AOL Deep Learning - Multiclass Image Problem - Indonesian Batik Motifs

1. Davin Edbert Santoso Halim - 2602067086
2. Felicia Andrea Tandoko - 26020593432
3. Steve Marcello Liem - 2602071410

Import Library

```
import os
import shutil
import random
import cv2
from PIL import Image
from PIL import UnidentifiedImageError

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import itertools

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import skimage
from skimage.feature import greycomatrix, greycoprops

import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.densenet import DenseNet121
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2

import warnings
warnings.filterwarnings("ignore")

from google.colab import drive


seed_value = 42
random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
```

Mount to Google Drive

```
drive.mount('/content/drive')
```

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

```
DATASET_PATH = "/content/drive/MyDrive/AOL/batik"
CLASS_NAMES = ['batik-parang', 'batik-megamendung', 'batik-kawung']
IMAGE_SIZE =  (224,224)
BATCH_SIZE = 32
```

Data Exploration

Print out the number of images per class

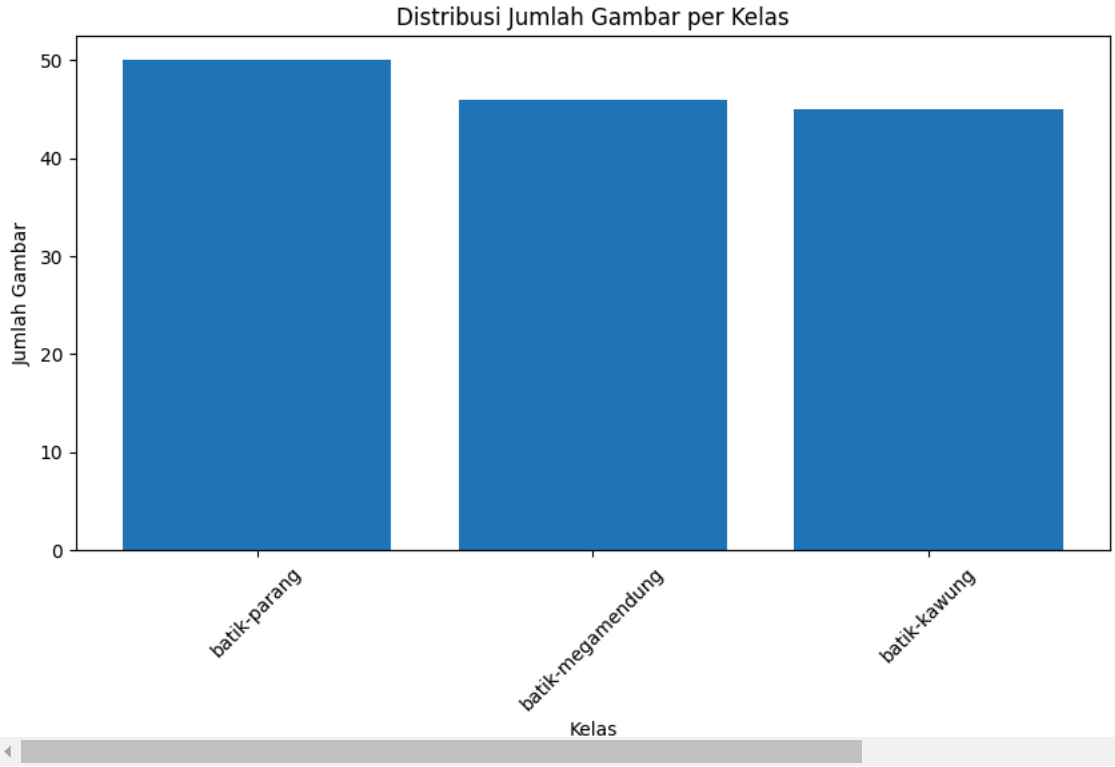
```
class_counts = {}
for class_name in CLASS_NAMES:
    class_dir = os.path.join(DATASET_PATH, class_name)
```

```
    count = len(os.listdir(class_dir))
    class_counts[class_name] = count

print("Jumlah gambar per kelas:")
for class_name, count in class_counts.items():
    print(f"{class_name}: {count}")
```

```
Jumlah gambar per kelas:
batik-parang: 50
batik-megamendung: 46
batik-kawung: 45
```

Visualize the distribution of the number of images per class using a histogram

```
plt.figure(figsize=(10, 5))
plt.bar(class_counts.keys(), class_counts.values())
plt.xlabel("Kelas")
plt.ylabel("Jumlah Gambar")
plt.title("Distribusi Jumlah Gambar per Kelas")
plt.xticks(rotation=45)
plt.show()
```



Print out image size statistics

```
image_sizes = []
for class_name in CLASS_NAMES:
    class_dir = os.path.join(DATASET_PATH, class_name)
    for image_name in os.listdir(class_dir):
        image_path = os.path.join(class_dir, image_name)
        with Image.open(image_path) as img:
            width, height = img.size
            image_sizes.append((width, height))

print("Statistik Ukuran Gambar:")
print(f"Ukuran Minimum: {min(image_sizes)}")
print(f"Ukuran Maksimum: {max(image_sizes)}")
print(f"Ukuran Rata-rata: {tuple(np.mean(image_sizes, axis=0))}")
```

```
Statistik Ukuran Gambar:
Ukuran Minimum: (229, 220)
Ukuran Maksimum: (3264, 2448)
Ukuran Rata-rata: (992.4822695035461, 885.7588652482269)
```

Visualize image samples from each class

```
def view_random_images(target_dir, target_class, num_images=1):
    # Setup target directory (we'll view images from here)
    target_folder = os.path.join(target_dir, target_class)

    # Get random image paths
    random_images = random.sample(os.listdir(target_folder), num_images)

    # Create a figure with subplots
    fig, axes = plt.subplots(1, num_images, figsize=(15, 5))

    for i, random_image in enumerate(random_images):
        # Read in the image and plot it using matplotlib
        img = mpimg.imread(os.path.join(target_folder, random_image))
        axes[i].imshow(img)
        image_shape = img.shape
        axes[i].set_title(f"{random_image}\nShape: {image_shape}")
        axes[i].axis("off")

    fig.suptitle(f"{num_images} Random Images from {target_class}")
    plt.show()

for class_name in CLASS_NAMES:
  view_random_images(DATASET_PATH, class_name, num_images=3)
```

Checking for the presence of damaged or unreadable images:

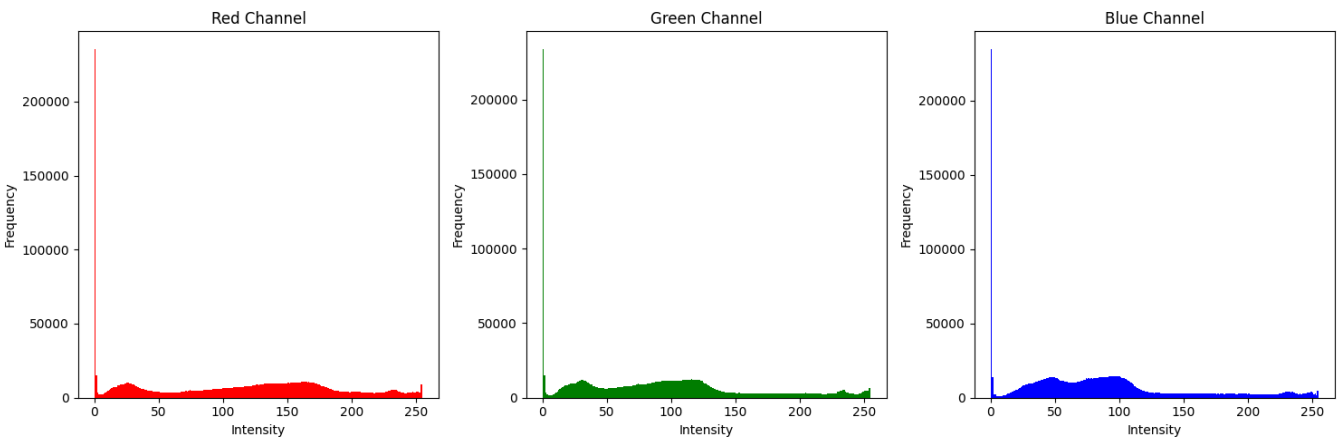
```
corrupt_images = []
for class_name in CLASS_NAMES:
    class_dir = os.path.join(DATASET_PATH, class_name)
    for image_name in os.listdir(class_dir):
        image_path = os.path.join(class_dir, image_name)
        try:
            with Image.open(image_path) as img:
```

```
                img.verify()
            except (IOError, SyntaxError) as e:
                corrupt_images.append(image_path)

print("Gambar yang rusak atau tidak dapat dibaca:")
for image_path in corrupt_images:
    print(image_path)
```

```
Gambar yang rusak atau tidak dapat dibaca:
```

Color distribution analysis

```
def plot_color_distribution(image_path):
    img = plt.imread(image_path)
    colors = img.reshape(-1, 3)

    fig, axs = plt.subplots(1, 3, figsize=(15, 5))
    axs = axs.ravel()

    for i, color in enumerate(['Red', 'Green', 'Blue']):
        axs[i].hist(colors[:, i], bins=256, color=color.lower())
        axs[i].set_title(f"{color} Channel")
        axs[i].set_xlabel("Intensity")
        axs[i].set_ylabel("Frequency")

    plt.tight_layout()
    plt.show()

image_path = os.path.join(DATASET_PATH, CLASS_NAMES[0], os.listdir(os.path.join(DATASET_PATH, CLASS_NAMES[0]))[0])
plot_color_distribution(image_path)
```



1. Kode tersebut menggunakan fungsi `plot_color_distribution` untuk memvisualisasikan distribusi intensitas warna pada sebuah gambar.
2. Gambar yang dianalisis diambil dari direktori dataset pada kelas pertama (`CLASS_NAMES[0]`) dan gambar pertama dalam direktori tersebut.
3. Output yang dihasilkan adalah tiga histogram yang menunjukkan distribusi intensitas warna pada setiap saluran warna (Red, Green, Blue) dari gambar yang dianalisis.
4. Setiap histogram menampilkan frekuensi (sumbu y) dari setiap nilai intensitas (sumbu x) pada masing-masing saluran warna.
5. Histogram untuk saluran warna Red (Merah):
    - Menunjukkan distribusi intensitas warna merah pada gambar.
    - Terdapat dua puncak utama pada histogram, satu di sekitar intensitas 50-100 dan satu lagi di sekitar intensitas 200-255.
    - Puncak pertama menunjukkan adanya piksel dengan intensitas merah rendah, sedangkan puncak kedua menunjukkan adanya piksel dengan intensitas merah tinggi.
6. Histogram untuk saluran warna Green (Hijau):
    - Menunjukkan distribusi intensitas warna hijau pada gambar.

- Histogram memiliki puncak yang lebar di sekitar intensitas 0-150, menunjukkan bahwa sebagian besar piksel memiliki intensitas hijau rendah hingga sedang.
- Terdapat sedikit puncak di sekitar intensitas 200-255, menunjukkan adanya beberapa piksel dengan intensitas hijau tinggi.

7. Histogram untuk saluran warna Blue (Biru):
    - Menunjukkan distribusi intensitas warna biru pada gambar.
    - Histogram memiliki puncak yang tinggi di sekitar intensitas 0-100, menunjukkan bahwa sebagian besar piksel memiliki intensitas biru rendah.
    - Terdapat sedikit puncak di sekitar intensitas 150-200, menunjukkan adanya beberapa piksel dengan intensitas biru sedang.

Secara keseluruhan, histogram-histogram tersebut memberikan informasi tentang distribusi intensitas warna pada gambar yang dianalisis. Puncak yang tinggi menunjukkan bahwa banyak piksel memiliki intensitas warna tertentu, sedangkan lembah atau area yang rendah menunjukkan sedikitnya piksel dengan intensitas tersebut.

Informasi distribusi warna ini dapat membantu dalam memahami karakteristik visual gambar, seperti dominasi warna tertentu, kontras, atau keseimbangan warna. Analisis distribusi warna juga dapat digunakan untuk membandingkan gambar-gambar dalam dataset dan mengidentifikasi pola atau perbedaan antara kelas-kelas yang berbeda.

Image texture analysis using GLCM (Gray Level Co-occurrence Matrix)

```
def analyze_texture(image_path):
    img = plt.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

    glcm = skimage.feature.graycomatrix(gray, distances=[1], angles=[0], levels=256, symmetric=True, normed=True)
    contrast = skimage.feature.graycoprops(glcm, 'contrast')[0, 0]
    dissimilarity = skimage.feature.graycoprops(glcm, 'dissimilarity')[0, 0]
    homogeneity = skimage.feature.graycoprops(glcm, 'homogeneity')[0, 0]
    energy = skimage.feature.graycoprops(glcm, 'energy')[0, 0]
    correlation = skimage.feature.graycoprops(glcm, 'correlation')[0, 0]

    print(f"Contrast: {contrast:.3f}")
    print(f"Dissimilarity: {dissimilarity:.3f}")
    print(f"Homogeneity: {homogeneity:.3f}")
    print(f"Energy: {energy:.3f}")
    print(f"Correlation: {correlation:.3f}")

image_path = os.path.join(DATASET_PATH, CLASS_NAMES[0], os.listdir(os.path.join(DATASET_PATH, CLASS_NAMES[0]))[0])
analyze_texture(image_path)
```

```
Contrast: 212.616
Dissimilarity: 8.229
Homogeneity: 0.306
Energy: 0.128
Correlation: 0.978
```

Hasil output dari analisis tekstur menggunakan Gray Level Co-occurrence Matrix (GLCM) pada gambar yang diberikan adalah sebagai berikut:

1. Contrast: 140.271
    - Nilai kontras yang tinggi menunjukkan bahwa gambar memiliki perbedaan intensitas yang besar antara piksel yang berdekatan.
    - Semakin tinggi nilai kontras, semakin besar perbedaan intensitas antara piksel yang berdekatan, menunjukkan adanya tepi atau batas yang jelas dalam gambar.
2. Dissimilarity: 7.593
    - Nilai dissimilarity mengukur perbedaan rata-rata intensitas antara pasangan piksel yang berdekatan.
    - Nilai dissimilarity yang lebih tinggi menunjukkan bahwa ada lebih banyak variasi atau perbedaan intensitas dalam gambar.
3. Homogeneity: 0.167
    - Nilai homogeneity mengukur keseragaman atau kesamaan intensitas dalam gambar.
    - Nilai homogeneity yang rendah (mendekati 0) menunjukkan bahwa gambar memiliki variasi intensitas yang tinggi dan tidak seragam.
    - Sebaliknya, nilai homogeneity yang tinggi (mendekati 1) menunjukkan bahwa gambar memiliki intensitas yang seragam dan sedikit variasi.
4. Energy: 0.016
    - Nilai energy mengukur keseragaman atau keteraturan tekstur dalam gambar.
    - Nilai energy yang rendah menunjukkan bahwa tekstur dalam gambar tidak teratur atau acak.
    - Sebaliknya, nilai energy yang tinggi menunjukkan bahwa tekstur dalam gambar seragam dan teratur.
5. Correlation: 0.980

- Nilai correlation mengukur ketergantungan linear antara pasangan piksel yang berdekatan.
- Nilai correlation yang tinggi (mendekati 1 atau -1) menunjukkan adanya hubungan linear yang kuat antara pasangan piksel, yang dapat menunjukkan adanya pola atau struktur dalam tekstur.
- Nilai correlation mendekati 0 menunjukkan bahwa tidak ada hubungan linear yang signifikan antara pasangan piksel.

