Assignment 2: Deep Convolutional Neural Network - Cats vs Dogs

Name : Akhila Kalpuri
Date : 03-23-2024

Loading all the required libraries and functions

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import requests
import seaborn as sns
import zipfile
import io
import os
import shutil
import pathlib
%matplotlib inline

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.utils import image_dataset_from_directory
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from io import BytesIO
from zipfile import ZipFile
```

Importing the JSON Activation Code

```
from google.colab import files
files.upload()
```

<IPython.core.display.HTML object>

Saving kaggle.json to kaggle (2).json

{'kaggle (2).json': b'{"username":"vini09","key":"366b6dda6f517e24aec2460856a9b32a"}'

```
!mkdir ~/.kaggle

!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json
```

mkdir: cannot create directory ‘/root/.kaggle’: File exists

Downloading the Data

```
!pip install tensorflow==2.12
```

```
!Kaggle competitions download -c dogs-vs-cats
```

dogs-vs-cats.zip: Skipping, found more recently modified local copy (use --forceto force download)

Unzipping the Data

```
!unzip -o -qq dogs-vs-cats.zip
```

Unzipping Train Data

```
!unzip -o -qq train.zip
```

Requirement already satisfied: tensorflow==2.12 in /usr/local/lib/python3.10/dist-packages (2.12.0)

Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist- packages (from tensorflow==2.12) (1.4.0)

Requirement already satisfied: astunparse>=1.6.0 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (1.6.3) Requirement already satisfied: flatbuffers>=2.0 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (23.5.26) Requirement already satisfied: gast<=0.4.0,>=0.2.1 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (0.4.0) Requirement already satisfied: google-pasta>=0.1.1 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (0.2.0) Requirement already satisfied: grpcio<2.0,>=1.24.3 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (1.59.0) Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist- packages (from tensorflow==2.12) (3.9.0)

Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.10/dist- packages (from tensorflow==2.12) (0.4.16)

Requirement already satisfied: keras<2.13,>=2.12.0 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (2.12.0) Requirement already satisfied: libclang>=13.0.0 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (16.0.6) Requirement already satisfied: numpy<1.24,>=1.22 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (1.23.5)

Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (3.3.0)

Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist- packages (from tensorflow==2.12) (23.2)

Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3

in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (3.20.3) Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist- packages (from tensorflow==2.12) (67.7.2)

Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist- packages (from tensorflow==2.12) (1.16.0)

Requirement already satisfied: tensorboard<2.13,>=2.12 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (2.12.3) Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (2.12.0) Requirement already satisfied: termcolor>=1.1.0 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (2.3.0) Requirement already satisfied: typing-extensions>=3.6.6 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (4.5.0) Requirement already satisfied: wrapt<1.15,>=1.11.0 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (1.14.1) Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in

/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (0.34.0) Requirement already satisfied: wheel<1.0,>=0.23.0 in

/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow==2.12) (0.41.2) Requirement already satisfied: ml-dtypes>=0.2.0 in

/usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow==2.12) (0.3.1)

Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.10/dist- packages (from jax>=0.3.15->tensorflow==2.12) (1.11.3)

Requirement already satisfied: google-auth<3,>=1.6.3 in

/usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12) (2.17.3)

Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in

/usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12) (1.0.0) Requirement already satisfied: markdown>=2.6.8 in

/usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12) (3.5) Requirement already satisfied: requests<3,>=2.21.0 in

/usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12) (2.31.0)

Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in

/usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12) (0.7.1) Requirement already satisfied: werkzeug>=1.0.1 in

/usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12) (3.0.0) Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google- auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12) (5.3.1) Requirement already satisfied: pyasn1-modules>=0.2.1 in

/usr/local/lib/python3.10/dist-packages (from google- auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12) (0.3.0)

Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist- packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12) (4.9)

Requirement already satisfied: requests-oauthlib>=0.7.0 in

/usr/local/lib/python3.10/dist-packages (from google-auth- oauthlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow==2.12) (1.3.1) Requirement already satisfied: charset-normalizer<4,>=2 in

/usr/local/lib/python3.10/dist-packages (from

requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12) (3.3.0) Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist- packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12) (3.4)

Requirement already satisfied: urllib3<3,>=1.21.1 in

/usr/local/lib/python3.10/dist-packages (from

requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12) (2.0.7) Requirement already satisfied: certifi>=2017.4.17 in

/usr/local/lib/python3.10/dist-packages (from

requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12) (2023.7.22) Requirement already satisfied: MarkupSafe>=2.1.1 in

/usr/local/lib/python3.10/dist-packages (from

werkzeug>=1.0.1->tensorboard<2.13,>=2.12->tensorflow==2.12) (2.1.3) Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in

/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google- auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12) (0.5.0) Requirement already satisfied: oauthlib>=3.0.0 in

/usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google- auth-oauthlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow==2.12) (3.2.2)

Consider the Cats & Dogs example. Start initially with a training sample of 1000, a validation sample of 500, and a test sample of 500 (like in the text). Use any technique to reduce overfitting and improve performance in developing a network that you train from scratch. What performance did you achieve?

Creating directory named cats vs dogs small to store the images into 3 subsets named train, validation and test and Dividing the training sample of 1000, a validation sample of 500, and a test sample of 500

```
import os, shutil, pathlib
shutil.rmtree("cats_vs_dogs_small/train/cat")

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")
```

```
def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir, exist_ok = True)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames: shutil.copyfile(src=original_dir /
            fname,
                                dst=dir / fname)

make_subset("train", start_index=0, end_index=500)
make_subset("validation", start_index=1000, end_index=1250)
make_subset("test", start_index=1500, end_index=1750)
```

Data Pre-Processing and functions - Using image_dataset_from_directory to read images and functions

```
train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

Found 1000 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Viewing the shape of the images

```
for data_batch, labels_batch in train_dataset:
    print("data batch shape:", data_batch.shape)
    print("labels batch shape:", labels_batch.shape)
    break
```

data batch shape: (32, 180, 180, 3)
labels batch shape: (32,)
Model - 1 MaxPooling Operation with Increase in filters from 32 to 256 in 5 Input Layers : Instantiating a small convnet for dogs vs. cats classification

```
#Building the model
#Instantiating a small convnet for dogs vs. cats classification
```

```
#*Model - 1 MaxPooling Operation with Increase in filters from 32 to 256 in 5 Input Layers*

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)


model = keras.Model(inputs=inputs, outputs=outputs)
```

Summary of Model - 1

```
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_5 (Conv2D)           (None, 178, 178, 32)      896

 max_pooling2d_4 (MaxPooling  (None, 89, 89, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_7 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_6 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_8 (Conv2D)           (None, 18, 18, 256)       295168
```

max_pooling2d_7 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_9 (Conv2D)	(None, 7, 7, 256)	590080
flatten_1 (Flatten)	(None, 12544)	0
dense_1 (Dense)	(None, 1)	12545

```
# Compilimg the results of the model
model.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])


# Saving the results of the model
callbacks = ModelCheckpoint(
             filepath= "model1.keras",
             save_best_only= True,
             monitor= "val_loss"
             )


# Fitting/Running the Model
Model_1 = model.fit(
           train_dataset,
           epochs= 30,
           validation_data= validation_dataset,
           callbacks= callbacks)
```

```
Epoch 1/30
32/32 [==============================] - 3s 30ms/step - loss: 0.6899 - accuracy:
0.5390 - val_loss: 0.6743 - val_accuracy: 0.5820 Epoch 2/30
32/32 [==============================] - 1s 24ms/step - loss: 0.6543 - accuracy:
0.6320 - val_loss: 0.6380 - val_accuracy: 0.6420 Epoch 3/30
32/32 [==============================] - 1s 23ms/step - loss: 0.6406 - accuracy:
0.6390 - val_loss: 0.6706 - val_accuracy: 0.5860

Epoch 4/30
32/32 [==============================] - 1s 26ms/step - loss: 0.6141 - accuracy:
0.6760 - val_loss: 0.6253 - val_accuracy: 0.6500 Epoch 5/30
32/32 [==============================] - 1s 26ms/step - loss: 0.5596 - accuracy:
0.7320 - val_loss: 0.6162 - val_accuracy: 0.6520 Epoch 6/30
```

32/32 [==========] - 1s 26ms/step - loss: 0.5109 - accuracy: 0.7480 - val_loss: 0.5976 - val_accuracy: 0.6900 Epoch 7/30
32/32 [==========] - 1s 26ms/step - loss: 0.4869 - accuracy: 0.7690 - val_loss: 0.6162 - val_accuracy: 0.6440 Epoch 8/30
32/32 [==========] - 1s 27ms/step - loss: 0.4562 - accuracy: 0.7710 - val_loss: 0.5936 - val_accuracy: 0.7020 Epoch 9/30
32/32 [==========] - 1s 26ms/step - loss: 0.3715 - accuracy: 0.8410 - val_loss: 0.5821 - val_accuracy: 0.7120 Epoch 10/30
32/32 [==========] - 1s 27ms/step - loss: 0.3177 - accuracy: 0.8630 - val_loss: 0.6811 - val_accuracy: 0.6940 Epoch 11/30
32/32 [==========] - 1s 24ms/step - loss: 0.2177 - accuracy: 0.9170 - val_loss: 0.8881 - val_accuracy: 0.6960 Epoch 12/30
32/32 [==========] - 1s 24ms/step - loss: 0.1723 - accuracy: 0.9300 - val_loss: 0.7746 - val_accuracy: 0.7200 Epoch 13/30
32/32 [==========] - 1s 24ms/step - loss: 0.1326 - accuracy: 0.9420 - val_loss: 0.9028 - val_accuracy: 0.7280 Epoch 14/30
32/32 [==========] - 1s 24ms/step - loss: 0.1020 - accuracy: 0.9670 - val_loss: 0.8975 - val_accuracy: 0.7360 Epoch 15/30
32/32 [==========] - 1s 23ms/step - loss: 0.0588 - accuracy: 0.9780 - val_loss: 1.2818 - val_accuracy: 0.7080 Epoch 16/30
32/32 [==========] - 1s 23ms/step - loss: 0.0889 - accuracy: 0.9730 - val_loss: 1.1592 - val_accuracy: 0.6920 Epoch 17/30
32/32 [==========] - 1s 23ms/step - loss: 0.0810 - accuracy: 0.9690 - val_loss: 1.0890 - val_accuracy: 0.7240 Epoch 18/30
32/32 [==========] - 1s 23ms/step - loss: 0.0419 - accuracy: 0.9890 - val_loss: 1.1962 - val_accuracy: 0.7180 Epoch 19/30
32/32 [==========] - 1s 23ms/step - loss: 0.0448 - accuracy: 0.9850 - val_loss: 1.5015 - val_accuracy: 0.6820

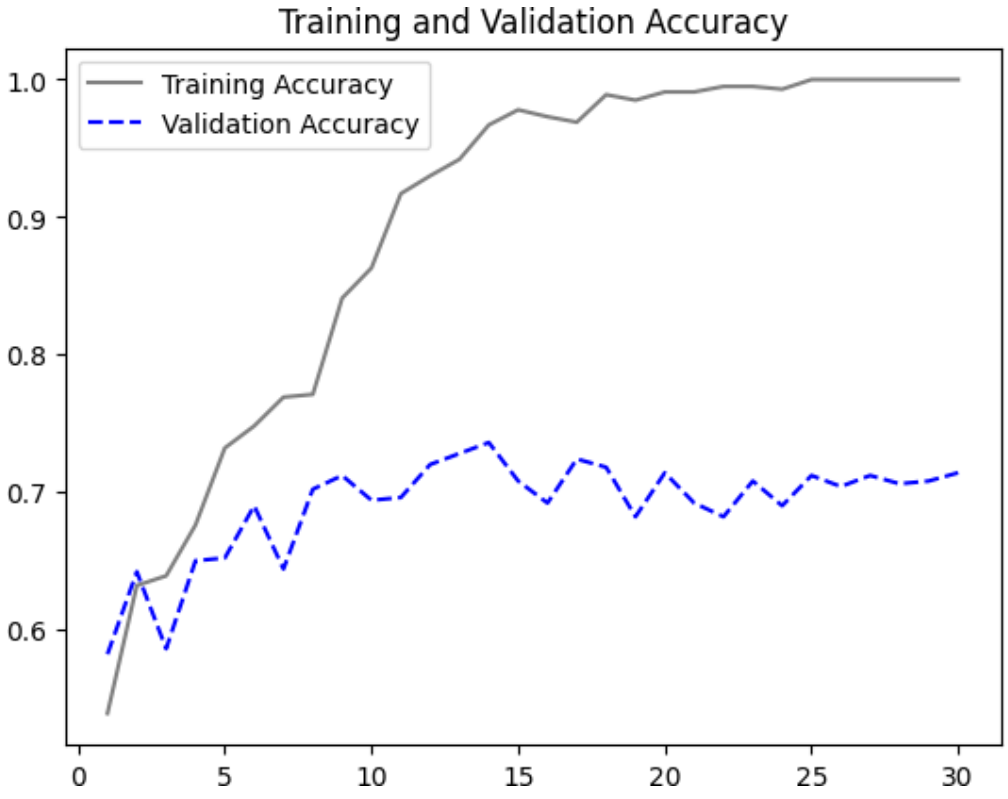
Epoch 20/30
32/32 [==========] - 1s 23ms/step - loss: 0.0243 - accuracy: 0.9910 - val_loss: 1.8326 - val_accuracy: 0.7140 Epoch 21/30
32/32 [==========] - 1s 24ms/step - loss: 0.0288 - accuracy: 0.9910 - val_loss: 1.6035 - val_accuracy: 0.6920 Epoch 22/30
32/32 [==========] - 1s 24ms/step - loss: 0.0204 - accuracy: 0.9950 - val_loss: 1.8064 - val_accuracy: 0.6820 Epoch 23/30
32/32 [==========] - 1s 24ms/step - loss: 0.0298 - accuracy: 0.9950 - val_loss: 1.4813 - val_accuracy: 0.7080 Epoch 24/30

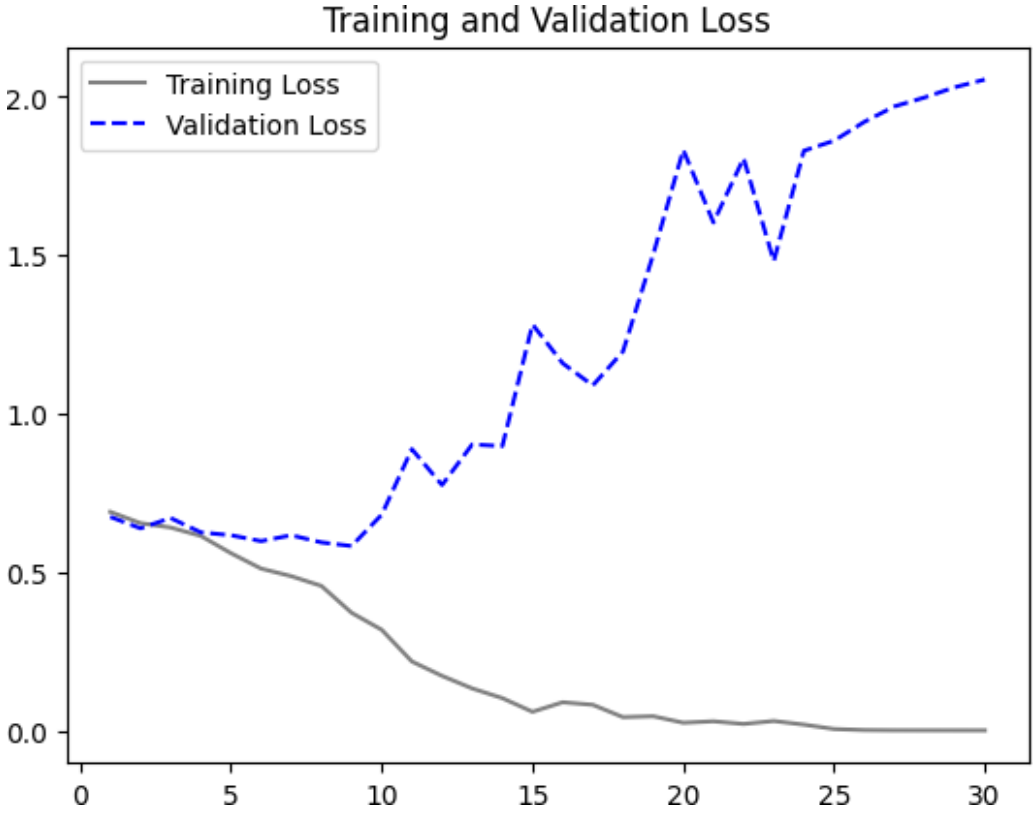
32/32 [==========] - 1s 22ms/step - loss: 0.0183 - accuracy: 0.9930 - val_loss: 1.8299 - val_accuracy: 0.6900 Epoch 25/30
32/32 [==========] - 1s 23ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 1.8617 - val_accuracy: 0.7120 Epoch 26/30
32/32 [==========] - 1s 23ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 1.9209 - val_accuracy: 0.7040 Epoch 27/30
32/32 [==========] - 1s 23ms/step - loss: 4.2510e-04 - accuracy: 1.0000 - val_loss: 1.9698 - val_accuracy: 0.7120 Epoch 28/30
32/32 [==========] - 1s 23ms/step - loss: 3.0441e-04 - accuracy: 1.0000 - val_loss: 1.9980 - val_accuracy: 0.7060 Epoch 29/30
32/32 [==========] - 1s 24ms/step - loss: 2.4816e-04 - accuracy: 1.0000 - val_loss: 2.0304 - val_accuracy: 0.7080 Epoch 30/30
32/32 [==========] - 1s 24ms/step - loss: 1.9589e-04 - accuracy: 1.0000 - val_loss: 2.0548 - val_accuracy: 0.7140

Looking at the visuals of the Training and Validation Accuracy/Loss

```
accuracy = Model_1.history["accuracy"] val_accuracy =
Model_1.history["val_accuracy"]

loss = Model_1.history["loss"] val_loss =
Model_1.history["val_loss"]

epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, color="grey", label="Training Accuracy")
plt.plot(epochs, val_accuracy, color="blue",linestyle="dashed",␣
↪label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
```

```
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
1.0
0.9
0.8
0.7
0.6
0
5
10
15
20
25
30


Training and Validation Loss
Training Loss
Validation Loss
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
30

Evaluating the performance of Model_1 on test set

```
test_model = keras.models.load_model("model1.keras")
Model1_Results = test_model.evaluate(test_dataset)
print(f"Loss: {Model1_Results[0]:.3f}")
print(f"Accuracy: {Model1_Results[1]:.3f}")
```

16/16 [==============================] - 0s 10ms/step - loss: 0.6533 - accuracy:
0.6960
Loss: 0.653
Accuracy: 0.696
Using Measures to Avoid OverfittingData
Augmentation

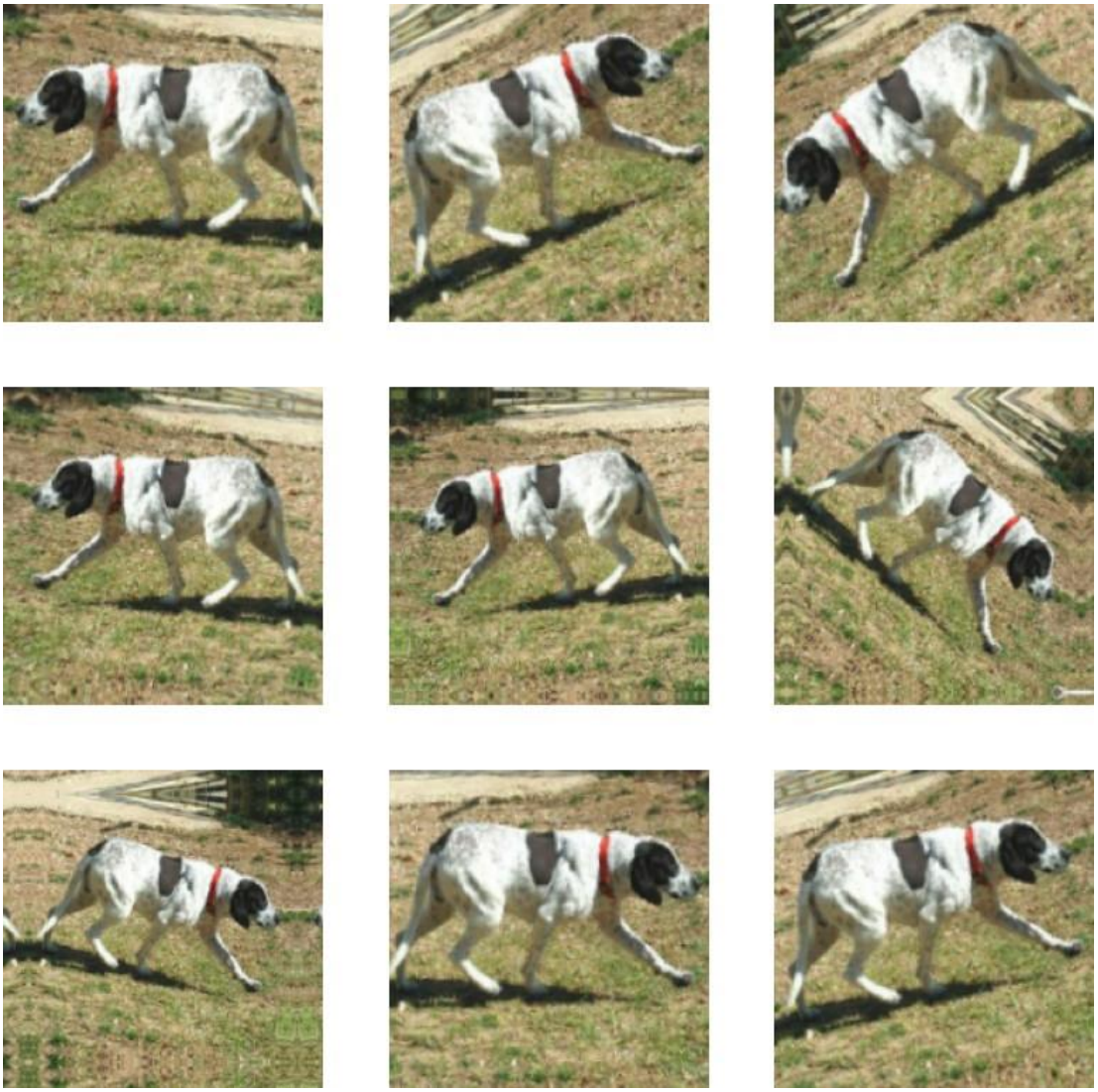
```
# To deprecate warnings that are making the output look clumsy
import logging
logging.getLogger('tensorflow').disabled = True
```

Using few of the techniques such as random flip, random zoom, random rotation so as to create augmented versions of the image

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2)
    ]
)
```

Looking at the augmented images

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```



Model - 2 MaxPooling Operation with Increase in filters from 32 to 256 in 5 Input Layers with thedata being used from the Augmented Images and a droput rate of 0.5*

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
```

```
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

Training the model 2

```
# Compiling the model
model.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])

# Saving the results of the model
callbacks = ModelCheckpoint(
              filepath= "model2.keras",
              save_best_only= True,
              monitor= "val_loss")

# Fitting/Running the Model
Model_2 = model.fit(
            train_dataset,
            epochs= 30,
            validation_data= validation_dataset,
            callbacks= callbacks)
```

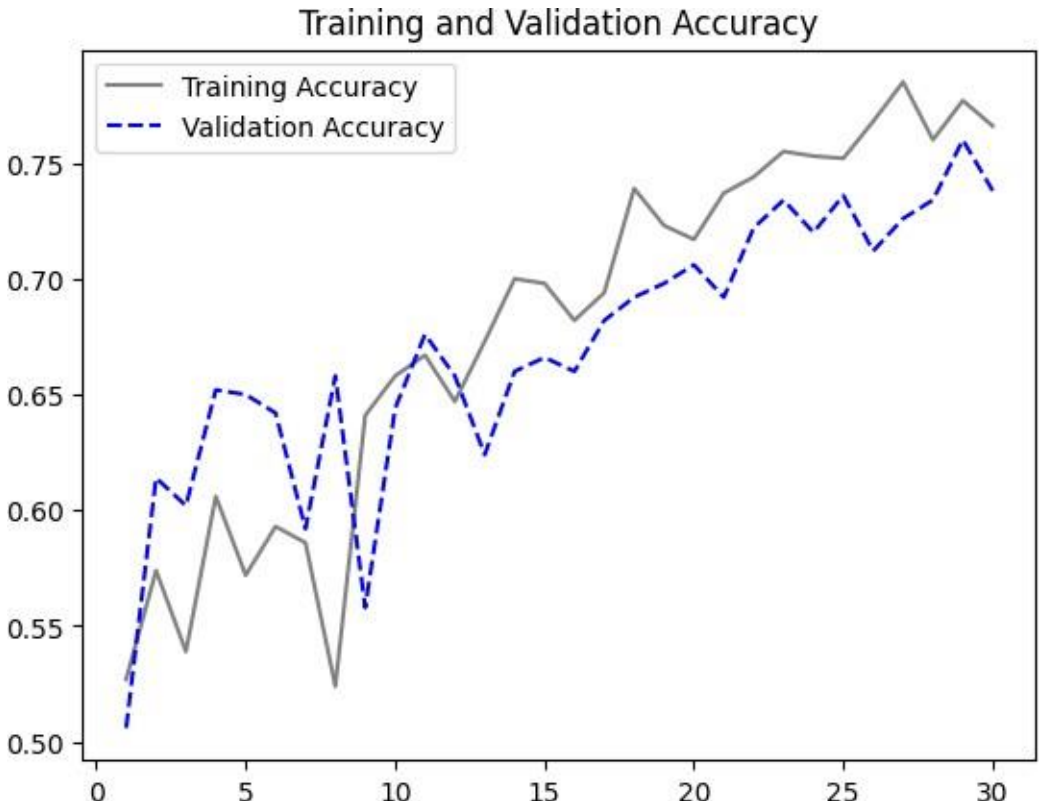
```
Epoch 1/30
32/32 [==============================] - 4s 32ms/step - loss: 0.6955 - accuracy:
0.5270 - val_loss: 0.6861 - val_accuracy: 0.5060 Epoch 2/30
32/32 [==============================] - 1s 27ms/step - loss: 0.6850 - accuracy:
0.5740 - val_loss: 0.6730 - val_accuracy: 0.6140 Epoch 3/30
32/32 [==============================] - 1s 27ms/step - loss: 0.6842 - accuracy:
0.5390 - val_loss: 0.6574 - val_accuracy: 0.6020 Epoch 4/30
32/32 [==============================] - 1s 24ms/step - loss: 0.6685 - accuracy:
0.6060 - val_loss: 0.6798 - val_accuracy: 0.6520 Epoch 5/30
32/32 [==============================] - 1s 26ms/step - loss: 0.6824 - accuracy:
0.5720 - val_loss: 0.6402 - val_accuracy: 0.6500 Epoch 6/30
32/32 [==============================] - 1s 26ms/step - loss: 0.6649 - accuracy:
0.5930 - val_loss: 0.6360 - val_accuracy: 0.6420 Epoch 7/30
32/32 [==============================] - 1s 25ms/step - loss: 0.6803 - accuracy:
```

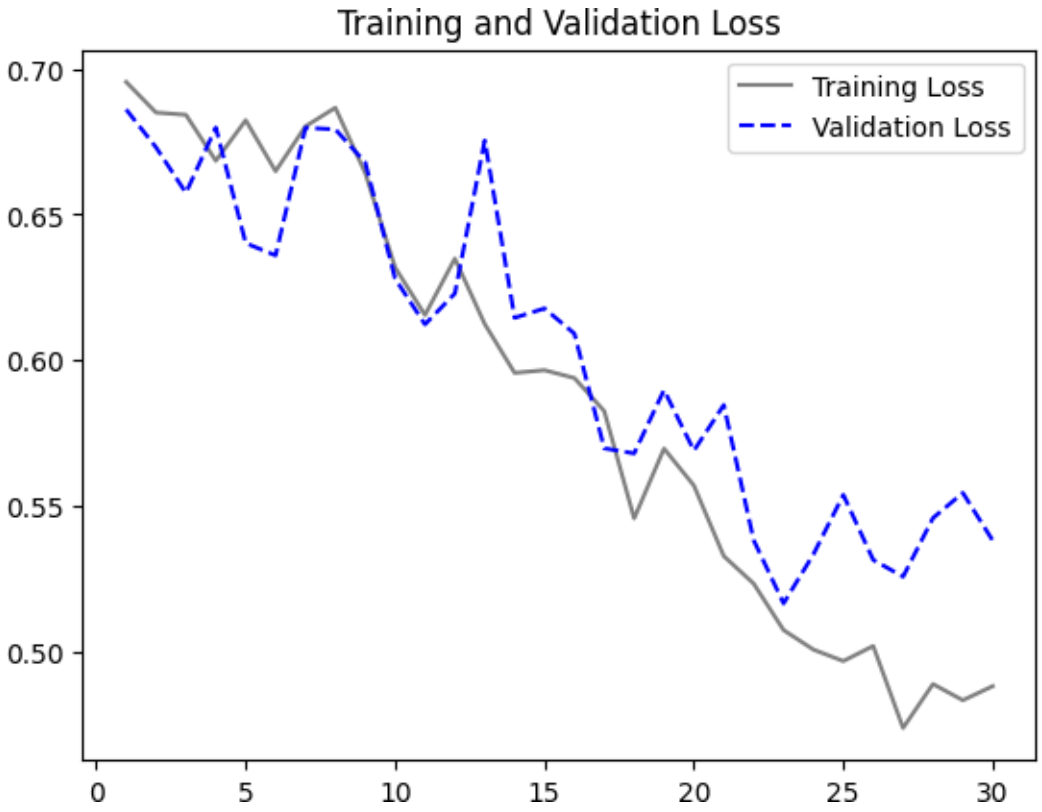
0.5860 - val_loss: 0.6798 - val_accuracy: 0.5920 Epoch 8/30
32/32 [==============================] - 1s 24ms/step - loss: 0.6867 - accuracy: 0.5240 - val_loss: 0.6792 - val_accuracy: 0.6580 Epoch 9/30
32/32 [==============================] - 1s 24ms/step - loss: 0.6643 - accuracy: 0.6410 - val_loss: 0.6679 - val_accuracy: 0.5580 Epoch 10/30
32/32 [==============================] - 1s 26ms/step - loss: 0.6318 - accuracy: 0.6580 - val_loss: 0.6280 - val_accuracy: 0.6440 Epoch 11/30
32/32 [==============================] - 1s 26ms/step - loss: 0.6155 - accuracy: 0.6670 - val_loss: 0.6123 - val_accuracy: 0.6760 Epoch 12/30
32/32 [==============================] - 1s 25ms/step - loss: 0.6349 - accuracy: 0.6470 - val_loss: 0.6229 - val_accuracy: 0.6580 Epoch 13/30
32/32 [==============================] - 1s 25ms/step - loss: 0.6124 - accuracy: 0.6730 - val_loss: 0.6754 - val_accuracy: 0.6240 Epoch 14/30
32/32 [==============================] - 1s 25ms/step - loss: 0.5956 - accuracy: 0.7000 - val_loss: 0.6145 - val_accuracy: 0.6600 Epoch 15/30
32/32 [==============================] - 1s 26ms/step - loss: 0.5965 - accuracy: 0.6980 - val_loss: 0.6177 - val_accuracy: 0.6660 Epoch 16/30
32/32 [==============================] - 1s 28ms/step - loss: 0.5939 - accuracy: 0.6820 - val_loss: 0.6091 - val_accuracy: 0.6600 Epoch 17/30
32/32 [==============================] - 1s 27ms/step - loss: 0.5825 - accuracy: 0.6940 - val_loss: 0.5697 - val_accuracy: 0.6820 Epoch 18/30
32/32 [==============================] - 1s 25ms/step - loss: 0.5457 - accuracy: 0.7390 - val_loss: 0.5680 - val_accuracy: 0.6920 Epoch 19/30
32/32 [==============================] - 1s 24ms/step - loss: 0.5696 - accuracy: 0.7230 - val_loss: 0.5897 - val_accuracy: 0.6980 Epoch 20/30
32/32 [==============================] - 1s 24ms/step - loss: 0.5569 - accuracy: 0.7170 - val_loss: 0.5689 - val_accuracy: 0.7060 Epoch 21/30
32/32 [==============================] - 1s 25ms/step - loss: 0.5327 - accuracy: 0.7370 - val_loss: 0.5846 - val_accuracy: 0.6920 Epoch 22/30
32/32 [==============================] - 1s 26ms/step - loss: 0.5233 - accuracy: 0.7440 - val_loss: 0.5382 - val_accuracy: 0.7220 Epoch 23/30
32/32 [==============================] - 1s 26ms/step - loss: 0.5073 - accuracy: 0.7550 - val_loss: 0.5165 - val_accuracy: 0.7340 Epoch 24/30
32/32 [==============================] - 1s 25ms/step - loss: 0.5006 - accuracy: 0.7530 - val_loss: 0.5333 - val_accuracy: 0.7200 Epoch 25/30
32/32 [==============================] - 1s 24ms/step - loss: 0.4967 - accuracy: 0.7520 - val_loss: 0.5537 - val_accuracy: 0.7360 Epoch 26/30
32/32 [==============================] - 1s 24ms/step - loss: 0.5018 - accuracy: 0.7680 - val_loss: 0.5314 - val_accuracy: 0.7120 Epoch 27/30

