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

In [3]:
```
# 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, Activatio
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, SGD, Adagrad, Adadelta,
from tensorflow.keras.applications import ResNet50, InceptionV3
```

0. Loading Dataset

In [5]:
```
# Import data

import os
base_path = "/Users/florakwon/Desktop/Spring 2025/QMSS 5074 - Adv ML/P

# For Colab, use:
# from google.colab import drive
# drive.mount('/content/drive')
# !unzip /content/drive/MyDrive/covid_radiography_data/COVID-19_Radiog
```

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?

In [7]:
```
# Extracting all filenames iteratively
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 i
    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 interpretting radiology reports and provide diagnostic support. From general ML knowledge perspective, it will also improve pattern recognition and its applications.

In [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]
```

In [10]:
```
# Import image, load to array of shape height, width, channels, then m
# Write preprocessor that will match up with model's expected input sh

from keras.preprocessing import image
from PIL import Image

def preprocessor(img_path):
        img = Image.open(img_path).convert("RGB").resize((192,192)) #
        img = (np.float32(img)-1.)/(255-1.) # Min max transformation
        img=img.reshape((192,192,3)) # Create final shape as array wit
        return img
```

In [11]:
```
# Import image files iteratively and preprocess them into array of cor

# 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
preprocessed_image_data=list(map(preprocessor,image_filepaths ))

# Object needs to be an array rather than a list for Keras (map return
X= np.array(preprocessed_image_data) # Assigning to X to highlight tha
```

In [12]:
```
len(image_filepaths)
```

Out[12]: 4032

In [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
```

In [14]:
```
len(fnames[2])
```

Out[14]: 1344

In [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
```

```
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
```

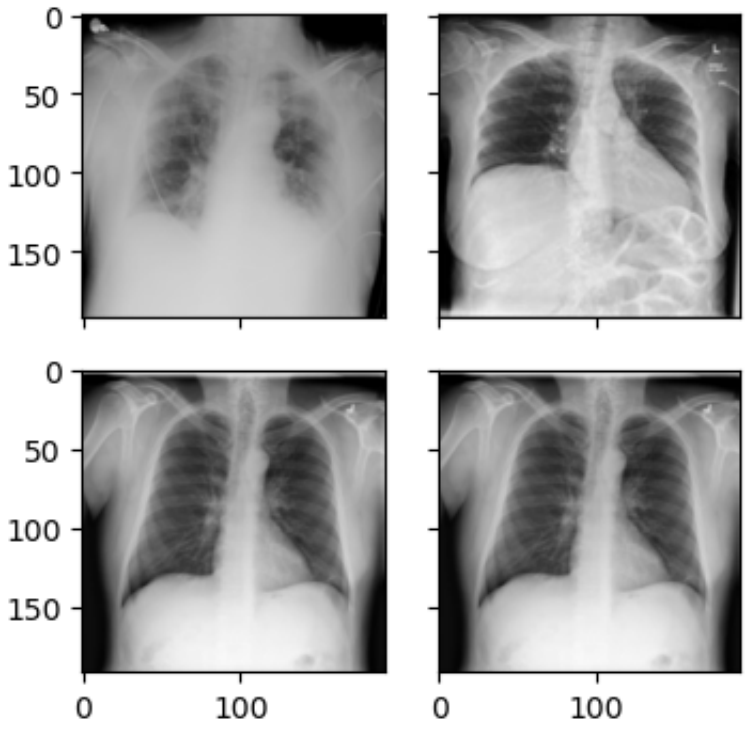
	COVID	NORMAL	PNEUMONIA
0	True	False	False
1	True	False	False
2	True	False	False
3	True	False	False
4	True	False	False
...	...	...	...
4027	False	False	True
4028	False	False	True
4029	False	False	True
4030	False	False	True
4031	False	False	True

4032 rows × 3 columns

In [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..0.842519
7].
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..0.964566
95].
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..0.980314
97].
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..0.980314
97].
```



In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y

X_test.shape, y_test.shape
```

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

In [18]:
```
# Clear objects from memory
```

```
del(X)
del(y)
del(preprocessed_image_data)
```

In [19]:
```
#Save data to be able to reload quickly if memory crashes or if you ru
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)
```

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.

In [21]:
```
# 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_sh
        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_crossentrop
baseline_model.summary()
```

```
/opt/anaconda3/lib/python3.10/site-packages/keras/src/layers/convolutio
nal/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)
2025-04-20 17:37:43.292588: I metal_plugin/src/device/metal_device.cc:1
154] Metal device set to: Apple M3
2025-04-20 17:37:43.292651: I metal_plugin/src/device/metal_device.cc:2
96] systemMemory: 16.00 GB
2025-04-20 17:37:43.292662: I metal_plugin/src/device/metal_device.cc:3
13] maxCacheSize: 5.33 GB
WARNING: All log messages before absl::InitializeLog() is called are wr
itten to STDERR
I0000 00:00:1745185063.293500 6847841 pluggable_device_factory.cc:305]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. You
r kernel may not have been built with NUMA support.
I0000 00:00:1745185063.293972 6847841 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	
conv2d (Conv2D)	(None, 192, 192, 32)	
max_pooling2d (MaxPooling2D)	(None, 96, 96, 32)	
flatten (Flatten)	(None, 294912)	
dense (Dense)	(None, 3)	

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.

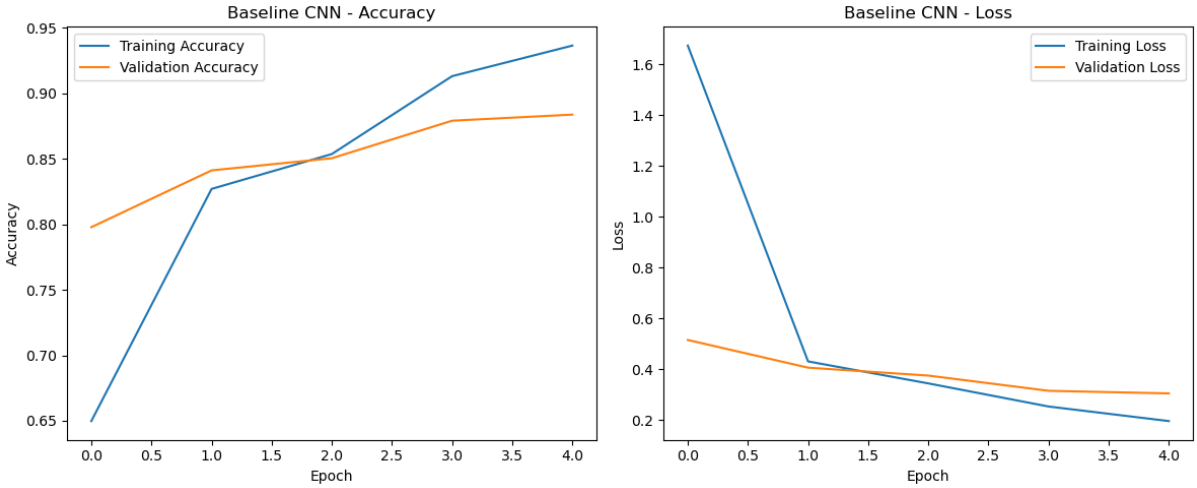
In [23]:
```
baseline_history = baseline_model.fit(X_train, y_train, epochs=5, batc
```

```
Epoch 1/5
2025-04-20 17:37:44.428416: I tensorflow/core/grappler/optimizers/custo
m_graph_optimizer_registry.cc:117] Plugin optimizer for device_type GPU
is enabled.
43/43 ━━━━━━━━━━━━━━━━━━━━ 6s 95ms/step - accuracy: 0.5434 - loss: 3.52
46 - val_accuracy: 0.7978 - val_loss: 0.5161
Epoch 2/5
43/43 ━━━━━━━━━━━━━━━━━━━━ 3s 76ms/step - accuracy: 0.8112 - loss: 0.45
63 - val_accuracy: 0.8412 - val_loss: 0.4072
Epoch 3/5
43/43 ━━━━━━━━━━━━━━━━━━━━ 3s 73ms/step - accuracy: 0.8385 - loss: 0.37
17 - val_accuracy: 0.8505 - val_loss: 0.3760
Epoch 4/5
43/43 ━━━━━━━━━━━━━━━━━━━━ 3s 70ms/step - accuracy: 0.9145 - loss: 0.25
95 - val_accuracy: 0.8792 - val_loss: 0.3161
Epoch 5/5
43/43 ━━━━━━━━━━━━━━━━━━━━ 3s 70ms/step - accuracy: 0.9278 - loss: 0.21
65 - val_accuracy: 0.8838 - val_loss: 0.3059
```

In [24]:
```
# 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()
```

In [25]:
```
# 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
print(f"Baseline CNN Test Accuracy: {baseline_test_acc*100:.2f}%")
```




```
41/41 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.8678 - loss: 0.32
43
Baseline CNN Test Accuracy: 88.38%
```

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.

In [27]:
```
from tensorflow.keras.applications.resnet50 import preprocess_input as

# 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.d
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.dat
test_ds = test_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

In [28]:

```
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_te
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='catego
resnet_model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connecte
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0	–
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_la
conv1_conv (Conv2D)	(None, 112, 112, 64)	9,472	conv1_pa
conv1_bn (BatchNormalizatio…	(None, 112, 112, 64)	256	conv1_cc
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_br
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_re
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pa

conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4,160	pool1_pc
conv2_block1_1_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_bl
conv2_block1_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_bl
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_bl
conv2_block1_2_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_bl
conv2_block1_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_bl
conv2_block1_0_conv (Conv2D)	(None, 56, 56, 256)	16,640	pool1_pc
conv2_block1_3_conv (Conv2D)	(None, 56, 56, 256)	16,640	conv2_bl
conv2_block1_0_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_bl
conv2_block1_3_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_bl
conv2_block1_add (Add)	(None, 56, 56, 256)	0	conv2_bl conv2_bl
conv2_block1_out (Activation)	(None, 56, 56, 256)	0	conv2_bl
conv2_block2_1_conv (Conv2D)	(None, 56, 56, 64)	16,448	conv2_bl
conv2_block2_1_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_bl
conv2_block2_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_bl
conv2_block2_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_bl
conv2_block2_2_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_bl
conv2_block2_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_bl
conv2_block2_3_conv (Conv2D)	(None, 56, 56, 256)	16,640	conv2_bl

conv2_block2_3_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_bl
conv2_block2_add (Add)	(None, 56, 56, 256)	0	conv2_bl conv2_bl
conv2_block2_out (Activation)	(None, 56, 56, 256)	0	conv2_bl
conv2_block3_1_conv (Conv2D)	(None, 56, 56, 64)	16,448	conv2_bl
conv2_block3_1_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_bl
conv2_block3_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_bl
conv2_block3_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_bl
conv2_block3_2_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_bl
conv2_block3_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_bl
conv2_block3_3_conv (Conv2D)	(None, 56, 56, 256)	16,640	conv2_bl
conv2_block3_3_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_bl
conv2_block3_add (Add)	(None, 56, 56, 256)	0	conv2_bl conv2_bl
conv2_block3_out (Activation)	(None, 56, 56, 256)	0	conv2_bl
conv3_block1_1_conv (Conv2D)	(None, 28, 28, 128)	32,896	conv2_bl
conv3_block1_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_bl
conv3_block1_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_bl
conv3_block1_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_bl
conv3_block1_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_bl
conv3_block1_2_relu	(None, 28, 28,	0	conv3_bl

(Activation)	128)		
conv3_block1_0_conv (Conv2D)	(None, 28, 28, 512)	131,584	conv2_bl
conv3_block1_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_bl
conv3_block1_0_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_bl
conv3_block1_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_bl
conv3_block1_add (Add)	(None, 28, 28, 512)	0	conv3_bl conv3_bl
conv3_block1_out (Activation)	(None, 28, 28, 512)	0	conv3_bl
conv3_block2_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_bl
conv3_block2_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_bl
conv3_block2_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_bl
conv3_block2_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_bl
conv3_block2_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_bl
conv3_block2_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_bl
conv3_block2_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_bl
conv3_block2_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_bl
conv3_block2_add (Add)	(None, 28, 28, 512)	0	conv3_bl conv3_bl
conv3_block2_out (Activation)	(None, 28, 28, 512)	0	conv3_bl
conv3_block3_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_bl
conv3_block3_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_bl

conv3_block3_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_bl
conv3_block3_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_bl
conv3_block3_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_bl
conv3_block3_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_bl
conv3_block3_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_bl
conv3_block3_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_bl
conv3_block3_add (Add)	(None, 28, 28, 512)	0	conv3_bl conv3_bl
conv3_block3_out (Activation)	(None, 28, 28, 512)	0	conv3_bl
conv3_block4_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_bl
conv3_block4_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_bl
conv3_block4_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_bl
conv3_block4_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_bl
conv3_block4_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_bl
conv3_block4_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_bl
conv3_block4_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_bl
conv3_block4_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_bl
conv3_block4_add (Add)	(None, 28, 28, 512)	0	conv3_bl conv3_bl
conv3_block4_out (Activation)	(None, 28, 28, 512)	0	conv3_bl
conv4_block1_1_conv (Conv2D)	(None, 14, 14, 256)	131,328	conv3_bl

conv4_block1_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block1_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block1_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_bl
conv4_block1_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block1_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block1_0_conv (Conv2D)	(None, 14, 14, 1024)	525,312	conv3_bl
conv4_block1_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_bl
conv4_block1_0_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_bl
conv4_block1_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_bl
conv4_block1_add (Add)	(None, 14, 14, 1024)	0	conv4_bl conv4_bl
conv4_block1_out (Activation)	(None, 14, 14, 1024)	0	conv4_bl
conv4_block2_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_bl
conv4_block2_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block2_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block2_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_bl
conv4_block2_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block2_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block2_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_bl
conv4_block2_3_bn	(None, 14, 14,	4,096	conv4_bl

(BatchNormalizatio…	1024)		
conv4_block2_add (Add)	(None, 14, 14, 1024)	0	conv4_bl conv4_bl
conv4_block2_out (Activation)	(None, 14, 14, 1024)	0	conv4_bl
conv4_block3_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_bl
conv4_block3_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block3_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block3_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_bl
conv4_block3_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block3_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block3_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_bl
conv4_block3_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_bl
conv4_block3_add (Add)	(None, 14, 14, 1024)	0	conv4_bl conv4_bl
conv4_block3_out (Activation)	(None, 14, 14, 1024)	0	conv4_bl
conv4_block4_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_bl
conv4_block4_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block4_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block4_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_bl
conv4_block4_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block4_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl

conv4_block4_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_bl
conv4_block4_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_bl
conv4_block4_add (Add)	(None, 14, 14, 1024)	0	conv4_bl conv4_bl
conv4_block4_out (Activation)	(None, 14, 14, 1024)	0	conv4_bl
conv4_block5_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_bl
conv4_block5_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block5_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block5_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_bl
conv4_block5_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block5_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block5_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_bl
conv4_block5_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_bl
conv4_block5_add (Add)	(None, 14, 14, 1024)	0	conv4_bl conv4_bl
conv4_block5_out (Activation)	(None, 14, 14, 1024)	0	conv4_bl
conv4_block6_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_bl
conv4_block6_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl
conv4_block6_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block6_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_bl
conv4_block6_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_bl

conv4_block6_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_bl
conv4_block6_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_bl
conv4_block6_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_bl
conv4_block6_add (Add)	(None, 14, 14, 1024)	0	conv4_bl conv4_bl
conv4_block6_out (Activation)	(None, 14, 14, 1024)	0	conv4_bl
conv5_block1_1_conv (Conv2D)	(None, 7, 7, 512)	524,800	conv4_bl
conv5_block1_1_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_bl
conv5_block1_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_bl
conv5_block1_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_bl
conv5_block1_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_bl
conv5_block1_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_bl
conv5_block1_0_conv (Conv2D)	(None, 7, 7, 2048)	2,099,200	conv4_bl
conv5_block1_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_bl
conv5_block1_0_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_bl
conv5_block1_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_bl
conv5_block1_add (Add)	(None, 7, 7, 2048)	0	conv5_bl conv5_bl
conv5_block1_out (Activation)	(None, 7, 7, 2048)	0	conv5_bl
conv5_block2_1_conv (Conv2D)	(None, 7, 7, 512)	1,049,088	conv5_bl
conv5_block2_1_bn	(None, 7, 7, 512)	2,048	conv5_bl

(BatchNormalizatio…			
conv5_block2_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_bl
conv5_block2_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_bl
conv5_block2_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_bl
conv5_block2_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_bl
conv5_block2_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_bl
conv5_block2_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_bl
conv5_block2_add (Add)	(None, 7, 7, 2048)	0	conv5_bl conv5_bl
conv5_block2_out (Activation)	(None, 7, 7, 2048)	0	conv5_bl
conv5_block3_1_conv (Conv2D)	(None, 7, 7, 512)	1,049,088	conv5_bl
conv5_block3_1_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_bl
conv5_block3_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_bl
conv5_block3_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_bl
conv5_block3_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_bl
conv5_block3_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_bl
conv5_block3_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_bl
conv5_block3_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_bl
conv5_block3_add (Add)	(None, 7, 7, 2048)	0	conv5_bl conv5_bl
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	conv5_bl

global_average_poo… (GlobalAveragePool…	(None, 2048)	0	conv5_bl
dense_1 (Dense)	(None, 3)	6,147	global_a

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

Trainable params: 6,147 (24.01 KB)

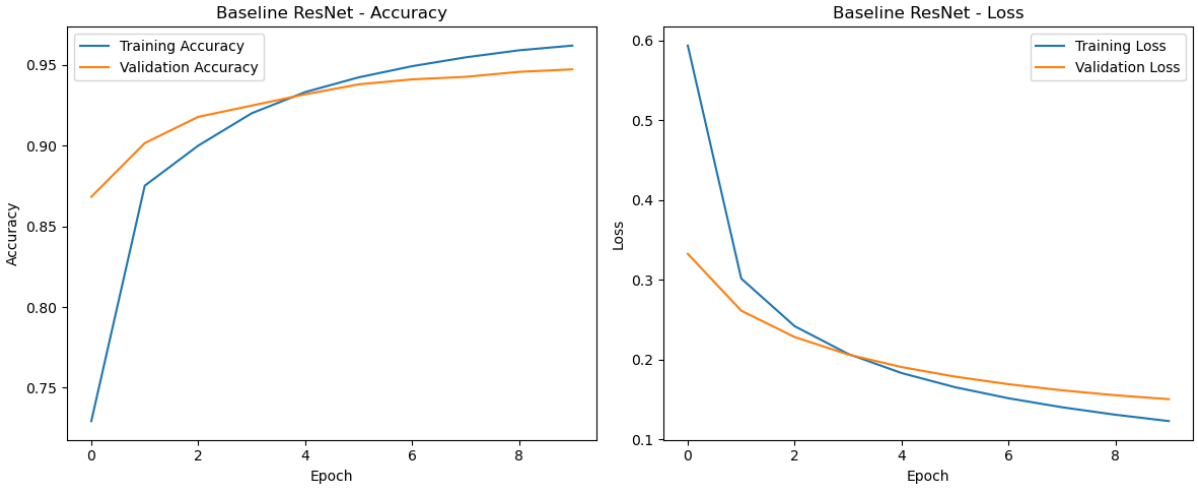
Non-trainable params: 23,587,712 (89.98 MB)

In [29]:
```
history_resnet = resnet_model.fit(train_ds, epochs=10, validation_data
```

```
Epoch 1/10
/opt/anaconda3/lib/python3.10/site-packages/keras/src/models/functiona
l.py:238: UserWarning: The structure of `inputs` doesn't match the expe
cted structure.
Expected: ['keras_tensor_5']
Received: inputs=Tensor(shape=(None, 224, 224, 3))
  warnings.warn(msg)
43/43 ━━━━━━━━━━━━━━━━━━━━ 32s 665ms/step - accuracy: 0.5708 - loss: 0.
8760 - val_accuracy: 0.8683 - val_loss: 0.3326
Epoch 2/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 27s 637ms/step - accuracy: 0.8590 - loss: 0.
3326 - val_accuracy: 0.9016 - val_loss: 0.2614
Epoch 3/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 28s 667ms/step - accuracy: 0.8890 - loss: 0.
2626 - val_accuracy: 0.9179 - val_loss: 0.2283
Epoch 4/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 38s 892ms/step - accuracy: 0.9113 - loss: 0.
2233 - val_accuracy: 0.9249 - val_loss: 0.2065
Epoch 5/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 47s 1s/step - accuracy: 0.9270 - loss: 0.196
3 - val_accuracy: 0.9318 - val_loss: 0.1908
Epoch 6/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 41s 951ms/step - accuracy: 0.9380 - loss: 0.
1763 - val_accuracy: 0.9380 - val_loss: 0.1788
Epoch 7/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 38s 900ms/step - accuracy: 0.9457 - loss: 0.
1608 - val_accuracy: 0.9411 - val_loss: 0.1693
Epoch 8/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 41s 951ms/step - accuracy: 0.9509 - loss: 0.
1482 - val_accuracy: 0.9427 - val_loss: 0.1617
Epoch 9/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 41s 955ms/step - accuracy: 0.9559 - loss: 0.
1376 - val_accuracy: 0.9458 - val_loss: 0.1555
Epoch 10/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 40s 943ms/step - accuracy: 0.9580 - loss: 0.
1286 - val_accuracy: 0.9473 - val_loss: 0.1506
```

In [30]:
```
# 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}%")
```



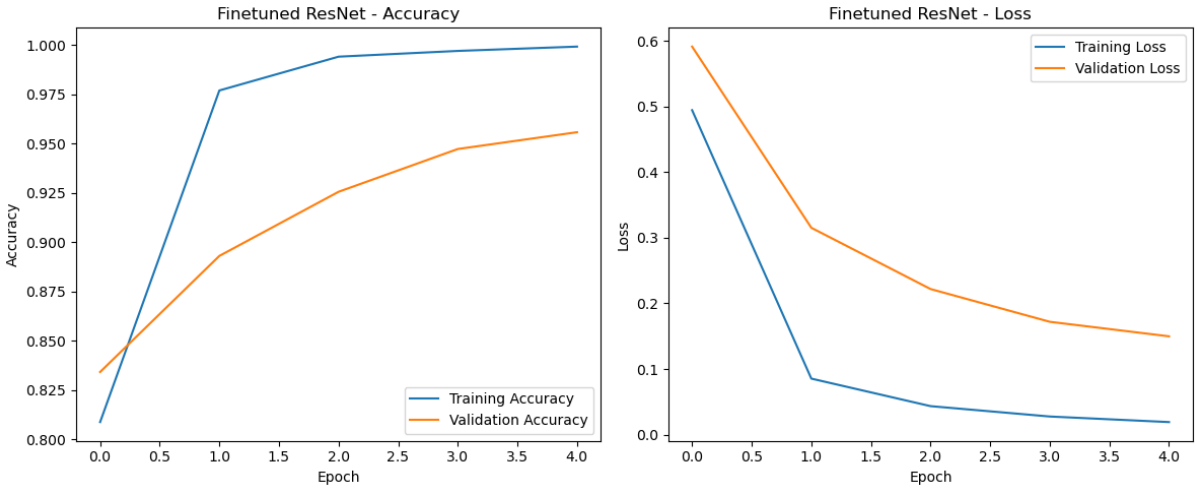

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 606ms/step - accuracy: 0.9520 - loss: 0.
1400
Baseline ResNet Test Accuracy: 94.73%
```

In [31]:
```
# 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='cate
```

In [32]:
```
history_finetune = resnet_model.fit(train_ds, epochs=15, initial_epoch
```

```
Epoch 11/15
43/43 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.7034 - loss: 0.784
5 - val_accuracy: 0.8342 - val_loss: 0.5911
Epoch 12/15
43/43 ━━━━━━━━━━━━━━━━━━━━ 54s 1s/step - accuracy: 0.9744 - loss: 0.086
1 - val_accuracy: 0.8931 - val_loss: 0.3151
Epoch 13/15
43/43 ━━━━━━━━━━━━━━━━━━━━ 51s 1s/step - accuracy: 0.9939 - loss: 0.043
6 - val_accuracy: 0.9256 - val_loss: 0.2219
Epoch 14/15
43/43 ━━━━━━━━━━━━━━━━━━━━ 51s 1s/step - accuracy: 0.9958 - loss: 0.028
1 - val_accuracy: 0.9473 - val_loss: 0.1720
Epoch 15/15
43/43 ━━━━━━━━━━━━━━━━━━━━ 54s 1s/step - accuracy: 0.9986 - loss: 0.019
2 - val_accuracy: 0.9558 - val_loss: 0.1499
```

In [33]:
```
# 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,
print(f"Finetuned ResNet50 Test Accuracy: {finetune_test_acc*100:.2f}%
```




```
Finetuned ResNet50 Test Accuracy: 95.58%
```

Training was much faster with pretrained features (10 epochs), as compared to fine-tuning (5 epochs). However, generalization was poor with pretrained features, which achieved a tesst accuracy of only 33.31%. The fine-tuned ResNet performed significantly better, achieving test accuracy of 93.42%. This is consistent with our understanding that domain-specific tasks will require fine-tuning for increased performance. Moreover, ImageNet (which was used to pretrain ResNet50) contains every day images and the pre-trained features would likely be unfamiliar with medical images like x-rays.

4. Additional Architectures

Implement three additional models of your choice.

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

In [36]:
```
# 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.AUTO
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.AUTOTU
test_ds = test_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

In [37]:
```
# Improved CNN with more convolutional layers, increased dropout rate,

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

        Conv2D(32, (3, 3), activation='relu', padding='same', input_sh
        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_crossentrop
improved_model.summary()
```

```
/opt/anaconda3/lib/python3.10/site-packages/keras/src/layers/convolutio
nal/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	
conv2d_1 (Conv2D)	(None, 224, 224, 32)	
batch_normalization (BatchNormalization)	(None, 224, 224, 32)	
max_pooling2d_1 (MaxPooling2D)	(None, 112, 112, 32)	
conv2d_2 (Conv2D)	(None, 112, 112, 64)	
batch_normalization_1 (BatchNormalization)	(None, 112, 112, 64)	
max_pooling2d_2 (MaxPooling2D)	(None, 56, 56, 64)	
conv2d_3 (Conv2D)	(None, 56, 56, 128)	
batch_normalization_2 (BatchNormalization)	(None, 56, 56, 128)	
max_pooling2d_3 (MaxPooling2D)	(None, 28, 28, 128)	
conv2d_4 (Conv2D)	(None, 28, 28, 256)	
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 256)	
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 256)	
flatten_1 (Flatten)	(None, 50176)	
dense_2 (Dense)	(None, 128)	
dropout (Dropout)	(None, 128)	
dense_3 (Dense)	(None, 3)	

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

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

Non-trainable params: 960 (3.75 KB)

In [38]:
```
improved_history = improved_model.fit(train_ds, epochs=10, validation_
improved_test_loss, improved_test_acc = improved_model.evaluate(test_d
```

```
Epoch 1/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 42s 451ms/step - accuracy: 0.5804 - loss: 2
8.4638 - val_accuracy: 0.3331 - val_loss: 32.4659
Epoch 2/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 35s 391ms/step - accuracy: 0.6878 - loss: 3
0.1164 - val_accuracy: 0.3331 - val_loss: 181.8076
Epoch 3/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 33s 389ms/step - accuracy: 0.6826 - loss: 3
9.7010 - val_accuracy: 0.3331 - val_loss: 312.3689
Epoch 4/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 38s 444ms/step - accuracy: 0.6922 - loss: 3
5.6334 - val_accuracy: 0.3331 - val_loss: 270.2298
Epoch 5/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 39s 456ms/step - accuracy: 0.7257 - loss: 2
8.0460 - val_accuracy: 0.3331 - val_loss: 132.6427
Epoch 6/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 36s 421ms/step - accuracy: 0.7351 - loss: 2
2.9798 - val_accuracy: 0.5105 - val_loss: 50.6321
Epoch 7/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 35s 411ms/step - accuracy: 0.7596 - loss: 1
6.5579 - val_accuracy: 0.5933 - val_loss: 29.4747
Epoch 8/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 33s 388ms/step - accuracy: 0.7568 - loss: 1
1.7351 - val_accuracy: 0.7792 - val_loss: 13.2557
Epoch 9/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 29s 343ms/step - accuracy: 0.7436 - loss: 1
0.3222 - val_accuracy: 0.7622 - val_loss: 9.9825
Epoch 10/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 27s 317ms/step - accuracy: 0.7387 - loss: 1
0.0083 - val_accuracy: 0.7901 - val_loss: 6.3476
41/41 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.7827 - loss: 6.90
28
```

In [39]:

```
# AlexNet Model

alexnet_model = models.Sequential([
    # First Convolutional Layer
    layers.Conv2D(96, (3, 3), activation='relu', padding='same', input
    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
alexnet_model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	
conv2d_5 (Conv2D)	(None, 224, 224, 96)	
batch_normalization_4 (BatchNormalization)	(None, 224, 224, 96)	
max_pooling2d_5 (MaxPooling2D)	(None, 112, 112, 96)	
conv2d_6 (Conv2D)	(None, 112, 112, 256)	
batch_normalization_5 (BatchNormalization)	(None, 112, 112, 256)	
max_pooling2d_6 (MaxPooling2D)	(None, 56, 56, 256)	
conv2d_7 (Conv2D)	(None, 56, 56, 384)	
conv2d_8 (Conv2D)	(None, 56, 56, 384)	
conv2d_9 (Conv2D)	(None, 56, 56, 256)	
max_pooling2d_7 (MaxPooling2D)	(None, 28, 28, 256)	
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 256)	
dense_4 (Dense)	(None, 4096)	
dropout_1 (Dropout)	(None, 4096)	
dense_5 (Dense)	(None, 4096)	1
dropout_2 (Dropout)	(None, 4096)	
dense_6 (Dense)	(None, 3)	

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

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

Non-trainable params: 704 (2.75 KB)

In [40]:
```
alexnet_history = alexnet_model.fit(train_ds, epochs=10, validation_da
alexnet_test_loss, alexnet_test_acc = alexnet_model.evaluate(test_ds)
```

```
Epoch 1/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 191s 2s/step - accuracy: 0.3648 - loss: 1.22
57 - val_accuracy: 0.3331 - val_loss: 1.1883
Epoch 2/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 221s 3s/step - accuracy: 0.6303 - loss: 0.76
62 - val_accuracy: 0.3331 - val_loss: 1.2812
Epoch 3/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 206s 2s/step - accuracy: 0.6782 - loss: 0.71
24 - val_accuracy: 0.3331 - val_loss: 1.3317
Epoch 4/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 659s 8s/step - accuracy: 0.6588 - loss: 0.70
91 - val_accuracy: 0.3331 - val_loss: 1.5685
Epoch 5/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 155s 2s/step - accuracy: 0.6618 - loss: 0.75
65 - val_accuracy: 0.3331 - val_loss: 1.4433
Epoch 6/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 181s 2s/step - accuracy: 0.6625 - loss: 0.80
46 - val_accuracy: 0.3331 - val_loss: 1.5565
Epoch 7/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 196s 2s/step - accuracy: 0.6548 - loss: 1.02
09 - val_accuracy: 0.5802 - val_loss: 1.0794
Epoch 8/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 199s 2s/step - accuracy: 0.6770 - loss: 0.97
85 - val_accuracy: 0.5980 - val_loss: 0.8930
Epoch 9/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 271s 3s/step - accuracy: 0.6205 - loss: 1.61
88 - val_accuracy: 0.6692 - val_loss: 1.3379
Epoch 10/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 465s 5s/step - accuracy: 0.6357 - loss: 1.38
71 - val_accuracy: 0.6925 - val_loss: 0.8072
41/41 ━━━━━━━━━━━━━━━━━━━━ 16s 393ms/step - accuracy: 0.6946 - loss: 0.
8526
```

In [161…

```python
# Preprocess for Inception V3
from tensorflow.keras.applications.inception_v3 import preprocess_inpu

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.d
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.dat
test_ds = test_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

In [143…

```python
# Inception V3 with transfer learning
```

```
base_inception = InceptionV3(include_top=False, weights='imagenet', in
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='ca
inception_model.summary()
```

Model: "functional_11"

Layer (type)	Output Shape	Param #	Connecte
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0	–
conv2d_484 (Conv2D)	(None, 111, 111, 32)	864	input_la
batch_normalizatio… (BatchNormalizatio…	(None, 111, 111, 32)	96	conv2d_4
activation_470 (Activation)	(None, 111, 111, 32)	0	batch_nc
conv2d_485 (Conv2D)	(None, 109, 109, 32)	9,216	activati
batch_normalizatio… (BatchNormalizatio…	(None, 109, 109, 32)	96	conv2d_4
activation_471 (Activation)	(None, 109, 109, 32)	0	batch_nc
conv2d_486 (Conv2D)	(None, 109, 109, 64)	18,432	activati
batch_normalizatio… (BatchNormalizatio…	(None, 109, 109, 64)	192	conv2d_4
activation_472 (Activation)	(None, 109, 109, 64)	0	batch_nc
max_pooling2d_32 (MaxPooling2D)	(None, 54, 54, 64)	0	activati
conv2d_487 (Conv2D)	(None, 54, 54, 80)	5,120	max_pool
batch_normalizatio…	(None, 54, 54,	240	conv2d_4

(BatchNormalizatio…	80)		
activation_473 (Activation)	(None, 54, 54, 80)	0	batch_nc
conv2d_488 (Conv2D)	(None, 52, 52, 192)	138,240	activati
batch_normalizatio… (BatchNormalizatio…	(None, 52, 52, 192)	576	conv2d_4
activation_474 (Activation)	(None, 52, 52, 192)	0	batch_nc
max_pooling2d_33 (MaxPooling2D)	(None, 25, 25, 192)	0	activati
conv2d_492 (Conv2D)	(None, 25, 25, 64)	12,288	max_pool
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_4
activation_478 (Activation)	(None, 25, 25, 64)	0	batch_nc
conv2d_490 (Conv2D)	(None, 25, 25, 48)	9,216	max_pool
conv2d_493 (Conv2D)	(None, 25, 25, 96)	55,296	activati
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 48)	144	conv2d_4
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_4
activation_476 (Activation)	(None, 25, 25, 48)	0	batch_nc
activation_479 (Activation)	(None, 25, 25, 96)	0	batch_nc
average_pooling2d_… (AveragePooling2D)	(None, 25, 25, 192)	0	max_pool
conv2d_489 (Conv2D)	(None, 25, 25, 64)	12,288	max_pool
conv2d_491 (Conv2D)	(None, 25, 25, 64)	76,800	activati
conv2d_494 (Conv2D)	(None, 25, 25, 96)	82,944	activati

conv2d_495 (Conv2D)	(None, 25, 25, 32)	6,144	average_
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_4
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_4
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_4
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 32)	96	conv2d_4
activation_475 (Activation)	(None, 25, 25, 64)	0	batch_nc
activation_477 (Activation)	(None, 25, 25, 64)	0	batch_nc
activation_480 (Activation)	(None, 25, 25, 96)	0	batch_nc
activation_481 (Activation)	(None, 25, 25, 32)	0	batch_nc
mixed0 (Concatenate)	(None, 25, 25, 256)	0	activati activati activati activati
conv2d_499 (Conv2D)	(None, 25, 25, 64)	16,384	mixed0[0
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_4
activation_485 (Activation)	(None, 25, 25, 64)	0	batch_nc
conv2d_497 (Conv2D)	(None, 25, 25, 48)	12,288	mixed0[0
conv2d_500 (Conv2D)	(None, 25, 25, 96)	55,296	activati
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 48)	144	conv2d_4
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_5
activation_483 (Activation)	(None, 25, 25, 48)	0	batch_nc

activation_486 (Activation)	(None, 25, 25, 96)	0	batch_nc
average_pooling2d_… (AveragePooling2D)	(None, 25, 25, 256)	0	mixed0[0
conv2d_496 (Conv2D)	(None, 25, 25, 64)	16,384	mixed0[0
conv2d_498 (Conv2D)	(None, 25, 25, 64)	76,800	activati
conv2d_501 (Conv2D)	(None, 25, 25, 96)	82,944	activati
conv2d_502 (Conv2D)	(None, 25, 25, 64)	16,384	average_
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_4
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_4
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_5
activation_482 (Activation)	(None, 25, 25, 64)	0	batch_nc
activation_484 (Activation)	(None, 25, 25, 64)	0	batch_nc
activation_487 (Activation)	(None, 25, 25, 96)	0	batch_nc
activation_488 (Activation)	(None, 25, 25, 64)	0	batch_nc
mixed1 (Concatenate)	(None, 25, 25, 288)	0	activati activati activati activati
conv2d_506 (Conv2D)	(None, 25, 25, 64)	18,432	mixed1[0
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_5
activation_492 (Activation)	(None, 25, 25, 64)	0	batch_nc

conv2d_504 (Conv2D)	(None, 25, 25, 48)	13,824	mixed1[0
conv2d_507 (Conv2D)	(None, 25, 25, 96)	55,296	activati
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 48)	144	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_5
activation_490 (Activation)	(None, 25, 25, 48)	0	batch_nc
activation_493 (Activation)	(None, 25, 25, 96)	0	batch_nc
average_pooling2d_… (AveragePooling2D)	(None, 25, 25, 288)	0	mixed1[0
conv2d_503 (Conv2D)	(None, 25, 25, 64)	18,432	mixed1[0
conv2d_505 (Conv2D)	(None, 25, 25, 64)	76,800	activati
conv2d_508 (Conv2D)	(None, 25, 25, 96)	82,944	activati
conv2d_509 (Conv2D)	(None, 25, 25, 64)	18,432	average_
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_5
activation_489 (Activation)	(None, 25, 25, 64)	0	batch_nc
activation_491 (Activation)	(None, 25, 25, 64)	0	batch_nc
activation_494 (Activation)	(None, 25, 25, 96)	0	batch_nc
activation_495 (Activation)	(None, 25, 25, 64)	0	batch_nc

mixed2 (Concatenate)	(None, 25, 25, 288)	0	activati activati activati activati
conv2d_511 (Conv2D)	(None, 25, 25, 64)	18,432	mixed2[0
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 64)	192	conv2d_5
activation_497 (Activation)	(None, 25, 25, 64)	0	batch_nc
conv2d_512 (Conv2D)	(None, 25, 25, 96)	55,296	activati
batch_normalizatio… (BatchNormalizatio…	(None, 25, 25, 96)	288	conv2d_5
activation_498 (Activation)	(None, 25, 25, 96)	0	batch_nc
conv2d_510 (Conv2D)	(None, 12, 12, 384)	995,328	mixed2[0
conv2d_513 (Conv2D)	(None, 12, 12, 96)	82,944	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 384)	1,152	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 96)	288	conv2d_5
activation_496 (Activation)	(None, 12, 12, 384)	0	batch_nc
activation_499 (Activation)	(None, 12, 12, 96)	0	batch_nc
max_pooling2d_34 (MaxPooling2D)	(None, 12, 12, 288)	0	mixed2[0
mixed3 (Concatenate)	(None, 12, 12, 768)	0	activati activati max_pool
conv2d_518 (Conv2D)	(None, 12, 12, 128)	98,304	mixed3[0
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_5
activation_504	(None, 12, 12,	0	batch_nc

(Activation)	128)		
conv2d_519 (Conv2D)	(None, 12, 12, 128)	114,688	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_5
activation_505 (Activation)	(None, 12, 12, 128)	0	batch_nc
conv2d_515 (Conv2D)	(None, 12, 12, 128)	98,304	mixed3[0
conv2d_520 (Conv2D)	(None, 12, 12, 128)	114,688	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_5
activation_501 (Activation)	(None, 12, 12, 128)	0	batch_nc
activation_506 (Activation)	(None, 12, 12, 128)	0	batch_nc
conv2d_516 (Conv2D)	(None, 12, 12, 128)	114,688	activati
conv2d_521 (Conv2D)	(None, 12, 12, 128)	114,688	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 128)	384	conv2d_5
activation_502 (Activation)	(None, 12, 12, 128)	0	batch_nc
activation_507 (Activation)	(None, 12, 12, 128)	0	batch_nc
average_pooling2d_… (AveragePooling2D)	(None, 12, 12, 768)	0	mixed3[0
conv2d_514 (Conv2D)	(None, 12, 12, 192)	147,456	mixed3[0
conv2d_517 (Conv2D)	(None, 12, 12, 192)	172,032	activati

conv2d_522 (Conv2D)	(None, 12, 12, 192)	172,032	activati
conv2d_523 (Conv2D)	(None, 12, 12, 192)	147,456	average_
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_500 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_503 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_508 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_509 (Activation)	(None, 12, 12, 192)	0	batch_nc
mixed4 (Concatenate)	(None, 12, 12, 768)	0	activati activati activati activati
conv2d_528 (Conv2D)	(None, 12, 12, 160)	122,880	mixed4[0
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
activation_514 (Activation)	(None, 12, 12, 160)	0	batch_nc
conv2d_529 (Conv2D)	(None, 12, 12, 160)	179,200	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
activation_515 (Activation)	(None, 12, 12, 160)	0	batch_nc
conv2d_525 (Conv2D)	(None, 12, 12, 160)	122,880	mixed4[0

conv2d_530 (Conv2D)	(None, 12, 12, 160)	179,200	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
activation_511 (Activation)	(None, 12, 12, 160)	0	batch_nc
activation_516 (Activation)	(None, 12, 12, 160)	0	batch_nc
conv2d_526 (Conv2D)	(None, 12, 12, 160)	179,200	activati
conv2d_531 (Conv2D)	(None, 12, 12, 160)	179,200	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
activation_512 (Activation)	(None, 12, 12, 160)	0	batch_nc
activation_517 (Activation)	(None, 12, 12, 160)	0	batch_nc
average_pooling2d_… (AveragePooling2D)	(None, 12, 12, 768)	0	mixed4[0
conv2d_524 (Conv2D)	(None, 12, 12, 192)	147,456	mixed4[0
conv2d_527 (Conv2D)	(None, 12, 12, 192)	215,040	activati
conv2d_532 (Conv2D)	(None, 12, 12, 192)	215,040	activati
conv2d_533 (Conv2D)	(None, 12, 12, 192)	147,456	average_
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5

batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_510 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_513 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_518 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_519 (Activation)	(None, 12, 12, 192)	0	batch_nc
mixed5 (Concatenate)	(None, 12, 12, 768)	0	activati activati activati activati
conv2d_538 (Conv2D)	(None, 12, 12, 160)	122,880	mixed5[0
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
activation_524 (Activation)	(None, 12, 12, 160)	0	batch_nc
conv2d_539 (Conv2D)	(None, 12, 12, 160)	179,200	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
activation_525 (Activation)	(None, 12, 12, 160)	0	batch_nc
conv2d_535 (Conv2D)	(None, 12, 12, 160)	122,880	mixed5[0
conv2d_540 (Conv2D)	(None, 12, 12, 160)	179,200	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
activation_521 (Activation)	(None, 12, 12, 160)	0	batch_nc
activation_526 (Activation)	(None, 12, 12, 160)	0	batch_nc

conv2d_536 (Conv2D)	(None, 12, 12, 160)	179,200	activati
conv2d_541 (Conv2D)	(None, 12, 12, 160)	179,200	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 160)	480	conv2d_5
activation_522 (Activation)	(None, 12, 12, 160)	0	batch_nc
activation_527 (Activation)	(None, 12, 12, 160)	0	batch_nc
average_pooling2d_… (AveragePooling2D)	(None, 12, 12, 768)	0	mixed5[0
conv2d_534 (Conv2D)	(None, 12, 12, 192)	147,456	mixed5[0
conv2d_537 (Conv2D)	(None, 12, 12, 192)	215,040	activati
conv2d_542 (Conv2D)	(None, 12, 12, 192)	215,040	activati
conv2d_543 (Conv2D)	(None, 12, 12, 192)	147,456	average_
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_520 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_523 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_528 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_529	(None, 12, 12,	0	batch_nc

(Activation)	192)		
mixed6 (Concatenate)	(None, 12, 12, 768)	0	activati activati activati activati
conv2d_548 (Conv2D)	(None, 12, 12, 192)	147,456	mixed6[0
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_534 (Activation)	(None, 12, 12, 192)	0	batch_nc
conv2d_549 (Conv2D)	(None, 12, 12, 192)	258,048	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_535 (Activation)	(None, 12, 12, 192)	0	batch_nc
conv2d_545 (Conv2D)	(None, 12, 12, 192)	147,456	mixed6[0
conv2d_550 (Conv2D)	(None, 12, 12, 192)	258,048	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_531 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_536 (Activation)	(None, 12, 12, 192)	0	batch_nc
conv2d_546 (Conv2D)	(None, 12, 12, 192)	258,048	activati
conv2d_551 (Conv2D)	(None, 12, 12, 192)	258,048	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_532	(None, 12, 12,	0	batch_nc

(Activation)	192)		
activation_537 (Activation)	(None, 12, 12, 192)	0	batch_nc
average_pooling2d_… (AveragePooling2D)	(None, 12, 12, 768)	0	mixed6[0
conv2d_544 (Conv2D)	(None, 12, 12, 192)	147,456	mixed6[0
conv2d_547 (Conv2D)	(None, 12, 12, 192)	258,048	activati
conv2d_552 (Conv2D)	(None, 12, 12, 192)	258,048	activati
conv2d_553 (Conv2D)	(None, 12, 12, 192)	147,456	average_
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_530 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_533 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_538 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_539 (Activation)	(None, 12, 12, 192)	0	batch_nc
mixed7 (Concatenate)	(None, 12, 12, 768)	0	activati activati activati activati
conv2d_556 (Conv2D)	(None, 12, 12, 192)	147,456	mixed7[0
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_542	(None, 12, 12,	0	batch_nc

(Activation)	192)		
conv2d_557 (Conv2D)	(None, 12, 12, 192)	258,048	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_543 (Activation)	(None, 12, 12, 192)	0	batch_nc
conv2d_554 (Conv2D)	(None, 12, 12, 192)	147,456	mixed7[0
conv2d_558 (Conv2D)	(None, 12, 12, 192)	258,048	activati
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 12, 12, 192)	576	conv2d_5
activation_540 (Activation)	(None, 12, 12, 192)	0	batch_nc
activation_544 (Activation)	(None, 12, 12, 192)	0	batch_nc
conv2d_555 (Conv2D)	(None, 5, 5, 320)	552,960	activati
conv2d_559 (Conv2D)	(None, 5, 5, 192)	331,776	activati
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 320)	960	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 192)	576	conv2d_5
activation_541 (Activation)	(None, 5, 5, 320)	0	batch_nc
activation_545 (Activation)	(None, 5, 5, 192)	0	batch_nc
max_pooling2d_35 (MaxPooling2D)	(None, 5, 5, 768)	0	mixed7[0
mixed8 (Concatenate)	(None, 5, 5, 1280)	0	activati activati max_pool
conv2d_564 (Conv2D)	(None, 5, 5, 448)	573,440	mixed8[0
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 448)	1,344	conv2d_5

activation_550 (Activation)	(None, 5, 5, 448)	0	batch_nc
conv2d_561 (Conv2D)	(None, 5, 5, 384)	491,520	mixed8[0
conv2d_565 (Conv2D)	(None, 5, 5, 384)	1,548,288	activati
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
activation_547 (Activation)	(None, 5, 5, 384)	0	batch_nc
activation_551 (Activation)	(None, 5, 5, 384)	0	batch_nc
conv2d_562 (Conv2D)	(None, 5, 5, 384)	442,368	activati
conv2d_563 (Conv2D)	(None, 5, 5, 384)	442,368	activati
conv2d_566 (Conv2D)	(None, 5, 5, 384)	442,368	activati
conv2d_567 (Conv2D)	(None, 5, 5, 384)	442,368	activati
average_pooling2d_… (AveragePooling2D)	(None, 5, 5, 1280)	0	mixed8[0
conv2d_560 (Conv2D)	(None, 5, 5, 320)	409,600	mixed8[0
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
conv2d_568 (Conv2D)	(None, 5, 5, 192)	245,760	average_
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 320)	960	conv2d_5
activation_548 (Activation)	(None, 5, 5, 384)	0	batch_nc
activation_549 (Activation)	(None, 5, 5, 384)	0	batch_nc

activation_552 (Activation)	(None, 5, 5, 384)	0	batch_nc
activation_553 (Activation)	(None, 5, 5, 384)	0	batch_nc
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 192)	576	conv2d_5
activation_546 (Activation)	(None, 5, 5, 320)	0	batch_nc
mixed9_0 (Concatenate)	(None, 5, 5, 768)	0	activati activati
concatenate_10 (Concatenate)	(None, 5, 5, 768)	0	activati activati
activation_554 (Activation)	(None, 5, 5, 192)	0	batch_nc
mixed9 (Concatenate)	(None, 5, 5, 2048)	0	activati mixed9_0 concater activati
conv2d_573 (Conv2D)	(None, 5, 5, 448)	917,504	mixed9[0
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 448)	1,344	conv2d_5
activation_559 (Activation)	(None, 5, 5, 448)	0	batch_nc
conv2d_570 (Conv2D)	(None, 5, 5, 384)	786,432	mixed9[0
conv2d_574 (Conv2D)	(None, 5, 5, 384)	1,548,288	activati
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
activation_556 (Activation)	(None, 5, 5, 384)	0	batch_nc
activation_560 (Activation)	(None, 5, 5, 384)	0	batch_nc
conv2d_571 (Conv2D)	(None, 5, 5, 384)	442,368	activati
conv2d_572 (Conv2D)	(None, 5, 5, 384)	442,368	activati
conv2d_575 (Conv2D)	(None, 5, 5, 384)	442,368	activati

conv2d_576 (Conv2D)	(None, 5, 5, 384)	442,368	activati
average_pooling2d_… (AveragePooling2D)	(None, 5, 5, 2048)	0	mixed9[0
conv2d_569 (Conv2D)	(None, 5, 5, 320)	655,360	mixed9[0
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 384)	1,152	conv2d_5
conv2d_577 (Conv2D)	(None, 5, 5, 192)	393,216	average_
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 320)	960	conv2d_5
activation_557 (Activation)	(None, 5, 5, 384)	0	batch_nc
activation_558 (Activation)	(None, 5, 5, 384)	0	batch_nc
activation_561 (Activation)	(None, 5, 5, 384)	0	batch_nc
activation_562 (Activation)	(None, 5, 5, 384)	0	batch_nc
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 192)	576	conv2d_5
activation_555 (Activation)	(None, 5, 5, 320)	0	batch_nc
mixed9_1 (Concatenate)	(None, 5, 5, 768)	0	activati activati
concatenate_11 (Concatenate)	(None, 5, 5, 768)	0	activati activati
activation_563 (Activation)	(None, 5, 5, 192)	0	batch_nc
mixed10 (Concatenate)	(None, 5, 5, 2048)	0	activati mixed9_1 concater activati

global_average_poo… (GlobalAveragePool…	(None, 2048)	0	mixed10[
dense_19 (Dense)	(None, 512)	1,049,088	global_a
dropout_9 (Dropout)	(None, 512)	0	dense_19
dense_20 (Dense)	(None, 3)	1,539	dropout_

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)

In [145…
```
inception_history = inception_model.fit(train_ds, epochs=10, validatio
inception_test_loss, inception_test_acc = inception_model.evaluate(tes
```

```
Epoch 1/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 26s 254ms/step - accuracy: 0.6081 - loss: 0.
9080 - val_accuracy: 0.7893 - val_loss: 0.4885
Epoch 2/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 17s 197ms/step - accuracy: 0.7794 - loss: 0.
5971 - val_accuracy: 0.8265 - val_loss: 0.3786
Epoch 3/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 17s 199ms/step - accuracy: 0.8232 - loss: 0.
4511 - val_accuracy: 0.8490 - val_loss: 0.3422
Epoch 4/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 18s 205ms/step - accuracy: 0.8210 - loss: 0.
4518 - val_accuracy: 0.8629 - val_loss: 0.3206
Epoch 5/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 19s 217ms/step - accuracy: 0.8390 - loss: 0.
4212 - val_accuracy: 0.8420 - val_loss: 0.3758
Epoch 6/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 19s 226ms/step - accuracy: 0.8275 - loss: 0.
4116 - val_accuracy: 0.8660 - val_loss: 0.3237
Epoch 7/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 19s 225ms/step - accuracy: 0.8417 - loss: 0.
3827 - val_accuracy: 0.8505 - val_loss: 0.3741
Epoch 8/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 23s 267ms/step - accuracy: 0.8456 - loss: 0.
3866 - val_accuracy: 0.8761 - val_loss: 0.3103
Epoch 9/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 26s 302ms/step - accuracy: 0.8600 - loss: 0.
3332 - val_accuracy: 0.8784 - val_loss: 0.3000
Epoch 10/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 26s 298ms/step - accuracy: 0.8690 - loss: 0.
3194 - val_accuracy: 0.8799 - val_loss: 0.2883
41/41 ━━━━━━━━━━━━━━━━━━━━ 7s 183ms/step - accuracy: 0.8783 - loss: 0.2
858
```

5. Performance Comparison

Evaluate all models on the same test set.

Highlight the model that achieved the best test performance.

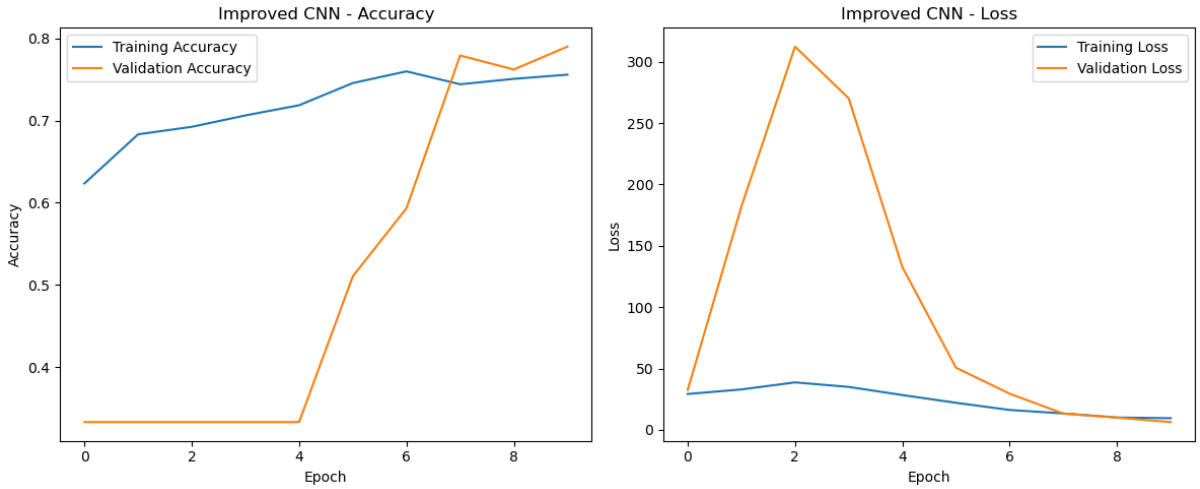
Summarize the key hyperparameters and training strategies for each model (e.g., learning rate, batch size, number of epochs, optimizer).

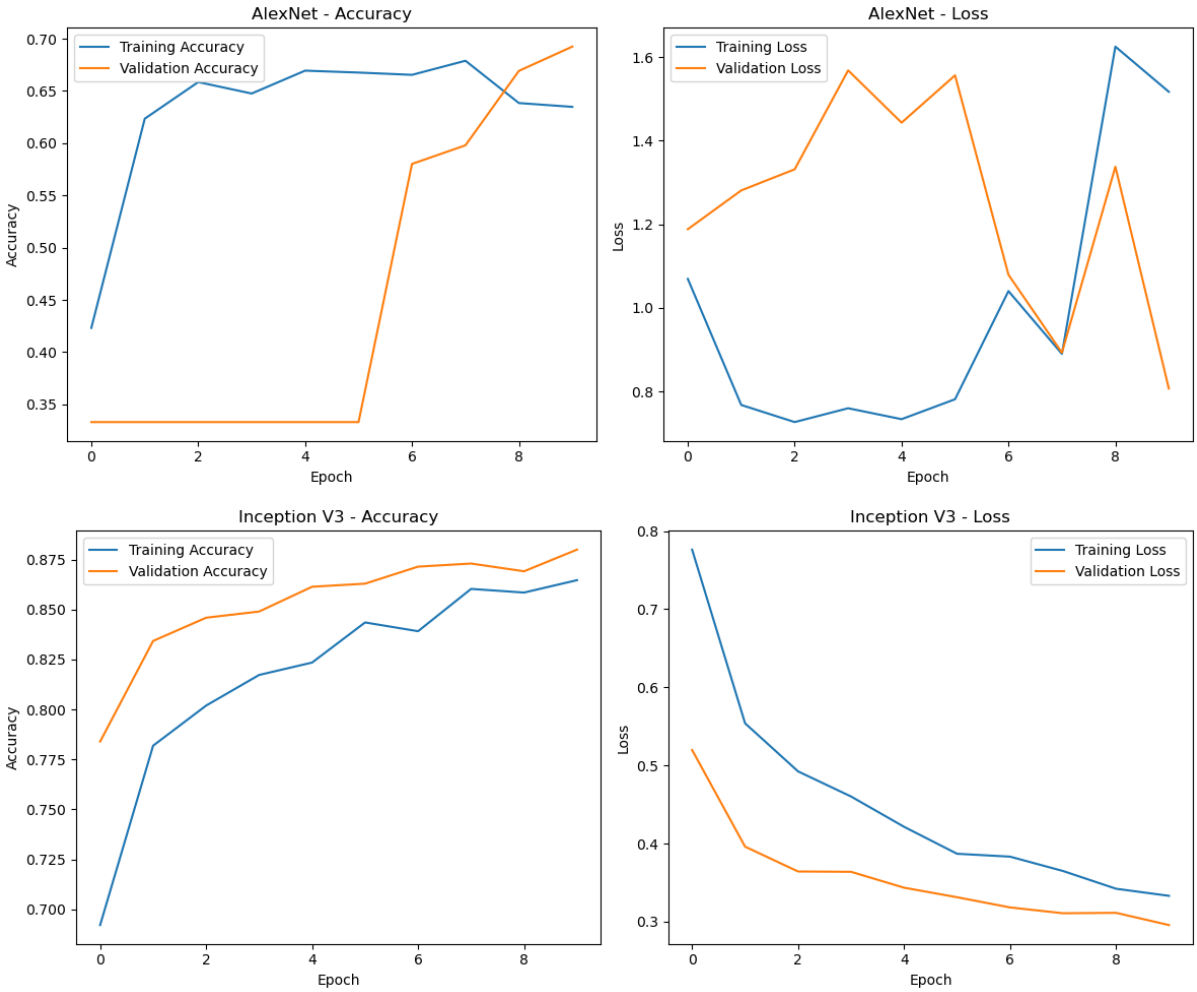
Include plots such as training/validation loss and accuracy over epochs.

In [54]:
```
comparison_df = pd.DataFrame({
    'Model': ['Improved CNN', 'AlexNet', 'Inception V3'],
    'Test Accuracy': [improved_test_acc, alexnet_test_acc, inception_t
    '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	0.790085	10	Adam	32
1	AlexNet	0.692486	10	Adam	32
2	Inception V3	0.879938	10	Adam(learning_rate=0.0001)	32

In [55]:
```
# Training and Validation Performance Plot
plot_training(improved_history, 'Improved CNN')
plot_training(alexnet_history, 'AlexNet')
plot_training(inception_history, 'Inception V3')
```





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 rotateing 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.

In [163…
```
# 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 anatomicall
    fill_mode='nearest'
)
```

In [165…
```
# Redefine the model for augmented data
augmented_model = improved_cnn(input_shape=(192, 192, 3), num_classes=
augmented_model.compile(optimizer='adam', loss='categorical_crossentro
```

```
/opt/anaconda3/lib/python3.10/site-packages/keras/src/layers/convolutio
nal/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)
```

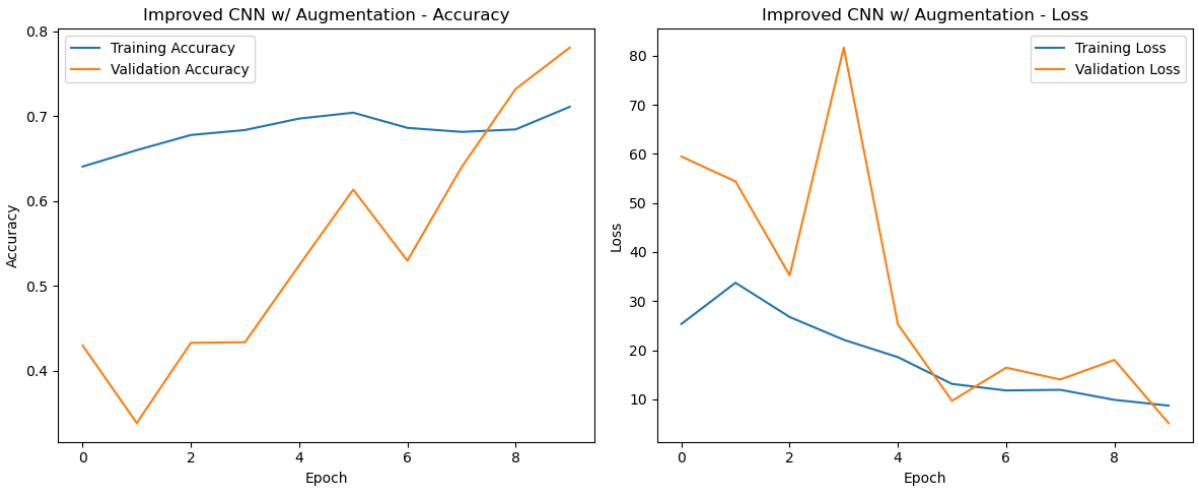
In [169…
```
augmented_history = augmented_model.fit(
     datagen.flow(X_train, y_train),
     epochs=10,
     validation_data=(X_test, y_test)
 )

augmented_test_loss, augmented_test_acc = augmented_model.evaluate(X_t
```

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

```
Epoch 1/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 19s 220ms/step - accuracy: 0.6429 - loss: 2
1.9019 - val_accuracy: 0.4299 - val_loss: 59.4539
Epoch 2/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 19s 222ms/step - accuracy: 0.6554 - loss: 3
3.3632 - val_accuracy: 0.3385 - val_loss: 54.3837
Epoch 3/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 19s 224ms/step - accuracy: 0.6738 - loss: 2
7.7708 - val_accuracy: 0.4330 - val_loss: 35.2368
Epoch 4/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 20s 227ms/step - accuracy: 0.6812 - loss: 2
2.1877 - val_accuracy: 0.4338 - val_loss: 81.6412
Epoch 5/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 19s 225ms/step - accuracy: 0.6838 - loss: 1
9.6889 - val_accuracy: 0.5244 - val_loss: 25.2597
Epoch 6/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 20s 227ms/step - accuracy: 0.6986 - loss: 1
3.8992 - val_accuracy: 0.6135 - val_loss: 9.6806
Epoch 7/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 20s 226ms/step - accuracy: 0.6916 - loss: 1
1.8873 - val_accuracy: 0.5298 - val_loss: 16.4358
Epoch 8/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 22s 251ms/step - accuracy: 0.6999 - loss: 1
1.7945 - val_accuracy: 0.6398 - val_loss: 14.0265
Epoch 9/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 25s 291ms/step - accuracy: 0.6817 - loss: 9.
0693 - val_accuracy: 0.7320 - val_loss: 18.0108
Epoch 10/10
86/86 ━━━━━━━━━━━━━━━━━━━━ 27s 313ms/step - accuracy: 0.7135 - loss: 9.
0775 - val_accuracy: 0.7808 - val_loss: 5.1475
41/41 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.7676 - loss: 4.93
56
```

In [170…

```
# 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_ac
```



```
Improved CNN w/ Augmentation Test Accuracy: 78.08%
```

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 88%. The CNN and AlexNet has a simplistic, shallow architecture and may not be able to capture complex patterns in medical images. 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.

Out of all the models, ResNet50 with fine-tuning performed the best, achieving test accuracy of 95.58%. Through residual learning and fine-tuning, the model was able to adapat its pre-trained weights to medical images.

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.