```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import os
```

```
from google.colab import drive

# Manual mounting with user interaction
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
import os
drive_path = '/content/drive/My Drive'
```

```
original='/content/drive/MyDrive/Unsl'
```

```
print(len(original))
```

```
     27
```

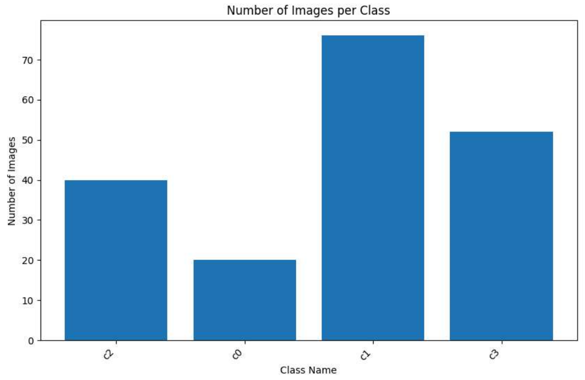
```
num_classes = len(os.listdir(original))
print(f"Number of classes found: {num_classes}")
```

```
     Number of classes found: 4
```

```
class_counts = {}

# Loop through all subfolders (classes)
for class_name in os.listdir(original):
  # Construct the full path to the class folder
  class_path = os.path.join(original, class_name)

  # Check if it's a directory (avoid hidden files)
  if os.path.isdir(class_path):
    # Count the number of image files in the class folder
    image_count = 0
    for filename in os.listdir(class_path):
      if filename.endswith(('.jpg')):
        image_count += 1

    # Store the count for this class
    class_counts[class_name] = image_count

# Print the class names and image counts
for class_name, count in class_counts.items():
  print(f"Class: {class_name}, Image Count: {count}")
```

```
     Class: c2, Image Count: 40
     Class: c0, Image Count: 20
     Class: c1, Image Count: 76
     Class: c3, Image Count: 52
```

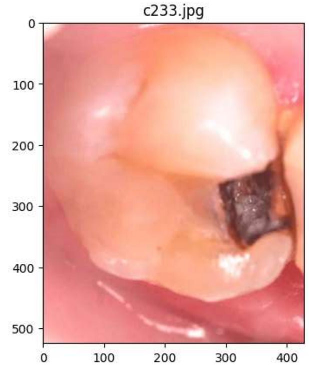
```
import matplotlib.pyplot as plt
# Create the bar graph
plt.figure(figsize=(10, 6))  # Adjust figure size as needed
plt.bar(class_counts.keys(), class_counts.values())  # Class names on x-axis, counts on y-axis
plt.xlabel("Class Name")
plt.ylabel("Number of Images")
plt.title("Number of Images per Class")
plt.xticks(rotation=45, ha='right')  # Rotate class names for better readability if many classes

# Display the graph (optional, graph won't be shown automatically in Colab)
plt.show()
```



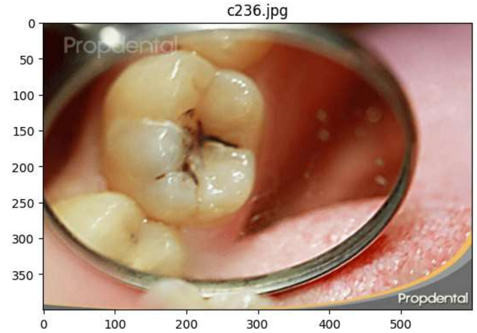
```
import matplotlib.pyplot as plt
import os

# Define the number of images to display per class
num_images_per_class = 3

# Loop through each class
for class_name in class_counts:
  # Construct the full path to the class folder
  class_path = os.path.join(original, class_name)

  # Get a list of image filenames in the class folder
  image_filenames = os.listdir(class_path)

  # Select the first `num_images_per_class` images
  selected_images = image_filenames[:num_images_per_class]

  # Print the class name
  print(f"Class: {class_name}")

  # Loop through the selected images
  for image_filename in selected_images:
    # Construct the full path to the image
    image_path = os.path.join(class_path, image_filename)

    # Load the image
    image = plt.imread(image_path)

    # Display the image with the image name
    plt.imshow(image)
    plt.title(image_filename)
    plt.show()

  # Print a new line for spacing
  print()
```

```
Class: c2
```

