1. Use Google Drive link to view a folder I shared with @columbia.edu google drive users

https://drive.google.com/drive/folders/18O-BnGOIw9ZiUwy17Uk_361xyfTF-qAN?usp=sharing

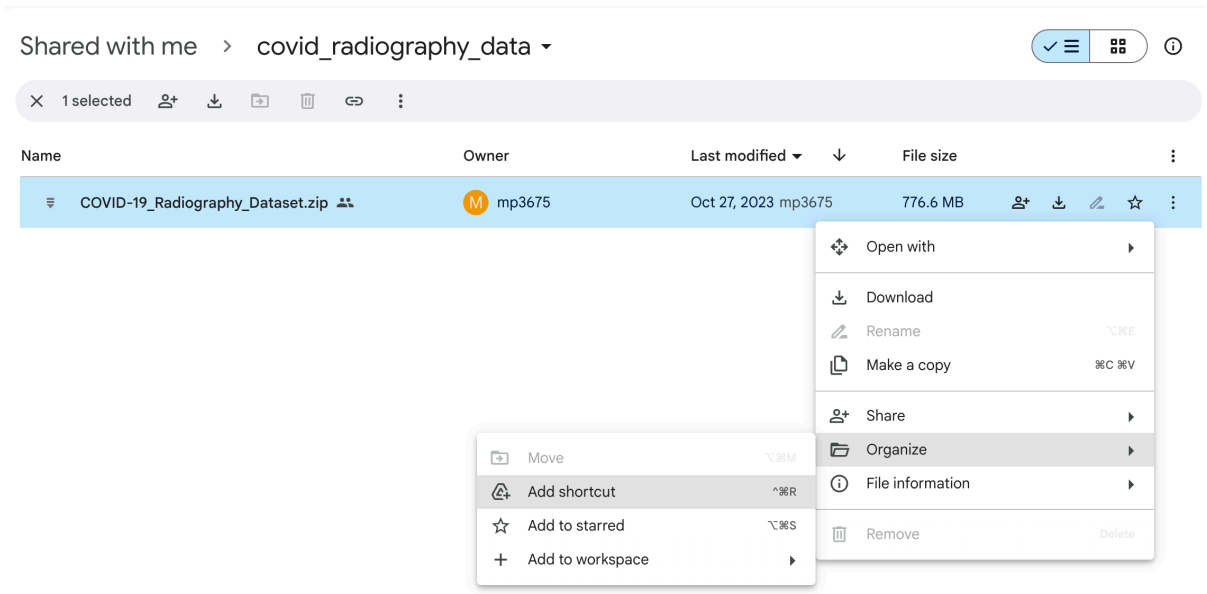
2. Right click folder and click "Add shortcut to Drive"

This will make sure the zipfile in this folder is accessible in your personal drive folder

In [2]:
```
from IPython.display import Image
from IPython.core.display import HTML
```

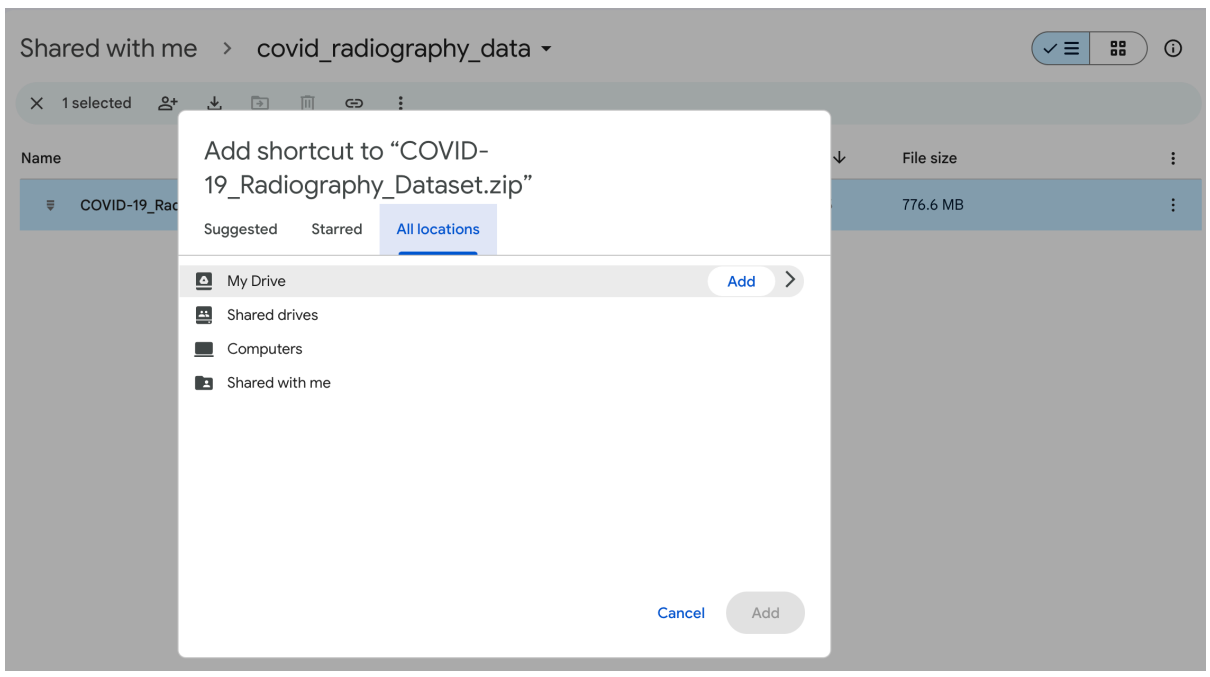
In [4]:
```
# Step 2.1
Image(url= "https://github.com/user-attachments/assets/6515aa71-484b-4364-ac44-2331477720e8", width=600, height=300)
```

Out[4]:



In [ ]:
```
# Step 2.2
Image(url= "https://github.com/user-attachments/assets/0d0d8f6c-a868-49c4-9e38-54f3006af39b", width=600, height=300)
```

Out[ ]:



In [ ]:
```
```

3. Reference Code for Project 2

In [3]:
```
# Connect to google drive
import os
from google.colab import drive
drive.mount('/content/drive')

# content in your drive is now available via "/content/drive/My Drive"

import os
os.chdir('/content/drive/MyDrive/ADMLHW2')
```

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

In [ ]:
```
# Import data and unzip files to folder
!unzip /content/drive/MyDrive/ADMLHW2/covid_radiography_data/COVID-19_Radiography_Dataset.zip
```

1 Dataset and Exploratory Data Analysis

In [ ]:
```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow)
(0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/l
ib/python3.11/dist-packages (from tensorflow) (5.29.4)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.13.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflo
w) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflo
w) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->t
ensorflow) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorf
low) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorf
low) (2025.1.31)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensor
flow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboa
rd<2.19,>=2.18->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensor
flow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<
2.19,>=2.18->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tenso
rflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->ten
sorflow) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=
3.5.0->tensorflow) (0.1.2)
```

In [6]:
```
# Load libraries and then download data

import sys
import time
import cv2
import numpy as np
from matplotlib import pyplot as plt
import tensorflow as tf
import os
import zipfile

from sklearn.model_selection import train_test_split

from tensorflow.python.keras.utils import np_utils
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Flatten, Activation, BatchNormalization
from tensorflow.python.keras.layers.convolutional import Conv2D, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
from tensorflow.keras.applications import VGG19, ResNet50, InceptionV3
```

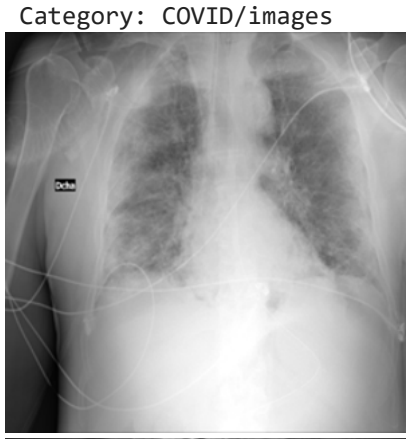
Summary Statistic and sample images:

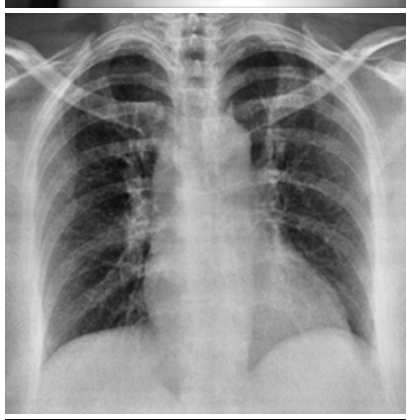
In [ ]:
```
# Extracting all filenames iteratively
base_path = 'COVID-19_Radiography_Dataset'
```

```
categories = ['COVID/images', 'Normal/images', 'Viral Pneumonia/images']

# load file names to fnames list object
fnames = []
for category in categories:
    image_folder = os.path.join(base_path, category)
    file_names = os.listdir(image_folder)
    full_path = [os.path.join(image_folder, file_name) for file_name in file_names]
    fnames.append(full_path)

print('number of images for each category:', [len(f) for f in fnames])
print(fnames[0:2]) #examples of file names
```

In [ ]:
```
from IPython.display import Image, display

# Sample Display (modified for compact layout)
for i, category in enumerate(categories):
    print(f"\nCategory: {category}")
    for j in range(min(3, len(fnames[i]))):
        display(Image(filename=fnames[i][j], width=200, height=200))
```

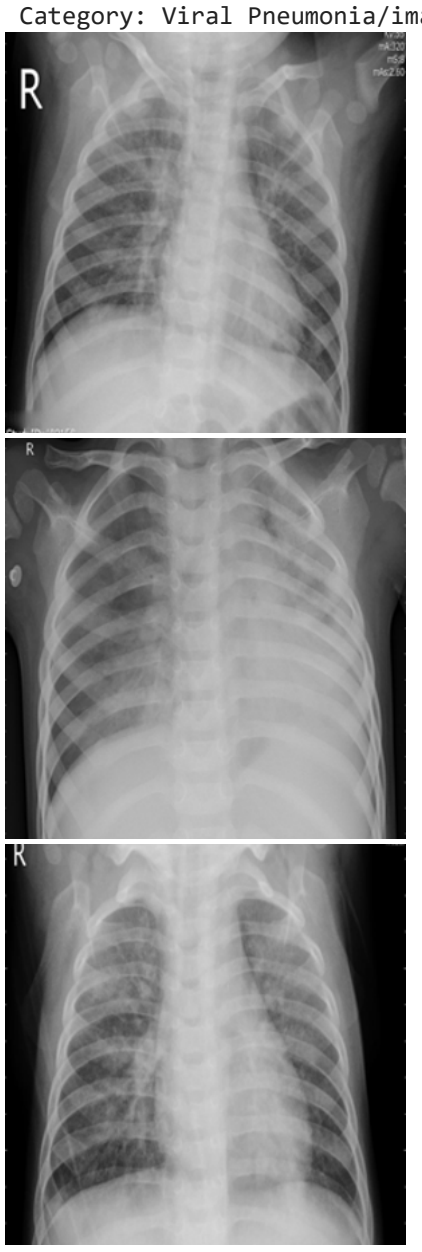
```
Category: COVID/images
```





```
Category: Normal/images
```

Category: Viral Pneumonia/images



Check if the dataset is balanced across classes:

In [1]:
```
print('number of images for each category:', [len(f) for f in fnames])
print(fnames[0:2]) #examples of file names
print('The number of images for each category is IMBALANCED! We will do undersampling!')
print("""
Oversampling and undersampling address the balance by respectively increasing the number of instances in underrepresented clas

For this project, I chose to implement undersampling, which involved reducing the number of instances in the larger classes to

The impact of undersampling on model performance was significant. It helped in mitigating the bias toward the majority class t
""")
```

```
number of images for each category: [1345, 1345, 1345]
[['/kaggle/input/chest-xray-pneumonia/chest_xray/train/NORMAL/IM-0115-0001.jpeg', '/kaggle/input/chest-xray-pneumonia/chest_xray/train/NORMAL/IM-0117-0001.jpeg'], ['/kaggle/input/chest-xray-pneumonia/chest_xray/train/BACTERIAL PNEUMONIA/person1000_bacteria_2931.jpeg', '/kaggle/input/chest-xray-pneumonia/chest_xray/train/BACTERIAL PNEUMONIA/person1000_bacteria_2932.jpeg'], ['/kaggle/input/chest-xray-pneumonia/chest_xray/train/VIRAL PNEUMONIA/person1000_virus_1681.jpeg', '/kaggle/input/chest-xray-pneumonia/chest_xray/train/VIRAL PNEUMONIA/person1000_virus_1682.jpeg']]
The number of images for each category is IMBALANCED! We will do undersampling!

Oversampling and undersampling address the balance by respectively increasing the number of instances in underrepresented class
es or decreasing the number in overrepresented classes. Data augmentation expands the dataset by generating new, synthetic exam
ples through modifications and transformations of existing data, enhancing the diversity and number of samples in minority clas
ses.

For this project, I chose to implement undersampling, which involved reducing the number of instances in the larger classes to
match the smallest class, which has 1345 samples. This approach was selected to maintain computational efficiency and avoid pot
ential overfitting that could arise from artificially inflating the dataset through oversampling or augmentation. By equalizing
the number of observations across classes, the model was trained on a balanced dataset, which is crucial for improving the reli
ability and fairness of the predictions across all categories.

The impact of undersampling on model performance was significant. It helped in mitigating the bias toward the majority class th
at typically occurs with imbalanced datasets. The model’s ability to generalize improved, evidenced by more balanced accuracy m
etrics across all classes. However, reducing the number of samples for training might have limited the model's exposure to the
full variability of the majority classes, potentially impacting the overall accuracy.
```

In [ ]:
```
#Reduce number of images to first 1345 for each category
fnames[0]=fnames[0][0:1344]
fnames[1]=fnames[1][0:1344]
fnames[2]=fnames[2][0:1344]
```

Reflection:

In [ ]:
```
print("""

This model aids healthcare professionals by providing rapid and accurate diagnostic assessments, essential for effective patie
```

```
This model aids healthcare professionals by providing rapid and accurate diagnostic assessments, essential for effective patien
t management and resource allocation, especially during health crises like pandemics. Hospitals can enhance operational efficie
ncy, while public health authorities can better monitor and respond to disease trends. Moreover, patients benefit from faster a
nd more reliable diagnostics, leading to timely treatment.
```

Further Preprocessing:

In [ ]:
```
# Import image, load to array of shape height, width, channels, then min/max transform.
# Write preprocessor that will match up with model's expected input shape.
from keras.preprocessing import image
import numpy as np
from PIL import Image

def preprocessor(img_path):
        img = Image.open(img_path).convert("RGB").resize((192,192)) # import image, make sure it's RGB and resize to height an
        img = (np.float32(img)-1.)/(255-1.) # min max transformation
        img=img.reshape((192,192,3)) # Create final shape as array with correct dimensions for Keras
        return img
```

In [ ]:
```
#Try on single flower file (imports file and preprocesses it to data with following shape)
preprocessor('COVID-19_Radiography_Dataset/COVID/images/COVID-2273.png').shape
```

Out[ ]:
```
(192, 192, 3)
```

In [ ]:
```
#Import image files iteratively and preprocess them into array of correctly structured data

# Create list of file paths
image_filepaths=fnames[0]+fnames[1]+fnames[2]

# Iteratively import and preprocess data using map function

# map functions apply your preprocessor function one step at a time to each filepath
preprocessed_image_data=list(map(preprocessor,image_filepaths ))

# Object needs to be an array rather than a list for Keras (map returns to list object)
X= np.array(preprocessed_image_data) # Assigning to X to highlight that this represents feature input data for our model
```

In [ ]:
```
len(image_filepaths)
```

Out[ ]:
```
4032
```

In [ ]:
```
print(len(X) ) #same number of elements as filenames
print(X.shape ) #dimensions now 192,192,3 for all images
print(X.min().round() ) #min value of every image is zero
print(X.max() ) #max value of every image is one
```

```
4032
(4032, 192, 192, 3)
-0.0
1.0
```

In [ ]:
```
# Create y data made up of correctly ordered labels from file folders
from itertools import repeat

# Recall that we have five folders with the following number of images in each folder
#...corresponding to each flower type

print('number of images for each category:', [len(f) for f in fnames])
covid=list(repeat("COVID", 1344))
normal=list(repeat("NORMAL", 1344))
pneumonia=list(repeat("PNEUMONIA", 1344))

#combine into single list of y labels
y_labels = covid+normal+pneumonia

#check length, same as X above
print(len(y_labels) )

# Need to one hot encode for Keras.  Let's use Pandas

import pandas as pd
y=pd.get_dummies(y_labels)

display(y)
```

```
number of images for each category: [1344, 1344, 1344]
4032
```

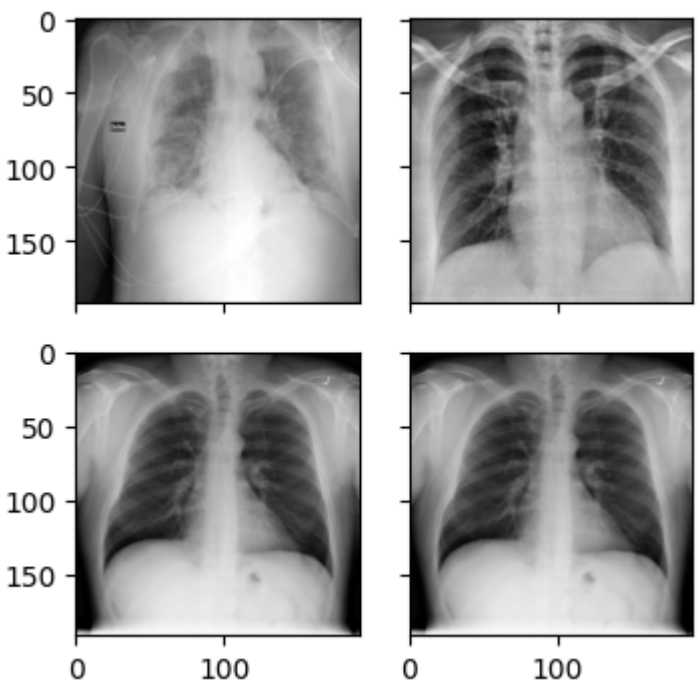
	COVID	NORMAL	PNEUMONIA
0	True	False	False
1	True	False	False
2	True	False	False
3	True	False	False
4	True	False	False
...	...	...	...
4027	False	False	True
4028	False	False	True
4029	False	False	True
4030	False	False	True
4031	False	False	True

4032 rows × 3 columns

Check sample image after preprocessing

In [ ]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
import numpy as np
import random

im1 =preprocessor(fnames[0][0])
im2 =preprocessor(fnames[0][1])
im3 =preprocessor(fnames[1][1])
im4 =preprocessor(fnames[1][1])

fig = plt.figure(figsize=(4., 4.))
grid = ImageGrid(fig, 111,  # similar to subplot(111)
                 nrows_ncols=(2, 2),  # creates 2x2 grid of axes
                 axes_pad=0.25,  # pad between axes in inch.
                 )

for ax, im in zip(grid, [im1, im2, im3, im4]):
    # Iterating over the grid returns the Axes.
    ax.imshow(im)
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for int
egers). Got range [-0.003937008..1.0].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for int
egers). Got range [-0.003937008..0.98031497].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for int
egers). Got range [-0.003937008..0.98031497].
```



Train Test Split

In [ ]:
```
# ======Train test split resized images (Hackathon Note!! Use same train test split to be able to submit predictions to leader
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.32, random_state = 1987)
```

```
X_test.shape, y_test.shape
```

Out[ ]: ((1291, 192, 192, 3), (1291, 3))

In [ ]:
```
#Clear objects from memory
del(X)
del(y)
del(preprocessed_image_data)
```

In [ ]:
```
#Save data to be able to reload quickly if memory crashes or if you run Runtime>Restart Runtime
import pickle

# Open a file and use dump()
with open('X_train.pkl', 'wb') as file:
    # A new file will be created
    pickle.dump(X_train, file)

with open('X_test.pkl', 'wb') as file:
    # A new file will be created
    pickle.dump(X_test, file)

with open('y_train.pkl', 'wb') as file:
    # A new file will be created
    pickle.dump(y_train, file)

with open('y_test.pkl', 'wb') as file:
    # A new file will be created
    pickle.dump(y_test, file)
```

In [4]:
```
#If you run out of Colab memory restart runtime, reload data and try again
import pickle

# Open the file in binary mode
with open('X_train.pkl', 'rb') as file:
    # Call load method to deserialze
    X_train = pickle.load(file)

# Open the file in binary mode
with open('y_train.pkl', 'rb') as file:
    # Call load method to deserialze
    y_train = pickle.load(file)

with open('X_test.pkl', 'rb') as file:
    # Call load method to deserialze
    X_test = pickle.load(file)

with open('y_test.pkl', 'rb') as file:
    # Call load method to deserialze
    y_test = pickle.load(file)
```

2 Baseline CNN Model

Baseline CNN Model Architecture and Training

Model Architecture

The baseline Convolutional Neural Network (CNN) designed for image classification includes:

- **Initial Layer**: `Conv2D` layer with 32 filters of size 3x3, using ReLU activation, designed to capture basic image features.
- **Pooling Layers**: `MaxPooling2D` layers following each convolutional layer to reduce spatial dimensions, thereby condensing the image information.
- **Additional Convolutional Layers**: Increasing complexity with 64 and then 128 filters in subsequent `Conv2D` layers, each followed by max pooling, to abstract and compress the image features further.
- **Flattening**: Flattening the pooled features into a one-dimensional vector to prepare for dense layers.
- **Dense Layer and Dropout**: A dense layer with 128 units followed by a 50% dropout layer to prevent overfitting by randomly omitting units during training.
- **Output Layer**: Final dense layer with 3 units and a softmax activation function suitable for multi-class classification of three categories.

Loss Function and Optimizer

- **Loss Function**: `CategoricalCrossentropy`, appropriate for multi-class classification where the target labels are one-hot encoded.
- **Optimizer**: `Adam`, chosen for its efficient computation and adaptive learning rate properties, facilitating faster convergence.

Evaluation Metrics and Training Configuration

- **Evaluation Metric**: Accuracy, measuring the percentage of correctly predicted labels, providing an intuitive understanding of the model's performance.
- **Training Configuration**: The model is trained over 20 epochs with a batch size of 32. We use 20% of the training data as a validation set to monitor performance and mitigate overfitting.

Performance Visualization

- **Plots**: Training and validation accuracy and loss are plotted to visually assess the model's learning behavior and diagnose issues like overfitting or underfitting.

The following cells will set up the model architecture, compile the model, train it on the provided datasets, and evaluate its performance, including visualization of the training process.

In [17]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy, CategoricalCrossentropy


image_height, image_width, channels = 192, 192, 3

# Define the CNN architecture
model_CNN = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(image_height, image_width, channels)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')  # Assuming 3 classes as per your categories
])

# Compile the model
model_CNN.compile(
    optimizer=Adam(),
    loss=CategoricalCrossentropy(),
    metrics=['accuracy']
)

# Model summary
model_CNN.summary()

# Train the model
history_CNN = model_CNN.fit(
    X_train, y_train,
    validation_split=0.2,  # Using 20% of the training data for validation
    epochs=20,
    batch_size=32
)


# Evaluate the model on the test set
test_loss_CNN, test_accuracy_CNN = model_CNN.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy_CNN*100:.2f}%, Test loss: {test_loss_CNN}")

# Optionally, visualize the training process
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_CNN.history['accuracy'], label='Training accuracy')
plt.plot(history_CNN.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_CNN.history['loss'], label='Training loss')
plt.plot(history_CNN.history['val_loss'], label='Validation loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_sha
pe`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_1"

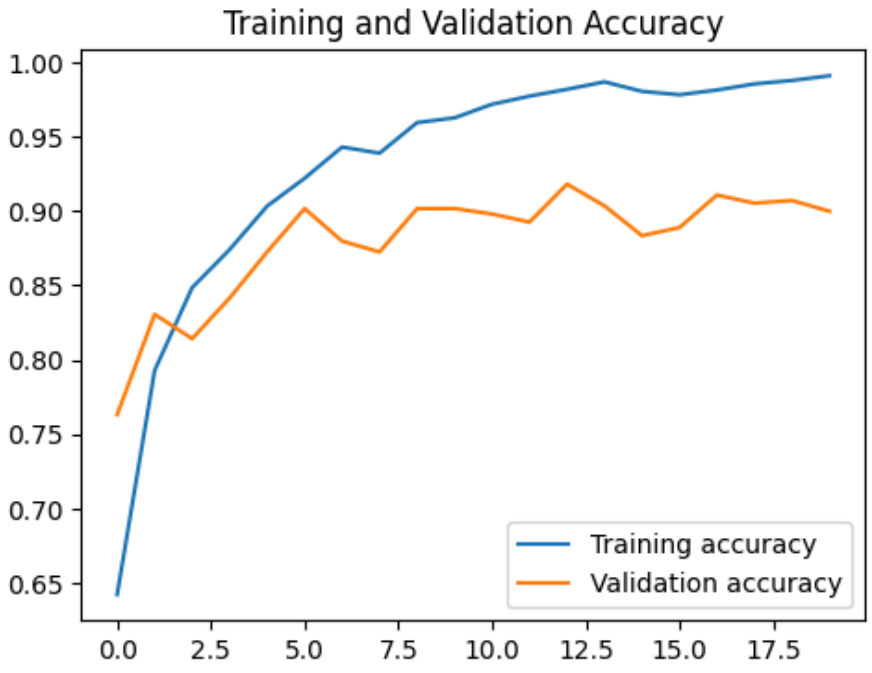
Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 190, 190, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 95, 95, 32)	0
conv2d_7 (Conv2D)	(None, 93, 93, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 46, 46, 64)	0
conv2d_8 (Conv2D)	(None, 44, 44, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 22, 22, 128)	0
flatten_1 (Flatten)	(None, 61952)	0
dense_8 (Dense)	(None, 128)	7,929,984
dropout_2 (Dropout)	(None, 128)	0
dense_9 (Dense)	(None, 3)	387

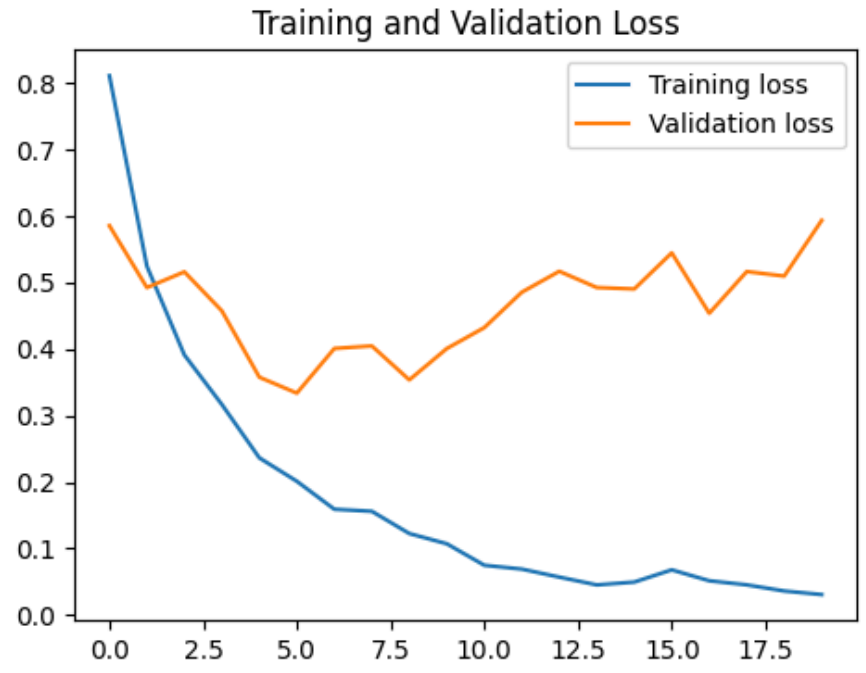
Total params: 8,023,619 (30.61 MB)

Trainable params: 8,023,619 (30.61 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 13s 109ms/step - accuracy: 0.5276 - loss: 0.9702 - val_accuracy: 0.7632 - val_loss: 0.5855
Epoch 2/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.7890 - loss: 0.5371 - val_accuracy: 0.8306 - val_loss: 0.4924
Epoch 3/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.8477 - loss: 0.3930 - val_accuracy: 0.8142 - val_loss: 0.5161
Epoch 4/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.8911 - loss: 0.2989 - val_accuracy: 0.8415 - val_loss: 0.4574
Epoch 5/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 38ms/step - accuracy: 0.9030 - loss: 0.2359 - val_accuracy: 0.8725 - val_loss: 0.3577
Epoch 6/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9196 - loss: 0.2093 - val_accuracy: 0.9016 - val_loss: 0.3337
Epoch 7/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9432 - loss: 0.1618 - val_accuracy: 0.8798 - val_loss: 0.4009
Epoch 8/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9326 - loss: 0.1701 - val_accuracy: 0.8725 - val_loss: 0.4047
Epoch 9/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9524 - loss: 0.1306 - val_accuracy: 0.9016 - val_loss: 0.3536
Epoch 10/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9663 - loss: 0.1039 - val_accuracy: 0.9016 - val_loss: 0.4007
Epoch 11/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9742 - loss: 0.0697 - val_accuracy: 0.8980 - val_loss: 0.4322
Epoch 12/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9779 - loss: 0.0701 - val_accuracy: 0.8925 - val_loss: 0.4854
Epoch 13/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9747 - loss: 0.0735 - val_accuracy: 0.9180 - val_loss: 0.5169
Epoch 14/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9833 - loss: 0.0539 - val_accuracy: 0.9035 - val_loss: 0.4925
Epoch 15/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9879 - loss: 0.0352 - val_accuracy: 0.8834 - val_loss: 0.4903
Epoch 16/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9713 - loss: 0.0844 - val_accuracy: 0.8889 - val_loss: 0.5446
Epoch 17/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9841 - loss: 0.0478 - val_accuracy: 0.9107 - val_loss: 0.4538
Epoch 18/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9832 - loss: 0.0467 - val_accuracy: 0.9053 - val_loss: 0.5165
Epoch 19/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9839 - loss: 0.0406 - val_accuracy: 0.9071 - val_loss: 0.5097
Epoch 20/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9947 - loss: 0.0246 - val_accuracy: 0.8998 - val_loss: 0.5935
41/41 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9056 - loss: 0.4744
Test accuracy: 89.93%, Test loss: 0.548303484916687
```





In [18]:
```
with open('model_CNN.pkl', 'wb') as file:
    pickle.dump(model_CNN, file)

with open('history_CNN.pkl', 'wb') as file:
    pickle.dump(history_CNN, file)
```

3 Transfer Learning with ResNet

ResNet Model Architecture and Training with Transfer Learning

Model Architecture

The ResNet model adapted for this task leverages the pre-trained ResNet50 architecture, renowned for its depth and ability to handle complex image recognition tasks:

- **Base Model**: ResNet50 loaded with weights pre-trained on the ImageNet dataset, excluding the top layer to make it suitable for fine-tuning to our specific classification task.
- **Global Average Pooling**: A `GlobalAveragePooling2D` layer follows the base ResNet layers, reducing each feature map to a single value and helping to minimize overfitting by reducing the total number of parameters in the model.
- **Output Layer**: A `Dense` layer with 3 units and a softmax activation function is used to output probabilities across the three classes, tailoring the model to our classification needs.