32/32 [==========] - 1s 24ms/step - loss: 0.4737 - accuracy: 0.7850 - val_loss: 0.5256 - val_accuracy: 0.7260 Epoch 28/30
32/32 [==========] - 1s 24ms/step - loss: 0.4888 - accuracy: 0.7600 - val_loss: 0.5458 - val_accuracy: 0.7340 Epoch 29/30
32/32 [==========] - 1s 25ms/step - loss: 0.4832 - accuracy: 0.7770 - val_loss: 0.5544 - val_accuracy: 0.7600 Epoch 30/30
32/32 [==========] - 1s 26ms/step - loss: 0.4880 - accuracy: 0.7660 - val_loss: 0.5380 - val_accuracy: 0.7380

Visualizing the Training and Validation Accuracy/Loss 2

```
accuracy = Model_2.history["accuracy"]
val_accuracy = Model_2.history["val_accuracy"]

loss = Model_2.history["loss"]
val_loss = Model_2.history["val_loss"]

epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, color="grey", label="Training Accuracy")
plt.plot(epochs, val_accuracy, color="blue",linestyle="dashed",
 ↪label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
 ↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```





Evaluating the performance of Model_2 on the test set

```
test_model = keras.models.load_model("model2.keras")
Model2_Results = test_model.evaluate(test_dataset)
print(f"Loss: {Model2_Results[0]:.3f}")
print(f"Accuracy: {Model2_Results[1]:.3f}")
```

16/16 [==============================] - 0s 9ms/step - loss: 0.5880 - accuracy:
0.7140

Loss: 0.588

Accuracy: 0.714

Comparing the Model 1 and Model 2 : we can clearly see that the accuracy rate of model 2 is higher than model 1

Model - 3 MaxPooling Operation with Increase in filters from 32 to 512 in 6 Input Layers with theuse of Augmented Images and Dropout rate of 0.5

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
```

```
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

Model: "model_4"

Layer (type) Output Shape Param #

==========

input_5 (InputLayer)[(None, 180, 180, 3)] 0

```
sequential_1 (Sequential)   (None,180,   180,   3)      0
rescaling_4 (Rescaling)     (None,180,   180,   3)      0
conv2d_21 (Conv2D)          (None,178,   178,   32)     896
max_pooling2d_17 (MaxPoolin    (None, 89, 89, 32)   0
g2D)


conv2d_22 (Conv2D)          (None, 87, 87, 64)   18496
max_pooling2d_18 (MaxPoolin g2D)      (None, 43, 43, 64)   0
conv2d_23 (Conv2D)          (None, 41, 41, 128) 73856
max_pooling2d_19 (MaxPoolin g2D)      (None, 20, 20, 128) 0
conv2d_24 (Conv2D)          (None, 18, 18, 256) 295168
max_pooling2d_20 (MaxPoolin g2D)      (None, 9, 9, 256)    0


conv2d_25 (Conv2D)          (None, 7, 7, 256)    590080
max_pooling2d_21 (MaxPoolin    (None, 3, 3, 256)    0
g2D)

conv2d_26   (Conv2D)     (None,1, 1,   512)   1180160
flatten_4     (Flatten)     (None,512)          0
dropout_2     (Dropout)     (None,512)          0
dense_4 (Dense)     (None, 1)      513
=================================================================
Total params: 2,159,169
Trainable params: 2,159,169
Non-trainable params: 0
```

Training the model 3

```
# Compiling the Model
model.compile(loss= "binary_crossentropy",
              optimizer= "adam",
              metrics= ["accuracy"])


# Monitoring the best validation loss using Callbacks
callbacks = ModelCheckpoint(
              filepath= "model3.keras",
              save_best_only= True,
              monitor= "val_loss")


# Model Fit
```

Epoch 1/30
32/32 [==============================] - 4s 36ms/step - loss: 0.6964 - accuracy: 0.4870 - val_loss: 0.6907 - val_accuracy: 0.5000 Epoch 2/30
32/32 [==============================] - 1s 26ms/step - loss: 0.6923 - accuracy: 0.5110 - val_loss: 0.6923 - val_accuracy: 0.5040

Epoch 3/30
32/32 [==============================] - 1s 27ms/step - loss: 0.6922 - accuracy: 0.5110 - val_loss: 0.6945 - val_accuracy: 0.5000 Epoch 4/30
32/32 [==============================] - 1s 28ms/step - loss: 0.6863 - accuracy: 0.5780 - val_loss: 0.6790 - val_accuracy: 0.5180 Epoch 5/30
32/32 [==============================] - 1s 28ms/step - loss: 0.6562 - accuracy: 0.6360 - val_loss: 0.6386 - val_accuracy: 0.6460 Epoch 6/30
32/32 [==============================] - 1s 25ms/step - loss: 0.6602 - accuracy: 0.6080 - val_loss: 0.6814 - val_accuracy: 0.5180 Epoch 7/30
32/32 [==============================] - 1s 25ms/step - loss: 0.6547 - accuracy: 0.6260 - val_loss: 0.6712 - val_accuracy: 0.5180 Epoch 8/30
32/32 [==============================] - 1s 26ms/step - loss: 0.6442 - accuracy: 0.6410 - val_loss: 0.6418 - val_accuracy: 0.6200 Epoch 9/30
32/32 [==============================] - 1s 28ms/step - loss: 0.6379 - accuracy: 0.6170 - val_loss: 0.6172 - val_accuracy: 0.6760 Epoch 10/30
32/32 [==============================] - 1s 25ms/step - loss: 0.6240 - accuracy: 0.6730 - val_loss: 0.6410 - val_accuracy: 0.6680 Epoch 11/30
32/32 [==============================] - 1s 26ms/step - loss: 0.6504 - accuracy: 0.6660 - val_loss: 0.6282 - val_accuracy: 0.6520 Epoch 12/30
32/32 [==============================] - 1s 25ms/step - loss: 0.6274 - accuracy: 0.6520 - val_loss: 0.6781 - val_accuracy: 0.5360 Epoch 13/30
32/32 [==============================] - 1s 28ms/step - loss: 0.6202 - accuracy: 0.6680 - val_loss: 0.6041 - val_accuracy: 0.6960 Epoch 14/30
32/32 [==============================] - 1s 26ms/step - loss: 0.6165 - accuracy:

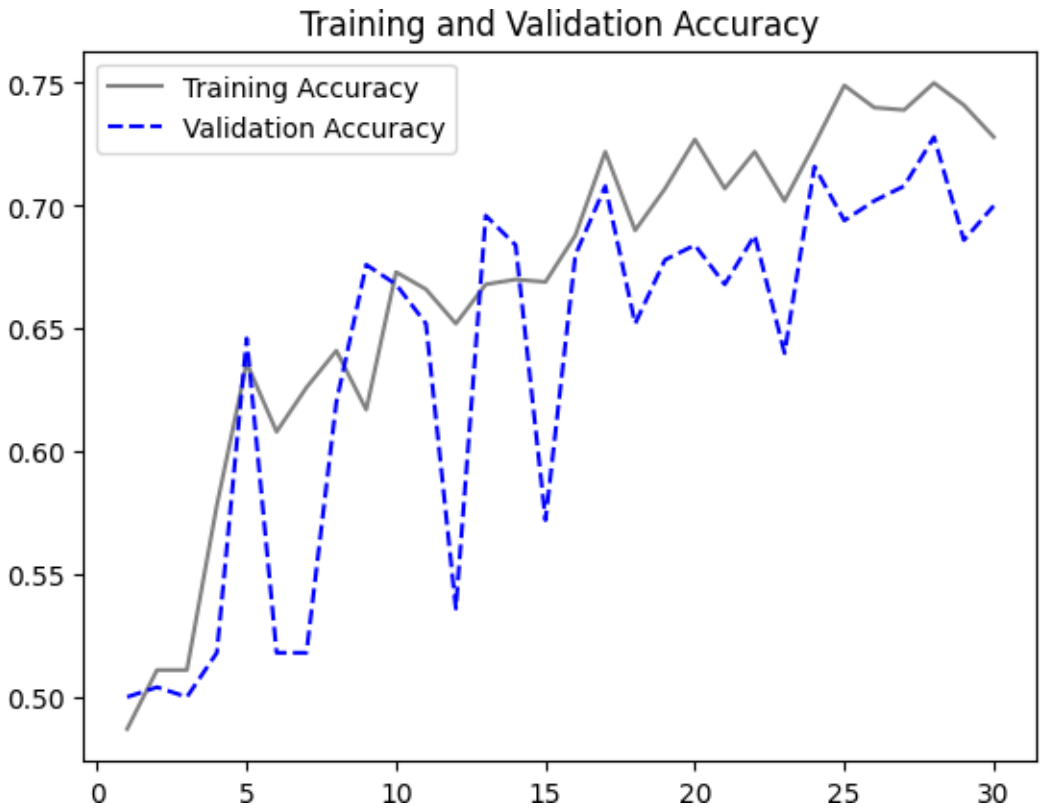
0.6700 - val_loss: 0.6110 - val_accuracy: 0.6840 Epoch 15/30
32/32 [==============================] - 1s 27ms/step - loss: 0.6128 - accuracy: 0.6690 - val_loss: 0.6662 - val_accuracy: 0.5720 Epoch 16/30
32/32 [==============================] - 1s 26ms/step - loss: 0.6125 - accuracy: 0.6880 - val_loss: 0.6049 - val_accuracy: 0.6800 Epoch 17/30
32/32 [==============================] - 1s 29ms/step - loss: 0.5724 - accuracy: 0.7220 - val_loss: 0.5846 - val_accuracy: 0.7080 Epoch 18/30
32/32 [==============================] - 1s 26ms/step - loss: 0.5896 - accuracy: 0.6900 - val_loss: 0.6149 - val_accuracy: 0.6520

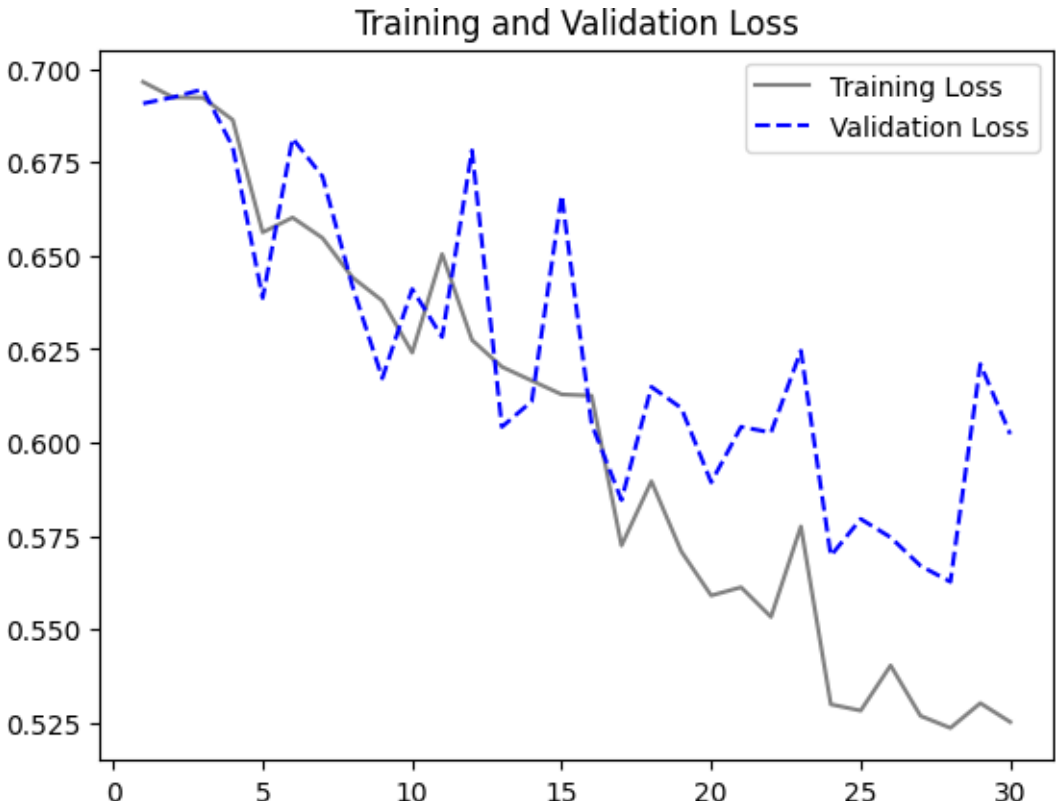
Epoch 19/30
32/32 [==============================] - 1s 27ms/step - loss: 0.5708 - accuracy: 0.7070 - val_loss: 0.6092 - val_accuracy: 0.6780 Epoch 20/30
32/32 [==============================] - 1s 26ms/step - loss: 0.5590 - accuracy: 0.7270 - val_loss: 0.5893 - val_accuracy: 0.6840 Epoch 21/30
32/32 [==============================] - 1s 26ms/step - loss: 0.5612 - accuracy: 0.7070 - val_loss: 0.6042 - val_accuracy: 0.6680 Epoch 22/30
32/32 [==============================] - 1s 25ms/step - loss: 0.5533 - accuracy: 0.7220 - val_loss: 0.6027 - val_accuracy: 0.6880 Epoch 23/30
32/32 [==============================] - 1s 25ms/step - loss: 0.5775 - accuracy: 0.7020 - val_loss: 0.6245 - val_accuracy: 0.6400 Epoch 24/30
32/32 [==============================] - 1s 28ms/step - loss: 0.5299 - accuracy: 0.7250 - val_loss: 0.5697 - val_accuracy: 0.7160 Epoch 25/30
32/32 [==============================] - 1s 25ms/step - loss: 0.5282 - accuracy: 0.7490 - val_loss: 0.5795 - val_accuracy: 0.6940 Epoch 26/30
32/32 [==============================] - 1s 25ms/step - loss: 0.5403 - accuracy: 0.7400 - val_loss: 0.5746 - val_accuracy: 0.7020 Epoch 27/30
32/32 [==============================] - 1s 28ms/step - loss: 0.5268 - accuracy: 0.7390 - val_loss: 0.5670 - val_accuracy: 0.7080 Epoch 28/30
32/32 [==============================] - 1s 28ms/step - loss: 0.5236 - accuracy: 0.7500 - val_loss: 0.5627 - val_accuracy: 0.7280 Epoch 29/30
32/32 [==============================] - 1s 26ms/step - loss: 0.5302 - accuracy: 0.7410 - val_loss: 0.6210 - val_accuracy: 0.6860 Epoch 30/30
32/32 [==============================] - 1s 26ms/step - loss: 0.5252 - accuracy: 0.7280 - val_loss: 0.6022 - val_accuracy: 0.7000

Visualizing the Training and Validation Accuracy/Loss 2

```
accuracy = Model_3.history["accuracy"]
val_accuracy = Model_3.history["val_accuracy"]

loss = Model_3.history["loss"]
val_loss = Model_3.history["val_loss"]

epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, color="grey", label="Training Accuracy")
```

```
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed",␣
↪label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```





Evaluating the performance of Model_2 on the test set

```
best_model = keras.models.load_model("model3.keras")
Model3_Results = best_model.evaluate(test_dataset)
print(f"Loss: {Model3_Results[0]:.3f}")
print(f"Accuracy: {Model3_Results[1]:.3f}")
```

16/16 [==============================] - 0s 10ms/step - loss: 0.6202 - accuracy: 0.6740

Loss: 0.620

Accuracy: 0.674

Model - 4 MaxPooling Operation with Increase in filters from 64 to 1024 in 5 Input Layers withthe use of Augmented Images and Dropout rate of 0.6

```
inputs = keras.Input(shape=(180,180,3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
```

```
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=1024, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.6)(x)

outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

```
Model: "model_5"

Layer (type)   Output Shape         Param #
=================================================================
input_6 (InputLayer)[(None, 180, 180, 3)]        0

sequential_1 (Sequential)   (None,180,   180,   3)      0
rescaling_5 (Rescaling)      (None,180,   180,   3)      0
conv2d_27 (Conv2D)           (None,178,   178,   64)     1792
max_pooling2d_22 (MaxPoolin    (None, 89, 89, 64)   0
g2D)


conv2d_28 (Conv2D)           (None, 87, 87, 128) 73856
max_pooling2d_23 (MaxPoolin g2D)      (None, 43, 43, 128) 0
conv2d_29 (Conv2D)           (None, 41, 41, 256) 295168
max_pooling2d_24 (MaxPoolin g2D)      (None, 20, 20, 256) 0
conv2d_30 (Conv2D)           (None, 18, 18, 512) 1180160
max_pooling2d_25 (MaxPoolin g2D)      (None, 9, 9, 512)     0
conv2d_31 (Conv2D)           (None, 7, 7, 1024)   4719616
flatten_5 (Flatten)    (None, 50176)        0


dropout_3 (Dropout)          (None, 50176)        0
dense_5 (Dense)      (None, 1)       50177
```

```
=================================================================
```
Total params: 6,320,769

Trainable params: 6,320,769

Non-trainable params: 0

```
# Compiling the Model
model.compile(loss= "binary_crossentropy",
              optimizer= "adam",
              metrics= ['accuracy'])


# Monitoring the best validation loss using Callbacks
callbacks = ModelCheckpoint(
              filepath = "model4.keras",
              save_best_only= True,
              monitor= "val_loss"
              )

# Model Fit
Model_4 = model.fit(
            train_dataset,
            epochs= 30,
            validation_data= validation_dataset,
            callbacks= callbacks
            )
```

```
Epoch 1/30
32/32 [==============================] - 6s 66ms/step - loss: 0.7233 - accuracy:
0.5100 - val_loss: 0.6917 - val_accuracy: 0.5380Epoch
2/30
32/32 [==============================] - 1s 44ms/step - loss: 0.6933 - accuracy:
0.4990 - val_loss: 0.6862 - val_accuracy: 0.5420Epoch
3/30
32/32 [==============================] - 1s 39ms/step - loss: 0.6897 - accuracy:
0.5320 - val_loss: 0.6877 - val_accuracy: 0.5800Epoch
4/30
32/32 [==============================] - 1s 44ms/step - loss: 0.6934 - accuracy:
0.5300 - val_loss: 0.6826 - val_accuracy: 0.5680Epoch
5/30
32/32 [==============================] - 1s 39ms/step - loss: 0.6887 - accuracy:
0.5280 - val_loss: 0.6877 - val_accuracy: 0.4960Epoch
```

32/32 [==============================] - 1s 39ms/step - loss: 0.6816 - accuracy: 0.5380 - val_loss: 0.7028 - val_accuracy: 0.5100Epoch 7/30
32/32 [==============================] - 1s 39ms/step - loss: 0.6902 - accuracy: 0.5290 - val_loss: 0.6838 - val_accuracy: 0.5280Epoch 8/30
32/32 [==============================] - 1s 39ms/step - loss: 0.6737 - accuracy: 0.5920 - val_loss: 0.6870 - val_accuracy: 0.5400Epoch 9/30
32/32 [==============================] - 2s 45ms/step - loss: 0.6619 - accuracy: 0.6110 - val_loss: 0.6615 - val_accuracy: 0.6180Epoch 10/30
32/32 [==============================] - 1s 40ms/step - loss: 0.6695 - accuracy: 0.6070 - val_loss: 0.6997 - val_accuracy: 0.5840Epoch 11/30
32/32 [==============================] - 1s 39ms/step - loss: 0.6685 - accuracy: 0.5850 - val_loss: 0.8176 - val_accuracy: 0.5620Epoch 12/30
32/32 [==============================] - 2s 46ms/step - loss: 0.6513 - accuracy: 0.6280 - val_loss: 0.6424 - val_accuracy: 0.6580Epoch 13/30
32/32 [==============================] - 1s 39ms/step - loss: 0.6372 - accuracy: 0.6430 - val_loss: 0.6508 - val_accuracy: 0.6300Epoch 14/30
32/32 [==============================] - 1s 39ms/step - loss: 0.6345 - accuracy: 0.6540 - val_loss: 0.6442 - val_accuracy: 0.6340Epoch 15/30
32/32 [==============================] - 1s 44ms/step - loss: 0.6228 - accuracy: 0.6810 - val_loss: 0.6168 - val_accuracy: 0.6680Epoch 16/30
32/32 [==============================] - 1s 38ms/step - loss: 0.6065 - accuracy: 0.6830 - val_loss: 0.6339 - val_accuracy: 0.6640Epoch 17/30
32/32 [==============================] - 1s 43ms/step - loss: 0.6077 - accuracy: 0.6790 - val_loss: 0.6096 - val_accuracy: 0.6480Epoch 18/30
32/32 [==============================] - 1s 44ms/step - loss: 0.6121 - accuracy: 0.6890 - val_loss: 0.6080 - val_accuracy: 0.6660Epoch 19/30

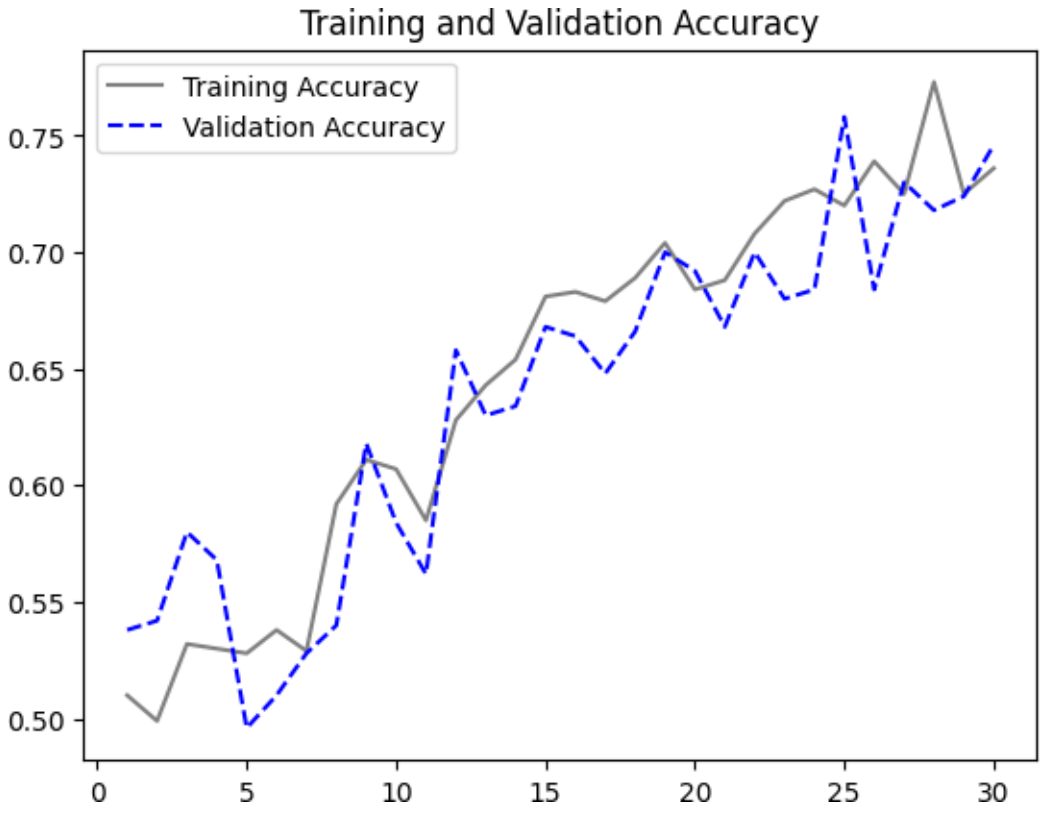
32/32 [==============================] - 2s 45ms/step - loss: 0.5897 - accuracy: 0.7040 - val_loss: 0.5931 - val_accuracy: 0.7000Epoch 20/30
32/32 [==============================] - 1s 40ms/step - loss: 0.5935 - accuracy: 0.6840 - val_loss: 0.6003 - val_accuracy: 0.6920Epoch 21/30
32/32 [==============================] - 1s 39ms/step - loss: 0.6061 - accuracy: 0.6880 - val_loss: 0.6036 - val_accuracy: 0.6680Epoch 22/30

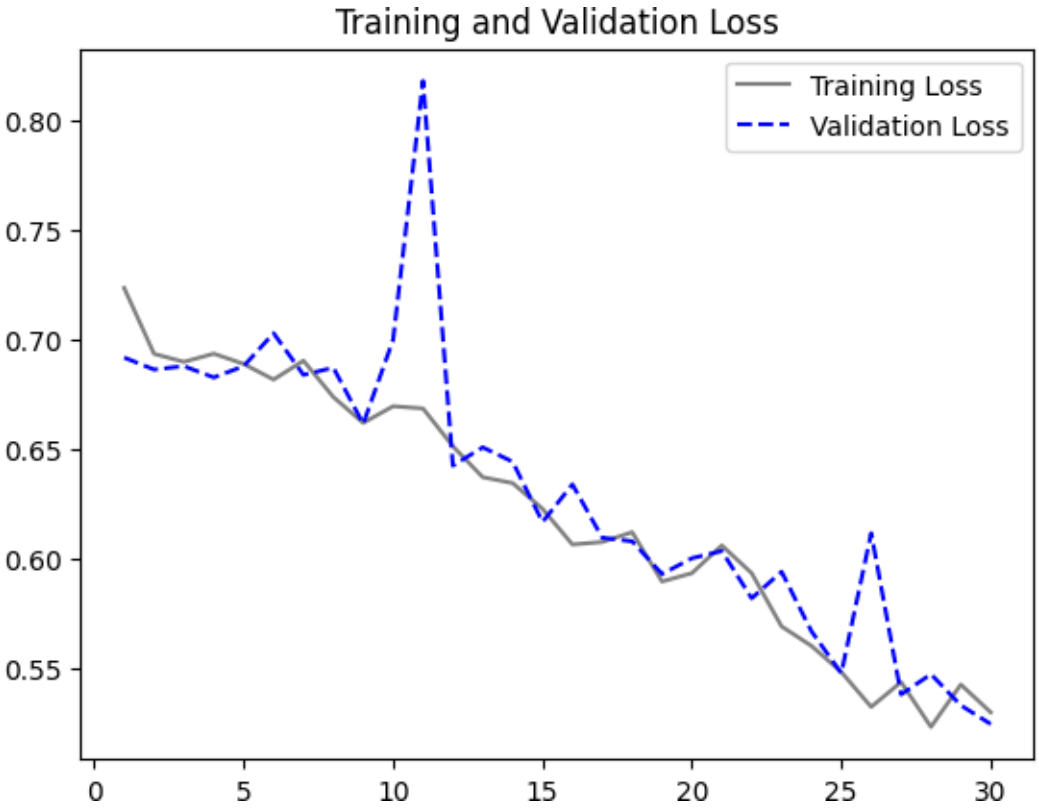
```
32/32 [==============================] - 1s 44ms/step - loss: 0.5935 - accuracy:
0.7080 - val_loss: 0.5820 - val_accuracy: 0.7000Epoch
23/30
32/32 [==============================] - 1s 39ms/step - loss: 0.5692 - accuracy:
0.7220 - val_loss: 0.5942 - val_accuracy: 0.6800Epoch
24/30
32/32 [==============================] - 1s 44ms/step - loss: 0.5604 - accuracy:
0.7270 - val_loss: 0.5670 - val_accuracy: 0.6840Epoch
25/30
32/32 [==============================] - 1s 44ms/step - loss: 0.5485 - accuracy:
0.7200 - val_loss: 0.5479 - val_accuracy: 0.7580Epoch
26/30
32/32 [==============================] - 1s 39ms/step - loss: 0.5326 - accuracy:
0.7390 - val_loss: 0.6117 - val_accuracy: 0.6840Epoch
27/30
32/32 [==============================] - 1s 44ms/step - loss: 0.5439 - accuracy:
0.7250 - val_loss: 0.5383 - val_accuracy: 0.7300Epoch
28/30
32/32 [==============================] - 1s 39ms/step - loss: 0.5235 - accuracy:
0.7730 - val_loss: 0.5475 - val_accuracy: 0.7180Epoch
29/30
32/32 [==============================] - 2s 45ms/step - loss: 0.5428 - accuracy:
0.7250 - val_loss: 0.5333 - val_accuracy: 0.7240Epoch
30/30
32/32 [==============================] - 2s 46ms/step - loss: 0.5301 - accuracy:
0.7360 - val_loss: 0.5248 - val_accuracy: 0.7460
```

Visualizing the Training and Validation Accuracy/Loss 2

```
accuracy = Model_4.history["accuracy"]
val_accuracy = Model_4.history["val_accuracy"]

loss = Model_4.history["loss"]
val_loss = Model_4.history["val_loss"]

epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, color="grey", label="Training Accuracy")
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed",␣
↪label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
```

```
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
0.75
0.70
0.65
0.60
0.55
0.50
0
5
10
15
20
25
30



Evaluating the performance of Model_2 on the test set

```
best_model = keras.models.load_model("model4.keras")
Model4_Results = best_model.evaluate(test_dataset)
print(f"Loss: {Model4_Results[0]:.3f}")
print(f"Accuracy: {Model4_Results[1]:.3f}")
```

16/16 [==============================] - 1s 14ms/step - loss: 0.6037 - accuracy: 0.6800
Loss: 0.604
Accuracy: 0.680

Summary for Question 1: We did attempt to construct four models using a thousand as the training sample. Let's now evaluate the accuracy and loss of all four models to determine which produces the best results.