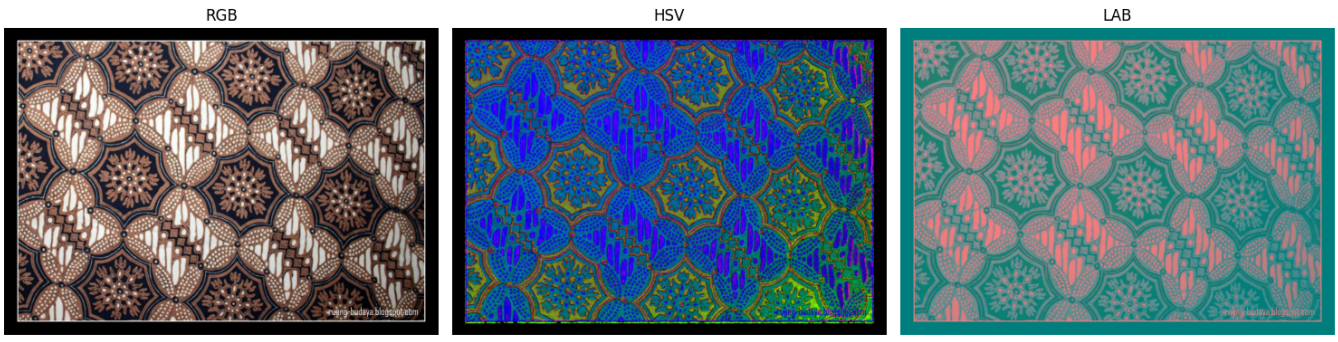
Berdasarkan hasil analisis tekstur di atas, dapat disimpulkan bahwa gambar yang dianalisis memiliki karakteristik sebagai berikut:

- Kontras yang tinggi, menunjukkan adanya perbedaan intensitas yang signifikan antara piksel yang berdekatan.
- Dissimilarity yang cukup tinggi, menunjukkan adanya variasi intensitas dalam gambar.
- Homogeneity yang rendah, menunjukkan bahwa gambar memiliki variasi intensitas yang tinggi dan tidak seragam.
- Energy yang rendah, menunjukkan bahwa tekstur dalam gambar tidak teratur atau acak.
- Correlation yang tinggi, menunjukkan adanya hubungan linear yang kuat antara pasangan piksel, yang dapat mengindikasikan adanya pola atau struktur dalam tekstur.

Karakteristik tekstur ini dapat memberikan informasi tambahan tentang sifat visual gambar, seperti kekasaran, kehalusan, keteraturan, atau kompleksitas tekstur. Informasi ini dapat digunakan sebagai fitur dalam tugas-tugas seperti klasifikasi gambar, segmentasi, atau analisis pola.

Image visualization in different color spaces (for example, HSV, LAB)

```
def visualize_color_spaces(image_path):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    img_lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    fig, axs = plt.subplots(1, 3, figsize=(15, 5))
    axs = axs.ravel()

    axs[0].imshow(img_rgb)
    axs[0].set_title("RGB")
    axs[0].axis("off")

    axs[1].imshow(img_hsv)
    axs[1].set_title("HSV")
    axs[1].axis("off")

    axs[2].imshow(img_lab)
    axs[2].set_title("LAB")
    axs[2].axis("off")

    plt.tight_layout()
    plt.show()

image_path = os.path.join(DATASET_PATH, CLASS_NAMES[0], os.listdir(os.path.join(DATASET_PATH, CLASS_NAMES[0]))[0])
visualize_color_spaces(image_path)
```



Gambar yang ditampilkan menunjukkan visualisasi sebuah citra batik dalam tiga ruang warna yang berbeda: RGB, HSV, dan LAB.

1. Gambar pertama (RGB):
   - Menampilkan citra batik dalam representasi ruang warna RGB (Red, Green, Blue).
   - Pola dan warna batik terlihat sesuai dengan persepsi visual manusia.
   - Warna-warna yang dominan adalah biru, putih, dan coklat, dengan pola geometris yang khas dari motif batik.
2. Gambar kedua (HSV):

- Menampilkan citra batik dalam representasi ruang warna HSV (Hue, Saturation, Value).
- Komponen Hue (corak warna) ditunjukkan dengan variasi warna yang berbeda. Warna biru dan hijau mendominasi, menunjukkan corak warna yang spesifik dari motif batik.
- Komponen Saturation (kejenuhan warna) terlihat dari intensitas warna yang bervariasi. Beberapa area memiliki warna yang lebih jenuh (cerah) dibandingkan area lainnya.
- Komponen Value (kecerahan) ditunjukkan oleh tingkat kecerahan atau gelap terangnya warna. Pola batik terlihat dengan jelas karena perbedaan nilai kecerahan antara motif dan latar belakang.

3. Gambar ketiga (LAB):

- Menampilkan citra batik dalam representasi ruang warna LAB (L*ab*).
- Komponen L* (Lightness) mewakili kecerahan atau intensitas cahaya. Pola batik terlihat jelas dengan kontras antara area yang lebih terang dan lebih gelap.
- Komponen a* mewakili warna dari hijau ke merah. Dalam gambar ini, warna hijau cenderung lebih dominan, menunjukkan keberadaan unsur warna hijau dalam motif batik.
- Komponen b* mewakili warna dari biru ke kuning. Warna biru terlihat lebih menonjol, menunjukkan dominasi unsur warna biru dalam motif batik.

Visualisasi dalam ruang warna yang berbeda memberikan perspektif yang berbeda tentang karakteristik warna dan pola dari citra batik. Ruang warna HSV dan LAB memisahkan informasi warna menjadi komponen-komponen yang lebih intuitif dan berguna untuk analisis citra, seperti segmentasi warna atau ekstraksi fitur warna.

Dengan melihat representasi citra batik dalam ruang warna yang berbeda, kita dapat memperoleh pemahaman yang lebih baik tentang distribusi warna, corak, dan pola yang ada dalam motif batik tersebut. Informasi ini dapat digunakan untuk tugas-tugas seperti klasifikasi motif batik, analisis tekstur, atau segmentasi citra batik.

Data Preparation

Split to Train, Test, and Val

```
# Path dataset baru untuk training, validation, dan test
TRAIN_PATH = "/content/drive/MyDrive/AOL/Train/"
VAL_PATH = "/content/drive/MyDrive/AOL/Val/"
TEST_PATH = "/content/drive/MyDrive/AOL/Test/"


# # Fungsi untuk resize gambar
# def resize_image(input_path, output_path, target_size):
#     img = Image.open(input_path)
#     img = img.convert("RGB")
#     img_resized = img.resize(target_size, Image.LANCZOS)
#     img_resized.save(output_path)

# # Pembagian data: 80% training, 10% validation, 10% test
# train_ratio = 0.8
# val_ratio = 0.1
# test_ratio = 0.1

# # Membuat direktori baru
# for path in [TRAIN_PATH, VAL_PATH, TEST_PATH]:
#     if not os.path.exists(path):
#         os.makedirs(path)

# # Looping untuk setiap kelas
# for class_name in CLASS_NAMES:
#     # Mengambil list semua file dalam kelas
#     class_path = os.path.join(DATASET_PATH, class_name)
#     files = os.listdir(class_path)
#     random.shuffle(files)  # Acak urutan file

#     # Menghitung jumlah file untuk masing-masing set
#     num_files = len(files)
#     num_train = int(train_ratio * num_files)
#     num_val = int(val_ratio * num_files)

#     # Memisahkan file ke dalam set
#     train_files = files[:num_train]
#     val_files = files[num_train:num_train + num_val]
#     test_files = files[num_train + num_val:]

#     # Memindahkan dan meresize gambar ke set masing-masing
#     for filename in train_files:
```

```
#         input_path = os.path.join(class_path, filename)
#         output_path = os.path.join(TRAIN_PATH, class_name, filename)
#         if not os.path.exists(os.path.join(TRAIN_PATH, class_name)):
#             os.makedirs(os.path.join(TRAIN_PATH, class_name))
#         resize_image(input_path, output_path, (224, 224))

#     for filename in val_files:
#         input_path = os.path.join(class_path, filename)
#         output_path = os.path.join(VAL_PATH, class_name, filename)
#         if not os.path.exists(os.path.join(VAL_PATH, class_name)):
#             os.makedirs(os.path.join(VAL_PATH, class_name))
#         resize_image(input_path, output_path, (224, 224))

#     for filename in test_files:
#         input_path = os.path.join(class_path, filename)
#         output_path = os.path.join(TEST_PATH, class_name, filename)
#         if not os.path.exists(os.path.join(TEST_PATH, class_name)):
#             os.makedirs(os.path.join(TEST_PATH, class_name))
#         resize_image(input_path, output_path, (224, 224))


for set_name, set_path in zip(["Train", "Val", "Test"], [TRAIN_PATH, VAL_PATH, TEST_PATH]):
    print(f"{set_name} set:")
    for class_name in CLASS_NAMES:
        num_images = len(os.listdir(os.path.join(set_path, class_name)))
        print(f"- {class_name}: {num_images} images")
```

```
Train set:
- batik-parang: 40 images
- batik-megamendung: 36 images
- batik-kawung: 36 images
Val set:
- batik-parang: 5 images
- batik-megamendung: 4 images
- batik-kawung: 4 images
Test set:
- batik-parang: 5 images
- batik-megamendung: 6 images
- batik-kawung: 5 images
```

Train Data W/o Augmentation

```
na_train_datagen = ImageDataGenerator(rescale=1./255)

na_train_generator = na_train_datagen.flow_from_directory(
    TRAIN_PATH,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 112 images belonging to 3 classes.
```

Data Augmentation

Dalam kasus ini, kami melakukan data augmentation hanya pada set data train. Alasannya adalah sebagai berikut:

1. Set data train digunakan untuk melatih model dan membangun kemampuan generalisasi model. Dengan melakukan augmentation pada set data train, kami dapat meningkatkan variasi gambar yang dilihat oleh model selama pelatihan. Hal ini membantu model untuk belajar fitur-fitur yang lebih beragam dan mengurangi overfitting.
2. Set data validation (validasi) digunakan untuk mengevaluasi performa model selama pelatihan dan melakukan penyetelan hyperparameter. Augmentation tidak boleh dilakukan pada set data validation karena tujuannya adalah untuk mengevaluasi performa model pada data yang belum pernah dilihat sebelumnya. Jika kami melakukan augmentation pada set data validation, evaluasi performa model menjadi bias dan tidak mencerminkan kemampuan generalisasi yang sebenarnya.
3. Set data test (uji) digunakan untuk mengevaluasi performa akhir model setelah pelatihan selesai. Sama seperti set data validation, augmentation tidak boleh dilakukan pada set data test. Set data test harus tetap asli dan tidak dimodifikasi agar dapat memberikan evaluasi yang objektif terhadap performa model pada data yang benar-benar baru.

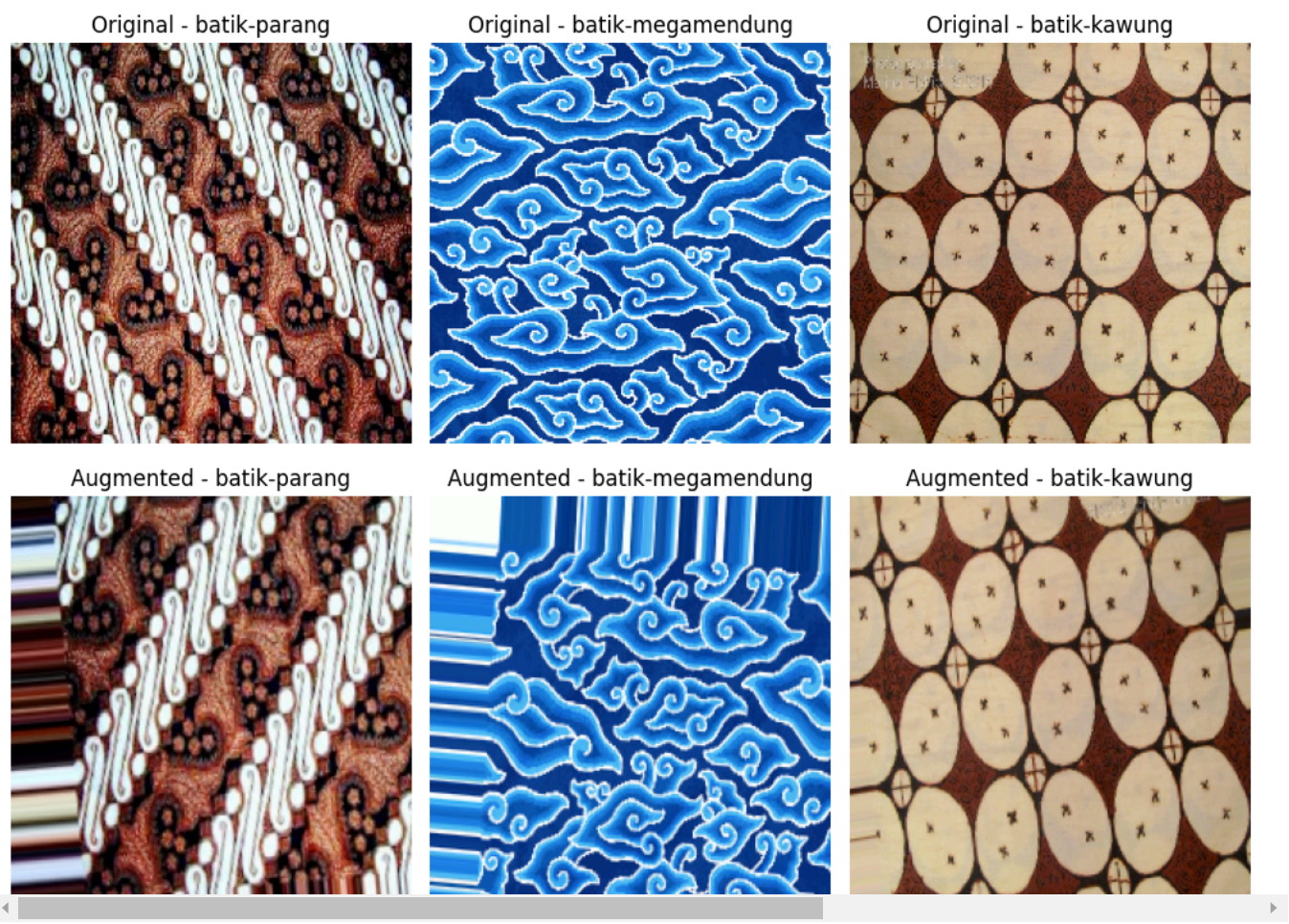
```
# Definisikan augmentation yang akan diterapkan pada set data train
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
```

```
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Baca gambar dari direktori set data train dengan augmentation
train_generator = train_datagen.flow_from_directory(
    TRAIN_PATH,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

# Baca gambar dari direktori set data validation tanpa augmentation
val_datagen = ImageDataGenerator(rescale=1./255)
val_generator = val_datagen.flow_from_directory(
    VAL_PATH,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

# Baca gambar dari direktori set data test tanpa augmentation
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    TEST_PATH,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
```

```
Found 112 images belonging to 3 classes.
Found 13 images belonging to 3 classes.
Found 16 images belonging to 3 classes.
```

```
# Fungsi untuk menampilkan gambar
def plot_images(images, titles, figsize=(10, 5)):
    fig, axes = plt.subplots(1, len(images), figsize=figsize)
    for i, (image, title) in enumerate(zip(images, titles)):
        axes[i].imshow(image.astype('uint8'))
        axes[i].set_title(title)
        axes[i].axis('off')
    plt.tight_layout()
    plt.show()

# Mendapatkan sampel gambar sebelum augmentasi
original_images = []
original_titles = []
for class_name in CLASS_NAMES:
    image_path = os.path.join(TRAIN_PATH, class_name)
    image_files = os.listdir(image_path)
    if len(image_files) > 0:
        sample_image_path = os.path.join(image_path, image_files[0])
        sample_image = load_img(sample_image_path, target_size=(224, 224))
        sample_image = img_to_array(sample_image)
        original_images.append(sample_image)
        original_titles.append(f"Original - {class_name}")

# Membuat generator augmentasi
augmentation_generator = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Mendapatkan sampel gambar setelah augmentasi
augmented_images = []
augmented_titles = []
for image, class_name in zip(original_images, CLASS_NAMES):
    augmented_image = augmentation_generator.random_transform(image)
    augmented_images.append(augmented_image.astype('uint8'))
    augmented_titles.append(f"Augmented - {class_name}")

# Menampilkan gambar sebelum augmentasi
plot_images(original_images, original_titles)
```

```
# Menampilkan gambar setelah augmentasi
plot_images(augmented_images, augmented_titles)
```



Gambar yang diaugmentasi menggunakan ImageDataGenerator sebenarnya tidak disimpan secara permanen di direktori. Augmentasi dilakukan secara real-time saat proses pelatihan berlangsung. Setiap kali model meminta batch data selama pelatihan, ImageDataGenerator akan menghasilkan gambar yang diaugmentasi secara dinamis berdasarkan gambar asli dan parameter augmentasi yang ditentukan.

