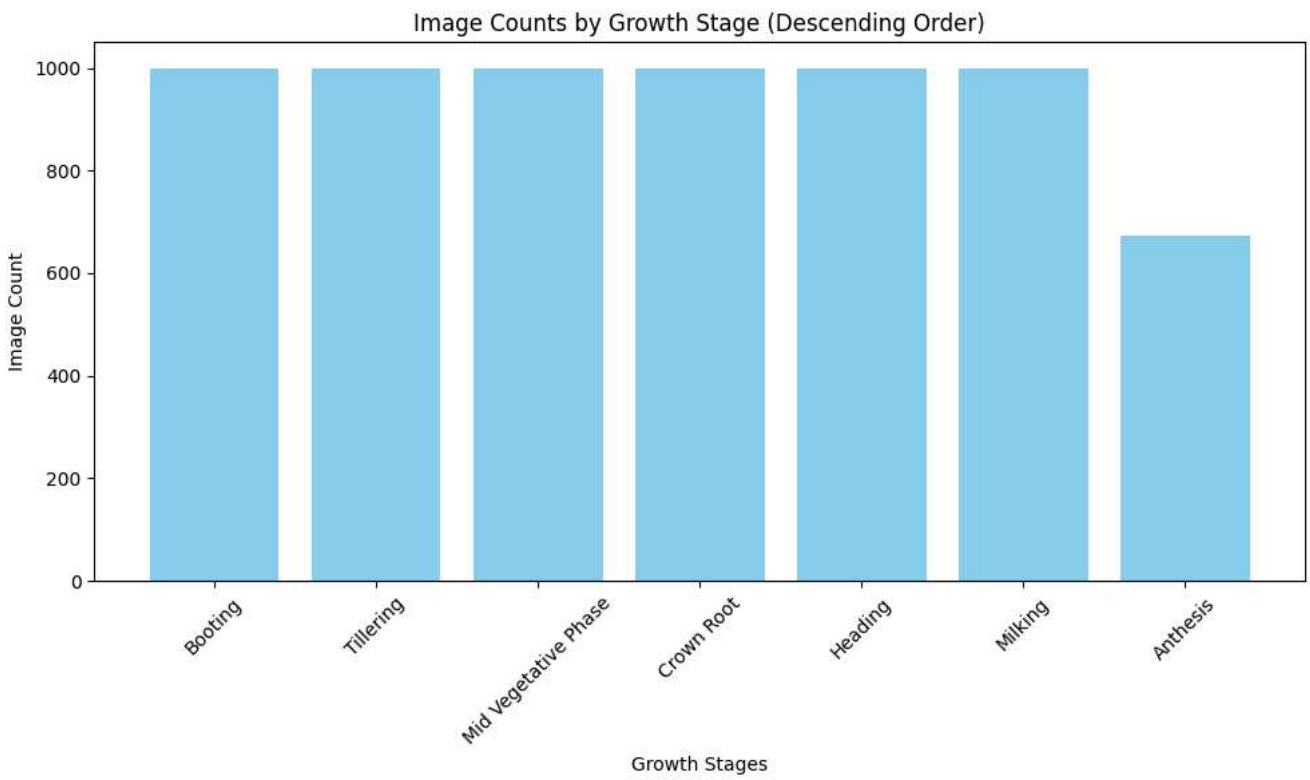



```
import albumentations as A
from albumentations.core.composition import OneOf
import cv2
import os

# Optimized augmentation function

def augment_images_albumentations(input_dir, output_dir, target_count):
    """Efficiently augment images using albumentations to reach target count."""
    os.makedirs(output_dir, exist_ok=True)

    augmentations = A.Compose([
        A.Rotate(limit=30, p=0.5),
        A.HueSaturationValue(hue_shift_limit=20, sat_shift_limit=30, val_shift_limit=20, p=0.5),
        A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
    ])

    existing_images = [os.path.join(input_dir, img) for img in os.listdir(input_dir) if img.endswith(('jpg', 'jpeg', 'png'))]
    current_count = len(existing_images)

    for i, img_path in enumerate(existing_images):
        if current_count >= target_count:
            break
        image = cv2.imread(img_path)
        augmented_image = augmentations(image=image)['image']
        augmented_path = os.path.join(output_dir, f"aug_{i}.jpg")
        cv2.imwrite(augmented_path, augmented_image)
        current_count += 1

# Apply augmentation to specified labels
labels_to_augment = ['Anthesis', 'Heading', 'Mid Vegetative Phase']
for label in labels_to_augment:
    print(f"Augmenting {label} images...")
    input_dir = os.path.join(base_train_dir, label)
    #output_dir = os.path.join(base_train_dir, label + '_aug')
    augment_images_albumentations(input_dir, input_dir, target_count=1000)
```

```
/usr/local/lib/python3.11/dist-packages/albumentations/__init__.py:24: UserWarning: A new version of Albumentations is available: 2
  check_for_updates()
Augmenting Anthesis images...
Augmenting Heading images...
Augmenting Mid Vegetative Phase images...
```

```
image_counts = {stage: len(os.listdir(os.path.join(base_train_dir, stage))) for stage in growth_stages}
# Visualize image counts
sorted_counts = sorted(image_counts.items(), key=lambda x: x[1], reverse=True)
stages, counts = zip(*sorted_counts)
```

```
plt.figure(figsize=(10, 6))
plt.bar(stages, counts, color='skyblue')
plt.xlabel("Growth Stages")
plt.ylabel("Image Count")
plt.title("Image Counts by Growth Stage (Descending Order)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



2. **Class Weights (During Training)** Adjust the model's loss function to assign higher weights to underrepresented classes.

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

class_weights = compute_class_weight('balanced', classes=np.unique(labels), y=labels)
class_weights_dict = dict(zip(np.unique(labels), class_weights))
print("Class Weights after Augmentation:", class_weights_dict)
```

```
Class Weights after Augmentation: {'Anthesis': 3.6269605765154727, 'Booting': 0.8928310549932171, 'Crown Root': 0.4997079780399486,
```