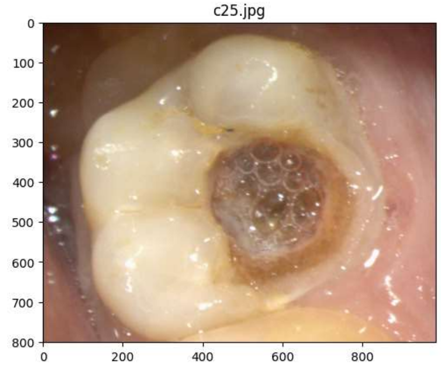
c233.jpg



c236.jpg

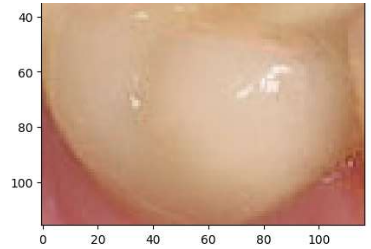


c25.jpg

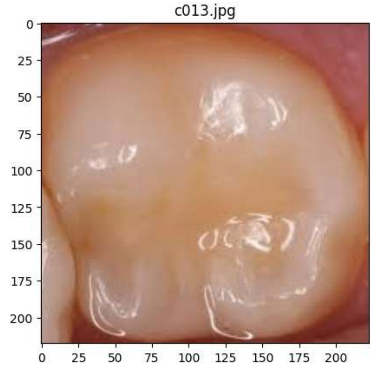


```
Class: c0
```

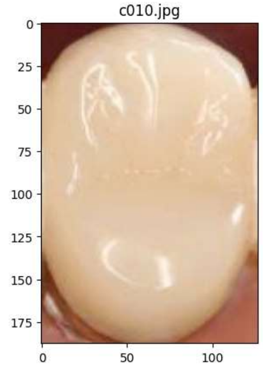
c014.jpg



c013.jpg

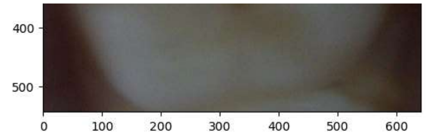


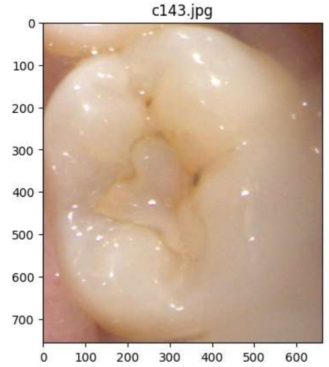
c010.jpg

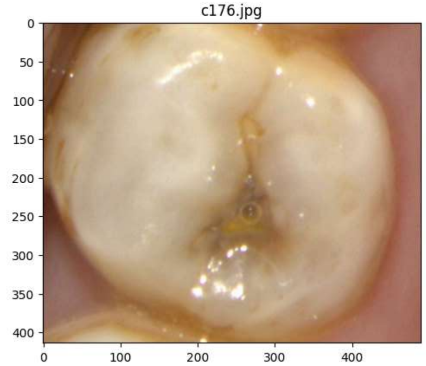


```
Class: c1
```

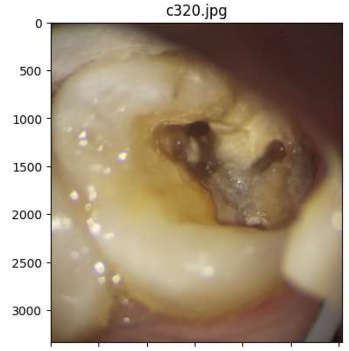
c164.jpg

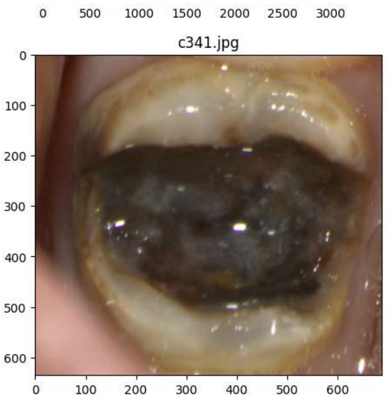


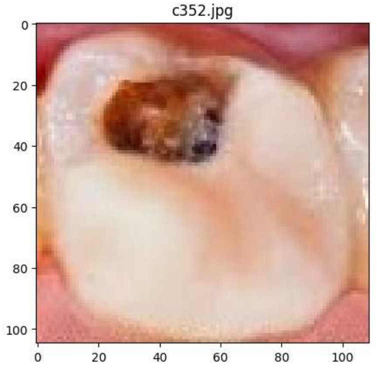




```
Class: c3
```







```
from collections import Counter
import cv2
image_extensions='.jpg'
def image_eda(original, image_extensions):
    image_sizes=[]
    color_spaces=Counter()
    dominant_colors=Counter()


def get_image_size(image_path):
  image = cv2.imread(image_path)  # Read the image