Namun, disini kami dapat menggunakan ImageDataGenerator untuk menghasilkan gambar yang diaugmentasi secara eksplisit dan melihat persebaran kelas setelah augmentasi untuk data train.

```
# Membuat generator augmentasi
augmentation_generator = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Menghasilkan gambar yang diaugmentasi
augmented_images = []
augmented_labels = []

for class_name in CLASS_NAMES:
    image_path = os.path.join(TRAIN_PATH, class_name)
    image_files = os.listdir(image_path)
    for image_file in image_files:
        image = load_img(os.path.join(image_path, image_file), target_size=(224, 224))
        image = img_to_array(image)
        image = image.reshape((1,) + image.shape)

        # Menghasilkan 7 gambar yang diaugmentasi untuk setiap gambar asli
        for _ in range(7):
            augmented_image = augmentation_generator.flow(image, batch_size=1)[0]
            augmented_images.append(augmented_image)
            augmented_labels.append(class_name)

# Menampilkan persebaran kelas setelah augmentasi
class_counts = {}
```

```
for label in augmented_labels:
    if label in class_counts:
        class_counts[label] += 1
    else:
        class_counts[label] = 1

print("Persebaran Kelas Sebelum Augmentasi:")
for set_name, set_path in zip(["Train"], [TRAIN_PATH]):
    print(f"{set_name} set:")
    for class_name in CLASS_NAMES:
        num_images = len(os.listdir(os.path.join(set_path, class_name)))
        print(f"- {class_name}: {num_images} images")

print("\nPersebaran Kelas Setelah Augmentasi:")
for class_name, count in class_counts.items():
    print(f"{class_name}: {count} gambar")
```

```
Persebaran Kelas Sebelum Augmentasi:
Train set:
- batik-parang: 40 images
- batik-megamendung: 36 images
- batik-kawung: 36 images

Persebaran Kelas Setelah Augmentasi:
batik-parang: 280 gambar
batik-megamendung: 252 gambar
batik-kawung: 252 gambar
```

```
# Function for Training Val History (Loss, Accuracy)
def plot_training_history(history):
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

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

```
# Function for Confusion Matrix Visualization
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion Matrix', cmap=plt.cm.Blues):
    plt.figure(figsize=(8, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

Model

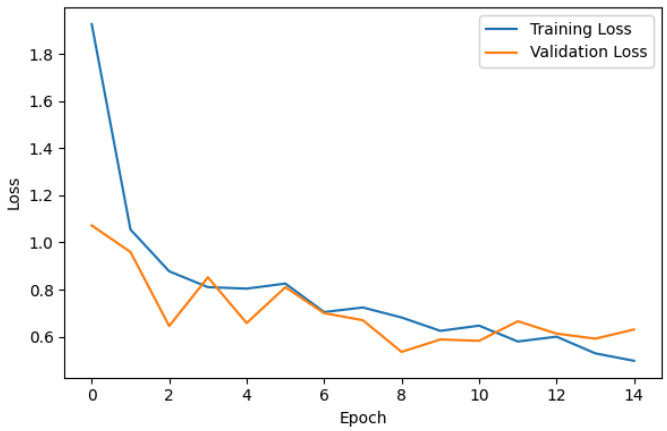
Design from Scratch

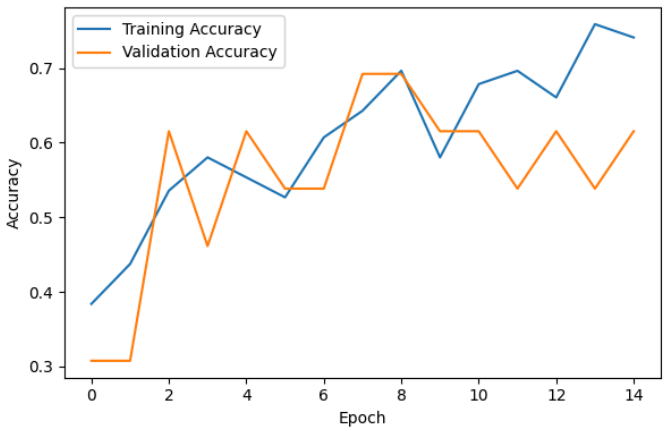
Model 1 (no augment)

```
model_scratch_1_na = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(len(CLASS_NAMES), activation='softmax')
])


model_scratch_1_na.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])


if not os.path.exists(TRAIN_PATH):
    raise ValueError(f"Training path {TRAIN_PATH} does not exist")
history_scratch_1_na = model_scratch_1_na.fit(
    na_train_generator,
    steps_per_epoch=len(na_train_generator),
    epochs=15,
    validation_data=val_generator,
    validation_steps=len(val_generator)
)
```

```
Epoch 1/15
4/4 [==============================] - 4s 438ms/step - loss: 1.9262 - accuracy: 0.3839 - val_loss: 1.0721 - val_accuracy: 0.3077
Epoch 2/15
4/4 [==============================] - 2s 616ms/step - loss: 1.0549 - accuracy: 0.4375 - val_loss: 0.9595 - val_accuracy: 0.3077
Epoch 3/15
4/4 [==============================] - 2s 471ms/step - loss: 0.8775 - accuracy: 0.5357 - val_loss: 0.6453 - val_accuracy: 0.6154
Epoch 4/15
4/4 [==============================] - 3s 689ms/step - loss: 0.8102 - accuracy: 0.5804 - val_loss: 0.8520 - val_accuracy: 0.4615
Epoch 5/15
4/4 [==============================] - 2s 443ms/step - loss: 0.8039 - accuracy: 0.5536 - val_loss: 0.6579 - val_accuracy: 0.6154
Epoch 6/15
4/4 [==============================] - 2s 626ms/step - loss: 0.8254 - accuracy: 0.5268 - val_loss: 0.8100 - val_accuracy: 0.5385
Epoch 7/15
4/4 [==============================] - 2s 470ms/step - loss: 0.7046 - accuracy: 0.6071 - val_loss: 0.6999 - val_accuracy: 0.5385
Epoch 8/15
4/4 [==============================] - 2s 649ms/step - loss: 0.7239 - accuracy: 0.6429 - val_loss: 0.6699 - val_accuracy: 0.6923
Epoch 9/15
4/4 [==============================] - 3s 633ms/step - loss: 0.6815 - accuracy: 0.6964 - val_loss: 0.5354 - val_accuracy: 0.6923
Epoch 10/15
4/4 [==============================] - 2s 611ms/step - loss: 0.6247 - accuracy: 0.5804 - val_loss: 0.5883 - val_accuracy: 0.6154
Epoch 11/15
4/4 [==============================] - 2s 448ms/step - loss: 0.6468 - accuracy: 0.6786 - val_loss: 0.5824 - val_accuracy: 0.6154
Epoch 12/15
4/4 [==============================] - 2s 469ms/step - loss: 0.5796 - accuracy: 0.6964 - val_loss: 0.6655 - val_accuracy: 0.5385
Epoch 13/15
4/4 [==============================] - 2s 677ms/step - loss: 0.6001 - accuracy: 0.6607 - val_loss: 0.6131 - val_accuracy: 0.6154
Epoch 14/15
4/4 [==============================] - 4s 673ms/step - loss: 0.5293 - accuracy: 0.7589 - val_loss: 0.5916 - val_accuracy: 0.5385
Epoch 15/15
4/4 [==============================] - 2s 490ms/step - loss: 0.4978 - accuracy: 0.7411 - val_loss: 0.6306 - val_accuracy: 0.6154
```

```
plot_training_history(history_scratch_1_na)
```



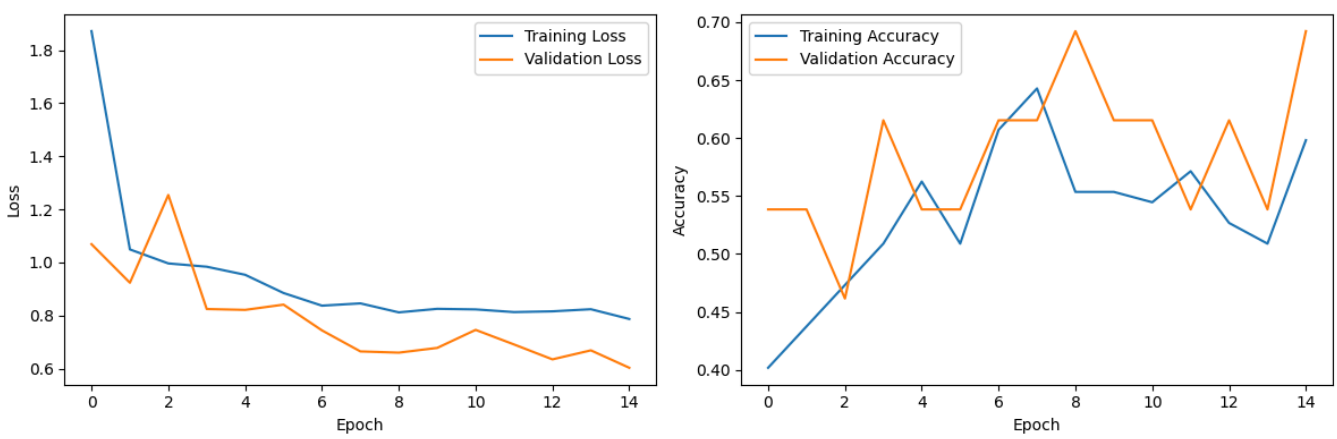


Model 1 (with augment)

```
model_scratch_1 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(len(CLASS_NAMES), activation='softmax')
])

model_scratch_1.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])

history_scratch_1 = model_scratch_1.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=15,
    validation_data=val_generator,
    validation_steps=len(val_generator)
)
```

```
Epoch 1/15
4/4 [==============================] - 7s 1s/step - loss: 1.8713 - accuracy: 0.4018 - val_loss: 1.0688 - val_accuracy: 0.5385
Epoch 2/15
4/4 [==============================] - 4s 1s/step - loss: 1.0490 - accuracy: 0.4375 - val_loss: 0.9235 - val_accuracy: 0.5385
Epoch 3/15
4/4 [==============================] - 3s 853ms/step - loss: 0.9962 - accuracy: 0.4732 - val_loss: 1.2541 - val_accuracy: 0.4615
Epoch 4/15
4/4 [==============================] - 3s 836ms/step - loss: 0.9839 - accuracy: 0.5089 - val_loss: 0.8245 - val_accuracy: 0.6154
Epoch 5/15
4/4 [==============================] - 5s 1s/step - loss: 0.9534 - accuracy: 0.5625 - val_loss: 0.8212 - val_accuracy: 0.5385
Epoch 6/15
4/4 [==============================] - 3s 774ms/step - loss: 0.8847 - accuracy: 0.5089 - val_loss: 0.8410 - val_accuracy: 0.5385
Epoch 7/15
4/4 [==============================] - 4s 1s/step - loss: 0.8371 - accuracy: 0.6071 - val_loss: 0.7439 - val_accuracy: 0.6154
Epoch 8/15
4/4 [==============================] - 3s 701ms/step - loss: 0.8457 - accuracy: 0.6429 - val_loss: 0.6647 - val_accuracy: 0.6154
Epoch 9/15
4/4 [==============================] - 3s 901ms/step - loss: 0.8119 - accuracy: 0.5536 - val_loss: 0.6601 - val_accuracy: 0.6923
Epoch 10/15
4/4 [==============================] - 3s 755ms/step - loss: 0.8252 - accuracy: 0.5536 - val_loss: 0.6780 - val_accuracy: 0.6154
Epoch 11/15
4/4 [==============================] - 5s 1s/step - loss: 0.8230 - accuracy: 0.5446 - val_loss: 0.7458 - val_accuracy: 0.6154
Epoch 12/15
4/4 [==============================] - 3s 772ms/step - loss: 0.8129 - accuracy: 0.5714 - val_loss: 0.6910 - val_accuracy: 0.5385
Epoch 13/15
4/4 [==============================] - 3s 761ms/step - loss: 0.8158 - accuracy: 0.5268 - val_loss: 0.6347 - val_accuracy: 0.6154
Epoch 14/15
4/4 [==============================] - 3s 760ms/step - loss: 0.8237 - accuracy: 0.5089 - val_loss: 0.6688 - val_accuracy: 0.5385
Epoch 15/15
4/4 [==============================] - 3s 763ms/step - loss: 0.7869 - accuracy: 0.5982 - val_loss: 0.6031 - val_accuracy: 0.6923
```

```
plot_training_history(history_scratch_1)
```

