CIS 579-001-Artificial Intelligence

DEEP LEARNING BASED CLASSIFICATION OF KIDNEY DISEASES

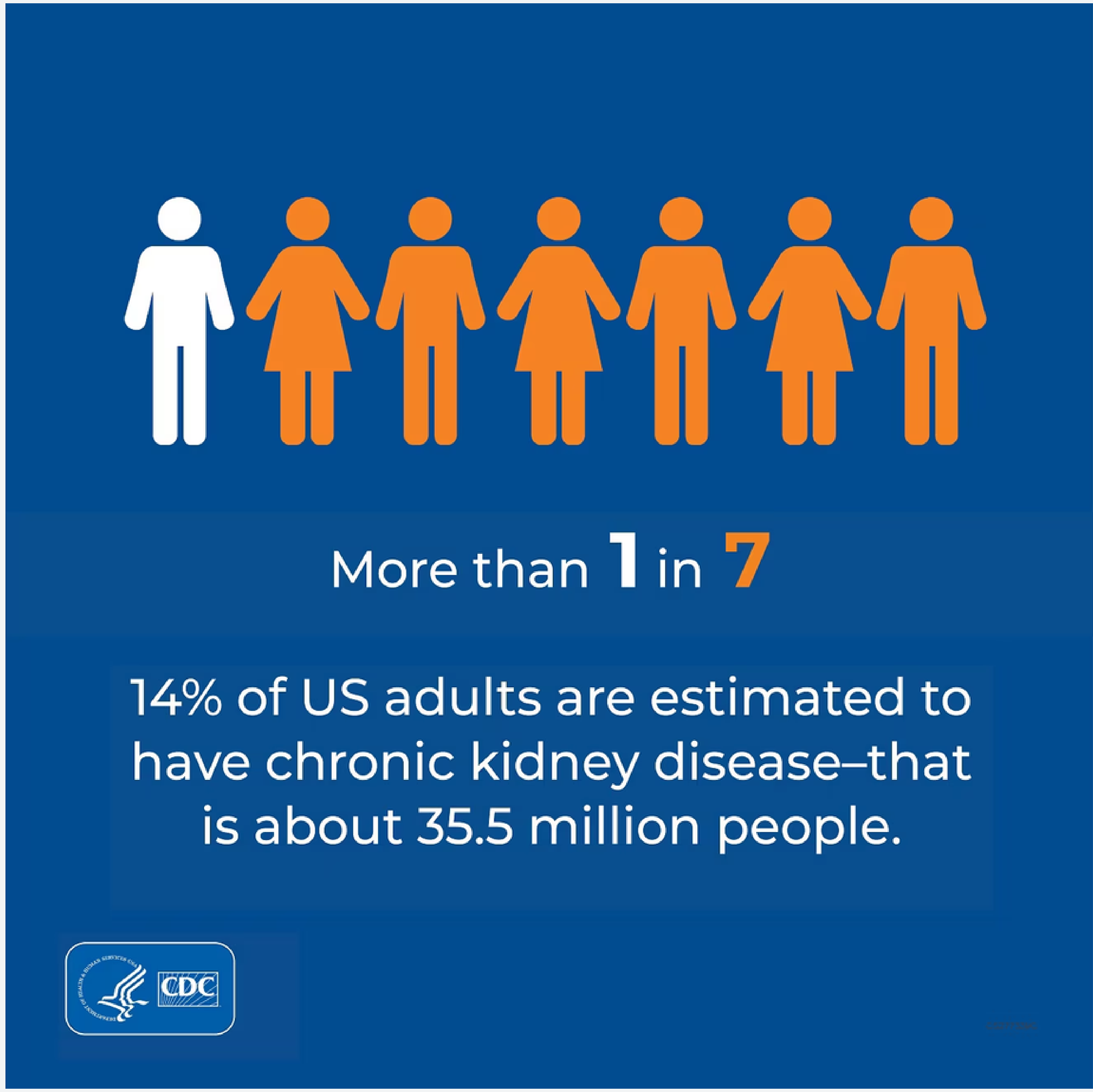
CONTENTS

INTRODUCTION

Kidney diseases encompass a range of conditions that affect the kidneys, the vital organs responsible for filtering waste products from the blood and excreting them through urine. These conditions can be acute, occurring suddenly, or chronic, developing over a longer period.

Diagnosing kidney diseases involves a combination of clinical evaluation, laboratory tests, and imaging techniques.

- patient's medical history
- physical examination
- urine and blood tests
- Imaging tests (ultrasound and computed tomography (CT) scans)



PROBLEM STATEMENT

- The goal of this project is to develop a reliable system that can automatically classify kidney CT images into one of four different categories of kidney diseases.

- By using advanced deep learning techniques, such as convolutional neural networks (CNNs) and well-known architectures like MobileNet, Inception V3, and DenseNet169 we are aiming to find the best model for the classification.

- This project is intended to reduce the manual effort needed to analyze images, minimize diagnostic errors, and potentially improve patient outcomes by facilitating early and accurate detection of kidney ailments.

DATASET DESCRIPTION

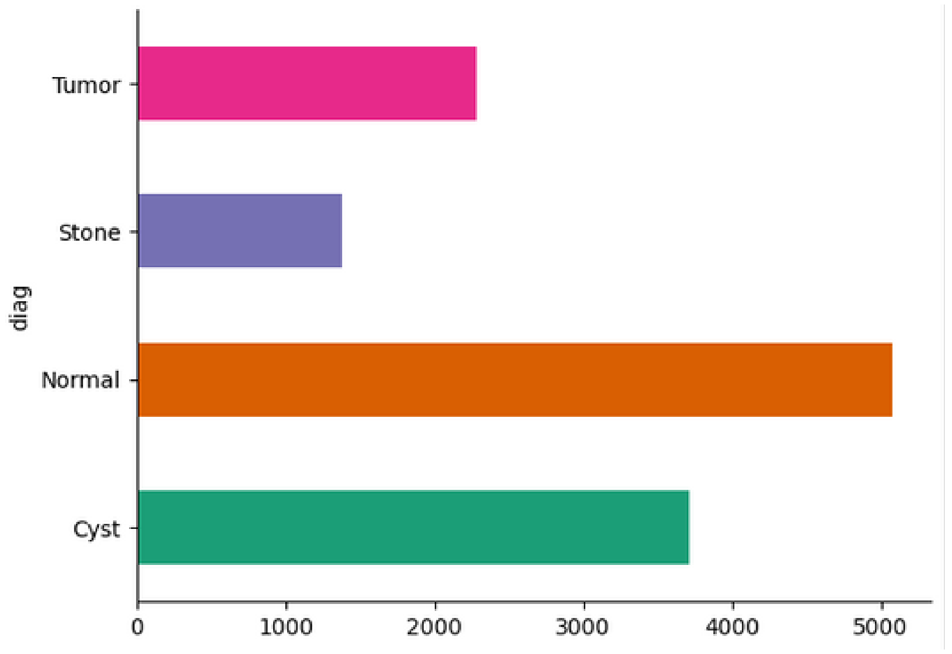
SOURCE: Kaggle

The dataset originated from PACS (Picture Archiving and Communication System) sourced from various hospitals across Dhaka, Bangladesh. It encompassed patients previously diagnosed with kidney conditions, including tumors, cysts, normal cases, and instances of kidney stones.
The dataset incorporated both Coronal and Axial cuts from contrast and non-contrast studies, following protocols for comprehensive abdominal assessments and urograms.

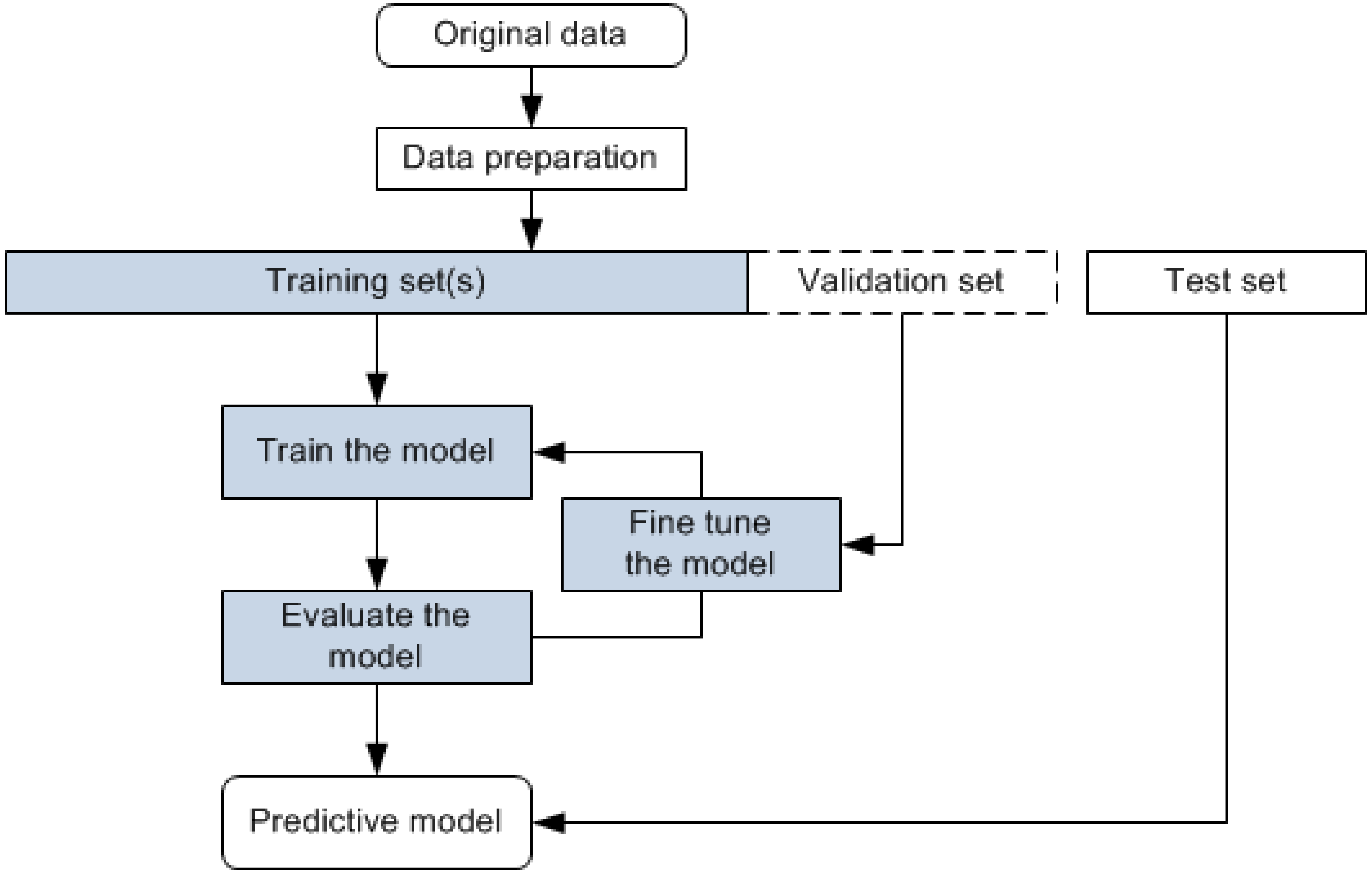
The size of data was initially 12446 images split into 4 different classes