  if image is None:
    raise ValueError(f"Error reading image: {image_path}")
  height, width = image.shape[:2]  # Extract height and width (avoid alpha channel)
  return height, width
```

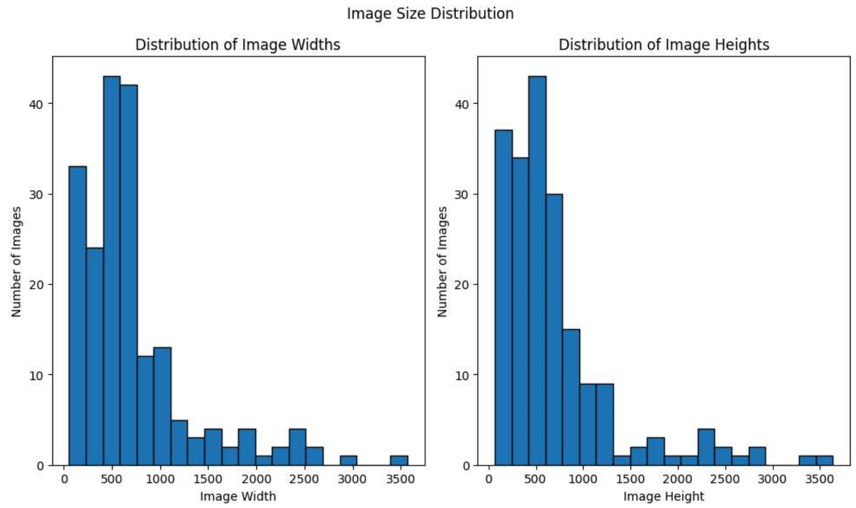
```
!pip install fast_colorthief
```

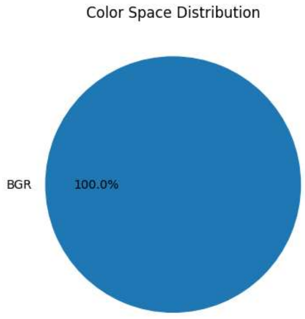
```
Collecting fast_colorthief
  Downloading fast_colorthief-0.0.5-cp310-cp310-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (67 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.1/67.1 kB 2.2 MB/s eta 0:00:00
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from fast_colorthief) (9.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from fast_colorthief) (1.25.2)
Installing collected packages: fast_colorthief
Successfully installed fast_colorthief-0.0.5
```

