```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & R
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px

import os
import random
import shutil
from pathlib import Path
from PIL import ImageFile, Image
ImageFile.LOAD_TRUNCATED_IMAGES = True

from time import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tqdm.auto import tqdm

%matplotlib inline



wikiart_dataset_path = "/kaggle/input/wikiart-all-artpieces/wikiart/wikiart"

count = 0
image_mode_not_rbg_count = 0

for _, _, filesnames in os.walk(wikiart_dataset_path):
  print("---")
  # print(filesnames)
  for files in range(len(filesnames)):
    files = filesnames[files]
    # print(f"Images: {files}")
    try:
      size = os.path.getsize(os.path.join(wikiart_dataset_path, files))
      # print(f"size: {size}")
      if size == 0:
        print(files)

      image = Image.open(os.path.join(wikiart_dataset_path, files))
      image_mode = image.mode

      # print(f"Image Mode: {image_mode}")

      if image_mode != "RGB":
          image_mode_not_rbg_count = image_mode_not_rbg_count + 1
          image = image.convert("RGB")
          image.save(os.path.join(wikiart_dataset_path, files))

    except OSError as e:
        count = count + 1
#         print(f"Image file: {files} is truncated, exception raised: {e}")
```

```
print(f"\nImages converted to RGB: {image_mode_not_rbg_count}")
print(f"Corrupted Images Count: {count}")
```

```
---
/usr/local/lib/python3.10/dist-packages/PIL/Image.py:1054: UserWarning: Palette images with Transparency expressed in bytes should be cc
  warnings.warn(

Images converted to RGB: 2374
Corrupted Images Count: 2374
```

Creating Dataframe

```
wikiart_df = pd.read_csv("/kaggle/input/wikiart-all-artpieces/wikiart_art_pieces.csv")
wikiart_df.head(5)
```

	artist	style	genre	movement	tags	url	img	
0	Byzantine Mosaics	Early Byzantine (c. 330–750)	religious painting	Byzantine Art	['Holyplaces', 'Byzantinearchitecture', 'Arch'...	https://www.wikiart.org/en/byzantine-mosaics/e...	https://uploads2.wikiart.org/00211/images/byza...	a
1	Byzantine Mosaics	Early Byzantine (c. 330–750)	religious painting	Byzantine Art	['Holyplaces', 'Byzantinearchitecture', 'Arch']	https://www.wikiart.org/en/byzantine-mosaics/e...	https://uploads2.wikiart.org/00211/images/byza...	a
2	Byzantine Mosaics	Early Byzantine (c. 330–750)	religious painting	Byzantine Art	['Prophet', 'History']	https://www.wikiart.org/en/byzantine-mosaics/e...	https://uploads2.wikiart.org/00211/images/byza...	a

```
print(f"Shape of wikiart dataframe: {wikiart_df.shape}")
```

```
Shape of wikiart dataframe: (176436, 8)
```

```
wikiart_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 176436 entries, 0 to 176435
Data columns (total 8 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   artist     176436 non-null  object
 1   style      176436 non-null  object
 2   genre      176436 non-null  object
 3   movement   176436 non-null  object
 4   tags       127710 non-null  object
 5   url        176436 non-null  object
 6   img        176436 non-null  object
 7   file_name  176436 non-null  object
dtypes: object(8)
memory usage: 10.8+ MB
```

```
# Checking null values
wikiart_df.isnull().sum()
```

```
artist           0
style            0
genre            0
movement         0
tags         48726
url              0
img              0
file_name        0
dtype: int64
```

```
print(f'Uniques Value of Movement column: {len(wikiart_df["movement"].unique())}')
print(f'Uniques Value of Artist column: {len(wikiart_df["artist"].unique())}')
print(f'Uniques Value of Style column: {len(wikiart_df["style"].unique())}')
print(f'Uniques Value of Genre column: {len(wikiart_df["genre"].unique())}')
```

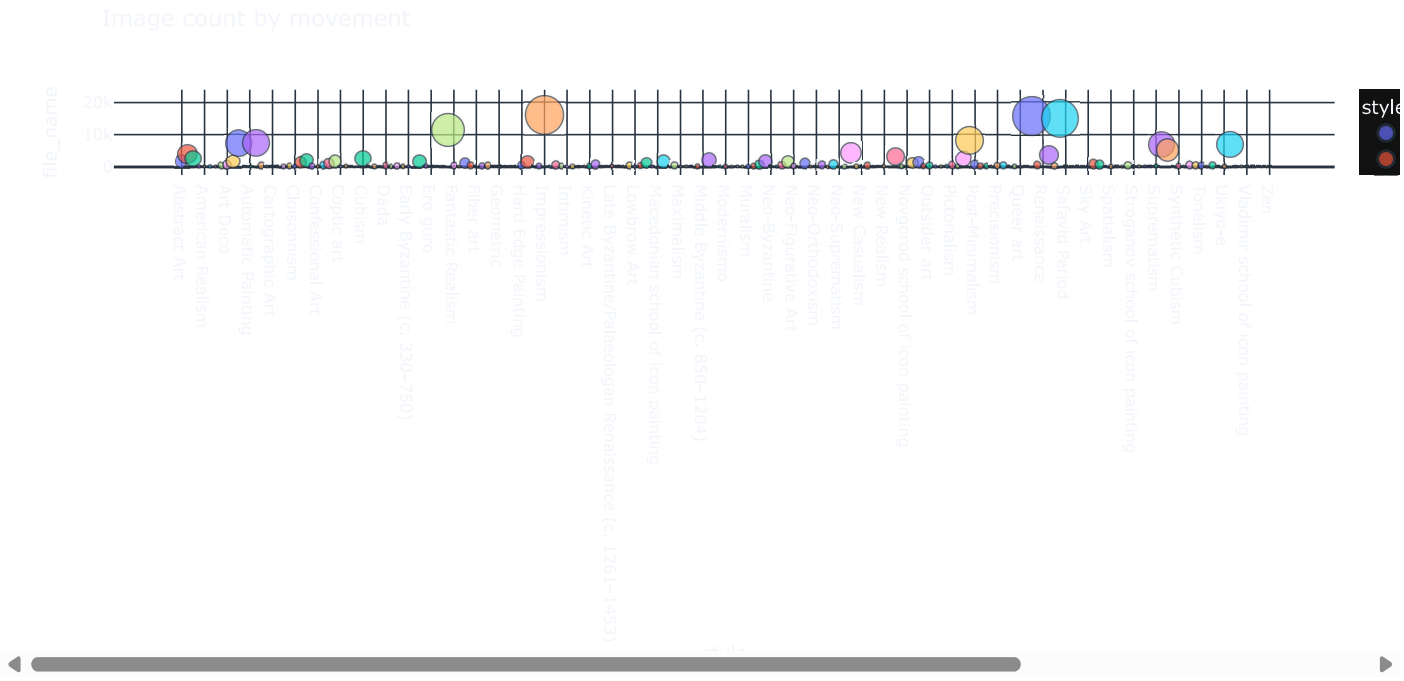
```
Uniques Value of Movement column: 124
Uniques Value of Artist column: 3209
Uniques Value of Style column: 193
Uniques Value of Genre column: 646
```