4 Classes of data :

- Normal- 5077 images
- Stone -1371 images
- Tumor- 2283 images
- Cyst- 3709 images



WORKFLOW



Original data
Data preparation
Training set(s)
Validation set
Test set
Train the model
Fine tune the model
Evaluate the model
Predictive model

TOOLS AND TECHNOLOGIES

- PYTHON
- NUMPY
- PANDAS
- MATPLOTLIB
- TENSORFLOW
- KERAS
- GOOGLE COLAB

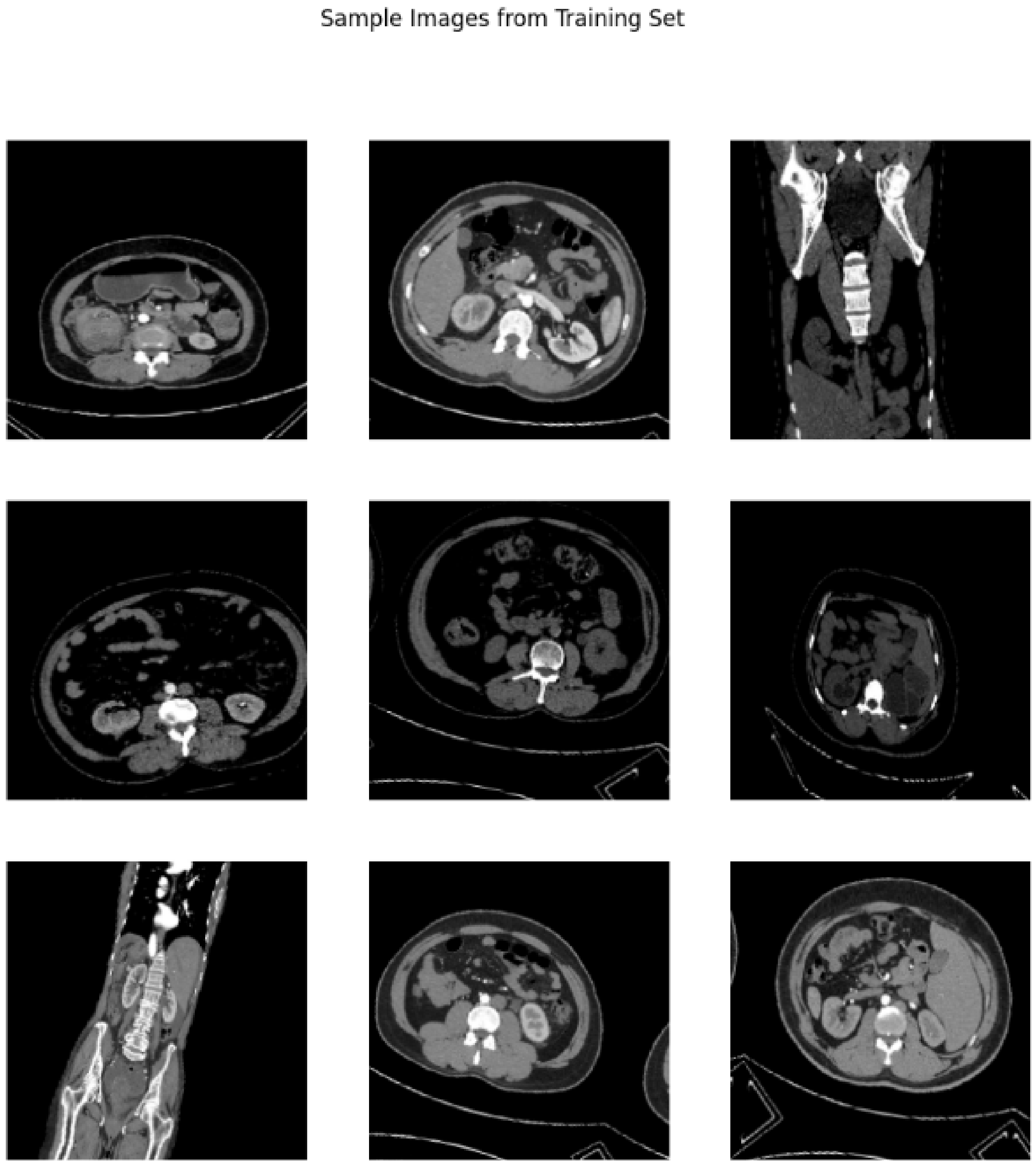
PRE-PROCESSING

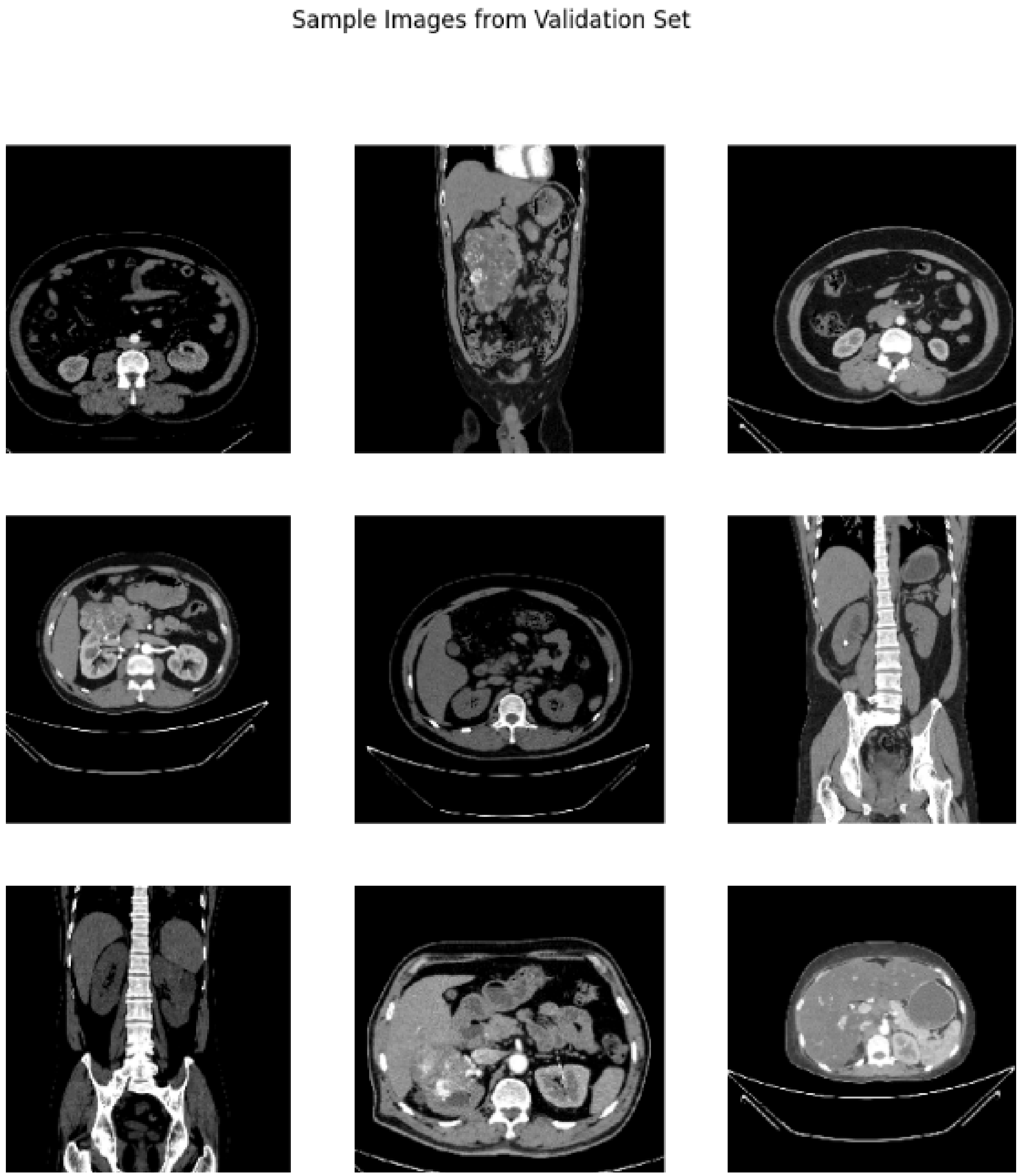
- Splitting folders essentially means taking a large folder that contains many images and dividing it into smaller folders, typically to organize data for machine learning or deep learning projects.
- These smaller folders are usually categorized into at least three types: training, validation, and testing.
- Here we have split it in the ratio of 80, 10, 10.

```
[ ]  import splitfolders
     splitfolders.ratio(
         "/content/drive/MyDrive/AIproject/CT-KIDNEY-DATASET-Normal-Cyst-Tumor-Stone/CT-KIDNEY-DATASET-Normal-Cyst-Tumor-Stone",
        output="./dataset",
        seed=1337,
        ratio=(0.8,0.1, 0.1))

     Copying files: 12446 files [06:03, 34.26 files/s]
```

PRE-PROCESSING





PRE-PROCESSING

Data augmentation is a technique used to artificially increase the diversity of your dataset by making small modifications to the existing data.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1/255.0,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range = 0.2,
    vertical_flip=True,
    horizontal_flip = True,
    fill_mode="reflect")

train_generator = train_datagen.flow_from_directory(
    '/kaggle/working/Finaldataset/train',
    target_size=(224, 224),
    class_mode='categorical',
    color_mode='rgb',
    batch_size = 32,)

val_datagen = ImageDataGenerator(rescale = 1/255.0)
val_generator = train_datagen.flow_from_directory(
     '/kaggle/working/Finaldataset/val',
    target_size=(224, 224),
    class_mode='categorical',
     color_mode='rgb',
    batch_size = 32,)
```

```
Found 9955 images belonging to 4 classes.
Found 1242 images belonging to 4 classes.
```

PRE-PROCESSING