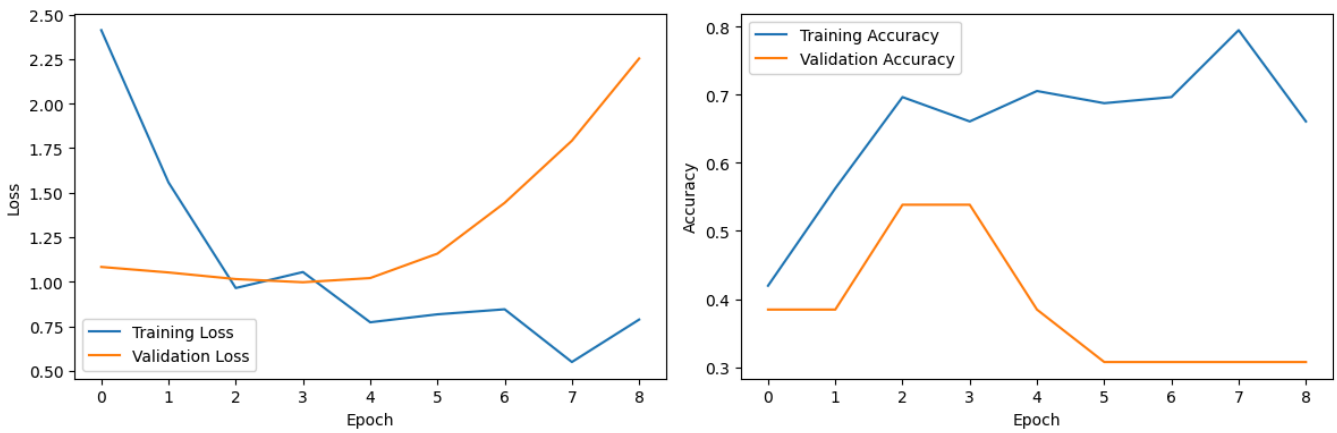


Model 2

```
model_scratch_2 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    BatchNormalization(),
    MaxPooling2D(2, 2),

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

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

    Flatten(),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(len(CLASS_NAMES), activation='softmax')
])


model_scratch_2.compile(loss='categorical_crossentropy',
                        optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
                        metrics=['accuracy'])


early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)


history_scratch_2 = model_scratch_2.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=50,
    validation_data=val_generator,
    validation_steps=len(val_generator),
    callbacks=[early_stopping]
)
```

```
Epoch 1/50
4/4 [==============================] - 7s 845ms/step - loss: 2.4119 - accuracy: 0.4196 - val_loss: 1.0828 - val_accuracy: 0.3846
Epoch 2/50
4/4 [==============================] - 3s 774ms/step - loss: 1.5572 - accuracy: 0.5625 - val_loss: 1.0518 - val_accuracy: 0.3846
Epoch 3/50
4/4 [==============================] - 3s 776ms/step - loss: 0.9644 - accuracy: 0.6964 - val_loss: 1.0146 - val_accuracy: 0.5385
Epoch 4/50
4/4 [==============================] - 3s 956ms/step - loss: 1.0542 - accuracy: 0.6607 - val_loss: 0.9969 - val_accuracy: 0.5385
Epoch 5/50
4/4 [==============================] - 3s 921ms/step - loss: 0.7722 - accuracy: 0.7054 - val_loss: 1.0203 - val_accuracy: 0.3846
Epoch 6/50
4/4 [==============================] - 3s 742ms/step - loss: 0.8168 - accuracy: 0.6875 - val_loss: 1.1576 - val_accuracy: 0.3077
Epoch 7/50
4/4 [==============================] - 4s 1s/step - loss: 0.8450 - accuracy: 0.6964 - val_loss: 1.4427 - val_accuracy: 0.3077
Epoch 8/50
4/4 [==============================] - 5s 1s/step - loss: 0.5491 - accuracy: 0.7946 - val_loss: 1.7914 - val_accuracy: 0.3077
Epoch 9/50
4/4 [==============================] - 3s 795ms/step - loss: 0.7873 - accuracy: 0.6607 - val_loss: 2.2529 - val_accuracy: 0.3077
```

```
plot_training_history(history_scratch_2)
```



Transfer Learning

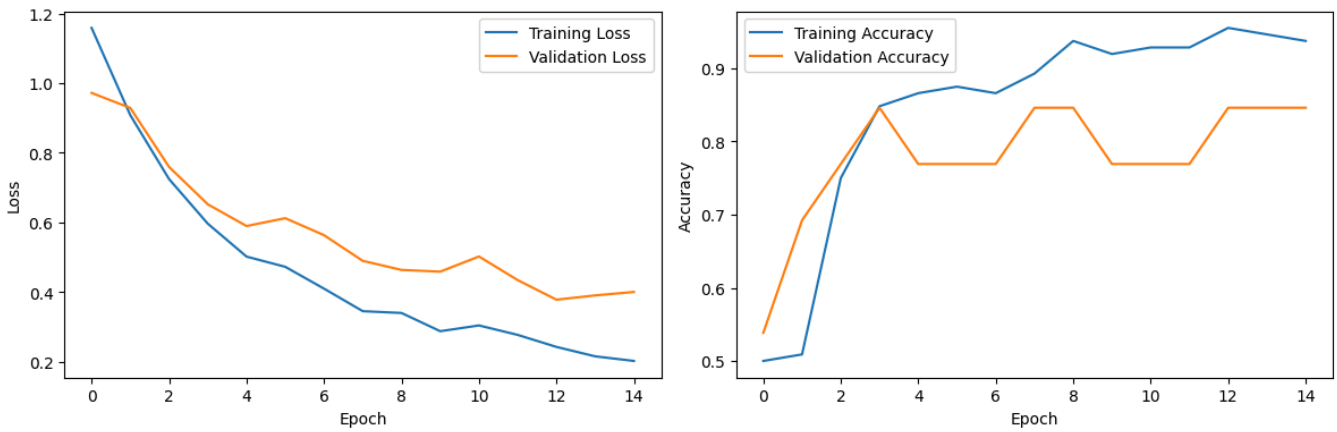
Pretrained Model 1: VGG-16

```
base_model_VGG16 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))


x = base_model_VGG16.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(x)

VGG_16 = Model(inputs=base_model_VGG16.input, outputs=predictions)


for layer in base_model_VGG16.layers:
    layer.trainable = False


VGG_16.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])


history_vgg_16 = VGG_16.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=15,
    validation_data=val_generator,
    validation_steps=len(val_generator)
)
```

```
Epoch 1/15
4/4 [==============================] - 6s 1s/step - loss: 1.1588 - accuracy: 0.5000 - val_loss: 0.9719 - val_accuracy: 0.5385
Epoch 2/15
4/4 [==============================] - 4s 736ms/step - loss: 0.9079 - accuracy: 0.5089 - val_loss: 0.9287 - val_accuracy: 0.6923
Epoch 3/15
4/4 [==============================] - 3s 891ms/step - loss: 0.7244 - accuracy: 0.7500 - val_loss: 0.7592 - val_accuracy: 0.7692
Epoch 4/15
4/4 [==============================] - 3s 736ms/step - loss: 0.5962 - accuracy: 0.8482 - val_loss: 0.6519 - val_accuracy: 0.8462
Epoch 5/15
4/4 [==============================] - 5s 1s/step - loss: 0.5016 - accuracy: 0.8661 - val_loss: 0.5894 - val_accuracy: 0.7692
Epoch 6/15
4/4 [==============================] - 3s 772ms/step - loss: 0.4722 - accuracy: 0.8750 - val_loss: 0.6119 - val_accuracy: 0.7692
Epoch 7/15
4/4 [==============================] - 3s 900ms/step - loss: 0.4095 - accuracy: 0.8661 - val_loss: 0.5631 - val_accuracy: 0.7692
Epoch 8/15
4/4 [==============================] - 3s 797ms/step - loss: 0.3447 - accuracy: 0.8929 - val_loss: 0.4894 - val_accuracy: 0.8462
Epoch 9/15
4/4 [==============================] - 5s 1s/step - loss: 0.3396 - accuracy: 0.9375 - val_loss: 0.4633 - val_accuracy: 0.8462
Epoch 10/15
4/4 [==============================] - 3s 784ms/step - loss: 0.2872 - accuracy: 0.9196 - val_loss: 0.4584 - val_accuracy: 0.7692
Epoch 11/15
4/4 [==============================] - 3s 783ms/step - loss: 0.3037 - accuracy: 0.9286 - val_loss: 0.5019 - val_accuracy: 0.7692
Epoch 12/15
4/4 [==============================] - 5s 1s/step - loss: 0.2766 - accuracy: 0.9286 - val_loss: 0.4340 - val_accuracy: 0.7692
Epoch 13/15
```

```
4/4 [==============================] - 3s 818ms/step - loss: 0.2420 - accuracy: 0.9554 - val_loss: 0.3777 - val_accuracy: 0.8462
Epoch 14/15
4/4 [==============================] - 3s 883ms/step - loss: 0.2150 - accuracy: 0.9464 - val_loss: 0.3900 - val_accuracy: 0.8462
Epoch 15/15
4/4 [==============================] - 5s 1s/step - loss: 0.2017 - accuracy: 0.9375 - val_loss: 0.4000 - val_accuracy: 0.8462
```

```
plot_training_history(history_vgg_16)
```



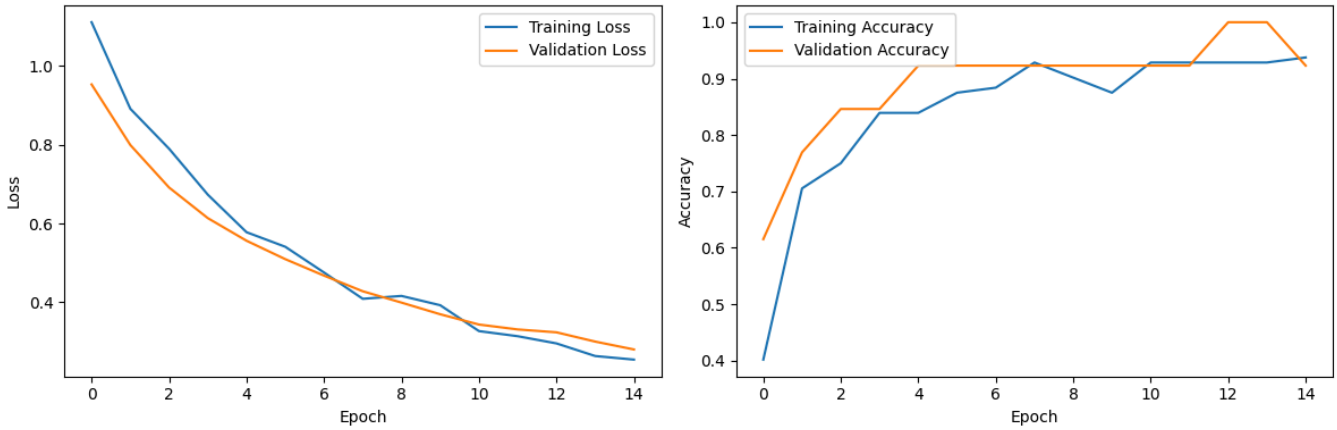
Pretrained Model 2: Xception

```
base_model_xception = Xception(weights='imagenet', include_top=False, input_shape=(224, 224, 3))


xx = base_model_xception.output
xx = GlobalAveragePooling2D()(xx)
xx = Dense(256, activation='relu')(xx)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(xx)

Xception = Model(inputs=base_model_xception.input, outputs=predictions)


for layer in base_model_xception.layers:
    layer.trainable = False


Xception.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])


history_xception = Xception.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=15,
    validation_data=val_generator,
    validation_steps=len(val_generator)
)
```

```
Epoch 1/15
4/4 [==============================] - 7s 1s/step - loss: 1.1100 - accuracy: 0.4018 - val_loss: 0.9524 - val_accuracy: 0.6154
Epoch 2/15
4/4 [==============================] - 4s 1s/step - loss: 0.8907 - accuracy: 0.7054 - val_loss: 0.7989 - val_accuracy: 0.7692
Epoch 3/15
4/4 [==============================] - 3s 783ms/step - loss: 0.7895 - accuracy: 0.7500 - val_loss: 0.6911 - val_accuracy: 0.8462
Epoch 4/15
4/4 [==============================] - 3s 810ms/step - loss: 0.6730 - accuracy: 0.8393 - val_loss: 0.6136 - val_accuracy: 0.8462
Epoch 5/15
4/4 [==============================] - 4s 862ms/step - loss: 0.5779 - accuracy: 0.8393 - val_loss: 0.5562 - val_accuracy: 0.9231
Epoch 6/15
4/4 [==============================] - 3s 934ms/step - loss: 0.5410 - accuracy: 0.8750 - val_loss: 0.5096 - val_accuracy: 0.9231
Epoch 7/15
4/4 [==============================] - 3s 785ms/step - loss: 0.4758 - accuracy: 0.8839 - val_loss: 0.4678 - val_accuracy: 0.9231
Epoch 8/15
4/4 [==============================] - 3s 934ms/step - loss: 0.4091 - accuracy: 0.9286 - val_loss: 0.4282 - val_accuracy: 0.9231
Epoch 9/15
4/4 [==============================] - 5s 1s/step - loss: 0.4164 - accuracy: 0.9018 - val_loss: 0.3995 - val_accuracy: 0.9231
Epoch 10/15
4/4 [==============================] - 3s 777ms/step - loss: 0.3928 - accuracy: 0.8750 - val_loss: 0.3699 - val_accuracy: 0.9231
Epoch 11/15
4/4 [==============================] - 3s 932ms/step - loss: 0.3272 - accuracy: 0.9286 - val_loss: 0.3438 - val_accuracy: 0.9231
Epoch 12/15
4/4 [==============================] - 5s 1s/step - loss: 0.3143 - accuracy: 0.9286 - val_loss: 0.3314 - val_accuracy: 0.9231
```

```
Epoch 13/15
4/4 [==============================] - 3s 907ms/step - loss: 0.2961 - accuracy: 0.9286 - val_loss: 0.3241 - val_accuracy: 1.0000
Epoch 14/15
4/4 [==============================] - 3s 783ms/step - loss: 0.2641 - accuracy: 0.9286 - val_loss: 0.3006 - val_accuracy: 1.0000
Epoch 15/15
4/4 [==============================] - 3s 753ms/step - loss: 0.2550 - accuracy: 0.9375 - val_loss: 0.2807 - val_accuracy: 0.9231
```

```
plot_training_history(history_xception)
```