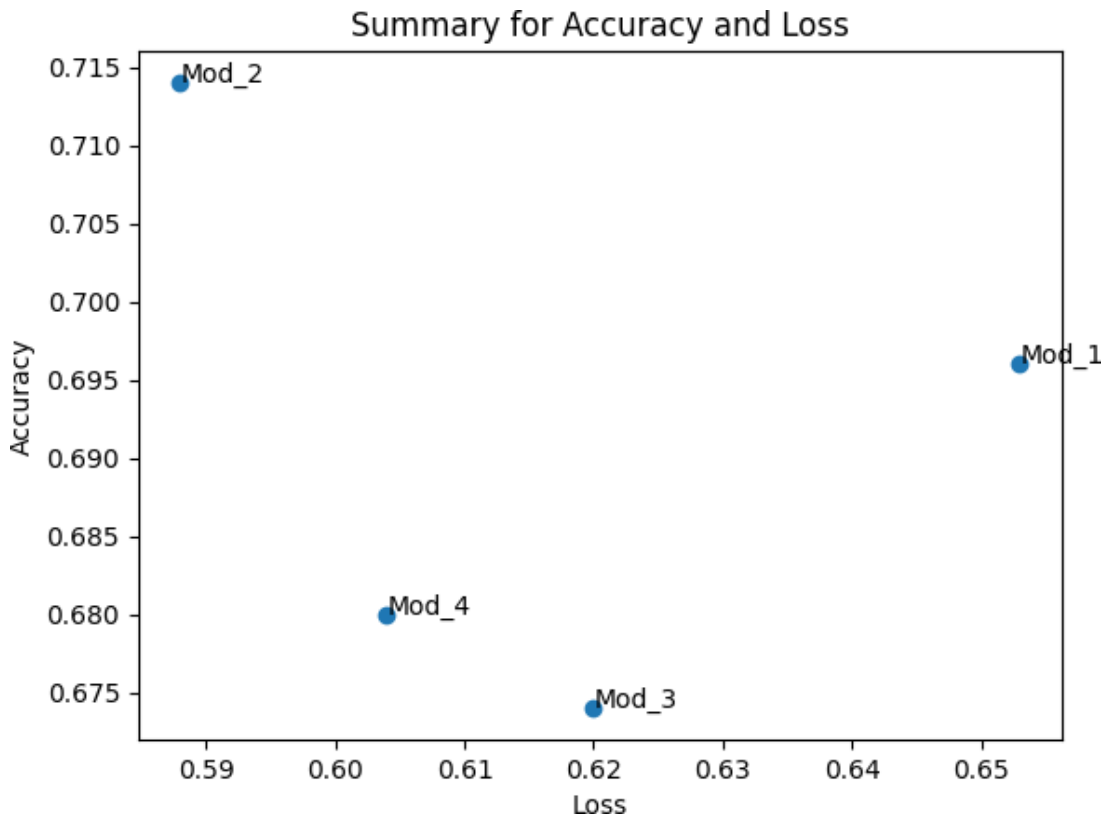
Model 1: filters from 32 to 256, 5 Input Layers
Model 2: filters from 32 to 256, 5 Input Layers, Augmented Images and Droput rate of 0.5 Model 3: filters from 32 to 512, 6 Input Layers, Augmented Images and Dropout rate of 0.5 Model 4: filters from 64 to 1024, 5 Input Layers,

Augmented Images and Dropout rate of 0.6

```
Model_1 = (0.653, 0.696)
Model_2 = (0.588, 0.714)
Model_3 = (0.620,0.674)
Model_4 = (0.604, 0.680)
```

```
Models = ("Mod_1","Mod_2","Mod_3","Mod_4")
Loss = (Model_1[0],Model_2[0],Model_3[0],Model_4[0])
Accuracy = (Model_1[1],Model_2[1],Model_3[1],Model_4[1])
```

```
fig, ax = plt.subplots()
ax.scatter(Loss,Accuracy)
for i, txt in enumerate(Models):
    ax.annotate(txt, (Loss[i],Accuracy[i] ))
plt.title("Summary for Accuracy and Loss")
plt.ylabel("Accuracy")
plt.xlabel("Loss")

plt.show()
```



Conclusions: From the above graph we can conclude that model 2 is the best among all with higher accuracy and minimum loss, however model model 4 has the highest loss

Recommendation: As we can see model 2 is performing best amoung all 4 models hence we should choose model with filters from 32 to 256, 5 Input Layers, Augmented Images and Dropout rate of 0.5

Increase your training sample size. You may pick any amount. Keep the validation and test samples the same as above. Optimize your network (again training from scratch).What performance did you achieve?

Considering Training Sample – 2000

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1250)
make_subset("test", start_index=1500, end_index=1750)
```

Data Pre-Processing: Using image_dataset_from_directory to read images

```
train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

Found 2000 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.

Viewing the size of the images

```
for data_batch, labels_batch in train_dataset:
    print("data batch shape:", data_batch.shape)
    print("labels batch shape:", labels_batch.shape)
    break
```

data batch shape: (32, 180, 180, 3)labels
batch shape: (32,)

Viewing the size of the images

```
data_augmentation_1 = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.15),
        layers.RandomZoom(0.25)
    ]
)
```

Model - 5 MaxPooling Operation with Increase in filters from 32 to 256 in 5 Input Layers with the data being used from the Augmented Images and a droput rate of 0.5 (Training Sample - 2000)

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation_1(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

```
Model: "model_6" Layer (type)         Output Shape  Param #

=================================================================


rescaling_6 (Rescaling)          (None, 180,     180, 3) 0
conv2d_32 (Conv2D)  (None, 178,     178, 32)         896
max_pooling2d_26 (MaxPoolin g2D) (None, 89,      89, 32) 0
conv2d_33 (Conv2D)  (None, 87, 87, 64)       18496
max_pooling2d_27 (MaxPoolin g2D) (None, 43, 43, 64)      0
conv2d_34 (Conv2D)  (None, 41, 41, 128)     73856
max_pooling2d_28 (MaxPoolin g2D) (None, 20, 20, 128)     0
conv2d_35 (Conv2D)  (None, 18, 18, 256)     295168
max_pooling2d_29 (MaxPoolin g2D) (None, 9, 9, 256)       0
conv2d_36 (Conv2D)  (None, 7, 7, 256)        590080
flatten_6 (Flatten)      (None, 12544) 0
dropout_4 (Dropout)  (None, 12544) 0
dense_6 (Dense)         (None, 1)         12545

=================================================================
Total  params:  991,041
Trainable  params:  991,041
Non-trainable params: 0
_________________________________________________________________
```

Training the model 5

```
# Compiling  the  Model
model.compile(loss= "binary_crossentropy",
                optimizer= "adam",
                metrics= ['accuracy'])


# Monitoring  the  best  validation  loss  using  Callbacks
callbacks = ModelCheckpoint(
                filepath  =  "model5.keras",
```

```
            save_best_only= True,
            monitor= "val_loss"
            )

# Model Fit
Model_5 = model.fit(
            train_dataset,
            epochs= 50,
            validation_data= validation_dataset,
            callbacks= callbacks
            )
```

```
Epoch 1/50
63/63 [==============================] - 5s 28ms/step - loss: 0.6929 - accuracy:
0.5080 - val_loss: 0.6919 - val_accuracy: 0.5200Epoch
2/50
63/63 [==============================] - 1s 22ms/step - loss: 0.6923 - accuracy:
0.5315 - val_loss: 0.6891 - val_accuracy: 0.5060Epoch
3/50
63/63 [==============================] - 1s 21ms/step - loss: 0.6939 - accuracy:
0.5160 - val_loss: 0.6923 - val_accuracy: 0.5000Epoch
4/50
63/63 [==============================] - 1s 22ms/step - loss: 0.6915 - accuracy:
0.5110 - val_loss: 0.6818 - val_accuracy: 0.5400Epoch
5/50
63/63 [==============================] - 1s 22ms/step - loss: 0.6784 - accuracy:
0.5490 - val_loss: 0.6802 - val_accuracy: 0.5420Epoch
6/50
63/63 [==============================] - 1s 22ms/step - loss: 0.6753 - accuracy:
0.5820 - val_loss: 0.6553 - val_accuracy: 0.6360Epoch
7/50
63/63 [==============================] - 1s 22ms/step - loss: 0.6520 - accuracy:
0.6335 - val_loss: 0.6544 - val_accuracy: 0.6360Epoch
8/50
63/63 [==============================] - 1s 22ms/step - loss: 0.6560 - accuracy:
0.6250 - val_loss: 0.6437 - val_accuracy: 0.6120Epoch
9/50
63/63 [==============================] - 2s 23ms/step - loss: 0.6394 - accuracy:
0.6525 - val_loss: 0.6315 - val_accuracy: 0.6500Epoch
10/50
```

63/63 [==============================] - 2s 23ms/step - loss: 0.6324 - accuracy: 0.6445 - val_loss: 0.6222 - val_accuracy: 0.6580Epoch 11/50
63/63 [==============================] - 2s 23ms/step - loss: 0.6170 - accuracy: 0.6675 - val_loss: 0.5998 - val_accuracy: 0.6600Epoch 12/50
63/63 [==============================] - 1s 21ms/step - loss: 0.5911 - accuracy:

0.6905 - val_loss: 0.6325 - val_accuracy: 0.6320Epoch
13/50
63/63 [==============================] - 1s 22ms/step - loss: 0.6122 - accuracy:
0.6660 - val_loss: 0.5813 - val_accuracy: 0.6880Epoch
14/50
63/63 [==============================] - 1s 22ms/step - loss: 0.5735 - accuracy:
0.7005 - val_loss: 0.5572 - val_accuracy: 0.7020Epoch
15/50
63/63 [==============================] - 1s 21ms/step - loss: 0.5708 - accuracy:
0.7025 - val_loss: 0.5659 - val_accuracy: 0.7080Epoch
16/50
63/63 [==============================] - 1s 21ms/step - loss: 0.5676 - accuracy:
0.7020 - val_loss: 0.5585 - val_accuracy: 0.7000Epoch
17/50
63/63 [==============================] - 1s 21ms/step - loss: 0.5638 - accuracy:
0.7115 - val_loss: 0.5795 - val_accuracy: 0.7000Epoch
18/50
63/63 [==============================] - 1s 22ms/step - loss: 0.5470 - accuracy:
0.7280 - val_loss: 0.5901 - val_accuracy: 0.6940Epoch
19/50
63/63 [==============================] - 2s 23ms/step - loss: 0.5239 - accuracy:
0.7555 - val_loss: 0.5207 - val_accuracy: 0.7420Epoch
20/50
63/63 [==============================] - 2s 23ms/step - loss: 0.5292 - accuracy:
0.7255 - val_loss: 0.5168 - val_accuracy: 0.7360Epoch
21/50
63/63 [==============================] - 2s 22ms/step - loss: 0.5009 - accuracy:
0.7620 - val_loss: 0.4870 - val_accuracy: 0.7760Epoch
22/50
63/63 [==============================] - 1s 21ms/step - loss: 0.4882 - accuracy:
0.7580 - val_loss: 0.5182 - val_accuracy: 0.7520Epoch
23/50
63/63 [==============================] - 1s 22ms/step - loss: 0.4812 - accuracy:
0.7655 - val_loss: 0.4819 - val_accuracy: 0.7760Epoch
24/50
63/63 [==============================] - 1s 22ms/step - loss: 0.4701 - accuracy:
0.7695 - val_loss: 0.4780 - val_accuracy: 0.7800Epoch
25/50
63/63 [==============================] - 1s 21ms/step - loss: 0.4606 - accuracy:

0.7750 - val_loss: 0.5185 - val_accuracy: 0.7580Epoch
26/50
63/63 [==============================] - 1s 22ms/step - loss: 0.4738 - accuracy:
0.7590 - val_loss: 0.4672 - val_accuracy: 0.7680Epoch
27/50
63/63 [==============================] - 1s 21ms/step - loss: 0.4837 - accuracy:
0.7665 - val_loss: 0.5049 - val_accuracy: 0.7500Epoch
28/50
63/63 [==============================] - 1s 21ms/step - loss: 0.4443 - accuracy:

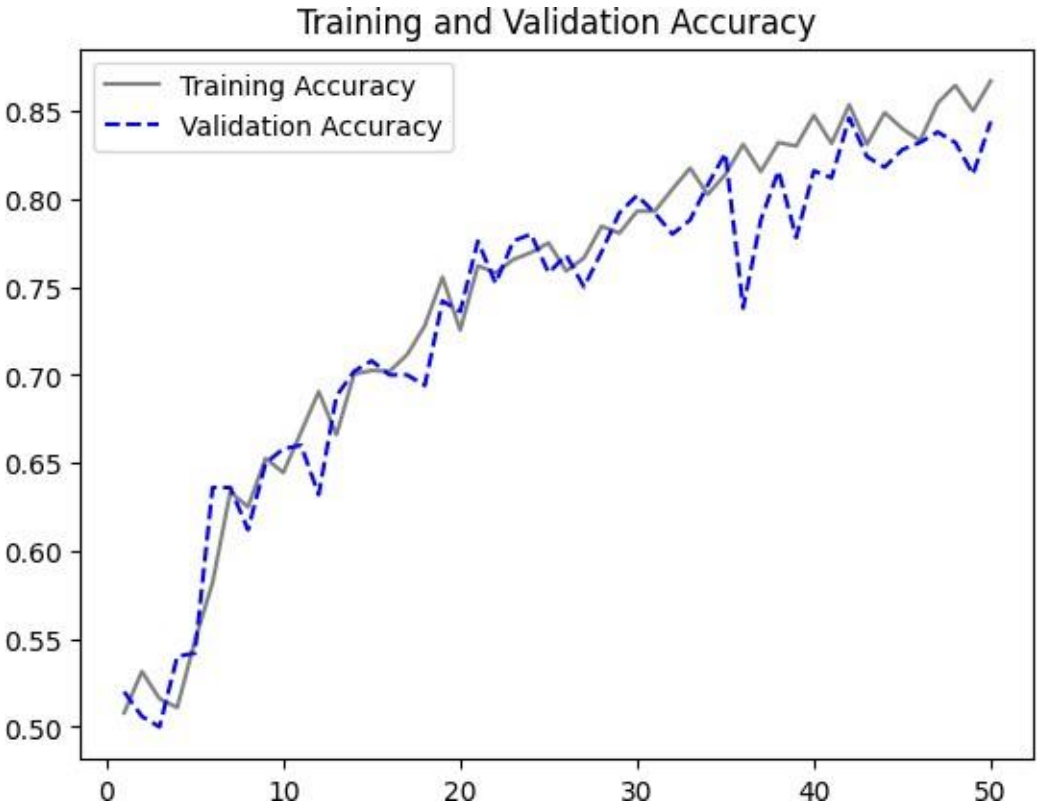
0.7845 - val_loss: 0.5045 - val_accuracy: 0.7700Epoch
29/50
63/63 [==============================] - 1s 22ms/step - loss: 0.4652 - accuracy:
0.7805 - val_loss: 0.4933 - val_accuracy: 0.7920Epoch
30/50
63/63 [==============================] - 2s 22ms/step - loss: 0.4394 - accuracy:
0.7930 - val_loss: 0.4766 - val_accuracy: 0.8020Epoch
31/50
63/63 [==============================] - 2s 23ms/step - loss: 0.4311 - accuracy:
0.7930 - val_loss: 0.4633 - val_accuracy: 0.7920Epoch
32/50
63/63 [==============================] - 1s 22ms/step - loss: 0.4287 - accuracy:
0.8055 - val_loss: 0.4620 - val_accuracy: 0.7800Epoch
33/50
63/63 [==============================] - 1s 22ms/step - loss: 0.4039 - accuracy:
0.8175 - val_loss: 0.4385 - val_accuracy: 0.7880Epoch
34/50
63/63 [==============================] - 1s 22ms/step - loss: 0.4069 - accuracy:
0.8025 - val_loss: 0.4331 - val_accuracy: 0.8080Epoch
35/50
63/63 [==============================] - 1s 22ms/step - loss: 0.4094 - accuracy:
0.8140 - val_loss: 0.4185 - val_accuracy: 0.8260Epoch
36/50
63/63 [==============================] - 1s 21ms/step - loss: 0.3839 - accuracy:
0.8310 - val_loss: 0.5114 - val_accuracy: 0.7380Epoch
37/50
63/63 [==============================] - 1s 21ms/step - loss: 0.4054 - accuracy:
0.8155 - val_loss: 0.4780 - val_accuracy: 0.7880Epoch

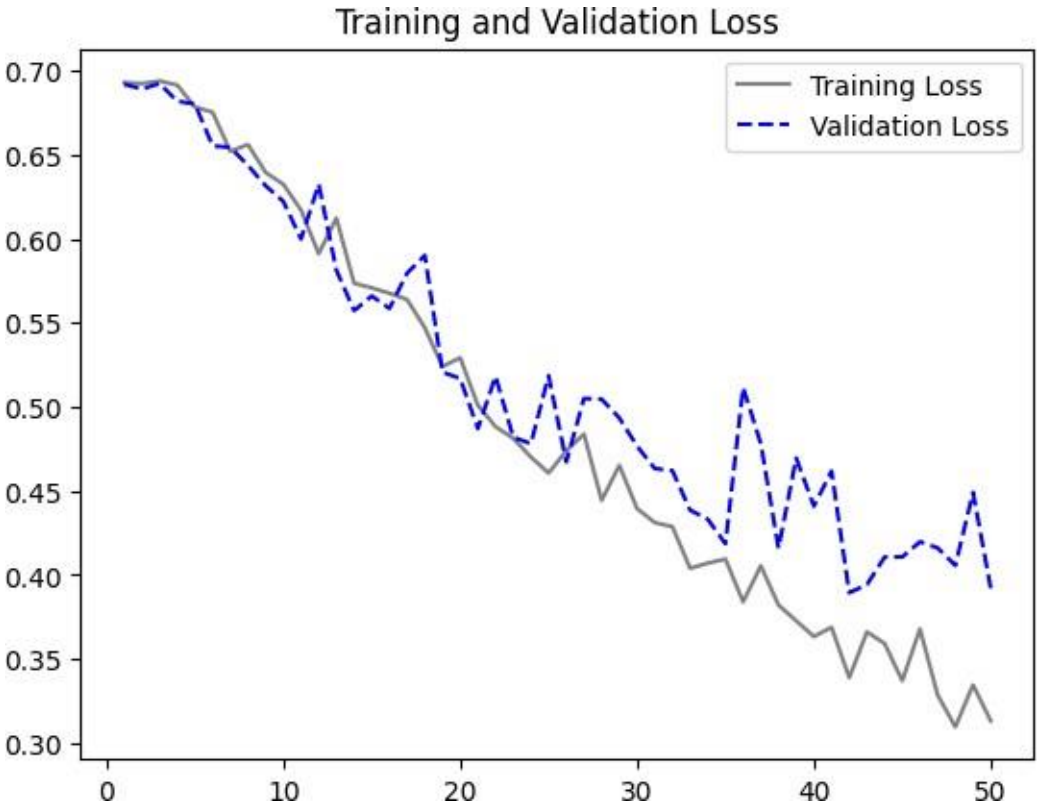
38/50
63/63 [==============================] - 1s 22ms/step - loss: 0.3820 - accuracy: 0.8320 - val_loss: 0.4158 - val_accuracy: 0.8160Epoch 39/50
63/63 [==============================] - 2s 22ms/step - loss: 0.3727 - accuracy: 0.8300 - val_loss: 0.4695 - val_accuracy: 0.7780Epoch 40/50
63/63 [==============================] - 1s 22ms/step - loss: 0.3632 - accuracy: 0.8475 - val_loss: 0.4408 - val_accuracy: 0.8160Epoch 41/50
63/63 [==============================] - 1s 21ms/step - loss: 0.3688 - accuracy: 0.8315 - val_loss: 0.4618 - val_accuracy: 0.8120Epoch 42/50
63/63 [==============================] - 1s 22ms/step - loss: 0.3388 - accuracy: 0.8535 - val_loss: 0.3893 - val_accuracy: 0.8460Epoch 43/50
63/63 [==============================] - 1s 21ms/step - loss: 0.3660 - accuracy: 0.8310 - val_loss: 0.3941 - val_accuracy: 0.8240Epoch 44/50
63/63 [==============================] - 1s 21ms/step - loss: 0.3591 - accuracy:

```
0.8490 - val_loss: 0.4106 - val_accuracy: 0.8180Epoch
45/50
63/63 [==============================] - 1s 21ms/step - loss: 0.3370 - accuracy:
0.8400 - val_loss: 0.4107 - val_accuracy: 0.8280Epoch
46/50
63/63 [==============================] - 1s 21ms/step - loss: 0.3678 - accuracy:
0.8330 - val_loss: 0.4197 - val_accuracy: 0.8320Epoch
47/50
63/63 [==============================] - 1s 21ms/step - loss: 0.3285 - accuracy:
0.8545 - val_loss: 0.4160 - val_accuracy: 0.8380Epoch
48/50
63/63 [==============================] - 1s 21ms/step - loss: 0.3095 - accuracy:
0.8645 - val_loss: 0.4056 - val_accuracy: 0.8320Epoch
49/50
63/63 [==============================] - 1s 22ms/step - loss: 0.3345 - accuracy:
0.8500 - val_loss: 0.4490 - val_accuracy: 0.8140Epoch
50/50
63/63 [==============================] - 2s 22ms/step - loss: 0.3131 - accuracy:
0.8670 - val_loss: 0.3920 - val_accuracy: 0.8440
```

Visualizing the Training and Validation Accuracy/Loss 2

```
accuracy = Model_5.history["accuracy"]
val_accuracy = Model_5.history["val_accuracy"]

loss = Model_5.history["loss"]
val_loss = Model_5.history["val_loss"]

epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, color="grey", label="Training Accuracy")
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed",␣
↪label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
10
20
30
40
50



Evaluating the performance of Model_5 on test set

```
best_model = keras.models.load_model("model5.keras")
Model5_Results = best_model.evaluate(test_dataset)
print(f"Loss: {Model5_Results[0]:.3f}")
print(f"Accuracy: {Model5_Results[1]:.3f}")
```

16/16 [==============================] - 0s 9ms/step - loss: 0.4590 - accuracy: 0.8140

Loss: 0.459

Accuracy: 0.814

Summary : The accuracy of the second model, which was only generated with 1000 training examples, was 71%, but the accuracy of the same model increased to 81%, or a 10% improvement, when training samples were increased to 2000.

Model - 6 Strides Operation with Padding being “Same” with Increase in filters from 32 to 256in 5 Input Layers with the data being used from the Augmented Images and a droput rate of 0.5 (Training Sample - 2000)

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation_1(inputs)
```

```
x =  layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, strides=2, activation="relu",␣
 ↪padding="same")(x)
x = layers.Conv2D(filters=64, kernel_size=3, strides=2, activation="relu",␣
 ↪padding="same")(x)
x = layers.Conv2D(filters=128, kernel_size=3, strides=2, activation="relu",␣
 ↪padding="same")(x)
x = layers.Conv2D(filters=256, kernel_size=3, strides=2, activation="relu",␣
 ↪padding="same")(x)
x = layers.Conv2D(filters=256, kernel_size=3, strides=2, activation="relu",␣
 ↪padding="same")(x)
x =  layers.Flatten()(x)
x =  layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

Model: "model_8"

Layer (type)	Output Shape	Param #
input_9 (InputLayer)	[(None, 180, 180, 3)]	
sequential_2 (Sequential)	(None, 180, 180, 3)	
rescaling_8 (Rescaling)	(None, 180, 180, 3)	
conv2d_42 (Conv2D)	(None, 90, 90, 32)	
conv2d_43 (Conv2D)	(None, 45, 45, 64)	
conv2d_44 (Conv2D)	(None, 23, 23, 128)	
conv2d_45 (Conv2D)	(None, 12, 12, 256)	
conv2d_46 (Conv2D)	(None, 6, 6, 256)	

```
 se_8 (Dense)                 (None, 1)                 9217

=================================================================
Total params: 987,713
Trainable params: 987,713
Non-trainable params: 0
_________________________________________________________________
```

den

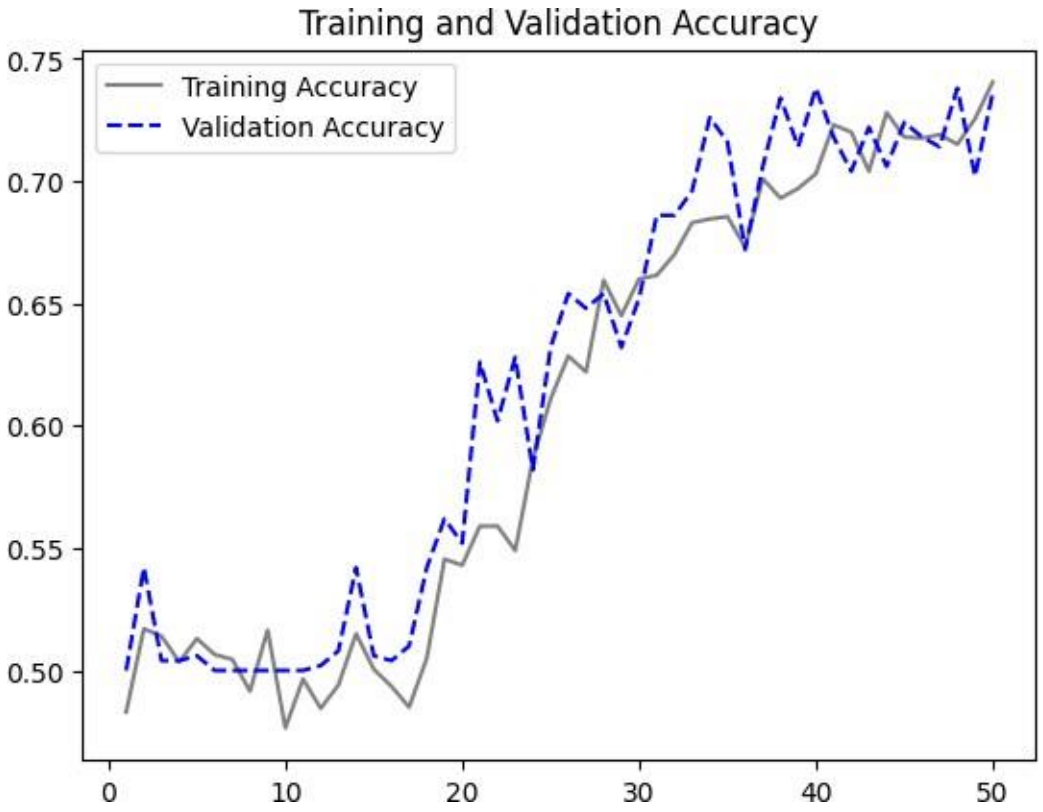
Training the model 6

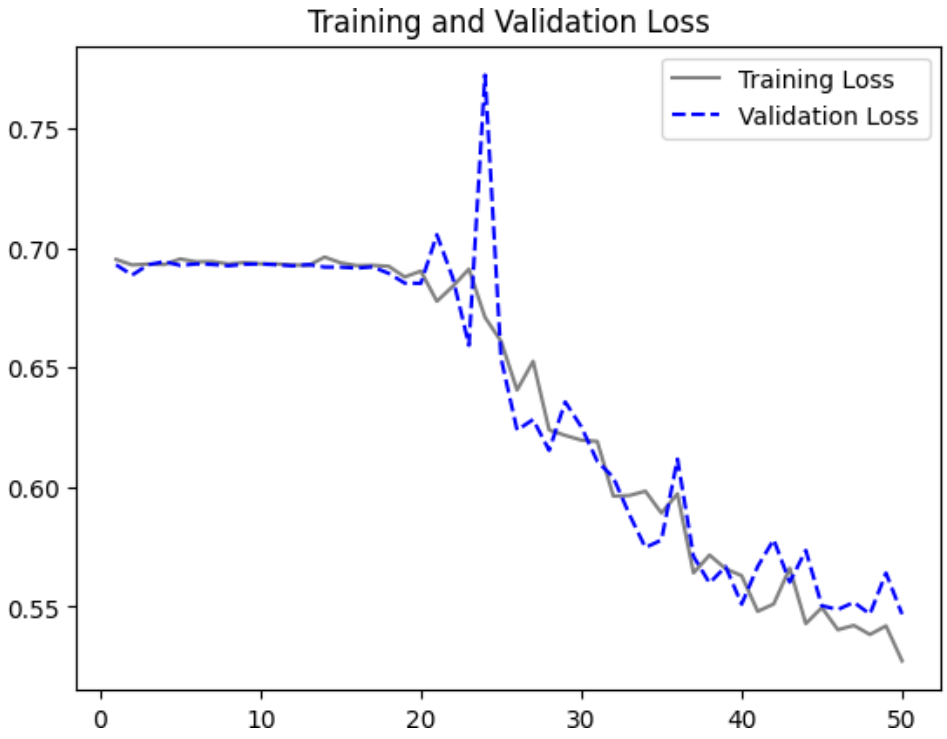
```
# Compiling the Model
model.compile(loss= "binary_crossentropy",
              optimizer= "adam",
              metrics= ['accuracy'])


# Monitoring the best validation loss using Callbacks
callbacks = ModelCheckpoint(
              filepath = "model6.keras",
              save_best_only= True,
              monitor= "val_loss"
              )

# Model Fit
Model_6 = model.fit(
            train_dataset,
            epochs= 50,
            validation_data= validation_dataset,
            callbacks= callbacks
            )
```

Visualizing the Training and Validation Accuracy/Loss

```
accuracy = Model_6.history["accuracy"]
val_accuracy = Model_6.history["val_accuracy"]

loss = Model_6.history["loss"]
val_loss = Model_6.history["val_loss"]

epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, color="grey", label="Training Accuracy")
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed",␣
↪label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
```

```
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```





16/16 [==============================] - 0s 8ms/step - loss: 0.6016 - accuracy: 0.6740

Loss: 0.602

Accuracy: 0.674

Summary for Question 2: We did try to build 2 more models with training sample being 2000. Now lets compare the loss and Accuracy of 3 models to see which model gives better result

Model 2: filters from 32 to 256, 5 Input Layers, Augmented Images and Droput rate of 0.5, training size 1000

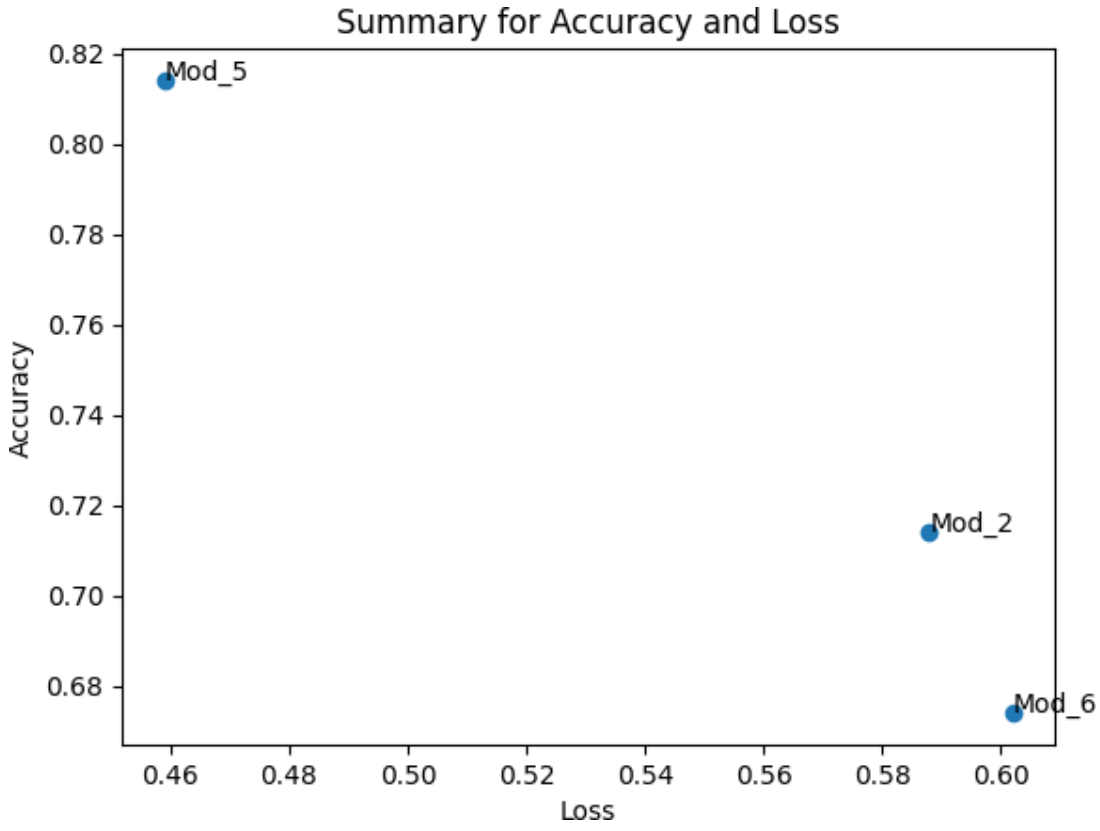
Model 5: filters from 32 to 256, 5 Input Layers, Augmented Images and Droput rate of 0.5, training size 2000

Model 6: filters from 32 to 256, 5 Input Layers, Augmented Images and Droput rate of 0.5, training size 2000, Padding being same

```
Model_5 = (0.459,0.814)
Model_6 = (0.602,0.674)
```

```
Models_2 = ("Mod_2","Mod_5","Mod_6")
Loss_2 = (Model_2[0],Model_5[0],Model_6[0])
Accuracy_2 = (Model_2[1],Model_5[1],Model_6[1])
```

```
fig, ax = plt.subplots()
ax.scatter(Loss_2,Accuracy_2)
for i, txt in enumerate(Models_2):
    ax.annotate(txt, (Loss_2[i],Accuracy_2[i] ))
plt.title("Summary for Accuracy and Loss")
plt.ylabel("Accuracy")
plt.xlabel("Loss")

plt.show()
```



When the models' performances were compared, it was found that the model did not gain much from using strides with padding. With the addition of a Max Pooling Layer, Model 5 outperformed the Strides model in accuracy by 14%. Additionally, an improved accuracy of 81%

was attained by fine-tuning the network and expanding the training dataset from 1000 to 2000 samples.