```
import os

def count_images(root_dir):
    # This function walks through the directory structure and counts files
    for directory in ['train', 'test', 'val']:
        path = os.path.join(root_dir, directory)
        total_images = 0
        for root, dirs, files in os.walk(path):
            total_images += len([file for file in files if file.lower().endswith(('.png', '.jpg', '.jpeg'))])
        print(f"Total images in {directory}: {total_images}")

# Set the path to the directory where the images are stored after splitting
dataset_directory = "/kaggle/working/Finaldataset"
count_images(dataset_directory)
```

```
Total images in train: 9955
Total images in test: 1249
Total images in val: 1242
```

MODELS

- Convolutional Neural network
- DenseNet 121
- Inception V3
- MobileNet
- VGG16
- Ensemble

1. Model Construction: A Sequential model is built using TensorFlow's Keras API. It begins with a convolutional layer (Conv2D) with 16 filters of size 3x3 and ReLU activation, suitable for extracting features from the input images sized 224x224 with 3 color channels.
2. Pooling and Flattening: Followed by a max pooling layer (MaxPooling2D) to reduce the spatial dimensions, which helps in reducing the number of parameters and computational complexity. The output is then flattened (Flatten) to transform the 2D features into a 1D vector suitable for input into the dense layers.
3. Fully Connected Layers: Two dense layers are added; the first with 64 units and ReLU activation to learn the non-linear combinations of the features, and the second with 4 units and softmax activation to output the probabilities for each of the four classes.
4. Compilation: The model is compiled with the Adam optimizer and categorical crossentropy loss function, which is typical for multi-class classification tasks, and it measures accuracy as the metric.
5. Training with Early Stopping: An EarlyStopping callback is used during training to prevent overfitting by halting training when the validation loss does not improve significantly for four epochs, thereby ensuring the model trains only until it is beneficial.

CNN ARCHITECTURE

Model Setup: Constructed a Sequential CNN with a Conv2D layer (16 filters, 3x3 kernel) for initial feature extraction from 224x224 color images, followed by MaxPooling2D for dimensionality reduction.

Dense Layers: Added a Flatten layer to convert 2D features to 1D, followed by two Dense layers—first with 64 neurons (ReLU activation) for deeper feature integration and a final layer with 4 neurons (softmax activation) for class probability output.

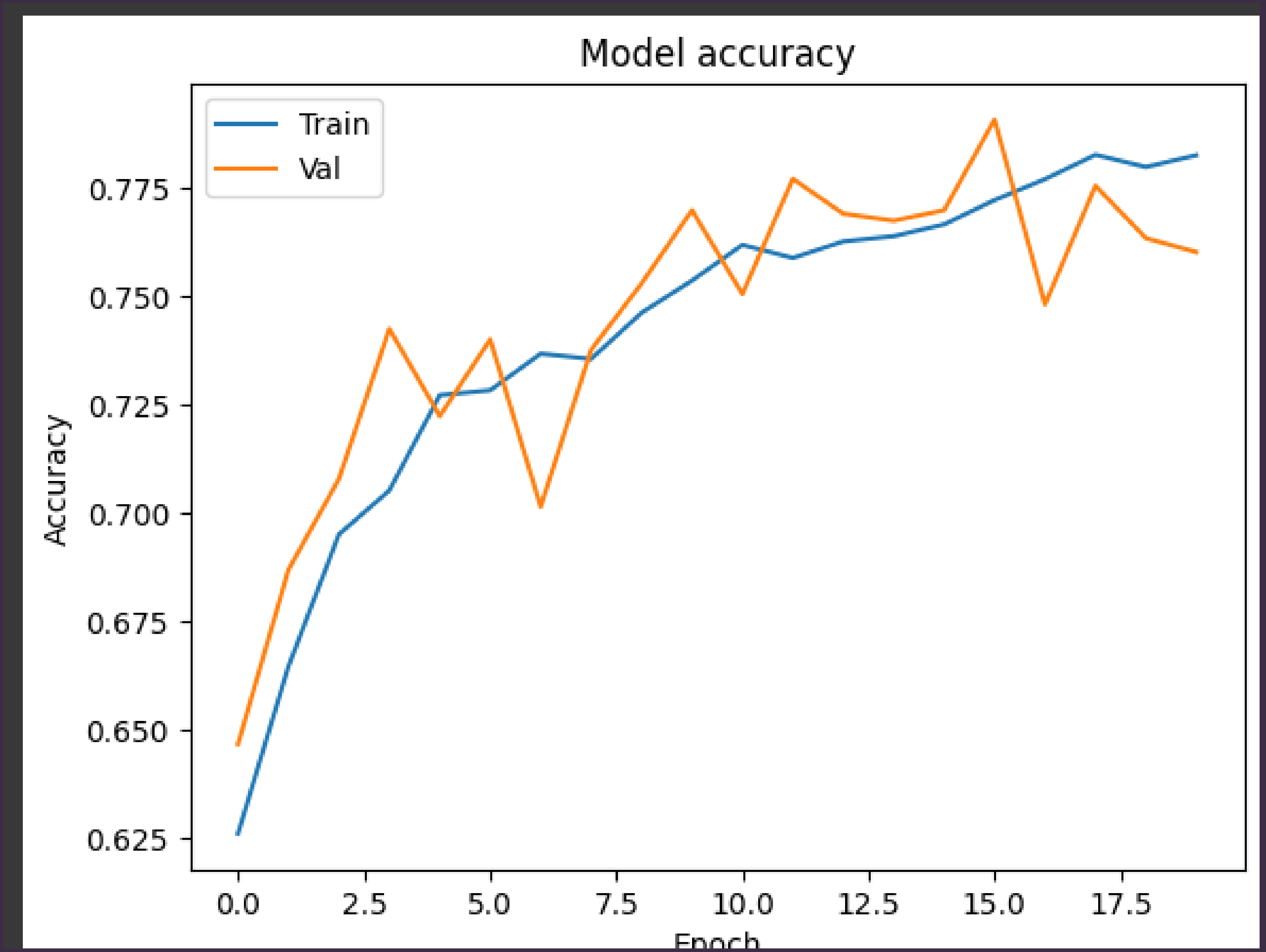
Compilation & Training: Compiled the model using the Adam optimizer and categorical crossentropy loss. Implemented EarlyStopping on validation loss to prevent overfitting, optimizing training duration and model performance.

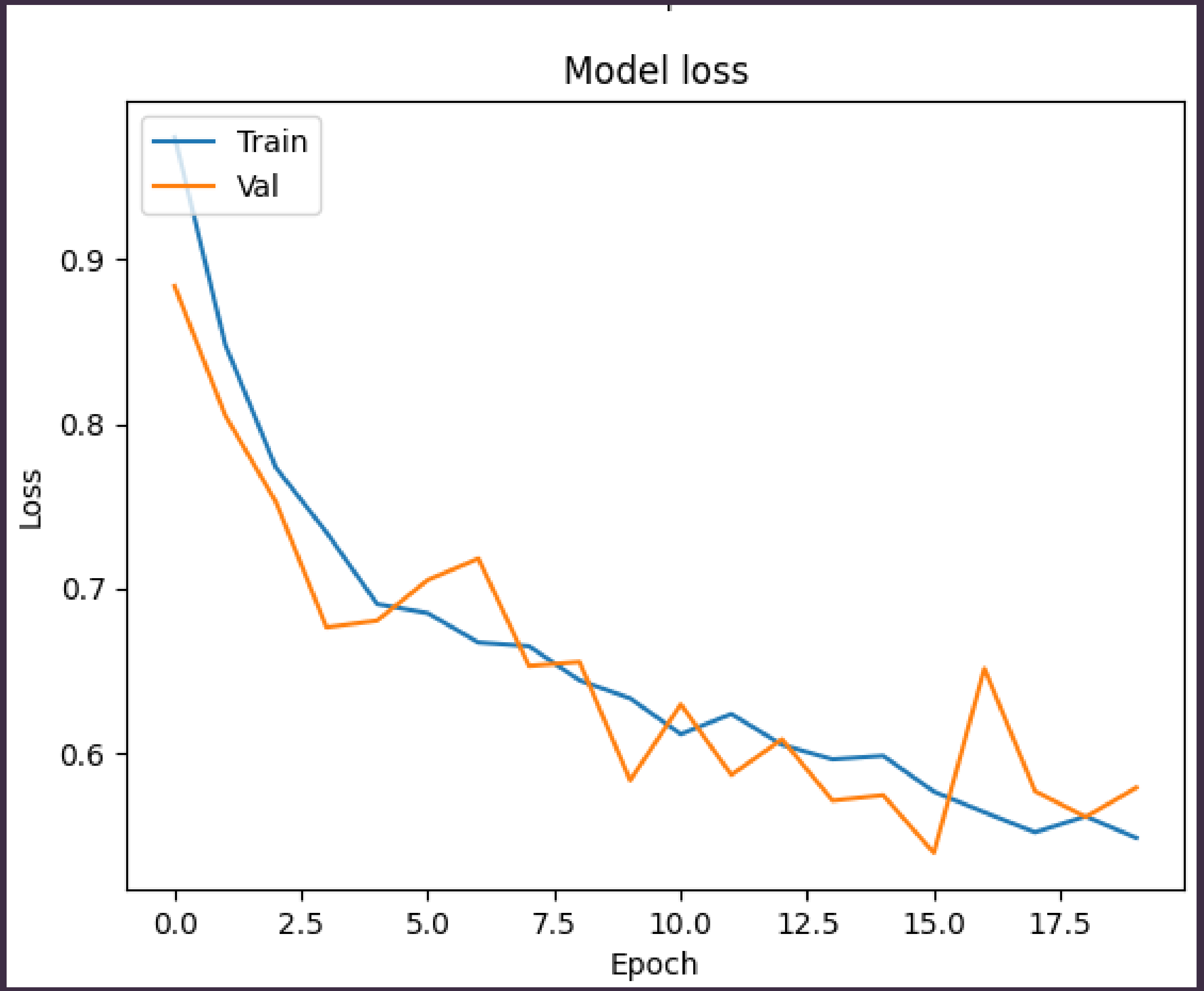
Convolutional Neural Network

```
history = model.fit(
        train_generator,
        validation_data=val_generator,
        epochs= 30,
        callbacks = [early_stopping]
)
```