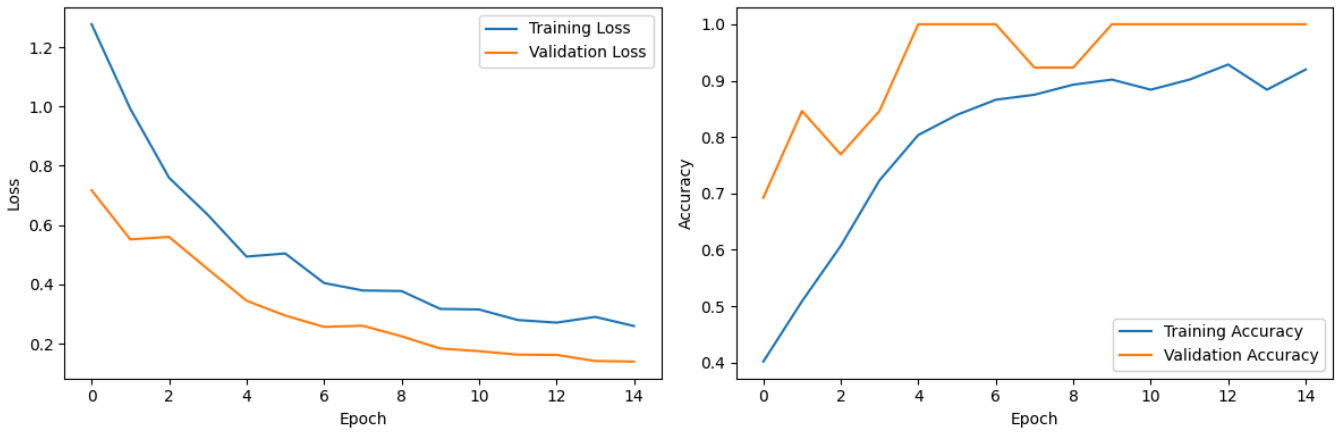
Pretrained Model 3: InceptionV3

```
base_model_inceptionv3 = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))


xxx = base_model_inceptionv3.output
xxx = GlobalAveragePooling2D()(xxx)
xxx = Dense(256, activation='relu')(xxx)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(xxx)

InceptionV3 = Model(inputs=base_model_inceptionv3.input, outputs=predictions)


for layer in base_model_inceptionv3.layers:
    layer.trainable = False


InceptionV3.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])


history_inceptionv3 = InceptionV3.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=15,
    validation_data=val_generator,
    validation_steps=len(val_generator)
)
```

```
Epoch 1/15
4/4 [==============================] - 9s 1s/step - loss: 1.2774 - accuracy: 0.4018 - val_loss: 0.7177 - val_accuracy: 0.6923
Epoch 2/15
4/4 [==============================] - 5s 1s/step - loss: 0.9921 - accuracy: 0.5089 - val_loss: 0.5518 - val_accuracy: 0.8462
Epoch 3/15
4/4 [==============================] - 3s 933ms/step - loss: 0.7597 - accuracy: 0.6071 - val_loss: 0.5602 - val_accuracy: 0.7692
Epoch 4/15
4/4 [==============================] - 3s 749ms/step - loss: 0.6347 - accuracy: 0.7232 - val_loss: 0.4519 - val_accuracy: 0.8462
Epoch 5/15
4/4 [==============================] - 4s 1s/step - loss: 0.4938 - accuracy: 0.8036 - val_loss: 0.3451 - val_accuracy: 1.0000
Epoch 6/15
4/4 [==============================] - 3s 933ms/step - loss: 0.5044 - accuracy: 0.8393 - val_loss: 0.2950 - val_accuracy: 1.0000
Epoch 7/15
4/4 [==============================] - 3s 742ms/step - loss: 0.4048 - accuracy: 0.8661 - val_loss: 0.2569 - val_accuracy: 1.0000
Epoch 8/15
4/4 [==============================] - 3s 752ms/step - loss: 0.3796 - accuracy: 0.8750 - val_loss: 0.2607 - val_accuracy: 0.9231
Epoch 9/15
4/4 [==============================] - 5s 1s/step - loss: 0.3777 - accuracy: 0.8929 - val_loss: 0.2252 - val_accuracy: 0.9231
Epoch 10/15
4/4 [==============================] - 3s 724ms/step - loss: 0.3174 - accuracy: 0.9018 - val_loss: 0.1840 - val_accuracy: 1.0000
Epoch 11/15
4/4 [==============================] - 3s 738ms/step - loss: 0.3155 - accuracy: 0.8839 - val_loss: 0.1749 - val_accuracy: 1.0000
Epoch 12/15
```

```
4/4 [==============================] - 5s 1s/step - loss: 0.2800 - accuracy: 0.9018 - val_loss: 0.1632 - val_accuracy: 1.0000
Epoch 13/15
4/4 [==============================] - 3s 941ms/step - loss: 0.2711 - accuracy: 0.9286 - val_loss: 0.1623 - val_accuracy: 1.0000
Epoch 14/15
4/4 [==============================] - 3s 915ms/step - loss: 0.2905 - accuracy: 0.8839 - val_loss: 0.1419 - val_accuracy: 1.0000
Epoch 15/15
4/4 [==============================] - 3s 785ms/step - loss: 0.2597 - accuracy: 0.9196 - val_loss: 0.1397 - val_accuracy: 1.0000
```

```
plot_training_history(history_inceptionv3)
```



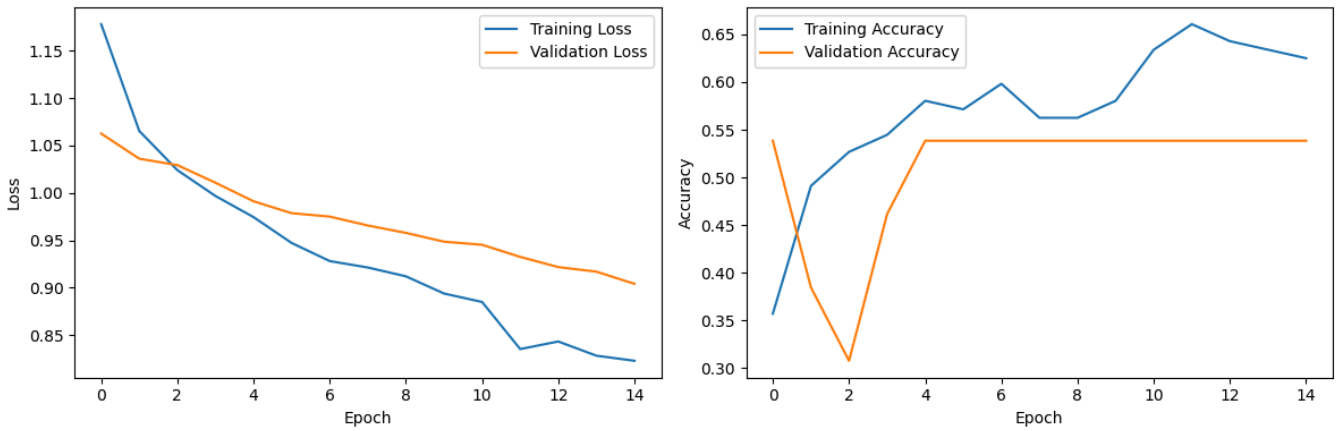
Pretrained Model 4: ResNet50

```
base_model_resnet50 = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))


xxxx = base_model_resnet50.output
xxxx = GlobalAveragePooling2D()(xxxx)
xxxx = Dense(256, activation='relu')(xxxx)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(xxxx)

ResNet50 = Model(inputs=base_model_resnet50.input, outputs=predictions)


for layer in base_model_resnet50.layers:
    layer.trainable = False


ResNet50.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])


history_resnet50 = ResNet50.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=15,
    validation_data=val_generator,
    validation_steps=len(val_generator)
)
```

```
Epoch 1/15
4/4 [==============================] - 9s 1s/step - loss: 1.1779 - accuracy: 0.3571 - val_loss: 1.0625 - val_accuracy: 0.5385
Epoch 2/15
4/4 [==============================] - 3s 970ms/step - loss: 1.0653 - accuracy: 0.4911 - val_loss: 1.0360 - val_accuracy: 0.3846
Epoch 3/15
4/4 [==============================] - 3s 864ms/step - loss: 1.0240 - accuracy: 0.5268 - val_loss: 1.0293 - val_accuracy: 0.3077
Epoch 4/15
4/4 [==============================] - 4s 942ms/step - loss: 0.9967 - accuracy: 0.5446 - val_loss: 1.0107 - val_accuracy: 0.4615
Epoch 5/15
4/4 [==============================] - 3s 751ms/step - loss: 0.9744 - accuracy: 0.5804 - val_loss: 0.9911 - val_accuracy: 0.5385
Epoch 6/15
4/4 [==============================] - 3s 747ms/step - loss: 0.9472 - accuracy: 0.5714 - val_loss: 0.9786 - val_accuracy: 0.5385
Epoch 7/15
4/4 [==============================] - 4s 1s/step - loss: 0.9281 - accuracy: 0.5982 - val_loss: 0.9750 - val_accuracy: 0.5385
Epoch 8/15
4/4 [==============================] - 4s 1s/step - loss: 0.9213 - accuracy: 0.5625 - val_loss: 0.9656 - val_accuracy: 0.5385
Epoch 9/15
4/4 [==============================] - 3s 778ms/step - loss: 0.9120 - accuracy: 0.5625 - val_loss: 0.9578 - val_accuracy: 0.5385
Epoch 10/15
4/4 [==============================] - 3s 807ms/step - loss: 0.8939 - accuracy: 0.5804 - val_loss: 0.9486 - val_accuracy: 0.5385
Epoch 11/15
4/4 [==============================] - 4s 1s/step - loss: 0.8850 - accuracy: 0.6339 - val_loss: 0.9453 - val_accuracy: 0.5385
```

```
Epoch 12/15
4/4 [==============================] - 3s 916ms/step - loss: 0.8353 - accuracy: 0.6607 - val_loss: 0.9325 - val_accuracy: 0.5385
Epoch 13/15
4/4 [==============================] - 3s 776ms/step - loss: 0.8433 - accuracy: 0.6429 - val_loss: 0.9218 - val_accuracy: 0.5385
Epoch 14/15
4/4 [==============================] - 3s 733ms/step - loss: 0.8284 - accuracy: 0.6339 - val_loss: 0.9169 - val_accuracy: 0.5385
Epoch 15/15
4/4 [==============================] - 5s 1s/step - loss: 0.8230 - accuracy: 0.6250 - val_loss: 0.9041 - val_accuracy: 0.5385
```

```
plot_training_history(history_resnet50)
```



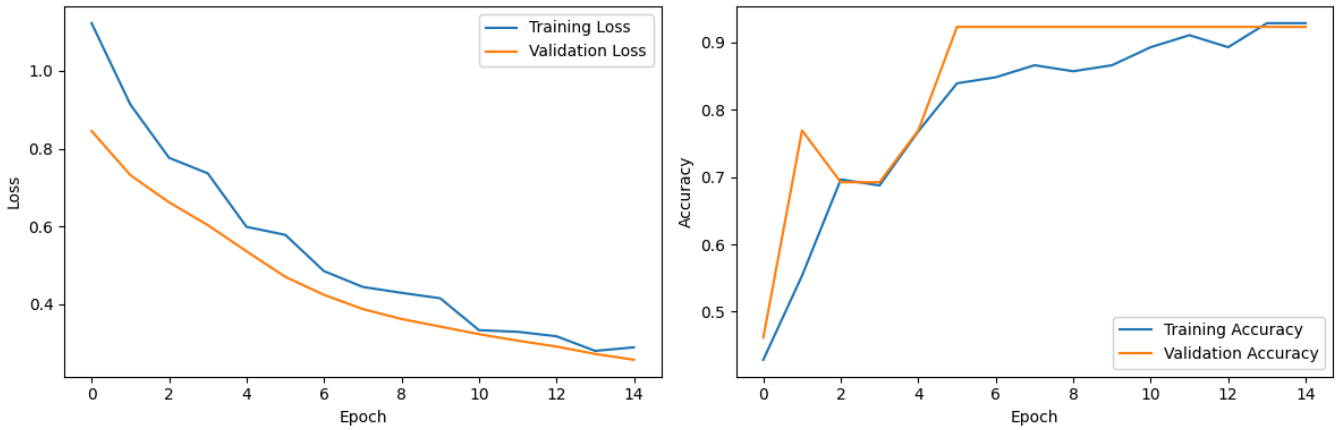
Pretrained Model 5: DenseNet121

```
base_model_densenet21 = DenseNet121(weights='imagenet', include_top=False, input_shape=(224, 224, 3))


xxxxx = base_model_densenet21.output
xxxxx = GlobalAveragePooling2D()(xxxxx)
xxxxx = Dense(256, activation='relu')(xxxxx)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(xxxxx)

DenseNet121 = Model(inputs=base_model_densenet21.input, outputs=predictions)


for layer in base_model_densenet21.layers:
    layer.trainable = False


DenseNet121.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])


history_densenet121 = DenseNet121.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=15,
    validation_data=val_generator,
    validation_steps=len(val_generator)
)
```

```
Epoch 1/15
4/4 [==============================] - 13s 2s/step - loss: 1.1225 - accuracy: 0.4286 - val_loss: 0.8455 - val_accuracy: 0.4615
Epoch 2/15
4/4 [==============================] - 4s 901ms/step - loss: 0.9136 - accuracy: 0.5536 - val_loss: 0.7322 - val_accuracy: 0.7692
Epoch 3/15
4/4 [==============================] - 3s 746ms/step - loss: 0.7765 - accuracy: 0.6964 - val_loss: 0.6621 - val_accuracy: 0.6923
Epoch 4/15
4/4 [==============================] - 3s 783ms/step - loss: 0.7366 - accuracy: 0.6875 - val_loss: 0.6036 - val_accuracy: 0.6923
Epoch 5/15
4/4 [==============================] - 4s 936ms/step - loss: 0.5990 - accuracy: 0.7679 - val_loss: 0.5362 - val_accuracy: 0.7692
Epoch 6/15
4/4 [==============================] - 3s 776ms/step - loss: 0.5786 - accuracy: 0.8393 - val_loss: 0.4707 - val_accuracy: 0.9231
Epoch 7/15
4/4 [==============================] - 3s 751ms/step - loss: 0.4853 - accuracy: 0.8482 - val_loss: 0.4244 - val_accuracy: 0.9231
Epoch 8/15
4/4 [==============================] - 3s 787ms/step - loss: 0.4447 - accuracy: 0.8661 - val_loss: 0.3878 - val_accuracy: 0.9231
Epoch 9/15
4/4 [==============================] - 3s 897ms/step - loss: 0.4295 - accuracy: 0.8571 - val_loss: 0.3623 - val_accuracy: 0.9231
Epoch 10/15
4/4 [==============================] - 3s 790ms/step - loss: 0.4155 - accuracy: 0.8661 - val_loss: 0.3428 - val_accuracy: 0.9231
Epoch 11/15
```

```
4/4 [==============================] - 4s 1s/step - loss: 0.3336 - accuracy: 0.8929 - val_loss: 0.3233 - val_accuracy: 0.9231
Epoch 12/15
4/4 [==============================] - 3s 931ms/step - loss: 0.3294 - accuracy: 0.9107 - val_loss: 0.3066 - val_accuracy: 0.9231
Epoch 13/15
4/4 [==============================] - 3s 823ms/step - loss: 0.3179 - accuracy: 0.8929 - val_loss: 0.2915 - val_accuracy: 0.9231
Epoch 14/15
4/4 [==============================] - 4s 1s/step - loss: 0.2803 - accuracy: 0.9286 - val_loss: 0.2725 - val_accuracy: 0.9231
Epoch 15/15
4/4 [==============================] - 3s 769ms/step - loss: 0.2895 - accuracy: 0.9286 - val_loss: 0.2575 - val_accuracy: 0.9231
```

```
plot_training_history(history_densenet121)
```