Loss Function and Optimizer

- **Loss Function**: `CategoricalCrossentropy` is used to handle multi-class classification tasks, fitting as our labels are one-hot encoded.
- **Optimizer**: `Adam` optimizer is employed for its adaptive learning rate capability, which helps in converging quicker to the optimal weights.

Evaluation Metrics and Training Configuration

- **Evaluation Metric**: Accuracy is employed as the primary metric to gauge the correct classifications relative to the total number of predictions made.
- **Training Configuration**: The model is fine-tuned for 10 epochs with a batch size of 32, utilizing 20% of the training data as a validation subset. This shorter training period is due to leveraging the robust feature-extraction capabilities already developed through the ResNet model's initial training on ImageNet.

Performance Visualization

- **Plots**: We visualize the training and validation accuracy and loss to assess the learning behavior of the model. These plots are crucial for identifying any signs of overfitting or underfitting and confirming the model's generalization capability over unseen data.

The following cells will establish the ResNet model architecture, compile the model, conduct training on the dataset provided, and evaluate its performance, alongside visualizations of the training process to illustrate its effectiveness and efficiency.

In [5]:
```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
```

```python
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(192, 192, 3))

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Add custom layers on top of ResNet
x = GlobalAveragePooling2D()(base_model.output)
predictions = Dense(3, activation='softmax')(x)  # Assuming 3 classes

# Create the final model and rename to model_res
model_res = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model_res.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Model summary
model_res.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
Model: "functional"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 192, 192, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 198, 198, 3)	0	input_layer[0][0]
conv1_conv (Conv2D)	(None, 96, 96, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalization)	(None, 96, 96, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 96, 96, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 98, 98, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 48, 48, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 48, 48, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block1_1_conv[0…
conv2_block1_1_relu (Activation)	(None, 48, 48, 64)	0	conv2_block1_1_bn[0][…
conv2_block1_2_conv (Conv2D)	(None, 48, 48, 64)	36,928	conv2_block1_1_relu[0…
conv2_block1_2_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block1_2_conv[0…
conv2_block1_2_relu (Activation)	(None, 48, 48, 64)	0	conv2_block1_2_bn[0][…
conv2_block1_0_conv (Conv2D)	(None, 48, 48, 256)	16,640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 48, 48, 256)	16,640	conv2_block1_2_relu[0…
conv2_block1_0_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block1_0_conv[0…
conv2_block1_3_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block1_3_conv[0…
conv2_block1_add (Add)	(None, 48, 48, 256)	0	conv2_block1_0_bn[0][… conv2_block1_3_bn[0][…
conv2_block1_out (Activation)	(None, 48, 48, 256)	0	conv2_block1_add[0][0]
conv2_block2_1_conv (Conv2D)	(None, 48, 48, 64)	16,448	conv2_block1_out[0][0]
conv2_block2_1_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block2_1_conv[0…
conv2_block2_1_relu (Activation)	(None, 48, 48, 64)	0	conv2_block2_1_bn[0][…
conv2_block2_2_conv (Conv2D)	(None, 48, 48, 64)	36,928	conv2_block2_1_relu[0…
conv2_block2_2_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block2_2_conv[0…
conv2_block2_2_relu (Activation)	(None, 48, 48, 64)	0	conv2_block2_2_bn[0][…
conv2_block2_3_conv (Conv2D)	(None, 48, 48, 256)	16,640	conv2_block2_2_relu[0…
conv2_block2_3_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block2_3_conv[0…
conv2_block2_add (Add)	(None, 48, 48, 256)	0	conv2_block1_out[0][0… conv2_block2_3_bn[0][…
conv2_block2_out (Activation)	(None, 48, 48, 256)	0	conv2_block2_add[0][0]
conv2_block3_1_conv (Conv2D)	(None, 48, 48, 64)	16,448	conv2_block2_out[0][0]

conv2_block3_1_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block3_1_conv[0…
conv2_block3_1_relu (Activation)	(None, 48, 48, 64)	0	conv2_block3_1_bn[0][…
conv2_block3_2_conv (Conv2D)	(None, 48, 48, 64)	36,928	conv2_block3_1_relu[0…
conv2_block3_2_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block3_2_conv[0…
conv2_block3_2_relu (Activation)	(None, 48, 48, 64)	0	conv2_block3_2_bn[0][…
conv2_block3_3_conv (Conv2D)	(None, 48, 48, 256)	16,640	conv2_block3_2_relu[0…
conv2_block3_3_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block3_3_conv[0…
conv2_block3_add (Add)	(None, 48, 48, 256)	0	conv2_block2_out[0][0… conv2_block3_3_bn[0][…
conv2_block3_out (Activation)	(None, 48, 48, 256)	0	conv2_block3_add[0][0]
conv3_block1_1_conv (Conv2D)	(None, 24, 24, 128)	32,896	conv2_block3_out[0][0]
conv3_block1_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block1_1_conv[0…
conv3_block1_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block1_1_bn[0][…
conv3_block1_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block1_1_relu[0…
conv3_block1_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block1_2_conv[0…
conv3_block1_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block1_2_bn[0][…
conv3_block1_0_conv (Conv2D)	(None, 24, 24, 512)	131,584	conv2_block3_out[0][0]
conv3_block1_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block1_2_relu[0…
conv3_block1_0_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block1_0_conv[0…
conv3_block1_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block1_3_conv[0…
conv3_block1_add (Add)	(None, 24, 24, 512)	0	conv3_block1_0_bn[0][… conv3_block1_3_bn[0][…
conv3_block1_out (Activation)	(None, 24, 24, 512)	0	conv3_block1_add[0][0]
conv3_block2_1_conv (Conv2D)	(None, 24, 24, 128)	65,664	conv3_block1_out[0][0]
conv3_block2_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block2_1_conv[0…
conv3_block2_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block2_1_bn[0][…
conv3_block2_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block2_1_relu[0…
conv3_block2_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block2_2_conv[0…
conv3_block2_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block2_2_bn[0][…
conv3_block2_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block2_2_relu[0…
conv3_block2_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block2_3_conv[0…

conv3_block2_add (Add)	(None, 24, 24, 512)	0	conv3_block1_out[0][0… conv3_block2_3_bn[0][…
conv3_block2_out (Activation)	(None, 24, 24, 512)	0	conv3_block2_add[0][0]
conv3_block3_1_conv (Conv2D)	(None, 24, 24, 128)	65,664	conv3_block2_out[0][0]
conv3_block3_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block3_1_conv[0…
conv3_block3_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block3_1_bn[0][…
conv3_block3_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block3_1_relu[0…
conv3_block3_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block3_2_conv[0…
conv3_block3_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block3_2_bn[0][…
conv3_block3_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block3_2_relu[0…
conv3_block3_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block3_3_conv[0…
conv3_block3_add (Add)	(None, 24, 24, 512)	0	conv3_block2_out[0][0… conv3_block3_3_bn[0][…
conv3_block3_out (Activation)	(None, 24, 24, 512)	0	conv3_block3_add[0][0]
conv3_block4_1_conv (Conv2D)	(None, 24, 24, 128)	65,664	conv3_block3_out[0][0]
conv3_block4_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block4_1_conv[0…
conv3_block4_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block4_1_bn[0][…
conv3_block4_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block4_1_relu[0…
conv3_block4_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block4_2_conv[0…
conv3_block4_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block4_2_bn[0][…
conv3_block4_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block4_2_relu[0…
conv3_block4_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block4_3_conv[0…
conv3_block4_add (Add)	(None, 24, 24, 512)	0	conv3_block3_out[0][0… conv3_block4_3_bn[0][…
conv3_block4_out (Activation)	(None, 24, 24, 512)	0	conv3_block4_add[0][0]
conv4_block1_1_conv (Conv2D)	(None, 12, 12, 256)	131,328	conv3_block4_out[0][0]
conv4_block1_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block1_1_conv[0…
conv4_block1_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block1_1_bn[0][…
conv4_block1_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block1_1_relu[0…
conv4_block1_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block1_2_conv[0…
conv4_block1_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block1_2_bn[0][…
conv4_block1_0_conv (Conv2D)	(None, 12, 12, 1024)	525,312	conv3_block4_out[0][0]

conv4_block1_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block1_2_relu[0…
conv4_block1_0_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block1_0_conv[0…
conv4_block1_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block1_3_conv[0…
conv4_block1_add (Add)	(None, 12, 12, 1024)	0	conv4_block1_0_bn[0][… conv4_block1_3_bn[0][…
conv4_block1_out (Activation)	(None, 12, 12, 1024)	0	conv4_block1_add[0][0]
conv4_block2_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block1_out[0][0]
conv4_block2_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block2_1_conv[0…
conv4_block2_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block2_1_bn[0][…
conv4_block2_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block2_1_relu[0…
conv4_block2_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block2_2_conv[0…
conv4_block2_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block2_2_bn[0][…
conv4_block2_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block2_2_relu[0…
conv4_block2_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block2_3_conv[0…
conv4_block2_add (Add)	(None, 12, 12, 1024)	0	conv4_block1_out[0][0… conv4_block2_3_bn[0][…
conv4_block2_out (Activation)	(None, 12, 12, 1024)	0	conv4_block2_add[0][0]
conv4_block3_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block2_out[0][0]
conv4_block3_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block3_1_conv[0…
conv4_block3_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block3_1_bn[0][…
conv4_block3_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block3_1_relu[0…
conv4_block3_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block3_2_conv[0…
conv4_block3_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block3_2_bn[0][…
conv4_block3_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block3_2_relu[0…
conv4_block3_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block3_3_conv[0…
conv4_block3_add (Add)	(None, 12, 12, 1024)	0	conv4_block2_out[0][0… conv4_block3_3_bn[0][…
conv4_block3_out (Activation)	(None, 12, 12, 1024)	0	conv4_block3_add[0][0]
conv4_block4_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block3_out[0][0]
conv4_block4_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block4_1_conv[0…
conv4_block4_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block4_1_bn[0][…
conv4_block4_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block4_1_relu[0…

conv4_block4_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block4_2_conv[0…
conv4_block4_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block4_2_bn[0][…
conv4_block4_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block4_2_relu[0…
conv4_block4_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block4_3_conv[0…
conv4_block4_add (Add)	(None, 12, 12, 1024)	0	conv4_block3_out[0][0… conv4_block4_3_bn[0][…
conv4_block4_out (Activation)	(None, 12, 12, 1024)	0	conv4_block4_add[0][0]
conv4_block5_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block4_out[0][0]
conv4_block5_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block5_1_conv[0…
conv4_block5_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block5_1_bn[0][…
conv4_block5_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block5_1_relu[0…
conv4_block5_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block5_2_conv[0…
conv4_block5_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block5_2_bn[0][…
conv4_block5_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block5_2_relu[0…
conv4_block5_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block5_3_conv[0…
conv4_block5_add (Add)	(None, 12, 12, 1024)	0	conv4_block4_out[0][0… conv4_block5_3_bn[0][…
conv4_block5_out (Activation)	(None, 12, 12, 1024)	0	conv4_block5_add[0][0]
conv4_block6_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block5_out[0][0]
conv4_block6_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block6_1_conv[0…
conv4_block6_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block6_1_bn[0][…
conv4_block6_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block6_1_relu[0…
conv4_block6_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block6_2_conv[0…
conv4_block6_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block6_2_bn[0][…
conv4_block6_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block6_2_relu[0…
conv4_block6_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block6_3_conv[0…
conv4_block6_add (Add)	(None, 12, 12, 1024)	0	conv4_block5_out[0][0… conv4_block6_3_bn[0][…
conv4_block6_out (Activation)	(None, 12, 12, 1024)	0	conv4_block6_add[0][0]
conv5_block1_1_conv (Conv2D)	(None, 6, 6, 512)	524,800	conv4_block6_out[0][0]
conv5_block1_1_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block1_1_conv[0…
conv5_block1_1_relu (Activation)	(None, 6, 6, 512)	0	conv5_block1_1_bn[0][…

conv5_block1_2_conv (Conv2D)	(None, 6, 6, 512)	2,359,808	conv5_block1_1_relu[0…
conv5_block1_2_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block1_2_conv[0…
conv5_block1_2_relu (Activation)	(None, 6, 6, 512)	0	conv5_block1_2_bn[0][…
conv5_block1_0_conv (Conv2D)	(None, 6, 6, 2048)	2,099,200	conv4_block6_out[0][0]
conv5_block1_3_conv (Conv2D)	(None, 6, 6, 2048)	1,050,624	conv5_block1_2_relu[0…
conv5_block1_0_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block1_0_conv[0…
conv5_block1_3_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block1_3_conv[0…
conv5_block1_add (Add)	(None, 6, 6, 2048)	0	conv5_block1_0_bn[0][… conv5_block1_3_bn[0][…
conv5_block1_out (Activation)	(None, 6, 6, 2048)	0	conv5_block1_add[0][0]
conv5_block2_1_conv (Conv2D)	(None, 6, 6, 512)	1,049,088	conv5_block1_out[0][0]
conv5_block2_1_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block2_1_conv[0…
conv5_block2_1_relu (Activation)	(None, 6, 6, 512)	0	conv5_block2_1_bn[0][…
conv5_block2_2_conv (Conv2D)	(None, 6, 6, 512)	2,359,808	conv5_block2_1_relu[0…
conv5_block2_2_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block2_2_conv[0…
conv5_block2_2_relu (Activation)	(None, 6, 6, 512)	0	conv5_block2_2_bn[0][…
conv5_block2_3_conv (Conv2D)	(None, 6, 6, 2048)	1,050,624	conv5_block2_2_relu[0…
conv5_block2_3_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block2_3_conv[0…
conv5_block2_add (Add)	(None, 6, 6, 2048)	0	conv5_block1_out[0][0… conv5_block2_3_bn[0][…
conv5_block2_out (Activation)	(None, 6, 6, 2048)	0	conv5_block2_add[0][0]
conv5_block3_1_conv (Conv2D)	(None, 6, 6, 512)	1,049,088	conv5_block2_out[0][0]
conv5_block3_1_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block3_1_conv[0…
conv5_block3_1_relu (Activation)	(None, 6, 6, 512)	0	conv5_block3_1_bn[0][…
conv5_block3_2_conv (Conv2D)	(None, 6, 6, 512)	2,359,808	conv5_block3_1_relu[0…
conv5_block3_2_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block3_2_conv[0…
conv5_block3_2_relu (Activation)	(None, 6, 6, 512)	0	conv5_block3_2_bn[0][…
conv5_block3_3_conv (Conv2D)	(None, 6, 6, 2048)	1,050,624	conv5_block3_2_relu[0…
conv5_block3_3_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block3_3_conv[0…
conv5_block3_add (Add)	(None, 6, 6, 2048)	0	conv5_block2_out[0][0… conv5_block3_3_bn[0][…
conv5_block3_out (Activation)	(None, 6, 6, 2048)	0	conv5_block3_add[0][0]

global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	conv5_block3_out[0][0]
dense (Dense)	(None, 3)	6,147	global_average_poolin…

Total params: 23,593,859 (90.00 MB)

Trainable params: 6,147 (24.01 KB)

Non-trainable params: 23,587,712 (89.98 MB)

In [6]:
```
from tensorflow.keras.callbacks import EarlyStopping
# Prepare data using ImageDataGenerator
train_datagen = ImageDataGenerator(validation_split=0.2)
train_generator = train_datagen.flow(X_train, y_train, subset='training', batch_size=32)
validation_generator = train_datagen.flow(X_train, y_train, subset='validation', batch_size=32)

early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)

# Train the model
history_res = model_res.fit(train_generator,
                            validation_data=validation_generator,
                            epochs=100,
                            callbacks=[early_stopping])  # Reduced epochs due to pre-trained weights

# Evaluate the model on the test set
test_loss_res, test_accuracy_res = model_res.evaluate(X_test, y_test)
print(f"ResNet Test accuracy: {test_accuracy_res*100:.2f}%, Test loss: {test_loss_res}")

# Optionally, visualize the training process of ResNet
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_res.history['accuracy'], label='ResNet Training accuracy')
plt.plot(history_res.history['val_accuracy'], label='ResNet Validation accuracy')
plt.title('ResNet Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_res.history['loss'], label='ResNet Training loss')
plt.plot(history_res.history['val_loss'], label='ResNet Validation loss')
plt.title('ResNet Training and Validation Loss')
plt.legend()
plt.show()
```

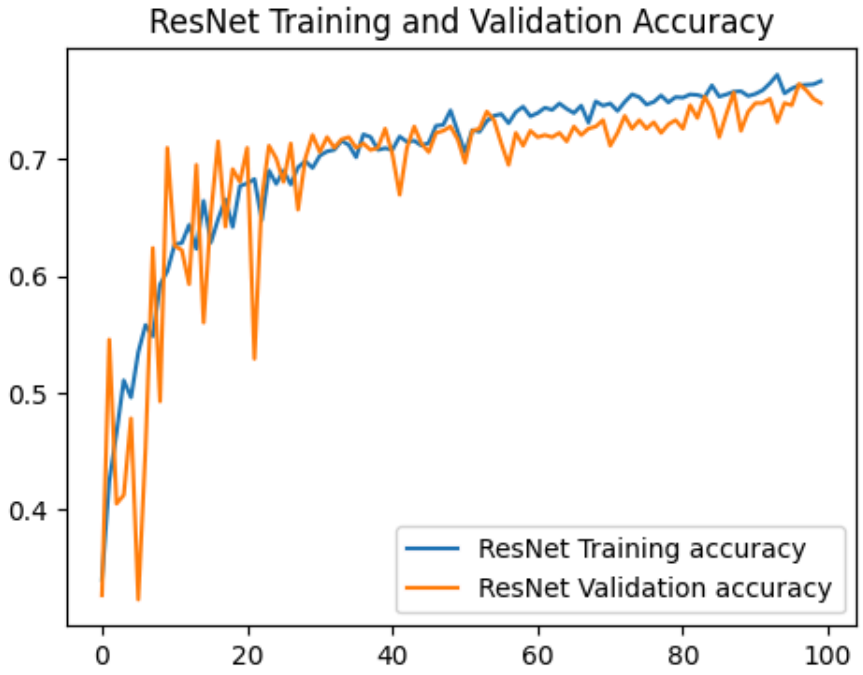
```
Epoch 1/100
```

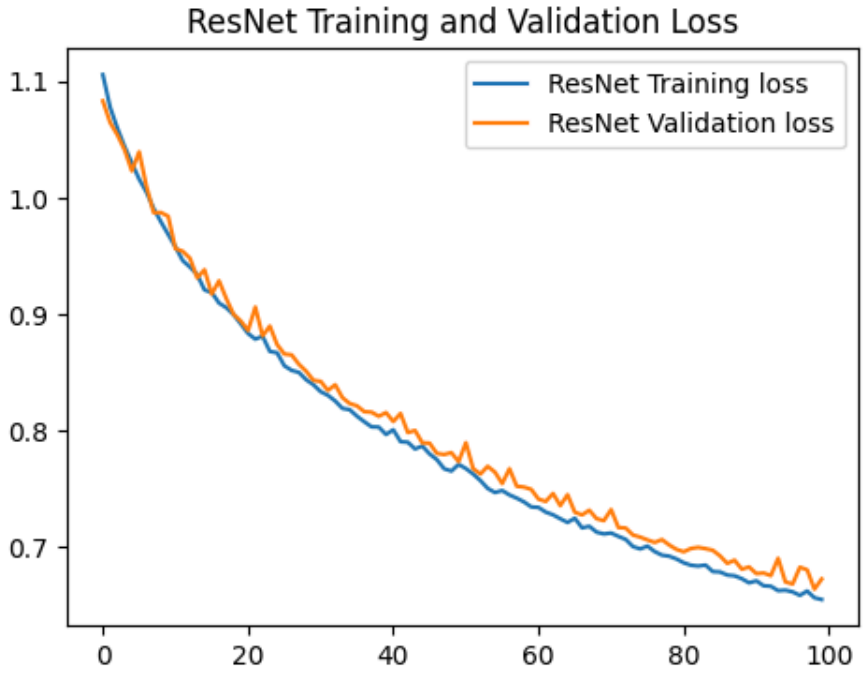
```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 26s 210ms/step - accuracy: 0.3317 - loss: 1.1134 - val_accuracy: 0.3266 - val_loss: 1.0833
Epoch 2/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 71ms/step - accuracy: 0.4093 - loss: 1.0819 - val_accuracy: 0.5456 - val_loss: 1.0648
Epoch 3/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 72ms/step - accuracy: 0.4726 - loss: 1.0632 - val_accuracy: 0.4051 - val_loss: 1.0541
Epoch 4/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 72ms/step - accuracy: 0.4624 - loss: 1.0509 - val_accuracy: 0.4124 - val_loss: 1.0412
Epoch 5/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.4665 - loss: 1.0328 - val_accuracy: 0.4781 - val_loss: 1.0231
Epoch 6/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.5511 - loss: 1.0136 - val_accuracy: 0.3230 - val_loss: 1.0396
Epoch 7/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.4961 - loss: 1.0206 - val_accuracy: 0.4526 - val_loss: 1.0101
Epoch 8/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.5363 - loss: 0.9929 - val_accuracy: 0.6241 - val_loss: 0.9873
Epoch 9/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6076 - loss: 0.9805 - val_accuracy: 0.4927 - val_loss: 0.9874
Epoch 10/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.5890 - loss: 0.9670 - val_accuracy: 0.7099 - val_loss: 0.9843
Epoch 11/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.6406 - loss: 0.9648 - val_accuracy: 0.6259 - val_loss: 0.9564
Epoch 12/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.6293 - loss: 0.9476 - val_accuracy: 0.6223 - val_loss: 0.9542
Epoch 13/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 76ms/step - accuracy: 0.6050 - loss: 0.9532 - val_accuracy: 0.5931 - val_loss: 0.9481
Epoch 14/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.6044 - loss: 0.9362 - val_accuracy: 0.6953 - val_loss: 0.9313
Epoch 15/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 76ms/step - accuracy: 0.6627 - loss: 0.9243 - val_accuracy: 0.5602 - val_loss: 0.9383
Epoch 16/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6111 - loss: 0.9262 - val_accuracy: 0.6533 - val_loss: 0.9174
Epoch 17/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.6170 - loss: 0.9123 - val_accuracy: 0.7153 - val_loss: 0.9288
Epoch 18/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6578 - loss: 0.9142 - val_accuracy: 0.6423 - val_loss: 0.9135
Epoch 19/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.6719 - loss: 0.8941 - val_accuracy: 0.6916 - val_loss: 0.9000
Epoch 20/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6705 - loss: 0.8919 - val_accuracy: 0.6807 - val_loss: 0.8942
Epoch 21/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6870 - loss: 0.8798 - val_accuracy: 0.7099 - val_loss: 0.8861
Epoch 22/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.6922 - loss: 0.8898 - val_accuracy: 0.5292 - val_loss: 0.9062
Epoch 23/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6171 - loss: 0.8934 - val_accuracy: 0.6661 - val_loss: 0.8815
Epoch 24/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6986 - loss: 0.8597 - val_accuracy: 0.7117 - val_loss: 0.8899
Epoch 25/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6915 - loss: 0.8571 - val_accuracy: 0.7007 - val_loss: 0.8740
Epoch 26/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6933 - loss: 0.8600 - val_accuracy: 0.6807 - val_loss: 0.8659
Epoch 27/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.6755 - loss: 0.8518 - val_accuracy: 0.7135 - val_loss: 0.8652
Epoch 28/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6853 - loss: 0.8621 - val_accuracy: 0.6569 - val_loss: 0.8571
Epoch 29/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.6798 - loss: 0.8496 - val_accuracy: 0.7007 - val_loss: 0.8512
Epoch 30/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7155 - loss: 0.8293 - val_accuracy: 0.7208 - val_loss: 0.8433
Epoch 31/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7058 - loss: 0.8370 - val_accuracy: 0.7062 - val_loss: 0.8424
Epoch 32/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7101 - loss: 0.8309 - val_accuracy: 0.7190 - val_loss: 0.8346
Epoch 33/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7148 - loss: 0.8261 - val_accuracy: 0.7099 - val_loss: 0.8395
Epoch 34/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7118 - loss: 0.8232 - val_accuracy: 0.7172 - val_loss: 0.8284
Epoch 35/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7121 - loss: 0.8242 - val_accuracy: 0.7190 - val_loss: 0.8234
Epoch 36/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7030 - loss: 0.8126 - val_accuracy: 0.7099 - val_loss: 0.8214
Epoch 37/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7325 - loss: 0.8036 - val_accuracy: 0.7135 - val_loss: 0.8164
Epoch 38/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7121 - loss: 0.8114 - val_accuracy: 0.7080 - val_loss: 0.8160
Epoch 39/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7248 - loss: 0.7967 - val_accuracy: 0.7099 - val_loss: 0.8124
Epoch 40/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7241 - loss: 0.7864 - val_accuracy: 0.7263 - val_loss: 0.8155
Epoch 41/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7136 - loss: 0.7881 - val_accuracy: 0.7044 - val_loss: 0.8080
Epoch 42/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7138 - loss: 0.7853 - val_accuracy: 0.6697 - val_loss: 0.8148
Epoch 43/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.6977 - loss: 0.7942 - val_accuracy: 0.7099 - val_loss: 0.7984
Epoch 44/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7167 - loss: 0.7850 - val_accuracy: 0.7281 - val_loss: 0.8002
```