Counting of images by style column

```
image_file_count_by_style = wikiart_df.groupby("style")["file_name"].count()

df = image_file_count_by_style.to_frame()

df.head()
```

style	file_name
Abstract Art	1729
Abstract Expressionism	3909
Academicism	2474
Action painting	105
American Realism	72

```
import plotly.io as pio

# Ensure file_name is numeric for size argument
df["file_count"] = df["file_name"].astype(float)

fig = px.scatter(df, x=df.index, y="file_name",
                 template="plotly_dark", size="file_count", color=df.index,
                 title="Image count by movement")

pio.show(fig)  # Explicitly show the plot
```



From the above plot we can see that:

- There are images > 5000 for 8 style
- There are many images < 1000 for many style
- So, even with data augmentation of images < 1000 will not effect on training data and accuracy for style < 1000 images will be low

```
df = df[(df["file_name"] > 2000) & (df["file_name"] < 5000)]
```

```
style_list = df.index.to_list()
print(f"Selected Style Lists: {style_list}")
```

```
Selected Style Lists: ['Abstract Expressionism', 'Academicism', 'Cubism', 'Minimalism', 'Neoclassicism', 'Northern Renaissance', 'Pop Ar
```

Creating new DataFrame and random sampling the dataframe

```
new_wikiart_df = wikiart_df.loc[wikiart_df["style"].isin(df.index)]
new_wikiart_df = new_wikiart_df.sample(frac=1, random_state=7)
new_wikiart_df.head()
```