We plotted Models 5 and 6 to answer the second question and provide a visual comparison of their performance. The graphs clearly show that Model 5 had the lowest loss of 45.9% and the highest accuracy of all the models, reaching 81%. The model performed significantly better once the training samples were increased to 2000 and various augmented photos were added.

1. Now change your training sample so that you achieve better performance than those from Steps1 and 2. This sample size may be larger, or smaller than those in the previous steps. The objectiveis to find the ideal training sample size to get best prediction results

As we saw in above graph that with the increase in training sample size the Accuracy is also increasing hence will increase the sample size to 3000 and 5000 for better performance

Training Sample 3000

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_1")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=1500)
make_subset("validation", start_index=1000, end_index=1250)
make_subset("test", start_index=1500, end_index=1750)
```

Found 3000 files belonging to 2 classes.

Found 500 files belonging to 2 classes.

```
for data_batch, labels_batch in train_dataset:
    print("data batch shape:", data_batch.shape)
    print("labels batch shape:", labels_batch.shape)
    break
```

data batch shape: (32, 180, 180, 3)
labels batch shape: (32,)

Using few of the techniques such as random flip, random zoom, random rotation so as to createaugmented versions of the image

```
data_augmentation_2 = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.15),
        layers.RandomZoom(0.25)
    ]
)
```

Model - 7 MaxPooling Operation with Increase in filters from 32 to 256 in 5 Input Layers with the data being used from the Augmented Images and a droput rate of 0.5 (Training Sample - 3000)

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation_2(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

```
Model: "model_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_10 (InputLayer)       [(None, 180, 180, 3)]     0
```

```
sequential_3 (Sequential)    (None, 180, 180, 3)       0
rescaling_9 (Rescaling)      (None, 180, 180, 3)       0
conv2d_47 (Conv2D)           (None, 178, 178, 32)      896
max_pooling2d_30 (MaxPoolin  (None, 89, 89, 32)        0
g2D)
conv2d_48 (Conv2D)           (None, 87, 87, 64)        18496
max_pooling2d_31 (MaxPoolin  (None, 43, 43, 64)        0
g2D)
conv2d_49 (Conv2D)           (None, 41, 41, 128)       73856
max_pooling2d_32 (MaxPoolin  (None, 20, 20, 128)       0
g2D)
conv2d_50 (Conv2D)           (None, 18, 18, 256)       295168
max_pooling2d_33 (MaxPoolin  (None, 9, 9, 256)         0
g2D)
conv2d_51 (Conv2D)           (None, 7, 7, 256)         590080
flatten_9 (Flatten)          (None, 12544)             0
dropout_7 (Dropout)          (None, 12544)             0
dense_9 (Dense)              (None, 1)                 12545
=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
_________________________________________________________________
```

```
# Compiling the Model
model.compile(loss= "binary_crossentropy",
              optimizer= "adam",
              metrics= ['accuracy'])


# Monitoring the best validation loss using Callbacks
callbacks = ModelCheckpoint(
```

```
            filepath = "model7.keras",
            save_best_only= True,
            monitor= "val_loss"
            )

# Model Fit
Model_7 = model.fit(
          train_dataset,
          epochs= 50,
          validation_data= validation_dataset,
          callbacks= callbacks
          )
```

```
Epoch 1/50
94/94 [==============================] - 5s 26ms/step - loss: 0.6936 - accuracy:
0.5023 - val_loss: 0.6928 - val_accuracy: 0.5000Epoch
2/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6941 - accuracy:
0.4923 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch
3/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6939 - accuracy:
0.4943 - val_loss: 0.6932 - val_accuracy: 0.5000Epoch
4/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6936 - accuracy:
0.5043 - val_loss: 0.6932 - val_accuracy: 0.5000Epoch
5/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6938 - accuracy:
0.4970 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch
6/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6933 - accuracy:
0.4960 - val_loss: 0.6926 - val_accuracy: 0.5000Epoch
7/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6938 - accuracy:
0.4967 - val_loss: 0.6932 - val_accuracy: 0.5000Epoch
8/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy:
0.5030 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch
9/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6933 - accuracy:
0.4910 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch
```

10/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6933 - accuracy: 0.4900 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 11/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.5003 - val_loss: 0.6932 - val_accuracy: 0.5000Epoch 12/50

94/94 [==============================] - 2s 20ms/step - loss: 0.6933 - accuracy: 0.4803 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 13/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4867 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 14/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4917 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 15/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4947 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 16/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6932 - accuracy: 0.4853 - val_loss: 0.6932 - val_accuracy: 0.4980Epoch 17/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6935 - accuracy: 0.4973 - val_loss: 0.6933 - val_accuracy: 0.5000Epoch 18/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6934 - accuracy: 0.4970 - val_loss: 0.6932 - val_accuracy: 0.5000Epoch 19/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6933 - accuracy: 0.4827 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 20/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4867 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 21/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4880 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 22/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4873 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 23/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6932 - accuracy: 0.4880 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 24/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6932 - accuracy: 0.4893 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 25/50

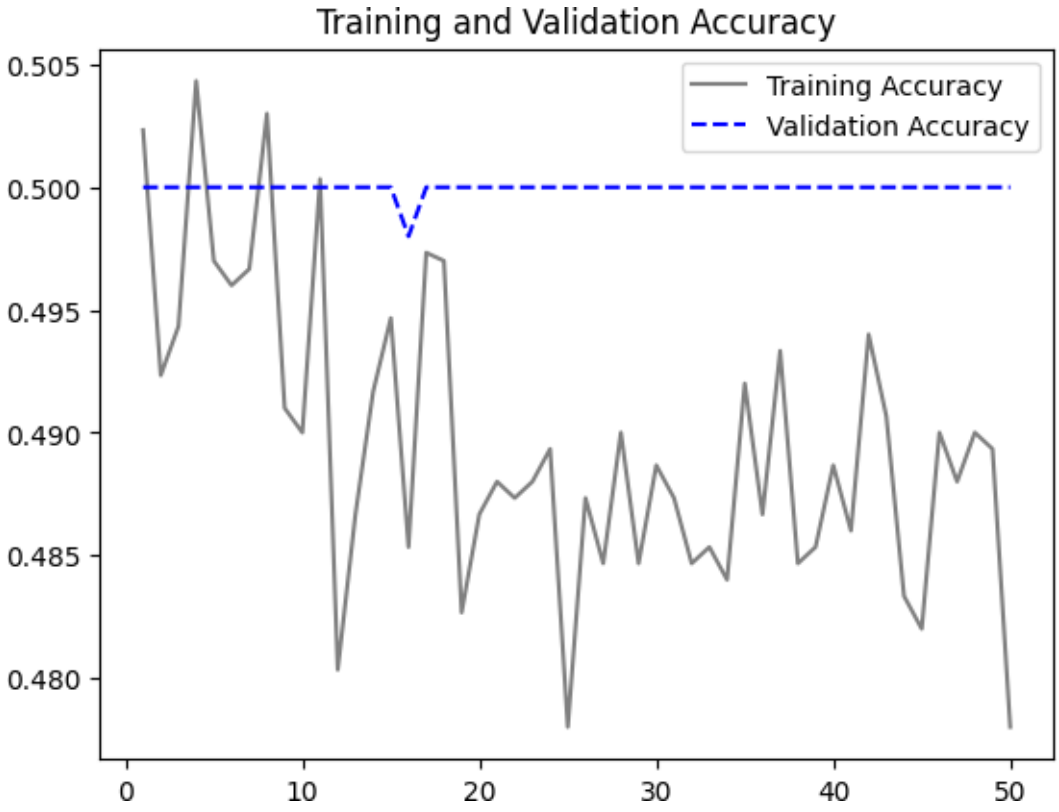
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4780 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 26/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4873 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 27/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4847 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 28/50

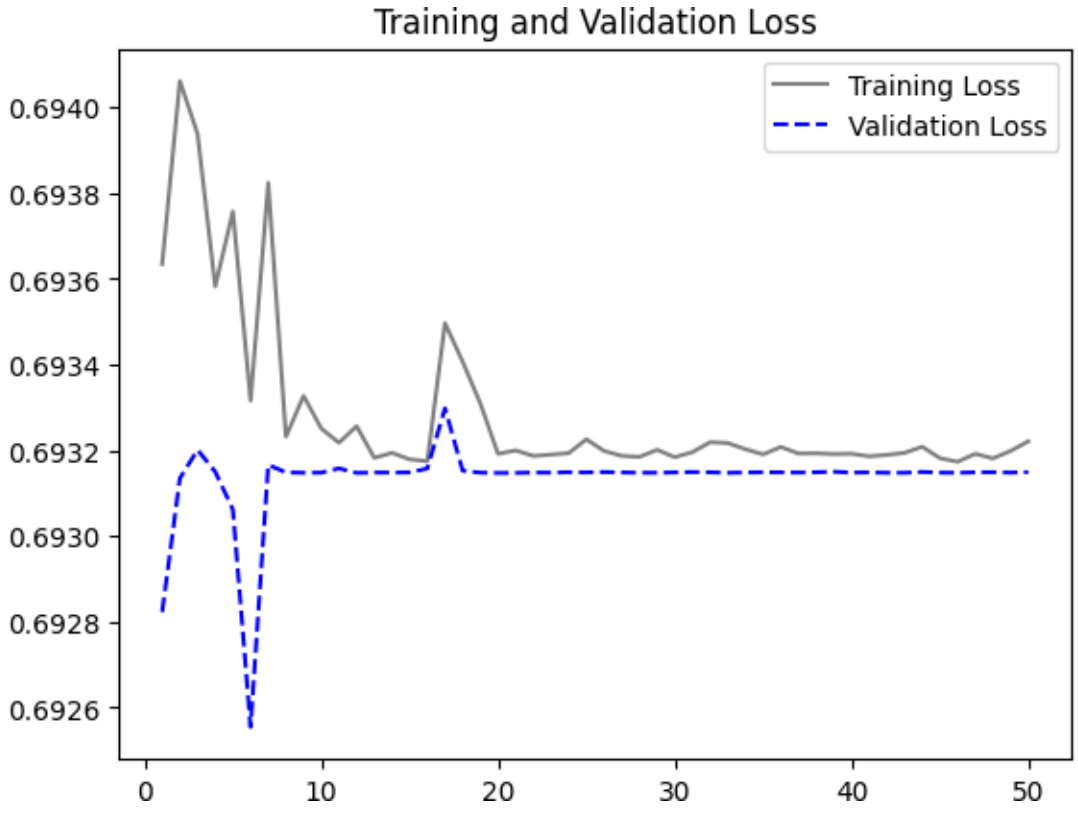
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4900 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 29/50
94/94 [==============================] - 2s 19ms/step - loss: 0.6932 - accuracy: 0.4847 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 30/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4887 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 31/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4873 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 32/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6932 - accuracy: 0.4847 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 33/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4853 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 34/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4840 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 35/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4920 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 36/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4867 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 37/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4933 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 38/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4847 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 39/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4853 - val_loss: 0.6932 - val_accuracy: 0.5000Epoch 40/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy: 0.4887 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 41/50

94/94 [==============================] - 2s 19ms/step - loss: 0.6932 - accuracy: 0.4860 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 42/50
94/94 [==============================] - 2s 19ms/step - loss: 0.6932 - accuracy: 0.4940 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 43/50
94/94 [==============================] - 2s 19ms/step - loss: 0.6932 - accuracy: 0.4907 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch 44/50

```
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy:
0.4833 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch
45/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy:
0.4820 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch
46/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy:
0.4900 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch
47/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6932 - accuracy:
0.4880 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch
48/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy:
0.4900 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch
49/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6932 - accuracy:
0.4893 - val_loss: 0.6931 - val_accuracy: 0.5000Epoch
50/50
94/94 [==============================] - 2s 19ms/step - loss: 0.6932 - accuracy:
0.4780 - val_loss: 0.6931 - val_accuracy: 0.5000
```

```
accuracy = Model_7.history["accuracy"]
val_accuracy = Model_7.history["val_accuracy"]

loss = Model_7.history["loss"]
val_loss = Model_7.history["val_loss"]

epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, color="grey", label="Training Accuracy")
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed",␣
↪label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
0.505
0.500
0.495
0.490
0.485
0.480
0
10
20
30
40
50


Training and Validation Loss
Training Loss
Validation Loss
0.6940
0.6938
0.6936
0.6934
0.6932
0.6930
0.6928
0.6926
0
10
20
30
40
50

```
best_model = keras.models.load_model("model7.keras")
Model7_Results = best_model.evaluate(test_dataset)
print(f"Loss: {Model7_Results[0]:.3f}")
print(f"Accuracy: {Model7_Results[1]:.3f}")
```

16/16 [==============================] - 0s 9ms/step - loss: 0.6930 - accuracy: 0.5000

Loss: 0.693

Accuracy: 0.500

In the previous Model 6, we attempted to replace the conventional max pooling operation with strides, but the results were not as promising as expected. and in model 7 we used Maxpooling only. Therefore, we are exploring a hybrid approach that combines both max pooling and stridesto evaluate the performance of this new model.

Max pooling is a downsampling operation that reduces the spatial dimensions of the feature map,aiming to capture the most prominent features while discarding less relevant information. On the other hand, strides determine the step rate of the sliding window used to extract and learn the features from the data. This hybrid approach aims to leverage the advantages of both tech- niques, potentially enhancing the model's ability to capture intricate patterns and features while maintaining computational efficiency.

Model - 8 MaxPooling + Strides of Step-Size 2 Operation with Increase in filters from 32 to 256 in 5 Input Layers with the data being used from the Augmented Images and a droput rate of 0.5 (Training Sample - 3000)

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation_2(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2,strides=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

conv2d_52 (Conv2D)	(None, 178, 178, 32)	896	
max_pooling2d_34 (MaxPooling2D)	(None, 89, 89, 32)	0	
conv2d_53 (Conv2D)	(None, 87, 87, 64)	18496	
max_pooling2d_35 (MaxPooling2D)	(None, 43, 43, 64)	0	
conv2d_54 (Conv2D)	(None, 41, 41, 128)	73856	
max_pooling2d_36 (MaxPooling2D)	(None, 20, 20, 128)	0	

conv2d_55 (Conv2D)	(None, 18, 18, 256)	295168
max_pooling2d_37 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_56 (Conv2D)	(None, 7, 7, 256)	590080
flatten_10 (Flatten)	(None, 12544)	0
dropout_8 (Dropout)	(None, 12544)	0
dense_10 (Dense)	(None, 1)	12545

===

Total params: 991,041

Trainable params: 991,041

Non-trainable params: 0

_____

```
# Compiling the Model
model.compile(loss= "binary_crossentropy",
              optimizer= "adam",
              metrics= ['accuracy'])


# Monitoring the best validation loss using Callbacks
callbacks = ModelCheckpoint(
             filepath = "model8.keras",
             save_best_only= True,
             monitor= "val_loss"
             )

# Model Fit
Model_8 = model.fit(
           train_dataset,
           epochs= 50,
           validation_data= validation_dataset,
           callbacks= callbacks
           )
```

Epoch 1/50

94/94 [==============================] - 5s 23ms/step - loss: 0.6938 - accuracy: 0.5100 - val_loss: 0.6942 - val_accuracy: 0.5000Epoch

2/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6936 - accuracy: 0.4983 - val_loss: 0.6925 - val_accuracy: 0.5000Epoch
3/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6930 - accuracy:

0.5043 - val_loss: 0.6860 - val_accuracy: 0.5280Epoch
4/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6941 - accuracy:
0.5357 - val_loss: 0.6895 - val_accuracy: 0.5140Epoch
5/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6923 - accuracy:
0.5283 - val_loss: 0.6779 - val_accuracy: 0.5960Epoch
6/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6636 - accuracy:
0.6100 - val_loss: 0.6525 - val_accuracy: 0.6080Epoch
7/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6574 - accuracy:
0.6167 - val_loss: 0.6636 - val_accuracy: 0.5900Epoch
8/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6490 - accuracy:
0.6207 - val_loss: 0.6100 - val_accuracy: 0.6780Epoch
9/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6166 - accuracy:
0.6590 - val_loss: 0.6242 - val_accuracy: 0.6540Epoch
10/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6142 - accuracy:
0.6663 - val_loss: 0.5849 - val_accuracy: 0.6700Epoch
11/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6060 - accuracy:
0.6783 - val_loss: 0.5497 - val_accuracy: 0.7160Epoch
12/50
94/94 [==============================] - 2s 20ms/step - loss: 0.5842 - accuracy:
0.6927 - val_loss: 0.5379 - val_accuracy: 0.7300Epoch
13/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5687 - accuracy:
0.7087 - val_loss: 0.5161 - val_accuracy: 0.7480Epoch
14/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5895 - accuracy:
0.6840 - val_loss: 0.5402 - val_accuracy: 0.7140Epoch
15/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5568 - accuracy:
0.7237 - val_loss: 0.5120 - val_accuracy: 0.7400Epoch
16/50
94/94 [==============================] - 2s 20ms/step - loss: 0.5681 - accuracy:

0.7040 - val_loss: 0.5505 - val_accuracy: 0.7020Epoch
17/50
94/94 [==============================] - 2s 20ms/step - loss: 0.5552 - accuracy:
0.7150 - val_loss: 0.5443 - val_accuracy: 0.7020Epoch
18/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5429 - accuracy:
0.7243 - val_loss: 0.5056 - val_accuracy: 0.7260Epoch
19/50
94/94 [==============================] - 2s 20ms/step - loss: 0.5190 - accuracy:

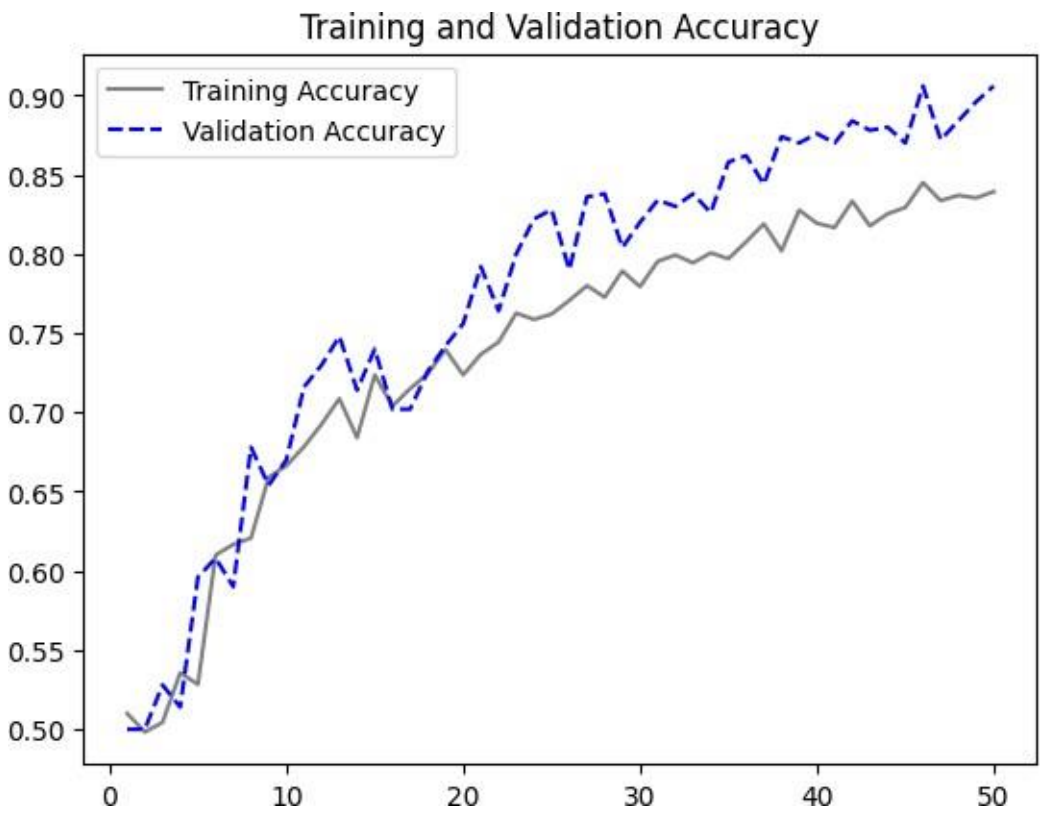
0.7397 - val_loss: 0.4963 - val_accuracy: 0.7420Epoch
20/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5277 - accuracy:
0.7237 - val_loss: 0.5128 - val_accuracy: 0.7560Epoch
21/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5307 - accuracy:
0.7367 - val_loss: 0.4508 - val_accuracy: 0.7920Epoch
22/50
94/94 [==============================] - 2s 20ms/step - loss: 0.5121 - accuracy:
0.7443 - val_loss: 0.4730 - val_accuracy: 0.7640Epoch
23/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4878 - accuracy:
0.7627 - val_loss: 0.4369 - val_accuracy: 0.8000Epoch
24/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4961 - accuracy:
0.7587 - val_loss: 0.4157 - val_accuracy: 0.8220Epoch
25/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4916 - accuracy:
0.7620 - val_loss: 0.3960 - val_accuracy: 0.8280Epoch
26/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4819 - accuracy:
0.7707 - val_loss: 0.4399 - val_accuracy: 0.7900Epoch
27/50
94/94 [==============================] - 2s 21ms/step - loss: 0.4730 - accuracy:
0.7800 - val_loss: 0.3948 - val_accuracy: 0.8360Epoch
28/50
94/94 [==============================] - 2s 21ms/step - loss: 0.4755 - accuracy:
0.7727 - val_loss: 0.3950 - val_accuracy: 0.8380Epoch
29/50
94/94 [==============================] - 2s 22ms/step - loss: 0.4522 - accuracy:
0.7893 - val_loss: 0.4043 - val_accuracy: 0.8040Epoch
30/50
94/94 [==============================] - 2s 22ms/step - loss: 0.4552 - accuracy:
0.7793 - val_loss: 0.3741 - val_accuracy: 0.8200Epoch
31/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4482 - accuracy:
0.7953 - val_loss: 0.3772 - val_accuracy: 0.8340Epoch
32/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4420 - accuracy:

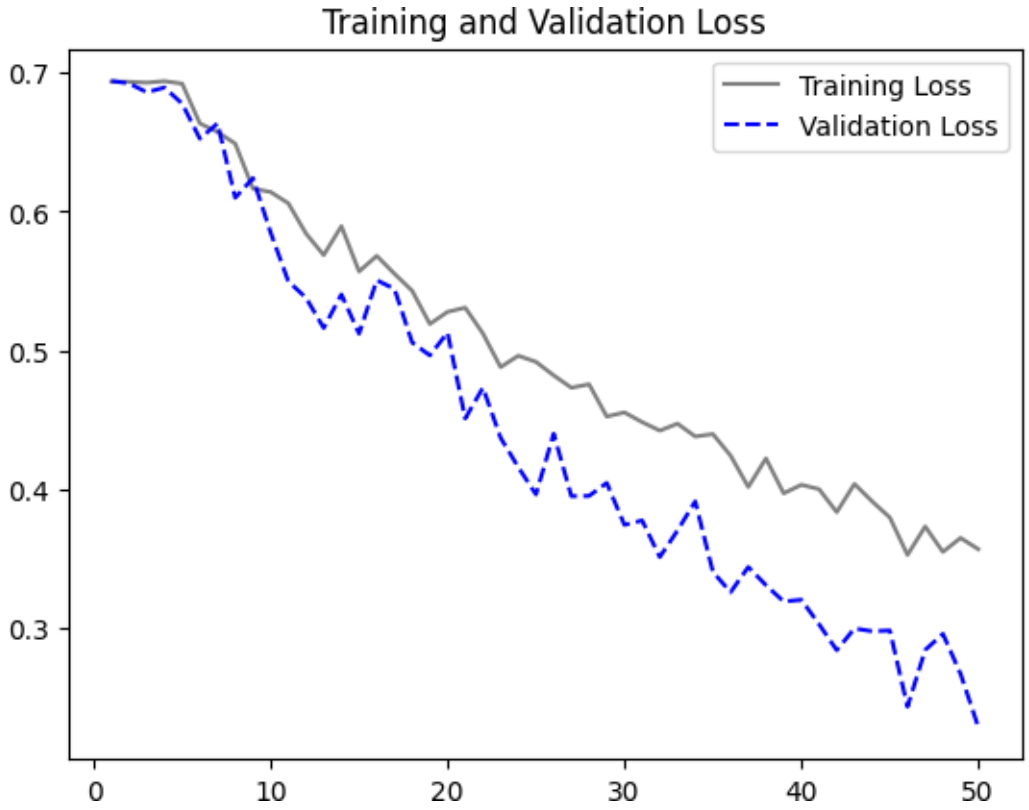
0.7993 - val_loss: 0.3509 - val_accuracy: 0.8300Epoch
33/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4471 - accuracy:
0.7943 - val_loss: 0.3701 - val_accuracy: 0.8380Epoch
34/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4378 - accuracy:
0.8007 - val_loss: 0.3910 - val_accuracy: 0.8260Epoch
35/50
94/94 [==============================] - 2s 21ms/step - loss: 0.4395 - accuracy:

0.7970 - val_loss: 0.3402 - val_accuracy: 0.8580Epoch
36/50
94/94 [==============================] - 2s 21ms/step - loss: 0.4240 - accuracy:
0.8077 - val_loss: 0.3255 - val_accuracy: 0.8620Epoch
37/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4014 - accuracy:
0.8190 - val_loss: 0.3437 - val_accuracy: 0.8440Epoch
38/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4220 - accuracy:
0.8020 - val_loss: 0.3306 - val_accuracy: 0.8740Epoch
39/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3968 - accuracy:
0.8277 - val_loss: 0.3188 - val_accuracy: 0.8700Epoch
40/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4030 - accuracy:
0.8193 - val_loss: 0.3201 - val_accuracy: 0.8760Epoch
41/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3999 - accuracy:
0.8167 - val_loss: 0.3024 - val_accuracy: 0.8700Epoch
42/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3833 - accuracy:
0.8333 - val_loss: 0.2837 - val_accuracy: 0.8840Epoch
43/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4036 - accuracy:
0.8177 - val_loss: 0.2994 - val_accuracy: 0.8780Epoch
44/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3910 - accuracy:
0.8253 - val_loss: 0.2974 - val_accuracy: 0.8800Epoch
45/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3794 - accuracy:
0.8293 - val_loss: 0.2979 - val_accuracy: 0.8700Epoch
46/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3524 - accuracy:
0.8450 - val_loss: 0.2430 - val_accuracy: 0.9060Epoch
47/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3729 - accuracy:
0.8337 - val_loss: 0.2839 - val_accuracy: 0.8720Epoch
48/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3546 - accuracy:

0.8370 - val_loss: 0.2954 - val_accuracy: 0.8840Epoch
49/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3647 - accuracy:
0.8353 - val_loss: 0.2666 - val_accuracy: 0.8960Epoch
50/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3566 - accuracy:
0.8393 - val_loss: 0.2284 - val_accuracy: 0.9060

```
accuracy = Model_8.history["accuracy"]
val_accuracy = Model_8.history["val_accuracy"]

loss = Model_8.history["loss"]
val_loss = Model_8.history["val_loss"]

epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, color="grey", label="Training Accuracy")
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed",
 ↪label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
 ↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```






```
best_model = keras.models.load_model("model8.keras")
Model8_Results = best_model.evaluate(test_dataset)
print(f"Loss: {Model8_Results[0]:.3f}")
print(f"Accuracy: {Model8_Results[1]:.3f}")
```

16/16 [==============================] - 1s 9ms/step - loss: 0.4570 - accuracy: 0.8120

Loss: 0.457

Accuracy: 0.812

Model - 9 MaxPooling + Strides of Step-Size 2 with Padding turned on Operation with Increase in filters from 32 to 512 in 5 Input Layers with the data being used from the Augmented Images and a droput rate of 0.5 (Training Sample - 3000)

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation_2(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x) x
= layers.MaxPooling2D(pool_size=2, strides=2, padding="same")(x) x =
layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x) x =
layers.MaxPooling2D(pool_size=2, strides=2, padding="same")(x) x =
layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
```

```
x = layers.MaxPooling2D(pool_size=2, strides=2, padding="same")(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2,strides=2, padding="same")(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