```
Epoch 14/30
312/312 [==============================] - 135s 433ms/step - loss: 0.5961 - accuracy: 0.7637 - val_loss: 0.5710 - val_accuracy: 0.7673
Epoch 15/30
312/312 [==============================] - 136s 435ms/step - loss: 0.5982 - accuracy: 0.7664 - val_loss: 0.5743 - val_accuracy: 0.7697
Epoch 16/30
312/312 [==============================] - 136s 436ms/step - loss: 0.5765 - accuracy: 0.7721 - val_loss: 0.5392 - val_accuracy: 0.7907
Epoch 17/30
312/312 [==============================] - 137s 441ms/step - loss: 0.5639 - accuracy: 0.7769 - val_loss: 0.6512 - val_accuracy: 0.7480
Epoch 18/30
312/312 [==============================] - 136s 435ms/step - loss: 0.5518 - accuracy: 0.7825 - val_loss: 0.5767 - val_accuracy: 0.7754
Epoch 19/30
312/312 [==============================] - 135s 432ms/step - loss: 0.5614 - accuracy: 0.7797 - val_loss: 0.5611 - val_accuracy: 0.7633
Epoch 20/30
312/312 [==============================] - 138s 441ms/step - loss: 0.5483 - accuracy: 0.7824 - val_loss: 0.5788 - val_accuracy: 0.7601
```

CNN Plot


Model accuracy
Train
Val
Accuracy
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


Model loss
Train
Val
Loss
0.9
0.8
0.7
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

VGG16

Base Model: Incorporated a pretrained VGG16 model without the top layer, using 'imagenet' weights and max pooling. Configured for 224x224 color images.

Extensions: Added Flatten for feature vectorization, a Dense layer with 512 neurons (ReLU), Batch Normalization for stability, and a Dropout of 0.5 to prevent overfitting.

Output: Appended a final Dense layer with 4 neurons (softmax activation) for classification.

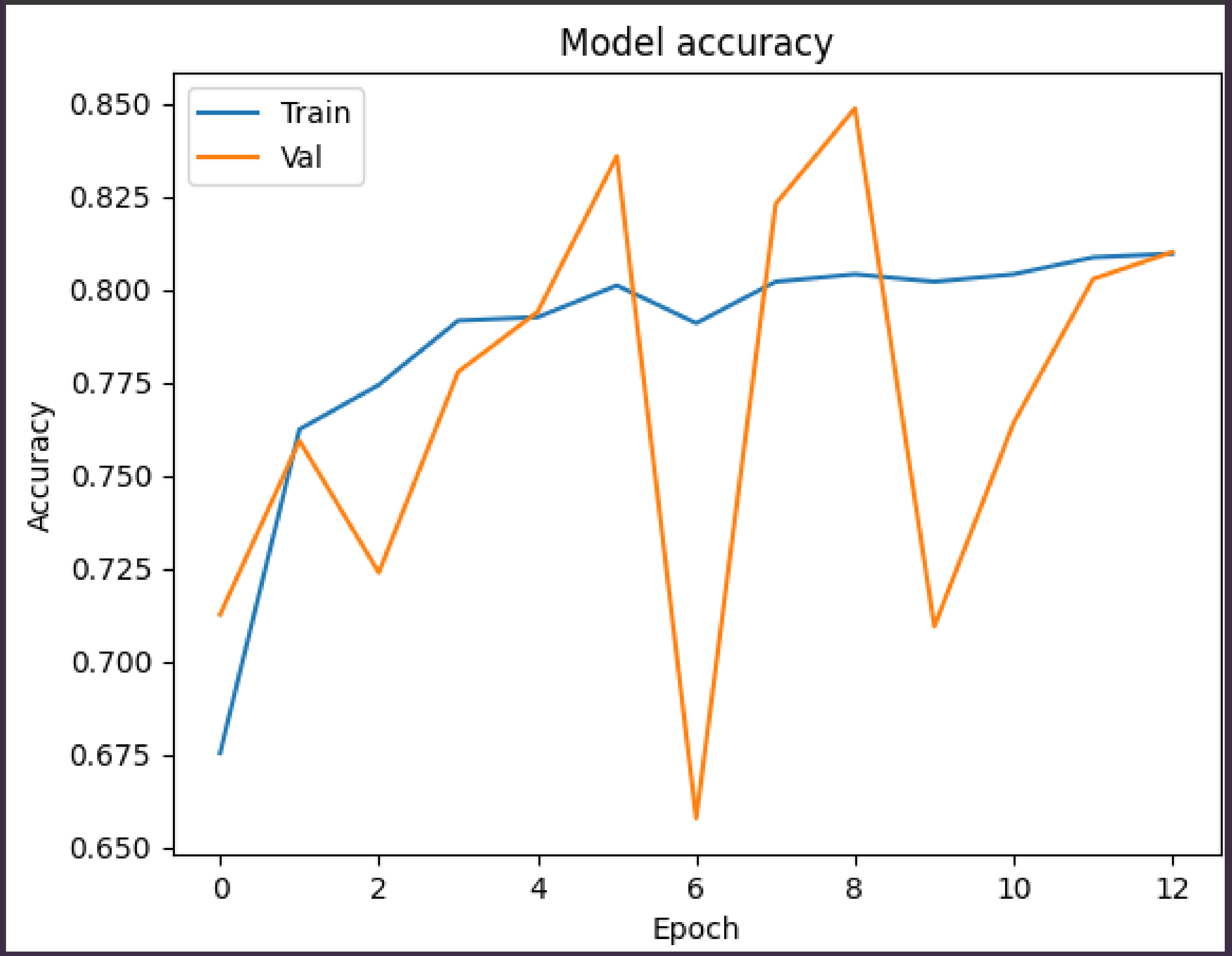
Final SetuP: Froze the pretrained layers, compiled with Adam optimizer, and categorical crossentropy loss, focusing on accuracy.

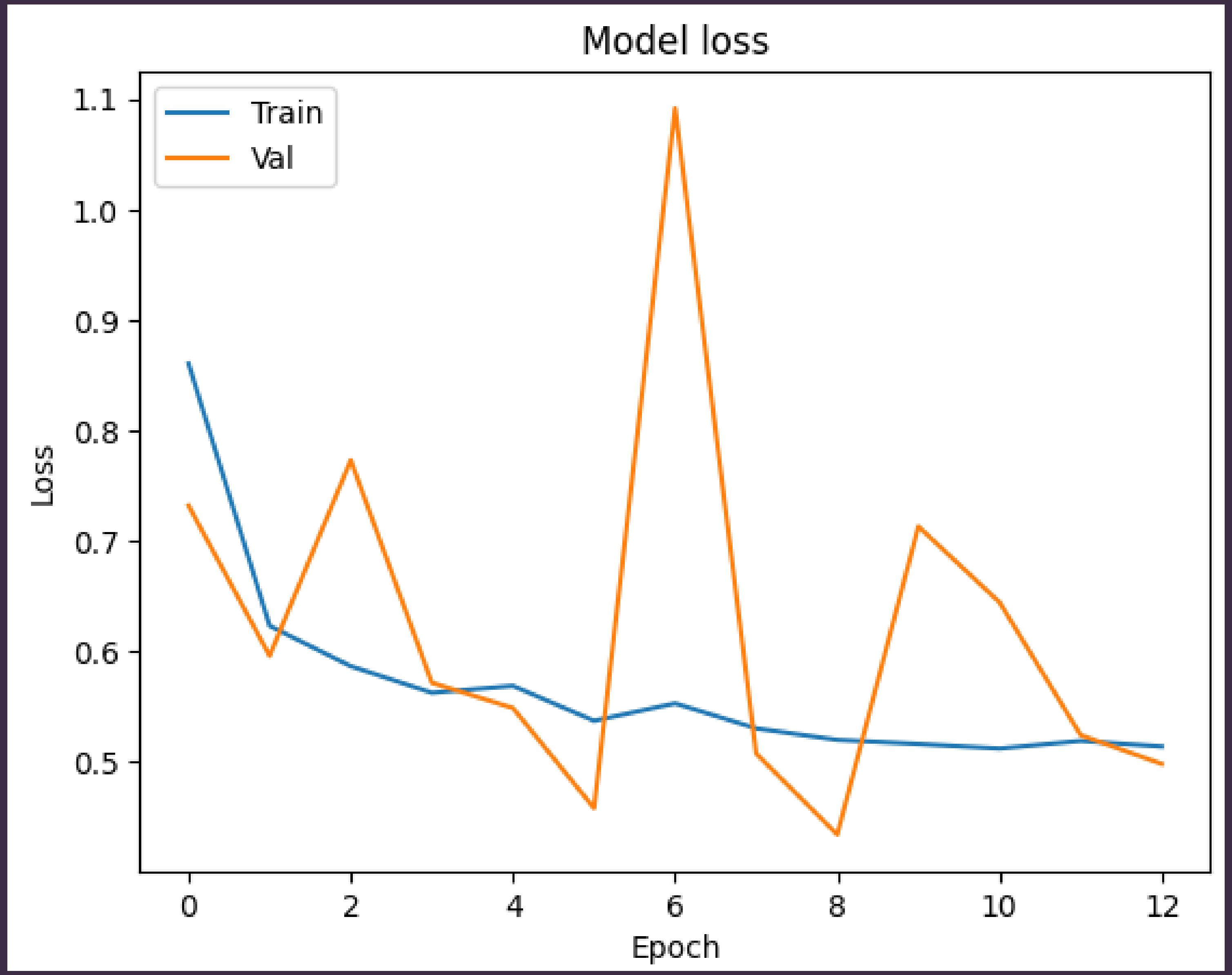
VGG16

```python
history_1 = VGG_model.fit(
    train_generator,
    validation_data=val_generator,
    epochs= 30,
    callbacks = [early_stopping])
```