	artist	style	genre	movement	tags	url	img
162042	Tadanori Yokoo	Pop Art	poster	Pop Art	NaN	https://www.wikiart.org/en/tadanori-yokoo/b-1-...	https://uploads6.wikiart.org/images/tadanori-y...
124819	Joan Mitchell	Abstract Expressionism	abstract	Abstract Art	NaN	https://www.wikiart.org/en/joan-mitchell/untit...	https://uploads7.wikiart.org/images/joan-mitch...
7860	Albrecht Durer	Northern Renaissance	religious painting	Northern Renaissance	['Christianity', 'saints-and-apostles', 'Jesus...	https://www.wikiart.org/en/albrecht-durer/st-c...	https://uploads5.wikiart.org/images/albrecht-d...

```
new_wikiart_df.shape
```

```
(24493, 8)
```

```
current_directory = "/kaggle/working"
print(f"Current Directory: {current_directory}")

# Define dataset folder
new_folder = "Dataset"
final_directory = os.path.join(current_directory, new_folder)

# Create dataset directory
os.makedirs(final_directory, exist_ok=True)
print(f"Dataset Directory Created: {final_directory}")

# Define subdirectories
subfolders = ["Train", "Test", "Val"]
for folder in subfolders:
    folder_path = os.path.join(final_directory, folder)
    os.makedirs(folder_path, exist_ok=True)
    print(f"Directory Created: {folder_path}")
```

```
Current Directory: /kaggle/working
Dataset Directory Created: /kaggle/working/Dataset
Directory Created: /kaggle/working/Dataset/Train
Directory Created: /kaggle/working/Dataset/Test
Directory Created: /kaggle/working/Dataset/Val
```

```
style_list = new_wikiart_df["style"].unique().tolist()
print(f"Count of unique Style: {len(style_list)}")
```

```
Count of unique Style: 8
```

```
current_directory = "/kaggle/working"
dataset_folder = os.path.join(current_directory, "Dataset")

# Create dataset structure
train_folder = os.path.join(dataset_folder, "Train")
val_folder = os.path.join(dataset_folder, "Val")
test_folder = os.path.join(dataset_folder, "Test")

for folder in [train_folder, val_folder, test_folder]:
    os.makedirs(folder, exist_ok=True)

# Define train/val/test split ratios
```

```
train_ratio = 0.7
val_ratio = 0.15

# Path to WikiArt dataset
wikiart_images_path = "/kaggle/input/wikiart-all-artpieces/wikiart/wikiart"
image_files = set(os.listdir(wikiart_images_path))

# Iterate over all unique styles
style_list = new_wikiart_df["style"].unique().tolist()

for style in style_list:
    print(f"Processing Style: {style}")

    # Get all images of this style
    images = new_wikiart_df[new_wikiart_df["style"] == style]["file_name"].tolist()
    total_images = len(images)

    print(f"Total Images: {total_images}")

    # Split into train, validation, and test
    train_count = round(total_images * train_ratio)
    val_count = round(total_images * val_ratio)
    test_count = total_images - (train_count + val_count)

    print(f"Train: {train_count}, Validation: {val_count}, Test: {test_count}")

    # Create style-specific directories
    train_style_dir = os.path.join(train_folder, style)
    val_style_dir = os.path.join(val_folder, style)
    test_style_dir = os.path.join(test_folder, style)

    for folder in [train_style_dir, val_style_dir, test_style_dir]:
        os.makedirs(folder, exist_ok=True)

    # Function to copy images safely
    def copy_images(image_list, source_folder, destination_folder):
        copied = 0
        for img in image_list:
            if img in image_files:
                src_path = os.path.join(source_folder, img)
                dest_path = os.path.join(destination_folder, img)
                shutil.copy(src_path, dest_path)
                copied += 1
        return copied

    # Copy images to respective folders
    train_copied = copy_images(images[:train_count], wikiart_images_path, train_style_dir)
    val_copied = copy_images(images[train_count:train_count + val_count], wikiart_images_path, val_style_dir)
    test_copied = copy_images(images[train_count + val_count:], wikiart_images_path, test_style_dir)

    print(f"Copied {train_copied} to Train, {val_copied} to Validation, {test_copied} to Test\n")
```

```
Processing Style: Pop Art
Total Images: 2517
Train: 1762, Validation: 378, Test: 377
Copied 1762 to Train, 378 to Validation, 377 to Test

Processing Style: Abstract Expressionism
Total Images: 3909
Train: 2736, Validation: 586, Test: 587
Copied 2736 to Train, 586 to Validation, 587 to Test

Processing Style: Northern Renaissance
Total Images: 3089
Train: 2162, Validation: 463, Test: 464
Copied 2162 to Train, 463 to Validation, 464 to Test

Processing Style: Cubism
Total Images: 2530
Train: 1771, Validation: 380, Test: 379
Copied 1771 to Train, 380 to Validation, 379 to Test

Processing Style: Rococo
Total Images: 3537
Train: 2476, Validation: 531, Test: 530
Copied 2476 to Train, 531 to Validation, 530 to Test

Processing Style: Academicism
Total Images: 2474
```

```
Train: 1732, Validation: 371, Test: 371
Copied 1732 to Train, 371 to Validation, 371 to Test

Processing Style: Neoclassicism
Total Images: 4360
Train: 3052, Validation: 654, Test: 654
Copied 3052 to Train, 654 to Validation, 654 to Test

Processing Style: Minimalism
Total Images: 2077
Train: 1454, Validation: 312, Test: 311
Copied 1454 to Train, 312 to Validation, 311 to Test
```

```
train_directory = os.path.join("/kaggle/working/Dataset", "Train")

folders_list = [folder for folder in os.listdir(train_directory) if os.path.isdir(os.path.join(train_directory, folder))]

print(f"Folders Count: {len(folders_list)}")
```

```
Folders Count: 8
```

```
def walk_through_directory(dir_path):
    """
    Prints the number of folders and images in the given directory.
    """
    for path, dirnames, filenames in os.walk(dir_path):
        print(f"There are {len(dirnames)} folders and {len(filenames)} painting images in {path}.")



walk_through_directory("/kaggle/working/Dataset")
```

```
There are 3 folders and 0 painting images in /kaggle/working/Dataset.
There are 8 folders and 0 painting images in /kaggle/working/Dataset/Test.
There are 0 folders and 587 painting images in /kaggle/working/Dataset/Test/Abstract Expressionism.
There are 0 folders and 654 painting images in /kaggle/working/Dataset/Test/Neoclassicism.
There are 0 folders and 377 painting images in /kaggle/working/Dataset/Test/Pop Art.
There are 0 folders and 379 painting images in /kaggle/working/Dataset/Test/Cubism.
There are 0 folders and 371 painting images in /kaggle/working/Dataset/Test/Academicism.
There are 0 folders and 530 painting images in /kaggle/working/Dataset/Test/Rococo.
There are 0 folders and 311 painting images in /kaggle/working/Dataset/Test/Minimalism.
There are 0 folders and 464 painting images in /kaggle/working/Dataset/Test/Northern Renaissance.
There are 8 folders and 0 painting images in /kaggle/working/Dataset/Val.
There are 0 folders and 586 painting images in /kaggle/working/Dataset/Val/Abstract Expressionism.
There are 0 folders and 654 painting images in /kaggle/working/Dataset/Val/Neoclassicism.
There are 0 folders and 378 painting images in /kaggle/working/Dataset/Val/Pop Art.
There are 0 folders and 380 painting images in /kaggle/working/Dataset/Val/Cubism.
There are 0 folders and 371 painting images in /kaggle/working/Dataset/Val/Academicism.
There are 0 folders and 531 painting images in /kaggle/working/Dataset/Val/Rococo.
There are 0 folders and 312 painting images in /kaggle/working/Dataset/Val/Minimalism.
There are 0 folders and 463 painting images in /kaggle/working/Dataset/Val/Northern Renaissance.
There are 8 folders and 0 painting images in /kaggle/working/Dataset/Train.
There are 0 folders and 2736 painting images in /kaggle/working/Dataset/Train/Abstract Expressionism.
There are 0 folders and 3052 painting images in /kaggle/working/Dataset/Train/Neoclassicism.
There are 0 folders and 1762 painting images in /kaggle/working/Dataset/Train/Pop Art.
There are 0 folders and 1771 painting images in /kaggle/working/Dataset/Train/Cubism.
There are 0 folders and 1732 painting images in /kaggle/working/Dataset/Train/Academicism.
There are 0 folders and 2476 painting images in /kaggle/working/Dataset/Train/Rococo.
There are 0 folders and 1454 painting images in /kaggle/working/Dataset/Train/Minimalism.
There are 0 folders and 2162 painting images in /kaggle/working/Dataset/Train/Northern Renaissance.
```