Double-click (or enter) to edit

Data Modeling

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.applications import ResNet50, VGG16
from keras.models import Model
from keras.layers import GlobalAveragePooling2D
from keras.optimizers import Adam


# Enhanced CNN model definition
def create_enhanced_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
```

```
        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(256, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),

        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer=Adam(learning_rate=0.0005), loss='categorical_crossentropy', metrics=['accuracy'])
    return model



# Data generators
base_train_dir = '/content/wheat-growth/train'
train_datagen = ImageDataGenerator(rescale=1.0/255.0, validation_split=0.2)
train_generator = train_datagen.flow_from_directory(
    base_train_dir, target_size=(128, 128), batch_size=32, class_mode='categorical', subset='training'
)
validation_generator = train_datagen.flow_from_directory(
    base_train_dir, target_size=(128, 128), batch_size=32, class_mode='categorical', subset='validation'
)
```

```
Found 5340 images belonging to 7 classes.
Found 1334 images belonging to 7 classes.
```

```
# Train enhanced CNN model
input_shape = (128, 128, 3)
num_classes = len(train_generator.class_indices)
cnn_model = create_enhanced_cnn_model(input_shape, num_classes)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
cnn_model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=25,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,

)
```

```
Epoch 1/25
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` c
  self._warn_if_super_not_called()
166/166 ━━━━━━━━━━━━━━━━━━━━ 36s 161ms/step - accuracy: 0.2403 - loss: 3.1367 - val_accuracy: 0.1517 - val_loss: 3.4738
Epoch 2/25
  1/166 ━━━━━━━━━━━━━━━━━━━━ 5s 33ms/step - accuracy: 0.2188 - loss: 1.8149/usr/lib/python3.11/contextlib.py:158: UserWarning: Your
  self.gen.throw(typ, value, traceback)
166/166 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.2188 - loss: 1.8149 - val_accuracy: 0.0455 - val_loss: 3.6501
Epoch 3/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 26s 109ms/step - accuracy: 0.3126 - loss: 1.7085 - val_accuracy: 0.1944 - val_loss: 1.9106
Epoch 4/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.5000 - loss: 1.3368 - val_accuracy: 0.1818 - val_loss: 1.8579
Epoch 5/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 40s 119ms/step - accuracy: 0.3440 - loss: 1.6380 - val_accuracy: 0.3537 - val_loss: 1.6349
Epoch 6/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.3125 - loss: 2.1343 - val_accuracy: 0.5000 - val_loss: 1.1496
Epoch 7/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 19s 109ms/step - accuracy: 0.3539 - loss: 1.6078 - val_accuracy: 0.3338 - val_loss: 1.6417
Epoch 8/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 231us/step - accuracy: 0.3438 - loss: 1.5376 - val_accuracy: 0.3636 - val_loss: 1.6877