```
Epoch 1/30
312/312 [==============================] - 152s 482ms/step - loss: 0.8604 - accuracy: 0.6753 - val_loss: 0.7318 - val_accuracy: 0.7126
Epoch 2/30
312/312 [==============================] - 141s 452ms/step - loss: 0.6229 - accuracy: 0.7623 - val_loss: 0.5958 - val_accuracy: 0.7593
Epoch 3/30
312/312 [==============================] - 141s 453ms/step - loss: 0.5864 - accuracy: 0.7743 - val_loss: 0.7729 - val_accuracy: 0.7238
Epoch 4/30
312/312 [==============================] - 142s 454ms/step - loss: 0.5624 - accuracy: 0.7916 - val_loss: 0.5714 - val_accuracy: 0.7778
Epoch 5/30
312/312 [==============================] - 141s 452ms/step - loss: 0.5686 - accuracy: 0.7925 - val_loss: 0.5486 - val_accuracy: 0.7939
Epoch 6/30
312/312 [==============================] - 146s 467ms/step - loss: 0.5370 - accuracy: 0.8010 - val_loss: 0.4579 - val_accuracy: 0.8357
Epoch 7/30
312/312 [==============================] - 144s 461ms/step - loss: 0.5528 - accuracy: 0.7909 - val_loss: 1.0918 - val_accuracy: 0.6578
Epoch 8/30
312/312 [==============================] - 143s 457ms/step - loss: 0.5301 - accuracy: 0.8020 - val_loss: 0.5073 - val_accuracy: 0.8229
Epoch 9/30
312/312 [==============================] - 143s 460ms/step - loss: 0.5197 - accuracy: 0.8040 - val_loss: 0.4340 - val_accuracy: 0.8486
Epoch 10/30
312/312 [==============================] - 143s 458ms/step - loss: 0.5160 - accuracy: 0.8020 - val_loss: 0.7129 - val_accuracy: 0.7093
Epoch 11/30
312/312 [==============================] - 147s 471ms/step - loss: 0.5118 - accuracy: 0.8040 - val_loss: 0.6443 - val_accuracy: 0.7641
Epoch 12/30
312/312 [==============================] - 143s 459ms/step - loss: 0.5187 - accuracy: 0.8085 - val_loss: 0.5240 - val_accuracy: 0.8027
Epoch 13/30
312/312 [==============================] - 145s 464ms/step - loss: 0.5137 - accuracy: 0.8095 - val_loss: 0.4979 - val_accuracy: 0.8100
```

VGG16 Plot


Model accuracy
Train
Val
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0.675
0.650
Accuracy
0
2
4
6
8
10
12
Epoch


Model loss
Train
Val
1.1
1.0
0.9
0.8
0.7
0.6
0.5
Loss
0
2
4
6
8
10
12
Epoch

DENSENET169

Base Model: Imported DenseNet169 from TensorFlow, pretrained on ImageNet, with the top layer omitted, set to process 224x224 color images.

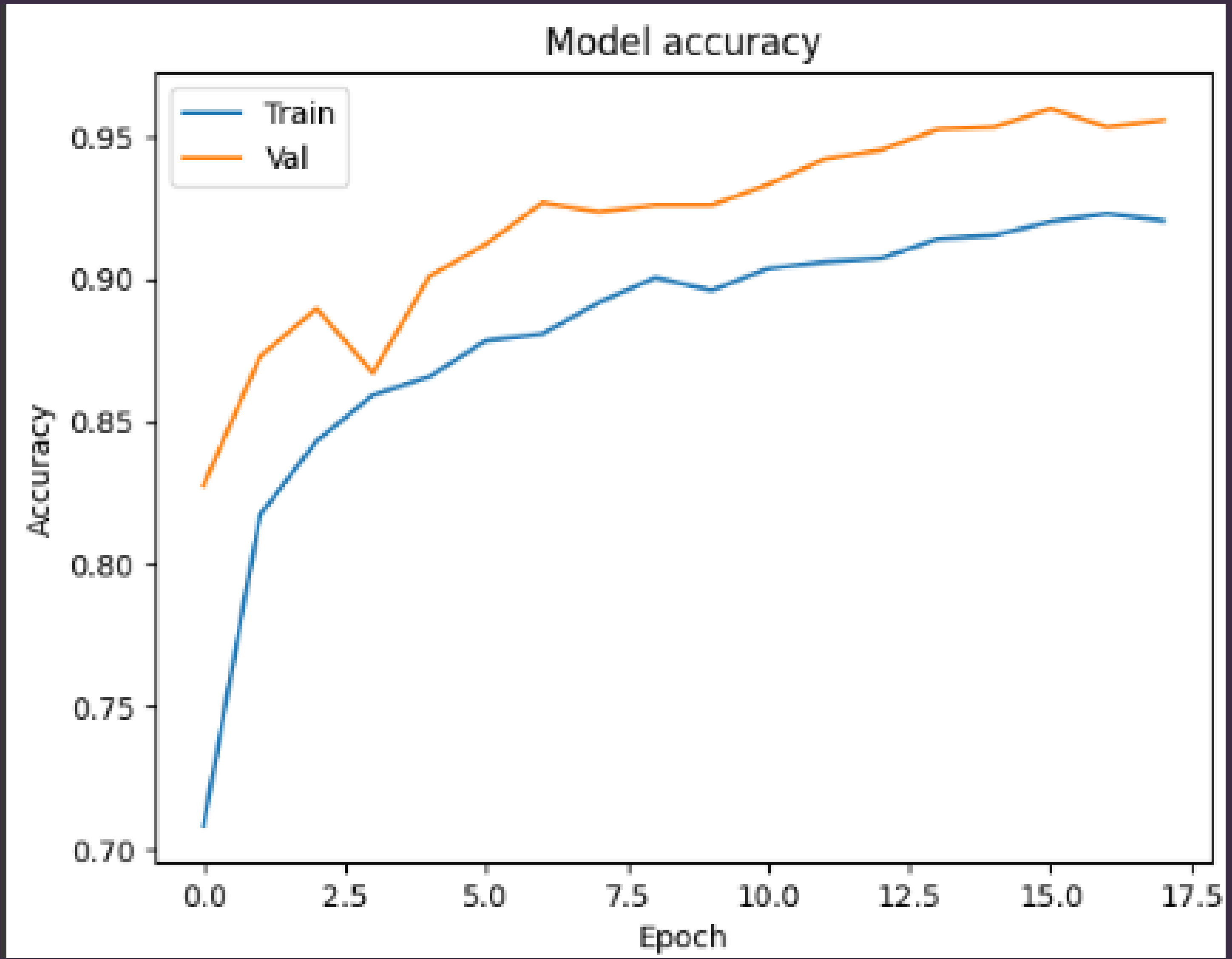
Layer Freezing: All layers in the DenseNet169 base model are set to non-trainable to preserve the learned ImageNet features during further training.

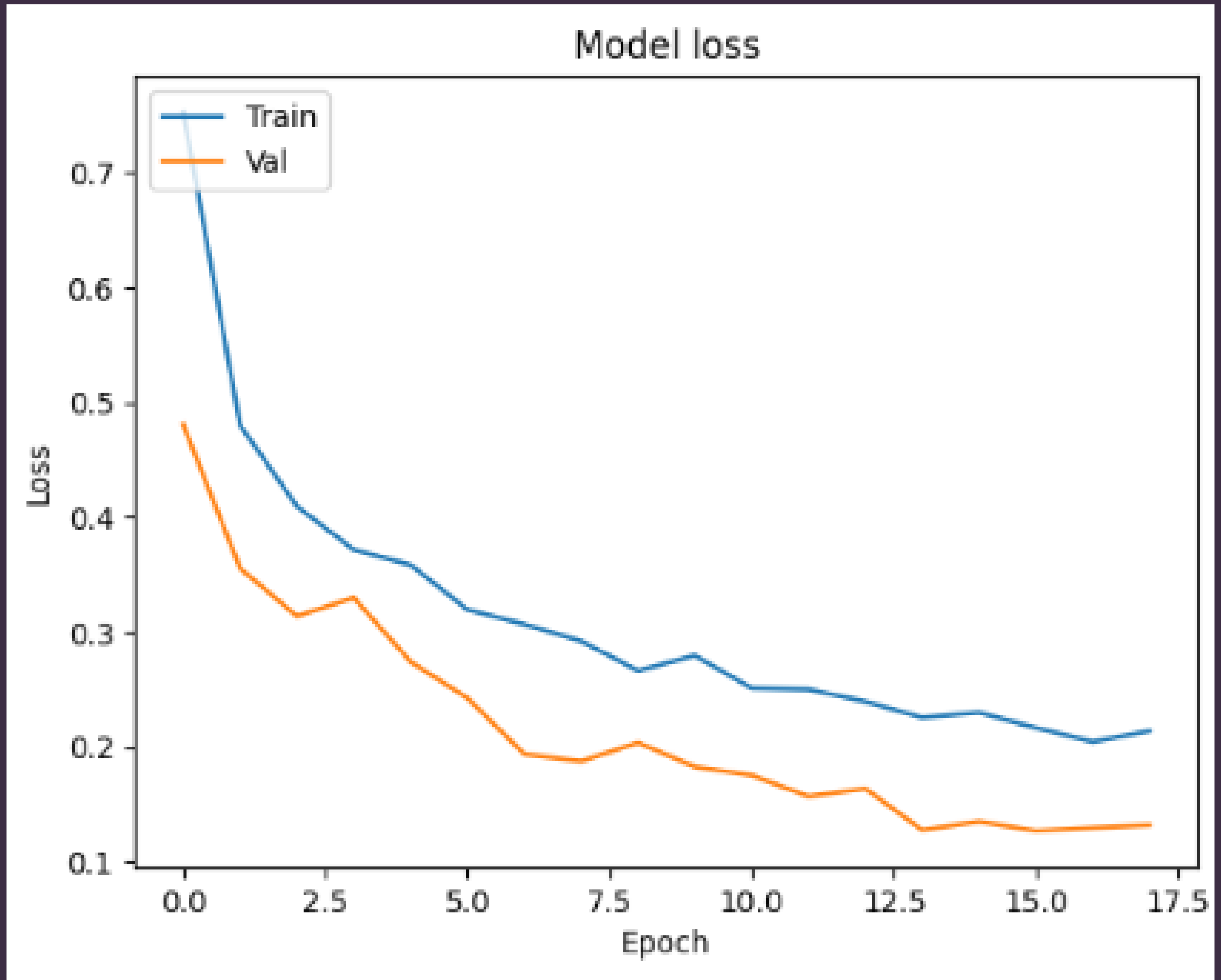
New Layers: Built upon the base model by adding:

- GlobalAveragePooling2D to condense feature maps into a single vector per map, reducing the number of parameters.
- Dense layer with 512 neurons (ReLU activation) for high-level feature learning.
- Dropout layer (0.5) to reduce overfitting risk.
- Final Dense layer with 4 outputs (softmax activation) tailored for classification into four classes.

Model Compilation: The complete model is compiled using an Adam optimizer (learning rate of 0.001), with categorical crossentropy as the loss function and accuracy as the performance metric.

DENSENET169 Plot





DENSENET169

```
history_3 = model_3.fit(
    train_generator,
    validation_data=val_generator,
    epochs= 30,
    callbacks = [early_stopping]
)
```