Visualizing the images

```
dataset_directory = Path("/kaggle/working/Dataset")

# Get all image paths (searching inside Train, Test, and Val)
image_path_list = list(dataset_directory.glob("*/*/*.jpg"))

# Ensure there are images available
if image_path_list:
    # Select a random image
    image_path = random.choice(image_path_list)

    # Get the class (folder name) of the image
    image_class = image_path.parent.stem  # Extracts folder name

    # Open image
```

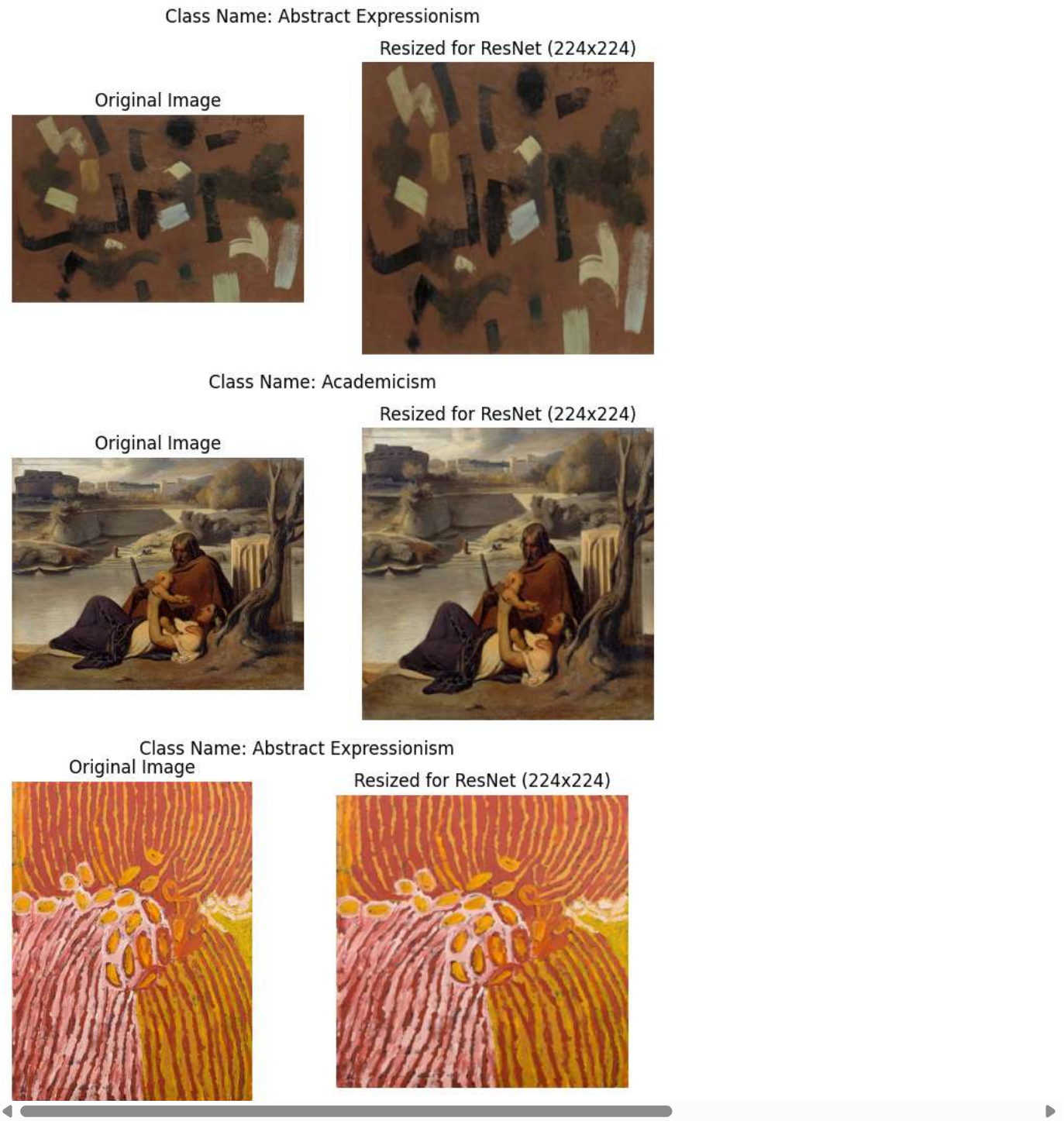
```
    img = Image.open(image_path)
    img_array = np.asarray(img)

    # Display image
    plt.figure(figsize=(12, 5))
    plt.imshow(img_array)
    plt.title(f"Image Class: {image_class} | Image Shape: {img_array.shape}")
    plt.axis(False)
    plt.show()
else:
    print("No images found in the dataset directory.")
```