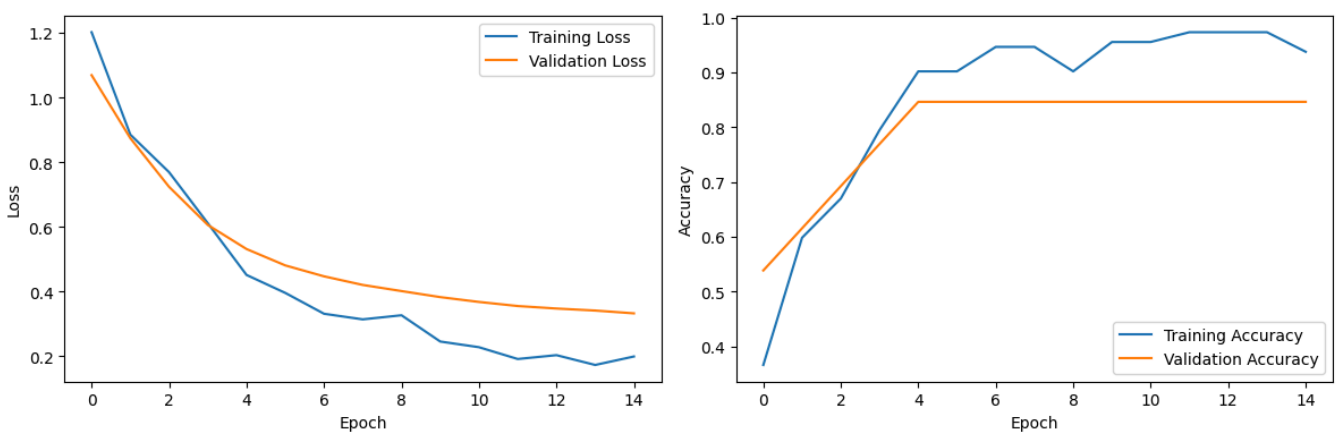
Pretrained Model 6: MobileNetV2

```
base_model_mobilenetv2 = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))


xxxxxx = base_model_mobilenetv2.output
xxxxxx = GlobalAveragePooling2D()(xxxxxx)
xxxxxx = Dense(256, activation='relu')(xxxxxx)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(xxxxxx)

MobileNetV2 = Model(inputs=base_model_mobilenetv2.input, outputs=predictions)


for layer in base_model_mobilenetv2.layers:
    layer.trainable = False


MobileNetV2.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])


history_mobilenetv2 = MobileNetV2.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=15,
    validation_data=val_generator,
    validation_steps=len(val_generator)
)
```

```
Epoch 1/15
4/4 [==============================] - 9s 1s/step - loss: 1.2018 - accuracy: 0.3661 - val_loss: 1.0690 - val_accuracy: 0.5385
Epoch 2/15
4/4 [==============================] - 3s 764ms/step - loss: 0.8851 - accuracy: 0.5982 - val_loss: 0.8740 - val_accuracy: 0.6154
Epoch 3/15
4/4 [==============================] - 4s 1s/step - loss: 0.7689 - accuracy: 0.6696 - val_loss: 0.7236 - val_accuracy: 0.6923
Epoch 4/15
4/4 [==============================] - 4s 998ms/step - loss: 0.6111 - accuracy: 0.7946 - val_loss: 0.6057 - val_accuracy: 0.7692
Epoch 5/15
4/4 [==============================] - 3s 864ms/step - loss: 0.4511 - accuracy: 0.9018 - val_loss: 0.5312 - val_accuracy: 0.8462
Epoch 6/15
4/4 [==============================] - 3s 948ms/step - loss: 0.3955 - accuracy: 0.9018 - val_loss: 0.4806 - val_accuracy: 0.8462
Epoch 7/15
4/4 [==============================] - 3s 728ms/step - loss: 0.3310 - accuracy: 0.9464 - val_loss: 0.4468 - val_accuracy: 0.8462
Epoch 8/15
4/4 [==============================] - 3s 773ms/step - loss: 0.3137 - accuracy: 0.9464 - val_loss: 0.4202 - val_accuracy: 0.8462
Epoch 9/15
4/4 [==============================] - 4s 954ms/step - loss: 0.3264 - accuracy: 0.9018 - val_loss: 0.4012 - val_accuracy: 0.8462
Epoch 10/15
4/4 [==============================] - 3s 898ms/step - loss: 0.2452 - accuracy: 0.9554 - val_loss: 0.3825 - val_accuracy: 0.8462
```

```
Epoch 11/15
4/4 [==============================] - 3s 850ms/step - loss: 0.2276 - accuracy: 0.9554 - val_loss: 0.3676 - val_accuracy: 0.8462
Epoch 12/15
4/4 [==============================] - 4s 1s/step - loss: 0.1911 - accuracy: 0.9732 - val_loss: 0.3549 - val_accuracy: 0.8462
Epoch 13/15
4/4 [==============================] - 3s 963ms/step - loss: 0.2030 - accuracy: 0.9732 - val_loss: 0.3472 - val_accuracy: 0.8462
Epoch 14/15
4/4 [==============================] - 3s 839ms/step - loss: 0.1727 - accuracy: 0.9732 - val_loss: 0.3409 - val_accuracy: 0.8462
Epoch 15/15
4/4 [==============================] - 3s 722ms/step - loss: 0.1989 - accuracy: 0.9375 - val_loss: 0.3323 - val_accuracy: 0.8462
```

```
plot_training_history(history_mobilenetv2)
```



Pretrained Model 7: EfficientNetV2

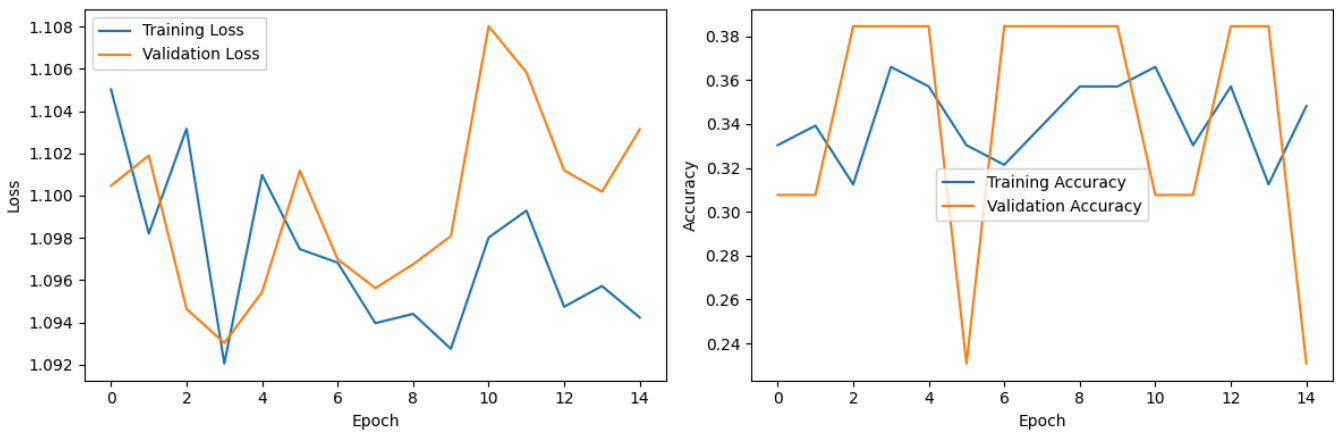
```
base_model_efficientnetv2 = tf.keras.applications.efficientnet_v2.EfficientNetV2B0(weights='imagenet', include_top=False, input_shape=(
```

```
vii = base_model_efficientnetv2.output
vii = GlobalAveragePooling2D()(vii)
vii = Dense(256, activation='relu')(vii)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(vii)

efficientnetv2 = Model(inputs=base_model_efficientnetv2.input, outputs=predictions)


for layer in base_model_efficientnetv2.layers:
    layer.trainable = False


efficientnetv2.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])


history_efficientnetv2 = efficientnetv2.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=15,
    validation_data=val_generator,
    validation_steps=len(val_generator)
)
```

```
Epoch 1/15
4/4 [==============================] - 15s 2s/step - loss: 1.1050 - accuracy: 0.3304 - val_loss: 1.1005 - val_accuracy: 0.3077
Epoch 2/15
4/4 [==============================] - 3s 722ms/step - loss: 1.0982 - accuracy: 0.3393 - val_loss: 1.1019 - val_accuracy: 0.3077
Epoch 3/15
4/4 [==============================] - 3s 807ms/step - loss: 1.1032 - accuracy: 0.3125 - val_loss: 1.0946 - val_accuracy: 0.3846
Epoch 4/15
4/4 [==============================] - 3s 857ms/step - loss: 1.0920 - accuracy: 0.3661 - val_loss: 1.0930 - val_accuracy: 0.3846
Epoch 5/15
4/4 [==============================] - 3s 743ms/step - loss: 1.1010 - accuracy: 0.3571 - val_loss: 1.0954 - val_accuracy: 0.3846
Epoch 6/15
4/4 [==============================] - 3s 745ms/step - loss: 1.0975 - accuracy: 0.3304 - val_loss: 1.1012 - val_accuracy: 0.2308
Epoch 7/15
4/4 [==============================] - 4s 1s/step - loss: 1.0968 - accuracy: 0.3214 - val_loss: 1.0970 - val_accuracy: 0.3846
Epoch 8/15
4/4 [==============================] - 3s 761ms/step - loss: 1.0940 - accuracy: 0.3393 - val_loss: 1.0956 - val_accuracy: 0.3846
Epoch 9/15
4/4 [==============================] - 3s 700ms/step - loss: 1.0944 - accuracy: 0.3571 - val_loss: 1.0968 - val_accuracy: 0.3846
Epoch 10/15
```

```
4/4 [==============================] - 4s 947ms/step - loss: 1.0927 - accuracy: 0.3571 - val_loss: 1.0981 - val_accuracy: 0.3846
Epoch 11/15
4/4 [==============================] - 4s 909ms/step - loss: 1.0980 - accuracy: 0.3661 - val_loss: 1.1080 - val_accuracy: 0.3077
Epoch 12/15
4/4 [==============================] - 3s 828ms/step - loss: 1.0993 - accuracy: 0.3304 - val_loss: 1.1058 - val_accuracy: 0.3077
Epoch 13/15
4/4 [==============================] - 3s 783ms/step - loss: 1.0947 - accuracy: 0.3571 - val_loss: 1.1012 - val_accuracy: 0.3846
Epoch 14/15
4/4 [==============================] - 5s 1s/step - loss: 1.0957 - accuracy: 0.3125 - val_loss: 1.1002 - val_accuracy: 0.3846
Epoch 15/15
4/4 [==============================] - 3s 854ms/step - loss: 1.0942 - accuracy: 0.3482 - val_loss: 1.1031 - val_accuracy: 0.2308
```

```
plot_training_history(history_efficientnetv2)
```



Evaluation

Design from Stratch

[ ] ↳ *9 cells hidden*

Transfer Learning

[ ] ↳ *21 cells hidden*

Experiment with the Highest Acc Model

```
checkpoint_path = "data_model_checkpoint"
checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                         save_weights_only=True, # save only the model weights
                                                         monitor="val_accuracy", # save the model weights which score the best validatic
                                                         save_best_only=True) # only keep the best model weights on file (delete the re
```

```
import keras
from tensorflow.keras import layers
```

```
base_model = InceptionV3(include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False

inputs = layers.Input(shape=(224, 224, 3), name="input_layer")
# x = d_augment(inputs)
x = inputs
x = base_model(x, training=False)
x = layers.GlobalAveragePooling2D(name="global_average_pooling")(x)
outputs = layers.Dense(len(CLASS_NAMES), activation="softmax", name="output_layer")(x)
model = tf.keras.Model(inputs, outputs)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering
87910968/87910968 [==============================] - 0s 0us/step
```

```
model.summary()
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 224, 224, 3)]     0

 inception_v3 (Functional)   (None, 5, 5, 2048)        21802784

 global_average_pooling (Gl  (None, 2048)              0
 obalAveragePooling2D)

 output_layer (Dense)        (None, 3)                 6147

=================================================================
Total params: 21808931 (83.19 MB)
Trainable params: 6147 (24.01 KB)
Non-trainable params: 21802784 (83.17 MB)
_________________________________________________________________
```

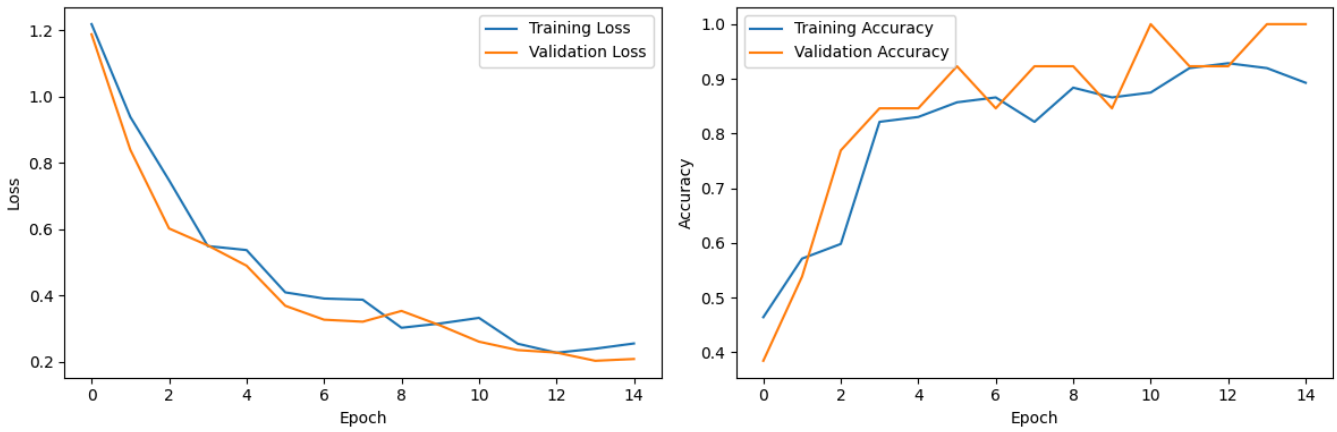
```
# Compile
model.compile(loss="categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# Fit
history_model = model.fit(train_generator,
                          epochs=15,
                          validation_data=val_generator,
                          validation_steps= len(val_generator),
                           callbacks=[checkpoint_callback]) # save best model weights to file
```

```
Epoch 1/15
4/4 [==============================] - 46s 11s/step - loss: 1.2177 - accuracy: 0.4643 - val_loss: 1.1871 - val_accuracy: 0.3846
Epoch 2/15
4/4 [==============================] - 19s 5s/step - loss: 0.9378 - accuracy: 0.5714 - val_loss: 0.8389 - val_accuracy: 0.5385
Epoch 3/15
4/4 [==============================] - 19s 4s/step - loss: 0.7469 - accuracy: 0.5982 - val_loss: 0.6015 - val_accuracy: 0.7692
Epoch 4/15
4/4 [==============================] - 29s 7s/step - loss: 0.5484 - accuracy: 0.8214 - val_loss: 0.5504 - val_accuracy: 0.8462
Epoch 5/15
4/4 [==============================] - 26s 6s/step - loss: 0.5364 - accuracy: 0.8304 - val_loss: 0.4893 - val_accuracy: 0.8462
Epoch 6/15
4/4 [==============================] - 21s 5s/step - loss: 0.4089 - accuracy: 0.8571 - val_loss: 0.3683 - val_accuracy: 0.9231
Epoch 7/15
4/4 [==============================] - 22s 6s/step - loss: 0.3902 - accuracy: 0.8661 - val_loss: 0.3264 - val_accuracy: 0.8462
Epoch 8/15
4/4 [==============================] - 23s 6s/step - loss: 0.3866 - accuracy: 0.8214 - val_loss: 0.3204 - val_accuracy: 0.9231
Epoch 9/15
4/4 [==============================] - 18s 5s/step - loss: 0.3021 - accuracy: 0.8839 - val_loss: 0.3528 - val_accuracy: 0.9231
Epoch 10/15
4/4 [==============================] - 19s 4s/step - loss: 0.3150 - accuracy: 0.8661 - val_loss: 0.3090 - val_accuracy: 0.8462
Epoch 11/15
4/4 [==============================] - 35s 8s/step - loss: 0.3319 - accuracy: 0.8750 - val_loss: 0.2601 - val_accuracy: 1.0000
Epoch 12/15
4/4 [==============================] - 21s 4s/step - loss: 0.2539 - accuracy: 0.9196 - val_loss: 0.2348 - val_accuracy: 0.9231
Epoch 13/15
4/4 [==============================] - 19s 4s/step - loss: 0.2268 - accuracy: 0.9286 - val_loss: 0.2273 - val_accuracy: 0.9231
Epoch 14/15
4/4 [==============================] - 19s 5s/step - loss: 0.2390 - accuracy: 0.9196 - val_loss: 0.2025 - val_accuracy: 1.0000
Epoch 15/15
4/4 [==============================] - 18s 4s/step - loss: 0.2547 - accuracy: 0.8929 - val_loss: 0.2079 - val_accuracy: 1.0000
```

```
# Evaluate model
results_feature_extraction_model = model.evaluate(test_generator)
results_feature_extraction_model
```

```
1/1 [==============================] - 2s 2s/step - loss: 0.8153 - accuracy: 0.6875
[0.8153325319290161, 0.6875]
```

```
plot_training_history(history_model)
```