```
Epoch 1/30
312/312 [==============================] - 170s 493ms/step - loss: 0.7529 - accuracy: 0.7084 - val_loss: 0.4804 - val_accuracy: 0.8277
Epoch 2/30
312/312 [==============================] - 145s 464ms/step - loss: 0.4800 - accuracy: 0.8173 - val_loss: 0.3554 - val_accuracy: 0.8728
Epoch 3/30
312/312 [==============================] - 146s 467ms/step - loss: 0.4093 - accuracy: 0.8432 - val_loss: 0.3137 - val_accuracy: 0.8897
Epoch 4/30
312/312 [==============================] - 146s 467ms/step - loss: 0.3715 - accuracy: 0.8594 - val_loss: 0.3298 - val_accuracy: 0.8671
Epoch 5/30
312/312 [==============================] - 150s 482ms/step - loss: 0.3583 - accuracy: 0.8658 - val_loss: 0.2740 - val_accuracy: 0.9010
Epoch 6/30
312/312 [==============================] - 147s 470ms/step - loss: 0.3194 - accuracy: 0.8785 - val_loss: 0.2424 - val_accuracy: 0.9122
Epoch 7/30
312/312 [==============================] - 146s 468ms/step - loss: 0.3066 - accuracy: 0.8808 - val_loss: 0.1935 - val_accuracy: 0.9267
Epoch 8/30
312/312 [==============================] - 147s 471ms/step - loss: 0.2920 - accuracy: 0.8918 - val_loss: 0.1874 - val_accuracy: 0.9235
Epoch 9/30
312/312 [==============================] - 148s 474ms/step - loss: 0.2663 - accuracy: 0.9005 - val_loss: 0.2035 - val_accuracy: 0.9259
Epoch 10/30
312/312 [==============================] - 147s 471ms/step - loss: 0.2794 - accuracy: 0.8960 - val_loss: 0.1826 - val_accuracy: 0.9259
Epoch 11/30
312/312 [==============================] - 146s 467ms/step - loss: 0.2510 - accuracy: 0.9037 - val_loss: 0.1751 - val_accuracy: 0.9332
Epoch 12/30
312/312 [==============================] - 145s 466ms/step - loss: 0.2500 - accuracy: 0.9060 - val_loss: 0.1568 - val_accuracy: 0.9420
Epoch 13/30
312/312 [==============================] - 146s 468ms/step - loss: 0.2393 - accuracy: 0.9073 - val_loss: 0.1635 - val_accuracy: 0.9452
Epoch 14/30
312/312 [==============================] - 149s 476ms/step - loss: 0.2254 - accuracy: 0.9140 - val_loss: 0.1274 - val_accuracy: 0.9525
Epoch 15/30
312/312 [==============================] - 147s 471ms/step - loss: 0.2298 - accuracy: 0.9153 - val_loss: 0.1349 - val_accuracy: 0.9533
Epoch 16/30
312/312 [==============================] - 146s 469ms/step - loss: 0.2164 - accuracy: 0.9201 - val_loss: 0.1267 - val_accuracy: 0.9597
Epoch 17/30
312/312 [==============================] - 146s 467ms/step - loss: 0.2043 - accuracy: 0.9230 - val_loss: 0.1294 - val_accuracy: 0.9533
Epoch 18/30
312/312 [==============================] - 147s 471ms/step - loss: 0.2137 - accuracy: 0.9204 - val_loss: 0.1318 - val_accuracy: 0.9557
```

MOBILENET169

Base Model: Utilized MobileNet pretrained on ImageNet, configured without the top layer and set for 224x224 color image inputs.

Layer Freezing: Froze all layers in the base MobileNet model to maintain pretrained features during training.

Model Construction: Created a new Sequential model adding:

- GlobalAveragePooling2D to reduce spatial dimensions.
- Dense layer with 512 neurons (ReLU activation) for deeper feature learning.
- Dropout layer (0.5) to minimize overfitting.
- Final Dense layer with 4 outputs (softmax activation) for class probability.

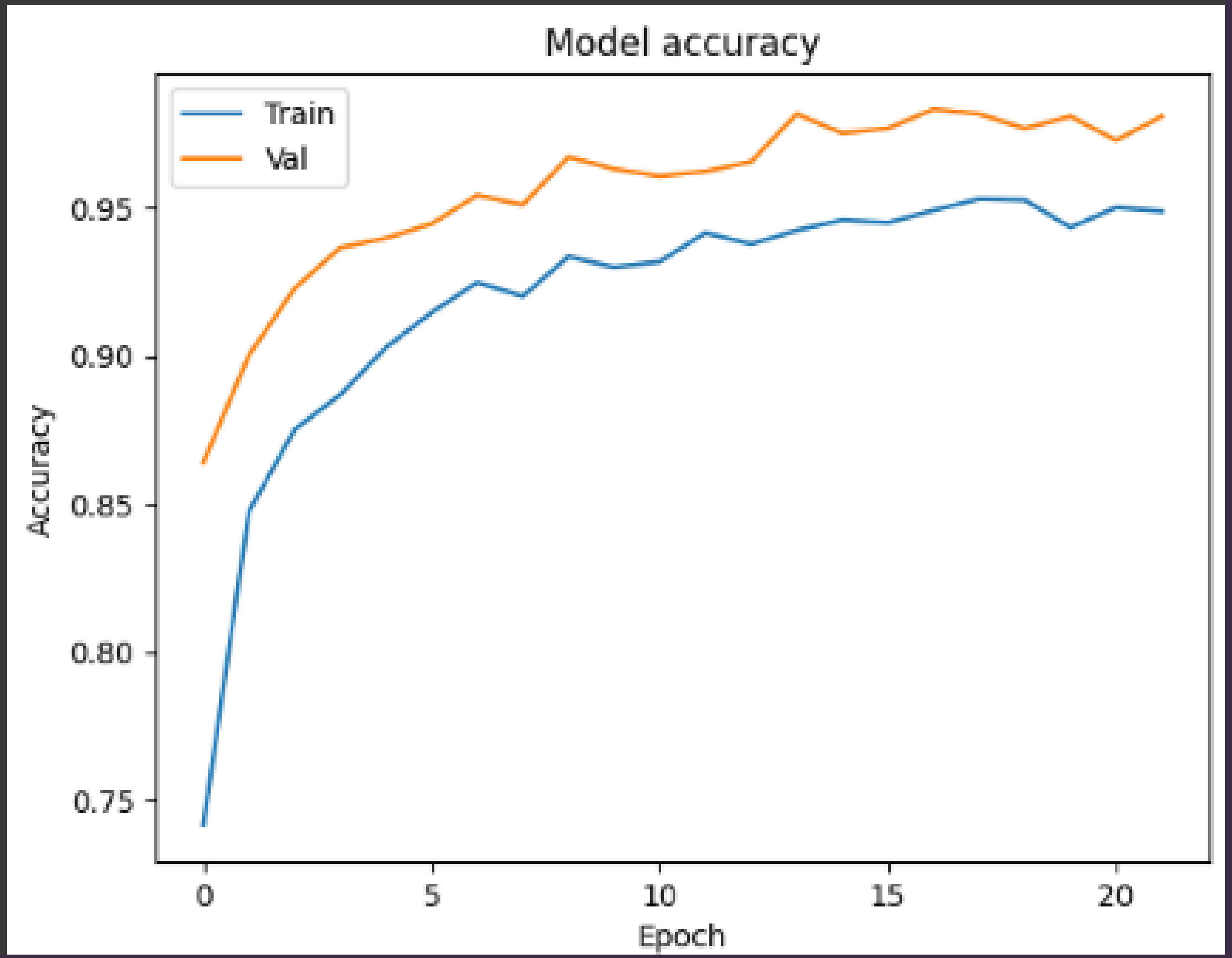
Compilation: Compiled the model with an Adam optimizer (learning rate of 0.001), using categorical crossentropy for loss and tracking accuracy as the metric.

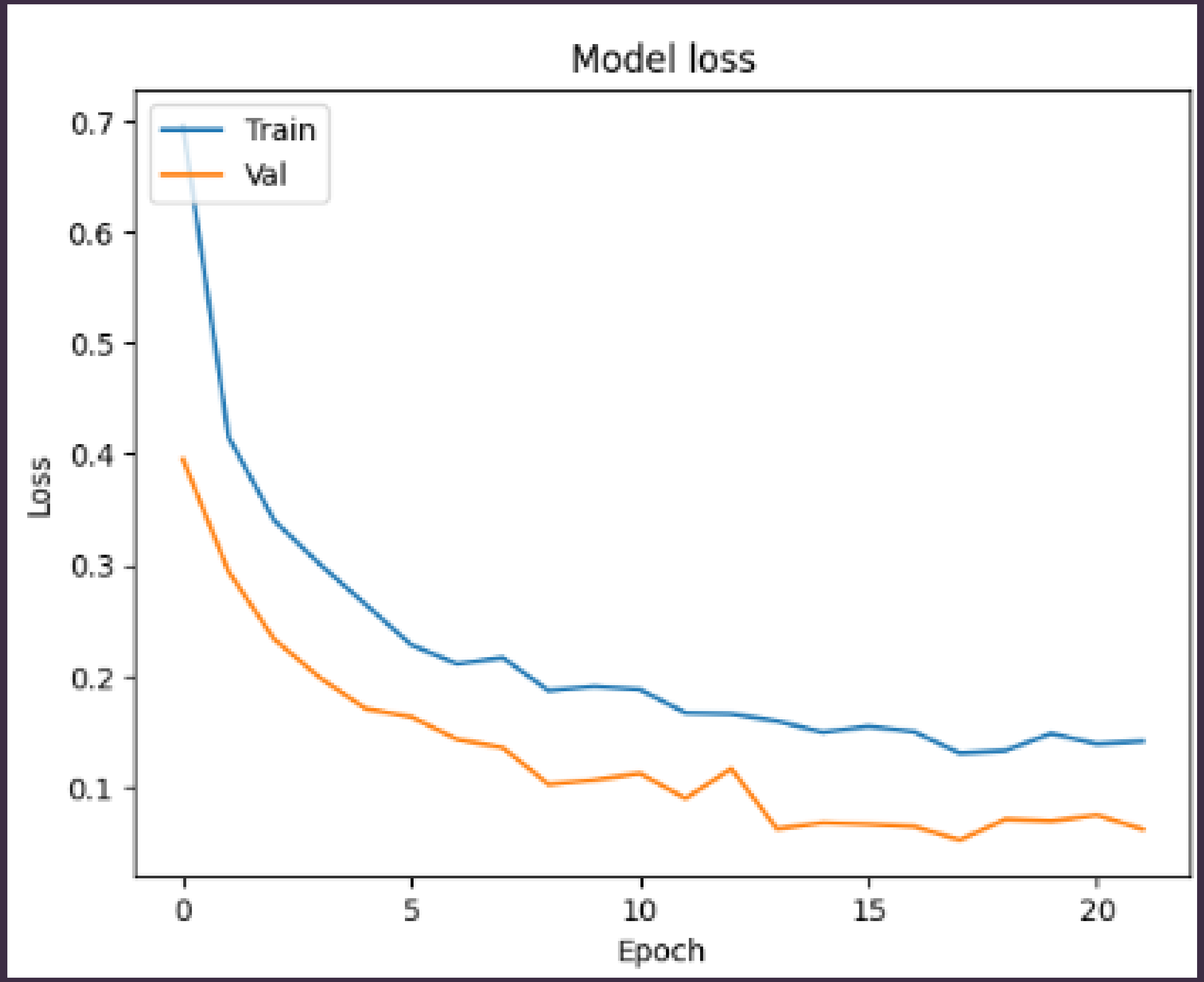
MOBILENET