Model Training

```
dataset_directory = Path("/kaggle/working/Dataset")
train_directory = dataset_directory / "Train"
val_directory = dataset_directory / "Val"
test_directory = dataset_directory / "Test"

# Image size for ResNet
IMG_SIZE = (224, 224)
BATCH_SIZE = 32

# Load datasets using TensorFlow
train_data = tf.keras.preprocessing.image_dataset_from_directory(
    train_directory,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    label_mode="int"  # Classes are inferred from folder structure
)

val_data = tf.keras.preprocessing.image_dataset_from_directory(
    val_directory,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    label_mode="int"
)

test_data = tf.keras.preprocessing.image_dataset_from_directory(
    test_directory,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    label_mode="int"
)
```

```
Found 17145 files belonging to 8 classes.
Found 3675 files belonging to 8 classes.
Found 3673 files belonging to 8 classes.
```

```
    # Function to plot transformed images
    def plot_images(image_path_list, n=3, seed=None):
        """ Select random images, apply resizing transformation, and plot original vs transformed images """

        if seed:
            random.seed(seed)

        image_samples = random.sample(image_path_list, k=n)

        for img_path in image_samples:
            with Image.open(img_path) as img:
                fig, ax = plt.subplots(1, 2, figsize=(8, 4))

                # Original Image
                ax[0].imshow(img)
                ax[0].set_title("Original Image")
                ax[0].axis(False)

                # Resized Image (Simulating TensorFlow Resizing)
                img_resized = img.resize(IMG_SIZE)
                ax[1].imshow(img_resized)
                ax[1].set_title("Resized for ResNet (224x224)")
                ax[1].axis(False)

                fig.suptitle(f"Class Name: {img_path.parent.stem}", fontsize=12)

    # Get all image paths
    image_path_list = list(dataset_directory.glob("*/*/*.jpg"))

    # Plot images
    plot_images(image_path_list, n=3, seed=42)
```




```
class_names = train_data.class_names
print(f"List of Class Names: {class_names}")
```

```
List of Class Names: ['Abstract Expressionism', 'Academicism', 'Cubism', 'Minimalism', 'Neoclassicism', 'Northern Renaissance', 'Pop Art
```

Training on ResNet-50 by unfreezing the parameters

```
from tensorflow.keras.applications import ResNet50
# Load pre-trained ResNet50 model without the top layer
base_model = tf.keras.applications.ResNet50(
    include_top=False, weights="imagenet", input_shape=(224, 224, 3)
)

# Freeze all layers (initially)
```

```
base_model.trainable = False

# Add new classification head
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(256, activation="relu"),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(len(class_names), activation="softmax")  # Multi-class classification
])

# Compile model (initial training with frozen layers)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

# Summary of the model
model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_no
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Model: "sequential"
```

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 8)	2,056

```
Total params: 24,114,312 (91.99 MB)
```

```
history_frozen = model.fit(
    train_data, validation_data=val_data, epochs=5
)
```

```
Epoch 1/5
536/536 ━━━━━━━━━━━━━━━━━━━━ 51s 74ms/step - accuracy: 0.5119 - loss: 1.3344 - val_accuracy: 0.6610 - val_loss: 0.9071
Epoch 2/5
536/536 ━━━━━━━━━━━━━━━━━━━━ 30s 56ms/step - accuracy: 0.6581 - loss: 0.8891 - val_accuracy: 0.6833 - val_loss: 0.8370
Epoch 3/5
536/536 ━━━━━━━━━━━━━━━━━━━━ 30s 56ms/step - accuracy: 0.6976 - loss: 0.7927 - val_accuracy: 0.7097 - val_loss: 0.7914
Epoch 4/5
536/536 ━━━━━━━━━━━━━━━━━━━━ 30s 55ms/step - accuracy: 0.7175 - loss: 0.7383 - val_accuracy: 0.7105 - val_loss: 0.7601
Epoch 5/5
536/536 ━━━━━━━━━━━━━━━━━━━━ 30s 56ms/step - accuracy: 0.7343 - loss: 0.6875 - val_accuracy: 0.7053 - val_loss: 0.7766
```

```
# Unfreeze the last 50% of layers
for layer in base_model.layers[-75:]:
    layer.trainable = True  # Allow training

# Recompile model with a lower learning rate for fine-tuning
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),  # Lower LR
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

# Model summary after unfreezing layers
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 8)	2,056

```
Total params: 24,114,312 (91.99 MB)
```

```
# Early stopping to prevent overfitting
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=10, restore_best_weights=True
)

# Reduce learning rate if validation loss plateaus
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss", factor=0.5, patience=3, min_lr=1e-6
)

# Model checkpoint: Save best model
checkpoint = tf.keras.callbacks.ModelCheckpoint(
    "best_resnet_finetuned.keras", monitor="val_loss", save_best_only=True
)

callbacks = [early_stopping, reduce_lr, checkpoint]