```
Epoch 45/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7189 - loss: 0.7889 - val_accuracy: 0.7135 - val_loss: 0.7896
Epoch 46/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7147 - loss: 0.7814 - val_accuracy: 0.7062 - val_loss: 0.7893
Epoch 47/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7151 - loss: 0.7838 - val_accuracy: 0.7226 - val_loss: 0.7805
Epoch 48/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7333 - loss: 0.7609 - val_accuracy: 0.7245 - val_loss: 0.7796
Epoch 49/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7336 - loss: 0.7794 - val_accuracy: 0.7281 - val_loss: 0.7812
Epoch 50/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7152 - loss: 0.7755 - val_accuracy: 0.7172 - val_loss: 0.7735
Epoch 51/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7164 - loss: 0.7593 - val_accuracy: 0.6971 - val_loss: 0.7897
Epoch 52/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7073 - loss: 0.7842 - val_accuracy: 0.7245 - val_loss: 0.7674
Epoch 53/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7194 - loss: 0.7625 - val_accuracy: 0.7263 - val_loss: 0.7628
Epoch 54/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7314 - loss: 0.7464 - val_accuracy: 0.7409 - val_loss: 0.7694
Epoch 55/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7388 - loss: 0.7470 - val_accuracy: 0.7336 - val_loss: 0.7644
Epoch 56/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7394 - loss: 0.7436 - val_accuracy: 0.7135 - val_loss: 0.7546
Epoch 57/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7410 - loss: 0.7374 - val_accuracy: 0.6953 - val_loss: 0.7674
Epoch 58/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7379 - loss: 0.7513 - val_accuracy: 0.7226 - val_loss: 0.7524
Epoch 59/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7372 - loss: 0.7588 - val_accuracy: 0.7117 - val_loss: 0.7516
Epoch 60/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7412 - loss: 0.7281 - val_accuracy: 0.7245 - val_loss: 0.7496
Epoch 61/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7437 - loss: 0.7290 - val_accuracy: 0.7190 - val_loss: 0.7413
Epoch 62/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.7485 - loss: 0.7242 - val_accuracy: 0.7208 - val_loss: 0.7393
Epoch 63/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7417 - loss: 0.7228 - val_accuracy: 0.7190 - val_loss: 0.7461
Epoch 64/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7413 - loss: 0.7293 - val_accuracy: 0.7226 - val_loss: 0.7357
Epoch 65/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7342 - loss: 0.7304 - val_accuracy: 0.7153 - val_loss: 0.7450
Epoch 66/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7270 - loss: 0.7368 - val_accuracy: 0.7281 - val_loss: 0.7300
Epoch 67/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7367 - loss: 0.7231 - val_accuracy: 0.7208 - val_loss: 0.7277
Epoch 68/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7300 - loss: 0.7148 - val_accuracy: 0.7263 - val_loss: 0.7317
Epoch 69/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7586 - loss: 0.7063 - val_accuracy: 0.7281 - val_loss: 0.7244
Epoch 70/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7497 - loss: 0.7062 - val_accuracy: 0.7336 - val_loss: 0.7228
Epoch 71/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7550 - loss: 0.7126 - val_accuracy: 0.7117 - val_loss: 0.7323
Epoch 72/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7637 - loss: 0.6882 - val_accuracy: 0.7226 - val_loss: 0.7169
Epoch 73/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7402 - loss: 0.7144 - val_accuracy: 0.7372 - val_loss: 0.7164
Epoch 74/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7302 - loss: 0.7133 - val_accuracy: 0.7263 - val_loss: 0.7105
Epoch 75/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7673 - loss: 0.6880 - val_accuracy: 0.7336 - val_loss: 0.7086
Epoch 76/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7469 - loss: 0.7007 - val_accuracy: 0.7263 - val_loss: 0.7062
Epoch 77/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7552 - loss: 0.6859 - val_accuracy: 0.7318 - val_loss: 0.7041
Epoch 78/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7536 - loss: 0.6997 - val_accuracy: 0.7226 - val_loss: 0.7065
Epoch 79/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7524 - loss: 0.6919 - val_accuracy: 0.7299 - val_loss: 0.7019
Epoch 80/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7618 - loss: 0.6789 - val_accuracy: 0.7336 - val_loss: 0.6981
Epoch 81/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7461 - loss: 0.6880 - val_accuracy: 0.7263 - val_loss: 0.6960
Epoch 82/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7711 - loss: 0.6753 - val_accuracy: 0.7464 - val_loss: 0.6989
Epoch 83/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7567 - loss: 0.6896 - val_accuracy: 0.7354 - val_loss: 0.6998
Epoch 84/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7636 - loss: 0.6775 - val_accuracy: 0.7536 - val_loss: 0.6987
Epoch 85/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7463 - loss: 0.6902 - val_accuracy: 0.7427 - val_loss: 0.6973
Epoch 86/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7622 - loss: 0.6692 - val_accuracy: 0.7190 - val_loss: 0.6925
Epoch 87/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7741 - loss: 0.6624 - val_accuracy: 0.7391 - val_loss: 0.6861
Epoch 88/100
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7578 - loss: 0.6836 - val_accuracy: 0.7573 - val_loss: 0.6888
Epoch 89/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7587 - loss: 0.6660 - val_accuracy: 0.7245 - val_loss: 0.6811
Epoch 90/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7522 - loss: 0.6640 - val_accuracy: 0.7409 - val_loss: 0.6830
Epoch 91/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7479 - loss: 0.6678 - val_accuracy: 0.7482 - val_loss: 0.6773
Epoch 92/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7646 - loss: 0.6524 - val_accuracy: 0.7482 - val_loss: 0.6778
Epoch 93/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7823 - loss: 0.6610 - val_accuracy: 0.7518 - val_loss: 0.6756
Epoch 94/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7742 - loss: 0.6687 - val_accuracy: 0.7318 - val_loss: 0.6904
Epoch 95/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7465 - loss: 0.6802 - val_accuracy: 0.7482 - val_loss: 0.6701
Epoch 96/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7529 - loss: 0.6663 - val_accuracy: 0.7464 - val_loss: 0.6684
Epoch 97/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7648 - loss: 0.6608 - val_accuracy: 0.7646 - val_loss: 0.6828
Epoch 98/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7875 - loss: 0.6439 - val_accuracy: 0.7591 - val_loss: 0.6805
Epoch 99/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7674 - loss: 0.6516 - val_accuracy: 0.7518 - val_loss: 0.6639
Epoch 100/100
69/69 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7693 - loss: 0.6527 - val_accuracy: 0.7482 - val_loss: 0.6727
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 124ms/step - accuracy: 0.7825 - loss: 0.6525
ResNet Test accuracy: 75.99%, Test loss: 0.6647061109542847
```





In [7]:
```python
with open('model_res.pkl', 'wb') as file:
    pickle.dump(model_res, file)
with open('history_res.pkl', 'wb') as file:
    pickle.dump(history_res, file)
```

Discussion on Lower Test Accuracy of ResNet Compared to Baseline CNN

Several factors might contribute to the observed lower test accuracy of the ResNet model when compared to the baseline CNN. Here are potential explanations:

- **Mismatch Between Pre-trained Features and New Data**: The features learned by ResNet50 on the ImageNet dataset might not transfer well to the specific task at hand. ImageNet contains a broad range of images that might differ significantly from the specific types of images (e.g., medical images) in the training dataset, leading to a mismatch in feature relevance.
- **Model Complexity and Dataset Specificity**: The baseline CNN, being less complex, might be better suited for the dataset if the distinguishing features between classes are relatively simple. In such cases, simpler models are less likely to overlearn the noise in the training data and thus generalize better on the test data.
- **Batch Size and Training Dynamics**: The chosen batch size and other training parameters might also influence how well the ResNet model learns. Smaller or larger batch sizes can affect the stability of the training process, impacting the final model performance on the test set.

These factors suggest that adjustments in the model architecture, training process, or data handling might be needed to enhance the ResNet model's performance and ensure it outperforms or matches the baseline CNN in future experiments.

4 Additional Architectures

4.1 Retarined ResNet

Fully Retrained ResNet Model Architecture and Training with Early Stopping

Model Architecture

The fully retrained ResNet model (`model_res_retrained`) is based on the robust architecture of ResNet50, but without utilizing pre-trained weights. This approach is designed to tailor the learning specifically to our dataset from the ground up:

- **Base Model**: ResNet50 architecture is employed but initialized without weights from ImageNet, allowing the model to learn features purely relevant to our specific classification task.
- **Global Average Pooling**: Following the ResNet base layers, a `GlobalAveragePooling2D` layer condenses each feature map to a single value, which helps in minimizing the model's complexity and reducing the risk of overfitting.
- **Output Layer**: The model concludes with a `Dense` layer comprising 3 units with a softmax activation function, designed to classify the images into three distinct categories based on the learned features.

Loss Function and Optimizer

- **Loss Function**: `CategoricalCrossentropy` is utilized, suitable for our one-hot encoded labels in a multi-class classification scenario.
- **Optimizer**: The `Adam` optimizer is chosen for its efficient and adaptive learning rate capabilities, which are essential for converging to optimal weights, especially when training from scratch.

Evaluation Metrics and Training Configuration

- **Evaluation Metric**: Accuracy is used as the primary metric to assess the effectiveness of the model in correctly classifying the images.
- **Training Configuration**: The model is configured to train potentially up to 50 epochs with a batch size of 32, but includes an early stopping mechanism. Early stopping monitors the validation loss and halts training if there is no improvement for three consecutive epochs, helping prevent overtraining and ensuring that the model generalizes well to unseen data.

Performance Visualization

- **Plots**: We provide visualizations of the training and validation accuracy and loss. These plots are crucial in observing the training dynamics and evaluating the effectiveness of the early stopping in preventing overfitting while maximizing performance.

This configuration ensures that the `model_res_retrained` is optimally trained to adapt to our specific dataset without the influence of pre-trained weights, potentially enhancing its ability to generalize well to new data while maintaining a robust learning process.

In [10]:
```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Load the ResNet50 model without pre-trained weights
base_model = ResNet50(weights=None, include_top=False, input_shape=(192, 192, 3))

# Set all layers to be trainable
for layer in base_model.layers:
    layer.trainable = True

# Add custom layers on top of ResNet
x = GlobalAveragePooling2D()(base_model.output)
predictions = Dense(3, activation='softmax')(x)  # Assuming 3 classes

# Create the final model and rename to model_res_retrained
model_res_retrained = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model_res_retrained.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Model summary
model_res_retrained.summary()

# Prepare data using ImageDataGenerator
train_datagen = ImageDataGenerator(validation_split=0.2)
train_generator = train_datagen.flow(X_train, y_train, subset='training', batch_size=32)
```

```
validation_generator = train_datagen.flow(X_train, y_train, subset='validation', batch_size=32)

# Setup early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=20, verbose=1, restore_best_weights=True)

# Train the model
history_res_retrained = model_res_retrained.fit(train_generator,
                          validation_data=validation_generator,
                          epochs=150,  # Set a higher potential max epochs
                          callbacks=[early_stopping])

# Evaluate the model on the test set
test_loss_res_retrained, test_accuracy_res_retrained = model_res_retrained.evaluate(X_test, y_test)
print(f"Retrained ResNet Test accuracy: {test_accuracy_res_retrained*100:.2f}%, Test loss: {test_loss_res_retrained}")

# Optionally, visualize the training process
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_res_retrained.history['accuracy'], label='Training accuracy')
plt.plot(history_res_retrained.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_res_retrained.history['loss'], label='Training loss')
plt.plot(history_res_retrained.history['val_loss'], label='Validation loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

Model: "functional_3"

Layer (type)	Output Shape	Param #	Connected to
input_layer_3 (InputLayer)	(None, 192, 192, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 198, 198, 3)	0	input_layer_3[0][0]
conv1_conv (Conv2D)	(None, 96, 96, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalization)	(None, 96, 96, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 96, 96, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 98, 98, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 48, 48, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 48, 48, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block1_1_conv[0…
conv2_block1_1_relu (Activation)	(None, 48, 48, 64)	0	conv2_block1_1_bn[0][…
conv2_block1_2_conv (Conv2D)	(None, 48, 48, 64)	36,928	conv2_block1_1_relu[0…
conv2_block1_2_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block1_2_conv[0…
conv2_block1_2_relu (Activation)	(None, 48, 48, 64)	0	conv2_block1_2_bn[0][…
conv2_block1_0_conv (Conv2D)	(None, 48, 48, 256)	16,640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 48, 48, 256)	16,640	conv2_block1_2_relu[0…
conv2_block1_0_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block1_0_conv[0…
conv2_block1_3_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block1_3_conv[0…
conv2_block1_add (Add)	(None, 48, 48, 256)	0	conv2_block1_0_bn[0][… conv2_block1_3_bn[0][…
conv2_block1_out (Activation)	(None, 48, 48, 256)	0	conv2_block1_add[0][0]
conv2_block2_1_conv (Conv2D)	(None, 48, 48, 64)	16,448	conv2_block1_out[0][0]
conv2_block2_1_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block2_1_conv[0…
conv2_block2_1_relu (Activation)	(None, 48, 48, 64)	0	conv2_block2_1_bn[0][…
conv2_block2_2_conv (Conv2D)	(None, 48, 48, 64)	36,928	conv2_block2_1_relu[0…
conv2_block2_2_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block2_2_conv[0…
conv2_block2_2_relu (Activation)	(None, 48, 48, 64)	0	conv2_block2_2_bn[0][…
conv2_block2_3_conv (Conv2D)	(None, 48, 48, 256)	16,640	conv2_block2_2_relu[0…
conv2_block2_3_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block2_3_conv[0…
conv2_block2_add (Add)	(None, 48, 48, 256)	0	conv2_block1_out[0][0… conv2_block2_3_bn[0][…
conv2_block2_out (Activation)	(None, 48, 48, 256)	0	conv2_block2_add[0][0]
conv2_block3_1_conv (Conv2D)	(None, 48, 48, 64)	16,448	conv2_block2_out[0][0]

conv2_block3_1_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block3_1_conv[0…
conv2_block3_1_relu (Activation)	(None, 48, 48, 64)	0	conv2_block3_1_bn[0][…
conv2_block3_2_conv (Conv2D)	(None, 48, 48, 64)	36,928	conv2_block3_1_relu[0…
conv2_block3_2_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block3_2_conv[0…
conv2_block3_2_relu (Activation)	(None, 48, 48, 64)	0	conv2_block3_2_bn[0][…
conv2_block3_3_conv (Conv2D)	(None, 48, 48, 256)	16,640	conv2_block3_2_relu[0…
conv2_block3_3_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block3_3_conv[0…
conv2_block3_add (Add)	(None, 48, 48, 256)	0	conv2_block2_out[0][0… conv2_block3_3_bn[0][…
conv2_block3_out (Activation)	(None, 48, 48, 256)	0	conv2_block3_add[0][0]
conv3_block1_1_conv (Conv2D)	(None, 24, 24, 128)	32,896	conv2_block3_out[0][0]
conv3_block1_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block1_1_conv[0…
conv3_block1_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block1_1_bn[0][…
conv3_block1_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block1_1_relu[0…
conv3_block1_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block1_2_conv[0…
conv3_block1_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block1_2_bn[0][…
conv3_block1_0_conv (Conv2D)	(None, 24, 24, 512)	131,584	conv2_block3_out[0][0]
conv3_block1_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block1_2_relu[0…
conv3_block1_0_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block1_0_conv[0…
conv3_block1_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block1_3_conv[0…
conv3_block1_add (Add)	(None, 24, 24, 512)	0	conv3_block1_0_bn[0][… conv3_block1_3_bn[0][…
conv3_block1_out (Activation)	(None, 24, 24, 512)	0	conv3_block1_add[0][0]
conv3_block2_1_conv (Conv2D)	(None, 24, 24, 128)	65,664	conv3_block1_out[0][0]
conv3_block2_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block2_1_conv[0…
conv3_block2_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block2_1_bn[0][…
conv3_block2_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block2_1_relu[0…
conv3_block2_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block2_2_conv[0…
conv3_block2_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block2_2_bn[0][…
conv3_block2_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block2_2_relu[0…
conv3_block2_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block2_3_conv[0…

conv3_block2_add (Add)	(None, 24, 24, 512)	0	conv3_block1_out[0][0… conv3_block2_3_bn[0][…
conv3_block2_out (Activation)	(None, 24, 24, 512)	0	conv3_block2_add[0][0]
conv3_block3_1_conv (Conv2D)	(None, 24, 24, 128)	65,664	conv3_block2_out[0][0]
conv3_block3_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block3_1_conv[0…
conv3_block3_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block3_1_bn[0][…
conv3_block3_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block3_1_relu[0…
conv3_block3_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block3_2_conv[0…
conv3_block3_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block3_2_bn[0][…
conv3_block3_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block3_2_relu[0…
conv3_block3_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block3_3_conv[0…
conv3_block3_add (Add)	(None, 24, 24, 512)	0	conv3_block2_out[0][0… conv3_block3_3_bn[0][…
conv3_block3_out (Activation)	(None, 24, 24, 512)	0	conv3_block3_add[0][0]
conv3_block4_1_conv (Conv2D)	(None, 24, 24, 128)	65,664	conv3_block3_out[0][0]
conv3_block4_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block4_1_conv[0…
conv3_block4_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block4_1_bn[0][…
conv3_block4_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block4_1_relu[0…
conv3_block4_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block4_2_conv[0…
conv3_block4_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block4_2_bn[0][…
conv3_block4_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block4_2_relu[0…
conv3_block4_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block4_3_conv[0…
conv3_block4_add (Add)	(None, 24, 24, 512)	0	conv3_block3_out[0][0… conv3_block4_3_bn[0][…
conv3_block4_out (Activation)	(None, 24, 24, 512)	0	conv3_block4_add[0][0]
conv4_block1_1_conv (Conv2D)	(None, 12, 12, 256)	131,328	conv3_block4_out[0][0]
conv4_block1_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block1_1_conv[0…
conv4_block1_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block1_1_bn[0][…
conv4_block1_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block1_1_relu[0…
conv4_block1_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block1_2_conv[0…
conv4_block1_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block1_2_bn[0][…
conv4_block1_0_conv (Conv2D)	(None, 12, 12, 1024)	525,312	conv3_block4_out[0][0]

conv4_block1_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block1_2_relu[0…
conv4_block1_0_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block1_0_conv[0…
conv4_block1_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block1_3_conv[0…
conv4_block1_add (Add)	(None, 12, 12, 1024)	0	conv4_block1_0_bn[0][… conv4_block1_3_bn[0][…
conv4_block1_out (Activation)	(None, 12, 12, 1024)	0	conv4_block1_add[0][0]
conv4_block2_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block1_out[0][0]
conv4_block2_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block2_1_conv[0…
conv4_block2_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block2_1_bn[0][…
conv4_block2_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block2_1_relu[0…
conv4_block2_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block2_2_conv[0…
conv4_block2_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block2_2_bn[0][…
conv4_block2_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block2_2_relu[0…
conv4_block2_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block2_3_conv[0…
conv4_block2_add (Add)	(None, 12, 12, 1024)	0	conv4_block1_out[0][0… conv4_block2_3_bn[0][…
conv4_block2_out (Activation)	(None, 12, 12, 1024)	0	conv4_block2_add[0][0]
conv4_block3_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block2_out[0][0]
conv4_block3_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block3_1_conv[0…
conv4_block3_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block3_1_bn[0][…
conv4_block3_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block3_1_relu[0…
conv4_block3_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block3_2_conv[0…
conv4_block3_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block3_2_bn[0][…
conv4_block3_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block3_2_relu[0…
conv4_block3_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block3_3_conv[0…
conv4_block3_add (Add)	(None, 12, 12, 1024)	0	conv4_block2_out[0][0… conv4_block3_3_bn[0][…
conv4_block3_out (Activation)	(None, 12, 12, 1024)	0	conv4_block3_add[0][0]
conv4_block4_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block3_out[0][0]
conv4_block4_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block4_1_conv[0…
conv4_block4_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block4_1_bn[0][…
conv4_block4_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block4_1_relu[0…

conv4_block4_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block4_2_conv[0…
conv4_block4_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block4_2_bn[0][…
conv4_block4_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block4_2_relu[0…
conv4_block4_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block4_3_conv[0…
conv4_block4_add (Add)	(None, 12, 12, 1024)	0	conv4_block3_out[0][0… conv4_block4_3_bn[0][…
conv4_block4_out (Activation)	(None, 12, 12, 1024)	0	conv4_block4_add[0][0]
conv4_block5_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block4_out[0][0]
conv4_block5_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block5_1_conv[0…
conv4_block5_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block5_1_bn[0][…
conv4_block5_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block5_1_relu[0…
conv4_block5_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block5_2_conv[0…
conv4_block5_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block5_2_bn[0][…
conv4_block5_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block5_2_relu[0…
conv4_block5_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block5_3_conv[0…
conv4_block5_add (Add)	(None, 12, 12, 1024)	0	conv4_block4_out[0][0… conv4_block5_3_bn[0][…
conv4_block5_out (Activation)	(None, 12, 12, 1024)	0	conv4_block5_add[0][0]
conv4_block6_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block5_out[0][0]
conv4_block6_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block6_1_conv[0…
conv4_block6_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block6_1_bn[0][…
conv4_block6_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block6_1_relu[0…
conv4_block6_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block6_2_conv[0…
conv4_block6_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block6_2_bn[0][…
conv4_block6_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block6_2_relu[0…
conv4_block6_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block6_3_conv[0…
conv4_block6_add (Add)	(None, 12, 12, 1024)	0	conv4_block5_out[0][0… conv4_block6_3_bn[0][…
conv4_block6_out (Activation)	(None, 12, 12, 1024)	0	conv4_block6_add[0][0]
conv5_block1_1_conv (Conv2D)	(None, 6, 6, 512)	524,800	conv4_block6_out[0][0]
conv5_block1_1_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block1_1_conv[0…
conv5_block1_1_relu (Activation)	(None, 6, 6, 512)	0	conv5_block1_1_bn[0][…

conv5_block1_2_conv (Conv2D)	(None, 6, 6, 512)	2,359,808	conv5_block1_1_relu[0…
conv5_block1_2_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block1_2_conv[0…
conv5_block1_2_relu (Activation)	(None, 6, 6, 512)	0	conv5_block1_2_bn[0][…
conv5_block1_0_conv (Conv2D)	(None, 6, 6, 2048)	2,099,200	conv4_block6_out[0][0]
conv5_block1_3_conv (Conv2D)	(None, 6, 6, 2048)	1,050,624	conv5_block1_2_relu[0…
conv5_block1_0_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block1_0_conv[0…
conv5_block1_3_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block1_3_conv[0…
conv5_block1_add (Add)	(None, 6, 6, 2048)	0	conv5_block1_0_bn[0][… conv5_block1_3_bn[0][…
conv5_block1_out (Activation)	(None, 6, 6, 2048)	0	conv5_block1_add[0][0]
conv5_block2_1_conv (Conv2D)	(None, 6, 6, 512)	1,049,088	conv5_block1_out[0][0]
conv5_block2_1_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block2_1_conv[0…
conv5_block2_1_relu (Activation)	(None, 6, 6, 512)	0	conv5_block2_1_bn[0][…
conv5_block2_2_conv (Conv2D)	(None, 6, 6, 512)	2,359,808	conv5_block2_1_relu[0…
conv5_block2_2_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block2_2_conv[0…
conv5_block2_2_relu (Activation)	(None, 6, 6, 512)	0	conv5_block2_2_bn[0][…
conv5_block2_3_conv (Conv2D)	(None, 6, 6, 2048)	1,050,624	conv5_block2_2_relu[0…
conv5_block2_3_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block2_3_conv[0…
conv5_block2_add (Add)	(None, 6, 6, 2048)	0	conv5_block1_out[0][0… conv5_block2_3_bn[0][…
conv5_block2_out (Activation)	(None, 6, 6, 2048)	0	conv5_block2_add[0][0]
conv5_block3_1_conv (Conv2D)	(None, 6, 6, 512)	1,049,088	conv5_block2_out[0][0]
conv5_block3_1_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block3_1_conv[0…
conv5_block3_1_relu (Activation)	(None, 6, 6, 512)	0	conv5_block3_1_bn[0][…
conv5_block3_2_conv (Conv2D)	(None, 6, 6, 512)	2,359,808	conv5_block3_1_relu[0…
conv5_block3_2_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block3_2_conv[0…
conv5_block3_2_relu (Activation)	(None, 6, 6, 512)	0	conv5_block3_2_bn[0][…
conv5_block3_3_conv (Conv2D)	(None, 6, 6, 2048)	1,050,624	conv5_block3_2_relu[0…
conv5_block3_3_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block3_3_conv[0…
conv5_block3_add (Add)	(None, 6, 6, 2048)	0	conv5_block2_out[0][0… conv5_block3_3_bn[0][…
conv5_block3_out (Activation)	(None, 6, 6, 2048)	0	conv5_block3_add[0][0]

global_average_pooling2d… (GlobalAveragePooling2D)	(None, 2048)	0	conv5_block3_out[0][0]
dense_3 (Dense)	(None, 3)	6,147	global_average_poolin…

Total params: 23,593,859 (90.00 MB)

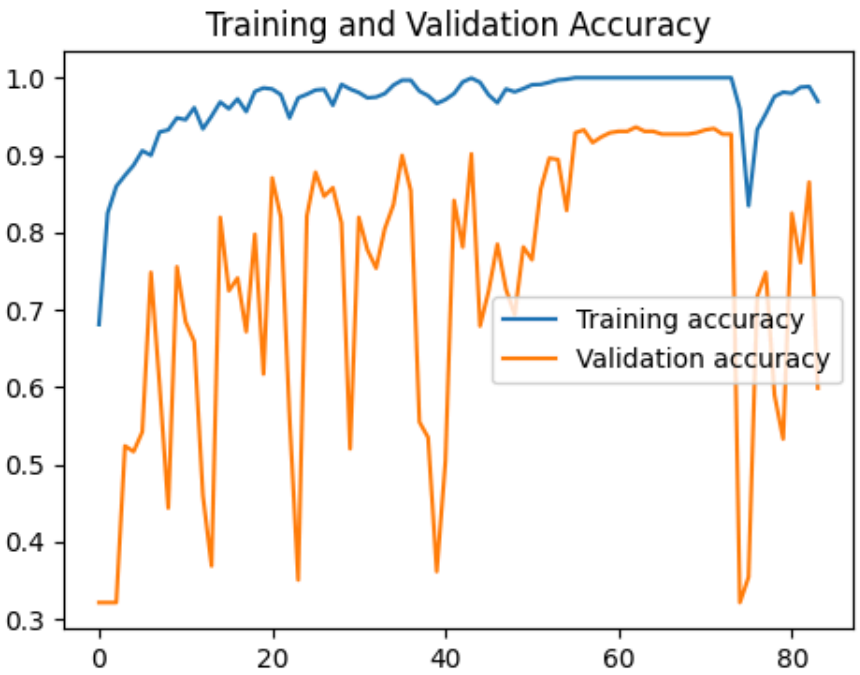
Trainable params: 23,540,739 (89.80 MB)