```
Model: "model_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_12 (InputLayer)       [(None, 180, 180, 3)]     0

 sequential_3 (Sequential)   (None, 180, 180, 3)       0

 rescaling_11 (Rescaling)    (None, 180, 180, 3)       0

 conv2d_57 (Conv2D)          (None, 178, 178, 32)      896

 max_pooling2d_38 (MaxPoolin (None, 89, 89, 32)        0
 g2D)

 conv2d_58 (Conv2D)          (None, 87, 87, 64)        18496

 max_pooling2d_39 (MaxPoolin (None, 44, 44, 64)        0
 g2D)

 conv2d_59 (Conv2D)          (None, 42, 42, 128)       73856

 max_pooling2d_40 (MaxPoolin (None, 21, 21, 128)       0
 g2D)

 conv2d_60 (Conv2D)          (None, 19, 19, 256)       295168

 max_pooling2d_41 (MaxPoolin (None, 10, 10, 256)       0
```

g2D)		
conv2d_61 (Conv2D)	(None, 8, 8, 512)	1180160
flatten_11 (Flatten)	(None, 32768)	0
dropout_9 (Dropout)	(None, 32768)	0

```
 dense_11 (Dense)             (None, 1)                 32769

=================================================================
Total params: 1,601,345
Trainable params: 1,601,345
Non-trainable params: 0
_________________________________________________________________
```

```python
# Compiling the Model
model.compile(loss= "binary_crossentropy",
              optimizer= "adam",
              metrics= ['accuracy'])


# Monitoring the best validation loss using Callbacks
callbacks = ModelCheckpoint(
             filepath = "model9.keras",
             save_best_only= True,
             monitor= "val_loss"
             )

# Model Fit
Model_9 = model.fit(
            train_dataset,
            epochs= 50,
            validation_data= validation_dataset,
            callbacks= callbacks
            )
```

```
Epoch 1/50
94/94 [==============================] - 5s 25ms/step - loss: 0.6943 - accuracy:
0.5223 - val_loss: 0.6780 - val_accuracy: 0.6320Epoch
2/50
94/94 [==============================] - 2s 22ms/step - loss: 0.6781 - accuracy:
0.5660 - val_loss: 0.6625 - val_accuracy: 0.5900Epoch
3/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6620 - accuracy:
0.6037 - val_loss: 0.7170 - val_accuracy: 0.5360Epoch
4/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6609 - accuracy:
```

0.6057 - val_loss: 0.6431 - val_accuracy: 0.6200Epoch
5/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6372 - accuracy:
0.6393 - val_loss: 0.6812 - val_accuracy: 0.6100Epoch
6/50
94/94 [==============================] - 2s 20ms/step - loss: 0.6213 - accuracy:
0.6593 - val_loss: 0.6584 - val_accuracy: 0.5500

```
Epoch 7/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6136 - accuracy:
0.6707 - val_loss: 0.5720 - val_accuracy: 0.6980Epoch
8/50
94/94 [==============================] - 2s 21ms/step - loss: 0.6005 - accuracy:
0.6743 - val_loss: 0.5669 - val_accuracy: 0.7260Epoch
9/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5941 - accuracy:
0.6890 - val_loss: 0.5684 - val_accuracy: 0.7040Epoch
10/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5889 - accuracy:
0.6870 - val_loss: 0.5436 - val_accuracy: 0.7300Epoch
11/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5593 - accuracy:
0.7120 - val_loss: 0.5168 - val_accuracy: 0.7540Epoch
12/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5537 - accuracy:
0.7140 - val_loss: 0.4523 - val_accuracy: 0.8120Epoch
13/50
94/94 [==============================] - 2s 20ms/step - loss: 0.5341 - accuracy:
0.7387 - val_loss: 0.5587 - val_accuracy: 0.7120Epoch
14/50
94/94 [==============================] - 2s 20ms/step - loss: 0.5299 - accuracy:
0.7347 - val_loss: 0.4961 - val_accuracy: 0.7700Epoch
15/50
94/94 [==============================] - 2s 20ms/step - loss: 0.5233 - accuracy:
0.7413 - val_loss: 0.4658 - val_accuracy: 0.8040Epoch
16/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5124 - accuracy:
0.7407 - val_loss: 0.5693 - val_accuracy: 0.7080Epoch
17/50
94/94 [==============================] - 2s 21ms/step - loss: 0.5045 - accuracy:
0.7597 - val_loss: 0.4244 - val_accuracy: 0.8140Epoch
18/50
94/94 [==============================] - 2s 21ms/step - loss: 0.4853 - accuracy:
0.7767 - val_loss: 0.4032 - val_accuracy: 0.8120Epoch
19/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4812 - accuracy:
0.7633 - val_loss: 0.4261 - val_accuracy: 0.8020Epoch
```

20/50
94/94 [==============================] - 2s 21ms/step - loss: 0.4809 - accuracy: 0.7717 - val_loss: 0.3830 - val_accuracy: 0.8460Epoch
21/50
94/94 [==============================] - 2s 22ms/step - loss: 0.4692 - accuracy: 0.7787 - val_loss: 0.3668 - val_accuracy: 0.8420Epoch
22/50
94/94 [==============================] - 2s 21ms/step - loss: 0.4480 - accuracy: 0.7903 - val_loss: 0.4268 - val_accuracy: 0.8100

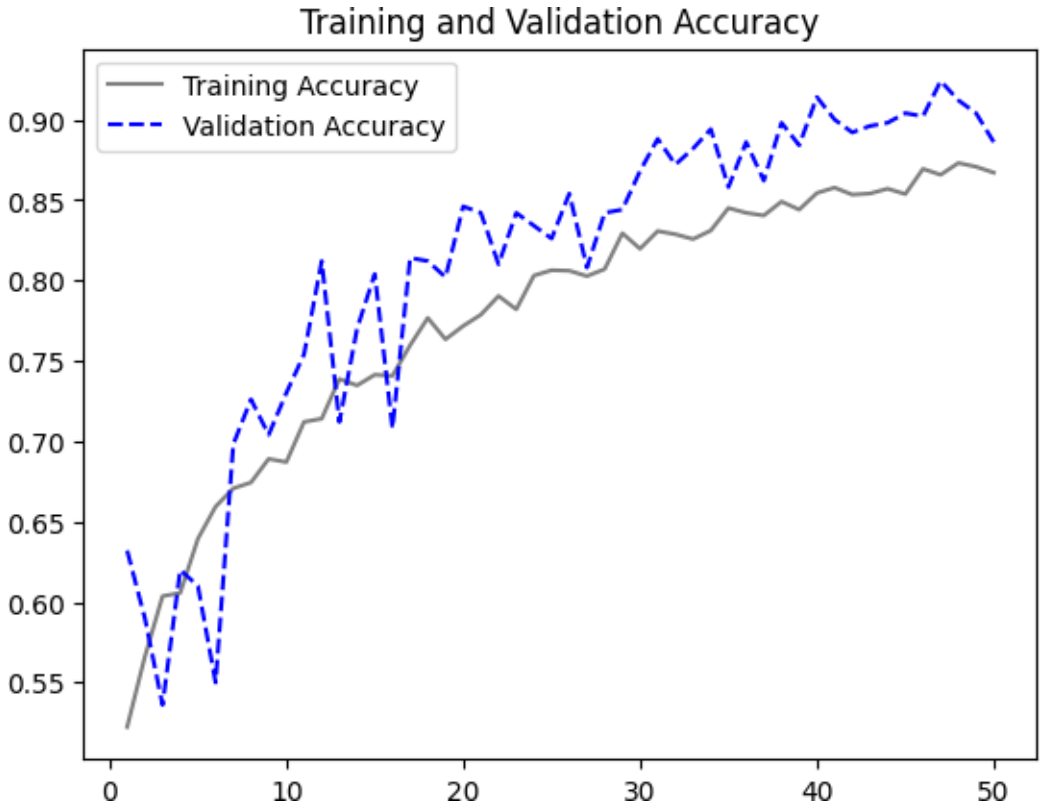
Epoch 23/50
94/94 [==============================] - 2s 22ms/step - loss: 0.4557 - accuracy: 0.7820 - val_loss: 0.3649 - val_accuracy: 0.8420Epoch 24/50
94/94 [==============================] - 2s 21ms/step - loss: 0.4379 - accuracy: 0.8030 - val_loss: 0.3709 - val_accuracy: 0.8340Epoch 25/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4321 - accuracy: 0.8063 - val_loss: 0.3698 - val_accuracy: 0.8260Epoch 26/50
94/94 [==============================] - 2s 21ms/step - loss: 0.4291 - accuracy: 0.8060 - val_loss: 0.3456 - val_accuracy: 0.8540Epoch 27/50
94/94 [==============================] - 2s 20ms/step - loss: 0.4157 - accuracy: 0.8027 - val_loss: 0.4344 - val_accuracy: 0.8080Epoch 28/50
94/94 [==============================] - 2s 21ms/step - loss: 0.4172 - accuracy: 0.8070 - val_loss: 0.3231 - val_accuracy: 0.8420Epoch 29/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3954 - accuracy: 0.8293 - val_loss: 0.3494 - val_accuracy: 0.8440Epoch 30/50
94/94 [==============================] - 2s 22ms/step - loss: 0.4075 - accuracy: 0.8197 - val_loss: 0.2916 - val_accuracy: 0.8680Epoch 31/50
94/94 [==============================] - 2s 22ms/step - loss: 0.3874 - accuracy: 0.8307 - val_loss: 0.2783 - val_accuracy: 0.8880Epoch 32/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3842 - accuracy: 0.8287 - val_loss: 0.3037 - val_accuracy: 0.8720Epoch 33/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3848 - accuracy: 0.8257 - val_loss: 0.2698 - val_accuracy: 0.8820Epoch 34/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3826 - accuracy: 0.8310 - val_loss: 0.2517 - val_accuracy: 0.8940Epoch 35/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3486 - accuracy: 0.8450 - val_loss: 0.3218 - val_accuracy: 0.8580Epoch

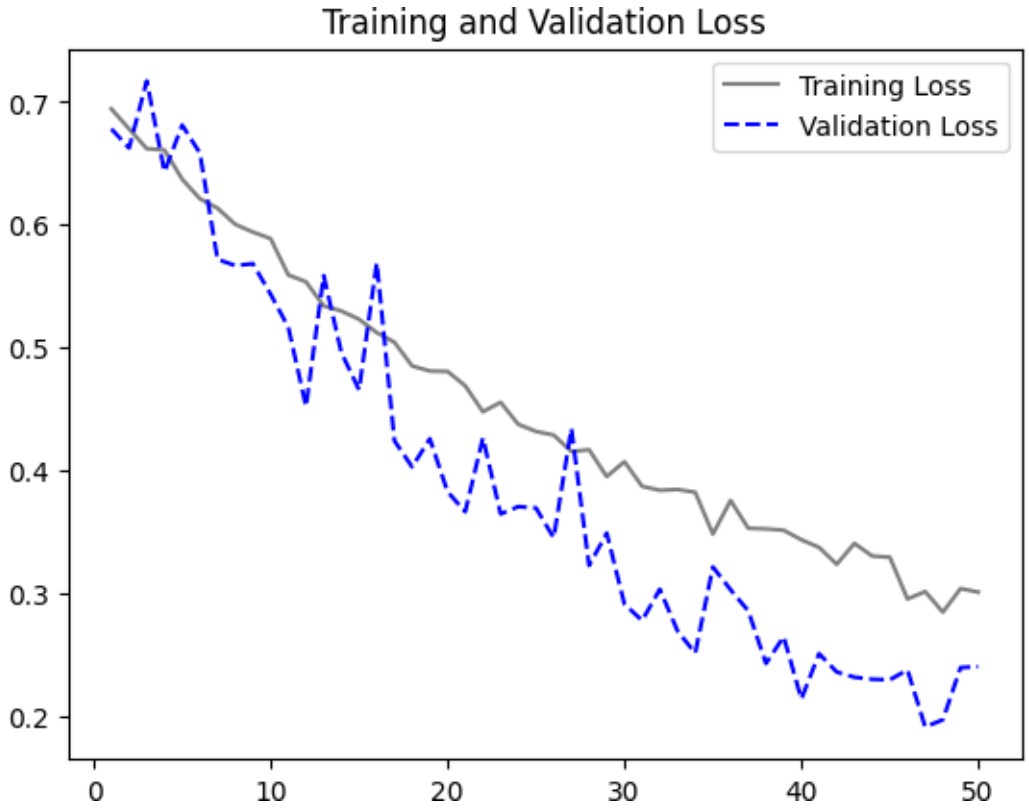
36/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3759 - accuracy: 0.8420 - val_loss: 0.3033 - val_accuracy: 0.8860Epoch
37/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3533 - accuracy: 0.8403 - val_loss: 0.2861 - val_accuracy: 0.8620Epoch
38/50
94/94 [==============================] - 2s 22ms/step - loss: 0.3528 - accuracy: 0.8490 - val_loss: 0.2434 - val_accuracy: 0.8980

```
Epoch 39/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3517 - accuracy:
0.8440 - val_loss: 0.2651 - val_accuracy: 0.8840Epoch
40/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3440 - accuracy:
0.8543 - val_loss: 0.2144 - val_accuracy: 0.9140Epoch
41/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3378 - accuracy:
0.8577 - val_loss: 0.2513 - val_accuracy: 0.9000Epoch
42/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3240 - accuracy:
0.8533 - val_loss: 0.2364 - val_accuracy: 0.8920Epoch
43/50
94/94 [==============================] - 2s 20ms/step - loss: 0.3410 - accuracy:
0.8540 - val_loss: 0.2322 - val_accuracy: 0.8960Epoch
44/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3307 - accuracy:
0.8570 - val_loss: 0.2306 - val_accuracy: 0.8980Epoch
45/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3298 - accuracy:
0.8537 - val_loss: 0.2301 - val_accuracy: 0.9040Epoch
46/50
94/94 [==============================] - 2s 20ms/step - loss: 0.2958 - accuracy:
0.8693 - val_loss: 0.2381 - val_accuracy: 0.9020Epoch
47/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3019 - accuracy:
0.8657 - val_loss: 0.1918 - val_accuracy: 0.9240Epoch
48/50
94/94 [==============================] - 2s 20ms/step - loss: 0.2851 - accuracy:
0.8730 - val_loss: 0.1976 - val_accuracy: 0.9120Epoch
49/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3043 - accuracy:
0.8707 - val_loss: 0.2400 - val_accuracy: 0.9040Epoch
50/50
94/94 [==============================] - 2s 21ms/step - loss: 0.3014 - accuracy:
0.8670 - val_loss: 0.2408 - val_accuracy: 0.8860
```

```
accuracy = Model_9.history["accuracy"]
val_accuracy = Model_9.history["val_accuracy"]

loss = Model_9.history["loss"]
val_loss = Model_9.history["val_loss"]

epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, color="grey", label="Training Accuracy")
```

```
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```






```
best_model = keras.models.load_model("model9.keras")
Model9_Results = best_model.evaluate(test_dataset)
print(f"Loss: {Model9_Results[0]:.3f}")
print(f"Accuracy: {Model9_Results[1]:.3f}")
```

16/16 [==============================] - 0s 9ms/step - loss: 0.4086 - accuracy:
0.8320
Loss: 0.409
Accuracy: 0.832

Let's see which of the models have best performance when the training sample was set to 3000. Note: Here models 8 and 9 were trained differently with strides being used with maxpooling andstrides being used with maxpooling and padding turned on.

Model 6: trides Operation with Padding being "Same" ,filters from 32 to 512, 5 Input Layers, droput rate of 0.5, Training Sample - 3000

Model 7: MaxPooling Operation,filters from 32 to 512, 5 Input Layers, droput rate of 0.5, Training Sample - 3000

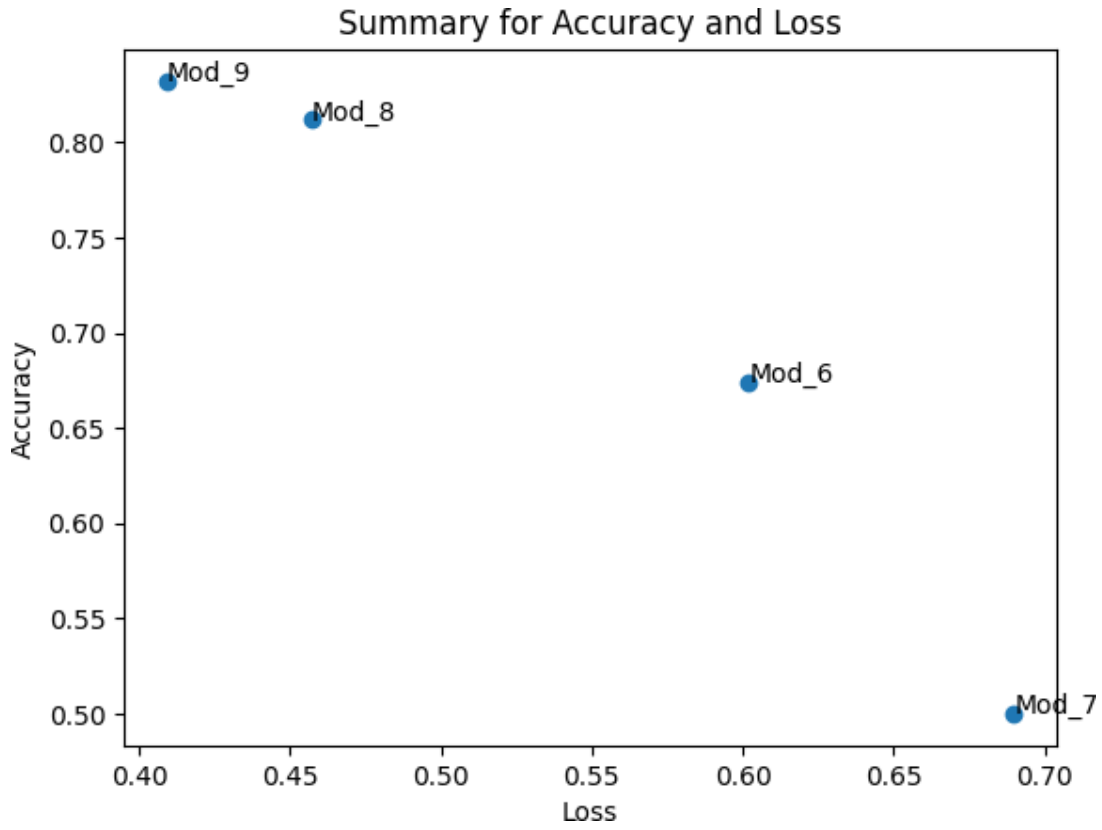
Model 8: MaxPooling + Strides of Step-Size 2,filters from 32 to 512, 5 Input Layers, droput rate of 0.5, Training Sample - 3000

Model 9: MaxPooling + Strides of Step-Size 2 with Padding turned on,filters from 32 to 512, 5

Input Layers, droput rate of 0.5, Training Sample - 3000

```
Model_7 = (0.69,0.500)
Model_8 = (0.457,0.812)
Model_9 = (0.409,0.832)
```

```
Models_3 = ("Mod_6","Mod_7","Mod_8","Mod_9")
Loss_3 = (Model_6[0],Model_7[0],Model_8[0],Model_9[0])
Accuracy_3 = (Model_6[1],Model_7[1],Model_8[1],Model_9[1])

fig, ax = plt.subplots()
ax.scatter(Loss_3,Accuracy_3)
for i, txt in enumerate(Models_3):
    ax.annotate(txt, (Loss_3[i],Accuracy_3[i] ))
plt.title("Summary for Accuracy and Loss")
plt.ylabel("Accuracy")
plt.xlabel("Loss")

plt.show()
```



Here we can clearly see that the model which was built with 5 layers using maxpooling along with

strides and padding on was giving the highest accuracy i.e. 83.2 % with least loss amoung the other2 models i.e. 40.9%.

Now, we are increasing the training sample to 5000 and building a model from scratch to check it'sperformance on the unseen data.

Training Sample - 5000

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_2")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=2500)
make_subset("validation", start_index=1000, end_index=1250)
make_subset("test", start_index=1500, end_index=1750)
```

```
train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

Found 5000 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.

```
for data_batch, labels_batch in train_dataset:
    print("data batch shape:", data_batch.shape)
    print("labels batch shape:", labels_batch.shape)
    break
```

data batch shape: (32, 180, 180, 3)labels batch shape: (32,)

```
data_augmentation_3 = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.15),
        layers.RandomZoom(0.25)
    ]
)
```

Model - 10 MaxPooling Operation with Increase in filters from 32 to 256 in 5 Input Layers with the data being used from the Augmented Images and a dropout rate of 0.5 (Training Sample - 5000)

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation_3(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

Model: "model_12"

conv2d_63 (Conv2D)	(None, 87, 87, 64)	18496
max_pooling2d_43 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_64 (Conv2D)	(None, 41, 41, 128)	73856
max_pooling2d_44 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_65 (Conv2D)	(None, 18, 18, 256)	295168
max_pooling2d_45 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_66 (Conv2D)	(None, 7, 7, 256)	590080
flatten_12 (Flatten)	(None, 12544)	0
dropout_10 (Dropout)	(None, 12544)	0
dense_12 (Dense)	(None, 1)	12545

Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0

```
# Compiling the Model
model.compile(loss= "binary_crossentropy",
              optimizer= "adam",
              metrics= ['accuracy'])


# Monitoring the best validation loss using Callbacks
callbacks = ModelCheckpoint(
              filepath = "model10.keras",
              save_best_only= True,
              monitor= "val_loss"
              )

# Model Fit
```

```
        callbacks= callbacks
        )
```

```
Epoch 1/50
157/157 [==============================] - 6s 21ms/step - loss: 0.6934 -
accuracy: 0.5148 - val_loss: 0.6919 - val_accuracy: 0.5000
Epoch 2/50
157/157 [==============================] - 3s 19ms/step - loss: 0.6908 -
accuracy: 0.5260 - val_loss: 0.6608 - val_accuracy: 0.6600
Epoch 3/50
157/157 [==============================] - 3s 19ms/step - loss: 0.6802 -
accuracy: 0.5706 - val_loss: 0.6825 - val_accuracy: 0.5000
Epoch 4/50
157/157 [==============================] - 3s 19ms/step - loss: 0.6852 -
accuracy: 0.5400 - val_loss: 0.6672 - val_accuracy: 0.5940
Epoch 5/50
157/157 [==============================] - 3s 19ms/step - loss: 0.6770 -
accuracy: 0.5682 - val_loss: 0.6569 - val_accuracy: 0.6400
Epoch 6/50
157/157 [==============================] - 3s 19ms/step - loss: 0.6565 -
accuracy: 0.6278 - val_loss: 0.6322 - val_accuracy: 0.6480
Epoch 7/50
157/157 [==============================] - 3s 19ms/step - loss: 0.6295 -
accuracy: 0.6506 - val_loss: 0.5962 - val_accuracy: 0.6860
Epoch 8/50
157/157 [==============================] - 3s 19ms/step - loss: 0.6257 -
accuracy: 0.6536 - val_loss: 0.6540 - val_accuracy: 0.5860
Epoch 9/50
157/157 [==============================] - 3s 20ms/step - loss: 0.6016 -
accuracy: 0.6782 - val_loss: 0.5675 - val_accuracy: 0.7100
Epoch 10/50
157/157 [==============================] - 3s 19ms/step - loss: 0.6087 -
accuracy: 0.6654 - val_loss: 0.5633 - val_accuracy: 0.7040
Epoch 11/50
157/157 [==============================] - 3s 19ms/step - loss: 0.5886 -
accuracy: 0.6954 - val_loss: 0.5526 - val_accuracy: 0.7260
Epoch 12/50
157/157 [==============================] - 3s 19ms/step - loss: 0.5777 -
accuracy: 0.7026 - val_loss: 0.5109 - val_accuracy: 0.7620
```

```
Epoch 13/50
157/157 [==============================] - 3s 19ms/step - loss: 0.5690 -
accuracy: 0.7028 - val_loss: 0.5130 - val_accuracy: 0.7500
Epoch 14/50
157/157 [==============================] - 3s 19ms/step - loss: 0.5576 -
accuracy: 0.7148 - val_loss: 0.5161 - val_accuracy: 0.7320
Epoch 15/50
157/157 [==============================] - 3s 19ms/step - loss: 0.5439 -
```

accuracy: 0.7250 - val_loss: 0.5266 - val_accuracy: 0.7340Epoch 16/50
157/157 [==============================] - 3s 19ms/step - loss: 0.5299 - accuracy: 0.7348 - val_loss: 0.4528 - val_accuracy: 0.7660
Epoch 17/50
157/157 [==============================] - 3s 19ms/step - loss: 0.5306 - accuracy: 0.7372 - val_loss: 0.4576 - val_accuracy: 0.7800
Epoch 18/50
157/157 [==============================] - 3s 19ms/step - loss: 0.5006 - accuracy: 0.7508 - val_loss: 0.4182 - val_accuracy: 0.7980
Epoch 19/50
157/157 [==============================] - 3s 20ms/step - loss: 0.4983 - accuracy: 0.7648 - val_loss: 0.4033 - val_accuracy: 0.8200
Epoch 20/50
157/157 [==============================] - 3s 20ms/step - loss: 0.4815 - accuracy: 0.7630 - val_loss: 0.4107 - val_accuracy: 0.8080
Epoch 21/50
157/157 [==============================] - 3s 19ms/step - loss: 0.4894 - accuracy: 0.7612 - val_loss: 0.4176 - val_accuracy: 0.8120
Epoch 22/50
157/157 [==============================] - 3s 19ms/step - loss: 0.4758 - accuracy: 0.7740 - val_loss: 0.3842 - val_accuracy: 0.8260
Epoch 23/50
157/157 [==============================] - 3s 19ms/step - loss: 0.4632 - accuracy: 0.7734 - val_loss: 0.3743 - val_accuracy: 0.8140
Epoch 24/50
157/157 [==============================] - 3s 20ms/step - loss: 0.4697 - accuracy: 0.7750 - val_loss: 0.3643 - val_accuracy: 0.8460
Epoch 25/50
157/157 [==============================] - 3s 19ms/step - loss: 0.4475 - accuracy: 0.7838 - val_loss: 0.3684 - val_accuracy: 0.8340
Epoch 26/50
157/157 [==============================] - 3s 19ms/step - loss: 0.4460 - accuracy: 0.7862 - val_loss: 0.3618 - val_accuracy: 0.8500
Epoch 27/50
157/157 [==============================] - 3s 19ms/step - loss: 0.4331 - accuracy: 0.8036 - val_loss: 0.3791 - val_accuracy: 0.8400
Epoch 28/50
157/157 [==============================] - 3s 19ms/step - loss: 0.4230 -

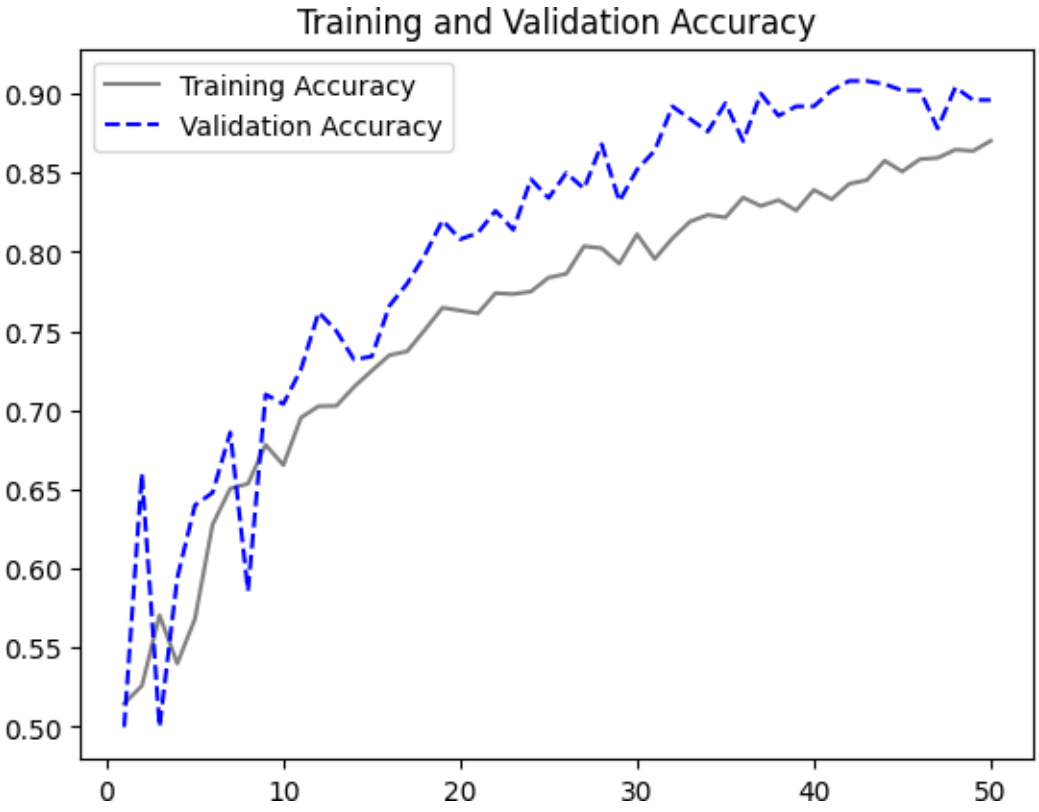
accuracy: 0.8024 - val_loss: 0.3248 - val_accuracy: 0.8680
Epoch 29/50
157/157 [==============================] - 3s 20ms/step - loss: 0.4312 -
accuracy: 0.7926 - val_loss: 0.3394 - val_accuracy: 0.8320
Epoch 30/50
157/157 [==============================] - 3s 19ms/step - loss: 0.4144 -
accuracy: 0.8112 - val_loss: 0.3272 - val_accuracy: 0.8520
Epoch 31/50
157/157 [==============================] - 3s 19ms/step - loss: 0.4368 -