history_finetune = model.fit(
    train_data, validation_data=val_data, epochs=50, callbacks=callbacks
)
```

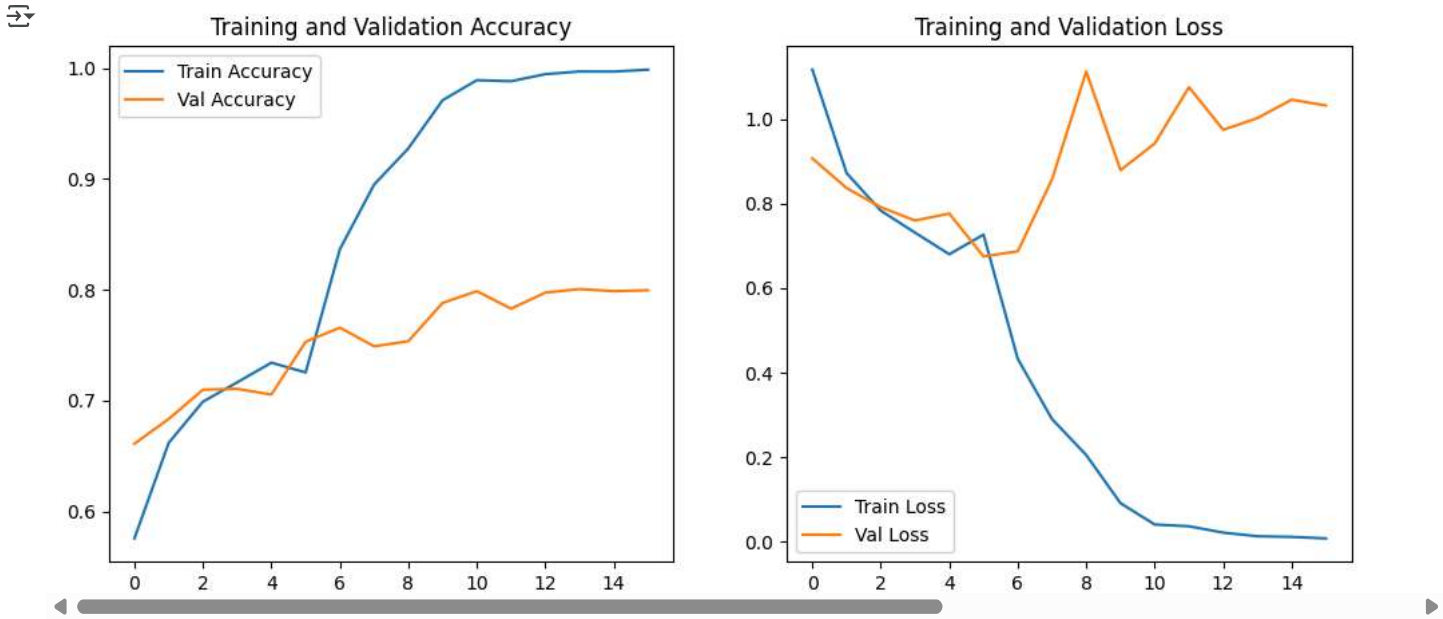
```
Epoch 1/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 97s 129ms/step - accuracy: 0.7056 - loss: 0.7935 - val_accuracy: 0.7529 - val_loss: 0.6748 - learning_rate:
Epoch 2/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 52s 97ms/step - accuracy: 0.8338 - loss: 0.4436 - val_accuracy: 0.7657 - val_loss: 0.6872 - learning_rate:
Epoch 3/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 52s 97ms/step - accuracy: 0.8904 - loss: 0.3003 - val_accuracy: 0.7488 - val_loss: 0.8574 - learning_rate:
Epoch 4/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 52s 97ms/step - accuracy: 0.9252 - loss: 0.2133 - val_accuracy: 0.7535 - val_loss: 1.1131 - learning_rate:
Epoch 5/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 52s 97ms/step - accuracy: 0.9615 - loss: 0.1178 - val_accuracy: 0.7880 - val_loss: 0.8793 - learning_rate:
Epoch 6/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 52s 97ms/step - accuracy: 0.9882 - loss: 0.0440 - val_accuracy: 0.7986 - val_loss: 0.9423 - learning_rate:
Epoch 7/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 52s 97ms/step - accuracy: 0.9865 - loss: 0.0360 - val_accuracy: 0.7829 - val_loss: 1.0758 - learning_rate:
Epoch 8/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 52s 97ms/step - accuracy: 0.9929 - loss: 0.0263 - val_accuracy: 0.7973 - val_loss: 0.9749 - learning_rate:
Epoch 9/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 52s 97ms/step - accuracy: 0.9967 - loss: 0.0142 - val_accuracy: 0.8005 - val_loss: 1.0026 - learning_rate:
Epoch 10/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 52s 97ms/step - accuracy: 0.9970 - loss: 0.0106 - val_accuracy: 0.7986 - val_loss: 1.0463 - learning_rate:
Epoch 11/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 52s 97ms/step - accuracy: 0.9983 - loss: 0.0088 - val_accuracy: 0.7995 - val_loss: 1.0323 - learning_rate:
```

```
# Evaluate on test set
test_loss, test_acc = model.evaluate(test_data)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 7s 61ms/step - accuracy: 0.7564 - loss: 0.6742
Test Accuracy: 0.7593
```

```
# Combine both training phases
acc = history_frozen.history["accuracy"] + history_finetune.history["accuracy"]
val_acc = history_frozen.history["val_accuracy"] + history_finetune.history["val_accuracy"]
loss = history_frozen.history["loss"] + history_finetune.history["loss"]
val_loss = history_frozen.history["val_loss"] + history_finetune.history["val_loss"]