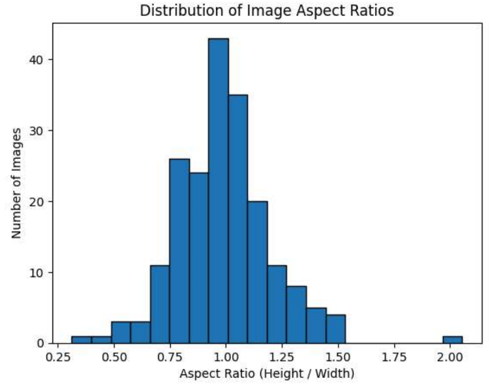
```
from collections import Counter
import fast_colorthief
import cv2
image_extensions='.jpg'
image_sizes=[]
color_spaces=Counter()
dominant_colors=Counter()
def image_eda(original, image_extensions):
    image_sizes=[]
    color_spaces=Counter()
    dominant_colors=Counter()
for root, _, filenames in os.walk(original):
  for filename in filenames:
    if filename.lower().endswith(tuple(image_extensions)):
      image_path = os.path.join(root, filename)
      image = cv2.imread(image_path)
      if image is None:
        print(f"Error reading image: {image_path}")
        continue
      def get_image_sizes(image_path):
        image = cv2.imread(image_path)  # Read the image
        if image is None:
          raise ValueError(f"Error reading image: {image_path}")
        height, width = image.shape[:2]  # Extract height and width (avoid alpha channel)
        return height, width
      height,width,channels = image.shape
      color_space="BGR" if channels == 3 else "Grayscale"
      dominant_color=fast_colorthief.get_dominant_color(image_path, quality=1)

      image_sizes.append((height, width))
      color_spaces[color_space] += 1
      dominant_colors[dominant_color] += 1  # If implemented

# Calculate summary statistics for image sizes
image_size_stats = {
    "min_width": min(w for h, w in image_sizes),
    "max_width": max(w for h, w in image_sizes),
    "min_height": min(h for h, w in image_sizes),
    "max_height": max(h for h, w in image_sizes),
    "aspect_ratios": [h / w for h, w in image_sizes],
}

# Print summary statistics
print("Image Sizes:")
for key, value in image_size_stats.items():
    print(f"{key.capitalize()}: {value}")

print("\nColor Space Distribution:")
for color_space, count in color_spaces.items():
    print(f"{color_space}: {count} images")

# Optional: Dominant color distribution (if implemented)
if dominant_colors:
    print("\nDominant Color Distribution:")
    for color, count in dominant_colors.items():
        print(f"{color}: {count} images")
```

```
2237, 0.6666666666666666, 0.8150406504065041, 1.0428954423592494, 1.0777096114519427, 0.9210526315789473, 0.7715736040609137, 1.048022
```

```
# Image size distribution (width and height)
plt.figure(figsize=(10, 6))
plt.subplot(121)
plt.hist([w for h, w in image_sizes], bins=20, edgecolor='black')
plt.xlabel('Image Width')
plt.ylabel('Number of Images')
plt.title('Distribution of Image Widths')

plt.subplot(122)
plt.hist([h for h, w in image_sizes], bins=20, edgecolor='black')
plt.xlabel('Image Height')
plt.ylabel('Number of Images')
plt.title('Distribution of Image Heights')

plt.suptitle('Image Size Distribution')
plt.tight_layout()
plt.show()

# Color space distribution (pie chart)
plt.pie(color_spaces.values(), labels=color_spaces.keys(), autopct="%1.1f%%")
plt.title('Color Space Distribution')
plt.show()

# Optional: Dominant color distribution (if implemented, similar to color space)
# ... (create pie chart for dominant colors)

# Aspect ratio distribution (histogram)
plt.hist(image_size_stats["aspect_ratios"], bins=20, edgecolor='black')
plt.xlabel('Aspect Ratio (Height / Width)')
plt.ylabel('Number of Images')
plt.title('Distribution of Image Aspect Ratios')
plt.show()
```







import os from tensorflow.keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img import os import cv2

Define the path to the directory where images are stored

base_dir=original

Define the diseases

diseases=["c0", "c1", "c2","c3"]

Create the directory rice_leaf_diseases_2 if it doesn't exist

output_dir='/content/drive/MyDrive/Edadonedataset/c0' os.makedirs(output_dir, exist_ok=True) aug=0