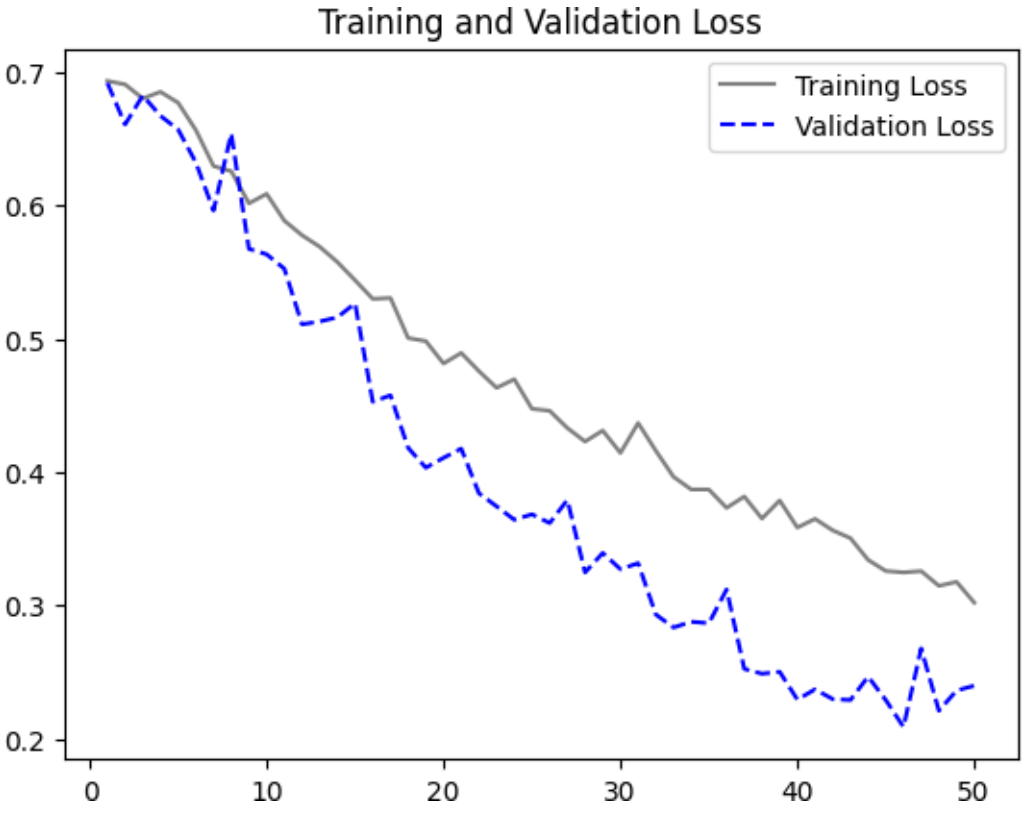
accuracy: 0.7956 - val_loss: 0.3317 - val_accuracy: 0.8640Epoch 32/50
157/157 [==============================] - 3s 19ms/step - loss: 0.4161 - accuracy: 0.8086 - val_loss: 0.2932 - val_accuracy: 0.8920
Epoch 33/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3965 - accuracy: 0.8192 - val_loss: 0.2835 - val_accuracy: 0.8840
Epoch 34/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3870 - accuracy: 0.8234 - val_loss: 0.2878 - val_accuracy: 0.8760
Epoch 35/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3870 - accuracy: 0.8218 - val_loss: 0.2867 - val_accuracy: 0.8940
Epoch 36/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3732 - accuracy: 0.8344 - val_loss: 0.3120 - val_accuracy: 0.8700
Epoch 37/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3817 - accuracy: 0.8290 - val_loss: 0.2525 - val_accuracy: 0.9000
Epoch 38/50
157/157 [==============================] - 3s 20ms/step - loss: 0.3652 - accuracy: 0.8328 - val_loss: 0.2489 - val_accuracy: 0.8860
Epoch 39/50
157/157 [==============================] - 3s 20ms/step - loss: 0.3788 - accuracy: 0.8262 - val_loss: 0.2503 - val_accuracy: 0.8920
Epoch 40/50
157/157 [==============================] - 3s 20ms/step - loss: 0.3585 - accuracy: 0.8392 - val_loss: 0.2294 - val_accuracy: 0.8920
Epoch 41/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3650 - accuracy: 0.8332 - val_loss: 0.2371 - val_accuracy: 0.9020
Epoch 42/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3563 - accuracy: 0.8430 - val_loss: 0.2298 - val_accuracy: 0.9080
Epoch 43/50
157/157 [==============================] - 3s 20ms/step - loss: 0.3505 - accuracy: 0.8454 - val_loss: 0.2291 - val_accuracy: 0.9080
Epoch 44/50
157/157 [==============================] - 3s 20ms/step - loss: 0.3342 -

accuracy: 0.8576 - val_loss: 0.2467 - val_accuracy: 0.9060
Epoch 45/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3259 -
accuracy: 0.8508 - val_loss: 0.2290 - val_accuracy: 0.9020
Epoch 46/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3248 -
accuracy: 0.8586 - val_loss: 0.2089 - val_accuracy: 0.9020
Epoch 47/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3258 -

```
accuracy: 0.8594 - val_loss: 0.2678 - val_accuracy: 0.8780Epoch
48/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3148 -
accuracy: 0.8646 - val_loss: 0.2211 - val_accuracy: 0.9040Epoch
49/50
157/157 [==============================] - 3s 20ms/step - loss: 0.3177 -
accuracy: 0.8638 - val_loss: 0.2363 - val_accuracy: 0.8960Epoch
50/50
157/157 [==============================] - 3s 19ms/step - loss: 0.3021 -
accuracy: 0.8702 - val_loss: 0.2398 - val_accuracy: 0.8960
```

```
accuracy = Model_10.history["accuracy"]
val_accuracy = Model_10.history["val_accuracy"]

loss = Model_10.history["loss"]
val_loss = Model_10.history["val_loss"]

epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, color="grey", label="Training Accuracy")
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed",␣
↪label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
10
20
30
40
50


Training and Validation Loss
Training Loss
Validation Loss
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50

```
best_model = keras.models.load_model("model10.keras")
Model10_Results = best_model.evaluate(test_dataset)
print(f"Loss: {Model10_Results[0]:.3f}")
print(f"Accuracy: {Model10_Results[1]:.3f}")
```

```
16/16 [==============================] - 0s 10ms/step - loss: 0.2834 - accuracy:
0.8840
Loss: 0.283
Accuracy: 0.884
```

Summary for Question 3,

We built four models, three of which were trained using a 3000 sample size. The best-performing model has an accuracy of 83.2%. Interestingly, the accuracy increased to 88.4% when we increased the training sample to 5000. Thus, we conclude that a training sample size of 5000 significantly improves the model's performance. As for the likely cause of the validation loss being less than the training loss, the split approach that was used is probably a factor.

Here, the validation and test sets are fixed at 500 apiece, but the training sample is almost five thousand times larger. Furthermore, it is crucial to recognize that regularizations like dropout or L1 and L2 regularizers are important during training and contribute to the training loss calculation. On the other hand, these regularizers are turned off during the validation or test phase, which could result in a smaller loss than the training loss.

1. *Repeat Steps 1-3, but now using a pretrained network. The sample sizes you use in Steps 2 and3 for the pretrained network may be the same or different from those using the network where you trained from scratch. Again, use any and all optimization techniques to get best performance*

Leveraging a Pre-Trained Model - VGG16
VGG - Model 1 (1000 Training Samples)

```
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step

```
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_14 (InputLayer)	[(None, 180, 180, 3)]	0
block1_conv1 (Conv2D)	(None, 180, 180, 64)	1792
block1_conv2 (Conv2D)	(None, 180, 180, 64)	36928
block1_pool (MaxPooling2D)	(None, 90, 90, 64)	0
block2_conv1 (Conv2D)	(None, 90, 90, 128)	73856
block2_conv2 (Conv2D)	(None, 90, 90, 128)	147584
block2_pool (MaxPooling2D)	(None, 45, 45, 128)	0
block3_conv1 (Conv2D)	(None, 45, 45, 256)	295168
block3_conv2 (Conv2D)	(None, 45, 45, 256)	590080
block3_conv3 (Conv2D)	(None, 45, 45, 256)	590080
block3_pool (MaxPooling2D)	(None, 22, 22, 256)	0
block4_conv1 (Conv2D)	(None, 22, 22, 512)	1180160
block4_conv2 (Conv2D)	(None, 22, 22, 512)	2359808

```
block4_conv3 (Conv2D)        (None, 22, 22, 512)       2359808
block4_pool (MaxPooling2D)   (None, 11, 11, 512)       0
block5_conv1 (Conv2D)        (None, 11, 11, 512)       2359808
block5_conv2 (Conv2D)        (None, 11, 11, 512)       2359808
block5_conv3 (Conv2D)        (None, 11, 11, 512)       2359808
block5_pool (MaxPooling2D)   (None, 5, 5, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

```python
#extracting VGG 16 features and Labels
def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = conv_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

train_features, train_labels =get_features_and_labels(train_dataset)
val_features, val_labels = get_features_and_labels(validation_dataset)
test_features, test_labels =get_features_and_labels(test_dataset)
```

```
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 24ms/step
```

```
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
```

```
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
```

```
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
```

```
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
```

```
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 27ms/step
```

```
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 35ms/step
```

```
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 1s 515ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
```

```
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 1s 689ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
```

```
train_features.shape
```

```
(5000, 5, 5, 512)
```

VGG - Model 1 Dense Layer with 256 Nodes and Droput Rate of 0.5 and optimizer being rmsprop with the Original Images

```
# Defining and Training the densely connected classifier
# The last dense stacked layer and the classifier
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
```

```
# Compiling the Model
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])


# Using CallBacks to monitor the best val_loss
callbacks = ModelCheckpoint(
             filepath="vgg_model1.keras",
             save_best_only=True,
             monitor="val_loss")


# Model Fit
VGG_Model_1 = model.fit(
              train_features, train_labels,
              epochs= 30,
              validation_data= (val_features, val_labels),
              callbacks= callbacks)
```

```
Epoch 1/30
157/157 [==============================] - 2s 5ms/step - loss: 11.3170 -
accuracy: 0.9444 - val_loss: 0.5029 - val_accuracy: 0.9900
Epoch 2/30
157/157 [==============================] - 1s 4ms/step - loss: 2.9963 -
accuracy: 0.9810 - val_loss: 0.1466 - val_accuracy: 0.9980Epoch
3/30
157/157 [==============================] - 1s 3ms/step - loss: 1.2910 -
accuracy: 0.9908 - val_loss: 2.4968 - val_accuracy: 0.9800Epoch
4/30
157/157 [==============================] - 1s 4ms/step - loss: 1.5284 -
accuracy: 0.9888 - val_loss: 1.9570e-08 - val_accuracy: 1.0000Epoch
5/30
157/157 [==============================] - 1s 4ms/step - loss: 0.9568 -
accuracy: 0.9922 - val_loss: 1.6928e-15 - val_accuracy: 1.0000Epoch
6/30
157/157 [==============================] - 1s 4ms/step - loss: 0.9920 -
accuracy: 0.9924 - val_loss: 1.5836e-21 - val_accuracy: 1.0000Epoch
7/30
157/157 [==============================] - 1s 4ms/step - loss: 0.8880 -
```

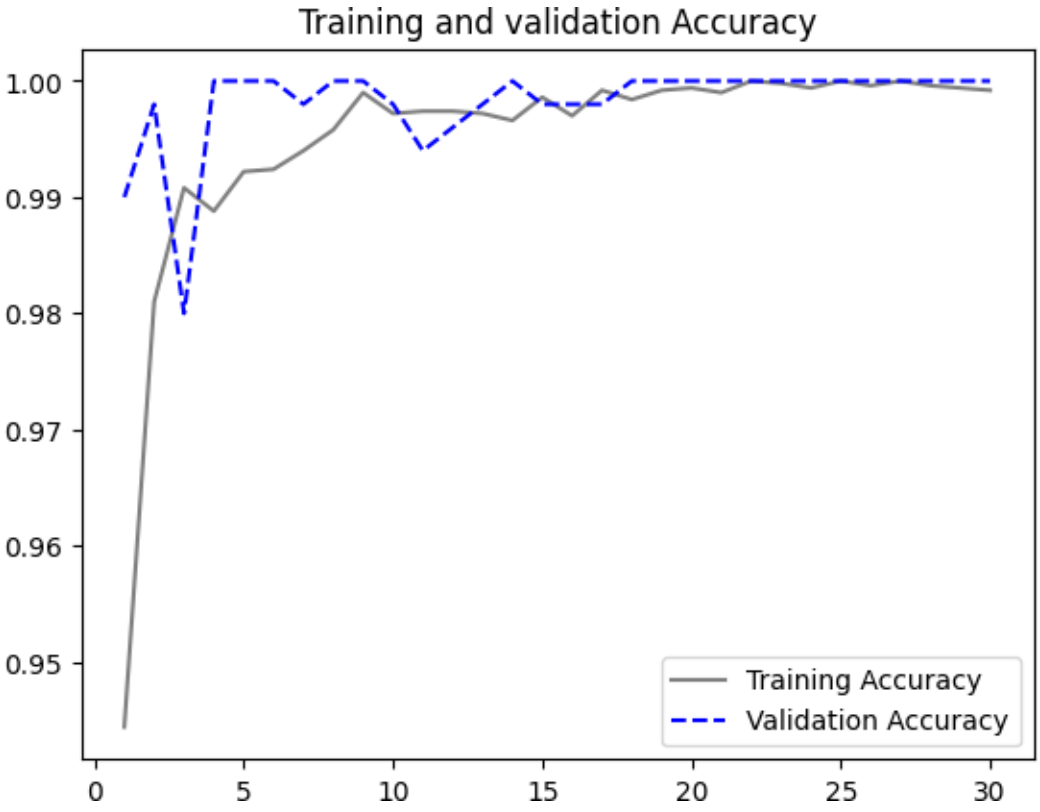
accuracy: 0.9940 - val_loss: 0.0336 - val_accuracy: 0.9980Epoch 8/30 0.3989 - 157/157 [==============================] - 1s 4ms/step - loss: accuracy: 0.9958 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 Epoch 9/30

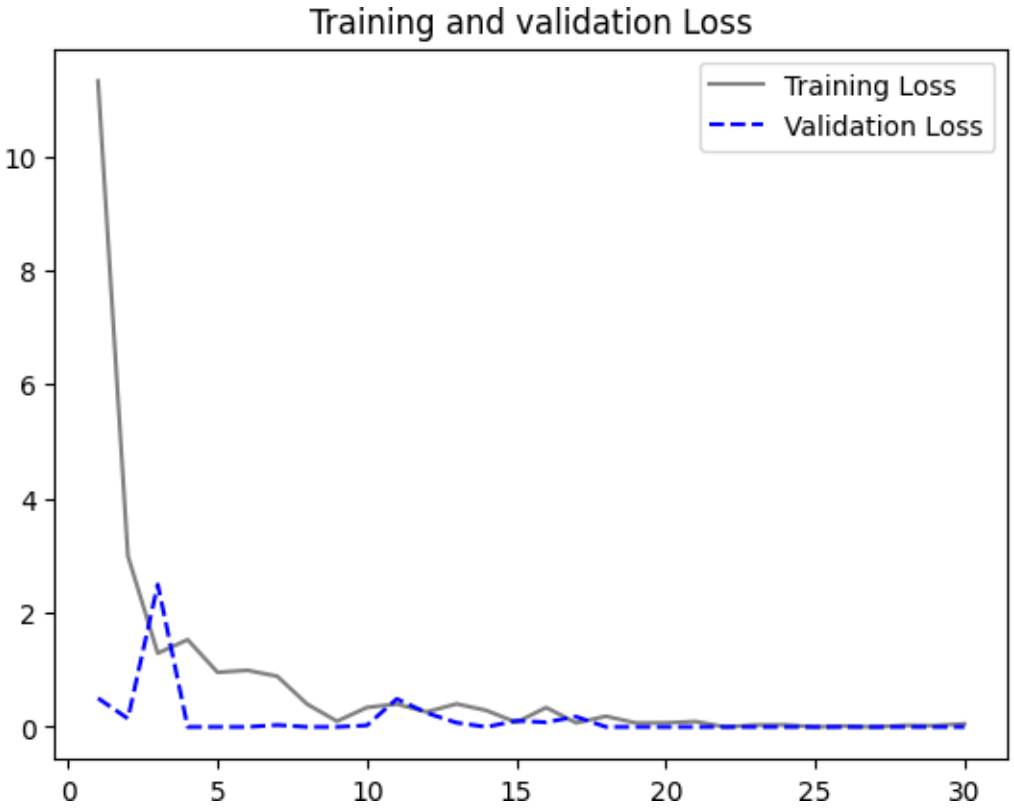
157/157 [==============================] - 1s 4ms/step - loss: 0.0996 - accuracy: 0.9990 - val_loss: 2.0550e-29 - val_accuracy: 1.0000
Epoch 10/30
157/157 [==============================] - 1s 4ms/step - loss: 0.3411 - accuracy: 0.9972 - val_loss: 0.0270 - val_accuracy: 0.9980
Epoch 11/30
157/157 [==============================] - 1s 4ms/step - loss: 0.4010 - accuracy: 0.9974 - val_loss: 0.4906 - val_accuracy: 0.9940
Epoch 12/30
157/157 [==============================] - 1s 4ms/step - loss: 0.2621 - accuracy: 0.9974 - val_loss: 0.2463 - val_accuracy: 0.9960
Epoch 13/30
157/157 [==============================] - 1s 3ms/step - loss: 0.4041 - accuracy: 0.9972 - val_loss: 0.0738 - val_accuracy: 0.9980
Epoch 14/30
157/157 [==============================] - 1s 3ms/step - loss: 0.2880 - accuracy: 0.9966 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 15/30
157/157 [==============================] - 1s 3ms/step - loss: 0.0803 - accuracy: 0.9986 - val_loss: 0.1080 - val_accuracy: 0.9980
Epoch 16/30
157/157 [==============================] - 1s 4ms/step - loss: 0.3397 - accuracy: 0.9970 - val_loss: 0.0819 - val_accuracy: 0.9980
Epoch 17/30
157/157 [==============================] - 1s 3ms/step - loss: 0.0713 - accuracy: 0.9992 - val_loss: 0.1798 - val_accuracy: 0.9980
Epoch 18/30
157/157 [==============================] - 1s 3ms/step - loss: 0.1872 - accuracy: 0.9984 - val_loss: 0.0000e+00 - val_accuracy: 1.0000Epoch 19/30
157/157 [==============================] - 1s 3ms/step - loss: 0.0689 - accuracy: 0.9992 - val_loss: 0.0000e+00 - val_accuracy: 1.0000Epoch 20/30
157/157 [==============================] - 1s 3ms/step - loss: 0.0711 - accuracy: 0.9994 - val_loss: 7.1133e-27 - val_accuracy: 1.0000Epoch 21/30
157/157 [==============================] - 1s 3ms/step - loss: 0.0946 - accuracy: 0.9990 - val_loss: 0.0000e+00 - val_accuracy: 1.0000Epoch 22/30

157/157 [==============================] - 1s 3ms/step - loss: 4.7128e-11 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000Epoch 23/30
157/157 [==============================] - 1s 3ms/step - loss: 0.0328 - accuracy: 0.9998 - val_loss: 6.0957e-27 - val_accuracy: 1.0000Epoch 24/30
157/157 [==============================] - 1s 3ms/step - loss: 0.0362 - accuracy: 0.9994 - val_loss: 0.0000e+00 - val_accuracy: 1.0000Epoch 25/30

```
157/157 [==============================] - 1s 3ms/step - loss: 2.6143e-07 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000Epoch
26/30
157/157 [==============================] - 1s 4ms/step - loss: 0.0141 -
accuracy: 0.9996 - val_loss: 0.0000e+00 - val_accuracy: 1.0000Epoch
27/30
157/157 [==============================] - 1s 3ms/step - loss: 2.9327e-18 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000Epoch
28/30
157/157 [==============================] - 1s 3ms/step - loss: 0.0268 -
accuracy: 0.9996 - val_loss: 0.0000e+00 - val_accuracy: 1.0000Epoch
29/30
157/157 [==============================] - 1s 3ms/step - loss: 0.0217 -
accuracy: 0.9994 - val_loss: 0.0000e+00 - val_accuracy: 1.0000Epoch
30/30
157/157 [==============================] - 1s 3ms/step - loss: 0.0547 -
accuracy: 0.9992 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

```
acc = VGG_Model_1.history["accuracy"]
val_acc = VGG_Model_1.history["val_accuracy"]

loss = VGG_Model_1.history["loss"]
val_loss = VGG_Model_1.history["val_loss"]

epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, color="grey", label="Training Accuracy")
plt.plot(epochs, val_acc, color="blue", linestyle="dashed", label="Validation␣
↪Accuracy")
plt.title("Training and validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and validation Loss")
plt.legend()
plt.show()
```


Training and validation Accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0
5
10
15
20
25
30
Training Accuracy
Validation Accuracy


Training and validation Loss
10
8
6
4
2
0
0
5
10
15
20
25
30
Training Loss
Validation Loss

```
best_model = keras.models.load_model("vgg_model1.keras")
VGG_Model_1_Results = best_model.evaluate(test_features,test_labels)
print(f'Loss: {VGG_Model_1_Results[0]:.3f}')
print(f'Accuracy: {VGG_Model_1_Results[1]:.3f}')
```

```
16/16 [==============================] - 0s 3ms/step - loss: 0.0000e+00 -
accuracy: 1.0000
Loss: 0.000
Accuracy: 1.000
```

VGG - Model 2 (1000 Training Samples)

Only the densely linked networks and the classifier are permitted to modify their weights while the pre-trained model is set to maintain its current weights throughout training.

This method helps avoid overfitting because the pre-trained model doesn't change, giving the model a solid base. Furthermore, freezing the pre-trained model training might be very helpful when working with limited training data and processing resources.

We can output the list of trainable weights both before and after freezing the pre-trained model to show the effect of this setting.

```
# Before Freezing
conv_base.trainable = True
print("This is the number of trainable weights "
      "before freezing the conv base:", len(conv_base.trainable_weights))
```

This is the number of trainable weights before freezing the conv base: 26

```
# After Freezing
conv_base.trainable = False
print("This is the number of trainable weights "
      "after freezing the conv base:", len(conv_base.trainable_weights))
```

This is the number of trainable weights after freezing the conv base: 0

```
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_14 (InputLayer)	[(None, 180, 180, 3)]	0
block1_conv1 (Conv2D)	(None, 180, 180, 64)	1792
block1_conv2 (Conv2D)	(None, 180, 180, 64)	36928
block1_pool (MaxPooling2D)	(None, 90, 90, 64)	0
block2_conv1 (Conv2D)	(None, 90, 90, 128)	73856
block2_conv2 (Conv2D)	(None, 90, 90, 128)	147584
block2_pool (MaxPooling2D)	(None, 45, 45, 128)	0
block3_conv1 (Conv2D)	(None, 45, 45, 256)	295168
block3_conv2 (Conv2D)	(None, 45, 45, 256)	590080
block3_conv3 (Conv2D)	(None, 45, 45, 256)	590080
block3_pool (MaxPooling2D)	(None, 22, 22, 256)	0
block4_conv1 (Conv2D)	(None, 22, 22, 512)	1180160
block4_conv2 (Conv2D)	(None, 22, 22, 512)	2359808
block4_conv3 (Conv2D)	(None, 22, 22, 512)	2359808

```
 block4_pool (MaxPooling2D)   (None, 11, 11, 512)        0

 block5_conv1 (Conv2D)        (None, 11, 11, 512)        2359808

 block5_conv2 (Conv2D)        (None, 11, 11, 512)        2359808

 block5_conv3 (Conv2D)        (None, 11, 11, 512)        2359808

 block5_pool (MaxPooling2D)   (None, 5, 5, 512)          0

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

VGG - Model 2 Dense Layer with 256 Nodes and Droput Rate of 0.5 and optimizer being rmsprop with the Augmented Images

```
# Data Augmentation -Adding a data augmentation stage to provide augmented training samples and a classifier to the convolutional base
data_augmentation = keras.Sequential([
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

# Adding the Classifier and Dense Network inputs =
keras.Input(shape=(180, 180, 3)) x =
data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)x =
conv_base(x)
x = layers.Flatten()(x)x =
layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)

outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs, outputs)
```

```
# Compiling the Model
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])

# Using CallBacks to monitor the best val_loss
```

```
callbacks = ModelCheckpoint(
             filepath= "vgg_model2.keras",
             save_best_only= True,
             monitor= "val_loss")

# Model Fit
VGG_Model_2 = model.fit(
              train_dataset,
              epochs= 30,
              validation_data=validation_dataset,
              callbacks=callbacks)
```

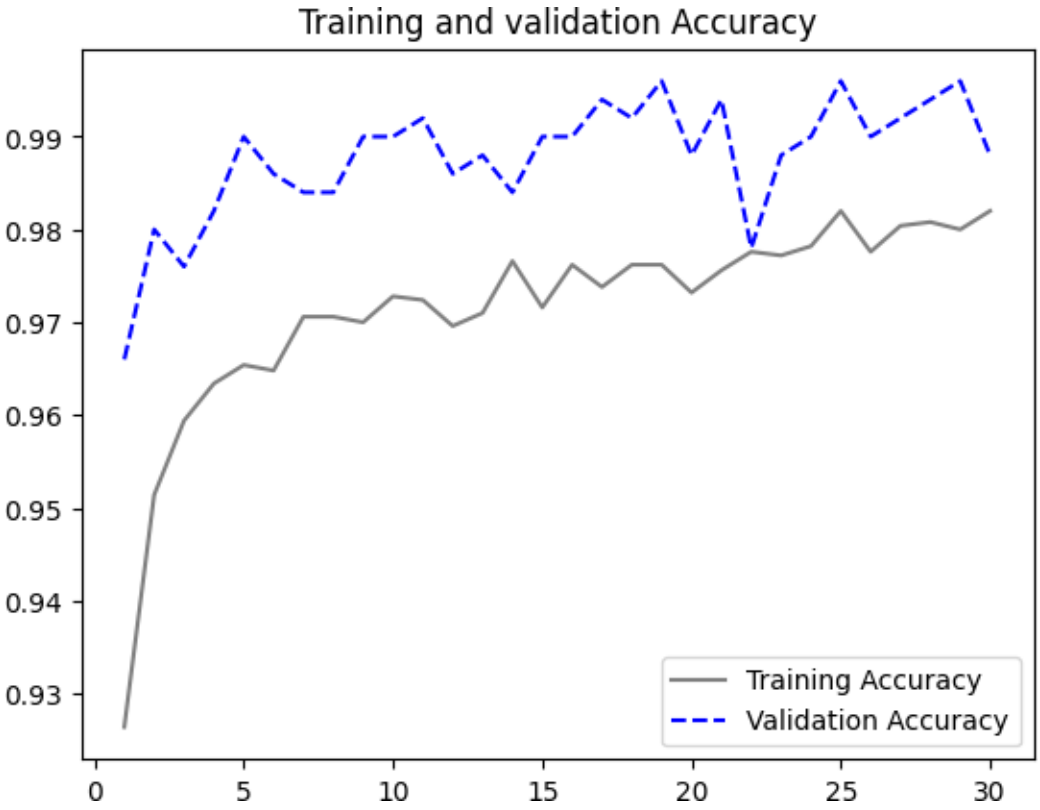
```
Epoch 1/30
157/157 [==============================] - 8s 40ms/step - loss: 11.6774 -
accuracy: 0.9264 - val_loss: 5.8878 - val_accuracy: 0.9660Epoch
2/30
157/157 [==============================] - 6s 39ms/step - loss: 6.1833 -
accuracy: 0.9514 - val_loss: 3.8140 - val_accuracy: 0.9800
Epoch 3/30
157/157 [==============================] - 6s 39ms/step - loss: 4.7273 -
accuracy: 0.9594 - val_loss: 2.3475 - val_accuracy: 0.9760
Epoch 4/30
157/157 [==============================] - 6s 39ms/step - loss: 3.2261 -
accuracy: 0.9634 - val_loss: 2.0974 - val_accuracy: 0.9820
Epoch 5/30
157/157 [==============================] - 6s 39ms/step - loss: 3.1476 -
accuracy: 0.9654 - val_loss: 0.5457 - val_accuracy: 0.9900
Epoch 6/30
157/157 [==============================] - 6s 37ms/step - loss: 2.1593 -
accuracy: 0.9648 - val_loss: 0.5819 - val_accuracy: 0.9860
Epoch 7/30
157/157 [==============================] - 6s 37ms/step - loss: 1.6923 -
accuracy: 0.9706 - val_loss: 1.0481 - val_accuracy: 0.9840
Epoch 8/30
157/157 [==============================] - 6s 38ms/step - loss: 1.6746 -
accuracy: 0.9706 - val_loss: 0.7510 - val_accuracy: 0.9840
```

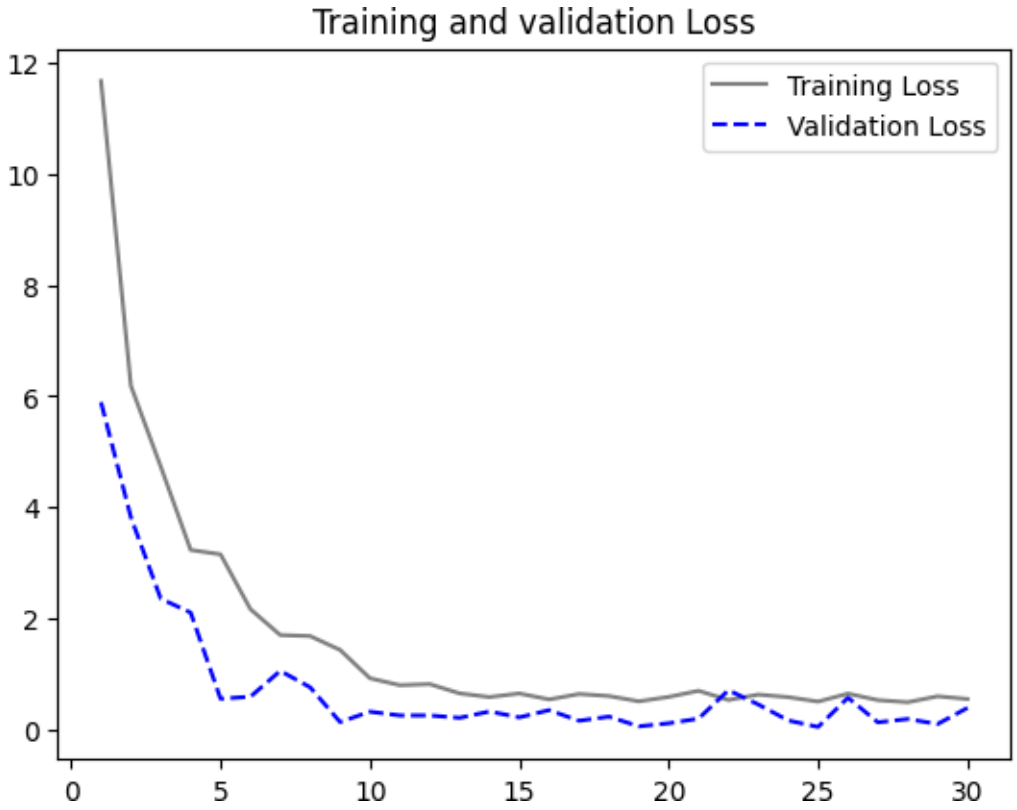
accuracy: 0.9696 - val_loss: 0.2444 - val_accuracy: 0.9860Epoch 13/30 loss: 0.6420 -
157/157 [==============================] - 6s 38ms/step -
accuracy: 0.9710 - val_loss: 0.2012 - val_accuracy: 0.9880
Epoch 14/30
157/157 [==============================] - 6s 38ms/step - loss: 0.5753 -
accuracy: 0.9766 - val_loss: 0.3152 - val_accuracy: 0.9840
Epoch 15/30
157/157 [==============================] - 6s 38ms/step - loss: 0.6403 -
accuracy: 0.9716 - val_loss: 0.2098 - val_accuracy: 0.9900
Epoch 16/30
157/157 [==============================] - 6s 38ms/step - loss: 0.5326 -
accuracy: 0.9762 - val_loss: 0.3397 - val_accuracy: 0.9900
Epoch 17/30
157/157 [==============================] - 6s 38ms/step - loss: 0.6307 -
accuracy: 0.9738 - val_loss: 0.1483 - val_accuracy: 0.9940
Epoch 18/30
157/157 [==============================] - 6s 38ms/step - loss: 0.5940 -
accuracy: 0.9762 - val_loss: 0.2225 - val_accuracy: 0.9920
Epoch 19/30
157/157 [==============================] - 6s 39ms/step - loss: 0.4982 -
accuracy: 0.9762 - val_loss: 0.0480 - val_accuracy: 0.9960
Epoch 20/30
157/157 [==============================] - 6s 38ms/step - loss: 0.5784 -
accuracy: 0.9732 - val_loss: 0.1027 - val_accuracy: 0.9880
Epoch 21/30
157/157 [==============================] - 6s 38ms/step - loss: 0.6884 -
accuracy: 0.9756 - val_loss: 0.1871 - val_accuracy: 0.9940
Epoch 22/30
157/157 [==============================] - 6s 38ms/step - loss: 0.5225 -
accuracy: 0.9776 - val_loss: 0.6998 - val_accuracy: 0.9780
Epoch 23/30
157/157 [==============================] - 6s 38ms/step - loss: 0.6201 -
accuracy: 0.9772 - val_loss: 0.4413 - val_accuracy: 0.9880
Epoch 24/30
157/157 [==============================] - 6s 38ms/step - loss: 0.5742 -
accuracy: 0.9782 - val_loss: 0.1537 - val_accuracy: 0.9900
Epoch 25/30
157/157 [==============================] - 6s 39ms/step - loss: 0.4939 -

accuracy: 0.9820 - val_loss: 0.0371 - val_accuracy: 0.9960
Epoch 26/30
157/157 [==============================] - 6s 38ms/step - loss: 0.6373 -
accuracy: 0.9776 - val_loss: 0.5649 - val_accuracy: 0.9900
Epoch 27/30
157/157 [==============================] - 6s 38ms/step - loss: 0.5205 -
accuracy: 0.9804 - val_loss: 0.1203 - val_accuracy: 0.9920
Epoch 28/30
157/157 [==============================] - 6s 38ms/step - loss: 0.4823 -

```
accuracy: 0.9808 - val_loss: 0.1801 - val_accuracy: 0.9940Epoch
29/30
157/157 [==============================] - 6s 38ms/step - loss: 0.5895 -
accuracy: 0.9800 - val_loss: 0.0893 - val_accuracy: 0.9960Epoch
30/30
157/157 [==============================] - 6s 38ms/step - loss: 0.5403 -
accuracy: 0.9820 - val_loss: 0.3804 - val_accuracy: 0.9880
```

```
acc = VGG_Model_2.history["accuracy"]
val_acc = VGG_Model_2.history["val_accuracy"]

loss = VGG_Model_2.history["loss"]
val_loss = VGG_Model_2.history["val_loss"]

epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, color="grey", label="Training Accuracy")
plt.plot(epochs, val_acc, color="blue", linestyle="dashed", label="Validation␣
↪Accuracy")
plt.title("Training and validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and validation Loss")
plt.legend()
plt.show()
```


Training and validation Accuracy
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0
5
10
15
20
25
30
Training Accuracy
Validation Accuracy




```
best_model = keras.models.load_model("vgg_model2.keras")
VGG_Model_2_Results= best_model.evaluate(test_dataset)
print(f"Loss: {VGG_Model_2_Results[0]:.3f}")
print(f"Accuracy: {VGG_Model_2_Results[1]:.3f}")
```

16/16 [==============================] - 1s 31ms/step - loss: 0.0260 - accuracy:
0.9980

Loss: 0.026

Accuracy: 0.998

Fine Tuning the VGG_Model_2

```
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```

It is important to realize that pre-trained networks are trained to handle a variety of use cases and classifications, and are not only employed for solitary picture classification tasks. The network's first layers are good at gathering broad data, while its later levels usually focus on extracting features that are unique to the issue at hand. By choosing to freeze the first few layers, we may effectively avoid overfitting and allow the model to incorporate more complex information related to our particular categorization problem. The model is encouraged to concentrate on understanding the subtle nuances of the target classification problem by using this strategic approach.

```
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])


callbacks = ModelCheckpoint(
              filepath="fine_tuning_vgg_model2.keras",
              save_best_only=True,
              monitor="val_loss")


FineTuned_VGG_Model_2 = model.fit(
                          train_dataset,
                          epochs=30,
                          validation_data=validation_dataset,
                          callbacks=callbacks)
```

```
Epoch 1/30
157/157 [==============================] - 9s 44ms/step - loss: 0.5814 -
accuracy: 0.9812 - val_loss: 0.1324 - val_accuracy: 0.9920
Epoch 2/30
157/157 [==============================] - 7s 41ms/step - loss: 0.3211 -
accuracy: 0.9838 - val_loss: 0.2911 - val_accuracy: 0.9900
Epoch 3/30
157/157 [==============================] - 7s 41ms/step - loss: 0.4245 -
accuracy: 0.9828 - val_loss: 0.1409 - val_accuracy: 0.9960
Epoch 4/30
157/157 [==============================] - 7s 41ms/step - loss: 0.2823 -
accuracy: 0.9856 - val_loss: 0.1580 - val_accuracy: 0.9960
Epoch 5/30
157/157 [==============================] - 7s 42ms/step - loss: 0.2966 -
accuracy: 0.9868 - val_loss: 0.0682 - val_accuracy: 0.9980
Epoch 6/30
157/157 [==============================] - 7s 41ms/step - loss: 0.2301 -
accuracy: 0.9866 - val_loss: 0.0912 - val_accuracy: 0.9960
```

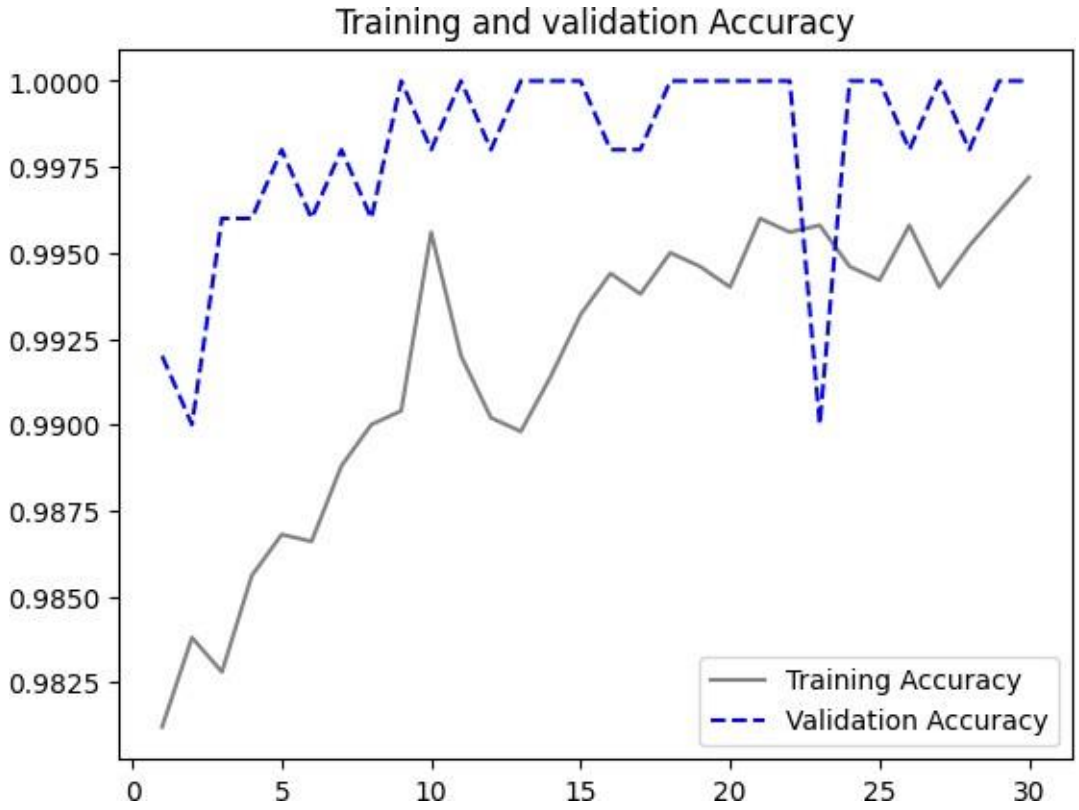
Epoch 7/30
157/157 [==============================] - 7s 41ms/step - loss: 0.1625 - accuracy: 0.9888 - val_loss: 0.0850 - val_accuracy: 0.9980
Epoch 8/30
157/157 [==============================] - 7s 41ms/step - loss: 0.1762 - accuracy: 0.9900 - val_loss: 0.1176 - val_accuracy: 0.9960
Epoch 9/30
157/157 [==============================] - 7s 43ms/step - loss: 0.1684 - accuracy: 0.9904 - val_loss: 1.0634e-04 - val_accuracy: 1.0000
Epoch 10/30
157/157 [==============================] - 7s 41ms/step - loss: 0.0748 -

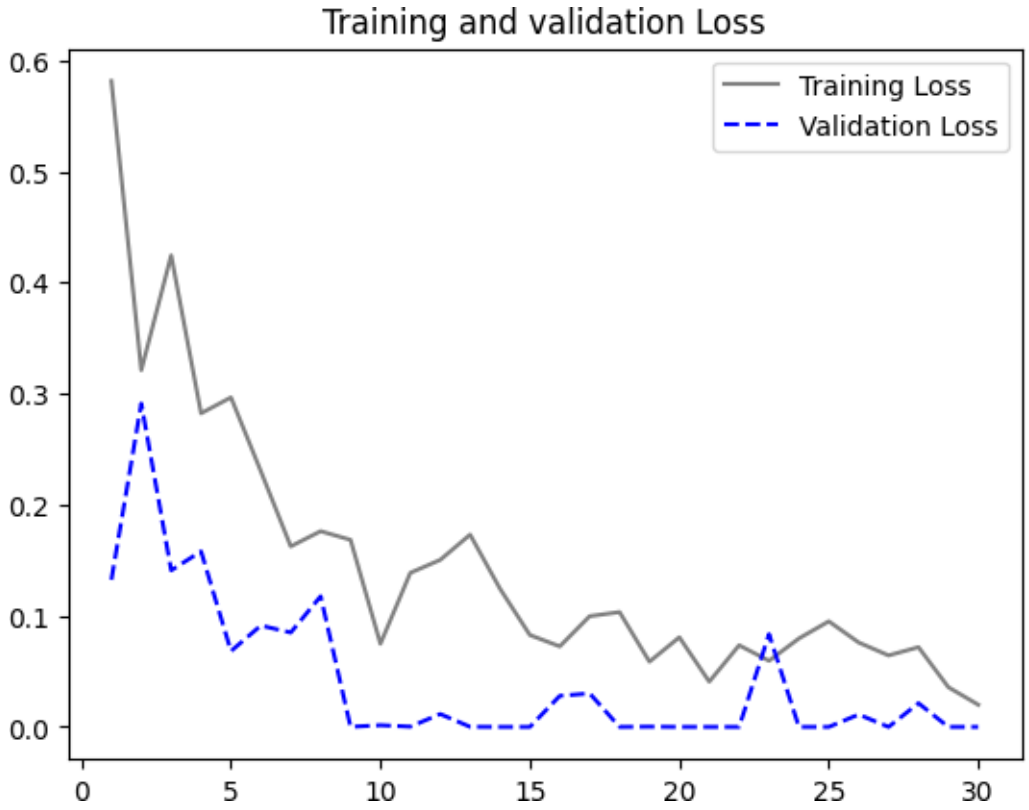
accuracy: 0.9956 - val_loss: 0.0016 - val_accuracy: 0.9980Epoch
11/30
157/157 [==============================] - 7s 41ms/step - loss: 0.1387 -
accuracy: 0.9920 - val_loss: 2.5302e-04 - val_accuracy: 1.0000Epoch
12/30
157/157 [==============================] - 7s 41ms/step - loss: 0.1503 -
accuracy: 0.9902 - val_loss: 0.0116 - val_accuracy: 0.9980Epoch
13/30
157/157 [==============================] - 7s 41ms/step - loss: 0.1731 -
accuracy: 0.9898 - val_loss: 1.1119e-04 - val_accuracy: 1.0000Epoch
14/30
157/157 [==============================] - 7s 42ms/step - loss: 0.1244 -
accuracy: 0.9914 - val_loss: 2.8760e-09 - val_accuracy: 1.0000Epoch
15/30
157/157 [==============================] - 7s 43ms/step - loss: 0.0827 -
accuracy: 0.9932 - val_loss: 6.3425e-11 - val_accuracy: 1.0000Epoch
16/30
157/157 [==============================] - 7s 41ms/step - loss: 0.0725 -
accuracy: 0.9944 - val_loss: 0.0280 - val_accuracy: 0.9980Epoch
17/30
157/157 [==============================] - 7s 41ms/step - loss: 0.0997 -
accuracy: 0.9938 - val_loss: 0.0302 - val_accuracy: 0.9980Epoch
18/30
157/157 [==============================] - 7s 41ms/step - loss: 0.1034 -
accuracy: 0.9950 - val_loss: 1.1098e-07 - val_accuracy: 1.0000Epoch
19/30
157/157 [==============================] - 7s 41ms/step - loss: 0.0587 -
accuracy: 0.9946 - val_loss: 2.2195e-04 - val_accuracy: 1.0000Epoch
20/30
157/157 [==============================] - 7s 43ms/step - loss: 0.0807 -
accuracy: 0.9940 - val_loss: 6.6733e-12 - val_accuracy: 1.0000Epoch
21/30
157/157 [==============================] - 7s 41ms/step - loss: 0.0407 -
accuracy: 0.9960 - val_loss: 1.4522e-10 - val_accuracy: 1.0000Epoch
22/30
157/157 [==============================] - 7s 43ms/step - loss: 0.0735 -
accuracy: 0.9956 - val_loss: 6.0817e-14 - val_accuracy: 1.0000Epoch
23/30
157/157 [==============================] - 7s 41ms/step - loss: 0.0595 -

accuracy: 0.9958 - val_loss: 0.0835 - val_accuracy: 0.9900Epoch 24/30
157/157 [==============================] - 7s 43ms/step - loss: 0.0796 - accuracy: 0.9946 - val_loss: 2.2641e-17 - val_accuracy: 1.0000Epoch 25/30
157/157 [==============================] - 7s 41ms/step - loss: 0.0951 - accuracy: 0.9942 - val_loss: 2.7673e-10 - val_accuracy: 1.0000Epoch 26/30
157/157 [==============================] - 7s 41ms/step - loss: 0.0760 -

```
accuracy: 0.9958 - val_loss: 0.0112 - val_accuracy: 0.9980Epoch
27/30
157/157 [==============================] - 7s 43ms/step - loss: 0.0644 -
accuracy: 0.9940 - val_loss: 2.9750e-19 - val_accuracy: 1.0000Epoch
28/30
157/157 [==============================] - 7s 42ms/step - loss: 0.0718 -
accuracy: 0.9952 - val_loss: 0.0213 - val_accuracy: 0.9980Epoch
29/30
157/157 [==============================] - 7s 41ms/step - loss: 0.0357 -
accuracy: 0.9962 - val_loss: 1.3778e-12 - val_accuracy: 1.0000Epoch
30/30
157/157 [==============================] - 7s 42ms/step - loss: 0.0200 -
accuracy: 0.9972 - val_loss: 3.2841e-07 - val_accuracy: 1.0000
```

```
acc = FineTuned_VGG_Model_2.history["accuracy"]
val_acc = FineTuned_VGG_Model_2.history["val_accuracy"]

loss = FineTuned_VGG_Model_2.history["loss"]
val_loss = FineTuned_VGG_Model_2.history["val_loss"]

epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, color="grey", label="Training Accuracy")
plt.plot(epochs, val_acc, color="blue", linestyle="dashed", label="Validation␣
↪Accuracy")
plt.title("Training and validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and validation Loss")
plt.legend()
plt.show()
```


Training and validation Accuracy
1.0000
0.9975
0.9950
0.9925
0.9900
0.9875
0.9850
0.9825
0
5
10
15
20
25
30
Training Accuracy
Validation Accuracy




```
best_model = keras.models.load_model("fine_tuning_vgg_model2.keras")
FineTuned_VGG_Model_2_Results = best_model.evaluate(test_dataset)
print(f"Loss:  {FineTuned_VGG_Model_2_Results[0]:.3f}")
print(f"Accuracy:  {FineTuned_VGG_Model_2_Results[1]:.3f}")
```

16/16 [==============================] - 1s 31ms/step - loss: 4.7333e-08 -
accuracy: 1.0000
Loss: 0.000
Accuracy: 1.000

Using the pre-trained network VGG16, we created three models for the examination of the above two VGG16 models. Interestingly, we found that accuracy increased when the pre-trained network was frozen in its first layers and prevented from altering its weights during training. As such, we want to use the same methods with a training sample size of 5000 to construct two models.

VGG - Model 3 (5000 Training Samples)

```
conv_base= keras.applications.vgg16.VGG16(
                weights="imagenet",
                include_top=False)
```

```
conv_base.trainable = False
```

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_17 (InputLayer)       [(None, None, None, 3)]   0
 block1_conv1 (Conv2D)       (None, None, None, 64)    1792
 block1_conv2 (Conv2D)       (None, None, None, 64)    36928
 block1_pool (MaxPooling2D)  (None, None, None, 64)    0
 block2_conv1 (Conv2D)       (None, None, None, 128)   73856
 block2_conv2 (Conv2D)       (None, None, None, 128)   147584
 block2_pool (MaxPooling2D)  (None, None, None, 128)   0
 block3_conv1 (Conv2D)       (None, None, None, 256)   295168
 block3_conv2 (Conv2D)       (None, None, None, 256)   590080
 block3_conv3 (Conv2D)       (None, None, None, 256)   590080
 block3_pool (MaxPooling2D)  (None, None, None, 256)   0
 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160
 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808
 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808
 block4_pool (MaxPooling2D)  (None, None, None, 512)   0
 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808
 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808
 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808
 block5_pool (MaxPooling2D)  (None, None, None, 512)   0
```

Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_____

```
train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

Found 5000 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.

```
# Data Augmentation
data_augmentation_4 = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.15),
        layers.RandomZoom(0.25),
    ]
)

# Adding the Classifier and Dense Network inputs
= keras.Input(shape=(180, 180, 3)) x =
data_augmentation_4(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)

outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs, outputs)

# Compiling the Model
model.compile(loss="binary_crossentropy",
              optimizer="adam",
```

```
                metrics=["accuracy"])

# Using CallBacks to monitor the best val_loss
callbacks = ModelCheckpoint(
                filepath= "vgg_model3.keras",
                save_best_only= True,
                monitor= "val_loss")

# Model Fit
VGG_Model_3 = model.fit(
                  train_dataset,
                  epochs= 50,
                  validation_data=validation_dataset,
                  callbacks=callbacks)
```

```
Epoch 1/50
157/157 [==============================] - 8s 41ms/step - loss: 10.5988 -
accuracy: 0.9154 - val_loss: 1.8303 - val_accuracy: 0.9860Epoch
2/50
157/157 [==============================] - 6s 40ms/step - loss: 6.1195 -
accuracy: 0.9448 - val_loss: 0.8026 - val_accuracy: 0.9860
Epoch 3/50
157/157 [==============================] - 6s 38ms/step - loss: 4.1963 -
accuracy: 0.9504 - val_loss: 1.0276 - val_accuracy: 0.9840
Epoch 4/50
157/157 [==============================] - 6s 40ms/step - loss: 2.2763 -
accuracy: 0.9618 - val_loss: 0.1023 - val_accuracy: 0.9960
Epoch 5/50
157/157 [==============================] - 6s 38ms/step - loss: 1.9037 -
accuracy: 0.9580 - val_loss: 0.2311 - val_accuracy: 0.9940
Epoch 6/50
157/157 [==============================] - 6s 40ms/step - loss: 0.9389 -
accuracy: 0.9644 - val_loss: 0.0859 - val_accuracy: 0.9960
Epoch 7/50
157/157 [==============================] - 6s 38ms/step - loss: 0.6627 -
accuracy: 0.9674 - val_loss: 0.1129 - val_accuracy: 0.9960
Epoch 8/50
157/157 [==============================] - 6s 40ms/step - loss: 0.4668 -
accuracy: 0.9668 - val_loss: 0.0529 - val_accuracy: 0.9920
Epoch 9/50
157/157 [==============================] - 6s 38ms/step - loss: 0.4064 -
accuracy: 0.9660 - val_loss: 0.0833 - val_accuracy: 0.9940
Epoch 10/50
157/157 [==============================] - 6s 39ms/step - loss: 0.3713 -
accuracy: 0.9600 - val_loss: 0.0431 - val_accuracy: 0.9960
Epoch 11/50
157/157 [==============================] - 6s 38ms/step - loss: 0.4639 -
```

accuracy: 0.9620 - val_loss: 0.0717 - val_accuracy: 0.9920Epoch 12/50
157/157 [==============================] - 6s 38ms/step - loss: 0.3162 - accuracy: 0.9696 - val_loss: 0.0694 - val_accuracy: 0.9900
Epoch 13/50
157/157 [==============================] - 6s 38ms/step - loss: 0.3262 - accuracy: 0.9664 - val_loss: 0.0674 - val_accuracy: 0.9860
Epoch 14/50
157/157 [==============================] - 6s 38ms/step - loss: 0.3931 - accuracy: 0.9618 - val_loss: 0.0870 - val_accuracy: 0.9920
Epoch 15/50
157/157 [==============================] - 6s 38ms/step - loss: 0.3704 - accuracy: 0.9678 - val_loss: 0.1370 - val_accuracy: 0.9880
Epoch 16/50
157/157 [==============================] - 6s 39ms/step - loss: 0.5008 - accuracy: 0.9616 - val_loss: 0.0080 - val_accuracy: 0.9960
Epoch 17/50
157/157 [==============================] - 6s 38ms/step - loss: 0.5328 - accuracy: 0.9596 - val_loss: 0.0858 - val_accuracy: 0.9940
Epoch 18/50
157/157 [==============================] - 6s 39ms/step - loss: 0.5890 - accuracy: 0.9610 - val_loss: 0.0608 - val_accuracy: 0.9860
Epoch 19/50
157/157 [==============================] - 6s 38ms/step - loss: 0.4884 - accuracy: 0.9666 - val_loss: 0.0246 - val_accuracy: 0.9960
Epoch 20/50
157/157 [==============================] - 6s 39ms/step - loss: 0.5917 - accuracy: 0.9616 - val_loss: 0.1491 - val_accuracy: 0.9900
Epoch 21/50
157/157 [==============================] - 6s 38ms/step - loss: 0.5693 - accuracy: 0.9610 - val_loss: 0.2390 - val_accuracy: 0.9940
Epoch 22/50
157/157 [==============================] - 6s 38ms/step - loss: 0.5711 - accuracy: 0.9658 - val_loss: 0.2256 - val_accuracy: 0.9860
Epoch 23/50
157/157 [==============================] - 6s 38ms/step - loss: 0.6966 - accuracy: 0.9640 - val_loss: 0.2493 - val_accuracy: 0.9860
Epoch 24/50
157/157 [==============================] - 6s 38ms/step - loss: 0.7505 -

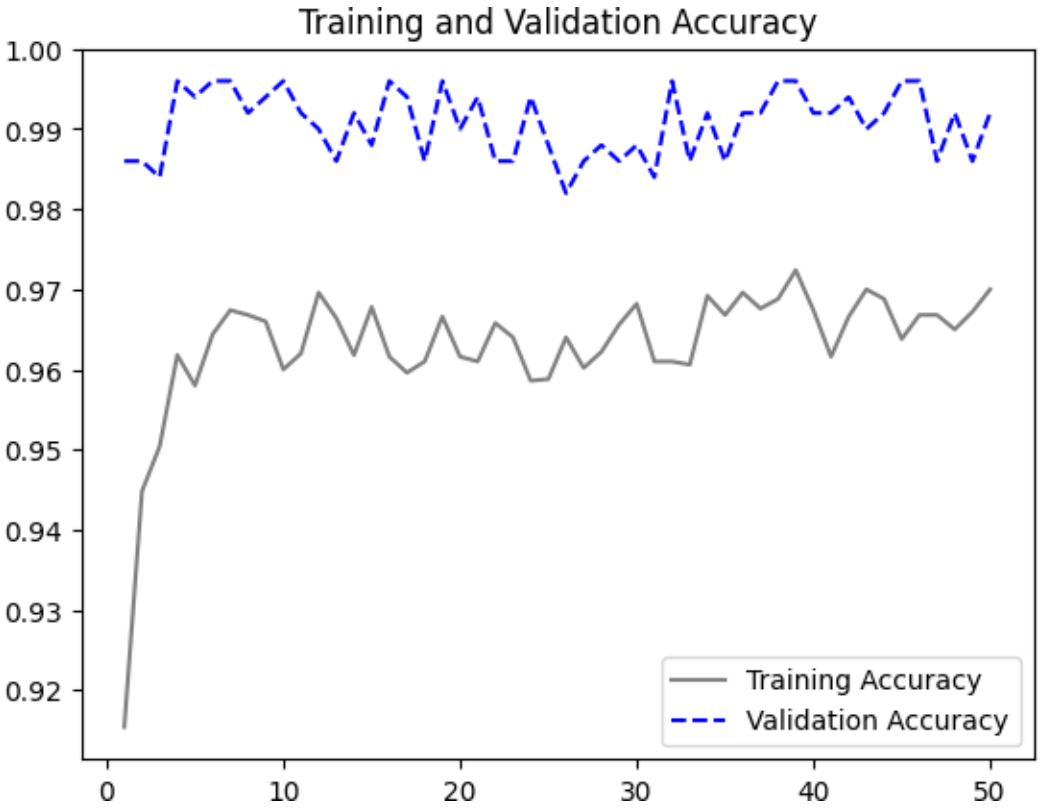
accuracy: 0.9586 - val_loss: 0.1758 - val_accuracy: 0.9940
Epoch 25/50
157/157 [==============================] - 6s 39ms/step - loss: 0.9393 -
accuracy: 0.9588 - val_loss: 0.1788 - val_accuracy: 0.9880
Epoch 26/50
157/157 [==============================] - 6s 38ms/step - loss: 0.9032 -
accuracy: 0.9640 - val_loss: 0.2811 - val_accuracy: 0.9820
Epoch 27/50
157/157 [==============================] - 6s 38ms/step - loss: 0.7521 -

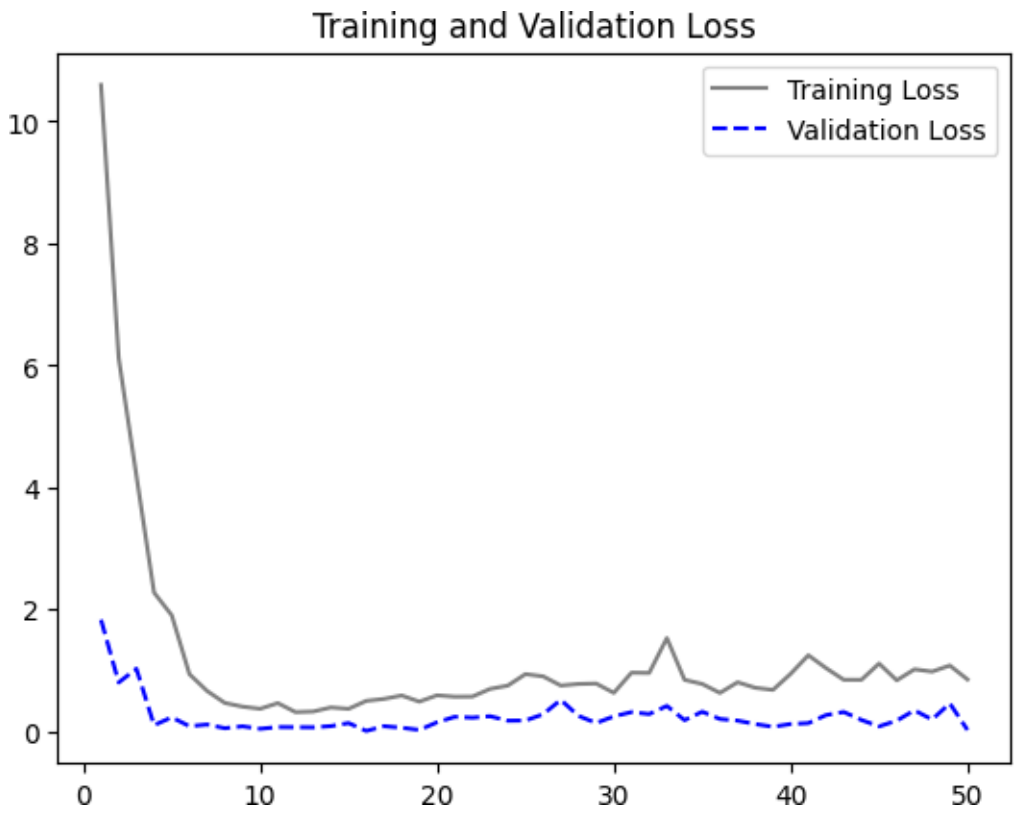
accuracy: 0.9602 - val_loss: 0.5175 - val_accuracy: 0.9860Epoch 28/50
157/157 [==============================] - 6s 38ms/step - loss: 0.7780 - accuracy: 0.9622 - val_loss: 0.2543 - val_accuracy: 0.9880
Epoch 29/50
157/157 [==============================] - 6s 38ms/step - loss: 0.7849 - accuracy: 0.9656 - val_loss: 0.1381 - val_accuracy: 0.9860
Epoch 30/50
157/157 [==============================] - 6s 38ms/step - loss: 0.6320 - accuracy: 0.9682 - val_loss: 0.2467 - val_accuracy: 0.9880
Epoch 31/50
157/157 [==============================] - 6s 38ms/step - loss: 0.9630 - accuracy: 0.9610 - val_loss: 0.3166 - val_accuracy: 0.9840
Epoch 32/50
157/157 [==============================] - 6s 38ms/step - loss: 0.9591 - accuracy: 0.9610 - val_loss: 0.2846 - val_accuracy: 0.9960
Epoch 33/50
157/157 [==============================] - 6s 38ms/step - loss: 1.5303 - accuracy: 0.9606 - val_loss: 0.4180 - val_accuracy: 0.9860
Epoch 34/50
157/157 [==============================] - 6s 38ms/step - loss: 0.8457 - accuracy: 0.9692 - val_loss: 0.1845 - val_accuracy: 0.9920
Epoch 35/50
157/157 [==============================] - 6s 38ms/step - loss: 0.7779 - accuracy: 0.9668 - val_loss: 0.3229 - val_accuracy: 0.9860
Epoch 36/50
157/157 [==============================] - 6s 38ms/step - loss: 0.6315 - accuracy: 0.9696 - val_loss: 0.2035 - val_accuracy: 0.9920
Epoch 37/50
157/157 [==============================] - 6s 38ms/step - loss: 0.8062 - accuracy: 0.9676 - val_loss: 0.1770 - val_accuracy: 0.9920
Epoch 38/50
157/157 [==============================] - 6s 38ms/step - loss: 0.7154 - accuracy: 0.9688 - val_loss: 0.1205 - val_accuracy: 0.9960
Epoch 39/50
157/157 [==============================] - 6s 38ms/step - loss: 0.6808 - accuracy: 0.9724 - val_loss: 0.0764 - val_accuracy: 0.9960
Epoch 40/50
157/157 [==============================] - 6s 38ms/step - loss: 0.9447 -

accuracy: 0.9674 - val_loss: 0.1216 - val_accuracy: 0.9920
Epoch 41/50
157/157 [==============================] - 6s 38ms/step - loss: 1.2487 -
accuracy: 0.9616 - val_loss: 0.1357 - val_accuracy: 0.9920
Epoch 42/50
157/157 [==============================] - 6s 38ms/step - loss: 1.0383 -
accuracy: 0.9666 - val_loss: 0.2674 - val_accuracy: 0.9940
Epoch 43/50
157/157 [==============================] - 6s 38ms/step - loss: 0.8472 -

```
accuracy: 0.9700 - val_loss: 0.3185 - val_accuracy: 0.9900Epoch
44/50
157/157 [==============================] - 6s 38ms/step - loss: 0.8462 -
accuracy: 0.9688 - val_loss: 0.1882 - val_accuracy: 0.9920Epoch
45/50
157/157 [==============================] - 6s 38ms/step - loss: 1.1127 -
accuracy: 0.9638 - val_loss: 0.0827 - val_accuracy: 0.9960Epoch
46/50
157/157 [==============================] - 6s 38ms/step - loss: 0.8377 -
accuracy: 0.9668 - val_loss: 0.1768 - val_accuracy: 0.9960Epoch
47/50
157/157 [==============================] - 6s 38ms/step - loss: 1.0179 -
accuracy: 0.9668 - val_loss: 0.3462 - val_accuracy: 0.9860Epoch
48/50
157/157 [==============================] - 6s 38ms/step - loss: 0.9812 -
accuracy: 0.9650 - val_loss: 0.1947 - val_accuracy: 0.9920Epoch
49/50
157/157 [==============================] - 6s 38ms/step - loss: 1.0804 -
accuracy: 0.9672 - val_loss: 0.4596 - val_accuracy: 0.9860Epoch
50/50
157/157 [==============================] - 6s 38ms/step - loss: 0.8524 -
accuracy: 0.9700 - val_loss: 0.0235 - val_accuracy: 0.9920
```

```
acc = VGG_Model_3.history["accuracy"]
val_acc = VGG_Model_3.history["val_accuracy"]

loss = VGG_Model_3.history["loss"]
val_loss = VGG_Model_3.history["val_loss"]

epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, color="grey", label="Training Accuracy")
plt.plot(epochs, val_acc, color="blue", linestyle="dashed", label="Validation␣
↪Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```


Training and Validation Accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0
10
20
30
40
50
Training Accuracy
Validation Accuracy




```
best_model = keras.models.load_model("vgg_model3.keras")
VGG_Model_3_Results = best_model.evaluate(test_dataset)
print(f"Loss: {VGG_Model_3_Results[0]:.3f}")
print(f"Accuracy: {VGG_Model_3_Results[1]:.3f}")
```

```
16/16 [==============================] - 1s 32ms/step - loss: 0.1335 - accuracy:
0.9840
Loss: 0.134
Accuracy: 0.984
```

Fine Tunning VGG_Model_3 (Training Samples - 5000)

We have decided to freeze the first four layers in order to optimize VGG_Model3. By using this tactic, we hope to stop the model from overfitting and free it up to focus just on identifying the unique characteristics that are pertinent to our specific categorization task.As such, we have simultaneously made sure that the first four layers stay frozen and configured the pre-trained layers to remain unchanged during training. The model performs better with these improvements applied, especially when working with a training sample size of 5000.