epochs_range = range(len(acc))
```

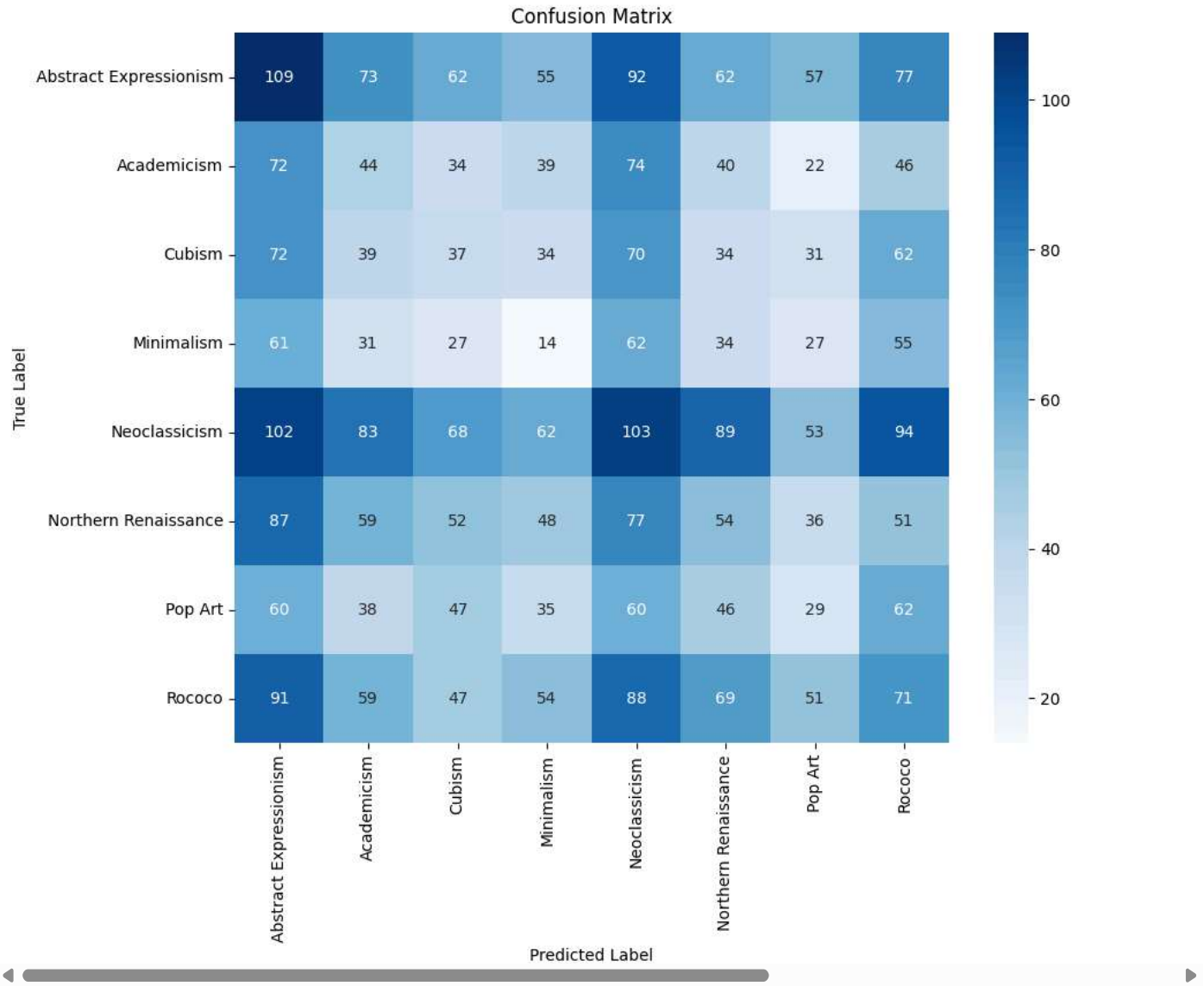
```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label="Train Accuracy")
plt.plot(epochs_range, val_acc, label="Val Accuracy")
plt.legend()
plt.title("Training and Validation Accuracy")

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label="Train Loss")
plt.plot(epochs_range, val_loss, label="Val Loss")
plt.legend()
plt.title("Training and Validation Loss")

plt.show()
```




```
predictions = model.predict(test_data)

# Convert to class labels
predicted_labels = np.argmax(predictions, axis=1)

# Get true labels
true_labels = np.concatenate([y for _, y in test_data], axis=0)
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 10s 64ms/step
```

```
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Compute confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Get class names from the dataset
class_names = list(test_data.class_names)  # Ensure test_data has class names

# Plot confusion matrix using Seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



Training on EfficientNetB0 by unfreezing the parameters

```
# Load EfficientNetB0 with pre-trained weights
base_model = tf.keras.applications.EfficientNetB0(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)

# Freeze the base model initially
base_model.trainable = False

# Add new classification head
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(256, activation="relu"),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(len(class_names), activation="softmax")  # Multi-class classification
])

# Compile model (initial training with frozen layers)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

# Summary of the model
```

```
model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,049,571
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1280)	0
dense_3 (Dense)	(None, 256)	327,936
dropout_2 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 8)	2,056

```
Total params: 4,379,563 (16.71 MB)
```

```
history = model.fit(train_data, validation_data=val_data, epochs=5)
```

```
Epoch 1/5
536/536 ━━━━━━━━━━━━━━━━━━━━ 61s 79ms/step - accuracy: 0.5650 - loss: 1.1394 - val_accuracy: 0.7007 - val_loss: 0.7872
Epoch 2/5
536/536 ━━━━━━━━━━━━━━━━━━━━ 21s 39ms/step - accuracy: 0.7104 - loss: 0.7649 - val_accuracy: 0.7235 - val_loss: 0.7214
Epoch 3/5
536/536 ━━━━━━━━━━━━━━━━━━━━ 23s 43ms/step - accuracy: 0.7545 - loss: 0.6454 - val_accuracy: 0.7263 - val_loss: 0.7124
Epoch 4/5
536/536 ━━━━━━━━━━━━━━━━━━━━ 21s 40ms/step - accuracy: 0.7733 - loss: 0.5966 - val_accuracy: 0.7371 - val_loss: 0.7034
Epoch 5/5
536/536 ━━━━━━━━━━━━━━━━━━━━ 21s 39ms/step - accuracy: 0.7998 - loss: 0.5313 - val_accuracy: 0.7420 - val_loss: 0.7011
```