Loop through each disease

for disease in diseases:

```
 # Create subdirectories for each disease in rice_leaf_diseases_2 if they don't exist
 disease_dir = os.path.join(output_dir, disease)
 os.makedirs(disease_dir, exist_ok=True)

 # Create an ImageDataGenerator for each disease
 datagen = ImageDataGenerator(
     rotation_range=40,
     width_shift_range=0.2,
     height_shift_range=0.2,
     shear_range=0.2,
     zoom_range=0.2,
     horizontal_flip=True,
     fill_mode='nearest',
     rescale=1./255,
     validation_split=0.2
 )

 # Load images for the current disease
 img_dir = os.path.join(base_dir, disease)
 img_filenames = os.listdir(img_dir)

 for img_filename in img_filenames:
     img_path = os.path.join(img_dir, img_filename)
     img = load_img(img_path)
     x = img_to_array(img)
     x = x.reshape((1, )+x.shape)

     # Generate and save augmented images for the current disease
     i = 0
     for batch in datagen.flow(x, batch_size=1, save_to_dir=disease_dir, save_prefix=f"{disease}_", save_format='jpg'):
         i += 1
         aug=aug+1
         if i > 5:
             break
```

```
#Data augmnetation done
```

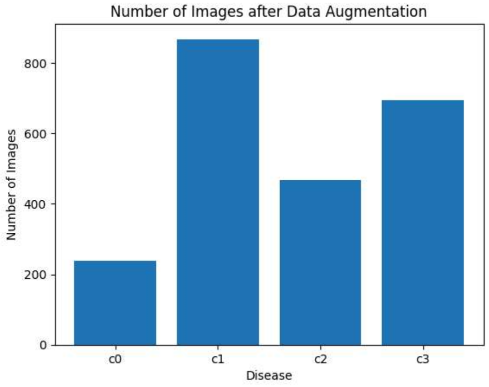
```
print(aug)
```

```
    1176
```

```
import os

# Define the path to the directory where augmented images are stored
augmented_dir = '/content/drive/MyDrive/Edadonedataset/c0'

# Define the diseases
diseases = ["c0", "c1", "c2", "c3"]

# Initialize a dictionary to store the number of images for each disease
disease_image_counts = {}

# Loop through each disease
for disease in diseases:
  # Initialize the counter for the current disease
  disease_image_counts[disease] = 0

  # Get the path to the directory for the current disease
  disease_dir = os.path.join(augmented_dir, disease)

  # Loop through each file in the directory
  for filename in os.listdir(disease_dir):
    # Check if the file is an image
    if filename.endswith(('.jpg', '.png', '.jpeg')):
      # Increment the counter for the current disease
      disease_image_counts[disease] += 1

# Print the number of images for each disease
for disease, count in disease_image_counts.items():
  print(f"Disease: {disease}, Number of Images: {count}")
```

```
Disease: c0, Number of Images: 238
Disease: c1, Number of Images: 867
Disease: c2, Number of Images: 468
Disease: c3, Number of Images: 694
```

```
import matplotlib.pyplot as plt

# Extract the disease names and image counts from the dictionary
disease_names = list(disease_image_counts.keys())
image_counts = list(disease_image_counts.values())

# Create a bar chart
plt.bar(disease_names, image_counts)
plt.xlabel("Disease")
plt.ylabel("Number of Images")
plt.title("Number of Images after Data Augmentation")
plt.show()
```



```
model = Sequential()
model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(256, 256, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(4, activation='softmax'))


model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_42 (Conv2D)          (None, 254, 254, 16)      448

 max_pooling2d_33 (MaxPooli  (None, 127, 127, 16)      0
 ng2D)

 conv2d_43 (Conv2D)          (None, 125, 125, 64)      9280

 max_pooling2d_34 (MaxPooli  (None, 62, 62, 64)        0
 ng2D)

 conv2d_44 (Conv2D)          (None, 60, 60, 64)        36928

 max_pooling2d_35 (MaxPooli  (None, 30, 30, 64)        0
 ng2D)

 conv2d_45 (Conv2D)          (None, 28, 28, 32)        18464

 flatten_12 (Flatten)        (None, 25088)             0