```
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```

```
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.Adam(learning_rate=1e-5),
              metrics=["accuracy"])


callbacks = ModelCheckpoint(
              filepath="fine_tuning_vgg_model3.keras",
              save_best_only=True,
              monitor="val_loss")


FineTuned_VGG_Model_3 = model.fit(
                          train_dataset,
                          epochs=50,
                          validation_data=validation_dataset,
                          callbacks=callbacks)
```

```
Epoch 1/50
157/157 [==============================] - 10s 44ms/step - loss: 0.6214 -
accuracy: 0.9730 - val_loss: 0.1435 - val_accuracy: 0.9880Epoch
2/50
157/157 [==============================] - 7s 43ms/step - loss: 0.5057 -
accuracy: 0.9704 - val_loss: 0.0525 - val_accuracy: 0.9920Epoch
3/50
157/157 [==============================] - 7s 41ms/step - loss: 0.3409 -
accuracy: 0.9768 - val_loss: 0.0897 - val_accuracy: 0.9960Epoch
4/50
157/157 [==============================] - 7s 41ms/step - loss: 0.3513 -
accuracy: 0.9760 - val_loss: 0.1144 - val_accuracy: 0.9960Epoch
5/50
157/157 [==============================] - 7s 44ms/step - loss: 0.2530 -
accuracy: 0.9808 - val_loss: 0.0221 - val_accuracy: 0.9980Epoch
6/50
157/157 [==============================] - 7s 42ms/step - loss: 0.2061 -
accuracy: 0.9836 - val_loss: 0.0499 - val_accuracy: 0.9960Epoch
7/50
157/157 [==============================] - 7s 42ms/step - loss: 0.1904 -
accuracy: 0.9840 - val_loss: 0.0526 - val_accuracy: 0.9960Epoch
8/50
157/157 [==============================] - 7s 42ms/step - loss: 0.1629 -
accuracy: 0.9866 - val_loss: 0.0543 - val_accuracy: 0.9960Epoch
```

9/50
157/157 [==============================] - 7s 42ms/step - loss: 0.1447 - accuracy: 0.9846 - val_loss: 0.0515 - val_accuracy: 0.9960Epoch 10/50
157/157 [==============================] - 7s 42ms/step - loss: 0.1259 - accuracy: 0.9880 - val_loss: 0.0298 - val_accuracy: 0.9980

```
Epoch 11/50
157/157 [==============================] - 7s 44ms/step - loss: 0.1303 -
accuracy: 0.9862 - val_loss: 0.0025 - val_accuracy: 0.9980
Epoch 12/50
157/157 [==============================] - 7s 42ms/step - loss: 0.1402 -
accuracy: 0.9862 - val_loss: 0.0121 - val_accuracy: 0.9980
Epoch 13/50
157/157 [==============================] - 7s 44ms/step - loss: 0.0903 -
accuracy: 0.9888 - val_loss: 3.9519e-04 - val_accuracy: 1.0000
Epoch 14/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0803 -
accuracy: 0.9902 - val_loss: 0.0234 - val_accuracy: 0.9980
Epoch 15/50
157/157 [==============================] - 7s 42ms/step - loss: 0.1087 -
accuracy: 0.9862 - val_loss: 0.0246 - val_accuracy: 0.9960
Epoch 16/50
157/157 [==============================] - 7s 42ms/step - loss: 0.1071 -
accuracy: 0.9870 - val_loss: 0.0136 - val_accuracy: 0.9960
Epoch 17/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0903 -
accuracy: 0.9906 - val_loss: 0.0116 - val_accuracy: 0.9960
Epoch 18/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0652 -
accuracy: 0.9890 - val_loss: 0.0104 - val_accuracy: 0.9960
Epoch 19/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0480 -
accuracy: 0.9922 - val_loss: 0.0203 - val_accuracy: 0.9980
Epoch 20/50
157/157 [==============================] - 7s 43ms/step - loss: 0.0712 -
accuracy: 0.9902 - val_loss: 1.0469e-04 - val_accuracy: 1.0000
Epoch 21/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0436 -
accuracy: 0.9924 - val_loss: 0.0030 - val_accuracy: 0.9980
Epoch 22/50
157/157 [==============================] - 7s 43ms/step - loss: 0.0439 -
accuracy: 0.9908 - val_loss: 2.4193e-07 - val_accuracy: 1.0000
Epoch 23/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0547 -
accuracy: 0.9930 - val_loss: 4.0447e-05 - val_accuracy: 1.0000
```

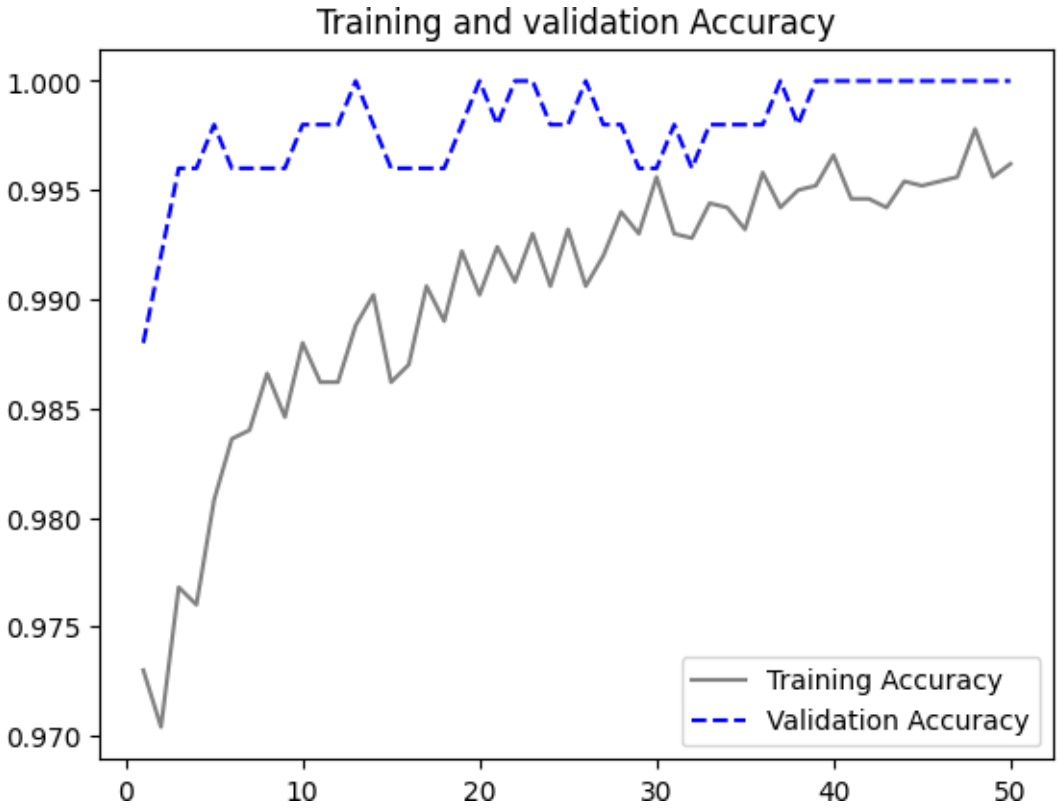
Epoch 24/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0620 - accuracy: 0.9906 - val_loss: 0.0081 - val_accuracy: 0.9980
Epoch 25/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0461 - accuracy: 0.9932 - val_loss: 0.0170 - val_accuracy: 0.9980
Epoch 26/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0581 - accuracy: 0.9906 - val_loss: 7.4745e-05 - val_accuracy: 1.0000

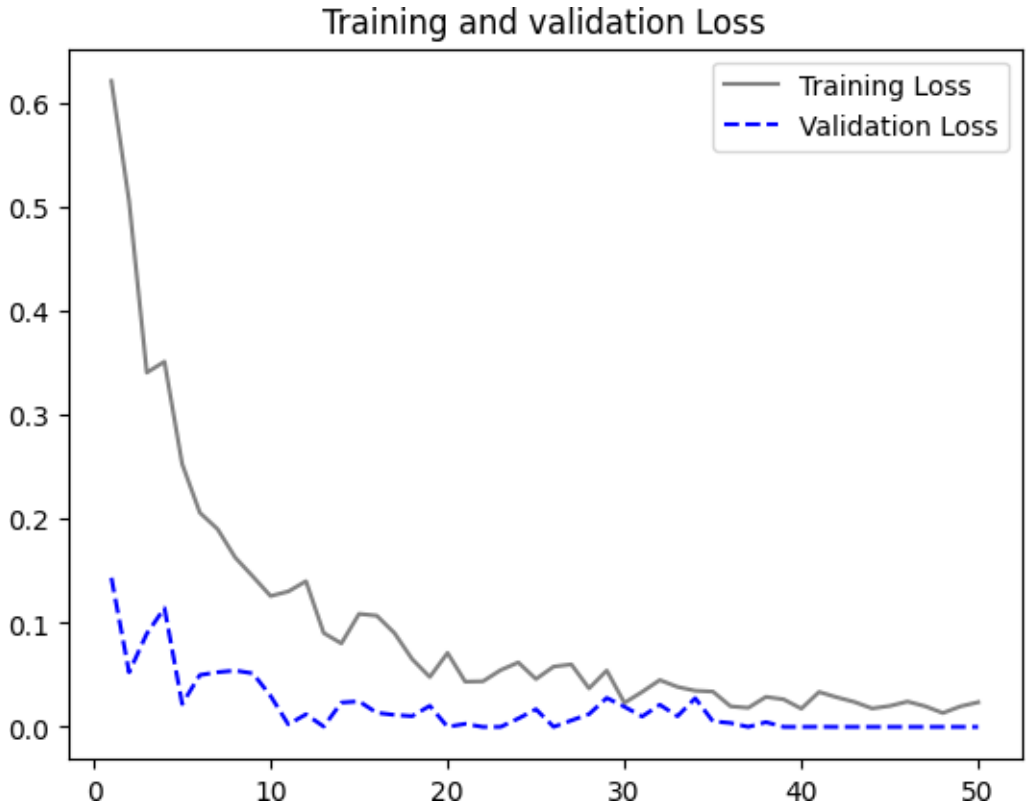
```
Epoch 27/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0601 -
accuracy: 0.9920 - val_loss: 0.0063 - val_accuracy: 0.9980
Epoch 28/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0370 -
accuracy: 0.9940 - val_loss: 0.0123 - val_accuracy: 0.9980
Epoch 29/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0543 -
accuracy: 0.9930 - val_loss: 0.0279 - val_accuracy: 0.9960
Epoch 30/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0231 -
accuracy: 0.9956 - val_loss: 0.0194 - val_accuracy: 0.9960
Epoch 31/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0337 -
accuracy: 0.9930 - val_loss: 0.0100 - val_accuracy: 0.9980
Epoch 32/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0451 -
accuracy: 0.9928 - val_loss: 0.0214 - val_accuracy: 0.9960
Epoch 33/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0385 -
accuracy: 0.9944 - val_loss: 0.0103 - val_accuracy: 0.9980
Epoch 34/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0348 -
accuracy: 0.9942 - val_loss: 0.0277 - val_accuracy: 0.9980
Epoch 35/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0340 -
accuracy: 0.9932 - val_loss: 0.0054 - val_accuracy: 0.9980
Epoch 36/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0198 -
accuracy: 0.9958 - val_loss: 0.0038 - val_accuracy: 0.9980
Epoch 37/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0184 -
accuracy: 0.9942 - val_loss: 2.5245e-05 - val_accuracy: 1.0000
Epoch 38/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0289 -
accuracy: 0.9950 - val_loss: 0.0046 - val_accuracy: 0.9980
Epoch 39/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0265 -
accuracy: 0.9952 - val_loss: 2.6175e-05 - val_accuracy: 1.0000
```

Epoch 40/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0176 - accuracy: 0.9966 - val_loss: 4.8281e-07 - val_accuracy: 1.0000
Epoch 41/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0336 - accuracy: 0.9946 - val_loss: 1.0537e-04 - val_accuracy: 1.0000
Epoch 42/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0285 - accuracy: 0.9946 - val_loss: 1.7322e-06 - val_accuracy: 1.0000

```
Epoch 43/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0240 -
accuracy: 0.9942 - val_loss: 2.9836e-06 - val_accuracy: 1.0000Epoch
44/50
157/157 [==============================] - 7s 43ms/step - loss: 0.0178 -
accuracy: 0.9954 - val_loss: 2.6146e-08 - val_accuracy: 1.0000Epoch
45/50
157/157 [==============================] - 7s 42ms/step - loss: 0.0202 -
accuracy: 0.9952 - val_loss: 2.4768e-06 - val_accuracy: 1.0000Epoch
46/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0244 -
accuracy: 0.9954 - val_loss: 1.0928e-06 - val_accuracy: 1.0000Epoch
47/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0199 -
accuracy: 0.9956 - val_loss: 2.2991e-06 - val_accuracy: 1.0000Epoch
48/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0134 -
accuracy: 0.9978 - val_loss: 3.6515e-05 - val_accuracy: 1.0000Epoch
49/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0198 -
accuracy: 0.9956 - val_loss: 2.3712e-05 - val_accuracy: 1.0000Epoch
50/50
157/157 [==============================] - 7s 41ms/step - loss: 0.0238 -
accuracy: 0.9962 - val_loss: 4.2211e-08 - val_accuracy: 1.0000
```

```
acc = FineTuned_VGG_Model_3.history["accuracy"]
val_acc = FineTuned_VGG_Model_3.history["val_accuracy"]

loss = FineTuned_VGG_Model_3.history["loss"]
val_loss = FineTuned_VGG_Model_3.history["val_loss"]

epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, color="grey", label="Training Accuracy")
plt.plot(epochs, val_acc, color="blue", linestyle="dashed", label="Validation␣
↪Accuracy")
plt.title("Training and validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, color="grey", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation␣
↪Loss")
plt.title("Training and validation Loss")
plt.legend()
plt.show()
```


Training and validation Accuracy
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0
10
20
30
40
50
Training Accuracy
Validation Accuracy




```
best_model = keras.models.load_model("fine_tuning_vgg_model3.keras")
FineTuned_VGG_Model_3_Results = best_model.evaluate(test_dataset)
print(f"Loss: {FineTuned_VGG_Model_3_Results[0]:.3f}")
print(f"Accuracy: {FineTuned_VGG_Model_3_Results[1]:.3f}")
```

```
16/16 [==============================] - 1s 32ms/step - loss: 1.5571e-08 -
accuracy: 1.0000
Loss: 0.000
Accuracy: 1.000
```

After building a total of 15 models—two of which are optimized copies of the original models—we are now prepared to do a comparative study in order to identify the top models in two different categories: Pre-Trained Models and Scratch Models. Our immediate goal is to assess which scratch-built model performs the best. The evaluation comprises a comparison of the accuracy and loss metrics of the ten models that were constructed using four distinct training data. Finding the ideal training sample size for the classification of cats and dogs is the main goal.

Model 1: filters from 32 to 256, 5 Input Layers

Model 2: filters from 32 to 256, 5 Input Layers, Augmented Images and Droput rate of 0.5
Model 3: filters from 32 to 512, 6 Input Layers, Augmented Images and Dropout rate of 0.5
Model 4: filters from 64 to 1024, 5 Input Layers, Augmented Images and Dropout rate of 0.6

Model 5: filters from 32 to 256, 5 Input Layers, Augmented Images and Droput rate of 0.5, training size 2000

Model 6: filters from 32 to 256, 5 Input Layers, Augmented Images and Droput rate of 0.5, training size 2000, Padding being same

Model 7: MaxPooling Operation,filters from 32 to 512, 5 Input Layers, Augmented Images, droput rate of 0.5, Training Sample - 3000

Model 8: MaxPooling + Strides of Step-Size 2,filters from 32 to 512, 5 Input Layers, Augmented Images, droput rate of 0.5, Training Sample - 3000

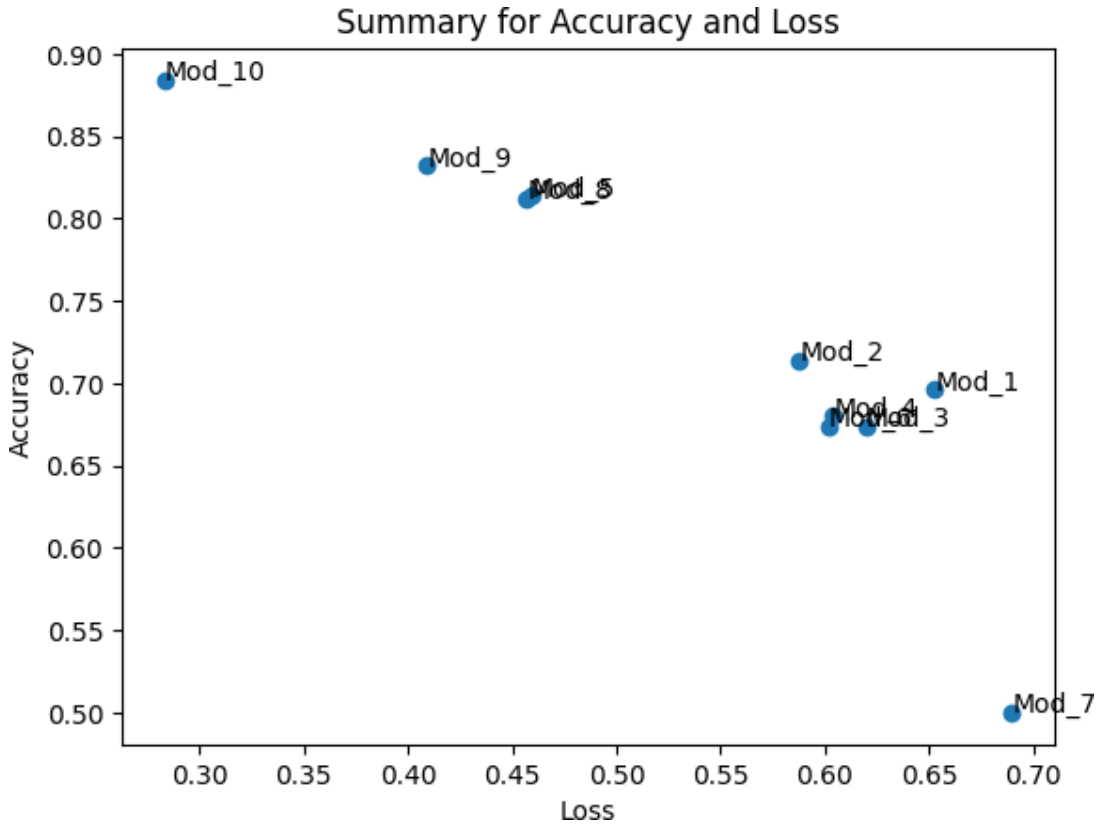
Model 9: MaxPooling + Strides of Step-Size 2 with Padding turned on,filters from 32 to 512, 5 Input Layers,Augmented Images, droput rate of 0.5, Training Sample - 3000

Model 10: filters from 32 to 512, 5 Input Layers,Augmented Images, droput rate of 0.5, Training

Sample - 500

```
# Scratch  Models
Model_1 = (0.653, 0.696)
Model_2 = (0.588, 0.714)
Model_3 = (0.620,0.674)
Model_4 = (0.604, 0.680)
Model_5 = (0.459,0.814)
Model_6 = (0.602,0.674)
Model_7 = (0.69,0.500)
Model_8 = (0.457,0.812)
Model_9 = (0.409,0.832)
Model_10 = (0.283,0.884)
```

```
Models_4 = \
  ↪("Mod_1","Mod_2","Mod_3","Mod_4","Mod_5","Mod_6","Mod_7","Mod_8","Mod_9","Mod_10")
Loss_4 = \
  ↪(Model_1[0],Model_2[0],Model_3[0],Model_4[0],Model_5[0],Model_6[0],Model_7[0],Model_8[0],Mo
Accuracy_4 = \
  ↪(Model_1[1],Model_2[1],Model_3[1],Model_4[1],Model_5[1],Model_6[1],Model_7[1],Model_8[1],Mo

fig, ax = plt.subplots()
ax.scatter(Loss_4,Accuracy_4)
for i, txt in enumerate(Models_4): ax.annotate(txt,
    (Loss_4[i],Accuracy_4[i] ))
plt.title("Summary for Accuracy and Loss")
plt.ylabel("Accuracy")
plt.xlabel("Loss")

plt.show()
```



Out of all the scratch models, Model_10, which was trained on 5000 samples, turned out to be the best model with an impressive 88.4% accuracy and a 28.3% loss on the test set. A five-layer architecture with filters ranging from 32 to 256 was used to build Model 10. Augmented photos were integrated into the model during training, along with a max-pooling layer and a 0.5 dropout rate.

following, we created five models with the pre-trained vgg16 network. The first three were created with a sample size of 1000 and an optimizer named rmsprop, while the following two were created with a sample size of 5000 and an optimizer named Adam.

VGG 1: filters from 32 to 256, 5 Input Layers

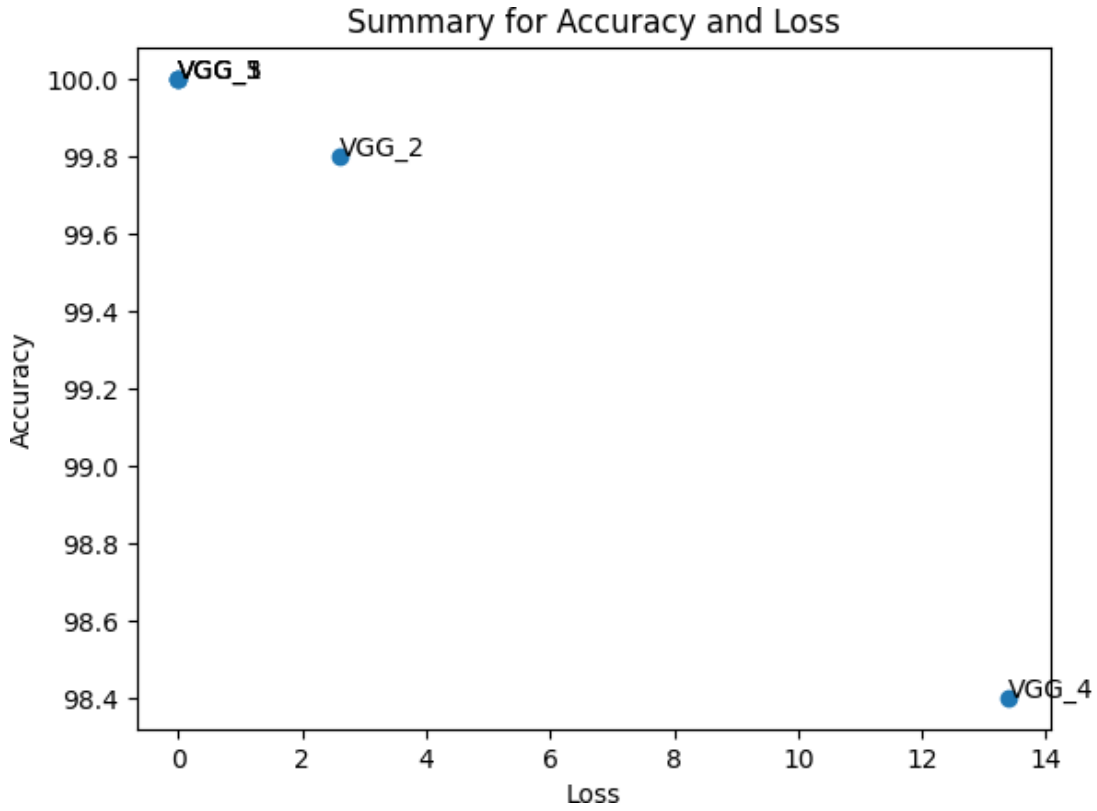
VGG 2: filters from 32 to 256, 5 Input Layers, Augmented Images and Droput rate of 0.5
VGG 3: filters from 32 to 512, 6 Input Layers, Augmented Images and Dropout rate of 0.5
VGG 4: VGG - Model 3 (5000 Training Samples)
VGG 5: Fine Tunning VGG_Model_3 (Training Samples - 5000)

```
# Pre-Trained Models
VGG_Model_1 = (0.00,100)
VGG_Model_2 = (2.6,99.8)
FineTuned_VGG_Model_2 = (0.00,100)
```

```
VGG_Model_3 = (13.4,98.4)
FineTuned_VGG_Model_3 = (0.00,100)
```

```
Models_5 = ("VGG_1","VGG_2","VGG_3","VGG_4","VGG_5")
Loss_5 = ⌴
  ↪(VGG_Model_1[0],VGG_Model_2[0],FineTuned_VGG_Model_2[0],VGG_Model_3[0],FineTuned_VGG
  _Model_
Accuracy_5 = ⌴
  ↪(VGG_Model_1[1],VGG_Model_2[1],FineTuned_VGG_Model_2[1],VGG_Model_3[1],FineTuned_VGG
  _Model_
```

```
fig, ax = plt.subplots()
ax.scatter(Loss_5,Accuracy_5)
for i, txt in enumerate(Models_5):
    ax.annotate(txt,
    (Loss_5[i],Accuracy_5[i] ))
plt.title("Summary for Accuracy
and Loss")
plt.ylabel("Accuracy")
plt.xlabel("Loss")

plt.show()
```



When it came to pre-trained models, the two best performers were Fine- Tuned_VGG_Model_2 and FineTuned_VGG_Model_3, or Model_5and model_3.

with a meagre 0.00% loss and an astounding 100% accuracy. This model was built with 2000 and 5000 training samples, and it was tuned at a learning rate of 0.000001 using the Adam optimizer.

Conclusion : According to the aforementioned analysis, a model's accuracy is closely related to the volume of training data and the underlying architecture, especially when the model is trained directly from its own data. On the other hand, if a pretrained model is used, the accuracy depends on the particular test set that is being assessed. It's important to keep in mind that some sample sets could be more difficult to work with than others, and strong results on one set might not translate to all other sets.

Scrach Model : The profound impact on test accuracy stems from both the size of the training dataset and the chosen model architecture. Notably, integrating

contemporary architectural features like residual connections, batch normalization, and depthwise separable convolutions into a basic model, alongside employing data augmentation and dropout techniques, significantly boosted test accuracy. Furthermore, expanding the training dataset from 1,000 to 3,000 samples led to substantial improvements in accuracy. Further increasing the dataset to 6,000 samples resulted in test accuracy levels similar to pretrained models. This phenomenon highlights overfitting, where a lack of samples limits the model's ability to generalize. Increasing the dataset size broadens the model's exposure to the underlying data distribution, enhancing test accuracy.

Pretrained network : However, even while test accuracy is highest when using a pretrained network, the size of the training sample has little bearing on the results. This is a result of the accuracy-measuring model not being trained using the data it handles. Furthermore, methods like data augmentation and fine-tuning do not significantly improve accuracy because the original pretrained model already has high-performance accuracy, owing to the large size of the dataset used in the pretrained VGG16 model (more than 500 MB in size and more than 138 million parameters).

Recommendations: With a rise in training data from 1,000 to 3,000 samples, the simple scratch model's accuracy increased significantly, from 0.696 to 0.88.4. A test accuracy of about 83.2% was obtained using 3,000 training samples, three contemporary architectures, data augmentation, and dropout approaches. Moreover, the test accuracy was 88% when 5,000 samples were used to train the model. Because Model 10 is the most accurate of the scratch models, it is advised to use it first.

Since we achieved 100% accuracy with the pre-tuned model, I also think that using pretrained models isn't always the best option. Context-independent reasoning states that the background or context of an image does not influence the ability to discriminate between a dog and a cat in the images used for this exercise. Therefore, as long as the sample photos belong to the same category as the target used to train the model, a pretrained model that can identify images can be applied to any image differentiation job.