FINE TUNING

```
def compare_history(original_history, new_history, initial_epochs=5):
    # Get original history measurements
    acc = original_history.history["accuracy"]
    loss = original_history.history["loss"]

    print(len(acc))

    val_acc = original_history.history["val_accuracy"]
    val_loss = original_history.history["val_loss"]

    # Combine original history with new history
    total_acc = acc + new_history.history["accuracy"]
    total_loss = loss + new_history.history["loss"]

    total_val_acc = val_acc + new_history.history["val_accuracy"]
    total_val_loss = val_loss + new_history.history["val_loss"]

    print(len(total_acc))
    print(total_acc)

    # Make plots
    plt.figure(figsize=(8, 8))
    plt.subplot(2, 1, 1)
    plt.plot(total_acc, label='Training Accuracy')
    plt.plot(total_val_acc, label='Validation Accuracy')
    plt.plot([initial_epochs-1, initial_epochs-1],
              plt.ylim(), label='Start Fine Tuning') # reshift plot around epochs
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(2, 1, 2)
    plt.plot(total_loss, label='Training Loss')
    plt.plot(total_val_loss, label='Validation Loss')
    plt.plot([initial_epochs-1, initial_epochs-1],
              plt.ylim(), label='Start Fine Tuning') # reshift plot around epochs
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.xlabel('epoch')
    plt.show()
```

bcs ModelCheckpoint callback, we saved weights of our already well-performing model

```
# Unfreeze all of the layers in the base model
base_model.trainable = True

# Refreeze every layer except for the last 5
for layer in base_model.layers[:-5]:
  layer.trainable = False


# Setup EarlyStopping callback to stop training if model's val_loss doesn't improve for 3 epochs
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_loss", # watch the val loss metric
                                                  patience=3)
```

```
# Creating learning rate reduction callback
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",
                                                 factor=0.2, # multiply the learning rate by 0.2 (reduce by 5x)
                                                 patience=2,
                                                 verbose=1, # print out when learning rate goes down
                                                 min_lr=1e-7)
```

When fine-tuning and unfreezing layers of pre-trained model, it's common practice to lower the learning rate for feature extraction model.

- 10x lower learning rate is usually a good place to to start.

```
model.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(1e-4), # 10x lower learning rate than default
              metrics=['accuracy'])
```

Checking the Layers

```
for layer in model.layers:
  print(layer.name, layer.trainable)
```

```
input_layer True
inception_v3 True
global_average_pooling True
output_layer True
```

Check which layer are trainable

```
for layer_number, layer in enumerate(base_model.layers):
  print(layer_number, layer.name, layer.trainable)
```

```
0 input_1 False
1 conv2d False
2 batch_normalization False
3 activation False
4 conv2d_1 False
5 batch_normalization_1 False
6 activation_1 False
7 conv2d_2 False
8 batch_normalization_2 False
9 activation_2 False
10 max_pooling2d False
11 conv2d_3 False
12 batch_normalization_3 False
13 activation_3 False
14 conv2d_4 False
15 batch_normalization_4 False
16 activation_4 False
17 max_pooling2d_1 False
18 conv2d_8 False
19 batch_normalization_8 False
20 activation_8 False
21 conv2d_6 False
22 conv2d_9 False
23 batch_normalization_6 False
24 batch_normalization_9 False
25 activation_6 False
26 activation_9 False
27 average_pooling2d False
28 conv2d_5 False
29 conv2d_7 False
30 conv2d_10 False
31 conv2d_11 False
32 batch_normalization_5 False
33 batch_normalization_7 False
34 batch_normalization_10 False
35 batch_normalization_11 False
36 activation_5 False
37 activation_7 False
38 activation_10 False
39 activation_11 False
40 mixed0 False
41 conv2d_15 False
42 batch_normalization_15 False
43 activation_15 False
44 conv2d_13 False
45 conv2d_16 False
46 batch_normalization_13 False
47 batch_normalization_16 False
48 activation_13 False
49 activation_16 False
50 average_pooling2d_1 False
51 conv2d_12 False
52 conv2d_14 False
```

```
53 conv2d_17 False
54 conv2d_18 False
55 batch_normalization_12 False
56 batch_normalization_14 False
57 batch_normalization_17 False
```

```
fine_tune_epochs = 50
history_model_finetune = model.fit(train_generator,
                                   epochs=fine_tune_epochs,
                                   steps_per_epoch=len(train_generator),
                                   validation_data= val_generator,
                                   validation_steps=len(val_generator),
                                   initial_epoch=history_model.epoch[-1], # start from previous last epoch
                                   callbacks=[early_stopping, reduce_lr])
```

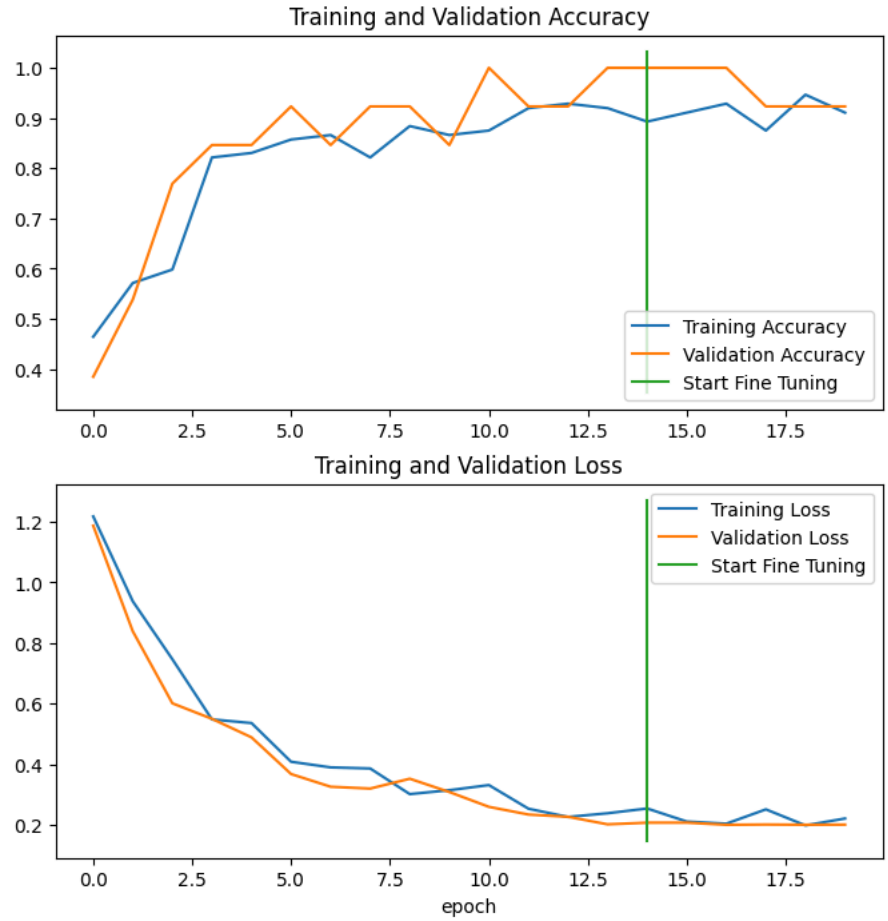
```
Epoch 15/50
4/4 [==============================] - 27s 5s/step - loss: 0.2116 - accuracy: 0.9107 - val_loss: 0.2080 - val_accuracy: 1.0000 - lr
Epoch 16/50
4/4 [==============================] - 18s 4s/step - loss: 0.2046 - accuracy: 0.9286 - val_loss: 0.2009 - val_accuracy: 1.0000 - lr
Epoch 17/50
4/4 [==============================] - 18s 4s/step - loss: 0.2518 - accuracy: 0.8750 - val_loss: 0.2018 - val_accuracy: 0.9231 - lr
Epoch 18/50
4/4 [==============================] - ETA: 0s - loss: 0.1989 - accuracy: 0.9464
Epoch 18: ReduceLROnPlateau reducing learning rate to 1.9999999494757503e-05.
4/4 [==============================] - 22s 5s/step - loss: 0.1989 - accuracy: 0.9464 - val_loss: 0.2011 - val_accuracy: 0.9231 - lr
Epoch 19/50
4/4 [==============================] - 18s 5s/step - loss: 0.2217 - accuracy: 0.9107 - val_loss: 0.2014 - val_accuracy: 0.9231 - lr
```

```
# Evaluate fine-tuned model on the whole test dataset
results_all_classes_finetune = model.evaluate(test_generator)
results_all_classes_finetune
```

```
1/1 [==============================] - 2s 2s/step - loss: 0.9032 - accuracy: 0.7500
[0.9031775593757629, 0.75]
```

```
compare_history(original_history=history_model,
                new_history=history_model_finetune,
                initial_epochs=15)
```

```
15
20
[0.4642857015132904, 0.5714285969734192, 0.5982142686843872, 0.8214285969734192, 0.8303571343421936, 0.8571428656578064, 0.86607140
```



```
pred_probs = model.predict(test_generator, verbose=1)
```

```
1/1 [==============================] - 4s 4s/step
```

```
len(pred_probs) #check sama kek test data apa ga
```

```
16
```

```
pred_probs.shape #check disini dah bener kan ,3
```

```
(16, 3)
```

```
pred_probs[:10] #check bentuk
```

```
array([[9.3140382e-01, 2.4309920e-02, 4.4286232e-02],
       [8.3333457e-01, 4.7880761e-02, 1.1878461e-01],
       [5.3153038e-01, 3.7775144e-02, 4.3069443e-01],
       [8.6913329e-01, 3.7360065e-02, 9.3506709e-02],
       [9.4987887e-01, 1.9862641e-02, 3.0258490e-02],
       [8.1086624e-03, 8.9177907e-01, 1.0011232e-01],
       [1.2307505e-03, 9.8676300e-01, 1.2006282e-02],
       [2.6143140e-01, 5.2681303e-01, 2.1175559e-01],
       [8.2336547e-04, 9.7068226e-01, 2.8494345e-02],
       [2.3795798e-02, 9.6480304e-01, 1.1401229e-02]], dtype=float32)
```

```
print(f"Number of prediction probabilities for sample 0: {len(pred_probs[0])}")
print(f"What prediction probability sample 0 looks like:\n {pred_probs[0]}")
print(f"The class with the highest predicted probability by the model for sample 0: {pred_probs[0].argmax()}")
```

```
Number of prediction probabilities for sample 0: 3
What prediction probability sample 0 looks like:
 [0.9314038  0.02430992 0.04428623]
The class with the highest predicted probability by the model for sample 0: 0
```

```
pred_classes = pred_probs.argmax(axis=1) # ambil class prediction buat tiap label
```

```
pred_classes[:30] # cuman check si mastiin kalo udah bener antara 0/1/2
```

```
array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 2])
```

```
y_labels = test_generator.labels  # This should be much faster
print(y_labels[:10])  # check what they look like (unshuffled)
```

```
[0 0 0 0 0 1 1 1 1 1]
```

```
len(y_labels) # harus sama kek jumlah test
```

```
16
```

```
# Get accuracy score by comparing predicted classes to ground truth labels
from sklearn.metrics import accuracy_score
sklearn_accuracy = accuracy_score(y_labels, pred_classes)
sklearn_accuracy
```

```
0.75
```

```
loaded_loss, loaded_accuracy = model.evaluate(test_generator)
loaded_loss, loaded_accuracy
```