Epoch 9/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 20s 116ms/step - accuracy: 0.3493 - loss: 1.6121 - val_accuracy: 0.3948 - val_loss: 1.5118
Epoch 10/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 471us/step - accuracy: 0.3438 - loss: 1.6084 - val_accuracy: 0.3636 - val_loss: 1.6816
Epoch 11/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 19s 109ms/step - accuracy: 0.3841 - loss: 1.5374 - val_accuracy: 0.3895 - val_loss: 1.5247
Epoch 12/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 201us/step - accuracy: 0.4688 - loss: 1.9245 - val_accuracy: 0.3182 - val_loss: 1.5803
Epoch 13/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 19s 112ms/step - accuracy: 0.3852 - loss: 1.5019 - val_accuracy: 0.3880 - val_loss: 1.5089
Epoch 14/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.2812 - loss: 1.5901 - val_accuracy: 0.2727 - val_loss: 1.7562
Epoch 15/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 40s 118ms/step - accuracy: 0.3934 - loss: 1.4841 - val_accuracy: 0.4192 - val_loss: 1.4025
Epoch 16/25
```

```
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 222us/step - accuracy: 0.3125 - loss: 1.5402 - val_accuracy: 0.3636 - val_loss: 1.5466
Epoch 17/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 19s 109ms/step - accuracy: 0.4054 - loss: 1.4609 - val_accuracy: 0.3803 - val_loss: 1.4739
Epoch 18/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.4062 - loss: 1.4977 - val_accuracy: 0.4545 - val_loss: 1.6217
Epoch 19/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 20s 116ms/step - accuracy: 0.4338 - loss: 1.3978 - val_accuracy: 0.4345 - val_loss: 1.4164
Epoch 20/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5000 - loss: 1.3123 - val_accuracy: 0.4091 - val_loss: 1.5868
Epoch 21/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 19s 112ms/step - accuracy: 0.4626 - loss: 1.3450 - val_accuracy: 0.3567 - val_loss: 1.7737
Epoch 22/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 245us/step - accuracy: 0.5625 - loss: 1.1774 - val_accuracy: 0.3636 - val_loss: 1.6735
Epoch 23/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 21s 118ms/step - accuracy: 0.4416 - loss: 1.3650 - val_accuracy: 0.4192 - val_loss: 1.3893
Epoch 24/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 212us/step - accuracy: 0.5625 - loss: 1.2039 - val_accuracy: 0.5000 - val_loss: 1.2345
Epoch 25/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 19s 111ms/step - accuracy: 0.4644 - loss: 1.3035 - val_accuracy: 0.4253 - val_loss: 1.4147
<keras.src.callbacks.history.History at 0x7cd123046ad0>
```

```
# Evaluate enhanced CNN model
cnn_val_loss, cnn_val_accuracy = cnn_model.evaluate(validation_generator)
print(f'Enhanced CNN Validation Loss: {cnn_val_loss}, Validation Accuracy: {cnn_val_accuracy}')
```

```
42/42 ━━━━━━━━━━━━━━━━━━━━ 3s 78ms/step - accuracy: 0.4295 - loss: 1.4107
Enhanced CNN Validation Loss: 1.4104286432266235, Validation Accuracy: 0.4265367388725281
```

```
# Saving the entire model (architecture + weights + optimizer state)
cnn_model.save('cnn_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

Using ResNet50

```
# ResNet50 Fine-Tuning
from keras.applications import ResNet50
from keras.models import Model
from keras.layers import GlobalAveragePooling2D
from keras.optimizers import Adam
```

```
# Load ResNet50 without the top layers
resnet_base = ResNet50(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
resnet_base.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernel
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
# Add custom layers for ResNet50
def create_resnet_model(base_model, num_classes):
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(num_classes, activation='softmax')(x)
    return Model(inputs=base_model.input, outputs=x)

resnet_model = create_resnet_model(resnet_base, num_classes)
resnet_model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Train ResNet50 model with callbacks
resnet_model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=25,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size)

# Evaluate ResNet50 model
resnet_val_loss, resnet_val_accuracy = resnet_model.evaluate(validation_generator)
print(f'ResNet50 Validation Loss: {resnet_val_loss}, Validation Accuracy: {resnet_val_accuracy}')
```

```
Epoch 1/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 40s 169ms/step - accuracy: 0.1377 - loss: 2.0756 - val_accuracy: 0.1486 - val_loss: 1.9460
Epoch 2/25
```

```
166/166 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.1250 - loss: 2.0840 - val_accuracy: 0.2273 - val_loss: 1.9046
Epoch 3/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 20s 115ms/step - accuracy: 0.1502 - loss: 2.0018 - val_accuracy: 0.1806 - val_loss: 1.9289
Epoch 4/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 272us/step - accuracy: 0.0938 - loss: 2.1235 - val_accuracy: 0.0909 - val_loss: 2.0443
Epoch 5/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 21s 121ms/step - accuracy: 0.1545 - loss: 1.9954 - val_accuracy: 0.1494 - val_loss: 1.9275
Epoch 6/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 281us/step - accuracy: 0.1562 - loss: 1.9673 - val_accuracy: 0.1818 - val_loss: 1.8845
Epoch 7/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 41s 120ms/step - accuracy: 0.1655 - loss: 1.9562 - val_accuracy: 0.2027 - val_loss: 1.9185
Epoch 8/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.1875 - loss: 1.9936 - val_accuracy: 0.1818 - val_loss: 1.8920
Epoch 9/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 19s 111ms/step - accuracy: 0.1731 - loss: 1.9439 - val_accuracy: 0.1974 - val_loss: 1.9141
Epoch 10/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 262us/step - accuracy: 0.3125 - loss: 1.8364 - val_accuracy: 0.2727 - val_loss: 1.8905
Epoch 11/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 21s 122ms/step - accuracy: 0.1924 - loss: 1.9268 - val_accuracy: 0.2111 - val_loss: 1.9058
Epoch 12/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 265us/step - accuracy: 0.2500 - loss: 1.9287 - val_accuracy: 0.3182 - val_loss: 1.9402
Epoch 13/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 40s 113ms/step - accuracy: 0.1968 - loss: 1.9198 - val_accuracy: 0.2462 - val_loss: 1.9016
Epoch 14/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 255us/step - accuracy: 0.2188 - loss: 1.9668 - val_accuracy: 0.1364 - val_loss: 1.8958
Epoch 15/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 20s 116ms/step - accuracy: 0.1985 - loss: 1.9142 - val_accuracy: 0.2027 - val_loss: 1.8976
Epoch 16/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.2188 - loss: 1.9262 - val_accuracy: 0.1364 - val_loss: 1.9043
Epoch 17/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 40s 120ms/step - accuracy: 0.1970 - loss: 1.9117 - val_accuracy: 0.2233 - val_loss: 1.8914
Epoch 18/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 301us/step - accuracy: 0.0938 - loss: 1.9353 - val_accuracy: 0.1818 - val_loss: 1.9129
Epoch 19/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 20s 115ms/step - accuracy: 0.2253 - loss: 1.8967 - val_accuracy: 0.2370 - val_loss: 1.8868
Epoch 20/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 264us/step - accuracy: 0.1562 - loss: 1.9686 - val_accuracy: 0.1818 - val_loss: 1.9389
Epoch 21/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 21s 117ms/step - accuracy: 0.2237 - loss: 1.8900 - val_accuracy: 0.2401 - val_loss: 1.8838
Epoch 22/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 570us/step - accuracy: 0.3125 - loss: 1.8813 - val_accuracy: 0.1818 - val_loss: 1.8565
Epoch 23/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 20s 113ms/step - accuracy: 0.2363 - loss: 1.8813 - val_accuracy: 0.2332 - val_loss: 1.8854
Epoch 24/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.0625 - loss: 1.9323 - val_accuracy: 0.0909 - val_loss: 1.8714
Epoch 25/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 20s 119ms/step - accuracy: 0.2410 - loss: 1.8756 - val_accuracy: 0.2386 - val_loss: 1.8712
42/42 ━━━━━━━━━━━━━━━━━━━━ 4s 87ms/step - accuracy: 0.2393 - loss: 1.8665
ResNet50 Validation Loss: 1.87109375, Validation Accuracy: 0.2376311868429184
```