 dense_27 (Dense)            (None, 16)                401424

 dropout_10 (Dropout)        (None, 16)                0

 dense_28 (Dense)            (None, 4)                 68

=================================================================
Total params: 466612 (1.78 MB)
Trainable params: 466612 (1.78 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from tensorflow.keras.optimizers import Adadelta
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.004), loss='categorical_crossentropy', metrics=['accuracy'])


# Define the training and validation generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Edadonedataset/c0',
    target_size=(256,256),
    batch_size=64,
    class_mode='categorical',
)

validation_generator = test_datagen.flow_from_directory(
    '/content/drive/My Drive/Unsl',
    target_size=(256,256),
    batch_size=32,
    class_mode='categorical',
)
```

```
Found 2267 images belonging to 4 classes.
Found 196 images belonging to 4 classes.
```

```
# Train the model
history=model.fit(
    train_generator,
    epochs=15,
    validation_data=validation_generator,
)
```

```
Epoch 1/15
36/36 [==============================] - 237s 7s/step - loss: 1.3239 - accuracy: 0.3754 - val_loss: 1.2951 - val_accuracy: 0.3878
Epoch 2/15
36/36 [==============================] - 227s 6s/step - loss: 1.3094 - accuracy: 0.3555 - val_loss: 1.2903 - val_accuracy: 0.3878
Epoch 3/15
36/36 [==============================] - 221s 6s/step - loss: 1.3052 - accuracy: 0.3824 - val_loss: 1.2740 - val_accuracy: 0.3878
Epoch 4/15
36/36 [==============================] - 224s 6s/step - loss: 1.2745 - accuracy: 0.3789 - val_loss: 1.2680 - val_accuracy: 0.3878
Epoch 5/15
36/36 [==============================] - 226s 6s/step - loss: 1.2452 - accuracy: 0.4332 - val_loss: 1.2526 - val_accuracy: 0.4439
Epoch 6/15
36/36 [==============================] - 223s 6s/step - loss: 1.2197 - accuracy: 0.4486 - val_loss: 1.1452 - val_accuracy: 0.5153
Epoch 7/15
36/36 [==============================] - 228s 6s/step - loss: 1.1919 - accuracy: 0.4768 - val_loss: 1.0989 - val_accuracy: 0.5255
Epoch 8/15
36/36 [==============================] - 223s 6s/step - loss: 1.1552 - accuracy: 0.4817 - val_loss: 1.0811 - val_accuracy: 0.5408
Epoch 9/15
36/36 [==============================] - 229s 6s/step - loss: 1.1577 - accuracy: 0.4808 - val_loss: 1.1148 - val_accuracy: 0.5102
Epoch 10/15
36/36 [==============================] - 224s 6s/step - loss: 1.2160 - accuracy: 0.4685 - val_loss: 1.3247 - val_accuracy: 0.3827
Epoch 11/15
36/36 [==============================] - 224s 6s/step - loss: 1.2884 - accuracy: 0.4010 - val_loss: 1.1924 - val_accuracy: 0.4133
Epoch 12/15
36/36 [==============================] - 225s 6s/step - loss: 1.2107 - accuracy: 0.4318 - val_loss: 1.1335 - val_accuracy: 0.5153
Epoch 13/15
36/36 [==============================] - 228s 6s/step - loss: 1.3117 - accuracy: 0.4266 - val_loss: 1.2395 - val_accuracy: 0.4337
Epoch 14/15
36/36 [==============================] - 224s 6s/step - loss: 1.2424 - accuracy: 0.4341 - val_loss: 1.1322 - val_accuracy: 0.5357
Epoch 15/15
36/36 [==============================] - 227s 6s/step - loss: 1.1702 - accuracy: 0.4817 - val_loss: 1.0751 - val_accuracy: 0.5459
```

```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(validation_generator)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
7/7 [==============================] - 7s 954ms/step - loss: 1.0751 - accuracy: 0.5459
```