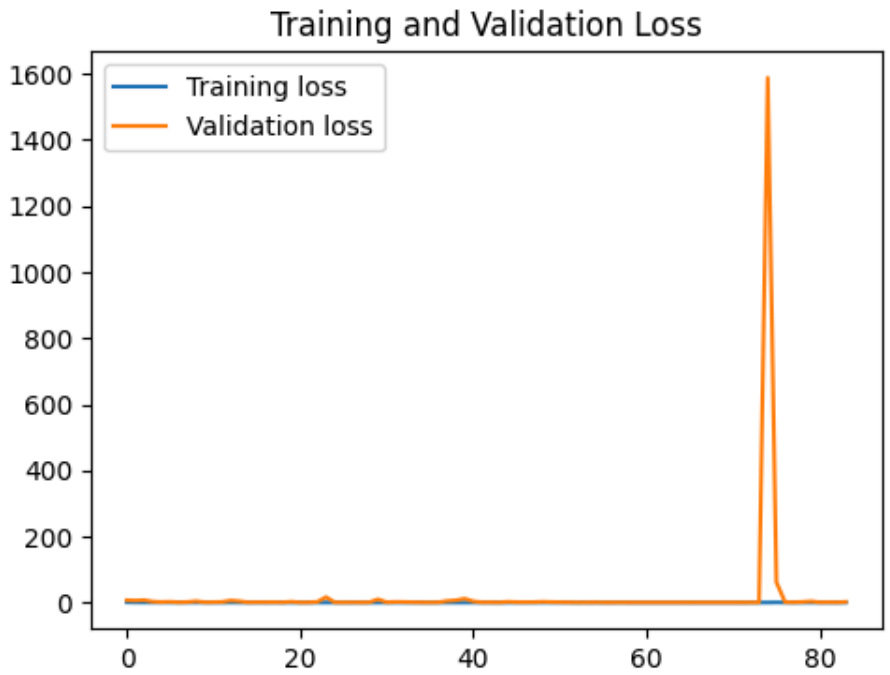
Non-trainable params: 53,120 (207.50 KB)

```
Epoch 1/150
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 83s 535ms/step - accuracy: 0.5885 - loss: 1.5964 - val_accuracy: 0.3212 - val_loss: 7.0555
Epoch 2/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 233ms/step - accuracy: 0.8264 - loss: 0.4585 - val_accuracy: 0.3212 - val_loss: 6.1293
Epoch 3/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.8716 - loss: 0.3522 - val_accuracy: 0.3212 - val_loss: 7.2804
Epoch 4/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 230ms/step - accuracy: 0.8785 - loss: 0.3333 - val_accuracy: 0.5237 - val_loss: 2.7886
Epoch 5/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.8973 - loss: 0.2800 - val_accuracy: 0.5164 - val_loss: 1.4754
Epoch 6/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.8924 - loss: 0.2862 - val_accuracy: 0.5420 - val_loss: 2.3336
Epoch 7/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 231ms/step - accuracy: 0.9024 - loss: 0.2417 - val_accuracy: 0.7482 - val_loss: 0.7072
Epoch 8/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9262 - loss: 0.2035 - val_accuracy: 0.6022 - val_loss: 1.5840
Epoch 9/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 230ms/step - accuracy: 0.9422 - loss: 0.1515 - val_accuracy: 0.4434 - val_loss: 3.9994
Epoch 10/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9511 - loss: 0.1485 - val_accuracy: 0.7555 - val_loss: 0.8036
Epoch 11/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9487 - loss: 0.1497 - val_accuracy: 0.6843 - val_loss: 1.2115
Epoch 12/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9650 - loss: 0.0964 - val_accuracy: 0.6588 - val_loss: 1.8221
Epoch 13/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9300 - loss: 0.1686 - val_accuracy: 0.4599 - val_loss: 6.1436
Epoch 14/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9617 - loss: 0.1159 - val_accuracy: 0.3686 - val_loss: 3.9957
Epoch 15/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 231ms/step - accuracy: 0.9557 - loss: 0.1155 - val_accuracy: 0.8193 - val_loss: 0.6142
Epoch 16/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9635 - loss: 0.0939 - val_accuracy: 0.7245 - val_loss: 1.1257
Epoch 17/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9726 - loss: 0.0743 - val_accuracy: 0.7409 - val_loss: 0.9468
Epoch 18/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9391 - loss: 0.1762 - val_accuracy: 0.6715 - val_loss: 1.1169
Epoch 19/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9847 - loss: 0.0474 - val_accuracy: 0.7974 - val_loss: 0.6868
Epoch 20/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9912 - loss: 0.0319 - val_accuracy: 0.6168 - val_loss: 2.4823
Epoch 21/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 230ms/step - accuracy: 0.9831 - loss: 0.0501 - val_accuracy: 0.8704 - val_loss: 0.4449
Epoch 22/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9852 - loss: 0.0390 - val_accuracy: 0.8193 - val_loss: 0.5911
Epoch 23/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9481 - loss: 0.1679 - val_accuracy: 0.5620 - val_loss: 2.0607
Epoch 24/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9741 - loss: 0.0684 - val_accuracy: 0.3504 - val_loss: 16.3409
Epoch 25/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9744 - loss: 0.0775 - val_accuracy: 0.8212 - val_loss: 0.6510
Epoch 26/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9831 - loss: 0.0450 - val_accuracy: 0.8777 - val_loss: 0.4647
Epoch 27/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9864 - loss: 0.0320 - val_accuracy: 0.8467 - val_loss: 0.6491
Epoch 28/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9658 - loss: 0.0787 - val_accuracy: 0.8577 - val_loss: 0.4866
Epoch 29/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9928 - loss: 0.0282 - val_accuracy: 0.8120 - val_loss: 0.6640
Epoch 30/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9835 - loss: 0.0361 - val_accuracy: 0.5201 - val_loss: 9.7426
Epoch 31/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9800 - loss: 0.0490 - val_accuracy: 0.8193 - val_loss: 0.8904
Epoch 32/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9739 - loss: 0.0676 - val_accuracy: 0.7774 - val_loss: 1.8557
Epoch 33/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9758 - loss: 0.0596 - val_accuracy: 0.7536 - val_loss: 1.6558
Epoch 34/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9808 - loss: 0.0620 - val_accuracy: 0.8047 - val_loss: 0.9413
Epoch 35/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9871 - loss: 0.0343 - val_accuracy: 0.8358 - val_loss: 0.6485
Epoch 36/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 230ms/step - accuracy: 0.9973 - loss: 0.0103 - val_accuracy: 0.8996 - val_loss: 0.3794
Epoch 37/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9983 - loss: 0.0053 - val_accuracy: 0.8540 - val_loss: 0.6895
Epoch 38/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9892 - loss: 0.0347 - val_accuracy: 0.5547 - val_loss: 4.8776
Epoch 39/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9814 - loss: 0.0629 - val_accuracy: 0.5347 - val_loss: 6.7389
Epoch 40/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9717 - loss: 0.0825 - val_accuracy: 0.3613 - val_loss: 11.7176
Epoch 41/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9782 - loss: 0.0640 - val_accuracy: 0.5036 - val_loss: 3.5913
Epoch 42/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9712 - loss: 0.0904 - val_accuracy: 0.8412 - val_loss: 0.6728
Epoch 43/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9943 - loss: 0.0182 - val_accuracy: 0.7810 - val_loss: 0.8857
Epoch 44/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9991 - loss: 0.0084 - val_accuracy: 0.9015 - val_loss: 0.4147
```

```
Epoch 45/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9965 - loss: 0.0091 - val_accuracy: 0.6788 - val_loss: 2.3140
Epoch 46/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9889 - loss: 0.0329 - val_accuracy: 0.7245 - val_loss: 1.2573
Epoch 47/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9636 - loss: 0.0904 - val_accuracy: 0.7847 - val_loss: 1.2039
Epoch 48/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9857 - loss: 0.0348 - val_accuracy: 0.7263 - val_loss: 1.1153
Epoch 49/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9832 - loss: 0.0482 - val_accuracy: 0.6934 - val_loss: 2.3112
Epoch 50/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9870 - loss: 0.0408 - val_accuracy: 0.7810 - val_loss: 1.4597
Epoch 51/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9931 - loss: 0.0178 - val_accuracy: 0.7646 - val_loss: 1.1155
Epoch 52/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9941 - loss: 0.0281 - val_accuracy: 0.8558 - val_loss: 0.7888
Epoch 53/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9947 - loss: 0.0166 - val_accuracy: 0.8960 - val_loss: 0.4773
Epoch 54/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9959 - loss: 0.0111 - val_accuracy: 0.8942 - val_loss: 0.4378
Epoch 55/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9986 - loss: 0.0039 - val_accuracy: 0.8285 - val_loss: 0.7671
Epoch 56/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 230ms/step - accuracy: 1.0000 - loss: 0.0037 - val_accuracy: 0.9288 - val_loss: 0.3673
Epoch 57/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 0.9325 - val_loss: 0.3793
Epoch 58/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 1.0000 - loss: 5.3586e-04 - val_accuracy: 0.9161 - val_loss: 0.4529
Epoch 59/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 1.0000 - loss: 2.6437e-04 - val_accuracy: 0.9234 - val_loss: 0.3686
Epoch 60/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 230ms/step - accuracy: 1.0000 - loss: 1.7951e-04 - val_accuracy: 0.9288 - val_loss: 0.3653
Epoch 61/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 230ms/step - accuracy: 1.0000 - loss: 9.7222e-05 - val_accuracy: 0.9307 - val_loss: 0.3553
Epoch 62/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 1.0000 - loss: 9.4838e-05 - val_accuracy: 0.9307 - val_loss: 0.3665
Epoch 63/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 1.0000 - loss: 1.3985e-04 - val_accuracy: 0.9361 - val_loss: 0.3621
Epoch 64/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 230ms/step - accuracy: 1.0000 - loss: 3.1126e-04 - val_accuracy: 0.9307 - val_loss: 0.3412
Epoch 65/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 1.0000 - loss: 1.1254e-04 - val_accuracy: 0.9307 - val_loss: 0.3639
Epoch 66/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 1.0000 - loss: 5.1451e-05 - val_accuracy: 0.9270 - val_loss: 0.3785
Epoch 67/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 1.0000 - loss: 7.1985e-05 - val_accuracy: 0.9270 - val_loss: 0.3923
Epoch 68/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 1.0000 - loss: 3.8832e-05 - val_accuracy: 0.9270 - val_loss: 0.3919
Epoch 69/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 227ms/step - accuracy: 1.0000 - loss: 4.4930e-05 - val_accuracy: 0.9270 - val_loss: 0.3911
Epoch 70/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 227ms/step - accuracy: 1.0000 - loss: 4.9561e-05 - val_accuracy: 0.9288 - val_loss: 0.3903
Epoch 71/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 1.0000 - loss: 5.1392e-05 - val_accuracy: 0.9325 - val_loss: 0.3862
Epoch 72/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 1.0000 - loss: 3.7137e-05 - val_accuracy: 0.9343 - val_loss: 0.3825
Epoch 73/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 227ms/step - accuracy: 1.0000 - loss: 5.5544e-04 - val_accuracy: 0.9270 - val_loss: 0.4255
Epoch 74/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 227ms/step - accuracy: 1.0000 - loss: 3.5353e-05 - val_accuracy: 0.9270 - val_loss: 0.4300
Epoch 75/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9886 - loss: 0.0388 - val_accuracy: 0.3212 - val_loss: 1588.4146
Epoch 76/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.8066 - loss: 0.5841 - val_accuracy: 0.3540 - val_loss: 61.4763
Epoch 77/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9249 - loss: 0.1907 - val_accuracy: 0.7172 - val_loss: 1.2802
Epoch 78/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9545 - loss: 0.1118 - val_accuracy: 0.7482 - val_loss: 0.7466
Epoch 79/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9730 - loss: 0.0708 - val_accuracy: 0.5894 - val_loss: 1.8251
Epoch 80/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9827 - loss: 0.0483 - val_accuracy: 0.5328 - val_loss: 3.6354
Epoch 81/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9817 - loss: 0.0590 - val_accuracy: 0.8248 - val_loss: 0.6736
Epoch 82/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9894 - loss: 0.0420 - val_accuracy: 0.7609 - val_loss: 1.0746
Epoch 83/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 229ms/step - accuracy: 0.9887 - loss: 0.0299 - val_accuracy: 0.8650 - val_loss: 0.6228
Epoch 84/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 16s 228ms/step - accuracy: 0.9768 - loss: 0.0639 - val_accuracy: 0.5985 - val_loss: 2.0447
Epoch 84: early stopping
Restoring model weights from the end of the best epoch: 64.
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 92ms/step - accuracy: 0.9389 - loss: 0.2821
Retrained ResNet Test accuracy: 93.49%, Test loss: 0.33154359459877014
```





In [11]:
```
with open('model_res_retrained.pkl', 'wb') as file:
    pickle.dump(model_res_retrained, file)
with open('history_res_retrained.pkl', 'wb') as file:
    pickle.dump(history_res_retrained, file)
```

4.2 More advanced CNN

Advanced CNN Model Architecture and Training (`model_pro`) with Early Stopping

Model Architecture

The advanced CNN model, referred to as `model_pro`, enhances the baseline architecture by incorporating additional convolutional layers, increased filter numbers, and inclusion of batch normalization and more dropout layers to manage the complexity and mitigate overfitting:

- **Initial Layers**: Start with two `Conv2D` layers with 64 filters each, followed by `MaxPooling2D`. This setup increases the model's ability to capture more complex features at the beginning of the network.
- **Middle Layers**: Continue with two sets of double `Conv2D` layers with increasing filters (128 and 256), each set followed by a `MaxPooling2D` layer. This structure allows deeper extraction of image features.
- **Batch Normalization**: Each convolutional layer is followed by `BatchNormalization` which normalizes the activations of the previous layer at each batch, maintaining mean output close to 0 and the output standard deviation close to 1. This helps in accelerating the training process and stabilizing the learning environment.
- **Flattening and Dense Layers**: Flatten the output from the convolutional layers to feed into the dense network, which includes a dense layer with 256 units followed by a dropout layer.
- **Output Layer**: Conclude with a `Dense` layer of 3 units and a softmax activation to classify into three categories.

Loss Function and Optimizer

- **Loss Function**: `CategoricalCrossentropy`, ideal for multi-class classification tasks where labels are one-hot encoded.
- **Optimizer**: Use `Adam` optimizer, which adjusts the learning rate throughout training, providing an optimal approach for this more complex model.

Evaluation Metrics and Training Configuration

- **Evaluation Metric**: Accuracy remains the metric of choice to evaluate the performance, providing direct insight into the proportion of correct predictions.
- **Training Configuration**: Train the model over a sufficient number of epochs with a batch size of 32, incorporating callbacks such as Early Stopping to monitor validation loss and prevent overfitting.

Performance Visualization

- **Plots**: Training and validation accuracy and loss are plotted to monitor the model's effectiveness over training epochs, highlighting the impact of added complexity on learning dynamics.

In [12]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
```

```python
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Define the advanced CNN architecture
model_CNN_pro = Sequential([
    Conv2D(64, (3, 3), activation='relu', input_shape=(192, 192, 3)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    BatchNormalization(),

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

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

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')
])

# Compile the model
model_CNN_pro.compile(optimizer=Adam(),
                      loss=CategoricalCrossentropy(),
                      metrics=['accuracy'])

# Model summary
model_CNN_pro.summary()

# Setup early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=20, verbose=1, restore_best_weights=True)

# Prepare data using ImageDataGenerator
train_datagen = ImageDataGenerator(validation_split=0.2)
train_generator = train_datagen.flow(X_train, y_train, subset='training', batch_size=32)
validation_generator = train_datagen.flow(X_train, y_train, subset='validation', batch_size=32)

# Train the model with early stopping
history_CNN_pro = model_CNN_pro.fit(train_generator,
                                    validation_data=validation_generator,
                                    epochs=150,  # Set a higher potential max epochs
                                    callbacks=[early_stopping])

# Evaluate the model on the test set
test_loss_CNN_pro, test_accuracy_CNN_pro = model_CNN_pro.evaluate(X_test, y_test)
print(f"Advanced Model Pro Test accuracy: {test_accuracy_CNN_pro*100:.2f}%, Test loss: {test_loss_CNN_pro}")

# Optionally, visualize the training process
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_CNN_pro.history['accuracy'], label='Training accuracy')
plt.plot(history_CNN_pro.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_CNN_pro.history['loss'], label='Training loss')
plt.plot(history_CNN_pro.history['val_loss'], label='Validation loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_sha
pe`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 190, 190, 64)	1,792
conv2d_1 (Conv2D)	(None, 188, 188, 64)	36,928
max_pooling2d (MaxPooling2D)	(None, 94, 94, 64)	0
batch_normalization (BatchNormalization)	(None, 94, 94, 64)	256
conv2d_2 (Conv2D)	(None, 92, 92, 128)	73,856
conv2d_3 (Conv2D)	(None, 90, 90, 128)	147,584
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 128)	0
batch_normalization_1 (BatchNormalization)	(None, 45, 45, 128)	512
conv2d_4 (Conv2D)	(None, 43, 43, 256)	295,168
conv2d_5 (Conv2D)	(None, 41, 41, 256)	590,080
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 256)	0
batch_normalization_2 (BatchNormalization)	(None, 20, 20, 256)	1,024
flatten (Flatten)	(None, 102400)	0
dense_4 (Dense)	(None, 256)	26,214,656
dropout (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 3)	771

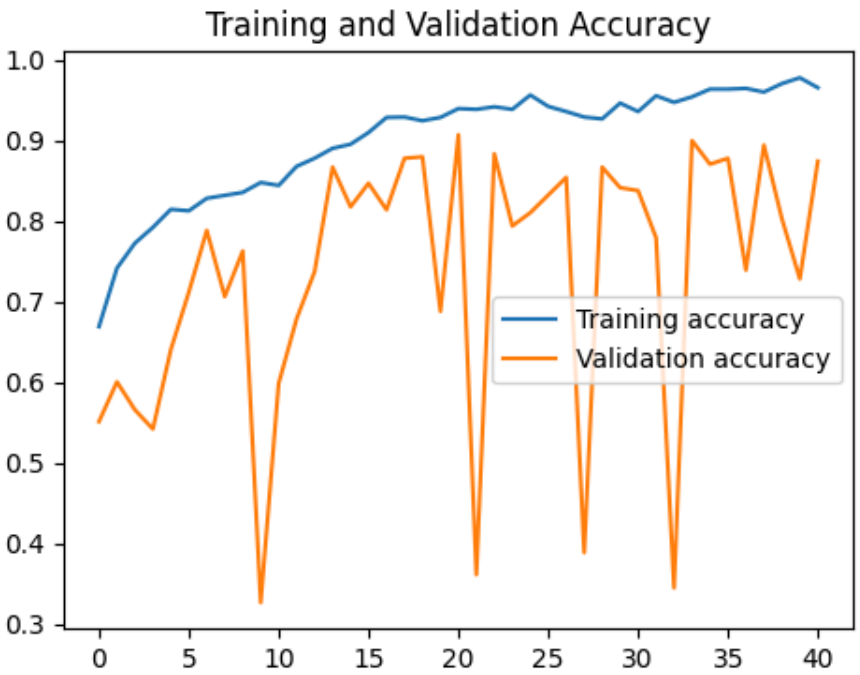
Total params: 27,362,627 (104.38 MB)

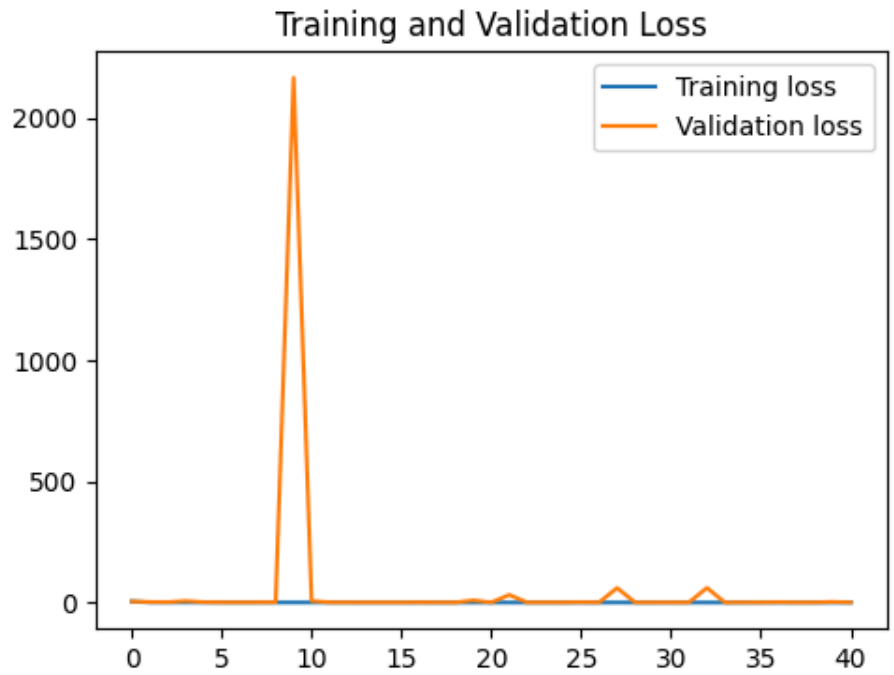
Trainable params: 27,361,731 (104.38 MB)

Non-trainable params: 896 (3.50 KB)

```
Epoch 1/150
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 62s 546ms/step - accuracy: 0.6156 - loss: 8.2015 - val_accuracy: 0.5511 - val_loss: 4.7928
Epoch 2/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 214ms/step - accuracy: 0.7330 - loss: 0.9569 - val_accuracy: 0.6004 - val_loss: 2.2508
Epoch 3/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 215ms/step - accuracy: 0.7636 - loss: 0.6235 - val_accuracy: 0.5657 - val_loss: 1.7105
Epoch 4/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 214ms/step - accuracy: 0.7866 - loss: 0.5389 - val_accuracy: 0.5420 - val_loss: 6.4755
Epoch 5/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 215ms/step - accuracy: 0.8189 - loss: 0.4562 - val_accuracy: 0.6405 - val_loss: 1.6461
Epoch 6/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 215ms/step - accuracy: 0.8232 - loss: 0.4321 - val_accuracy: 0.7117 - val_loss: 1.0244
Epoch 7/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 214ms/step - accuracy: 0.8249 - loss: 0.4212 - val_accuracy: 0.7883 - val_loss: 0.6811
Epoch 8/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 213ms/step - accuracy: 0.8198 - loss: 0.4469 - val_accuracy: 0.7062 - val_loss: 1.0331
Epoch 9/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.8400 - loss: 0.4098 - val_accuracy: 0.7628 - val_loss: 1.4203
Epoch 10/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 213ms/step - accuracy: 0.8350 - loss: 0.5070 - val_accuracy: 0.3266 - val_loss: 2165.2222
Epoch 11/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 213ms/step - accuracy: 0.8553 - loss: 0.4369 - val_accuracy: 0.5985 - val_loss: 6.4689
Epoch 12/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 213ms/step - accuracy: 0.8585 - loss: 0.3477 - val_accuracy: 0.6788 - val_loss: 1.0116
Epoch 13/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.8850 - loss: 0.2868 - val_accuracy: 0.7372 - val_loss: 0.7907
Epoch 14/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 214ms/step - accuracy: 0.8836 - loss: 0.3484 - val_accuracy: 0.8668 - val_loss: 0.4499
Epoch 15/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.8800 - loss: 0.3143 - val_accuracy: 0.8175 - val_loss: 0.7183
Epoch 16/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 214ms/step - accuracy: 0.9179 - loss: 0.2304 - val_accuracy: 0.8467 - val_loss: 0.4318
Epoch 17/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 213ms/step - accuracy: 0.9231 - loss: 0.1867 - val_accuracy: 0.8139 - val_loss: 1.0593
Epoch 18/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9300 - loss: 0.2124 - val_accuracy: 0.8777 - val_loss: 0.5879
Epoch 19/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9245 - loss: 0.1998 - val_accuracy: 0.8796 - val_loss: 0.4390
Epoch 20/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 213ms/step - accuracy: 0.9318 - loss: 0.1790 - val_accuracy: 0.6880 - val_loss: 9.1193
Epoch 21/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 214ms/step - accuracy: 0.9379 - loss: 0.1535 - val_accuracy: 0.9069 - val_loss: 0.4253
Epoch 22/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9301 - loss: 0.1722 - val_accuracy: 0.3613 - val_loss: 31.6846
Epoch 23/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9435 - loss: 0.1628 - val_accuracy: 0.8832 - val_loss: 0.5266
Epoch 24/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9373 - loss: 0.1563 - val_accuracy: 0.7938 - val_loss: 0.8755
Epoch 25/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9533 - loss: 0.1204 - val_accuracy: 0.8102 - val_loss: 0.7166
Epoch 26/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9476 - loss: 0.1323 - val_accuracy: 0.8321 - val_loss: 1.5124
Epoch 27/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9458 - loss: 0.1366 - val_accuracy: 0.8540 - val_loss: 0.6970
Epoch 28/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9299 - loss: 0.2423 - val_accuracy: 0.3887 - val_loss: 60.1108
Epoch 29/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9280 - loss: 0.1892 - val_accuracy: 0.8668 - val_loss: 0.5168
Epoch 30/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9537 - loss: 0.1481 - val_accuracy: 0.8412 - val_loss: 0.5588
Epoch 31/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 211ms/step - accuracy: 0.9350 - loss: 0.1814 - val_accuracy: 0.8376 - val_loss: 0.5703
Epoch 32/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9453 - loss: 0.1485 - val_accuracy: 0.7792 - val_loss: 0.7366
Epoch 33/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 211ms/step - accuracy: 0.9550 - loss: 0.2098 - val_accuracy: 0.3449 - val_loss: 60.6236
Epoch 34/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9439 - loss: 0.1183 - val_accuracy: 0.8996 - val_loss: 0.4559
Epoch 35/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9719 - loss: 0.0804 - val_accuracy: 0.8704 - val_loss: 0.7143
Epoch 36/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9577 - loss: 0.1115 - val_accuracy: 0.8777 - val_loss: 0.8088
Epoch 37/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9719 - loss: 0.0820 - val_accuracy: 0.7391 - val_loss: 1.5852
Epoch 38/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9590 - loss: 0.1110 - val_accuracy: 0.8942 - val_loss: 0.5284
Epoch 39/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9663 - loss: 0.0808 - val_accuracy: 0.8029 - val_loss: 0.7331
Epoch 40/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 212ms/step - accuracy: 0.9829 - loss: 0.0491 - val_accuracy: 0.7281 - val_loss: 2.7560
Epoch 41/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 15s 211ms/step - accuracy: 0.9702 - loss: 0.0810 - val_accuracy: 0.8741 - val_loss: 0.6652
Epoch 41: early stopping
Restoring model weights from the end of the best epoch: 21.
41/41 ━━━━━━━━━━━━━━━━━━━━ 6s 136ms/step - accuracy: 0.9197 - loss: 0.5644
Advanced Model Pro Test accuracy: 91.32%, Test loss: 0.43404921889305115
```





In [13]:
```
with open('model_CNN_pro.pkl', 'wb') as file:
    pickle.dump(model_CNN_pro, file)
with open('history_CNN_pro.pkl', 'wb') as file:
    pickle.dump(history_CNN_pro, file)
```

4.3 VGG16

Fine-Tuning VGG16 for Image Classification (`model_VGG`)

Model Architecture

The `model_VGG` utilizes the VGG16 architecture, which is renowned for its depth and effectiveness in image classification tasks, initially trained on the vast ImageNet dataset. This model has been adapted for our specific classification needs:

- **Base Model**: We start with the VGG16 model, utilizing its pre-trained weights from ImageNet. This model is known for its robustness and deep architecture featuring multiple convolutional layers.
- **Customization**: The top of the model (originally designed for 1000 classes) is replaced with new layers tailored for our dataset, which includes:
  - `GlobalAveragePooling2D` to reduce feature dimensions and prevent overfitting.
  - A `Dense` layer with 512 units followed by a `Dropout` layer at 50% to further mitigate overfitting.
  - A final `Dense` layer with 3 units and a softmax activation function, set up to output probabilities across our three target categories.

Training Configuration and Optimization

- **Freezing Layers**: The convolutional layers of the base VGG16 model are frozen, meaning their weights will not be updated during training. This focuses training on the newly added top layers.
- **Loss Function**: `CategoricalCrossentropy` is used, which is ideal for multi-class classification problems with one-hot encoded labels.
- **Optimizer**: The `Adam` optimizer is selected for its adaptive learning rate capabilities, helping to quickly converge to the optimal solution while fine-tuning.
- **Early Stopping**: To avoid overfitting and optimize training time, early stopping monitors the validation loss, stopping training if there's no improvement for three consecutive epochs. This is complemented by `restore_best_weights` to ensure the model retains the best possible configuration from training.

Evaluation Metrics

- **Accuracy**: This metric is critical for understanding the percentage of correctly classified images, providing a clear measure of model performance.
- **Validation Strategy**: The model is validated during training using a subset of the data (20% of training data), which helps gauge the model's generalization to new data.

Visualization

- **Training and Validation Curves**: We plot both accuracy and loss for training and validation phases to visually assess the model's learning progress and identify any signs of overfitting or underfitting.

This setup ensures that `model_VGG` leverages the sophisticated feature-detection capabilities of VGG16 while being efficiently adapted to our specific classification task. The use of early stopping and other strategies helps maintain an optimal balance

between training duration and model effectiveness.

In [14]:
```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Load the VGG16 model, pre-trained on ImageNet
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(192, 192, 3))

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Add custom layers on top of VGG16
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(3, activation='softmax')(x)  # Assuming 3 classes

# Create the final model
model_VGG = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model_VGG.compile(optimizer=Adam(),
                  loss=CategoricalCrossentropy(),
                  metrics=['accuracy'])

# Model summary
model_VGG.summary()

# Prepare data using ImageDataGenerator
train_datagen = ImageDataGenerator(validation_split=0.2)
train_generator = train_datagen.flow(X_train, y_train, subset='training', batch_size=32)
validation_generator = train_datagen.flow(X_train, y_train, subset='validation', batch_size=32)

# Setup early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=20, verbose=1, restore_best_weights=True)

# Train the model with early stopping
history_VGG = model_VGG.fit(train_generator,
                            validation_data=validation_generator,
                            epochs=150,  # Adjust epochs according to your needs
                            callbacks=[early_stopping])

# Evaluate the model on the test set
test_loss_VGG, test_accuracy_VGG = model_VGG.evaluate(X_test, y_test)
print(f"VGG16 Fine-tuned Model Test accuracy: {test_accuracy_VGG*100:.2f}%, Test loss: {test_loss_VGG}")

# Optionally, visualize the training process
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_VGG.history['accuracy'], label='Training accuracy')
plt.plot(history_VGG.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_VGG.history['loss'], label='Training loss')
plt.plot(history_VGG.history['val_loss'], label='Validation loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
Model: "functional_5"
```

Layer (type)	Output Shape	Param #
input_layer_5 (InputLayer)	(None, 192, 192, 3)	0
block1_conv1 (Conv2D)	(None, 192, 192, 64)	1,792
block1_conv2 (Conv2D)	(None, 192, 192, 64)	36,928
block1_pool (MaxPooling2D)	(None, 96, 96, 64)	0
block2_conv1 (Conv2D)	(None, 96, 96, 128)	73,856
block2_conv2 (Conv2D)	(None, 96, 96, 128)	147,584
block2_pool (MaxPooling2D)	(None, 48, 48, 128)	0
block3_conv1 (Conv2D)	(None, 48, 48, 256)	295,168
block3_conv2 (Conv2D)	(None, 48, 48, 256)	590,080
block3_conv3 (Conv2D)	(None, 48, 48, 256)	590,080
block3_pool (MaxPooling2D)	(None, 24, 24, 256)	0
block4_conv1 (Conv2D)	(None, 24, 24, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 24, 24, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 24, 24, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 12, 12, 512)	0
block5_conv1 (Conv2D)	(None, 12, 12, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 12, 12, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 12, 12, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 6, 6, 512)	0
global_average_pooling2d_4 (GlobalAveragePooling2D)	(None, 512)	0
dense_6 (Dense)	(None, 512)	262,656
dropout_1 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 3)	1,539

Total params: 14,978,883 (57.14 MB)

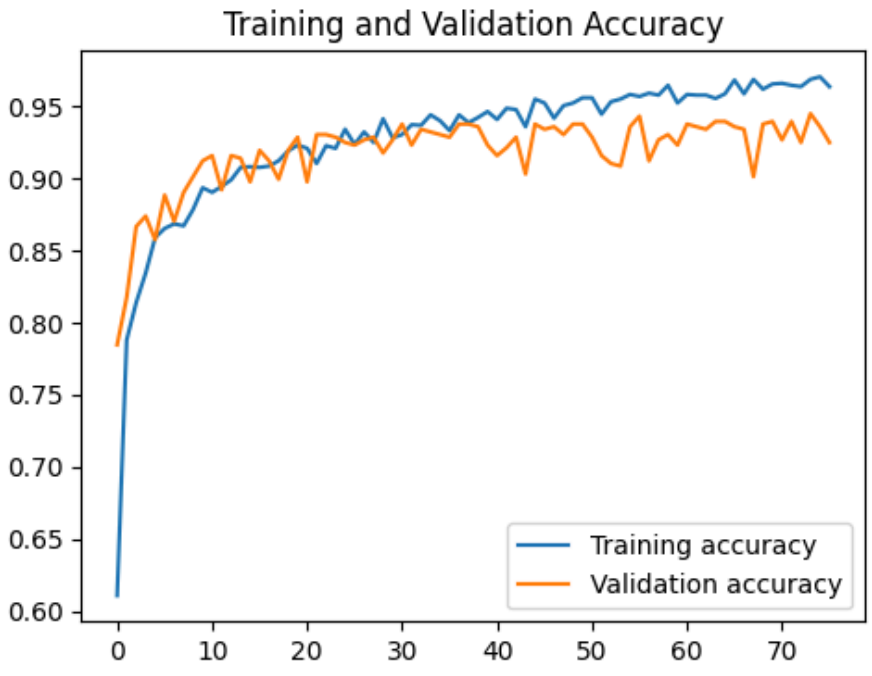
Trainable params: 264,195 (1.01 MB)

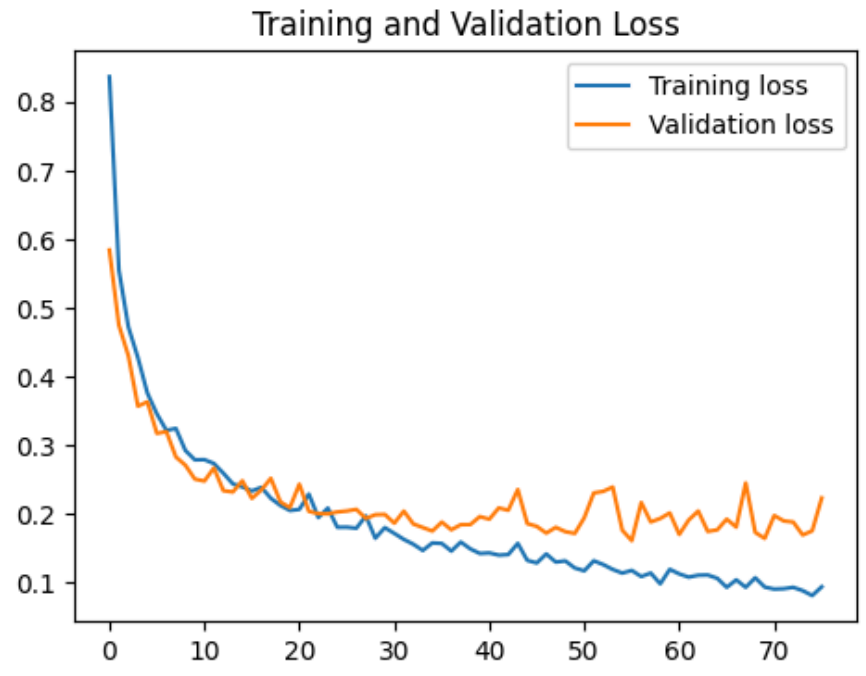
Non-trainable params: 14,714,688 (56.13 MB)

```
Epoch 1/150
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 31s 308ms/step - accuracy: 0.5015 - loss: 0.9962 - val_accuracy: 0.7847 - val_loss: 0.5845
Epoch 2/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 177ms/step - accuracy: 0.7671 - loss: 0.5970 - val_accuracy: 0.8175 - val_loss: 0.4750
Epoch 3/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 175ms/step - accuracy: 0.8044 - loss: 0.4869 - val_accuracy: 0.8668 - val_loss: 0.4304
Epoch 4/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.8327 - loss: 0.4374 - val_accuracy: 0.8741 - val_loss: 0.3571
Epoch 5/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 171ms/step - accuracy: 0.8620 - loss: 0.3803 - val_accuracy: 0.8577 - val_loss: 0.3635
Epoch 6/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 171ms/step - accuracy: 0.8644 - loss: 0.3396 - val_accuracy: 0.8887 - val_loss: 0.3175
Epoch 7/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 171ms/step - accuracy: 0.8656 - loss: 0.3204 - val_accuracy: 0.8704 - val_loss: 0.3201
Epoch 8/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.8595 - loss: 0.3239 - val_accuracy: 0.8905 - val_loss: 0.2832
Epoch 9/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.8755 - loss: 0.2995 - val_accuracy: 0.9015 - val_loss: 0.2711
Epoch 10/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.8965 - loss: 0.2770 - val_accuracy: 0.9124 - val_loss: 0.2510
Epoch 11/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.8880 - loss: 0.2826 - val_accuracy: 0.9161 - val_loss: 0.2481
Epoch 12/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.8985 - loss: 0.2684 - val_accuracy: 0.8923 - val_loss: 0.2671
Epoch 13/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9113 - loss: 0.2509 - val_accuracy: 0.9161 - val_loss: 0.2341
Epoch 14/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9123 - loss: 0.2447 - val_accuracy: 0.9142 - val_loss: 0.2323
Epoch 15/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9118 - loss: 0.2466 - val_accuracy: 0.8978 - val_loss: 0.2485
Epoch 16/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9118 - loss: 0.2243 - val_accuracy: 0.9197 - val_loss: 0.2228
Epoch 17/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9230 - loss: 0.2099 - val_accuracy: 0.9124 - val_loss: 0.2351
Epoch 18/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9154 - loss: 0.2043 - val_accuracy: 0.8996 - val_loss: 0.2520
Epoch 19/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9179 - loss: 0.2135 - val_accuracy: 0.9197 - val_loss: 0.2178
Epoch 20/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9171 - loss: 0.2249 - val_accuracy: 0.9288 - val_loss: 0.2094
Epoch 21/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9197 - loss: 0.2088 - val_accuracy: 0.8978 - val_loss: 0.2435
Epoch 22/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 172ms/step - accuracy: 0.9109 - loss: 0.2315 - val_accuracy: 0.9307 - val_loss: 0.2041
Epoch 23/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 174ms/step - accuracy: 0.9274 - loss: 0.1882 - val_accuracy: 0.9307 - val_loss: 0.2006
Epoch 24/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 175ms/step - accuracy: 0.9301 - loss: 0.1915 - val_accuracy: 0.9288 - val_loss: 0.2004
Epoch 25/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9464 - loss: 0.1687 - val_accuracy: 0.9252 - val_loss: 0.2031
Epoch 26/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9311 - loss: 0.1683 - val_accuracy: 0.9234 - val_loss: 0.2045
Epoch 27/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 171ms/step - accuracy: 0.9304 - loss: 0.1758 - val_accuracy: 0.9270 - val_loss: 0.2069
Epoch 28/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9329 - loss: 0.1756 - val_accuracy: 0.9288 - val_loss: 0.1936
Epoch 29/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9423 - loss: 0.1725 - val_accuracy: 0.9179 - val_loss: 0.1991
Epoch 30/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9273 - loss: 0.1713 - val_accuracy: 0.9270 - val_loss: 0.1995
Epoch 31/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9292 - loss: 0.1839 - val_accuracy: 0.9380 - val_loss: 0.1867
Epoch 32/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9329 - loss: 0.1686 - val_accuracy: 0.9234 - val_loss: 0.2044
Epoch 33/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9398 - loss: 0.1512 - val_accuracy: 0.9343 - val_loss: 0.1856
Epoch 34/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9522 - loss: 0.1319 - val_accuracy: 0.9325 - val_loss: 0.1805
Epoch 35/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9398 - loss: 0.1483 - val_accuracy: 0.9307 - val_loss: 0.1753
Epoch 36/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9348 - loss: 0.1645 - val_accuracy: 0.9288 - val_loss: 0.1883
Epoch 37/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9381 - loss: 0.1573 - val_accuracy: 0.9380 - val_loss: 0.1771
Epoch 38/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9474 - loss: 0.1387 - val_accuracy: 0.9380 - val_loss: 0.1846
Epoch 39/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 171ms/step - accuracy: 0.9431 - loss: 0.1466 - val_accuracy: 0.9361 - val_loss: 0.1847
Epoch 40/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9495 - loss: 0.1304 - val_accuracy: 0.9234 - val_loss: 0.1963
Epoch 41/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9449 - loss: 0.1389 - val_accuracy: 0.9161 - val_loss: 0.1926
Epoch 42/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9449 - loss: 0.1443 - val_accuracy: 0.9215 - val_loss: 0.2089
Epoch 43/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9553 - loss: 0.1385 - val_accuracy: 0.9288 - val_loss: 0.2056
Epoch 44/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9427 - loss: 0.1457 - val_accuracy: 0.9033 - val_loss: 0.2358
```

```
Epoch 45/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9583 - loss: 0.1406 - val_accuracy: 0.9380 - val_loss: 0.1862
Epoch 46/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9471 - loss: 0.1432 - val_accuracy: 0.9343 - val_loss: 0.1821
Epoch 47/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9384 - loss: 0.1372 - val_accuracy: 0.9361 - val_loss: 0.1723
Epoch 48/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9480 - loss: 0.1319 - val_accuracy: 0.9307 - val_loss: 0.1807
Epoch 49/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9504 - loss: 0.1425 - val_accuracy: 0.9380 - val_loss: 0.1745
Epoch 50/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9594 - loss: 0.1207 - val_accuracy: 0.9380 - val_loss: 0.1718
Epoch 51/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9499 - loss: 0.1351 - val_accuracy: 0.9288 - val_loss: 0.1947
Epoch 52/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9467 - loss: 0.1224 - val_accuracy: 0.9161 - val_loss: 0.2307
Epoch 53/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9508 - loss: 0.1362 - val_accuracy: 0.9106 - val_loss: 0.2331
Epoch 54/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9555 - loss: 0.1305 - val_accuracy: 0.9088 - val_loss: 0.2394
Epoch 55/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9592 - loss: 0.1253 - val_accuracy: 0.9361 - val_loss: 0.1765
Epoch 56/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9613 - loss: 0.1113 - val_accuracy: 0.9434 - val_loss: 0.1613
Epoch 57/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9586 - loss: 0.1095 - val_accuracy: 0.9124 - val_loss: 0.2168
Epoch 58/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 171ms/step - accuracy: 0.9530 - loss: 0.1179 - val_accuracy: 0.9270 - val_loss: 0.1885
Epoch 59/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9656 - loss: 0.1060 - val_accuracy: 0.9307 - val_loss: 0.1936
Epoch 60/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9484 - loss: 0.1329 - val_accuracy: 0.9234 - val_loss: 0.2017
Epoch 61/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9595 - loss: 0.1055 - val_accuracy: 0.9380 - val_loss: 0.1704
Epoch 62/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 173ms/step - accuracy: 0.9631 - loss: 0.0985 - val_accuracy: 0.9361 - val_loss: 0.1914
Epoch 63/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9561 - loss: 0.1146 - val_accuracy: 0.9343 - val_loss: 0.2044
Epoch 64/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9584 - loss: 0.1154 - val_accuracy: 0.9398 - val_loss: 0.1747
Epoch 65/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9537 - loss: 0.1165 - val_accuracy: 0.9398 - val_loss: 0.1771
Epoch 66/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9675 - loss: 0.0972 - val_accuracy: 0.9361 - val_loss: 0.1926
Epoch 67/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9670 - loss: 0.0896 - val_accuracy: 0.9343 - val_loss: 0.1811
Epoch 68/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9708 - loss: 0.0846 - val_accuracy: 0.9015 - val_loss: 0.2453
Epoch 69/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9601 - loss: 0.1128 - val_accuracy: 0.9380 - val_loss: 0.1733
Epoch 70/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9656 - loss: 0.0964 - val_accuracy: 0.9398 - val_loss: 0.1646
Epoch 71/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9719 - loss: 0.0814 - val_accuracy: 0.9270 - val_loss: 0.1979
Epoch 72/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9588 - loss: 0.1052 - val_accuracy: 0.9398 - val_loss: 0.1900
Epoch 73/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9666 - loss: 0.0903 - val_accuracy: 0.9252 - val_loss: 0.1884
Epoch 74/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9656 - loss: 0.0898 - val_accuracy: 0.9453 - val_loss: 0.1695
Epoch 75/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9731 - loss: 0.0770 - val_accuracy: 0.9361 - val_loss: 0.1756
Epoch 76/150
69/69 ━━━━━━━━━━━━━━━━━━━━ 12s 172ms/step - accuracy: 0.9699 - loss: 0.0870 - val_accuracy: 0.9252 - val_loss: 0.2235
Epoch 76: early stopping
Restoring model weights from the end of the best epoch: 56.
41/41 ━━━━━━━━━━━━━━━━━━━━ 10s 241ms/step - accuracy: 0.9344 - loss: 0.1669
VGG16 Fine-tuned Model Test accuracy: 93.18%, Test loss: 0.18661589920520782
```





In [15]:
```
with open('model_VGG.pkl', 'wb') as file:
    pickle.dump(model_VGG, file)
with open('history_VGG.pkl', 'wb') as file:
    pickle.dump(history_VGG, file)
```

5 Performance Comparison

In [29]:
```
model_test_results = {
    'model_CNN': {
        'test_loss':test_loss_CNN,
        'test_accuracy': test_accuracy_CNN
    },
    'model_res': {
        'test_loss': test_loss_res,
        'test_accuracy': test_accuracy_res
    },
    'model_res_retrained': {
        'test_loss': test_loss_res_retrained,
        'test_accuracy': test_accuracy_res_retrained
    },
    'model_CNN_pro': {
        'test_loss': test_loss_CNN_pro,
        'test_accuracy': test_accuracy_CNN_pro
    },
    'model_VGG': {
        'test_loss': test_loss_VGG,
        'test_accuracy': test_accuracy_VGG
    }
}

with open('model_test_results.pkl', 'wb') as file:
    pickle.dump(model_test_results, file)
```

In [32]:
```
import pandas as pd
import matplotlib.pyplot as plt

model_CNN = pd.read_pickle('model_CNN.pkl')
model_res = pd.read_pickle('model_res.pkl')
model_res_retrained = pd.read_pickle('model_res_retrained.pkl')
model_pro = pd.read_pickle('model_CNN_pro.pkl')
model_VGG = pd.read_pickle('model_VGG.pkl')

history_CNN = pd.read_pickle('history_CNN.pkl')
history_res = pd.read_pickle('history_res.pkl')
history_res_retrained = pd.read_pickle('history_res_retrained.pkl')
history_CNN_pro = pd.read_pickle('history_CNN_pro.pkl')
history_VGG = pd.read_pickle('history_VGG.pkl')

model_test_results = pd.read_pickle('model_test_results.pkl')
test_loss_CNN = model_test_results['model_CNN']['test_loss']
test_accuracy_CNN = model_test_results['model_CNN']['test_accuracy']
test_loss_res = model_test_results['model_res']['test_loss']
test_accuracy_res = model_test_results['model_res']['test_accuracy']
test_loss_res_retrained = model_test_results['model_res_retrained']['test_loss']
test_accuracy_res_retrained = model_test_results['model_res_retrained']['test_accuracy']
test_loss_CNN_pro = model_test_results['model_CNN_pro']['test_loss']
test_accuracy_CNN_pro = model_test_results['model_CNN_pro']['test_accuracy']
test_loss_VGG = model_test_results['model_VGG']['test_loss']
test_accuracy_VGG = model_test_results['model_VGG']['test_accuracy']


# First, gather the necessary data from each model
models = {
```

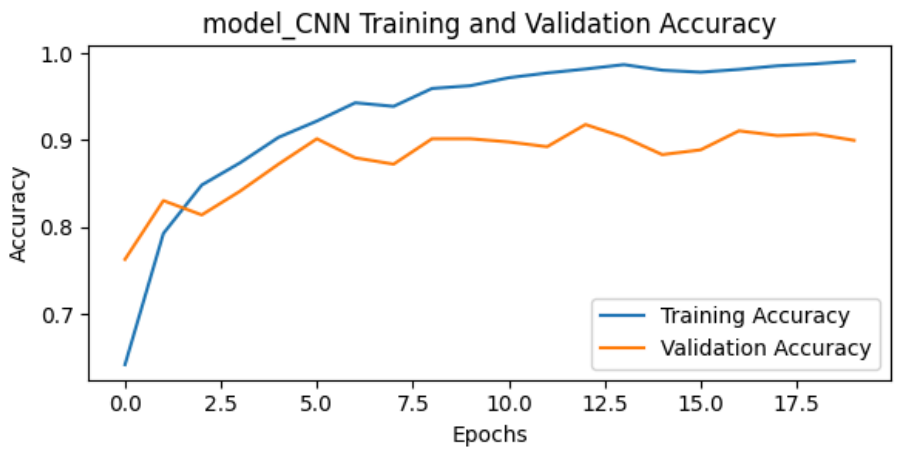
```
    'model_CNN': model_CNN,              # baseline CNN
    'model_res': model_res,              # pre-trained resnet
    'model_res_retrained': model_res_retrained,    # retrained resnet
    'model_CNN_pro': model_CNN_pro,            # more layer CNN
    'model_VGG': model_VGG              # pre-trained VGG16
}

# Create a dictionary to hold the data
data = {
    'Model': [],
    'Test Loss': [],
    'Test Accuracy': [],
    'Total Parameters': [],
    'Trainable Parameters': []
}

# Populate the dictionary with data from each model
for name, model in models.items():
    suffix = '_'.join(name.split('_')[1:])  # Join parts to handle names like 'model_CNN_pro'
    test_loss = globals()[f"test_loss_{suffix}"]
    test_accuracy = globals()[f"test_accuracy_{suffix}"] * 100  # Convert to percentage
    total_params = model.count_params()
    trainable_params = sum([tf.size(p).numpy() for p in model.trainable_weights])

    data['Model'].append(name)
    data['Test Loss'].append(test_loss)
    data['Test Accuracy'].append(test_accuracy)
    data['Total Parameters'].append(total_params)
    data['Trainable Parameters'].append(trainable_params)

# Create DataFrame
df_models = pd.DataFrame(data)

# Display the DataFrame
display(df_models)
```

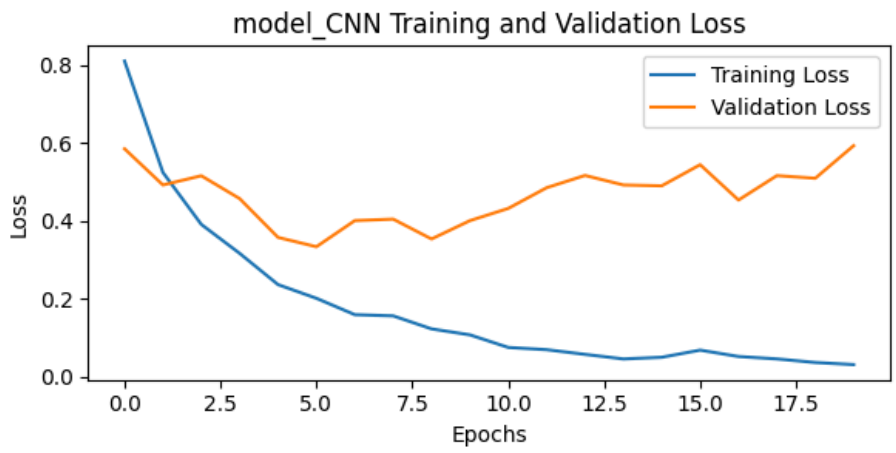
	Model	Test Loss	Test Accuracy	Total Parameters	Trainable Parameters
0	model_CNN	0.548303	89.930284	8023619	8023619
1	model_res	0.664706	75.987607	23593859	6147
2	model_res_retrained	0.331544	93.493414	23593859	23540739
3	model_CNN_pro	0.434049	91.324556	27362627	27361731
4	model_VGG	0.186616	93.183577	14978883	264195

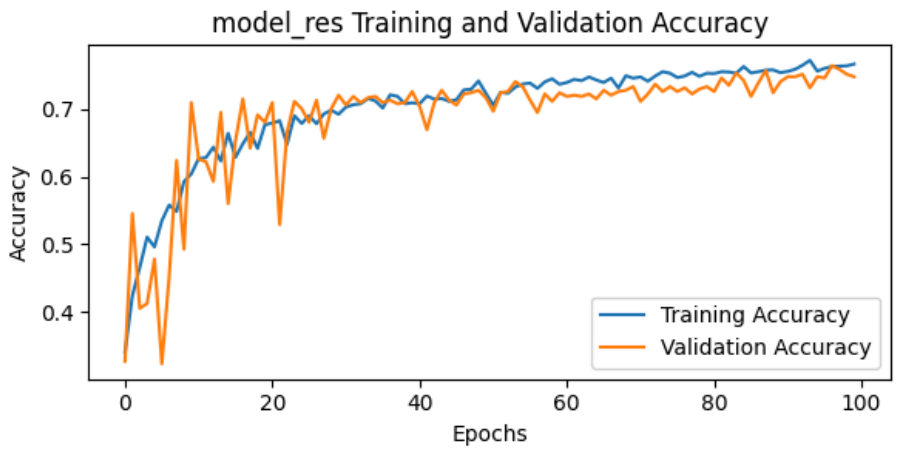
For the hyperparameters, I maintained stepsize all the same across all the models, which is set to the default value of Adam optimizer, which is 0.001. The optimizer are all Adam, and the loss functions are all CategoricalCrossentropy. The epoch numbers are cleverly defined using early stop in each senarios. The batch_size are all intentionally set to 32.
I intentionally set all those hyperparameters the same because I want to mainly focus on the difference in architecture. I want to study how the difference in number of layers and difference in ways of connections could affect model performance. The OOS test result is shown in the above, and the history of training is shown below.

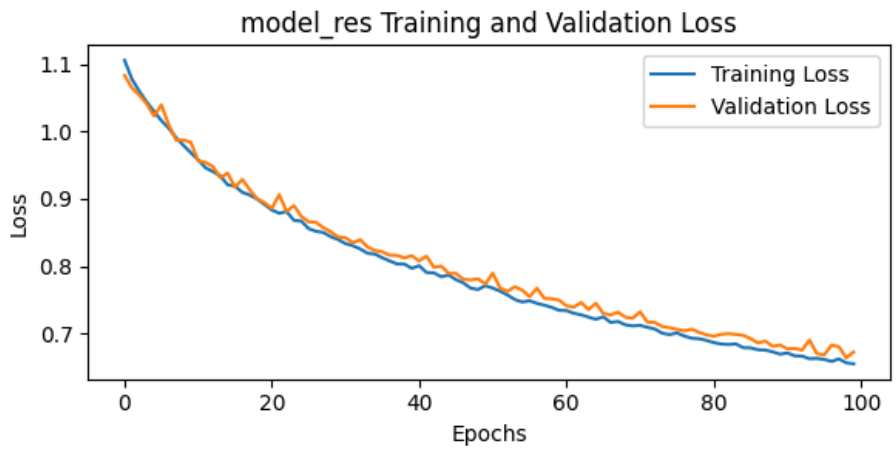
In [33]:
```
import matplotlib.pyplot as plt

# Assuming you have the following histories from your models
histories = {
    'model_CNN': history_CNN,
    'model_res': history_res,
    'model_res_retrained': history_res_retrained,
    'model_pro': history_CNN_pro,
    'model_VGG': history_VGG
}

# Number of plots
num_models = len(histories)
fig, axs = plt.subplots(num_models, 2, figsize=(12, 3 * num_models))  # 2 columns for accuracy and loss

for i, (name, history) in enumerate(histories.items()):
    # Plot training and validation accuracy
    axs[i, 0].plot(history.history['accuracy'], label='Training Accuracy')
    axs[i, 0].plot(history.history['val_accuracy'], label='Validation Accuracy')
    axs[i, 0].set_title(f'{name} Training and Validation Accuracy')
    axs[i, 0].set_xlabel('Epochs')
    axs[i, 0].set_ylabel('Accuracy')
    axs[i, 0].legend()

    # Plot training and validation loss
    axs[i, 1].plot(history.history['loss'], label='Training Loss')
    axs[i, 1].plot(history.history['val_loss'], label='Validation Loss')
    axs[i, 1].set_title(f'{name} Training and Validation Loss')
    axs[i, 1].set_xlabel('Epochs')
    axs[i, 1].set_ylabel('Loss')
```

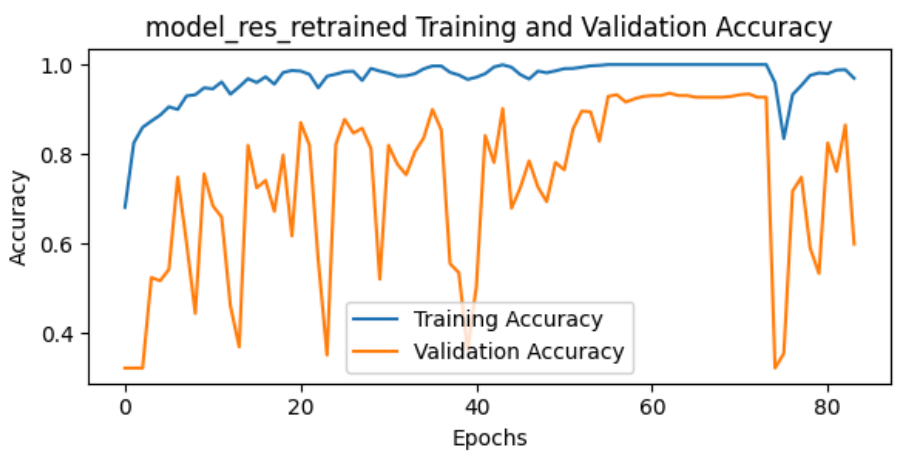
```
    axs[i, 1].legend()

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

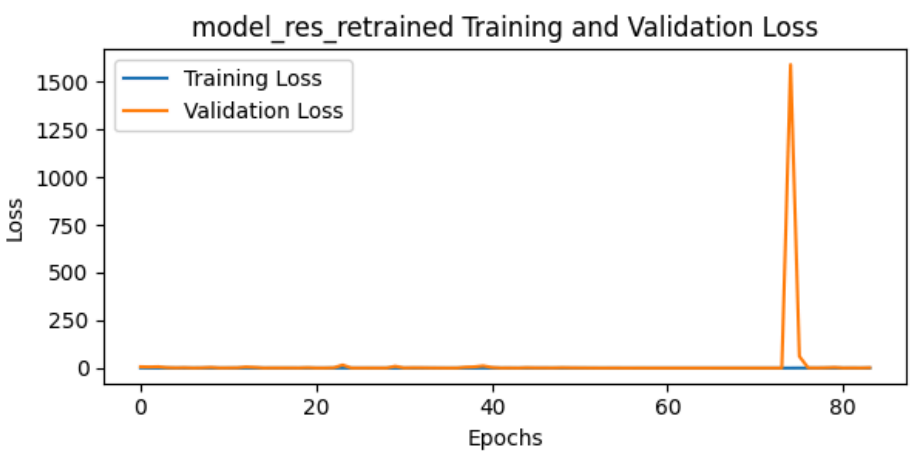


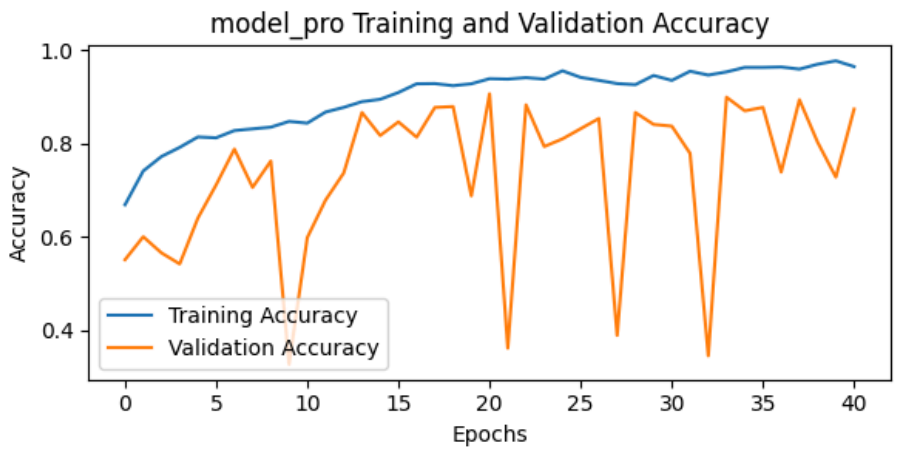


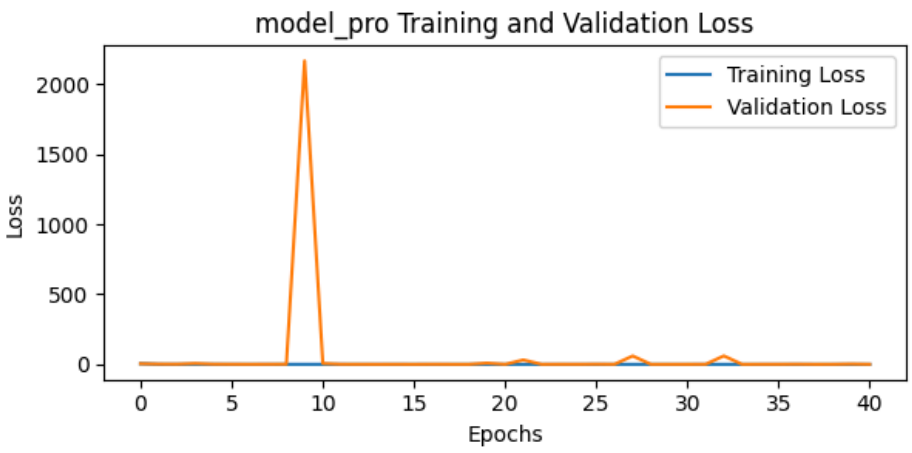


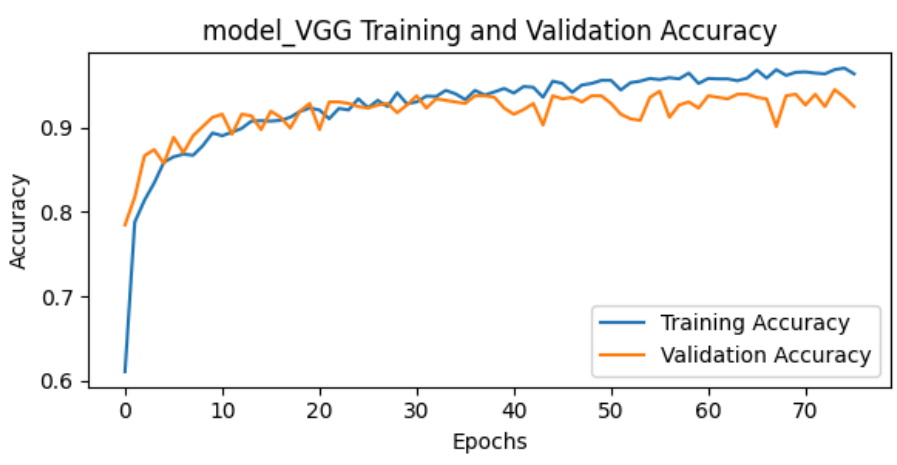


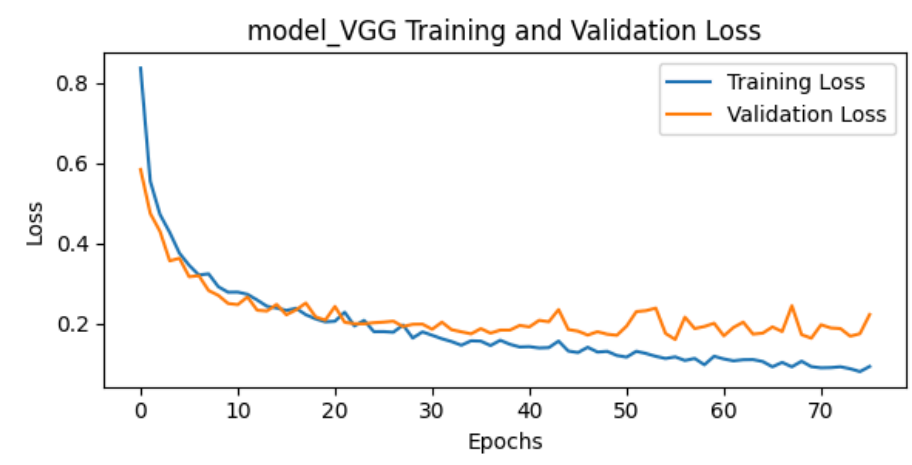












6 Augmentation

Enhanced Training of ResNet50 with Data Augmentation (`model_res_retrained_aug`)

Model Overview

The `model_res_retrained_aug` leverages the architecture of ResNet50, a powerful deep neural network, known for its effectiveness in large-scale image recognition tasks. Unlike its predecessors, this model is trained from scratch without leveraging pre-trained weights and incorporates data augmentation to improve its robustness and ability to generalize from the training data.

Architecture Adjustments

- **Base Model**: Utilizes ResNet50's architecture, excluding the top layer to allow customization for our specific task of classifying images into three categories.
- **Global Average Pooling**: Reduces each feature map to a single vector, decreasing the model's complexity and computational demand.
- **Output Layer**: A dense layer with three units (one per class) uses softmax activation to output class probabilities.

Data Augmentation

To enhance the model's ability to generalize and perform well on unseen data, data augmentation techniques are applied during training:

- **Rotation**: Random rotations of up to 20 degrees help the model handle varying orientations.
- **Width and Height Shifts**: Shifts of up to 20% in both dimensions teach the model to recognize objects that aren't perfectly centered.
- **Shear**: Introduces shear transformations to simulate the effect of viewing angles.
- **Zoom**: Random zooms help the model deal with objects of varying sizes.
- **Horizontal Flip**: Mirrors images, doubling the dataset size in terms of horizontal variance.
- **Fill Mode**: 'Nearest' fill mode is used to handle newly introduced pixels after transformations.

Training Configuration

- **Optimizer**: The `Adam` optimizer is used for its adaptive learning rate capabilities, which helps in finding optimal weights efficiently.
- **Loss Function**: `CategoricalCrossentropy` is ideal for multi-class classification, providing a measure of the difference between the predicted probabilities and the actual distribution.
- **Early Stopping**: Implemented to cease training if the validation loss does not improve for three consecutive epochs, ensuring the model does not overfit and can generalize well to new data.

Model Evaluation and Visualization

- **Evaluation**: The model is assessed on a separate test set to measure its accuracy and loss, ensuring that it performs as expected on data it has never seen.
- **Visualization**: Training and validation accuracy and loss are plotted to track the model's performance over epochs, providing insights into its learning dynamics and the effectiveness of data augmentation.

This detailed setup ensures that `model_res_retrained_aug` not only learns from the training data but also develops an ability to perform well across diverse scenarios, which is crucial for practical applications where conditions can vary significantly from the training environment.

In [39]:
```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Load the ResNet50 model without pre-trained weights
base_model = ResNet50(weights=None, include_top=False, input_shape=(192, 192, 3))

# Set all layers to be trainable
for layer in base_model.layers:
    layer.trainable = True

# Add custom layers on top of ResNet
x = GlobalAveragePooling2D()(base_model.output)
predictions = Dense(3, activation='softmax')(x)  # Assuming 3 classes

# Create the final model and rename to model_res_retrained_aug
model_res_retrained_aug = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model_res_retrained_aug.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Model summary
model_res_retrained_aug.summary()
```

```python
# Prepare data using ImageDataGenerator with augmentation
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2  # using 20% of the data for validation
)

# Create training and validation generators
train_generator = train_datagen.flow(X_train, y_train, subset='training', batch_size=32)
validation_generator = train_datagen.flow(X_train, y_train, subset='validation', batch_size=32)

# Setup early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=100, verbose=1, restore_best_weights=True)

# Train the model with early stopping
history_res_retrained_aug = model_res_retrained_aug.fit(train_generator,
                            validation_data=validation_generator,
                            epochs=1000,  # Set a higher potential max epochs
                            callbacks=[early_stopping])

# Evaluate the model on the test set
test_loss_res_retrained_aug, test_accuracy_res_retrained_aug = model_res_retrained_aug.evaluate(X_test, y_test)
print(f"Retrained ResNet with Augmentation Test accuracy: {test_accuracy_res_retrained_aug*100:.2f}%, Test loss: {test_loss_re

# Optionally, visualize the training process
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_res_retrained_aug.history['accuracy'], label='Training accuracy')
plt.plot(history_res_retrained_aug.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_res_retrained_aug.history['loss'], label='Training loss')
plt.plot(history_res_retrained_aug.history['val_loss'], label='Validation loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

Model: "functional_428"

Layer (type)	Output Shape	Param #	Connected to
input_layer_12 (InputLayer)	(None, 192, 192, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 198, 198, 3)	0	input_layer_12[0][0]
conv1_conv (Conv2D)	(None, 96, 96, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalization)	(None, 96, 96, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 96, 96, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 98, 98, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 48, 48, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 48, 48, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block1_1_conv[0…
conv2_block1_1_relu (Activation)	(None, 48, 48, 64)	0	conv2_block1_1_bn[0][…
conv2_block1_2_conv (Conv2D)	(None, 48, 48, 64)	36,928	conv2_block1_1_relu[0…
conv2_block1_2_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block1_2_conv[0…
conv2_block1_2_relu (Activation)	(None, 48, 48, 64)	0	conv2_block1_2_bn[0][…
conv2_block1_0_conv (Conv2D)	(None, 48, 48, 256)	16,640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 48, 48, 256)	16,640	conv2_block1_2_relu[0…
conv2_block1_0_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block1_0_conv[0…
conv2_block1_3_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block1_3_conv[0…
conv2_block1_add (Add)	(None, 48, 48, 256)	0	conv2_block1_0_bn[0][… conv2_block1_3_bn[0][…
conv2_block1_out (Activation)	(None, 48, 48, 256)	0	conv2_block1_add[0][0]
conv2_block2_1_conv (Conv2D)	(None, 48, 48, 64)	16,448	conv2_block1_out[0][0]
conv2_block2_1_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block2_1_conv[0…
conv2_block2_1_relu (Activation)	(None, 48, 48, 64)	0	conv2_block2_1_bn[0][…
conv2_block2_2_conv (Conv2D)	(None, 48, 48, 64)	36,928	conv2_block2_1_relu[0…
conv2_block2_2_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block2_2_conv[0…
conv2_block2_2_relu (Activation)	(None, 48, 48, 64)	0	conv2_block2_2_bn[0][…
conv2_block2_3_conv (Conv2D)	(None, 48, 48, 256)	16,640	conv2_block2_2_relu[0…
conv2_block2_3_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block2_3_conv[0…
conv2_block2_add (Add)	(None, 48, 48, 256)	0	conv2_block1_out[0][0… conv2_block2_3_bn[0][…
conv2_block2_out (Activation)	(None, 48, 48, 256)	0	conv2_block2_add[0][0]
conv2_block3_1_conv (Conv2D)	(None, 48, 48, 64)	16,448	conv2_block2_out[0][0]

conv2_block3_1_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block3_1_conv[0…
conv2_block3_1_relu (Activation)	(None, 48, 48, 64)	0	conv2_block3_1_bn[0][…
conv2_block3_2_conv (Conv2D)	(None, 48, 48, 64)	36,928	conv2_block3_1_relu[0…
conv2_block3_2_bn (BatchNormalization)	(None, 48, 48, 64)	256	conv2_block3_2_conv[0…
conv2_block3_2_relu (Activation)	(None, 48, 48, 64)	0	conv2_block3_2_bn[0][…
conv2_block3_3_conv (Conv2D)	(None, 48, 48, 256)	16,640	conv2_block3_2_relu[0…
conv2_block3_3_bn (BatchNormalization)	(None, 48, 48, 256)	1,024	conv2_block3_3_conv[0…
conv2_block3_add (Add)	(None, 48, 48, 256)	0	conv2_block2_out[0][0… conv2_block3_3_bn[0][…
conv2_block3_out (Activation)	(None, 48, 48, 256)	0	conv2_block3_add[0][0]
conv3_block1_1_conv (Conv2D)	(None, 24, 24, 128)	32,896	conv2_block3_out[0][0]
conv3_block1_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block1_1_conv[0…
conv3_block1_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block1_1_bn[0][…
conv3_block1_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block1_1_relu[0…
conv3_block1_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block1_2_conv[0…
conv3_block1_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block1_2_bn[0][…
conv3_block1_0_conv (Conv2D)	(None, 24, 24, 512)	131,584	conv2_block3_out[0][0]
conv3_block1_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block1_2_relu[0…
conv3_block1_0_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block1_0_conv[0…
conv3_block1_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block1_3_conv[0…
conv3_block1_add (Add)	(None, 24, 24, 512)	0	conv3_block1_0_bn[0][… conv3_block1_3_bn[0][…
conv3_block1_out (Activation)	(None, 24, 24, 512)	0	conv3_block1_add[0][0]
conv3_block2_1_conv (Conv2D)	(None, 24, 24, 128)	65,664	conv3_block1_out[0][0]
conv3_block2_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block2_1_conv[0…
conv3_block2_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block2_1_bn[0][…
conv3_block2_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block2_1_relu[0…
conv3_block2_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block2_2_conv[0…
conv3_block2_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block2_2_bn[0][…
conv3_block2_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block2_2_relu[0…
conv3_block2_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block2_3_conv[0…

conv3_block2_add (Add)	(None, 24, 24, 512)	0	conv3_block1_out[0][0… conv3_block2_3_bn[0][…
conv3_block2_out (Activation)	(None, 24, 24, 512)	0	conv3_block2_add[0][0]
conv3_block3_1_conv (Conv2D)	(None, 24, 24, 128)	65,664	conv3_block2_out[0][0]
conv3_block3_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block3_1_conv[0…
conv3_block3_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block3_1_bn[0][…
conv3_block3_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block3_1_relu[0…
conv3_block3_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block3_2_conv[0…
conv3_block3_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block3_2_bn[0][…
conv3_block3_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block3_2_relu[0…
conv3_block3_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block3_3_conv[0…
conv3_block3_add (Add)	(None, 24, 24, 512)	0	conv3_block2_out[0][0… conv3_block3_3_bn[0][…
conv3_block3_out (Activation)	(None, 24, 24, 512)	0	conv3_block3_add[0][0]
conv3_block4_1_conv (Conv2D)	(None, 24, 24, 128)	65,664	conv3_block3_out[0][0]
conv3_block4_1_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block4_1_conv[0…
conv3_block4_1_relu (Activation)	(None, 24, 24, 128)	0	conv3_block4_1_bn[0][…
conv3_block4_2_conv (Conv2D)	(None, 24, 24, 128)	147,584	conv3_block4_1_relu[0…
conv3_block4_2_bn (BatchNormalization)	(None, 24, 24, 128)	512	conv3_block4_2_conv[0…
conv3_block4_2_relu (Activation)	(None, 24, 24, 128)	0	conv3_block4_2_bn[0][…
conv3_block4_3_conv (Conv2D)	(None, 24, 24, 512)	66,048	conv3_block4_2_relu[0…
conv3_block4_3_bn (BatchNormalization)	(None, 24, 24, 512)	2,048	conv3_block4_3_conv[0…
conv3_block4_add (Add)	(None, 24, 24, 512)	0	conv3_block3_out[0][0… conv3_block4_3_bn[0][…
conv3_block4_out (Activation)	(None, 24, 24, 512)	0	conv3_block4_add[0][0]
conv4_block1_1_conv (Conv2D)	(None, 12, 12, 256)	131,328	conv3_block4_out[0][0]
conv4_block1_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block1_1_conv[0…
conv4_block1_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block1_1_bn[0][…
conv4_block1_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block1_1_relu[0…
conv4_block1_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block1_2_conv[0…
conv4_block1_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block1_2_bn[0][…
conv4_block1_0_conv (Conv2D)	(None, 12, 12, 1024)	525,312	conv3_block4_out[0][0]

conv4_block1_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block1_2_relu[0…
conv4_block1_0_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block1_0_conv[0…
conv4_block1_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block1_3_conv[0…
conv4_block1_add (Add)	(None, 12, 12, 1024)	0	conv4_block1_0_bn[0][… conv4_block1_3_bn[0][…
conv4_block1_out (Activation)	(None, 12, 12, 1024)	0	conv4_block1_add[0][0]
conv4_block2_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block1_out[0][0]
conv4_block2_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block2_1_conv[0…
conv4_block2_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block2_1_bn[0][…
conv4_block2_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block2_1_relu[0…
conv4_block2_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block2_2_conv[0…
conv4_block2_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block2_2_bn[0][…
conv4_block2_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block2_2_relu[0…
conv4_block2_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block2_3_conv[0…
conv4_block2_add (Add)	(None, 12, 12, 1024)	0	conv4_block1_out[0][0… conv4_block2_3_bn[0][…
conv4_block2_out (Activation)	(None, 12, 12, 1024)	0	conv4_block2_add[0][0]
conv4_block3_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block2_out[0][0]
conv4_block3_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block3_1_conv[0…
conv4_block3_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block3_1_bn[0][…
conv4_block3_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block3_1_relu[0…
conv4_block3_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block3_2_conv[0…
conv4_block3_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block3_2_bn[0][…
conv4_block3_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block3_2_relu[0…
conv4_block3_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block3_3_conv[0…
conv4_block3_add (Add)	(None, 12, 12, 1024)	0	conv4_block2_out[0][0… conv4_block3_3_bn[0][…
conv4_block3_out (Activation)	(None, 12, 12, 1024)	0	conv4_block3_add[0][0]
conv4_block4_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block3_out[0][0]
conv4_block4_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block4_1_conv[0…
conv4_block4_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block4_1_bn[0][…
conv4_block4_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block4_1_relu[0…

conv4_block4_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block4_2_conv[0…
conv4_block4_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block4_2_bn[0][…
conv4_block4_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block4_2_relu[0…
conv4_block4_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block4_3_conv[0…
conv4_block4_add (Add)	(None, 12, 12, 1024)	0	conv4_block3_out[0][0… conv4_block4_3_bn[0][…
conv4_block4_out (Activation)	(None, 12, 12, 1024)	0	conv4_block4_add[0][0]
conv4_block5_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block4_out[0][0]
conv4_block5_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block5_1_conv[0…
conv4_block5_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block5_1_bn[0][…
conv4_block5_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block5_1_relu[0…
conv4_block5_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block5_2_conv[0…
conv4_block5_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block5_2_bn[0][…
conv4_block5_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block5_2_relu[0…
conv4_block5_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block5_3_conv[0…
conv4_block5_add (Add)	(None, 12, 12, 1024)	0	conv4_block4_out[0][0… conv4_block5_3_bn[0][…
conv4_block5_out (Activation)	(None, 12, 12, 1024)	0	conv4_block5_add[0][0]
conv4_block6_1_conv (Conv2D)	(None, 12, 12, 256)	262,400	conv4_block5_out[0][0]
conv4_block6_1_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block6_1_conv[0…
conv4_block6_1_relu (Activation)	(None, 12, 12, 256)	0	conv4_block6_1_bn[0][…
conv4_block6_2_conv (Conv2D)	(None, 12, 12, 256)	590,080	conv4_block6_1_relu[0…
conv4_block6_2_bn (BatchNormalization)	(None, 12, 12, 256)	1,024	conv4_block6_2_conv[0…
conv4_block6_2_relu (Activation)	(None, 12, 12, 256)	0	conv4_block6_2_bn[0][…
conv4_block6_3_conv (Conv2D)	(None, 12, 12, 1024)	263,168	conv4_block6_2_relu[0…
conv4_block6_3_bn (BatchNormalization)	(None, 12, 12, 1024)	4,096	conv4_block6_3_conv[0…
conv4_block6_add (Add)	(None, 12, 12, 1024)	0	conv4_block5_out[0][0… conv4_block6_3_bn[0][…
conv4_block6_out (Activation)	(None, 12, 12, 1024)	0	conv4_block6_add[0][0]
conv5_block1_1_conv (Conv2D)	(None, 6, 6, 512)	524,800	conv4_block6_out[0][0]
conv5_block1_1_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block1_1_conv[0…
conv5_block1_1_relu (Activation)	(None, 6, 6, 512)	0	conv5_block1_1_bn[0][…

conv5_block1_2_conv (Conv2D)	(None, 6, 6, 512)	2,359,808	conv5_block1_1_relu[0…
conv5_block1_2_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block1_2_conv[0…
conv5_block1_2_relu (Activation)	(None, 6, 6, 512)	0	conv5_block1_2_bn[0][…
conv5_block1_0_conv (Conv2D)	(None, 6, 6, 2048)	2,099,200	conv4_block6_out[0][0]
conv5_block1_3_conv (Conv2D)	(None, 6, 6, 2048)	1,050,624	conv5_block1_2_relu[0…
conv5_block1_0_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block1_0_conv[0…
conv5_block1_3_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block1_3_conv[0…
conv5_block1_add (Add)	(None, 6, 6, 2048)	0	conv5_block1_0_bn[0][… conv5_block1_3_bn[0][…
conv5_block1_out (Activation)	(None, 6, 6, 2048)	0	conv5_block1_add[0][0]
conv5_block2_1_conv (Conv2D)	(None, 6, 6, 512)	1,049,088	conv5_block1_out[0][0]
conv5_block2_1_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block2_1_conv[0…
conv5_block2_1_relu (Activation)	(None, 6, 6, 512)	0	conv5_block2_1_bn[0][…
conv5_block2_2_conv (Conv2D)	(None, 6, 6, 512)	2,359,808	conv5_block2_1_relu[0…
conv5_block2_2_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block2_2_conv[0…
conv5_block2_2_relu (Activation)	(None, 6, 6, 512)	0	conv5_block2_2_bn[0][…
conv5_block2_3_conv (Conv2D)	(None, 6, 6, 2048)	1,050,624	conv5_block2_2_relu[0…
conv5_block2_3_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block2_3_conv[0…
conv5_block2_add (Add)	(None, 6, 6, 2048)	0	conv5_block1_out[0][0… conv5_block2_3_bn[0][…
conv5_block2_out (Activation)	(None, 6, 6, 2048)	0	conv5_block2_add[0][0]
conv5_block3_1_conv (Conv2D)	(None, 6, 6, 512)	1,049,088	conv5_block2_out[0][0]
conv5_block3_1_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block3_1_conv[0…
conv5_block3_1_relu (Activation)	(None, 6, 6, 512)	0	conv5_block3_1_bn[0][…
conv5_block3_2_conv (Conv2D)	(None, 6, 6, 512)	2,359,808	conv5_block3_1_relu[0…
conv5_block3_2_bn (BatchNormalization)	(None, 6, 6, 512)	2,048	conv5_block3_2_conv[0…
conv5_block3_2_relu (Activation)	(None, 6, 6, 512)	0	conv5_block3_2_bn[0][…
conv5_block3_3_conv (Conv2D)	(None, 6, 6, 2048)	1,050,624	conv5_block3_2_relu[0…
conv5_block3_3_bn (BatchNormalization)	(None, 6, 6, 2048)	8,192	conv5_block3_3_conv[0…
conv5_block3_add (Add)	(None, 6, 6, 2048)	0	conv5_block2_out[0][0… conv5_block3_3_bn[0][…
conv5_block3_out (Activation)	(None, 6, 6, 2048)	0	conv5_block3_add[0][0]

global_average_pooling2d… (GlobalAveragePooling2D)	(None, 2048)	0	conv5_block3_out[0][0]
dense_15 (Dense)	(None, 3)	6,147	global_average_poolin…

Total params: 23,593,859 (90.00 MB)

Trainable params: 23,540,739 (89.80 MB)

Non-trainable params: 53,120 (207.50 KB)

```
Epoch 1/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 90s 601ms/step - accuracy: 0.5289 - loss: 1.5834 - val_accuracy: 0.3394 - val_loss: 1.5446
Epoch 2/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.7126 - loss: 0.7860 - val_accuracy: 0.3193 - val_loss: 2.6742
Epoch 3/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.7041 - loss: 0.7069 - val_accuracy: 0.3230 - val_loss: 6.0924
Epoch 4/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 273ms/step - accuracy: 0.7617 - loss: 0.6094 - val_accuracy: 0.4069 - val_loss: 2.1127
Epoch 5/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.7920 - loss: 0.5196 - val_accuracy: 0.3266 - val_loss: 6.2622
Epoch 6/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 274ms/step - accuracy: 0.7981 - loss: 0.4769 - val_accuracy: 0.4745 - val_loss: 2.8911
Epoch 7/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.8086 - loss: 0.4642 - val_accuracy: 0.3723 - val_loss: 5.6223
Epoch 8/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.8271 - loss: 0.4311 - val_accuracy: 0.4507 - val_loss: 1.6328
Epoch 9/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.7987 - loss: 0.5152 - val_accuracy: 0.3193 - val_loss: 54.9267
Epoch 10/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.7886 - loss: 0.5293 - val_accuracy: 0.4161 - val_loss: 11.0837
Epoch 11/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.8109 - loss: 0.5016 - val_accuracy: 0.4672 - val_loss: 4.0836
Epoch 12/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 273ms/step - accuracy: 0.8303 - loss: 0.4297 - val_accuracy: 0.3577 - val_loss: 6.9628
Epoch 13/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.8145 - loss: 0.4491 - val_accuracy: 0.6405 - val_loss: 2.7252
Epoch 14/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.8408 - loss: 0.3979 - val_accuracy: 0.3248 - val_loss: 46.9976
Epoch 15/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.8319 - loss: 0.4491 - val_accuracy: 0.6661 - val_loss: 0.9821
Epoch 16/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.8488 - loss: 0.3747 - val_accuracy: 0.4872 - val_loss: 2.0542
Epoch 17/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.8318 - loss: 0.3966 - val_accuracy: 0.5675 - val_loss: 0.9875
Epoch 18/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.8588 - loss: 0.3628 - val_accuracy: 0.5511 - val_loss: 1.1371
Epoch 19/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.8508 - loss: 0.3819 - val_accuracy: 0.6588 - val_loss: 1.1857
Epoch 20/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.8334 - loss: 0.4162 - val_accuracy: 0.3741 - val_loss: 7.4450
Epoch 21/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.8616 - loss: 0.3407 - val_accuracy: 0.4799 - val_loss: 2.4875
Epoch 22/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.8640 - loss: 0.3405 - val_accuracy: 0.4051 - val_loss: 2.2070
Epoch 23/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.8528 - loss: 0.3753 - val_accuracy: 0.3558 - val_loss: 2.7819
Epoch 24/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.8661 - loss: 0.3349 - val_accuracy: 0.7153 - val_loss: 1.2740
Epoch 25/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.8842 - loss: 0.3039 - val_accuracy: 0.6953 - val_loss: 1.2185
Epoch 26/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.8788 - loss: 0.3089 - val_accuracy: 0.4672 - val_loss: 3.4655
Epoch 27/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.8586 - loss: 0.3344 - val_accuracy: 0.4234 - val_loss: 2.7029
Epoch 28/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.8721 - loss: 0.3212 - val_accuracy: 0.6734 - val_loss: 0.9428
Epoch 29/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.8771 - loss: 0.3168 - val_accuracy: 0.6150 - val_loss: 1.3832
Epoch 30/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.8413 - loss: 0.4340 - val_accuracy: 0.6150 - val_loss: 1.3718
Epoch 31/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 291ms/step - accuracy: 0.8443 - loss: 0.3781 - val_accuracy: 0.4562 - val_loss: 1.8891
Epoch 32/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.8883 - loss: 0.2881 - val_accuracy: 0.7555 - val_loss: 0.6721
Epoch 33/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.8849 - loss: 0.2964 - val_accuracy: 0.6770 - val_loss: 0.8941
Epoch 34/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.8666 - loss: 0.3361 - val_accuracy: 0.7792 - val_loss: 0.6209
Epoch 35/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.8752 - loss: 0.3204 - val_accuracy: 0.8321 - val_loss: 0.4180
Epoch 36/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.8938 - loss: 0.2967 - val_accuracy: 0.7354 - val_loss: 0.6905
Epoch 37/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.8791 - loss: 0.3052 - val_accuracy: 0.5310 - val_loss: 2.4239
Epoch 38/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.8914 - loss: 0.2748 - val_accuracy: 0.6861 - val_loss: 0.8162
Epoch 39/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.8969 - loss: 0.2687 - val_accuracy: 0.7445 - val_loss: 0.7083
Epoch 40/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.8898 - loss: 0.2719 - val_accuracy: 0.6642 - val_loss: 0.8912
Epoch 41/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.8959 - loss: 0.2698 - val_accuracy: 0.7646 - val_loss: 0.7249
Epoch 42/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9071 - loss: 0.2453 - val_accuracy: 0.7755 - val_loss: 0.8283
Epoch 43/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9028 - loss: 0.2566 - val_accuracy: 0.6588 - val_loss: 0.8315
Epoch 44/1000
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.8969 - loss: 0.2854 - val_accuracy: 0.6843 - val_loss: 1.7349
Epoch 45/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 292ms/step - accuracy: 0.9076 - loss: 0.2423 - val_accuracy: 0.5931 - val_loss: 1.1361
Epoch 46/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9135 - loss: 0.2338 - val_accuracy: 0.6223 - val_loss: 1.1969
Epoch 47/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.8983 - loss: 0.2456 - val_accuracy: 0.6679 - val_loss: 2.5114
Epoch 48/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9142 - loss: 0.2241 - val_accuracy: 0.5292 - val_loss: 2.1031
Epoch 49/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9070 - loss: 0.2590 - val_accuracy: 0.6423 - val_loss: 1.4485
Epoch 50/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9137 - loss: 0.2202 - val_accuracy: 0.6588 - val_loss: 1.4289
Epoch 51/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9033 - loss: 0.2701 - val_accuracy: 0.4781 - val_loss: 6.9530
Epoch 52/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.8899 - loss: 0.2754 - val_accuracy: 0.5055 - val_loss: 2.5120
Epoch 53/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.8951 - loss: 0.2774 - val_accuracy: 0.7391 - val_loss: 1.0292
Epoch 54/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9087 - loss: 0.2363 - val_accuracy: 0.4252 - val_loss: 4.2968
Epoch 55/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9149 - loss: 0.2281 - val_accuracy: 0.6734 - val_loss: 1.0850
Epoch 56/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 290ms/step - accuracy: 0.9012 - loss: 0.2515 - val_accuracy: 0.5547 - val_loss: 1.6337
Epoch 57/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9311 - loss: 0.1879 - val_accuracy: 0.8102 - val_loss: 0.6643
Epoch 58/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9354 - loss: 0.1976 - val_accuracy: 0.6150 - val_loss: 1.4199
Epoch 59/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 272ms/step - accuracy: 0.9180 - loss: 0.2284 - val_accuracy: 0.6223 - val_loss: 3.8821
Epoch 60/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9261 - loss: 0.1953 - val_accuracy: 0.5310 - val_loss: 1.5855
Epoch 61/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 288ms/step - accuracy: 0.9065 - loss: 0.2203 - val_accuracy: 0.5383 - val_loss: 1.4524
Epoch 62/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9117 - loss: 0.2275 - val_accuracy: 0.8066 - val_loss: 0.5290
Epoch 63/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 272ms/step - accuracy: 0.9073 - loss: 0.2235 - val_accuracy: 0.8303 - val_loss: 0.4331
Epoch 64/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9027 - loss: 0.2525 - val_accuracy: 0.7682 - val_loss: 0.5795
Epoch 65/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.8994 - loss: 0.2652 - val_accuracy: 0.4964 - val_loss: 11.1474
Epoch 66/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9244 - loss: 0.2164 - val_accuracy: 0.7993 - val_loss: 0.7366
Epoch 67/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9010 - loss: 0.2486 - val_accuracy: 0.6606 - val_loss: 1.2240
Epoch 68/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.8845 - loss: 0.3183 - val_accuracy: 0.7135 - val_loss: 0.8957
Epoch 69/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.8921 - loss: 0.2496 - val_accuracy: 0.7153 - val_loss: 1.3511
Epoch 70/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9241 - loss: 0.1988 - val_accuracy: 0.6953 - val_loss: 1.7173
Epoch 71/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.8956 - loss: 0.2638 - val_accuracy: 0.7007 - val_loss: 1.1069
Epoch 72/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9118 - loss: 0.2405 - val_accuracy: 0.6989 - val_loss: 1.0738
Epoch 73/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9336 - loss: 0.1777 - val_accuracy: 0.4964 - val_loss: 2.8114
Epoch 74/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9406 - loss: 0.1749 - val_accuracy: 0.6460 - val_loss: 1.0943
Epoch 75/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9308 - loss: 0.1747 - val_accuracy: 0.6387 - val_loss: 1.6726
Epoch 76/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9187 - loss: 0.1949 - val_accuracy: 0.6223 - val_loss: 1.6357
Epoch 77/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 273ms/step - accuracy: 0.9155 - loss: 0.2170 - val_accuracy: 0.7993 - val_loss: 0.6351
Epoch 78/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9417 - loss: 0.1551 - val_accuracy: 0.8084 - val_loss: 0.5990
Epoch 79/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9242 - loss: 0.1992 - val_accuracy: 0.5931 - val_loss: 2.2344
Epoch 80/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9238 - loss: 0.1915 - val_accuracy: 0.6314 - val_loss: 1.5326
Epoch 81/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9089 - loss: 0.2224 - val_accuracy: 0.8303 - val_loss: 0.5855
Epoch 82/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9307 - loss: 0.1720 - val_accuracy: 0.6296 - val_loss: 1.4037
Epoch 83/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9261 - loss: 0.1968 - val_accuracy: 0.7628 - val_loss: 0.6769
Epoch 84/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9165 - loss: 0.2185 - val_accuracy: 0.7865 - val_loss: 0.5880
Epoch 85/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9293 - loss: 0.1800 - val_accuracy: 0.7062 - val_loss: 0.9675
Epoch 86/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9032 - loss: 0.2404 - val_accuracy: 0.8777 - val_loss: 0.6718
Epoch 87/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9226 - loss: 0.1971 - val_accuracy: 0.8869 - val_loss: 0.3489
```

```
Epoch 88/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9325 - loss: 0.1799 - val_accuracy: 0.7993 - val_loss: 0.6160
Epoch 89/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9299 - loss: 0.1739 - val_accuracy: 0.6880 - val_loss: 0.7623
Epoch 90/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9290 - loss: 0.1848 - val_accuracy: 0.8029 - val_loss: 0.5021
Epoch 91/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9312 - loss: 0.1902 - val_accuracy: 0.7682 - val_loss: 0.7421
Epoch 92/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9409 - loss: 0.1705 - val_accuracy: 0.7536 - val_loss: 1.3264
Epoch 93/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9374 - loss: 0.1688 - val_accuracy: 0.7737 - val_loss: 0.6067
Epoch 94/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9319 - loss: 0.1811 - val_accuracy: 0.8704 - val_loss: 0.4154
Epoch 95/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9286 - loss: 0.1851 - val_accuracy: 0.8905 - val_loss: 0.3947
Epoch 96/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9540 - loss: 0.1344 - val_accuracy: 0.7664 - val_loss: 0.5190
Epoch 97/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9223 - loss: 0.1823 - val_accuracy: 0.8796 - val_loss: 0.4048
Epoch 98/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9389 - loss: 0.1488 - val_accuracy: 0.6953 - val_loss: 1.0014
Epoch 99/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9483 - loss: 0.1454 - val_accuracy: 0.8047 - val_loss: 0.7214
Epoch 100/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 291ms/step - accuracy: 0.9329 - loss: 0.1727 - val_accuracy: 0.8869 - val_loss: 0.3172
Epoch 101/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9351 - loss: 0.1647 - val_accuracy: 0.7792 - val_loss: 0.9072
Epoch 102/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9496 - loss: 0.1586 - val_accuracy: 0.8412 - val_loss: 0.4893
Epoch 103/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9489 - loss: 0.1439 - val_accuracy: 0.7956 - val_loss: 0.8066
Epoch 104/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9166 - loss: 0.2044 - val_accuracy: 0.8449 - val_loss: 0.5064
Epoch 105/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9333 - loss: 0.1917 - val_accuracy: 0.6405 - val_loss: 3.6607
Epoch 106/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9308 - loss: 0.2052 - val_accuracy: 0.8905 - val_loss: 0.3164
Epoch 107/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9275 - loss: 0.1814 - val_accuracy: 0.4416 - val_loss: 3.3092
Epoch 108/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 273ms/step - accuracy: 0.9260 - loss: 0.2070 - val_accuracy: 0.8449 - val_loss: 0.4979
Epoch 109/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9365 - loss: 0.1523 - val_accuracy: 0.7336 - val_loss: 1.0768
Epoch 110/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9444 - loss: 0.1350 - val_accuracy: 0.8011 - val_loss: 0.7234
Epoch 111/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9374 - loss: 0.1615 - val_accuracy: 0.6569 - val_loss: 0.9856
Epoch 112/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9524 - loss: 0.1310 - val_accuracy: 0.7974 - val_loss: 0.8463
Epoch 113/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9407 - loss: 0.1508 - val_accuracy: 0.7810 - val_loss: 0.6010
Epoch 114/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9401 - loss: 0.1425 - val_accuracy: 0.7135 - val_loss: 1.2985
Epoch 115/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9499 - loss: 0.1262 - val_accuracy: 0.6168 - val_loss: 3.5282
Epoch 116/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9270 - loss: 0.1897 - val_accuracy: 0.6569 - val_loss: 15.6634
Epoch 117/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 288ms/step - accuracy: 0.9425 - loss: 0.1430 - val_accuracy: 0.9069 - val_loss: 0.3620
Epoch 118/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9540 - loss: 0.1281 - val_accuracy: 0.5876 - val_loss: 5.3982
Epoch 119/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9436 - loss: 0.1900 - val_accuracy: 0.8066 - val_loss: 0.5892
Epoch 120/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9386 - loss: 0.1440 - val_accuracy: 0.6131 - val_loss: 2.4282
Epoch 121/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9635 - loss: 0.1175 - val_accuracy: 0.8996 - val_loss: 0.3220
Epoch 122/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9540 - loss: 0.1457 - val_accuracy: 0.6861 - val_loss: 1.6225
Epoch 123/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 289ms/step - accuracy: 0.9188 - loss: 0.2219 - val_accuracy: 0.6880 - val_loss: 1.3454
Epoch 124/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9429 - loss: 0.1581 - val_accuracy: 0.8485 - val_loss: 0.4361
Epoch 125/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9447 - loss: 0.1252 - val_accuracy: 0.9124 - val_loss: 0.2860
Epoch 126/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9455 - loss: 0.1232 - val_accuracy: 0.5055 - val_loss: 3.1589
Epoch 127/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 21s 299ms/step - accuracy: 0.9465 - loss: 0.1533 - val_accuracy: 0.7609 - val_loss: 1.0200
Epoch 128/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9459 - loss: 0.1552 - val_accuracy: 0.9197 - val_loss: 0.2382
Epoch 129/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9468 - loss: 0.1295 - val_accuracy: 0.6770 - val_loss: 1.3865
Epoch 130/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9528 - loss: 0.1144 - val_accuracy: 0.9106 - val_loss: 0.2850
Epoch 131/1000
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 288ms/step - accuracy: 0.9436 - loss: 0.1466 - val_accuracy: 0.8577 - val_loss: 0.3896
Epoch 132/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 21s 305ms/step - accuracy: 0.9545 - loss: 0.1151 - val_accuracy: 0.8978 - val_loss: 0.3384
Epoch 133/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 21s 300ms/step - accuracy: 0.9312 - loss: 0.1809 - val_accuracy: 0.6332 - val_loss: 2.7797
Epoch 134/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 292ms/step - accuracy: 0.9638 - loss: 0.0951 - val_accuracy: 0.8832 - val_loss: 0.4073
Epoch 135/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9645 - loss: 0.0961 - val_accuracy: 0.8248 - val_loss: 0.5505
Epoch 136/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9616 - loss: 0.1173 - val_accuracy: 0.9033 - val_loss: 0.2947
Epoch 137/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9503 - loss: 0.1290 - val_accuracy: 0.9215 - val_loss: 0.2427
Epoch 138/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 294ms/step - accuracy: 0.9605 - loss: 0.1137 - val_accuracy: 0.8431 - val_loss: 0.4576
Epoch 139/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9494 - loss: 0.1272 - val_accuracy: 0.8595 - val_loss: 0.4003
Epoch 140/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 21s 303ms/step - accuracy: 0.9524 - loss: 0.1091 - val_accuracy: 0.8066 - val_loss: 0.5461
Epoch 141/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 21s 297ms/step - accuracy: 0.9550 - loss: 0.1236 - val_accuracy: 0.9106 - val_loss: 0.2749
Epoch 142/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9621 - loss: 0.1043 - val_accuracy: 0.7536 - val_loss: 1.0327
Epoch 143/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9526 - loss: 0.1112 - val_accuracy: 0.8942 - val_loss: 0.3720
Epoch 144/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9664 - loss: 0.0945 - val_accuracy: 0.9179 - val_loss: 0.2458
Epoch 145/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9500 - loss: 0.1254 - val_accuracy: 0.7701 - val_loss: 0.7130
Epoch 146/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9536 - loss: 0.1234 - val_accuracy: 0.7920 - val_loss: 0.8242
Epoch 147/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 291ms/step - accuracy: 0.9580 - loss: 0.1165 - val_accuracy: 0.5584 - val_loss: 2.4562
Epoch 148/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9476 - loss: 0.1388 - val_accuracy: 0.8449 - val_loss: 0.4224
Epoch 149/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 289ms/step - accuracy: 0.9517 - loss: 0.1173 - val_accuracy: 0.6259 - val_loss: 8.8477
Epoch 150/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 290ms/step - accuracy: 0.9612 - loss: 0.1093 - val_accuracy: 0.9106 - val_loss: 0.3128
Epoch 151/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9627 - loss: 0.1016 - val_accuracy: 0.8704 - val_loss: 0.3619
Epoch 152/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9600 - loss: 0.1080 - val_accuracy: 0.9088 - val_loss: 0.3286
Epoch 153/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 289ms/step - accuracy: 0.9642 - loss: 0.1017 - val_accuracy: 0.8613 - val_loss: 0.3731
Epoch 154/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9630 - loss: 0.0945 - val_accuracy: 0.7974 - val_loss: 0.5564
Epoch 155/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9523 - loss: 0.1335 - val_accuracy: 0.6588 - val_loss: 1.7511
Epoch 156/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9672 - loss: 0.0997 - val_accuracy: 0.8887 - val_loss: 0.3993
Epoch 157/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9645 - loss: 0.1102 - val_accuracy: 0.8850 - val_loss: 0.4653
Epoch 158/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9498 - loss: 0.1299 - val_accuracy: 0.6971 - val_loss: 1.4661
Epoch 159/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9662 - loss: 0.0906 - val_accuracy: 0.8029 - val_loss: 0.6132
Epoch 160/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9566 - loss: 0.1079 - val_accuracy: 0.9252 - val_loss: 0.2050
Epoch 161/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9721 - loss: 0.0756 - val_accuracy: 0.8869 - val_loss: 0.3585
Epoch 162/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9701 - loss: 0.0777 - val_accuracy: 0.8047 - val_loss: 0.7372
Epoch 163/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9612 - loss: 0.1025 - val_accuracy: 0.7810 - val_loss: 0.7094
Epoch 164/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9658 - loss: 0.0975 - val_accuracy: 0.8777 - val_loss: 0.3312
Epoch 165/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9516 - loss: 0.1526 - val_accuracy: 0.4872 - val_loss: 4.0078
Epoch 166/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 289ms/step - accuracy: 0.9645 - loss: 0.1017 - val_accuracy: 0.6004 - val_loss: 5.2025
Epoch 167/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9600 - loss: 0.1031 - val_accuracy: 0.8540 - val_loss: 0.4765
Epoch 168/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 290ms/step - accuracy: 0.9693 - loss: 0.0858 - val_accuracy: 0.9051 - val_loss: 0.3245
Epoch 169/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9561 - loss: 0.1105 - val_accuracy: 0.8923 - val_loss: 0.3206
Epoch 170/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9588 - loss: 0.1011 - val_accuracy: 0.7993 - val_loss: 0.7903
Epoch 171/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9599 - loss: 0.1107 - val_accuracy: 0.9197 - val_loss: 0.3023
Epoch 172/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9686 - loss: 0.0884 - val_accuracy: 0.8394 - val_loss: 0.7319
Epoch 173/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9782 - loss: 0.0696 - val_accuracy: 0.8120 - val_loss: 0.8374
Epoch 174/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9540 - loss: 0.1239 - val_accuracy: 0.8923 - val_loss: 0.3368
```

```
Epoch 175/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9647 - loss: 0.0853 - val_accuracy: 0.9288 - val_loss: 0.2305
Epoch 176/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9772 - loss: 0.0676 - val_accuracy: 0.8431 - val_loss: 0.4386
Epoch 177/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9577 - loss: 0.0863 - val_accuracy: 0.8102 - val_loss: 0.7400
Epoch 178/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9702 - loss: 0.0854 - val_accuracy: 0.8668 - val_loss: 0.5328
Epoch 179/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 288ms/step - accuracy: 0.9625 - loss: 0.1029 - val_accuracy: 0.9051 - val_loss: 0.2928
Epoch 180/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9664 - loss: 0.0872 - val_accuracy: 0.9051 - val_loss: 0.3119
Epoch 181/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9733 - loss: 0.0652 - val_accuracy: 0.8193 - val_loss: 0.5337
Epoch 182/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9574 - loss: 0.1102 - val_accuracy: 0.9088 - val_loss: 0.3624
Epoch 183/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9706 - loss: 0.0713 - val_accuracy: 0.7372 - val_loss: 1.3585
Epoch 184/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 21s 300ms/step - accuracy: 0.9730 - loss: 0.0729 - val_accuracy: 0.9361 - val_loss: 0.2067
Epoch 185/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9808 - loss: 0.0588 - val_accuracy: 0.8905 - val_loss: 0.3217
Epoch 186/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 274ms/step - accuracy: 0.9679 - loss: 0.0986 - val_accuracy: 0.7372 - val_loss: 1.0576
Epoch 187/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9636 - loss: 0.1024 - val_accuracy: 0.8960 - val_loss: 0.3062
Epoch 188/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9733 - loss: 0.0632 - val_accuracy: 0.7609 - val_loss: 1.2167
Epoch 189/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9685 - loss: 0.0850 - val_accuracy: 0.8303 - val_loss: 0.4747
Epoch 190/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9727 - loss: 0.0641 - val_accuracy: 0.7007 - val_loss: 1.7465
Epoch 191/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9693 - loss: 0.0865 - val_accuracy: 0.9398 - val_loss: 0.2220
Epoch 192/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9650 - loss: 0.0874 - val_accuracy: 0.8978 - val_loss: 0.3391
Epoch 193/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9696 - loss: 0.0751 - val_accuracy: 0.9288 - val_loss: 0.2399
Epoch 194/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9771 - loss: 0.0615 - val_accuracy: 0.8394 - val_loss: 0.6191
Epoch 195/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9778 - loss: 0.0671 - val_accuracy: 0.6277 - val_loss: 2.1870
Epoch 196/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9599 - loss: 0.1154 - val_accuracy: 0.9033 - val_loss: 0.3189
Epoch 197/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9634 - loss: 0.0911 - val_accuracy: 0.8869 - val_loss: 0.3013
Epoch 198/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 290ms/step - accuracy: 0.9744 - loss: 0.0634 - val_accuracy: 0.9197 - val_loss: 0.2572
Epoch 199/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9733 - loss: 0.0719 - val_accuracy: 0.8412 - val_loss: 0.7220
Epoch 200/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9754 - loss: 0.0702 - val_accuracy: 0.9015 - val_loss: 0.3528
Epoch 201/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9770 - loss: 0.0687 - val_accuracy: 0.9015 - val_loss: 0.3323
Epoch 202/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9683 - loss: 0.0861 - val_accuracy: 0.4672 - val_loss: 4.3973
Epoch 203/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9500 - loss: 0.1432 - val_accuracy: 0.7391 - val_loss: 1.2513
Epoch 204/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.8842 - loss: 0.3092 - val_accuracy: 0.5237 - val_loss: 5.0238
Epoch 205/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 279ms/step - accuracy: 0.9295 - loss: 0.2003 - val_accuracy: 0.7318 - val_loss: 1.0348
Epoch 206/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9574 - loss: 0.1101 - val_accuracy: 0.8942 - val_loss: 0.3409
Epoch 207/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9685 - loss: 0.0795 - val_accuracy: 0.8540 - val_loss: 0.5648
Epoch 208/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9776 - loss: 0.0691 - val_accuracy: 0.9343 - val_loss: 0.1829
Epoch 209/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9657 - loss: 0.0841 - val_accuracy: 0.9033 - val_loss: 0.2934
Epoch 210/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9745 - loss: 0.0828 - val_accuracy: 0.8175 - val_loss: 0.7650
Epoch 211/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9784 - loss: 0.0551 - val_accuracy: 0.8777 - val_loss: 0.3833
Epoch 212/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9718 - loss: 0.0716 - val_accuracy: 0.9088 - val_loss: 0.2645
Epoch 213/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9735 - loss: 0.0690 - val_accuracy: 0.9161 - val_loss: 0.2695
Epoch 214/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9754 - loss: 0.0716 - val_accuracy: 0.8704 - val_loss: 0.5601
Epoch 215/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9732 - loss: 0.0693 - val_accuracy: 0.8394 - val_loss: 0.5465
Epoch 216/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 272ms/step - accuracy: 0.9659 - loss: 0.0900 - val_accuracy: 0.9252 - val_loss: 0.2244
Epoch 217/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 289ms/step - accuracy: 0.9833 - loss: 0.0507 - val_accuracy: 0.8212 - val_loss: 0.7439
Epoch 218/1000
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9795 - loss: 0.0500 - val_accuracy: 0.9288 - val_loss: 0.2088
Epoch 219/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9638 - loss: 0.0905 - val_accuracy: 0.8741 - val_loss: 0.4310
Epoch 220/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9739 - loss: 0.0722 - val_accuracy: 0.8668 - val_loss: 0.3860
Epoch 221/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9732 - loss: 0.0540 - val_accuracy: 0.9434 - val_loss: 0.2040
Epoch 222/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 271ms/step - accuracy: 0.9856 - loss: 0.0431 - val_accuracy: 0.9215 - val_loss: 0.3053
Epoch 223/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9857 - loss: 0.0378 - val_accuracy: 0.8960 - val_loss: 0.3775
Epoch 224/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9803 - loss: 0.0559 - val_accuracy: 0.7993 - val_loss: 0.6439
Epoch 225/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9773 - loss: 0.0642 - val_accuracy: 0.8905 - val_loss: 0.4789
Epoch 226/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9822 - loss: 0.0581 - val_accuracy: 0.8777 - val_loss: 0.3612
Epoch 227/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9708 - loss: 0.0700 - val_accuracy: 0.9179 - val_loss: 0.2537
Epoch 228/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9702 - loss: 0.0716 - val_accuracy: 0.7792 - val_loss: 1.4878
Epoch 229/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9378 - loss: 0.1986 - val_accuracy: 0.3777 - val_loss: 54.5292
Epoch 230/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9252 - loss: 0.2041 - val_accuracy: 0.7026 - val_loss: 2.2358
Epoch 231/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9489 - loss: 0.1343 - val_accuracy: 0.9051 - val_loss: 0.2780
Epoch 232/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9636 - loss: 0.1017 - val_accuracy: 0.8741 - val_loss: 0.4089
Epoch 233/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9692 - loss: 0.0913 - val_accuracy: 0.8905 - val_loss: 0.3386
Epoch 234/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9704 - loss: 0.0845 - val_accuracy: 0.9142 - val_loss: 0.2629
Epoch 235/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9671 - loss: 0.0863 - val_accuracy: 0.8942 - val_loss: 0.2937
Epoch 236/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9753 - loss: 0.0712 - val_accuracy: 0.8796 - val_loss: 0.3158
Epoch 237/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9662 - loss: 0.0801 - val_accuracy: 0.9124 - val_loss: 0.2297
Epoch 238/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9839 - loss: 0.0472 - val_accuracy: 0.9215 - val_loss: 0.2272
Epoch 239/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9623 - loss: 0.0848 - val_accuracy: 0.9197 - val_loss: 0.2995
Epoch 240/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9725 - loss: 0.0658 - val_accuracy: 0.8540 - val_loss: 0.5526
Epoch 241/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 274ms/step - accuracy: 0.9809 - loss: 0.0664 - val_accuracy: 0.8485 - val_loss: 0.6763
Epoch 242/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9779 - loss: 0.0547 - val_accuracy: 0.9179 - val_loss: 0.2793
Epoch 243/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9799 - loss: 0.0490 - val_accuracy: 0.9325 - val_loss: 0.2564
Epoch 244/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9776 - loss: 0.0543 - val_accuracy: 0.9088 - val_loss: 0.3702
Epoch 245/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9825 - loss: 0.0472 - val_accuracy: 0.9197 - val_loss: 0.2209
Epoch 246/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9725 - loss: 0.0655 - val_accuracy: 0.8704 - val_loss: 0.5415
Epoch 247/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9858 - loss: 0.0454 - val_accuracy: 0.9288 - val_loss: 0.2952
Epoch 248/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9814 - loss: 0.0524 - val_accuracy: 0.9069 - val_loss: 0.3486
Epoch 249/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9838 - loss: 0.0586 - val_accuracy: 0.7464 - val_loss: 1.1082
Epoch 250/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9835 - loss: 0.0549 - val_accuracy: 0.9398 - val_loss: 0.2494
Epoch 251/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9861 - loss: 0.0411 - val_accuracy: 0.8650 - val_loss: 0.4846
Epoch 252/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9810 - loss: 0.0516 - val_accuracy: 0.8631 - val_loss: 0.5155
Epoch 253/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9719 - loss: 0.0652 - val_accuracy: 0.9179 - val_loss: 0.2777
Epoch 254/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9695 - loss: 0.0761 - val_accuracy: 0.9161 - val_loss: 0.2920
Epoch 255/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9863 - loss: 0.0372 - val_accuracy: 0.8613 - val_loss: 0.4281
Epoch 256/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 21s 285ms/step - accuracy: 0.9807 - loss: 0.0445 - val_accuracy: 0.8577 - val_loss: 0.5810
Epoch 257/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9705 - loss: 0.0730 - val_accuracy: 0.8850 - val_loss: 0.5595
Epoch 258/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9727 - loss: 0.0652 - val_accuracy: 0.8303 - val_loss: 0.7183
Epoch 259/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9798 - loss: 0.0540 - val_accuracy: 0.9051 - val_loss: 0.3360
Epoch 260/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9855 - loss: 0.0401 - val_accuracy: 0.9380 - val_loss: 0.2572
Epoch 261/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9835 - loss: 0.0395 - val_accuracy: 0.9069 - val_loss: 0.3170
```

```
Epoch 262/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9861 - loss: 0.0429 - val_accuracy: 0.7208 - val_loss: 1.7278
Epoch 263/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9842 - loss: 0.0489 - val_accuracy: 0.9015 - val_loss: 0.4481
Epoch 264/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9833 - loss: 0.0427 - val_accuracy: 0.8942 - val_loss: 0.4583
Epoch 265/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9827 - loss: 0.0520 - val_accuracy: 0.8942 - val_loss: 0.4134
Epoch 266/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9856 - loss: 0.0380 - val_accuracy: 0.9288 - val_loss: 0.2378
Epoch 267/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9868 - loss: 0.0377 - val_accuracy: 0.9252 - val_loss: 0.2375
Epoch 268/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9767 - loss: 0.0662 - val_accuracy: 0.9015 - val_loss: 0.3705
Epoch 269/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9800 - loss: 0.0648 - val_accuracy: 0.5456 - val_loss: 9.5161
Epoch 270/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9488 - loss: 0.1179 - val_accuracy: 0.8339 - val_loss: 0.7639
Epoch 271/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9839 - loss: 0.0524 - val_accuracy: 0.9343 - val_loss: 0.2441
Epoch 272/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 281ms/step - accuracy: 0.9787 - loss: 0.0665 - val_accuracy: 0.9088 - val_loss: 0.3164
Epoch 273/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9770 - loss: 0.0600 - val_accuracy: 0.9252 - val_loss: 0.3061
Epoch 274/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9867 - loss: 0.0514 - val_accuracy: 0.6113 - val_loss: 2.5478
Epoch 275/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9670 - loss: 0.0960 - val_accuracy: 0.5639 - val_loss: 1.9041
Epoch 276/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 291ms/step - accuracy: 0.9541 - loss: 0.1018 - val_accuracy: 0.8558 - val_loss: 1.5020
Epoch 277/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9431 - loss: 0.1615 - val_accuracy: 0.7847 - val_loss: 0.9783
Epoch 278/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9590 - loss: 0.1387 - val_accuracy: 0.8978 - val_loss: 0.3897
Epoch 279/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 292ms/step - accuracy: 0.9728 - loss: 0.0688 - val_accuracy: 0.6314 - val_loss: 3.6561
Epoch 280/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9703 - loss: 0.0802 - val_accuracy: 0.8339 - val_loss: 0.7085
Epoch 281/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9787 - loss: 0.0630 - val_accuracy: 0.6150 - val_loss: 2.7295
Epoch 282/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9844 - loss: 0.0429 - val_accuracy: 0.8248 - val_loss: 0.7251
Epoch 283/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9539 - loss: 0.1983 - val_accuracy: 0.3668 - val_loss: 6.7118
Epoch 284/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9145 - loss: 0.2459 - val_accuracy: 0.3285 - val_loss: 23.2734
Epoch 285/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.8392 - loss: 0.4676 - val_accuracy: 0.4635 - val_loss: 19.5020
Epoch 286/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9058 - loss: 0.2522 - val_accuracy: 0.8978 - val_loss: 0.2975
Epoch 287/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9427 - loss: 0.1624 - val_accuracy: 0.8960 - val_loss: 0.3579
Epoch 288/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9392 - loss: 0.1820 - val_accuracy: 0.9288 - val_loss: 0.2295
Epoch 289/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9556 - loss: 0.1270 - val_accuracy: 0.8759 - val_loss: 0.4070
Epoch 290/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9723 - loss: 0.0942 - val_accuracy: 0.9197 - val_loss: 0.2166
Epoch 291/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9497 - loss: 0.1352 - val_accuracy: 0.8960 - val_loss: 0.3368
Epoch 292/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9674 - loss: 0.0874 - val_accuracy: 0.8650 - val_loss: 0.4618
Epoch 293/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9728 - loss: 0.0789 - val_accuracy: 0.9343 - val_loss: 0.1739
Epoch 294/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9714 - loss: 0.0790 - val_accuracy: 0.8723 - val_loss: 0.4211
Epoch 295/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 289ms/step - accuracy: 0.9778 - loss: 0.0528 - val_accuracy: 0.8905 - val_loss: 0.4446
Epoch 296/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 272ms/step - accuracy: 0.9784 - loss: 0.0557 - val_accuracy: 0.9288 - val_loss: 0.2494
Epoch 297/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9720 - loss: 0.0851 - val_accuracy: 0.8741 - val_loss: 0.3750
Epoch 298/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9768 - loss: 0.0689 - val_accuracy: 0.8960 - val_loss: 0.3843
Epoch 299/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 274ms/step - accuracy: 0.9771 - loss: 0.0662 - val_accuracy: 0.9270 - val_loss: 0.2474
Epoch 300/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9784 - loss: 0.0567 - val_accuracy: 0.9398 - val_loss: 0.2318
Epoch 301/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9810 - loss: 0.0522 - val_accuracy: 0.8449 - val_loss: 0.5330
Epoch 302/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 288ms/step - accuracy: 0.9752 - loss: 0.0697 - val_accuracy: 0.8558 - val_loss: 0.5904
Epoch 303/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9851 - loss: 0.0496 - val_accuracy: 0.7993 - val_loss: 0.7193
Epoch 304/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9722 - loss: 0.0893 - val_accuracy: 0.9124 - val_loss: 0.3151
Epoch 305/1000
```

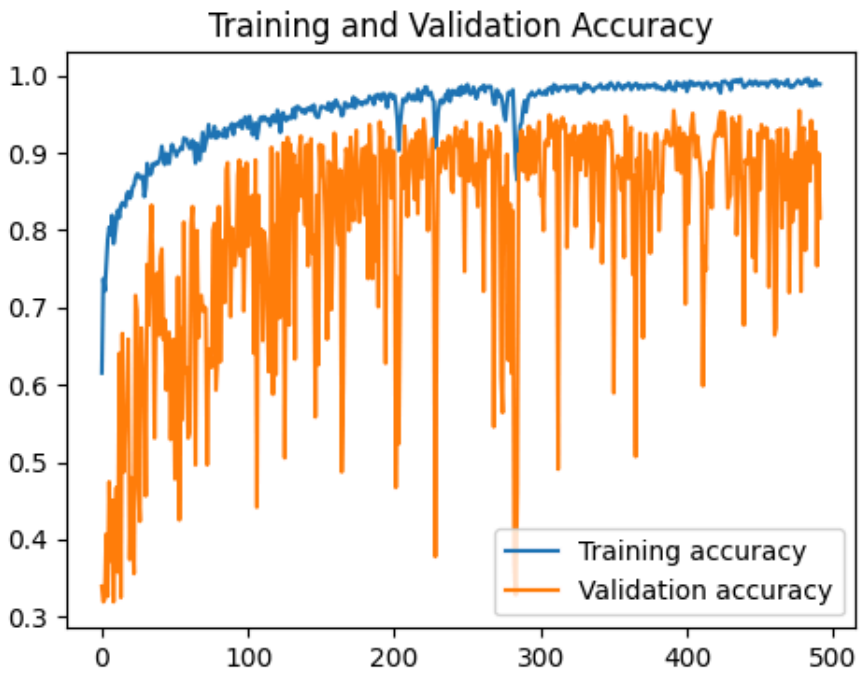
```
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9784 - loss: 0.0516 - val_accuracy: 0.9161 - val_loss: 0.3115
Epoch 306/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 288ms/step - accuracy: 0.9868 - loss: 0.0449 - val_accuracy: 0.9088 - val_loss: 0.3418
Epoch 307/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9820 - loss: 0.0429 - val_accuracy: 0.9489 - val_loss: 0.2253
Epoch 308/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 273ms/step - accuracy: 0.9881 - loss: 0.0376 - val_accuracy: 0.9197 - val_loss: 0.2634
Epoch 309/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9735 - loss: 0.0641 - val_accuracy: 0.9270 - val_loss: 0.2548
Epoch 310/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9888 - loss: 0.0306 - val_accuracy: 0.9380 - val_loss: 0.1907
Epoch 311/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9836 - loss: 0.0457 - val_accuracy: 0.8777 - val_loss: 0.6267
Epoch 312/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9793 - loss: 0.0585 - val_accuracy: 0.9416 - val_loss: 0.2458
Epoch 313/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9823 - loss: 0.0485 - val_accuracy: 0.4909 - val_loss: 5.0118
Epoch 314/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 290ms/step - accuracy: 0.9845 - loss: 0.0427 - val_accuracy: 0.9325 - val_loss: 0.2928
Epoch 315/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9802 - loss: 0.0466 - val_accuracy: 0.9307 - val_loss: 0.3587
Epoch 316/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 272ms/step - accuracy: 0.9877 - loss: 0.0338 - val_accuracy: 0.9453 - val_loss: 0.1999
Epoch 317/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9828 - loss: 0.0394 - val_accuracy: 0.9361 - val_loss: 0.2478
Epoch 318/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9903 - loss: 0.0312 - val_accuracy: 0.9361 - val_loss: 0.3180
Epoch 319/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 273ms/step - accuracy: 0.9770 - loss: 0.0567 - val_accuracy: 0.7774 - val_loss: 0.8742
Epoch 320/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9829 - loss: 0.0551 - val_accuracy: 0.8577 - val_loss: 0.7212
Epoch 321/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9883 - loss: 0.0462 - val_accuracy: 0.9234 - val_loss: 0.3028
Epoch 322/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9862 - loss: 0.0342 - val_accuracy: 0.8978 - val_loss: 0.3916
Epoch 323/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9883 - loss: 0.0280 - val_accuracy: 0.9179 - val_loss: 0.2768
Epoch 324/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9827 - loss: 0.0559 - val_accuracy: 0.9343 - val_loss: 0.2520
Epoch 325/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9855 - loss: 0.0343 - val_accuracy: 0.8047 - val_loss: 0.7096
Epoch 326/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 288ms/step - accuracy: 0.9903 - loss: 0.0303 - val_accuracy: 0.8796 - val_loss: 0.6848
Epoch 327/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9773 - loss: 0.0725 - val_accuracy: 0.9215 - val_loss: 0.2465
Epoch 328/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 291ms/step - accuracy: 0.9765 - loss: 0.0650 - val_accuracy: 0.8777 - val_loss: 0.4349
Epoch 329/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9832 - loss: 0.0486 - val_accuracy: 0.9234 - val_loss: 0.2579
Epoch 330/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9773 - loss: 0.0493 - val_accuracy: 0.8942 - val_loss: 0.3711
Epoch 331/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 290ms/step - accuracy: 0.9898 - loss: 0.0371 - val_accuracy: 0.9343 - val_loss: 0.2035
Epoch 332/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9852 - loss: 0.0352 - val_accuracy: 0.8704 - val_loss: 0.5515
Epoch 333/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9857 - loss: 0.0397 - val_accuracy: 0.8996 - val_loss: 0.3076
Epoch 334/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9887 - loss: 0.0312 - val_accuracy: 0.9380 - val_loss: 0.2295
Epoch 335/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9853 - loss: 0.0449 - val_accuracy: 0.9361 - val_loss: 0.1984
Epoch 336/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9875 - loss: 0.0372 - val_accuracy: 0.7774 - val_loss: 0.9970
Epoch 337/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 273ms/step - accuracy: 0.9815 - loss: 0.0501 - val_accuracy: 0.8029 - val_loss: 0.5167
Epoch 338/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9787 - loss: 0.0611 - val_accuracy: 0.9361 - val_loss: 0.2239
Epoch 339/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9829 - loss: 0.0470 - val_accuracy: 0.8796 - val_loss: 0.5886
Epoch 340/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9864 - loss: 0.0363 - val_accuracy: 0.9197 - val_loss: 0.2822
Epoch 341/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9910 - loss: 0.0307 - val_accuracy: 0.8120 - val_loss: 0.9270
Epoch 342/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9887 - loss: 0.0409 - val_accuracy: 0.9343 - val_loss: 0.2094
Epoch 343/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9840 - loss: 0.0506 - val_accuracy: 0.7573 - val_loss: 1.4920
Epoch 344/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9861 - loss: 0.0478 - val_accuracy: 0.8558 - val_loss: 0.5825
Epoch 345/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9793 - loss: 0.0597 - val_accuracy: 0.9398 - val_loss: 0.2556
Epoch 346/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9879 - loss: 0.0363 - val_accuracy: 0.9398 - val_loss: 0.2444
Epoch 347/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 288ms/step - accuracy: 0.9823 - loss: 0.0534 - val_accuracy: 0.9398 - val_loss: 0.1771
Epoch 348/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9909 - loss: 0.0265 - val_accuracy: 0.9234 - val_loss: 0.2960
```

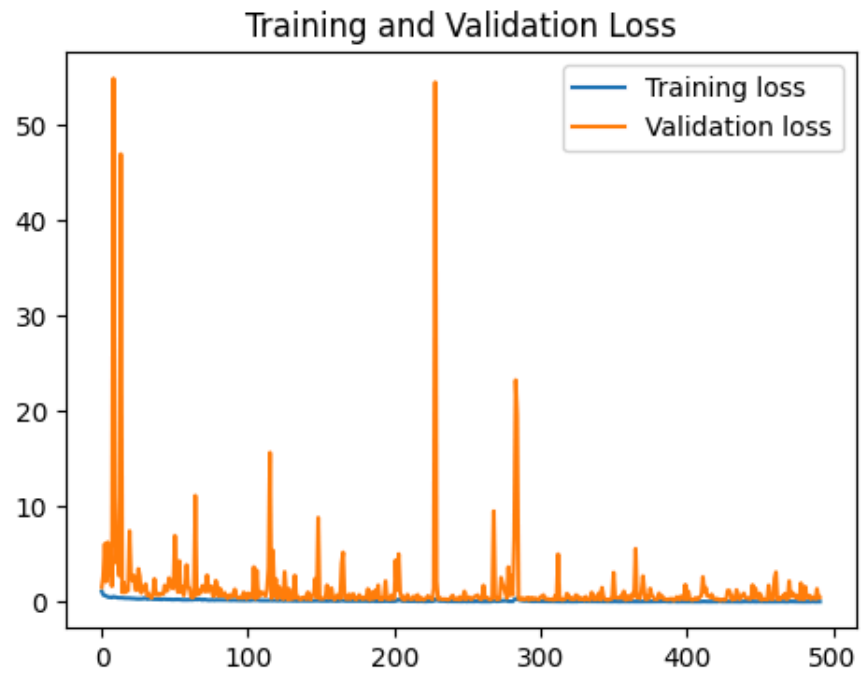
```
Epoch 349/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9899 - loss: 0.0400 - val_accuracy: 0.9288 - val_loss: 0.2732
Epoch 350/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9860 - loss: 0.0401 - val_accuracy: 0.9033 - val_loss: 0.3813
Epoch 351/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 290ms/step - accuracy: 0.9872 - loss: 0.0433 - val_accuracy: 0.5894 - val_loss: 3.0741
Epoch 352/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9816 - loss: 0.0488 - val_accuracy: 0.8777 - val_loss: 0.4716
Epoch 353/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9832 - loss: 0.0325 - val_accuracy: 0.8850 - val_loss: 0.4296
Epoch 354/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9893 - loss: 0.0419 - val_accuracy: 0.8960 - val_loss: 0.4061
Epoch 355/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9899 - loss: 0.0328 - val_accuracy: 0.8777 - val_loss: 0.4409
Epoch 356/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9827 - loss: 0.0366 - val_accuracy: 0.8522 - val_loss: 0.6433
Epoch 357/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9820 - loss: 0.0511 - val_accuracy: 0.8923 - val_loss: 0.3769
Epoch 358/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9856 - loss: 0.0445 - val_accuracy: 0.7646 - val_loss: 1.1230
Epoch 359/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 289ms/step - accuracy: 0.9868 - loss: 0.0323 - val_accuracy: 0.9471 - val_loss: 0.1994
Epoch 360/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9896 - loss: 0.0415 - val_accuracy: 0.9161 - val_loss: 0.2528
Epoch 361/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9902 - loss: 0.0316 - val_accuracy: 0.8668 - val_loss: 0.6608
Epoch 362/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9905 - loss: 0.0228 - val_accuracy: 0.9161 - val_loss: 0.3161
Epoch 363/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9773 - loss: 0.0630 - val_accuracy: 0.9325 - val_loss: 0.2675
Epoch 364/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9847 - loss: 0.0372 - val_accuracy: 0.7427 - val_loss: 0.8947
Epoch 365/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9893 - loss: 0.0346 - val_accuracy: 0.8704 - val_loss: 0.5978
Epoch 366/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 274ms/step - accuracy: 0.9939 - loss: 0.0220 - val_accuracy: 0.5073 - val_loss: 5.5773
Epoch 367/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9879 - loss: 0.0374 - val_accuracy: 0.8923 - val_loss: 0.4480
Epoch 368/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9836 - loss: 0.0377 - val_accuracy: 0.8668 - val_loss: 0.5987
Epoch 369/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 272ms/step - accuracy: 0.9838 - loss: 0.0522 - val_accuracy: 0.9252 - val_loss: 0.3592
Epoch 370/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9881 - loss: 0.0336 - val_accuracy: 0.7956 - val_loss: 1.1193
Epoch 371/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9834 - loss: 0.0471 - val_accuracy: 0.6606 - val_loss: 2.7020
Epoch 372/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9829 - loss: 0.0451 - val_accuracy: 0.8120 - val_loss: 1.1880
Epoch 373/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9920 - loss: 0.0260 - val_accuracy: 0.9234 - val_loss: 0.2362
Epoch 374/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9911 - loss: 0.0282 - val_accuracy: 0.8540 - val_loss: 0.5167
Epoch 375/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9839 - loss: 0.0513 - val_accuracy: 0.9015 - val_loss: 0.3496
Epoch 376/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9901 - loss: 0.0339 - val_accuracy: 0.7701 - val_loss: 1.4162
Epoch 377/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9897 - loss: 0.0341 - val_accuracy: 0.8558 - val_loss: 0.7105
Epoch 378/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 289ms/step - accuracy: 0.9778 - loss: 0.0430 - val_accuracy: 0.8540 - val_loss: 0.5404
Epoch 379/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9916 - loss: 0.0272 - val_accuracy: 0.9179 - val_loss: 0.2797
Epoch 380/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9786 - loss: 0.0390 - val_accuracy: 0.9252 - val_loss: 0.2828
Epoch 381/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9841 - loss: 0.0371 - val_accuracy: 0.9142 - val_loss: 0.2801
Epoch 382/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9856 - loss: 0.0367 - val_accuracy: 0.7993 - val_loss: 0.9110
Epoch 383/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9874 - loss: 0.0342 - val_accuracy: 0.8613 - val_loss: 0.4838
Epoch 384/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9866 - loss: 0.0448 - val_accuracy: 0.8485 - val_loss: 0.6369
Epoch 385/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9829 - loss: 0.0702 - val_accuracy: 0.9142 - val_loss: 0.2997
Epoch 386/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9783 - loss: 0.0613 - val_accuracy: 0.9124 - val_loss: 0.3329
Epoch 387/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9861 - loss: 0.0359 - val_accuracy: 0.9288 - val_loss: 0.2386
Epoch 388/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9861 - loss: 0.0331 - val_accuracy: 0.9270 - val_loss: 0.2313
Epoch 389/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9924 - loss: 0.0285 - val_accuracy: 0.9106 - val_loss: 0.3079
Epoch 390/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 21s 295ms/step - accuracy: 0.9856 - loss: 0.0320 - val_accuracy: 0.9088 - val_loss: 0.3208
Epoch 391/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9799 - loss: 0.0492 - val_accuracy: 0.9106 - val_loss: 0.3311
Epoch 392/1000
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9891 - loss: 0.0317 - val_accuracy: 0.9544 - val_loss: 0.1388
Epoch 393/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9801 - loss: 0.0451 - val_accuracy: 0.9343 - val_loss: 0.2246
Epoch 394/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9910 - loss: 0.0247 - val_accuracy: 0.8777 - val_loss: 0.5233
Epoch 395/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9884 - loss: 0.0261 - val_accuracy: 0.8777 - val_loss: 0.4351
Epoch 396/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9929 - loss: 0.0258 - val_accuracy: 0.9325 - val_loss: 0.2306
Epoch 397/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9887 - loss: 0.0345 - val_accuracy: 0.8741 - val_loss: 0.5993
Epoch 398/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9859 - loss: 0.0331 - val_accuracy: 0.9197 - val_loss: 0.3704
Epoch 399/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9918 - loss: 0.0282 - val_accuracy: 0.9252 - val_loss: 0.2981
Epoch 400/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9900 - loss: 0.0342 - val_accuracy: 0.7044 - val_loss: 1.8152
Epoch 401/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 274ms/step - accuracy: 0.9809 - loss: 0.0449 - val_accuracy: 0.9197 - val_loss: 0.2599
Epoch 402/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9870 - loss: 0.0367 - val_accuracy: 0.8084 - val_loss: 1.1179
Epoch 403/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9904 - loss: 0.0329 - val_accuracy: 0.9015 - val_loss: 0.3752
Epoch 404/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9875 - loss: 0.0299 - val_accuracy: 0.9343 - val_loss: 0.2102
Epoch 405/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9912 - loss: 0.0230 - val_accuracy: 0.9507 - val_loss: 0.1892
Epoch 406/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9864 - loss: 0.0323 - val_accuracy: 0.9197 - val_loss: 0.2874
Epoch 407/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9887 - loss: 0.0289 - val_accuracy: 0.8942 - val_loss: 0.4262
Epoch 408/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 287ms/step - accuracy: 0.9886 - loss: 0.0317 - val_accuracy: 0.9215 - val_loss: 0.2796
Epoch 409/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9889 - loss: 0.0237 - val_accuracy: 0.9051 - val_loss: 0.3196
Epoch 410/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9892 - loss: 0.0275 - val_accuracy: 0.8869 - val_loss: 0.5416
Epoch 411/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9898 - loss: 0.0315 - val_accuracy: 0.8650 - val_loss: 0.6540
Epoch 412/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9872 - loss: 0.0345 - val_accuracy: 0.5985 - val_loss: 2.6215
Epoch 413/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9854 - loss: 0.0468 - val_accuracy: 0.7792 - val_loss: 1.1052
Epoch 414/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9898 - loss: 0.0377 - val_accuracy: 0.7482 - val_loss: 1.4979
Epoch 415/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9889 - loss: 0.0258 - val_accuracy: 0.8741 - val_loss: 0.6251
Epoch 416/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9905 - loss: 0.0247 - val_accuracy: 0.8358 - val_loss: 0.5912
Epoch 417/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 273ms/step - accuracy: 0.9879 - loss: 0.0334 - val_accuracy: 0.8887 - val_loss: 0.4794
Epoch 418/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9858 - loss: 0.0358 - val_accuracy: 0.8285 - val_loss: 0.7827
Epoch 419/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9835 - loss: 0.0337 - val_accuracy: 0.8978 - val_loss: 0.3891
Epoch 420/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9915 - loss: 0.0251 - val_accuracy: 0.9197 - val_loss: 0.3405
Epoch 421/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9918 - loss: 0.0237 - val_accuracy: 0.9252 - val_loss: 0.2626
Epoch 422/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9855 - loss: 0.0334 - val_accuracy: 0.9215 - val_loss: 0.3410
Epoch 423/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9804 - loss: 0.0549 - val_accuracy: 0.9416 - val_loss: 0.2039
Epoch 424/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9694 - loss: 0.0683 - val_accuracy: 0.9526 - val_loss: 0.1574
Epoch 425/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9923 - loss: 0.0263 - val_accuracy: 0.9215 - val_loss: 0.2711
Epoch 426/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9939 - loss: 0.0191 - val_accuracy: 0.9526 - val_loss: 0.2216
Epoch 427/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9915 - loss: 0.0242 - val_accuracy: 0.9434 - val_loss: 0.1851
Epoch 428/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9944 - loss: 0.0204 - val_accuracy: 0.9015 - val_loss: 0.3212
Epoch 429/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9862 - loss: 0.0304 - val_accuracy: 0.8285 - val_loss: 1.2200
Epoch 430/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9901 - loss: 0.0237 - val_accuracy: 0.8412 - val_loss: 1.1974
Epoch 431/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9812 - loss: 0.0418 - val_accuracy: 0.8449 - val_loss: 0.7932
Epoch 432/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9906 - loss: 0.0252 - val_accuracy: 0.9453 - val_loss: 0.1965
Epoch 433/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 285ms/step - accuracy: 0.9893 - loss: 0.0299 - val_accuracy: 0.8741 - val_loss: 0.5137
Epoch 434/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9884 - loss: 0.0299 - val_accuracy: 0.9380 - val_loss: 0.2615
Epoch 435/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9956 - loss: 0.0173 - val_accuracy: 0.7938 - val_loss: 1.2613
```

```
Epoch 436/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 289ms/step - accuracy: 0.9930 - loss: 0.0242 - val_accuracy: 0.8485 - val_loss: 0.8538
Epoch 437/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 288ms/step - accuracy: 0.9931 - loss: 0.0255 - val_accuracy: 0.9471 - val_loss: 0.2704
Epoch 438/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9969 - loss: 0.0139 - val_accuracy: 0.8704 - val_loss: 0.4708
Epoch 439/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9857 - loss: 0.0310 - val_accuracy: 0.8650 - val_loss: 0.6592
Epoch 440/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9818 - loss: 0.0437 - val_accuracy: 0.6770 - val_loss: 0.9068
Epoch 441/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9895 - loss: 0.0362 - val_accuracy: 0.8905 - val_loss: 0.4353
Epoch 442/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9895 - loss: 0.0319 - val_accuracy: 0.8850 - val_loss: 0.4012
Epoch 443/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9882 - loss: 0.0344 - val_accuracy: 0.8960 - val_loss: 0.3716
Epoch 444/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9903 - loss: 0.0239 - val_accuracy: 0.9288 - val_loss: 0.2993
Epoch 445/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9902 - loss: 0.0292 - val_accuracy: 0.8613 - val_loss: 0.4052
Epoch 446/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9907 - loss: 0.0385 - val_accuracy: 0.7646 - val_loss: 1.7816
Epoch 447/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 288ms/step - accuracy: 0.9924 - loss: 0.0291 - val_accuracy: 0.8923 - val_loss: 0.4492
Epoch 448/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9942 - loss: 0.0161 - val_accuracy: 0.7464 - val_loss: 1.4134
Epoch 449/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 280ms/step - accuracy: 0.9913 - loss: 0.0296 - val_accuracy: 0.8376 - val_loss: 0.5532
Epoch 450/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9887 - loss: 0.0291 - val_accuracy: 0.9252 - val_loss: 0.3606
Epoch 451/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 276ms/step - accuracy: 0.9924 - loss: 0.0264 - val_accuracy: 0.8522 - val_loss: 0.8866
Epoch 452/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9877 - loss: 0.0255 - val_accuracy: 0.9361 - val_loss: 0.1916
Epoch 453/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9895 - loss: 0.0246 - val_accuracy: 0.9197 - val_loss: 0.3560
Epoch 454/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 274ms/step - accuracy: 0.9908 - loss: 0.0281 - val_accuracy: 0.9197 - val_loss: 0.2798
Epoch 455/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 273ms/step - accuracy: 0.9928 - loss: 0.0163 - val_accuracy: 0.9325 - val_loss: 0.2636
Epoch 456/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9843 - loss: 0.0415 - val_accuracy: 0.9307 - val_loss: 0.2254
Epoch 457/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 280ms/step - accuracy: 0.9845 - loss: 0.0402 - val_accuracy: 0.7263 - val_loss: 1.7339
Epoch 458/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9926 - loss: 0.0225 - val_accuracy: 0.8996 - val_loss: 0.3591
Epoch 459/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9924 - loss: 0.0236 - val_accuracy: 0.9161 - val_loss: 0.2694
Epoch 460/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 272ms/step - accuracy: 0.9935 - loss: 0.0199 - val_accuracy: 0.9106 - val_loss: 0.2877
Epoch 461/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9919 - loss: 0.0213 - val_accuracy: 0.6642 - val_loss: 2.4864
Epoch 462/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9923 - loss: 0.0254 - val_accuracy: 0.6734 - val_loss: 3.1274
Epoch 463/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9841 - loss: 0.0331 - val_accuracy: 0.9288 - val_loss: 0.2603
Epoch 464/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9896 - loss: 0.0280 - val_accuracy: 0.9215 - val_loss: 0.3316
Epoch 465/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9906 - loss: 0.0279 - val_accuracy: 0.9307 - val_loss: 0.2594
Epoch 466/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9936 - loss: 0.0196 - val_accuracy: 0.8029 - val_loss: 0.5298
Epoch 467/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9905 - loss: 0.0313 - val_accuracy: 0.8978 - val_loss: 0.4726
Epoch 468/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9887 - loss: 0.0378 - val_accuracy: 0.8394 - val_loss: 0.8603
Epoch 469/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9942 - loss: 0.0198 - val_accuracy: 0.9307 - val_loss: 0.2932
Epoch 470/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9948 - loss: 0.0135 - val_accuracy: 0.9215 - val_loss: 0.3350
Epoch 471/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9893 - loss: 0.0314 - val_accuracy: 0.7190 - val_loss: 2.1893
Epoch 472/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9860 - loss: 0.0514 - val_accuracy: 0.8905 - val_loss: 0.5282
Epoch 473/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9872 - loss: 0.0366 - val_accuracy: 0.8102 - val_loss: 0.9518
Epoch 474/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9877 - loss: 0.0334 - val_accuracy: 0.8449 - val_loss: 0.8780
Epoch 475/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 290ms/step - accuracy: 0.9909 - loss: 0.0231 - val_accuracy: 0.8996 - val_loss: 0.4847
Epoch 476/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9909 - loss: 0.0242 - val_accuracy: 0.8285 - val_loss: 0.6799
Epoch 477/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9914 - loss: 0.0250 - val_accuracy: 0.8905 - val_loss: 0.3861
Epoch 478/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 281ms/step - accuracy: 0.9914 - loss: 0.0363 - val_accuracy: 0.9544 - val_loss: 0.2660
Epoch 479/1000
```

```
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9880 - loss: 0.0341 - val_accuracy: 0.7208 - val_loss: 1.9702
Epoch 480/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9899 - loss: 0.0258 - val_accuracy: 0.9270 - val_loss: 0.3399
Epoch 481/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 272ms/step - accuracy: 0.9941 - loss: 0.0171 - val_accuracy: 0.9325 - val_loss: 0.2548
Epoch 482/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 275ms/step - accuracy: 0.9959 - loss: 0.0189 - val_accuracy: 0.7737 - val_loss: 1.6056
Epoch 483/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9917 - loss: 0.0237 - val_accuracy: 0.8796 - val_loss: 0.4757
Epoch 484/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 283ms/step - accuracy: 0.9964 - loss: 0.0107 - val_accuracy: 0.9142 - val_loss: 0.4991
Epoch 485/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9867 - loss: 0.0295 - val_accuracy: 0.8631 - val_loss: 0.6574
Epoch 486/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 282ms/step - accuracy: 0.9933 - loss: 0.0222 - val_accuracy: 0.9416 - val_loss: 0.2847
Epoch 487/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 277ms/step - accuracy: 0.9894 - loss: 0.0390 - val_accuracy: 0.8869 - val_loss: 0.5023
Epoch 488/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 284ms/step - accuracy: 0.9887 - loss: 0.0323 - val_accuracy: 0.8942 - val_loss: 0.4608
Epoch 489/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 292ms/step - accuracy: 0.9939 - loss: 0.0163 - val_accuracy: 0.9270 - val_loss: 0.3126
Epoch 490/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 20s 286ms/step - accuracy: 0.9898 - loss: 0.0244 - val_accuracy: 0.7536 - val_loss: 1.3706
Epoch 491/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 278ms/step - accuracy: 0.9853 - loss: 0.0436 - val_accuracy: 0.8996 - val_loss: 0.3964
Epoch 492/1000
69/69 ━━━━━━━━━━━━━━━━━━━━ 19s 279ms/step - accuracy: 0.9893 - loss: 0.0318 - val_accuracy: 0.8157 - val_loss: 0.4786
Epoch 492: early stopping
Restoring model weights from the end of the best epoch: 392.
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 96ms/step - accuracy: 0.9456 - loss: 0.2393
Retrained ResNet with Augmentation Test accuracy: 94.97%, Test loss: 0.2543785870075226
```





In [ ]:
```
with open('model_res_retrained_aug.pkl', 'wb') as file:
    pickle.dump(model_res_retrained_aug, file)
with open('history_res_retrained_aug.pkl', 'wb') as file:
    pickle.dump(history_res_retrained_aug, file)
```

7 Interpretability & Insights

Enhanced Training of ResNet50 with Data Augmentation (`model_res_retrained_aug`) performed the best!
This is an intuitive result because in this model we retrained all the params of resnet50 with an additional layer mapping to the final result. The retrained CNN layers could capture the specific patterns that is helpful for our medical task. Also, since we used augmentation as described in section 6, we make the model more robust and stronger in identifier patterns that could categorize different diseases and normal conditions. The detail of this model is clearly labeled in section 6. The OOS accuracy is 95%, which is better than all the models without augmentation, including baseline CNN, CNN pro, resnet50, resnet50 retrained, and VGG16.
The performance curve is also clearly plotted in section 6.
The practical use of my best performing model is that it could help doctor/patients to identify the disease. It could serve as a supplementary check in additional to doctor's dicretionary judgement. 95% OOS accuracy is helpful enough in practical settings. It could also be served as a pre-diagnostic that could give a very fast diagnostic without human intervention. This is particularly helpful when medical resources is contraining in certain pedanmic times.

In [ ]:
```
```