```
history_4 = mobmodel.fit(
    train_generator,
    validation_data=val_generator,
    epochs= 30,
    callbacks = [early_stopping]
)
```

```
Epoch 1/30
312/312 [==============================] - 144s 453ms/step - loss: 0.6961 - accuracy: 0.7415 - val_loss: 0.3953 - val_accuracy: 0.8639
Epoch 2/30
312/312 [==============================] - 138s 443ms/step - loss: 0.4154 - accuracy: 0.8474 - val_loss: 0.2941 - val_accuracy: 0.9002
Epoch 3/30
312/312 [==============================] - 139s 447ms/step - loss: 0.3400 - accuracy: 0.8750 - val_loss: 0.2330 - val_accuracy: 0.9227
Epoch 4/30
312/312 [==============================] - 140s 449ms/step - loss: 0.3006 - accuracy: 0.8868 - val_loss: 0.1984 - val_accuracy: 0.9364
Epoch 5/30
312/312 [==============================] - 142s 455ms/step - loss: 0.2645 - accuracy: 0.9027 - val_loss: 0.1705 - val_accuracy: 0.9396
Epoch 6/30
312/312 [==============================] - 139s 444ms/step - loss: 0.2280 - accuracy: 0.9145 - val_loss: 0.1631 - val_accuracy: 0.9444
Epoch 7/30
312/312 [==============================] - 140s 449ms/step - loss: 0.2108 - accuracy: 0.9247 - val_loss: 0.1429 - val_accuracy: 0.9541
Epoch 8/30
312/312 [==============================] - 143s 459ms/step - loss: 0.2164 - accuracy: 0.9200 - val_loss: 0.1354 - val_accuracy: 0.9509
Epoch 9/30
312/312 [==============================] - 139s 445ms/step - loss: 0.1868 - accuracy: 0.9334 - val_loss: 0.1024 - val_accuracy: 0.9670
Epoch 10/30
312/312 [==============================] - 143s 459ms/step - loss: 0.1906 - accuracy: 0.9298 - val_loss: 0.1062 - val_accuracy: 0.9630
Epoch 11/30
312/312 [==============================] - 138s 443ms/step - loss: 0.1878 - accuracy: 0.9317 - val_loss: 0.1122 - val_accuracy: 0.9605
Epoch 12/30
312/312 [==============================] - 138s 443ms/step - loss: 0.1667 - accuracy: 0.9414 - val_loss: 0.0896 - val_accuracy: 0.9622
Epoch 13/30
312/312 [==============================] - 139s 445ms/step - loss: 0.1659 - accuracy: 0.9376 - val_loss: 0.1164 - val_accuracy: 0.9654
Epoch 14/30
312/312 [==============================] - 138s 443ms/step - loss: 0.1594 - accuracy: 0.9422 - val_loss: 0.0623 - val_accuracy: 0.9815
Epoch 15/30
312/312 [==============================] - 139s 447ms/step - loss: 0.1493 - accuracy: 0.9458 - val_loss: 0.0677 - val_accuracy: 0.9750
Epoch 16/30
312/312 [==============================] - 138s 442ms/step - loss: 0.1545 - accuracy: 0.9449 - val_loss: 0.0664 - val_accuracy: 0.9767
Epoch 17/30
312/312 [==============================] - 139s 447ms/step - loss: 0.1497 - accuracy: 0.9490 - val_loss: 0.0644 - val_accuracy: 0.9831
Epoch 18/30
312/312 [==============================] - 137s 440ms/step - loss: 0.1304 - accuracy: 0.9529 - val_loss: 0.0522 - val_accuracy: 0.9815
Epoch 19/30
312/312 [==============================] - 138s 441ms/step - loss: 0.1329 - accuracy: 0.9525 - val_loss: 0.0709 - val_accuracy: 0.9767
Epoch 20/30
312/312 [==============================] - 138s 443ms/step - loss: 0.1481 - accuracy: 0.9432 - val_loss: 0.0692 - val_accuracy: 0.9807
Epoch 21/30
312/312 [==============================] - 139s 447ms/step - loss: 0.1388 - accuracy: 0.9500 - val_loss: 0.0746 - val_accuracy: 0.9726
Epoch 22/30
312/312 [==============================] - 140s 449ms/step - loss: 0.1412 - accuracy: 0.9487 - val_loss: 0.0619 - val_accuracy: 0.9807
```

MOBILENET Plot





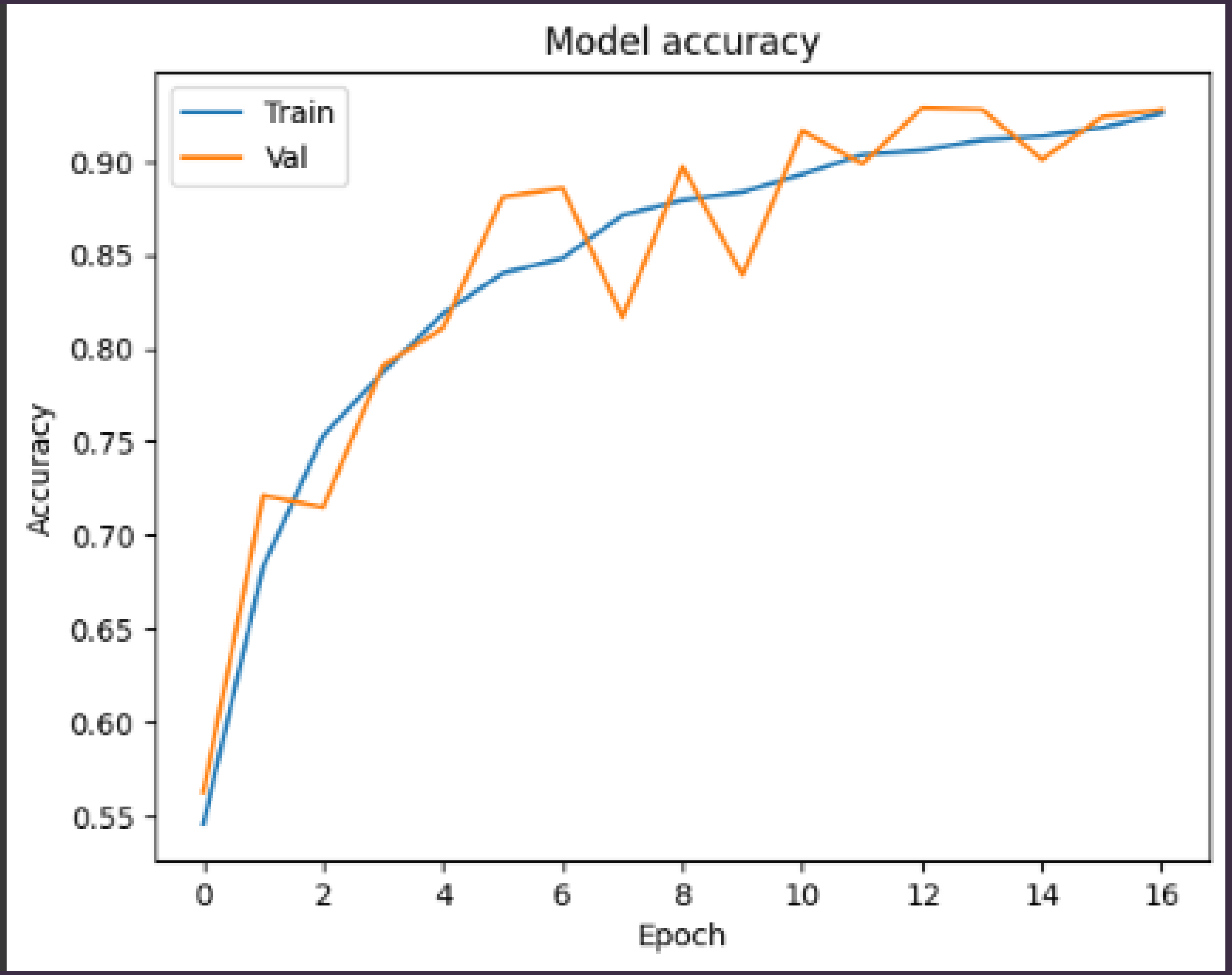
INCEPTIONV3

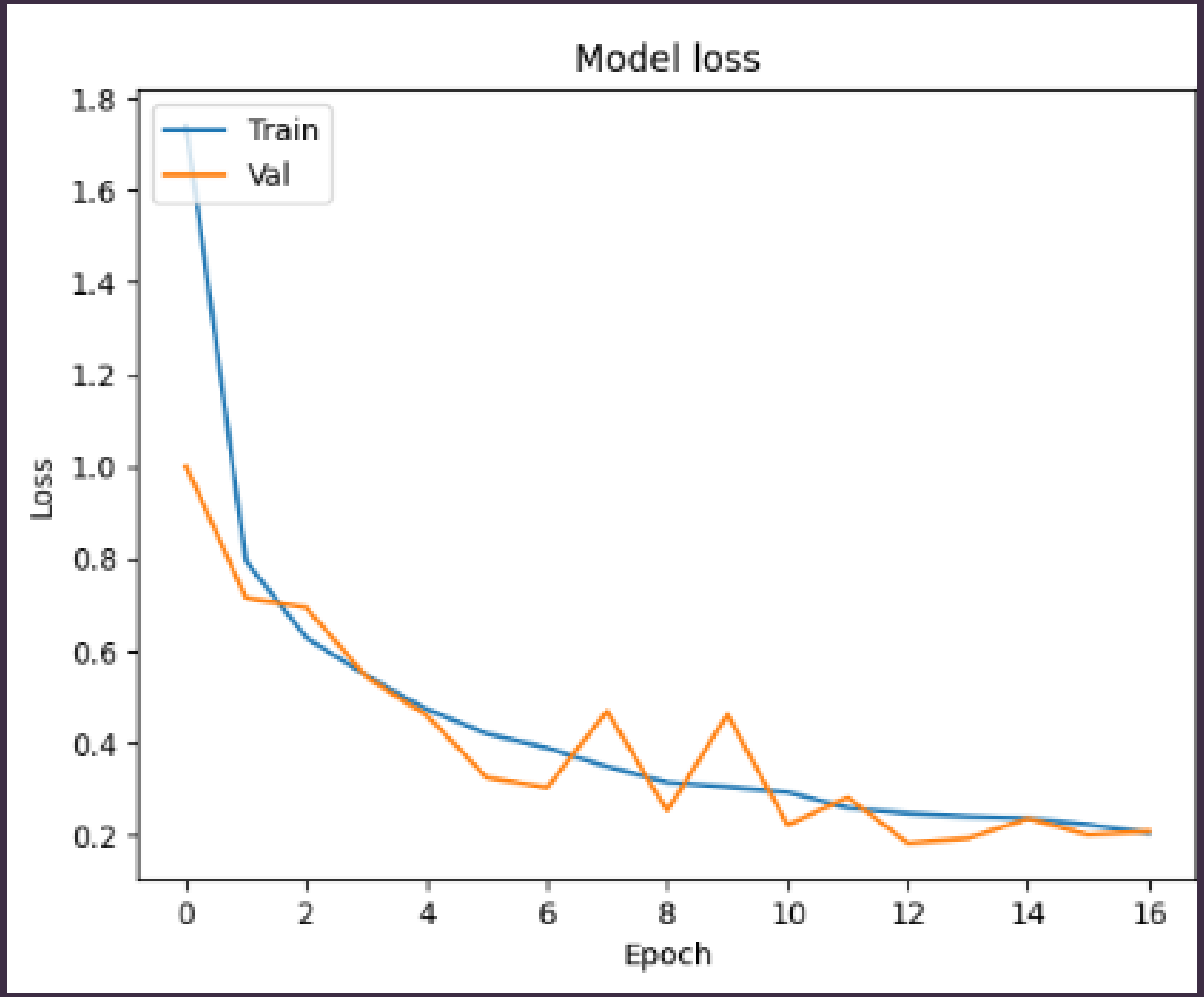
- Model Base: Uses InceptionV3, pre-trained on the ImageNet dataset, excluding the top layer to allow custom modifications.
- Custom Layers: Adds a Global Average Pooling layer and two Dense layers (1024 units with ReLU and 4 units with softmax) for task-specific learning.
- Layer Freezing: Freezes pre-trained layers to preserve learned features, focusing training on the newly added layers.
- Metrics: Compiles the model with precision, recall, accuracy, and F1 score to evaluate performance comprehensively.
- Early Stopping: Implements early stopping to halt training if validation loss does not improve, optimizing computational efficiency and preventing overfitting.

MOBILENET