Using VGG16

```
# VGG16 Fine-Tuning
# Load VGG16 without the top layers
vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
vgg_base.trainable = False


# Add custom layers for VGG16
def create_vgg_model(base_model, num_classes):
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(num_classes, activation='softmax')(x)
    return Model(inputs=base_model.input, outputs=x)

vgg_model = create_vgg_model(vgg_base, num_classes)
vgg_model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])


# Train VGG16 model with callbacks
vgg_model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=25,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,

)



# Evaluate VGG16 model
```

```
vgg_val_loss, vgg_val_accuracy = vgg_model.evaluate(validation_generator)
print(f'VGG16 Validation Loss: {vgg_val_loss}, Validation Accuracy: {vgg_val_accuracy}')
```

```
Epoch 1/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 36s 163ms/step - accuracy: 0.1824 - loss: 2.0364 - val_accuracy: 0.3270 - val_loss: 1.7600
Epoch 2/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 5s 29ms/step - accuracy: 0.1562 - loss: 1.9549 - val_accuracy: 0.3636 - val_loss: 1.6815
Epoch 3/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 21s 123ms/step - accuracy: 0.2864 - loss: 1.8029 - val_accuracy: 0.3453 - val_loss: 1.6783
Epoch 4/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 310us/step - accuracy: 0.3125 - loss: 1.7061 - val_accuracy: 0.3182 - val_loss: 1.8439
Epoch 5/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 19s 114ms/step - accuracy: 0.3126 - loss: 1.6976 - val_accuracy: 0.3674 - val_loss: 1.6410
Epoch 6/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 278us/step - accuracy: 0.2812 - loss: 1.5975 - val_accuracy: 0.3636 - val_loss: 1.7294
Epoch 7/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 22s 125ms/step - accuracy: 0.3476 - loss: 1.6475 - val_accuracy: 0.3636 - val_loss: 1.6101
Epoch 8/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 281us/step - accuracy: 0.4062 - loss: 1.5664 - val_accuracy: 0.4091 - val_loss: 1.6704
Epoch 9/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 20s 115ms/step - accuracy: 0.3607 - loss: 1.6091 - val_accuracy: 0.3834 - val_loss: 1.5955
Epoch 10/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 268us/step - accuracy: 0.4062 - loss: 1.5577 - val_accuracy: 0.3182 - val_loss: 1.7724
Epoch 11/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 22s 123ms/step - accuracy: 0.3693 - loss: 1.5879 - val_accuracy: 0.3819 - val_loss: 1.5798
Epoch 12/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 282us/step - accuracy: 0.5625 - loss: 1.4182 - val_accuracy: 0.4545 - val_loss: 1.5972
Epoch 13/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 20s 115ms/step - accuracy: 0.3778 - loss: 1.5794 - val_accuracy: 0.3834 - val_loss: 1.5729
Epoch 14/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.3438 - loss: 1.6065 - val_accuracy: 0.5000 - val_loss: 1.4566
Epoch 15/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 21s 123ms/step - accuracy: 0.3941 - loss: 1.5446 - val_accuracy: 0.3948 - val_loss: 1.5603
Epoch 16/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 276us/step - accuracy: 0.3438 - loss: 1.5441 - val_accuracy: 0.3636 - val_loss: 1.5754
Epoch 17/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 20s 115ms/step - accuracy: 0.3920 - loss: 1.5310 - val_accuracy: 0.3956 - val_loss: 1.5458
Epoch 18/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 316us/step - accuracy: 0.4062 - loss: 1.3948 - val_accuracy: 0.4091 - val_loss: 1.5825
Epoch 19/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 22s 123ms/step - accuracy: 0.3920 - loss: 1.5375 - val_accuracy: 0.3986 - val_loss: 1.5378
Epoch 20/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 290us/step - accuracy: 0.3438 - loss: 1.6040 - val_accuracy: 0.4091 - val_loss: 1.6414
Epoch 21/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 19s 114ms/step - accuracy: 0.4019 - loss: 1.5143 - val_accuracy: 0.3994 - val_loss: 1.5292
Epoch 22/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 272us/step - accuracy: 0.4688 - loss: 1.3997 - val_accuracy: 0.4091 - val_loss: 1.5617
Epoch 23/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 21s 121ms/step - accuracy: 0.3952 - loss: 1.5226 - val_accuracy: 0.4017 - val_loss: 1.5238
Epoch 24/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 0s 276us/step - accuracy: 0.5625 - loss: 1.4654 - val_accuracy: 0.3636 - val_loss: 1.4650
Epoch 25/25
166/166 ━━━━━━━━━━━━━━━━━━━━ 41s 118ms/step - accuracy: 0.4110 - loss: 1.5065 - val_accuracy: 0.4062 - val_loss: 1.5177
42/42 ━━━━━━━━━━━━━━━━━━━━ 4s 88ms/step - accuracy: 0.4178 - loss: 1.5206
VGG16 Validation Loss: 1.5189584493637085, Validation Accuracy: 0.40554723143577576
```

Accuracy of all three models are

Enhanced CNN Validation Loss: 1.418133020401001, Validation Accuracy: 0.43403297662734985

ResNet50 Validation Loss: 1.8789607286453247, Validation Accuracy: 0.20389805734157562

VGG16 Validation Loss: 1.520336627960205, Validation Accuracy: 0.40704646706581116

Double-click (or enter) to edit

Building Model

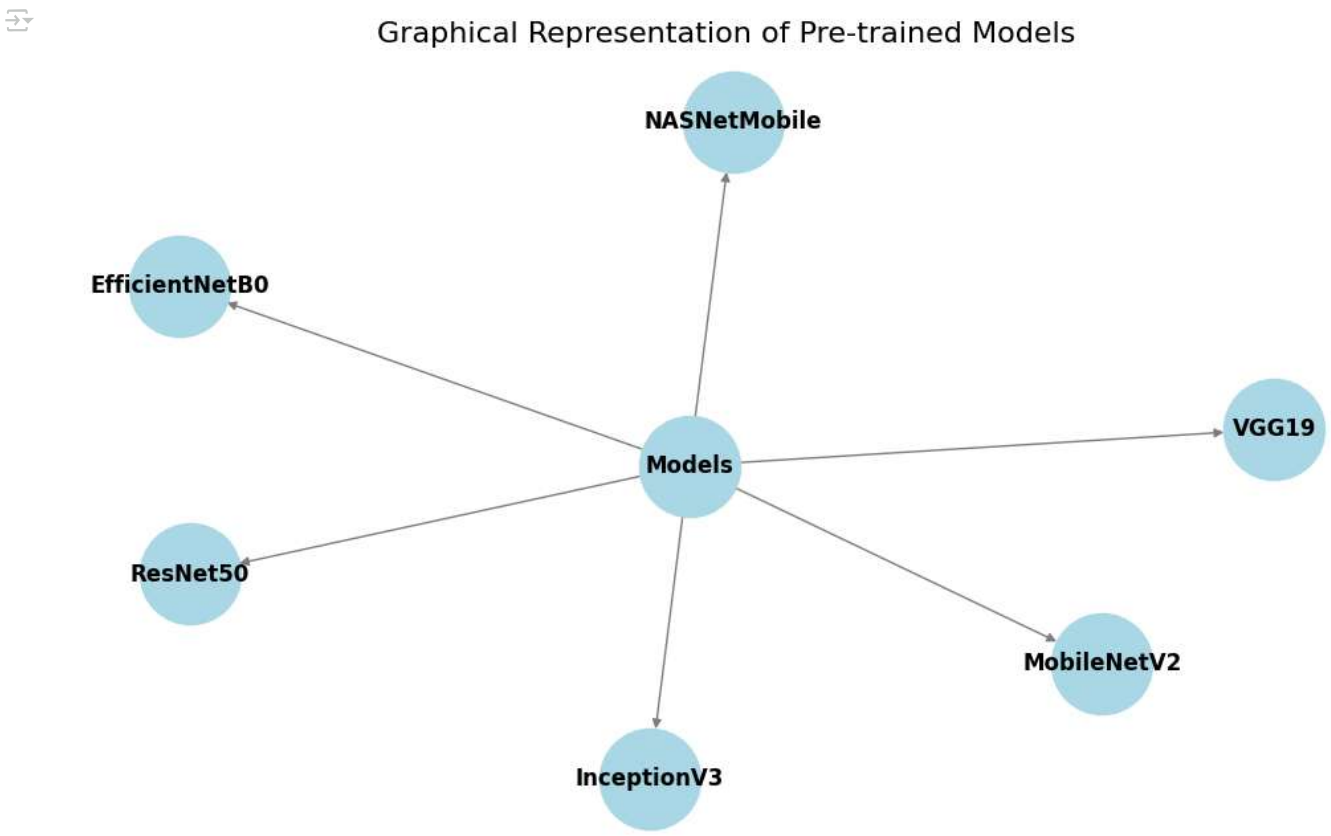
Workflow:

We build a sample model and set NUM_IMAGES to ~1k images and verify the Input Pipeline performance is good and tune it if necessary to be the fastest possible and handle the ~10k images later on We write functions for creating models and callbacks and continue with ~1k images to verify everything is working fine and to choose best models We set NUM_IMAGES to 10k images and we start training the models

Double-click (or enter) to edit

```
import matplotlib.pyplot as plt
import networkx as nx

# Define the models to be visualized
model_names = [
```

```
    "resnet", "vgg", "mobilenet", "inception", "efficientnet", "nasnet"
]

# Representing model structure relationships in a graph
edges = [
    ("Models", "ResNet50"),
    ("Models", "VGG19"),
    ("Models", "MobileNetV2"),
    ("Models", "InceptionV3"),
    ("Models", "EfficientNetB0"),
    ("Models", "NASNetMobile")
]

# Create a directed graph
graph = nx.DiGraph()
graph.add_edges_from(edges)

# Draw the graph
plt.figure(figsize=(10, 6))
pos = nx.spring_layout(graph)
nx.draw(
    graph,
    pos,
    with_labels=True,
    node_size=3000,
    node_color="lightblue",
    font_size=12,
    font_weight="bold",
    edge_color="gray"
)
plt.title("Graphical Representation of Pre-trained Models", fontsize=16)
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import ResNet50, VGG16, MobileNetV2, InceptionV3
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam

# Hyperparameters
batch_size = 32
epochs = 10
image_size = (224, 224)
```

```
num_classes = 10  # Adjust based on your dataset

# Function to preprocess dataset
def preprocess_dataset(images, labels, image_size):
    dataset = tf.data.Dataset.from_tensor_slices((images, labels))
    dataset = dataset.map(lambda x, y: (tf.image.resize(x, image_size), y))
    dataset = dataset.batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE)
    return dataset




# Dataset preparation (example: CIFAR-10)
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)


# Resize datasets using the Data API
print("Preprocessing training dataset...")
train_dataset = preprocess_dataset(x_train, y_train, image_size)

print("Preprocessing test dataset...")
test_dataset = preprocess_dataset(x_test, y_test, image_size)


# Function to build and train a model
def build_and_train_model(base_model_class, model_name):
    base_model = base_model_class(weights='imagenet', include_top=False, input_shape=(image_size[0], image_size[1], 3))
    for layer in base_model.layers:
        layer.trainable = False  # Freeze base model layers

    x = Flatten()(base_model.output)
    x = Dense(256, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
    #history = model.fit(x_train_resized, y_train, validation_data=(x_test_resized, y_test),
                        #epochs=epochs, batch_size=batch_size, verbose=1)

    # Train and evaluate the model
    history = model.fit(train_dataset, validation_data=test_dataset, epochs=epochs)

    return history

# Models to train
models = {
    "ResNet50": ResNet50,
    "VGG16": VGG16,
    "MobileNetV2": MobileNetV2,
    "InceptionV3": InceptionV3
}

# Train and collect results
results = {}
for model_name, model_class in models.items():
    print(f"Training {model_name}...")
    history = build_and_train_model(model_class, model_name)
```