```
# Unfreeze the last 50% of layers
for layer in base_model.layers[:-25]:
    layer.trainable = True  # Allow training

# Recompile model with a lower learning rate for fine-tuning
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),  # Lower LR
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

# Model summary after unfreezing layers
model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,049,571
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1280)	0
dense_3 (Dense)	(None, 256)	327,936
dropout_2 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 8)	2,056

```
Total params: 4,379,563 (16.71 MB)
```

```
# Early stopping to prevent overfitting
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=10, restore_best_weights=True
)

# Reduce learning rate if validation loss plateaus
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss", factor=0.5, patience=3, min_lr=1e-6
)

# Model checkpoint: Save best model
checkpoint = tf.keras.callbacks.ModelCheckpoint(
```

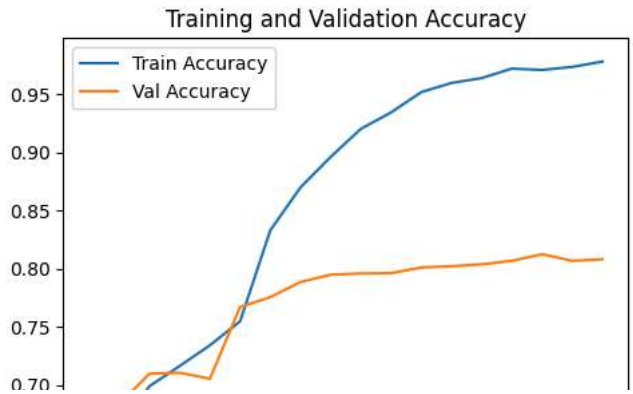
```
    "efficientnet_finetuned.keras", monitor="val_loss", save_best_only=True
)

callbacks = [early_stopping, reduce_lr, checkpoint]



history_finetune = model.fit(
    train_data, validation_data=val_data, epochs=50, callbacks=callbacks
)
```

```
Epoch 1/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 173s 207ms/step - accuracy: 0.7126 - loss: 0.7966 - val_accuracy: 0.7671 - val_loss: 0.6302 - learning_rate
Epoch 2/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 68s 126ms/step - accuracy: 0.8301 - loss: 0.4580 - val_accuracy: 0.7755 - val_loss: 0.6140 - learning_rate:
Epoch 3/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 68s 126ms/step - accuracy: 0.8629 - loss: 0.3601 - val_accuracy: 0.7886 - val_loss: 0.5966 - learning_rate:
Epoch 4/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 67s 124ms/step - accuracy: 0.8907 - loss: 0.2951 - val_accuracy: 0.7948 - val_loss: 0.6051 - learning_rate:
Epoch 5/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 66s 124ms/step - accuracy: 0.9176 - loss: 0.2341 - val_accuracy: 0.7959 - val_loss: 0.6117 - learning_rate:
Epoch 6/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 67s 124ms/step - accuracy: 0.9322 - loss: 0.1962 - val_accuracy: 0.7962 - val_loss: 0.6418 - learning_rate:
Epoch 7/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 67s 124ms/step - accuracy: 0.9495 - loss: 0.1479 - val_accuracy: 0.8011 - val_loss: 0.6416 - learning_rate:
Epoch 8/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 66s 124ms/step - accuracy: 0.9587 - loss: 0.1317 - val_accuracy: 0.8022 - val_loss: 0.6530 - learning_rate:
Epoch 9/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 67s 124ms/step - accuracy: 0.9630 - loss: 0.1139 - val_accuracy: 0.8038 - val_loss: 0.6510 - learning_rate:
Epoch 10/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 66s 124ms/step - accuracy: 0.9701 - loss: 0.0981 - val_accuracy: 0.8068 - val_loss: 0.6565 - learning_rate:
Epoch 11/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 66s 124ms/step - accuracy: 0.9701 - loss: 0.0938 - val_accuracy: 0.8125 - val_loss: 0.6670 - learning_rate:
Epoch 12/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 66s 124ms/step - accuracy: 0.9705 - loss: 0.0893 - val_accuracy: 0.8068 - val_loss: 0.6692 - learning_rate:
Epoch 13/50
536/536 ━━━━━━━━━━━━━━━━━━━━ 67s 125ms/step - accuracy: 0.9775 - loss: 0.0784 - val_accuracy: 0.8082 - val_loss: 0.6743 - learning_rate:
```

```
# Evaluate on test set
test_loss, test_acc = model.evaluate(test_data)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 7s 62ms/step - accuracy: 0.7881 - loss: 0.5860
Test Accuracy: 0.7925
```

```
# Combine both training phases
acc = history_frozen.history["accuracy"] + history_finetune.history["accuracy"]
val_acc = history_frozen.history["val_accuracy"] + history_finetune.history["val_accuracy"]
loss = history_frozen.history["loss"] + history_finetune.history["loss"]
val_loss = history_frozen.history["val_loss"] + history_finetune.history["val_loss"]

epochs_range = range(len(acc))

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label="Train Accuracy")
plt.plot(epochs_range, val_acc, label="Val Accuracy")
plt.legend()
plt.title("Training and Validation Accuracy")

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label="Train Loss")
plt.plot(epochs_range, val_loss, label="Val Loss")
plt.legend()
plt.title("Training and Validation Loss")

plt.show()
```




```
predictions = model.predict(test_data)

# Convert to class labels
predicted_labels = np.argmax(predictions, axis=1)

# Get true labels
true_labels = np.concatenate([y for _, y in test_data], axis=0)
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 13s 73ms/step
```

```
# Compute confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Get class names from the dataset
class_names = list(test_data.class_names)  # Ensure test_data has class names

# Plot confusion matrix using Seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```