```
history_5 = inception_model.fit(
    train_generator,
    validation_data=val_generator,
    epochs= 30,
    callbacks = [early_stopping]
)
```

```
Epoch 1/30
156/156 [==============================] - 175s 1s/step - loss: 1.7409 - accuracy: 0.5456 - precision: 0.5977 - recall: 0.4728 - f1_score: 0.4497 - val_loss: 0.9998 - val_accuracy: 0.5628 - val_precision: 0.6449 - val_recall: 0.4928 - val_f1_score: 0.4990
Epoch 2/30
156/156 [==============================] - 164s 1s/step - loss: 0.7940 - accuracy: 0.6834 - precision: 0.7443 - recall: 0.6096 - f1_score: 0.6021 - val_loss: 0.7147 - val_accuracy: 0.7214 - val_precision: 0.7880 - val_recall: 0.6433 - val_f1_score: 0.6511
Epoch 3/30
156/156 [==============================] - 163s 1s/step - loss: 0.6277 - accuracy: 0.7531 - precision: 0.8002 - recall: 0.6971 - f1_score: 0.6890 - val_loss: 0.6937 - val_accuracy: 0.7150 - val_precision: 0.7466 - val_recall: 0.6691 - val_f1_score: 0.6561
Epoch 4/30
156/156 [==============================] - 164s 1s/step - loss: 0.5459 - accuracy: 0.7870 - precision: 0.8238 - recall: 0.7464 - f1_score: 0.7300 - val_loss: 0.5427 - val_accuracy: 0.7907 - val_precision: 0.8309 - val_recall: 0.7319 - val_f1_score: 0.7697
Epoch 5/30
156/156 [==============================] - 164s 1s/step - loss: 0.4725 - accuracy: 0.8185 - precision: 0.8490 - recall: 0.7881 - f1_score: 0.7720 - val_loss: 0.4593 - val_accuracy: 0.8108 - val_precision: 0.8412 - val_recall: 0.7890 - val_f1_score: 0.7121
Epoch 6/30
156/156 [==============================] - 165s 1s/step - loss: 0.4193 - accuracy: 0.8401 - precision: 0.8626 - recall: 0.8136 - f1_score: 0.7990 - val_loss: 0.3244 - val_accuracy: 0.8808 - val_precision: 0.9052 - val_recall: 0.8607 - val_f1_score: 0.8460
Epoch 7/30
156/156 [==============================] - 164s 1s/step - loss: 0.3888 - accuracy: 0.8480 - precision: 0.8681 - recall: 0.8267 - f1_score: 0.8070 - val_loss: 0.3025 - val_accuracy: 0.8857 - val_precision: 0.9107 - val_recall: 0.8623 - val_f1_score: 0.8557
Epoch 8/30
156/156 [==============================] - 165s 1s/step - loss: 0.3482 - accuracy: 0.8710 - precision: 0.8885 - recall: 0.8529 - f1_score: 0.8365 - val_loss: 0.4681 - val_accuracy: 0.8164 - val_precision: 0.8394 - val_recall: 0.7995 - val_f1_score: 0.7687
Epoch 9/30
156/156 [==============================] - 165s 1s/step - loss: 0.3138 - accuracy: 0.8792 - precision: 0.8953 - recall: 0.8649 - f1_score: 0.8480 - val_loss: 0.2514 - val_accuracy: 0.8969 - val_precision: 0.9126 - val_recall: 0.8833 - val_f1_score: 0.8633
Epoch 10/30
156/156 [==============================] - 164s 1s/step - loss: 0.3037 - accuracy: 0.8835 - precision: 0.8969 - recall: 0.8707 - f1_score: 0.8525 - val_loss: 0.4619 - val_accuracy: 0.8390 - val_precision: 0.8474 - val_recall: 0.8269 - val_f1_score: 0.7966
Epoch 11/30
156/156 [==============================] - 165s 1s/step - loss: 0.2924 - accuracy: 0.8930 - precision: 0.9040 - recall: 0.8805 - f1_score: 0.8650 - val_loss: 0.2210 - val_accuracy: 0.9163 - val_precision: 0.9264 - val_recall: 0.9114 - val_f1_score: 0.8944
Epoch 12/30
156/156 [==============================] - 164s 1s/step - loss: 0.2579 - accuracy: 0.9035 - precision: 0.9141 - recall: 0.8927 - f1_score: 0.8781 - val_loss: 0.2806 - val_accuracy: 0.8986 - val_precision: 0.9072 - val_recall: 0.8897 - val_f1_score: 0.8751
Epoch 13/30
156/156 [==============================] - 164s 1s/step - loss: 0.2459 - accuracy: 0.9059 - precision: 0.9151 - recall: 0.8966 - f1_score: 0.8811 - val_loss: 0.1818 - val_accuracy: 0.9283 - val_precision: 0.9333 - val_recall: 0.9243 - val_f1_score: 0.9130
Epoch 14/30
156/156 [==============================] - 162s 1s/step - loss: 0.2390 - accuracy: 0.9113 - precision: 0.9202 - recall: 0.9032 - f1_score: 0.8889 - val_loss: 0.1927 - val_accuracy: 0.9275 - val_precision: 0.9355 - val_recall: 0.9227 - val_f1_score: 0.9119
Epoch 15/30
156/156 [==============================] - 164s 1s/step - loss: 0.2352 - accuracy: 0.9133 - precision: 0.9209 - recall: 0.9054 - f1_score: 0.8905 - val_loss: 0.2348 - val_accuracy: 0.9010 - val_precision: 0.9072 - val_recall: 0.8969 - val_f1_score: 0.8815
Epoch 16/30
156/156 [==============================] - 165s 1s/step - loss: 0.2218 - accuracy: 0.9178 - precision: 0.9246 - recall: 0.9114 - f1_score: 0.8976 - val_loss: 0.1999 - val_accuracy: 0.9235 - val_precision: 0.9331 - val_recall: 0.9211 - val_f1_score: 0.9036
Epoch 17/30
```


Model accuracy
Train
Val
Accuracy
Epoch
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0
2
4
6
8
10
12
14
16


Model loss
Train
Val
Loss
Epoch
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
0
2
4
6
8
10
12
14
16

OUTPUT

MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE
CNN	83	84	82	0.7495
DENSENET169	93	92	93	92
MOBILENET	96	96	96	95
VGG16	82	83	77	76
INCEPTIONV3	93	93	92	90

CONCLUSION

Mobilenet emerged as the top-performing model with impressive scores across all metrics: 96% accuracy, 96% precision, 96% recall, and an F1-score of 95. Its high scores indicate strong overall performance and balance in identifying each class correctly and consistently.
DenseNet169 and **InceptionV3** both performed very well, with DenseNet169 slightly outperforming InceptionV3 in terms of F1-score (92 vs. 90). Both models demonstrated high accuracy (93%), precision (92-93%), and recall (92-93%), making them reliable choices for tasks requiring robust feature extraction capabilities.
CNN and **VGG16** showed lower performance compared to the other models. CNN achieved an accuracy of 83%, precision of 84%, recall of 82%, and an F1-score of 0.7495, suggesting it struggled somewhat with balancing precision and recall.

TEAM MEMBERS AND CONTRIBUTION

- VIDARSHANA GOVILESH: DATA PREPROCESSING, DEEP LEARNING MODEL IMPLEMENTATION,EVALUATION METRICS
- ASWIN GUNAEKARAN: DATA PREPROCESSING ,2 DEEP LEARNING, MODEL IMPLEMENTATION,PREDICTION

Thank you!