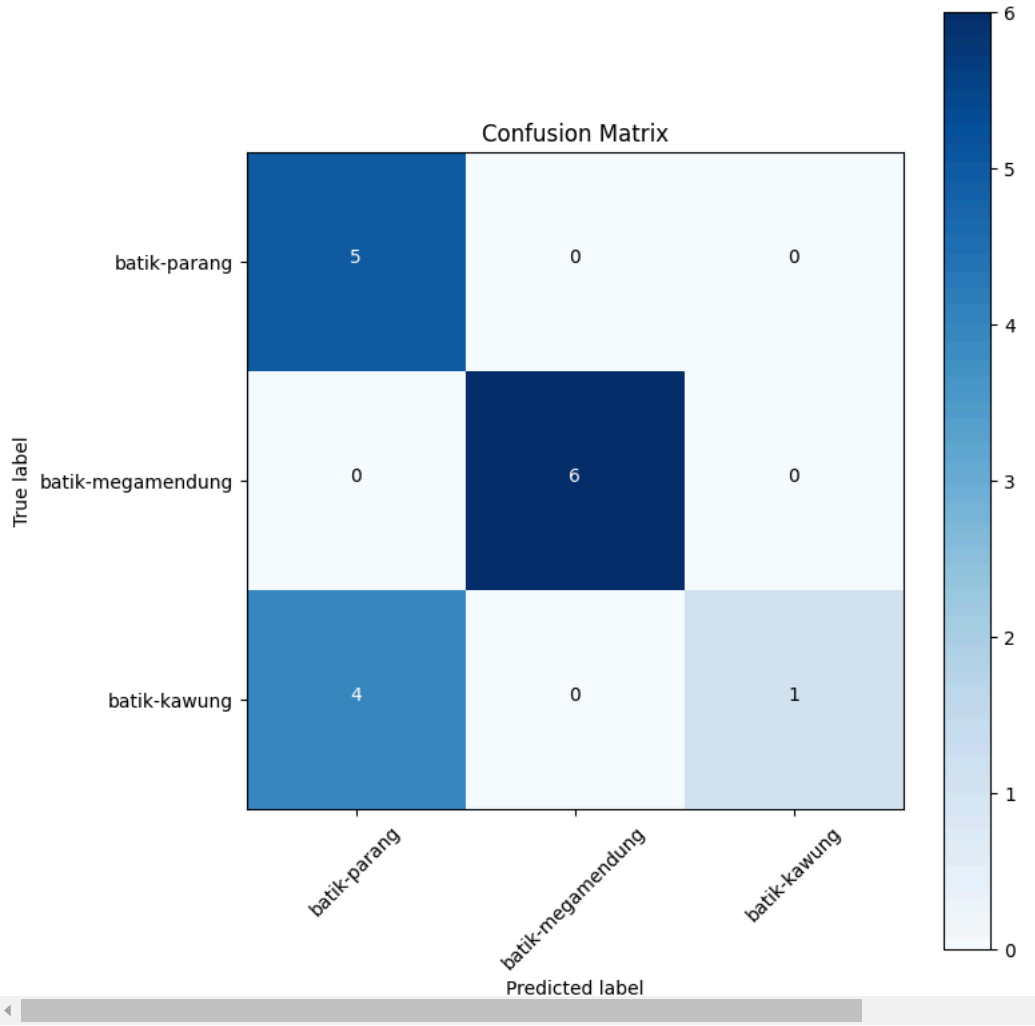
```
1/1 [==============================] - 2s 2s/step - loss: 0.9032 - accuracy: 0.7500
(0.9031775593757629, 0.75)
```

```
# Does the evaluate method compare to the Scikit-Learn measured accuracy?
import numpy as np
print(f"Close? {np.isclose(loaded_accuracy, sklearn_accuracy)} | Difference: {loaded_accuracy - sklearn_accuracy}")
```

```
Close? True | Difference: 0.0
```

```
y_pred = model.predict(test_generator)
y_pred = np.argmax(y_pred, axis=1)
y_true = test_generator.classes

cm = confusion_matrix(y_true, y_pred)
plot_confusion_matrix(cm, classes=CLASS_NAMES, normalize=False, title='Confusion Matrix')
```

```
1/1 [==============================] - 4s 4s/step
```




```
from sklearn.metrics import classification_report
print(classification_report(y_labels, pred_classes))
```

```
              precision    recall  f1-score   support

           0       0.56      1.00      0.71         5
           1       1.00      1.00      1.00         6
           2       1.00      0.20      0.33         5

    accuracy                           0.75        16
   macro avg       0.85      0.73      0.68        16
weighted avg       0.86      0.75      0.70        16
```

```
classification_report_dict = classification_report(y_labels, pred_classes, output_dict=True)
classification_report_dict
```

```
{'0': {'precision': 0.5555555555555556,
  'recall': 1.0,
  'f1-score': 0.7142857142857143,
  'support': 5},
 '1': {'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 6},
 '2': {'precision': 1.0,
  'recall': 0.2,
  'f1-score': 0.3333333333333337,
  'support': 5},
 'accuracy': 0.75,
 'macro avg': {'precision': 0.8518518518518517,
  'recall': 0.7333333333333334,
  'f1-score': 0.6825396825396827,
  'support': 16},
 'weighted avg': {'precision': 0.8611111111111112,
  'recall': 0.75,
  'f1-score': 0.7023809523809523,
  'support': 16}}
```

Since the f1-score combines precision and recall in one metric, let's focus on that.

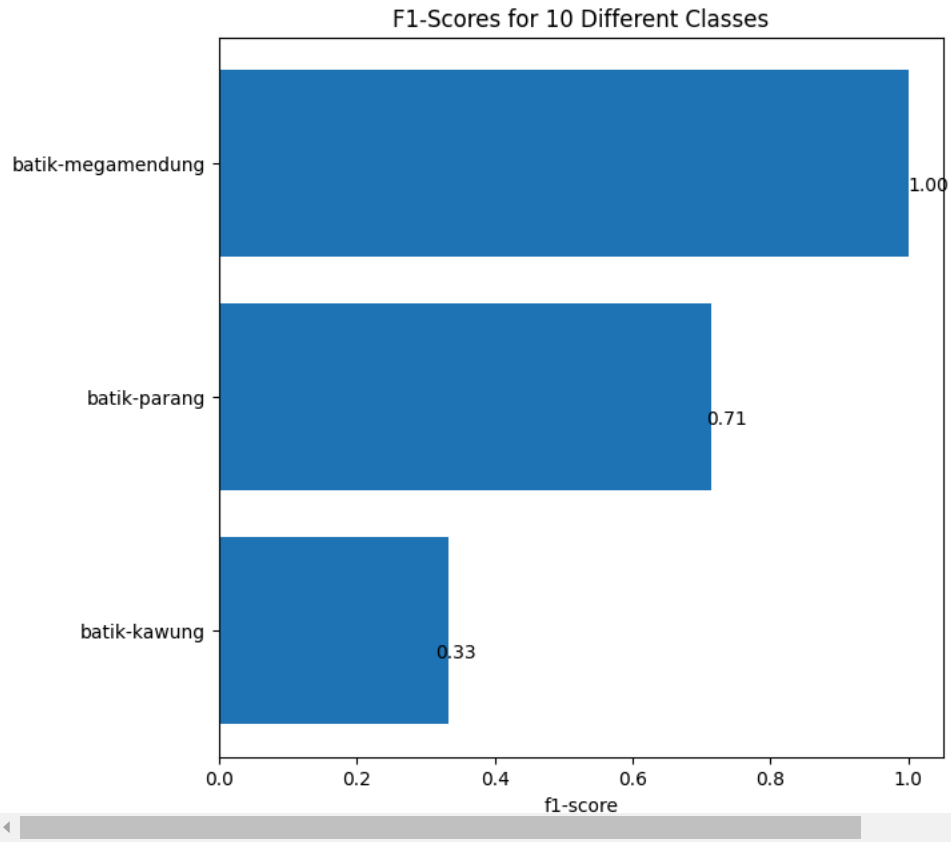
```
# Create empty dictionary
class_f1_scores = {}
# Loop through classification report items
for k, v in classification_report_dict.items():
  if k == "accuracy": # stop once we get to accuracy key
    break
  else:
    # Append class names and f1-scores to new dictionary
    class_f1_scores[CLASS_NAMES[int(k)]] = v["f1-score"]
class_f1_scores
```

```
{'batik-parang': 0.7142857142857143,
 'batik-megamendung': 1.0,
 'batik-kawung': 0.33333333333333337}
```

```
# Turn f1-scores into dataframe for visualization
import pandas as pd
f1_scores = pd.DataFrame({"class_name": list(class_f1_scores.keys()),
                          "f1-score": list(class_f1_scores.values())}).sort_values("f1-score", ascending=False)
f1_scores.head()
```

	class_name	f1-score
1	batik-megamendung	1.000000
0	batik-parang	0.714286
2	batik-kawung	0.333333

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(7, 7))
scores = ax.barh(range(len(f1_scores)), f1_scores["f1-score"].values)
ax.set_yticks(range(len(f1_scores)))
ax.set_yticklabels(list(f1_scores["class_name"]))
ax.set_xlabel("f1-score")
ax.set_title("F1-Scores for 10 Different Classes")
ax.invert_yaxis(); # reverse the order

def autolabel(rects):
  for rect in rects:
    width = rect.get_width()
    ax.text(1.03*width, rect.get_y() + rect.get_height()/1.5,
            f"{width:.2f}",
            ha='center', va='bottom')

autolabel(scores)
```



```
def load_and_prep_image(filename, img_shape=224, scale=True):
  img = tf.io.read_file(filename) # Read in the image
  img = tf.io.decode_image(img) # Decode it into a tensor
  img = tf.image.resize(img, [img_shape, img_shape]) # Resize the image

  if scale:
    # Rescale the image (get all values between 0 and 1)
    return img/255.
  else:
    return img


# Make preds on a series of random images
import os
import random

plt.figure(figsize=(22, 10))
for i in range(3):
  # Choose a random image from a random class
  class_name = random.choice(CLASS_NAMES)
  filename = random.choice(os.listdir(TEST_PATH + "/" + class_name))
  filepath = TEST_PATH + class_name + "/" + filename

  # Load the image and make predictions
  img = load_and_prep_image(filepath, scale=False) # don't scale images for EfficientNet predictions
  pred_prob = model.predict(tf.expand_dims(img, axis=0)) # model accepts tensors of shape [None, 224, 224, 3]
  pred_class = CLASS_NAMES[pred_prob.argmax()] # find the predicted class

  # Plot the image(s)
  plt.subplot(1, 3, i+1)
  plt.imshow(img/255.)
  if class_name == pred_class: # Change the color of text based on whether prediction is right or wrong
    title_color = "g"
  else:
    title_color = "r"
  plt.title(f"actual: {class_name}, pred: {pred_class}, prob: {pred_prob.max():.2f}", c=title_color)
  plt.axis(False);
```

```
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 0s 258ms/step
1/1 [==============================] - 0s 160ms/step
```


```
filepaths = test_generator.filepaths
print(filepaths[:10])
```

```
['/content/drive/MyDrive/AOL/Test/batik-kawung/1.jpg', '/content/drive/MyDrive/AOL/Test/batik-kawung/33.jpg', '/content/drive/MyDriv
```

Now we got all of the test image filepaths, combine them into a DataFrame along with:

- Their ground truth labels (y_labels).
- The class the model predicted (pred_classes).
- The maximum prediction probabilitity value (pred_probs.max(axis=1)).
- The ground truth class names.

- The predicted class names

```
import pandas as pd
pred_df = pd.DataFrame({"img_path": filepaths,
                        "y_true": y_labels,
                        "y_pred": pred_classes,
                        "pred_conf": pred_probs.max(axis=1), # get the maximum prediction probability value
                        "y_true_classname": [CLASS_NAMES[i] for i in y_labels],
                        "y_pred_classname": [CLASS_NAMES[i] for i in pred_classes]})
pred_df.head()
```

	img_path	y_true	y_pred	pred_conf	y_true_classname	y_pred_classname
0	/content/drive/MyDrive/AOL/Test/batik-kawung/1...	0	0	0.931404	batik-parang	batik-parang
1	/content/drive/MyDrive/AOL/Test/batik-kawung/3...	0	0	0.833335	batik-parang	batik-parang
2	/content/drive/MyDrive/AOL/Test/batik-kawung/3...	0	0	0.531530	batik-parang	batik-parang
3	/content/drive/MyDrive/AOL/Test/batik-kawung/3...	0	0	0.869133	batik-parang	batik-parang
4	/content/drive/MyDrive/AOL/Test/batik-kawung/4...	0	0	0.949879	batik-parang	batik-parang

make a simple column to tell whether the prediction is right or wrong?

```
pred_df["pred_correct"] = pred_df["y_true"] == pred_df["y_pred"]
pred_df.head()
```

	img_path	y_true	y_pred	pred_conf	y_true_classname	y_pred_classname	pred_correct
0	/content/drive/MyDrive/AOL/Test/batik-kawung/1...	0	0	0.931404	batik-parang	batik-parang	True
1	/content/drive/MyDrive/AOL/Test/batik-kawung/3...	0	0	0.833335	batik-parang	batik-parang	True
2	/content/drive/MyDrive/AOL/Test/batik-kawung/3...	0	0	0.531530	batik-parang	batik-parang	True
3	/content/drive/MyDrive/AOL/Test/batik-kawung/3...	0	0	0.869133	batik-parang	batik-parang	True
4	/content/drive/MyDrive/AOL/Test/batik-kawung/4...	0	0	0.949879	batik-parang	batik-parang	True

```
#Get the top wrong examples
top_wrong = pred_df[pred_df["pred_correct"] == False].sort_values("pred_conf", ascending=False)[:10]
top_wrong.head()
```

	img_path	y_true	y_pred	pred_conf	y_true_classname	y_pred_classname	pred_correct
14	/content/drive/MyDrive/AOL/Test/batik-parang/4...	2	0	0.986240	batik-kawung	batik-parang	False
11	/content/drive/MyDrive/AOL/Test/batik-parang/1...	2	0	0.918565	batik-kawung	batik-parang	False
13	/content/drive/MyDrive/AOL/Test/batik-parang/3...	2	0	0.885406	batik-kawung	batik-parang	False
12	/content/drive/MyDrive/AOL/Test/batik-parang/1...	2	0	0.670315	batik-kawung	batik-parang	False

```
images_to_view = 9
start_index = 0
plt.figure(figsize=(15, 10))
for i, row in enumerate(top_100_wrong[start_index:start_index+images_to_view].itertuples()):
  plt.subplot(3, 3, i+1)
  img = load_and_prep_image(row[1], scale=True)
  _, _, _, _, pred_prob, y_true, y_pred, _ = row # only interested in a few parameters of each row
  plt.imshow(img)
  plt.title(f"actual: {y_true}, pred: {y_pred} \nprob: {pred_prob:.2f}")
  plt.axis(False)
```

Analyzing Incorrect Predictions Reviewing a model's most incorrect predictions can highlight important areas for improvement:

1. Label Accuracy: Correcting Mislabeled Data: Consistent incorrect predictions might indicate errors in the ground truth labels. By identifying and correcting these, we can improve the dataset's quality, a process known as active learning.
2. Data Collection: Adding More Samples: If a model struggles with certain classes, it suggests a need for more samples of those classes. Collecting additional data can help the model learn better and improve its predictions. By addressing these areas, we can refine our dataset and enhance the model's performance.

```
import os

folder_path = "/content/drive/MyDrive/AOL/random_img/"

custom_food_images = [os.path.join(folder_path, img_path) for img_path in os.listdir(folder_path)]
custom_food_images
```

```
['/content/drive/MyDrive/AOL/random_img/th (4).jpeg',
 '/content/drive/MyDrive/AOL/random_img/th (3).jpeg',
 '/content/drive/MyDrive/AOL/random_img/th.jpeg',
 '/content/drive/MyDrive/AOL/random_img/th (1).jpeg',
 '/content/drive/MyDrive/AOL/random_img/th (2).jpeg']
```

```
for img in custom_food_images:
  img = load_and_prep_image(img, scale=False) # load in target image and turn it into tensor
  pred_prob = model.predict(tf.expand_dims(img, axis=0)) # make prediction on image with shape [None, 224, 224, 3]
  pred_class = CLASS_NAMES[pred_prob.argmax()] # find the predicted class label
  # Plot the image with appropriate annotations
  plt.figure()
  plt.imshow(img/255.) # imshow() requires float inputs to be normalized
  plt.title(f"pred: {pred_class}, prob: {pred_prob.max():.2f}")
  plt.axis(False)

# Ki ketoke mang ancen kawung ki meresahkan og, tapi ki aku sengaja cari sg megamendung 3 biru 2 warna laen, ternyata tenanan model e gł
```

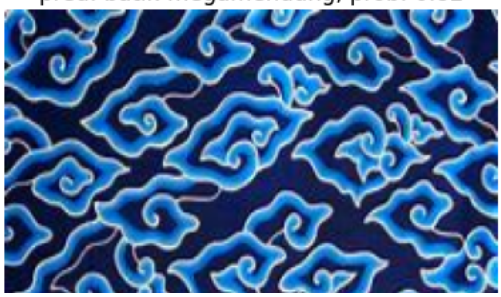
```
1/1 [==============================] - 0s 254ms/step
1/1 [==============================] - 0s 467ms/step
1/1 [==============================] - 0s 247ms/step
1/1 [==============================] - 0s 254ms/step
1/1 [==============================] - 0s 252ms/step
```

pred: batik-kawung, prob: 1.00

pred: batik-kawung, prob: 1.00

pred: batik-megamendung, prob: 0.95

pred: batik-megamendung, prob: 0.82



pred: batik-megamendung, prob: 0.51

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit