G_2_am6490,_cj2831,_hk3354_Project_2

April 21, 2025

UNI: am6490, cj2831, hk3354

Full name: Arsh Misra, Conor Jones, Flora Kwon

Link to Public Github repository with Final report: https://github.com/hyerhinkwon/QMSS5074-Adv-ML.git

[2]:
```
# Load libraries

import sys
import time
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.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Flatten, Activation,␣
↪BatchNormalization, Conv2D, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, SGD, Adagrad, Adadelta, RMSprop
from tensorflow.keras.applications import ResNet50, InceptionV3
```

0.1 0. Loading Dataset

[5]:
```
# Import data from desktop

import os
base_path = "/Users/florakwon/Desktop/Spring 2025/QMSS 5074 - Adv ML/Project_2/
↪Project 2/COVID-19_Radiography_Dataset"
```

0.2 1. Dataset and Exploratory Data Analysis

Start by describing the dataset. Include basic statistics and image samples to show the types of images available (e.g., COVID-positive and negative chest x-rays).

*Check if the dataset is balanced across classes. If it's imbalanced: * Discuss potential strategies such as class weighting, oversampling, undersampling, or augmentation. * Indicate which method you chose, and discuss how model performance changed as a result.*

Reflect on the practical value of this classification task. Who might benefit from your model in a real-world setting?

[7]:
```
# Extracting all filenames iteratively
base_path = 'COVID-19_Radiography_Dataset' #CONORJ: Added this, double check if␣
↪it's in the pdf
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])
```

```
number of images for each category: [3616, 10192, 1345]
```

The original data consists chest X-ray images, 3616 images each for COVID-19 pneumonia, 1345 for viral pneumonia, and 10192 for normal.

To address class imbalance, we can utilize: 1. Class weighting: Assign higher weights to minority classes during training 2. Oversampling: Create synthetic samples of minority classes (e.g., SMOTE) 3. Undersampling: Remove samples from majority classes 4. Data augmentation: Generate additional samples through transformations

For our approach, we decided to artificially balance the dataset (by preserving 1344 samples per class), same as the source paper. This means that all classes will contirubte equally to gradien updates and prevent model bias towards the larger viral pneumonia class and normal cllass. In the paper, this demonstrated improved test accuracy and balanced performance across classes for confusion matrices.

From this classification exercise, we can provide insights to aid healthcare professionals in interpreting radiology reports and provide diagnostic support. From general ML knowledge perspective, it will also improve pattern recognition and its applications.

[9]:
```
# 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]
```

[10]:

```
# 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
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 and width you want.
        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
```

[11]:

```
# 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
```

[12]:

```
len(image_filepaths)
```

[12]: 4032

[13]:

```
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
```

[14]:

```
len(fnames[2])
```

[14]: 1344

[15]:
```
# 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 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
```

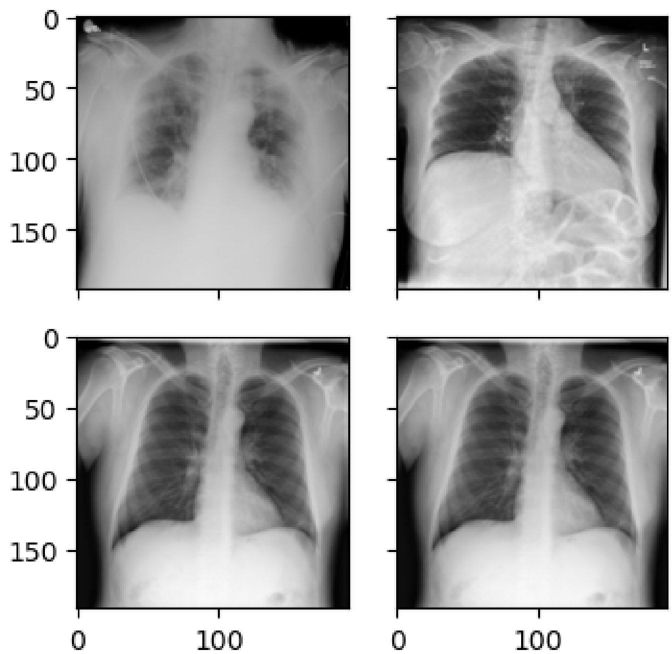
	COVID	NORMAL	PNEUMONIA
0	True	False	False
1	True	False	False
2	True	False	False
3	True	False	False
4	True	False	False
…	…	…	…
4027	False	False	True
4028	False	False	True
4029	False	False	True
4030	False	False	True
4031	False	False	True

[4032 rows x 3 columns]

[16]:
```
from mpl_toolkits.axes_grid1 import ImageGrid
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()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.003937008..0.8425197].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.003937008..0.96456695].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.003937008..0.98031497].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.003937008..0.98031497].
```



[17]:
```python
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
```

[17]: ((1291, 192, 192, 3), (1291, 3))

[18]:
```python
# Clear objects from memory
del(X)
del(y)
del(preprocessed_image_data)
```

[19]:
```python
#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)
```

0.3 2. Baseline CNN Model

Build and train a basic Convolutional Neural Network (CNN) to serve as a baseline.

Clearly describe the architecture, loss function, optimizer, evaluation metrics, and training configuration.

Report the model's training, validation, and test performance.

[21]:
```python
# Building baseline CNN

def baseline_cnn(input_shape=(192, 192, 3), num_classes=3):

    model = Sequential([
```

```
        Conv2D(32, (3, 3), activation='relu', padding='same',
 ↪input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(3, activation='softmax')
    ])
    return model

baseline_model = baseline_cnn(input_shape=(192, 192, 3), num_classes=3)
baseline_model.compile(optimizer='adam', loss='categorical_crossentropy',
 ↪metrics=['accuracy'])
baseline_model.summary()
```

```
/opt/anaconda3/lib/python3.10/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`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)
2025-04-21 22:05:22.097316: I metal_plugin/src/device/metal_device.cc:1154]
Metal device set to: Apple M3
2025-04-21 22:05:22.097343: I metal_plugin/src/device/metal_device.cc:296]
systemMemory: 16.00 GB
2025-04-21 22:05:22.097354: I metal_plugin/src/device/metal_device.cc:313]
maxCacheSize: 5.33 GB
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1745287522.097371 7678761 pluggable_device_factory.cc:305] Could not
identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not
have been built with NUMA support.
I0000 00:00:1745287522.097397 7678761 pluggable_device_factory.cc:271] Created
TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB
memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 192, 192, 32)	896
max_pooling2d (MaxPooling2D)	(None, 96, 96, 32)	0
flatten (Flatten)	(None, 294912)	0
dense (Dense)	(None, 3)	884,739

Total params: 885,635 (3.38 MB)

Trainable params: 885,635 (3.38 MB)

Non-trainable params: 0 (0.00 B)

The baseline model is a convolutional neural network built with Keras.

The architecture consists of a single convolutional layer with 32 filters followed by max-pooling to reduce spatial dimensions. The final dense layer with a softmax activation outputs probabilities for 3 classes.

We used Categorical Cross-entropy as the loss function. It is appropriate for multi-class classification problems with one-hot encoded labels, to measure the difference between the true label distribution and the predicted probabilities.

We used Adam as the optimizer, an adaptive learning rate optimizer for deep learning.

We used Accuracy as the evaluation metric, which would indicate proportion of correctly classified samples.

Training is run for up to 5 epochs. We use the validation set to monitor the performance after each epoch.

[23]:
```
baseline_history = baseline_model.fit(X_train, y_train, epochs=5,
    batch_size=64, validation_data=(X_test, y_test))
```

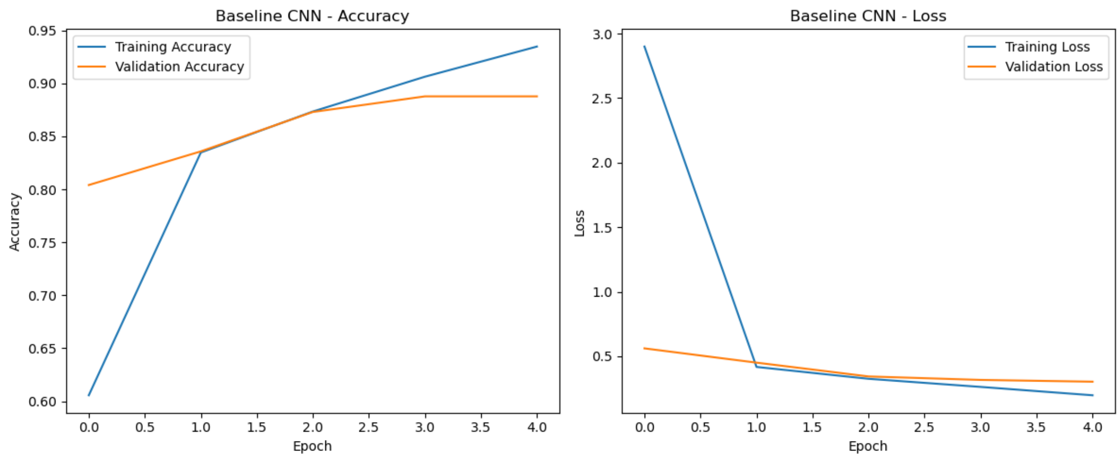
```
Epoch 1/5

2025-04-21 22:05:23.287479: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117]
Plugin optimizer for device_type GPU is enabled.

43/43              6s 94ms/step -
accuracy: 0.4692 - loss: 5.9074 - val_accuracy: 0.8040 - val_loss: 0.5611
Epoch 2/5
43/43              3s 77ms/step -
accuracy: 0.8282 - loss: 0.4311 - val_accuracy: 0.8358 - val_loss: 0.4506
Epoch 3/5
43/43              3s 72ms/step -
accuracy: 0.8615 - loss: 0.3511 - val_accuracy: 0.8730 - val_loss: 0.3437
Epoch 4/5
43/43              3s 71ms/step -
accuracy: 0.9061 - loss: 0.2602 - val_accuracy: 0.8877 - val_loss: 0.3168
Epoch 5/5
43/43              3s 72ms/step -
accuracy: 0.9270 - loss: 0.2132 - val_accuracy: 0.8877 - val_loss: 0.3030
```

[24]:

```python
# Code for Training and Validation Performance Plot

def plot_training(history, model_name):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(len(acc))

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, label='Training Accuracy')
    plt.plot(epochs, val_acc, label='Validation Accuracy')
    plt.title(f'{model_name} - Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, label='Training Loss')
    plt.plot(epochs, val_loss, label='Validation Loss')
    plt.title(f'{model_name} - Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

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

[25]:

```python
# Plot training history
plot_training(baseline_history, 'Baseline CNN')

# Evaluate the model on test data
baseline_test_loss, baseline_test_acc = baseline_model.evaluate(X_test, y_test)
print(f"Baseline CNN Test Accuracy: {baseline_test_acc*100:.2f}%")
```




```
41/41              1s 14ms/step -
accuracy: 0.8757 - loss: 0.3178
Baseline CNN Test Accuracy: 88.77%
```

These results indicate that even though our training accuracy achieves a rate of ~93%, this accuracy doesn't hold on the validation set which drops to ~87.2%. Analyzing the change in validation loss, it seems unlikely that adding more epochs alone would substantially improve the accuracy of this set since the loss has begun to flatline (we see a loss 0f 0.35 in epoch 4 and 0.31 in epoch 5). In fact, we suspect more epochs using this infrastructure may be more likely to lead to overfitting than to improved performance.

The final test accuracy is 89%.

0.4 3. Transfer Learning with ResNet

Implement ResNet using transfer learning.

Fine-tune the model and compare its performance with the baseline CNN.

Discuss how using pre-trained features influences your model's training and generalization.

[28]:
```python
from tensorflow.keras.applications.resnet50 import preprocess_input as␣
↪resnet_preprocess

# Create a tf.data pipeline that resizes images on the fly.
def preprocess_and_resize(image, label):
    # Resize image to 224x224 and cast to float32
    image = tf.image.resize(image, (224, 224))
    image = tf.cast(image * 255.0, tf.float32)
    # Apply the ResNet50 preprocessing function
    image = resnet_preprocess(image)
    return image, label

# Create tf.data datasets for train and test sets.
```

```
batch_size = 64

train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train_ds = train_ds.map(preprocess_and_resize, num_parallel_calls=tf.data.
 ↪AUTOTUNE)
train_ds = train_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)

test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test))
test_ds = test_ds.map(preprocess_and_resize, num_parallel_calls=tf.data.
 ↪AUTOTUNE)
test_ds = test_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

[29]:
```
from tensorflow.keras import layers, models
from tensorflow.keras.layers import Input, GlobalAveragePooling2D

# Load ResNet50 model
input_tensor = Input(shape=(224, 224, 3))
base_resnet = ResNet50(include_top=False, weights='imagenet',␣
 ↪input_tensor=input_tensor)
x = base_resnet.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(3, activation='softmax')(x)

# Freeze layers
for layer in base_resnet.layers:
    layer.trainable = False

# Build model with transfer learning
resnet_model = Model(inputs=base_resnet.input, outputs=predictions)
resnet_model.compile(optimizer=Adam(learning_rate=0.001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
resnet_model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_layer_1[0]…
conv1_conv (Conv2D)	(None, 112, 112, 64)	9,472	conv1_pad[0][0]

conv1_bn (BatchNormalizatio…	(None, 112, 112, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block1_1_c…
conv2_block1_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_block1_1_b…
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_block1_1_r…
conv2_block1_2_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block1_2_c…
conv2_block1_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_block1_2_b…
conv2_block1_0_conv (Conv2D)	(None, 56, 56, 256)	16,640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 56, 56, 256)	16,640	conv2_block1_2_r…
conv2_block1_0_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_block1_0_c…
conv2_block1_3_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_block1_3_c…
conv2_block1_add (Add)	(None, 56, 56, 256)	0	conv2_block1_0_b… conv2_block1_3_b…
conv2_block1_out (Activation)	(None, 56, 56, 256)	0	conv2_block1_add…

conv2_block2_1_conv (Conv2D)	(None, 56, 56, 64)	16,448	conv2_block1_out…
conv2_block2_1_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block2_1_c…
conv2_block2_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_block2_1_b…
conv2_block2_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_block2_1_r…
conv2_block2_2_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block2_2_c…
conv2_block2_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_block2_2_b…
conv2_block2_3_conv (Conv2D)	(None, 56, 56, 256)	16,640	conv2_block2_2_r…
conv2_block2_3_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_block2_3_c…
conv2_block2_add (Add)	(None, 56, 56, 256)	0	conv2_block1_out… conv2_block2_3_b…
conv2_block2_out (Activation)	(None, 56, 56, 256)	0	conv2_block2_add…
conv2_block3_1_conv (Conv2D)	(None, 56, 56, 64)	16,448	conv2_block2_out…
conv2_block3_1_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block3_1_c…
conv2_block3_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_block3_1_b…
conv2_block3_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_block3_1_r…
conv2_block3_2_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block3_2_c…
conv2_block3_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_block3_2_b…

conv2_block3_3_conv (Conv2D)	(None, 56, 56, 256)	16,640	conv2_block3_2_r…
conv2_block3_3_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_block3_3_c…
conv2_block3_add (Add)	(None, 56, 56, 256)	0	conv2_block2_out… conv2_block3_3_b…
conv2_block3_out (Activation)	(None, 56, 56, 256)	0	conv2_block3_add…
conv3_block1_1_conv (Conv2D)	(None, 28, 28, 128)	32,896	conv2_block3_out…
conv3_block1_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block1_1_c…
conv3_block1_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block1_1_b…
conv3_block1_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_block1_1_r…
conv3_block1_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block1_2_c…
conv3_block1_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_block1_2_b…
conv3_block1_0_conv (Conv2D)	(None, 28, 28, 512)	131,584	conv2_block3_out…
conv3_block1_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_block1_2_r…
conv3_block1_0_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_block1_0_c…
conv3_block1_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_block1_3_c…
conv3_block1_add (Add)	(None, 28, 28, 512)	0	conv3_block1_0_b… conv3_block1_3_b…
conv3_block1_out (Activation)	(None, 28, 28, 512)	0	conv3_block1_add…

conv3_block2_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_block1_out…
conv3_block2_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block2_1_c…
conv3_block2_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block2_1_b…
conv3_block2_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_block2_1_r…
conv3_block2_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block2_2_c…
conv3_block2_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_block2_2_b…
conv3_block2_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_block2_2_r…
conv3_block2_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_block2_3_c…
conv3_block2_add (Add)	(None, 28, 28, 512)	0	conv3_block1_out… conv3_block2_3_b…
conv3_block2_out (Activation)	(None, 28, 28, 512)	0	conv3_block2_add…
conv3_block3_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_block2_out…
conv3_block3_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block3_1_c…
conv3_block3_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block3_1_b…
conv3_block3_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_block3_1_r…
conv3_block3_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block3_2_c…
conv3_block3_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_block3_2_b…

conv3_block3_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_block3_2_r…
conv3_block3_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_block3_3_c…
conv3_block3_add (Add)	(None, 28, 28, 512)	0	conv3_block2_out… conv3_block3_3_b…
conv3_block3_out (Activation)	(None, 28, 28, 512)	0	conv3_block3_add…
conv3_block4_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_block3_out…
conv3_block4_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block4_1_c…
conv3_block4_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block4_1_b…
conv3_block4_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_block4_1_r…
conv3_block4_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block4_2_c…
conv3_block4_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_block4_2_b…
conv3_block4_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_block4_2_r…
conv3_block4_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_block4_3_c…
conv3_block4_add (Add)	(None, 28, 28, 512)	0	conv3_block3_out… conv3_block4_3_b…
conv3_block4_out (Activation)	(None, 28, 28, 512)	0	conv3_block4_add…
conv4_block1_1_conv (Conv2D)	(None, 14, 14, 256)	131,328	conv3_block4_out…
conv4_block1_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block1_1_c…

conv4_block1_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block1_1_b…
conv4_block1_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block1_1_r…
conv4_block1_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block1_2_c…
conv4_block1_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block1_2_b…
conv4_block1_0_conv (Conv2D)	(None, 14, 14, 1024)	525,312	conv3_block4_out…
conv4_block1_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block1_2_r…
conv4_block1_0_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_block1_0_c…
conv4_block1_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_block1_3_c…
conv4_block1_add (Add)	(None, 14, 14, 1024)	0	conv4_block1_0_b… conv4_block1_3_b…
conv4_block1_out (Activation)	(None, 14, 14, 1024)	0	conv4_block1_add…
conv4_block2_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_block1_out…
conv4_block2_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block2_1_c…
conv4_block2_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block2_1_b…
conv4_block2_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block2_1_r…
conv4_block2_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block2_2_c…
conv4_block2_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block2_2_b…

conv4_block2_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block2_2_r…
conv4_block2_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_block2_3_c…
conv4_block2_add (Add)	(None, 14, 14, 1024)	0	conv4_block1_out… conv4_block2_3_b…
conv4_block2_out (Activation)	(None, 14, 14, 1024)	0	conv4_block2_add…
conv4_block3_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_block2_out…
conv4_block3_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block3_1_c…
conv4_block3_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block3_1_b…
conv4_block3_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block3_1_r…
conv4_block3_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block3_2_c…
conv4_block3_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block3_2_b…
conv4_block3_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block3_2_r…
conv4_block3_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_block3_3_c…
conv4_block3_add (Add)	(None, 14, 14, 1024)	0	conv4_block2_out… conv4_block3_3_b…
conv4_block3_out (Activation)	(None, 14, 14, 1024)	0	conv4_block3_add…
conv4_block4_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_block3_out…
conv4_block4_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block4_1_c…

conv4_block4_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block4_1_b…
conv4_block4_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block4_1_r…
conv4_block4_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block4_2_c…
conv4_block4_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block4_2_b…
conv4_block4_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block4_2_r…
conv4_block4_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_block4_3_c…
conv4_block4_add (Add)	(None, 14, 14, 1024)	0	conv4_block3_out… conv4_block4_3_b…
conv4_block4_out (Activation)	(None, 14, 14, 1024)	0	conv4_block4_add…
conv4_block5_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_block4_out…
conv4_block5_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block5_1_c…
conv4_block5_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block5_1_b…
conv4_block5_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block5_1_r…
conv4_block5_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block5_2_c…
conv4_block5_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block5_2_b…
conv4_block5_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block5_2_r…
conv4_block5_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_block5_3_c…

conv4_block5_add (Add)	(None, 14, 14, 1024)	0	conv4_block4_out… conv4_block5_3_b…
conv4_block5_out (Activation)	(None, 14, 14, 1024)	0	conv4_block5_add…
conv4_block6_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_block5_out…
conv4_block6_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block6_1_c…
conv4_block6_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block6_1_b…
conv4_block6_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block6_1_r…
conv4_block6_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_block6_2_c…
conv4_block6_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block6_2_b…
conv4_block6_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block6_2_r…
conv4_block6_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_block6_3_c…
conv4_block6_add (Add)	(None, 14, 14, 1024)	0	conv4_block5_out… conv4_block6_3_b…
conv4_block6_out (Activation)	(None, 14, 14, 1024)	0	conv4_block6_add…
conv5_block1_1_conv (Conv2D)	(None, 7, 7, 512)	524,800	conv4_block6_out…
conv5_block1_1_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_block1_1_c…
conv5_block1_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_block1_1_b…
conv5_block1_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_block1_1_r…

conv5_block1_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_block1_2_c…
conv5_block1_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_block1_2_b…
conv5_block1_0_conv (Conv2D)	(None, 7, 7, 2048)	2,099,200	conv4_block6_out…
conv5_block1_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_block1_2_r…
conv5_block1_0_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_block1_0_c…
conv5_block1_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_block1_3_c…
conv5_block1_add (Add)	(None, 7, 7, 2048)	0	conv5_block1_0_b… conv5_block1_3_b…
conv5_block1_out (Activation)	(None, 7, 7, 2048)	0	conv5_block1_add…
conv5_block2_1_conv (Conv2D)	(None, 7, 7, 512)	1,049,088	conv5_block1_out…
conv5_block2_1_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_block2_1_c…
conv5_block2_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_block2_1_b…
conv5_block2_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_block2_1_r…
conv5_block2_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_block2_2_c…
conv5_block2_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_block2_2_b…
conv5_block2_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_block2_2_r…
conv5_block2_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_block2_3_c…

conv5_block2_add (Add)	(None, 7, 7, 2048)	0	conv5_block1_out… conv5_block2_3_b…
conv5_block2_out (Activation)	(None, 7, 7, 2048)	0	conv5_block2_add…
conv5_block3_1_conv (Conv2D)	(None, 7, 7, 512)	1,049,088	conv5_block2_out…
conv5_block3_1_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_block3_1_c…
conv5_block3_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_block3_1_b…
conv5_block3_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_block3_1_r…
conv5_block3_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_block3_2_c…
conv5_block3_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_block3_2_b…
conv5_block3_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_block3_2_r…
conv5_block3_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_block3_3_c…
conv5_block3_add (Add)	(None, 7, 7, 2048)	0	conv5_block2_out… conv5_block3_3_b…
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	conv5_block3_add…
global_average_poo… (GlobalAveragePool…	(None, 2048)	0	conv5_block3_out…
dense_1 (Dense)	(None, 3)	6,147	global_average_p…

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

Trainable params: 6,147 (24.01 KB)

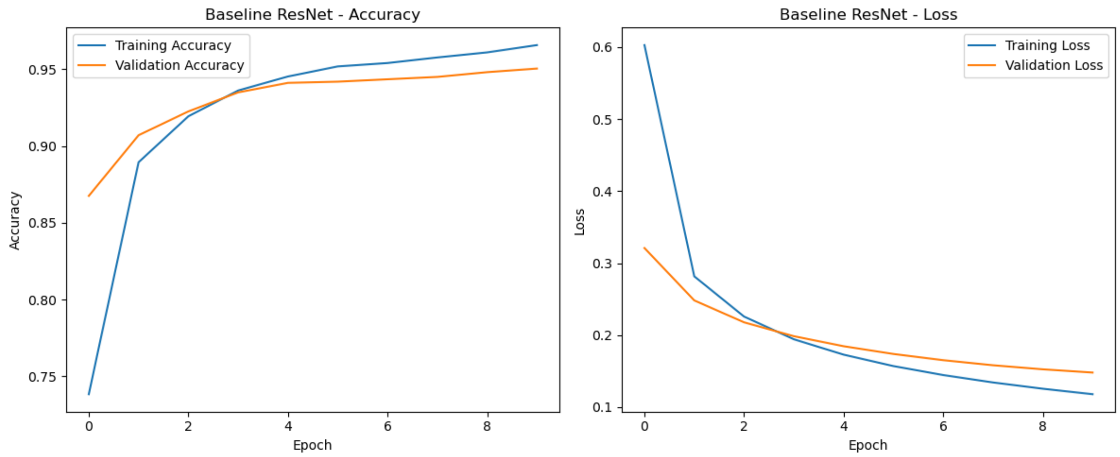
```
Non-trainable params: 23,587,712 (89.98 MB)
```

[30]:
```python
# Fit the ResNet model to our training and validation data sets
history_resnet = resnet_model.fit(train_ds, epochs=10, validation_data=test_ds)
```

```
Epoch 1/10

/opt/anaconda3/lib/python3.10/site-packages/keras/src/models/functional.py:238:
UserWarning: The structure of `inputs` doesn't match the expected structure.
Expected: ['keras_tensor_5']
Received: inputs=Tensor(shape=(None, 224, 224, 3))
  warnings.warn(msg)

43/43                 31s 662ms/step -
accuracy: 0.5900 - loss: 0.8971 - val_accuracy: 0.8675 - val_loss: 0.3210
Epoch 2/10
43/43                 27s 639ms/step -
accuracy: 0.8782 - loss: 0.3026 - val_accuracy: 0.9070 - val_loss: 0.2483
Epoch 3/10
43/43                 29s 671ms/step -
accuracy: 0.9108 - loss: 0.2387 - val_accuracy: 0.9225 - val_loss: 0.2178
Epoch 4/10
43/43                 30s 694ms/step -
accuracy: 0.9288 - loss: 0.2041 - val_accuracy: 0.9349 - val_loss: 0.1985
Epoch 5/10
43/43                 32s 744ms/step -
accuracy: 0.9393 - loss: 0.1809 - val_accuracy: 0.9411 - val_loss: 0.1846
Epoch 6/10
43/43                 36s 854ms/step -
accuracy: 0.9503 - loss: 0.1638 - val_accuracy: 0.9419 - val_loss: 0.1739
Epoch 7/10
43/43                 37s 855ms/step -
accuracy: 0.9534 - loss: 0.1503 - val_accuracy: 0.9435 - val_loss: 0.1652
Epoch 8/10
43/43                 34s 799ms/step -
accuracy: 0.9560 - loss: 0.1392 - val_accuracy: 0.9450 - val_loss: 0.1582
Epoch 9/10
43/43                 35s 816ms/step -
accuracy: 0.9591 - loss: 0.1297 - val_accuracy: 0.9481 - val_loss: 0.1525
Epoch 10/10
43/43                 38s 891ms/step -
accuracy: 0.9634 - loss: 0.1214 - val_accuracy: 0.9504 - val_loss: 0.1480
```

[31]:
```python
# Plot training history
plot_training(history_resnet, 'Baseline ResNet')

# Evaluate the model on test data
resnet_test_loss, resnet_test_acc = resnet_model.evaluate(test_ds)
```

```
print(f"Baseline ResNet Test Accuracy: {resnet_test_acc*100:.2f}%")
```




```
21/21                12s 593ms/step -
accuracy: 0.9527 - loss: 0.1351
Baseline ResNet Test Accuracy: 95.04%
```

The baseline ResNet model achieves a training accuracy of ~96% over the course of 10 epochs. Furthermore, it's performance on the validation set is ~95% which is a much lower drop from our baseline CNN. The final test accuracy is 95%.

Next we're going to try fine-tuning the model on the training data set over the course of 5 epochs.

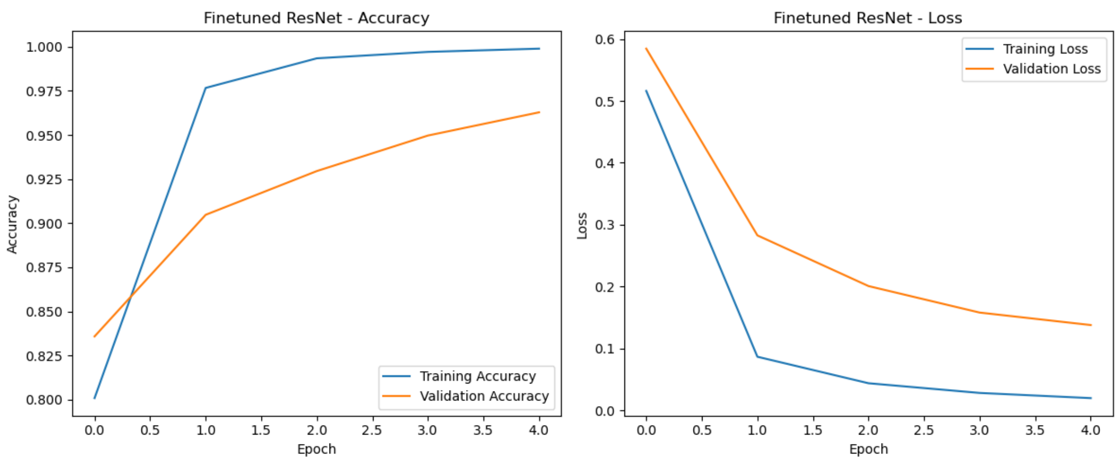
[33]:
```
# Unfreeze to fine-tune last 30 layers
for layer in base_resnet.layers[-30:]:
    layer.trainable = True

# Re-compile with a lower learning rate
resnet_model.compile(optimizer=Adam(learning_rate=0.00001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
```

[34]:
```
history_finetune = resnet_model.fit(train_ds, epochs=15, initial_epoch=10,␣
 ↪validation_data=test_ds)
```

```
Epoch 11/15

/opt/anaconda3/lib/python3.10/site-packages/keras/src/models/functional.py:238:
UserWarning: The structure of `inputs` doesn't match the expected structure.
Expected: ['keras_tensor_5']
Received: inputs=Tensor(shape=(None, 224, 224, 3))
  warnings.warn(msg)

43/43                58s 1s/step -
accuracy: 0.6962 - loss: 0.8197 - val_accuracy: 0.8358 - val_loss: 0.5845
Epoch 12/15
```

```
43/43                50s 1s/step -
accuracy: 0.9761 - loss: 0.0884 - val_accuracy: 0.9047 - val_loss: 0.2828
Epoch 13/15
43/43                52s 1s/step -
accuracy: 0.9934 - loss: 0.0446 - val_accuracy: 0.9295 - val_loss: 0.2008
Epoch 14/15
43/43                50s 1s/step -
accuracy: 0.9958 - loss: 0.0287 - val_accuracy: 0.9497 - val_loss: 0.1580
Epoch 15/15
43/43                50s 1s/step -
accuracy: 0.9981 - loss: 0.0199 - val_accuracy: 0.9628 - val_loss: 0.1378
```

[35]:
```python
# Plot training curves for fine-tuned ResNet50
plot_training(history_finetune, 'Finetuned ResNet')

# Evaluate fine-tuned ResNet50 on test data
finetune_test_loss, finetune_test_acc = resnet_model.evaluate(test_ds,␣
 ↪verbose=0)
print(f"Finetuned ResNet50 Test Accuracy: {finetune_test_acc*100:.2f}%")
```




```
Finetuned ResNet50 Test Accuracy: 96.28%
```

Training was much faster with pretrained features (10 epochs), as compared to fine-tuning (5 epochs). However, by the second fine-tuning epoch model accuracy had already slightly exceeded the base ResNet model, and by the end of the 5th epoch we had exceeded the prior accuracy. Furthermore, since loss continued to drop from epoch 4 to epoch 5 (0.15 to 0.14), that indicates that we potentially could have trained the fine tune model even further–though with a test accuracy of 96% in epoch 5, it is also possible the data at its current size might have been nearing the limits of its image differentiability.

0.5 4. Additional Architectures

Implement three additional models of your choice.

Use consistent data splits and preprocessing across all models to ensure fair comparison.

[38]:
```
# Define preprocessing for Improved CNN and AlexNet.

def preprocess_tf(image, label):
    image = tf.image.resize(image, [224, 224])
    image = tf.cast(image, tf.float32) / 255.0
    return image, label

batch_size = 32

train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train_ds = train_ds.map(preprocess_tf, num_parallel_calls=tf.data.AUTOTUNE)
train_ds = train_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)

test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test))
test_ds = test_ds.map(preprocess_tf, num_parallel_calls=tf.data.AUTOTUNE)
test_ds = test_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

[39]:
```
#Model 1:
# Improved CNN with more convolutional layers, increased dropout rate, and␣
↪increased number of dense layers
 # Through this model we can see whether increasing the depth of the model can␣
↪improve our accuracy measures

def improved_cnn(input_shape=(224, 224, 3), num_classes=3):
    model = Sequential([

        Conv2D(32, (3, 3), activation='relu', padding='same',␣
↪input_shape=input_shape),
        BatchNormalization(),
        MaxPooling2D((2, 2)),

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

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

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

        Flatten(),
        Dense(128, activation='relu'),
```

```
        Dropout(0.4),
        Dense(num_classes, activation='softmax')
    ])

    return model

improved_model = improved_cnn(input_shape=(224, 224, 3), num_classes=3)
improved_model.compile(optimizer='adam', loss='categorical_crossentropy',
 ↪metrics=['accuracy'])
improved_model.summary()
```

```
/opt/anaconda3/lib/python3.10/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`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_1 (Conv2D)	(None, 224, 224, 32)	896
batch_normalization (BatchNormalization)	(None, 224, 224, 32)	128
max_pooling2d_1 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_2 (Conv2D)	(None, 112, 112, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 112, 112, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_3 (Conv2D)	(None, 56, 56, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 56, 56, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 28, 28, 128)	0
conv2d_4 (Conv2D)	(None, 28, 28, 256)	295,168
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 256)	1,024

```
max_pooling2d_4 (MaxPooling2D)    (None, 14, 14, 256)                 0

flatten_1 (Flatten)               (None, 50176)                       0

dense_2 (Dense)                   (None, 128)                 6,422,656

dropout (Dropout)                 (None, 128)                         0

dense_3 (Dense)                   (None, 3)                         387
```

Total params: 6,813,379 (25.99 MB)

Trainable params: 6,812,419 (25.99 MB)

Non-trainable params: 960 (3.75 KB)

[40]:

```
#Furthermore, this model will employ a longer training period--using 10 epochs
improved_history = improved_model.fit(train_ds, epochs=10,␣
↪validation_data=(test_ds))
improved_test_loss, improved_test_acc = improved_model.evaluate(test_ds)
```

```
Epoch 1/10
86/86                 39s 414ms/step -
accuracy: 0.6141 - loss: 22.4603 - val_accuracy: 0.3338 - val_loss: 28.4353
Epoch 2/10
86/86                 34s 394ms/step -
accuracy: 0.6866 - loss: 26.1276 - val_accuracy: 0.3331 - val_loss: 30.8176
Epoch 3/10
86/86                 30s 352ms/step -
accuracy: 0.6952 - loss: 34.3526 - val_accuracy: 0.3331 - val_loss: 289.8344
Epoch 4/10
86/86                 31s 363ms/step -
accuracy: 0.7362 - loss: 34.4254 - val_accuracy: 0.3331 - val_loss: 300.9718
Epoch 5/10
86/86                 31s 358ms/step -
accuracy: 0.7618 - loss: 28.6019 - val_accuracy: 0.3331 - val_loss: 231.0389
Epoch 6/10
86/86                 31s 367ms/step -
accuracy: 0.8011 - loss: 21.7279 - val_accuracy: 0.3331 - val_loss: 166.1335
Epoch 7/10
86/86                 32s 371ms/step -
accuracy: 0.7854 - loss: 20.1342 - val_accuracy: 0.4508 - val_loss: 55.5544
Epoch 8/10
```

```
86/86             31s 359ms/step -
accuracy: 0.8215 - loss: 14.2369 - val_accuracy: 0.6654 - val_loss: 33.6830
Epoch 9/10
86/86             31s 356ms/step -
accuracy: 0.8148 - loss: 10.7389 - val_accuracy: 0.6902 - val_loss: 24.1362
Epoch 10/10
86/86             29s 343ms/step -
accuracy: 0.8169 - loss: 9.5812 - val_accuracy: 0.7088 - val_loss: 32.8145
41/41             2s 53ms/step -
accuracy: 0.7101 - loss: 31.6755
```

[41]:

```
#Model 2:
# AlexNet Model

alexnet_model = models.Sequential([
    # First Convolutional Layer
    layers.Conv2D(96, (3, 3), activation='relu', padding='same',␣
 ↪input_shape=(224, 224, 3)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2), strides=2),

    # Second Convolutional Layer
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2), strides=2),

    # Third Convolutional Layer
    layers.Conv2D(384, (3, 3), activation='relu', padding='same'),

    # Fourth Convolutional Layer
    layers.Conv2D(384, (3, 3), activation='relu', padding='same'),

    # Fifth Convolutional Layer
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=2),

    layers.GlobalAveragePooling2D(),

    # Fully Connected Layer 1
    layers.Dense(4096, activation='relu'),
    layers.Dropout(0.5),  # Dropout Layer

    # Fully Connected Layer 2
    layers.Dense(4096, activation='relu'),
    layers.Dropout(0.5),  # Dropout Layer

    # Output Layer
```

```
    layers.Dense(3, activation='softmax')
])

alexnet_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
alexnet_model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 224, 224, 96)	2,688
batch_normalization_4 (BatchNormalization)	(None, 224, 224, 96)	384
max_pooling2d_5 (MaxPooling2D)	(None, 112, 112, 96)	0
conv2d_6 (Conv2D)	(None, 112, 112, 256)	221,440
batch_normalization_5 (BatchNormalization)	(None, 112, 112, 256)	1,024
max_pooling2d_6 (MaxPooling2D)	(None, 56, 56, 256)	0
conv2d_7 (Conv2D)	(None, 56, 56, 384)	885,120
conv2d_8 (Conv2D)	(None, 56, 56, 384)	1,327,488
conv2d_9 (Conv2D)	(None, 56, 56, 256)	884,992
max_pooling2d_7 (MaxPooling2D)	(None, 28, 28, 256)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 256)	0
dense_4 (Dense)	(None, 4096)	1,052,672
dropout_1 (Dropout)	(None, 4096)	0
dense_5 (Dense)	(None, 4096)	16,781,312
dropout_2 (Dropout)	(None, 4096)	0
dense_6 (Dense)	(None, 3)	12,291

Total params: 21,169,411 (80.75 MB)

Trainable params: 21,168,707 (80.75 MB)

Non-trainable params: 704 (2.75 KB)

[42]:
```
alexnet_history = alexnet_model.fit(train_ds, epochs=10,␣
 ↪validation_data=(test_ds))
alexnet_test_loss, alexnet_test_acc = alexnet_model.evaluate(test_ds)
```

```
Epoch 1/10
86/86                201s 2s/step -
accuracy: 0.3908 - loss: 1.2197 - val_accuracy: 0.3331 - val_loss: 1.2050
Epoch 2/10
86/86                200s 2s/step -
accuracy: 0.6469 - loss: 0.7367 - val_accuracy: 0.3331 - val_loss: 1.2170
Epoch 3/10
86/86                197s 2s/step -
accuracy: 0.6729 - loss: 0.6943 - val_accuracy: 0.3331 - val_loss: 1.3854
Epoch 4/10
86/86                190s 2s/step -
accuracy: 0.6693 - loss: 0.6846 - val_accuracy: 0.3331 - val_loss: 1.5773
Epoch 5/10
86/86                197s 2s/step -
accuracy: 0.6707 - loss: 0.7504 - val_accuracy: 0.3331 - val_loss: 1.7915
Epoch 6/10
86/86                192s 2s/step -
accuracy: 0.6826 - loss: 0.7756 - val_accuracy: 0.5012 - val_loss: 1.1872
Epoch 7/10
86/86                516s 6s/step -
accuracy: 0.6633 - loss: 0.9655 - val_accuracy: 0.6274 - val_loss: 0.7961
Epoch 8/10
86/86                646s 8s/step -
accuracy: 0.6613 - loss: 0.9646 - val_accuracy: 0.5864 - val_loss: 1.4096
Epoch 9/10
86/86                505s 6s/step -
accuracy: 0.6672 - loss: 1.0738 - val_accuracy: 0.6375 - val_loss: 1.2013
Epoch 10/10
86/86                162s 2s/step -
accuracy: 0.6620 - loss: 1.4365 - val_accuracy: 0.7173 - val_loss: 0.9632
41/41                15s 371ms/step -
accuracy: 0.7040 - loss: 0.9986
```

[43]:
```
# Preprocess for Inception V3
from tensorflow.keras.applications.inception_v3 import preprocess_input as␣
↪inception_preprocess

def preprocess_and_resize(image, label):
    image = tf.image.resize(image, (224, 224))
    image = tf.cast(image * 255.0, tf.float32)
    image = inception_preprocess(image)
    return image, label

train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train_ds = train_ds.map(preprocess_and_resize, num_parallel_calls=tf.data.
↪AUTOTUNE)
train_ds = train_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)

test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test))
test_ds = test_ds.map(preprocess_and_resize, num_parallel_calls=tf.data.
↪AUTOTUNE)
test_ds = test_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

[44]:
```
# Model 3: Inception
# Inception V3 with transfer learning

base_inception = InceptionV3(include_top=False, weights='imagenet',␣
↪input_tensor=input_tensor)
x = base_inception.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.4)(x)
predictions = Dense(3, activation='softmax')(x)

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

inception_model = Model(inputs=input_tensor, outputs=predictions)
inception_model.compile(optimizer=Adam(learning_rate=0.0001),␣
↪loss='categorical_crossentropy', metrics=['accuracy'])
inception_model.summary()
```

Model: "functional_4"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0	-

conv2d_10 (Conv2D)	(None, 111, 111, 32)	864	input_layer_1[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 111, 111, 32)	96	conv2d_10[0][0]
activation (Activation)	(None, 111, 111, 32)	0	batch_normalizat…
conv2d_11 (Conv2D)	(None, 109, 109, 32)	9,216	activation[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 109, 109, 32)	96	conv2d_11[0][0]
activation_1 (Activation)	(None, 109, 109, 32)	0	batch_normalizat…
conv2d_12 (Conv2D)	(None, 109, 109, 64)	18,432	activation_1[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 109, 109, 64)	192	conv2d_12[0][0]
activation_2 (Activation)	(None, 109, 109, 64)	0	batch_normalizat…
max_pooling2d_8 (MaxPooling2D)	(None, 54, 54, 64)	0	activation_2[0][…
conv2d_13 (Conv2D)	(None, 54, 54, 80)	5,120	max_pooling2d_8[…
batch_normalizatio… (BatchNormalizatio…	(None, 54, 54, 80)	240	conv2d_13[0][0]
activation_3 (Activation)	(None, 54, 54, 80)	0	batch_normalizat…
conv2d_14 (Conv2D)	(None, 52, 52, 192)	138,240	activation_3[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 52, 52, 192)	576	conv2d_14[0][0]
activation_4 (Activation)	(None, 52, 52, 192)	0	batch_normalizat…

max_pooling2d_9 (MaxPooling2D)	(None, 25, 25, 192)	0	activation_4[0][…
conv2d_18 (Conv2D)	(None, 25, 25, 64)	12,288	max_pooling2d_9[…
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_18[0][0]
activation_8 (Activation)	(None, 25, 25, 64)	0	batch_normalizat…
conv2d_16 (Conv2D)	(None, 25, 25, 48)	9,216	max_pooling2d_9[…
conv2d_19 (Conv2D)	(None, 25, 25, 96)	55,296	activation_8[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 48)	144	conv2d_16[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_19[0][0]
activation_6 (Activation)	(None, 25, 25, 48)	0	batch_normalizat…
activation_9 (Activation)	(None, 25, 25, 96)	0	batch_normalizat…
average_pooling2d (AveragePooling2D)	(None, 25, 25, 192)	0	max_pooling2d_9[…
conv2d_15 (Conv2D)	(None, 25, 25, 64)	12,288	max_pooling2d_9[…
conv2d_17 (Conv2D)	(None, 25, 25, 64)	76,800	activation_6[0][…
conv2d_20 (Conv2D)	(None, 25, 25, 96)	82,944	activation_9[0][…
conv2d_21 (Conv2D)	(None, 25, 25, 32)	6,144	average_pooling2…
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_15[0][0]

batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_17[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_20[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 32)	96	conv2d_21[0][0]
activation_5 (Activation)	(None, 25, 25, 64)	0	batch_normalizat…
activation_7 (Activation)	(None, 25, 25, 64)	0	batch_normalizat…
activation_10 (Activation)	(None, 25, 25, 96)	0	batch_normalizat…
activation_11 (Activation)	(None, 25, 25, 32)	0	batch_normalizat…
mixed0 (Concatenate)	(None, 25, 25, 256)	0	activation_5[0][… activation_7[0][… activation_10[0]… activation_11[0]…
conv2d_25 (Conv2D)	(None, 25, 25, 64)	16,384	mixed0[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_25[0][0]
activation_15 (Activation)	(None, 25, 25, 64)	0	batch_normalizat…
conv2d_23 (Conv2D)	(None, 25, 25, 48)	12,288	mixed0[0][0]
conv2d_26 (Conv2D)	(None, 25, 25, 96)	55,296	activation_15[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 48)	144	conv2d_23[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_26[0][0]
activation_13	(None, 25, 25,	0	batch_normalizat…

```
(Activation)          48)

activation_16         (None, 25, 25,            0  batch_normalizat…
(Activation)          96)

average_pooling2d_1   (None, 25, 25,            0  mixed0[0][0]
(AveragePooling2D)    256)

conv2d_22 (Conv2D)    (None, 25, 25,       16,384  mixed0[0][0]
                      64)

conv2d_24 (Conv2D)    (None, 25, 25,       76,800  activation_13[0]…
                      64)

conv2d_27 (Conv2D)    (None, 25, 25,       82,944  activation_16[0]…
                      96)

conv2d_28 (Conv2D)    (None, 25, 25,       16,384  average_pooling2…
                      64)

batch_normalizatio…   (None, 25, 25,          192  conv2d_22[0][0]
(BatchNormalizatio…   64)

batch_normalizatio…   (None, 25, 25,          192  conv2d_24[0][0]
(BatchNormalizatio…   64)

batch_normalizatio…   (None, 25, 25,          288  conv2d_27[0][0]
(BatchNormalizatio…   96)

batch_normalizatio…   (None, 25, 25,          192  conv2d_28[0][0]
(BatchNormalizatio…   64)

activation_12         (None, 25, 25,            0  batch_normalizat…
(Activation)          64)

activation_14         (None, 25, 25,            0  batch_normalizat…
(Activation)          64)

activation_17         (None, 25, 25,            0  batch_normalizat…
(Activation)          96)

activation_18         (None, 25, 25,            0  batch_normalizat…
(Activation)          64)

mixed1                (None, 25, 25,            0  activation_12[0]…
(Concatenate)         288)                         activation_14[0]…
                                                   activation_17[0]…
                                                   activation_18[0]…
```

conv2d_32 (Conv2D)	(None, 25, 25, 64)	18,432	mixed1[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_32[0][0]
activation_22 (Activation)	(None, 25, 25, 64)	0	batch_normalizat…
conv2d_30 (Conv2D)	(None, 25, 25, 48)	13,824	mixed1[0][0]
conv2d_33 (Conv2D)	(None, 25, 25, 96)	55,296	activation_22[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 48)	144	conv2d_30[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_33[0][0]
activation_20 (Activation)	(None, 25, 25, 48)	0	batch_normalizat…
activation_23 (Activation)	(None, 25, 25, 96)	0	batch_normalizat…
average_pooling2d_2 (AveragePooling2D)	(None, 25, 25, 288)	0	mixed1[0][0]
conv2d_29 (Conv2D)	(None, 25, 25, 64)	18,432	mixed1[0][0]
conv2d_31 (Conv2D)	(None, 25, 25, 64)	76,800	activation_20[0]…
conv2d_34 (Conv2D)	(None, 25, 25, 96)	82,944	activation_23[0]…
conv2d_35 (Conv2D)	(None, 25, 25, 64)	18,432	average_pooling2…
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_29[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_31[0][0]

batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_34[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_35[0][0]
activation_19 (Activation)	(None, 25, 25, 64)	0	batch_normalizat…
activation_21 (Activation)	(None, 25, 25, 64)	0	batch_normalizat…
activation_24 (Activation)	(None, 25, 25, 96)	0	batch_normalizat…
activation_25 (Activation)	(None, 25, 25, 64)	0	batch_normalizat…
mixed2 (Concatenate)	(None, 25, 25, 288)	0	activation_19[0]… activation_21[0]… activation_24[0]… activation_25[0]…
conv2d_37 (Conv2D)	(None, 25, 25, 64)	18,432	mixed2[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_37[0][0]
activation_27 (Activation)	(None, 25, 25, 64)	0	batch_normalizat…
conv2d_38 (Conv2D)	(None, 25, 25, 96)	55,296	activation_27[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_38[0][0]
activation_28 (Activation)	(None, 25, 25, 96)	0	batch_normalizat…
conv2d_36 (Conv2D)	(None, 12, 12, 384)	995,328	mixed2[0][0]
conv2d_39 (Conv2D)	(None, 12, 12, 96)	82,944	activation_28[0]…

batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 384)	1,152	conv2d_36[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 96)	288	conv2d_39[0][0]
activation_26 (Activation)	(None, 12, 12, 384)	0	batch_normalizat…
activation_29 (Activation)	(None, 12, 12, 96)	0	batch_normalizat…
max_pooling2d_10 (MaxPooling2D)	(None, 12, 12, 288)	0	mixed2[0][0]
mixed3 (Concatenate)	(None, 12, 12, 768)	0	activation_26[0]… activation_29[0]… max_pooling2d_10…
conv2d_44 (Conv2D)	(None, 12, 12, 128)	98,304	mixed3[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_44[0][0]
activation_34 (Activation)	(None, 12, 12, 128)	0	batch_normalizat…
conv2d_45 (Conv2D)	(None, 12, 12, 128)	114,688	activation_34[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_45[0][0]
activation_35 (Activation)	(None, 12, 12, 128)	0	batch_normalizat…
conv2d_41 (Conv2D)	(None, 12, 12, 128)	98,304	mixed3[0][0]
conv2d_46 (Conv2D)	(None, 12, 12, 128)	114,688	activation_35[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_41[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_46[0][0]

activation_31 (Activation)	(None, 12, 12, 128)	0	batch_normalizat…
activation_36 (Activation)	(None, 12, 12, 128)	0	batch_normalizat…
conv2d_42 (Conv2D)	(None, 12, 12, 128)	114,688	activation_31[0]…
conv2d_47 (Conv2D)	(None, 12, 12, 128)	114,688	activation_36[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_42[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_47[0][0]
activation_32 (Activation)	(None, 12, 12, 128)	0	batch_normalizat…
activation_37 (Activation)	(None, 12, 12, 128)	0	batch_normalizat…
average_pooling2d_3 (AveragePooling2D)	(None, 12, 12, 768)	0	mixed3[0][0]
conv2d_40 (Conv2D)	(None, 12, 12, 192)	147,456	mixed3[0][0]
conv2d_43 (Conv2D)	(None, 12, 12, 192)	172,032	activation_32[0]…
conv2d_48 (Conv2D)	(None, 12, 12, 192)	172,032	activation_37[0]…
conv2d_49 (Conv2D)	(None, 12, 12, 192)	147,456	average_pooling2…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_40[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_43[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_48[0][0]

batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_49[0][0]
activation_30 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_33 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_38 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_39 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
mixed4 (Concatenate)	(None, 12, 12, 768)	0	activation_30[0]… activation_33[0]… activation_38[0]… activation_39[0]…
conv2d_54 (Conv2D)	(None, 12, 12, 160)	122,880	mixed4[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_54[0][0]
activation_44 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
conv2d_55 (Conv2D)	(None, 12, 12, 160)	179,200	activation_44[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_55[0][0]
activation_45 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
conv2d_51 (Conv2D)	(None, 12, 12, 160)	122,880	mixed4[0][0]
conv2d_56 (Conv2D)	(None, 12, 12, 160)	179,200	activation_45[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_51[0][0]

batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_56[0][0]
activation_41 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
activation_46 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
conv2d_52 (Conv2D)	(None, 12, 12, 160)	179,200	activation_41[0]…
conv2d_57 (Conv2D)	(None, 12, 12, 160)	179,200	activation_46[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_52[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_57[0][0]
activation_42 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
activation_47 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
average_pooling2d_4 (AveragePooling2D)	(None, 12, 12, 768)	0	mixed4[0][0]
conv2d_50 (Conv2D)	(None, 12, 12, 192)	147,456	mixed4[0][0]
conv2d_53 (Conv2D)	(None, 12, 12, 192)	215,040	activation_42[0]…
conv2d_58 (Conv2D)	(None, 12, 12, 192)	215,040	activation_47[0]…
conv2d_59 (Conv2D)	(None, 12, 12, 192)	147,456	average_pooling2…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_50[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_53[0][0]

batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_58[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_59[0][0]
activation_40 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_43 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_48 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_49 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
mixed5 (Concatenate)	(None, 12, 12, 768)	0	activation_40[0]… activation_43[0]… activation_48[0]… activation_49[0]…
conv2d_64 (Conv2D)	(None, 12, 12, 160)	122,880	mixed5[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_64[0][0]
activation_54 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
conv2d_65 (Conv2D)	(None, 12, 12, 160)	179,200	activation_54[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_65[0][0]
activation_55 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
conv2d_61 (Conv2D)	(None, 12, 12, 160)	122,880	mixed5[0][0]
conv2d_66 (Conv2D)	(None, 12, 12, 160)	179,200	activation_55[0]…
batch_normalizatio…	(None, 12, 12,	480	conv2d_61[0][0]

(BatchNormalizatio…	160)		
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_66[0][0]
activation_51 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
activation_56 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
conv2d_62 (Conv2D)	(None, 12, 12, 160)	179,200	activation_51[0]…
conv2d_67 (Conv2D)	(None, 12, 12, 160)	179,200	activation_56[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_62[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_67[0][0]
activation_52 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
activation_57 (Activation)	(None, 12, 12, 160)	0	batch_normalizat…
average_pooling2d_5 (AveragePooling2D)	(None, 12, 12, 768)	0	mixed5[0][0]
conv2d_60 (Conv2D)	(None, 12, 12, 192)	147,456	mixed5[0][0]
conv2d_63 (Conv2D)	(None, 12, 12, 192)	215,040	activation_52[0]…
conv2d_68 (Conv2D)	(None, 12, 12, 192)	215,040	activation_57[0]…
conv2d_69 (Conv2D)	(None, 12, 12, 192)	147,456	average_pooling2…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_60[0][0]
batch_normalizatio…	(None, 12, 12,	576	conv2d_63[0][0]

(BatchNormalizatio…	192)		
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_68[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_69[0][0]
activation_50 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_53 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_58 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_59 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
mixed6 (Concatenate)	(None, 12, 12, 768)	0	activation_50[0]… activation_53[0]… activation_58[0]… activation_59[0]…
conv2d_74 (Conv2D)	(None, 12, 12, 192)	147,456	mixed6[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_74[0][0]
activation_64 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
conv2d_75 (Conv2D)	(None, 12, 12, 192)	258,048	activation_64[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_75[0][0]
activation_65 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
conv2d_71 (Conv2D)	(None, 12, 12, 192)	147,456	mixed6[0][0]
conv2d_76 (Conv2D)	(None, 12, 12, 192)	258,048	activation_65[0]…

batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_71[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_76[0][0]
activation_61 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_66 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
conv2d_72 (Conv2D)	(None, 12, 12, 192)	258,048	activation_61[0]…
conv2d_77 (Conv2D)	(None, 12, 12, 192)	258,048	activation_66[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_72[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_77[0][0]
activation_62 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_67 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
average_pooling2d_6 (AveragePooling2D)	(None, 12, 12, 768)	0	mixed6[0][0]
conv2d_70 (Conv2D)	(None, 12, 12, 192)	147,456	mixed6[0][0]
conv2d_73 (Conv2D)	(None, 12, 12, 192)	258,048	activation_62[0]…
conv2d_78 (Conv2D)	(None, 12, 12, 192)	258,048	activation_67[0]…
conv2d_79 (Conv2D)	(None, 12, 12, 192)	147,456	average_pooling2…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_70[0][0]

batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_73[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_78[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_79[0][0]
activation_60 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_63 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_68 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_69 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
mixed7 (Concatenate)	(None, 12, 12, 768)	0	activation_60[0]… activation_63[0]… activation_68[0]… activation_69[0]…
conv2d_82 (Conv2D)	(None, 12, 12, 192)	147,456	mixed7[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_82[0][0]
activation_72 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
conv2d_83 (Conv2D)	(None, 12, 12, 192)	258,048	activation_72[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_83[0][0]
activation_73 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
conv2d_80 (Conv2D)	(None, 12, 12, 192)	147,456	mixed7[0][0]

conv2d_84 (Conv2D)	(None, 12, 12, 192)	258,048	activation_73[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_80[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_84[0][0]
activation_70 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
activation_74 (Activation)	(None, 12, 12, 192)	0	batch_normalizat…
conv2d_81 (Conv2D)	(None, 5, 5, 320)	552,960	activation_70[0]…
conv2d_85 (Conv2D)	(None, 5, 5, 192)	331,776	activation_74[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 320)	960	conv2d_81[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 192)	576	conv2d_85[0][0]
activation_71 (Activation)	(None, 5, 5, 320)	0	batch_normalizat…
activation_75 (Activation)	(None, 5, 5, 192)	0	batch_normalizat…
max_pooling2d_11 (MaxPooling2D)	(None, 5, 5, 768)	0	mixed7[0][0]
mixed8 (Concatenate)	(None, 5, 5, 1280)	0	activation_71[0]… activation_75[0]… max_pooling2d_11…
conv2d_90 (Conv2D)	(None, 5, 5, 448)	573,440	mixed8[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 448)	1,344	conv2d_90[0][0]
activation_80 (Activation)	(None, 5, 5, 448)	0	batch_normalizat…
conv2d_87 (Conv2D)	(None, 5, 5, 384)	491,520	mixed8[0][0]

conv2d_91 (Conv2D)	(None, 5, 5, 384)	1,548,288	activation_80[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_87[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_91[0][0]
activation_77 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…
activation_81 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…
conv2d_88 (Conv2D)	(None, 5, 5, 384)	442,368	activation_77[0]…
conv2d_89 (Conv2D)	(None, 5, 5, 384)	442,368	activation_77[0]…
conv2d_92 (Conv2D)	(None, 5, 5, 384)	442,368	activation_81[0]…
conv2d_93 (Conv2D)	(None, 5, 5, 384)	442,368	activation_81[0]…
average_pooling2d_7 (AveragePooling2D)	(None, 5, 5, 1280)	0	mixed8[0][0]
conv2d_86 (Conv2D)	(None, 5, 5, 320)	409,600	mixed8[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_88[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_89[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_92[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_93[0][0]
conv2d_94 (Conv2D)	(None, 5, 5, 192)	245,760	average_pooling2…
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 320)	960	conv2d_86[0][0]
activation_78 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…
activation_79	(None, 5, 5, 384)	0	batch_normalizat…

(Activation)			
activation_82 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…
activation_83 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 192)	576	conv2d_94[0][0]
activation_76 (Activation)	(None, 5, 5, 320)	0	batch_normalizat…
mixed9_0 (Concatenate)	(None, 5, 5, 768)	0	activation_78[0]… activation_79[0]…
concatenate (Concatenate)	(None, 5, 5, 768)	0	activation_82[0]… activation_83[0]…
activation_84 (Activation)	(None, 5, 5, 192)	0	batch_normalizat…
mixed9 (Concatenate)	(None, 5, 5, 2048)	0	activation_76[0]… mixed9_0[0][0], concatenate[0][0… activation_84[0]…
conv2d_99 (Conv2D)	(None, 5, 5, 448)	917,504	mixed9[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 448)	1,344	conv2d_99[0][0]
activation_89 (Activation)	(None, 5, 5, 448)	0	batch_normalizat…
conv2d_96 (Conv2D)	(None, 5, 5, 384)	786,432	mixed9[0][0]
conv2d_100 (Conv2D)	(None, 5, 5, 384)	1,548,288	activation_89[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_96[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_100[0][0]
activation_86 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…

activation_90 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…
conv2d_97 (Conv2D)	(None, 5, 5, 384)	442,368	activation_86[0]…
conv2d_98 (Conv2D)	(None, 5, 5, 384)	442,368	activation_86[0]…
conv2d_101 (Conv2D)	(None, 5, 5, 384)	442,368	activation_90[0]…
conv2d_102 (Conv2D)	(None, 5, 5, 384)	442,368	activation_90[0]…
average_pooling2d_8 (AveragePooling2D)	(None, 5, 5, 2048)	0	mixed9[0][0]
conv2d_95 (Conv2D)	(None, 5, 5, 320)	655,360	mixed9[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_97[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_98[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_101[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_102[0][0]
conv2d_103 (Conv2D)	(None, 5, 5, 192)	393,216	average_pooling2…
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 320)	960	conv2d_95[0][0]
activation_87 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…
activation_88 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…
activation_91 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…
activation_92 (Activation)	(None, 5, 5, 384)	0	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 192)	576	conv2d_103[0][0]

activation_85 (Activation)	(None, 5, 5, 320)	0	batch_normalizat…
mixed9_1 (Concatenate)	(None, 5, 5, 768)	0	activation_87[0]… activation_88[0]…
concatenate_1 (Concatenate)	(None, 5, 5, 768)	0	activation_91[0]… activation_92[0]…
activation_93 (Activation)	(None, 5, 5, 192)	0	batch_normalizat…
mixed10 (Concatenate)	(None, 5, 5, 2048)	0	activation_85[0]… mixed9_1[0][0], concatenate_1[0]… activation_93[0]…
global_average_poo… (GlobalAveragePool…	(None, 2048)	0	mixed10[0][0]
dense_7 (Dense)	(None, 512)	1,049,088	global_average_p…
dropout_3 (Dropout)	(None, 512)	0	dense_7[0][0]
dense_8 (Dense)	(None, 3)	1,539	dropout_3[0][0]

Total params: 22,853,411 (87.18 MB)

Trainable params: 1,050,627 (4.01 MB)

Non-trainable params: 21,802,784 (83.17 MB)

[45]:
```
inception_history = inception_model.fit(train_ds, epochs=10,␣
↪validation_data=test_ds)
inception_test_loss, inception_test_acc = inception_model.evaluate(test_ds)
```

```
Epoch 1/10
86/86               30s 291ms/step -
accuracy: 0.5769 - loss: 1.0591 - val_accuracy: 0.8025 - val_loss: 0.4688
Epoch 2/10
86/86               25s 286ms/step -
accuracy: 0.7621 - loss: 0.5917 - val_accuracy: 0.8389 - val_loss: 0.3907
Epoch 3/10
```

```
86/86                25s 287ms/step -
accuracy: 0.7972 - loss: 0.5099 - val_accuracy: 0.8327 - val_loss: 0.3720
Epoch 4/10
86/86                23s 270ms/step -
accuracy: 0.8103 - loss: 0.4731 - val_accuracy: 0.8606 - val_loss: 0.3392
Epoch 5/10
86/86                23s 263ms/step -
accuracy: 0.8276 - loss: 0.3998 - val_accuracy: 0.8629 - val_loss: 0.3312
Epoch 6/10
86/86                23s 269ms/step -
accuracy: 0.8371 - loss: 0.4361 - val_accuracy: 0.8699 - val_loss: 0.3128
Epoch 7/10
86/86                23s 265ms/step -
accuracy: 0.8355 - loss: 0.3904 - val_accuracy: 0.8683 - val_loss: 0.3132
Epoch 8/10
86/86                23s 273ms/step -
accuracy: 0.8597 - loss: 0.3525 - val_accuracy: 0.8761 - val_loss: 0.2981
Epoch 9/10
86/86                26s 298ms/step -
accuracy: 0.8406 - loss: 0.3614 - val_accuracy: 0.8722 - val_loss: 0.3105
Epoch 10/10
86/86                30s 351ms/step -
accuracy: 0.8615 - loss: 0.3316 - val_accuracy: 0.8722 - val_loss: 0.3107
41/41                11s 275ms/step -
accuracy: 0.8792 - loss: 0.3005
```

0.6 5. Performance Comparison

Evaluate all models on the same test set.

Highlight the model that achieved the best test performance.

Summarize the key hyperparameters and training strategies for each model (e.g., learning rate, batch size, number of epochs, optimizer).

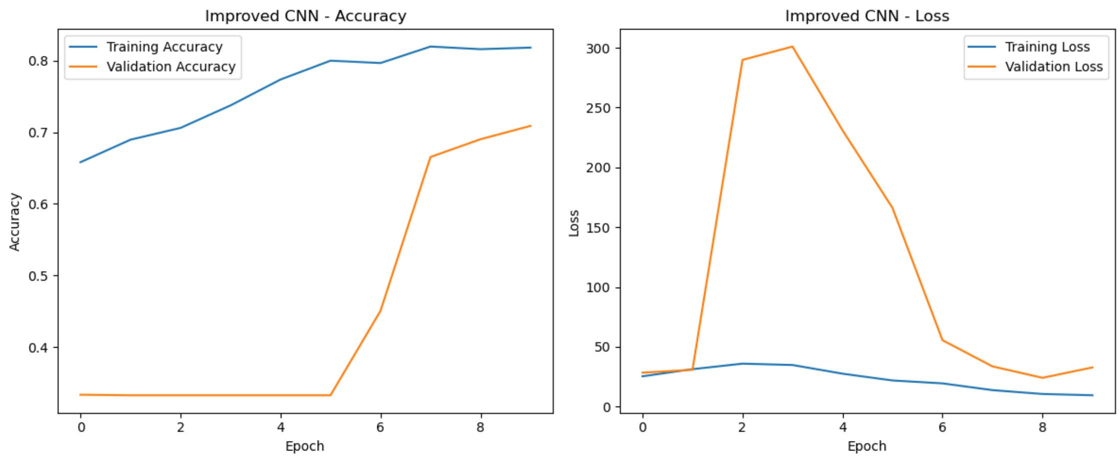
Include plots such as training/validation loss and accuracy over epochs.

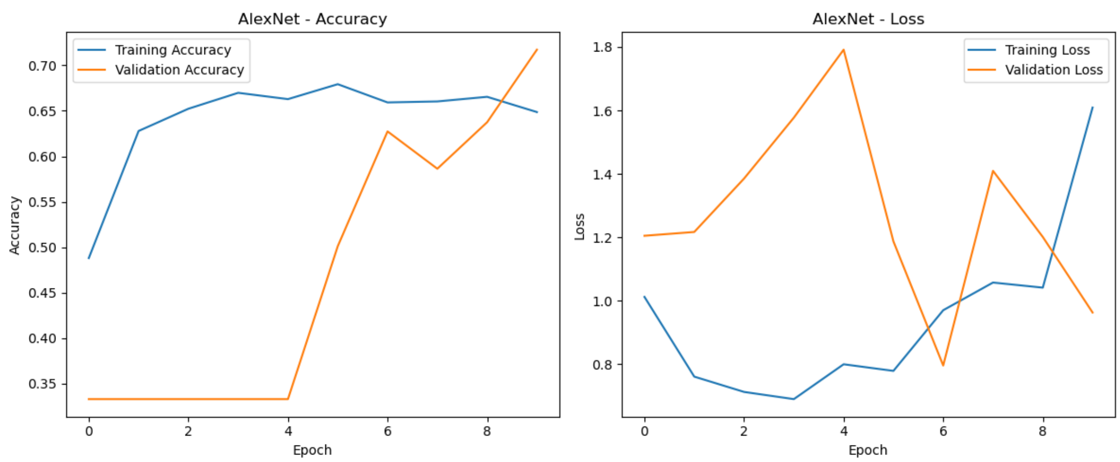
[47]:
```python
comparison_df = pd.DataFrame({
    'Model': ['Improved CNN', 'AlexNet', 'Inception V3'],
    'Test Accuracy': [improved_test_acc, alexnet_test_acc, inception_test_acc],
    'Epochs': [10, 10, 10],
    'Optimizer': ['Adam', 'Adam', 'Adam(learning_rate=0.0001)'],
    'Batch Size': [32, 32, 32]
})

display(comparison_df)
```

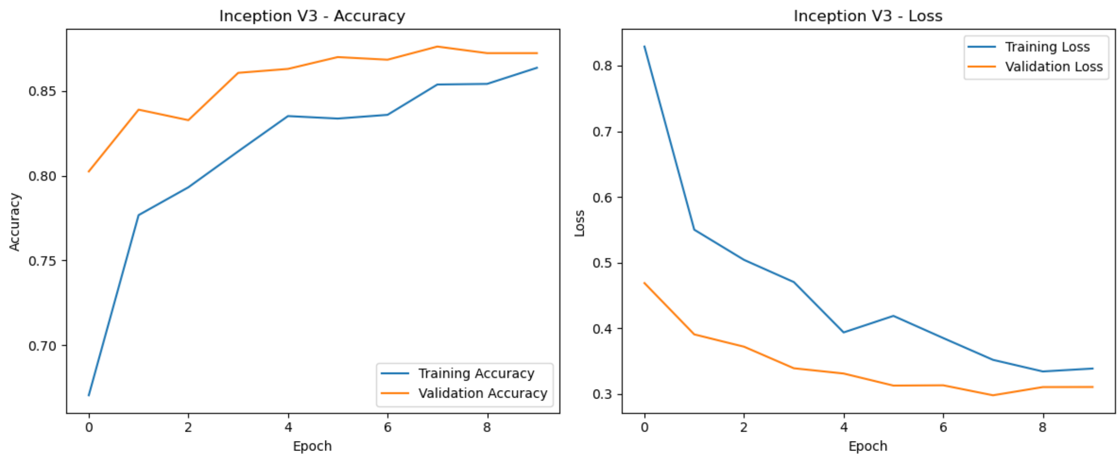
	Model	Test Accuracy	Epochs	Optimizer	Batch Size
0	Improved CNN	0.708753	10	Adam	32
1	AlexNet	0.717273	10	Adam	32

2	Inception V3	0.872192	10	Adam(learning_rate=0.0001)	32

[48]:
```python
# Training and Validation Performance Plot
plot_training(improved_history, 'Improved CNN')
plot_training(alexnet_history, 'AlexNet')
plot_training(inception_history, 'Inception V3')
```







0.7 6. Augmentation

For at least one model, re-train it using data augmentation techniques.

Describe the types of augmentations used (e.g., flipping, cropping, rotation) and how they affected performance.

We will re-train the Improved CNN model to see it can outperform Inception V3 through data augmentations. We applied the following augmentations: - Randomly rotating images by up to 10 degrees, either clockwise or counterclockwise - Randomly shifting images horizontally by up to 5% of the total width - Randomly shifting images vertically by up to 5% of the total height - Disabling random horizontal flipping of images, as that could create anatomically incorrect images - Randomly zooming images in or out by up to 5%

These augmentations will increase the size of the training data through artificial variations. This improve model generalization by forcing it to learn features that are consistent across the transformations.

[51]:
```
# Data Augmentation Example
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=False, #As flipping the image would be anatomically␣
 ↪incorrect.
    fill_mode='nearest'
)
```

[52]:
```
# Redefine the model for augmented data
augmented_model = improved_cnn(input_shape=(192, 192, 3), num_classes=3)
augmented_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

```
/opt/anaconda3/lib/python3.10/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`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)
```

[53]:
```
augmented_history = augmented_model.fit(
     datagen.flow(X_train, y_train),
     epochs=10,
     validation_data=(X_test, y_test)
 )

augmented_test_loss, augmented_test_acc = augmented_model.evaluate(X_test,␣
↪y_test)
```

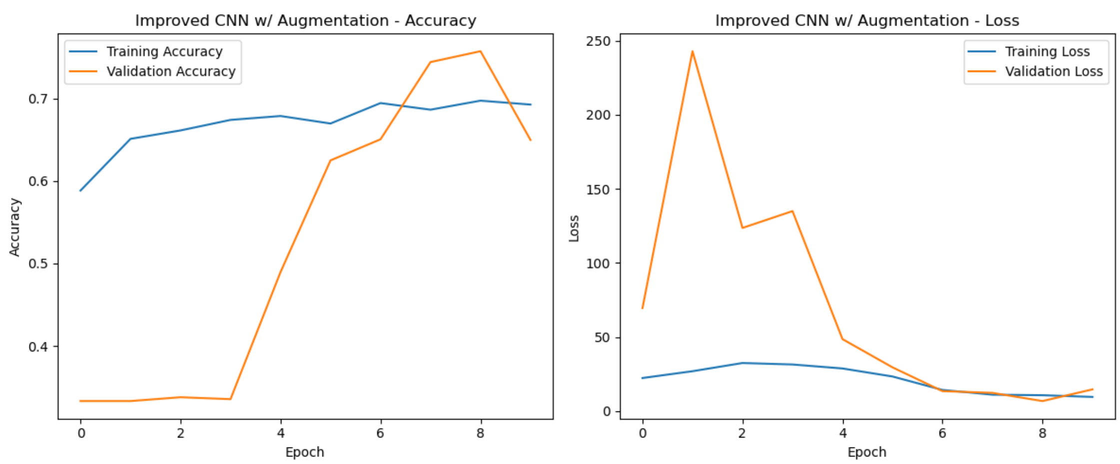
```
/opt/anaconda3/lib/python3.10/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

```
Epoch 1/10
86/86                 48s 529ms/step -
accuracy: 0.5573 - loss: 21.4972 - val_accuracy: 0.3331 - val_loss: 69.5498
Epoch 2/10
86/86                 35s 405ms/step -
accuracy: 0.6520 - loss: 24.7814 - val_accuracy: 0.3331 - val_loss: 242.8539
Epoch 3/10
86/86                 33s 386ms/step -
accuracy: 0.6722 - loss: 30.0041 - val_accuracy: 0.3377 - val_loss: 123.6023
Epoch 4/10
86/86                 33s 377ms/step -
accuracy: 0.6723 - loss: 32.2163 - val_accuracy: 0.3354 - val_loss: 134.9595
Epoch 5/10
86/86                 34s 388ms/step -
accuracy: 0.6789 - loss: 28.2994 - val_accuracy: 0.4895 - val_loss: 48.5694
Epoch 6/10
86/86                 65s 747ms/step -
accuracy: 0.6651 - loss: 24.8375 - val_accuracy: 0.6251 - val_loss: 29.5197
Epoch 7/10
86/86                 74s 857ms/step -
```

```
accuracy: 0.6949 - loss: 15.8522 - val_accuracy: 0.6507 - val_loss: 13.4765
Epoch 8/10
86/86                 64s 738ms/step -
accuracy: 0.6796 - loss: 11.8227 - val_accuracy: 0.7444 - val_loss: 12.2958
Epoch 9/10
86/86                 40s 466ms/step -
accuracy: 0.6910 - loss: 9.6342 - val_accuracy: 0.7576 - val_loss: 6.7764
Epoch 10/10
86/86                 33s 382ms/step -
accuracy: 0.6955 - loss: 9.1059 - val_accuracy: 0.6499 - val_loss: 14.6561
41/41                 2s 43ms/step -
accuracy: 0.6570 - loss: 13.1106
```

[54]:

```
# Plot training history
plot_training(augmented_history, 'Improved CNN w/ Augmentation')

# Evaluate the model on test data
print(f"Improved CNN w/ Augmentation Test Accuracy: {augmented_test_acc*100:.
  ↪2f}%")
```




```
Improved CNN w/ Augmentation Test Accuracy: 64.99%
```

0.8 7. Interpretability & Insights

Reflect on which model performed best and why.

Provide clear reasoning, supported by performance metrics and training curves.

*Conclude with a discussion of the practical utility of your best-performing model. * Who would benefit from using this model? * In what types of real-world scenarios would your solution be useful?*

It appears that Inception V3 performed the best out of our 3 models (Improved CNN, AlexNet, and Inception V3), achieving test accuracy of 87%. Even with augmentation, the CNN model had fewer parameters to learn the subtle variations. This is evident in the erratic training curves for both CNN and AlexNet, compared to the smoother curve for Inception V3. Contrasting the additional architectures we created, it was difficult to substantially improve the performance from the baseline model. As predicted, additional epochs did not appear to have a substantial impact on the actual accuracy; though what's more surprising is that adding additional architecture to increase the depth of the model did not overwhelmingly perform the performance. Furthermore, the "Improved CNN model" retrained with augmented images seemed to perform the least well of all of them. Potentially because the model we proposed is not "deep" enough, or have the appropriate regularization technique to learn the features that generalize well across the augmented data.

Out of all the models, ResNet50 with fine-tuning performed the best. Through residual learning and fine-tuning, the model was able to adapat its pre-trained weights to medical images. This may be the most important results as it shows the power that utilizing a powerful pre-trained model, even in a new context, can have on developing an image classifier in a novel context. Fine-tuning only served to improve this performance and to an impressive degree.

From this classification exercise, we can provide insights on how we can apply ML techniques specific to each domain. Further study using this dataset would be able to to aid healthcare professionals in interpretting radiology reports and provide diagnostic support. Without knowing the success rates of average doctors in their capacity to identify positive cases, it's unclear the breadth by which this model may be applied. At the least, however, this may be extremely helpful in hospital systems where the volume of cases may overwhelm doctors capacity to manually review, showing them the most likely images to flag incidents.