```
!mkdir -p ~/.kaggle
```

Start coding or generate with AI.

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

Choose Files kaggle.json

- **kaggle.json**(application/json) - 67 bytes, last modified: 10/20/2024 - 100% done

```
Saving kaggle.json to kaggle.json
```

```
!cp kaggle.json ~/.kaggle/
```

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

```
!ls ~/.kaggle
```

```
kaggle.json
```

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

```
Downloading dogs-vs-cats.zip to /content
 99% 804M/812M [00:04<00:00, 190MB/s]
100% 812M/812M [00:04<00:00, 201MB/s]
```

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

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

```
!ls train
```

```
cat.1947.jpg   cat.5161.jpg  cat.8377.jpg  dog.11591.jpg  dog.3556.jpg   dog.6771.jpg  dog.9987.jpg
cat.1948.jpg   cat.5162.jpg  cat.8378.jpg  dog.11592.jpg  dog.3557.jpg   dog.6772.jpg  dog.9988.jpg
cat.1949.jpg   cat.5163.jpg  cat.8379.jpg  dog.11593.jpg  dog.3558.jpg   dog.6773.jpg  dog.9989.jpg
cat.194.jpg    cat.5164.jpg  cat.837.jpg   dog.11594.jpg  dog.3559.jpg   dog.6774.jpg  dog.998.jpg
cat.1950.jpg   cat.5165.jpg  cat.8380.jpg  dog.11595.jpg  dog.355.jpg    dog.6775.jpg  dog.9990.jpg
cat.1951.jpg   cat.5166.jpg  cat.8381.jpg  dog.11596.jpg  dog.3560.jpg   dog.6776.jpg  dog.9991.jpg
cat.1952.jpg   cat.5167.jpg  cat.8382.jpg  dog.11597.jpg  dog.3561.jpg   dog.6777.jpg  dog.9992.jpg
cat.1953.jpg   cat.5168.jpg  cat.8383.jpg  dog.11598.jpg  dog.3562.jpg   dog.6778.jpg  dog.9993.jpg
cat.1954.jpg   cat.5169.jpg  cat.8384.jpg  dog.11599.jpg  dog.3563.jpg   dog.6779.jpg  dog.9994.jpg
cat.1955.jpg   cat.516.jpg   cat.8385.jpg  dog.1159.jpg   dog.3564.jpg   dog.677.jpg   dog.9995.jpg
cat.1956.jpg   cat.5170.jpg  cat.8386.jpg  dog.115.jpg    dog.3565.jpg   dog.6780.jpg  dog.9996.jpg
cat.1957.jpg   cat.5171.jpg  cat.8387.jpg  dog.11600.jpg  dog.3566.jpg   dog.6781.jpg  dog.9997.jpg
cat.1958.jpg   cat.5172.jpg  cat.8388.jpg  dog.11601.jpg  dog.3567.jpg   dog.6782.jpg  dog.9998.jpg
cat.1959.jpg   cat.5173.jpg  cat.8389.jpg  dog.11602.jpg  dog.3568.jpg   dog.6783.jpg  dog.9999.jpg
cat.195.jpg    cat.5174.jpg  cat.838.jpg   dog.11603.jpg  dog.3569.jpg   dog.6784.jpg  dog.999.jpg
cat.1960.jpg   cat.5175.jpg  cat.8390.jpg  dog.11604.jpg  dog.356.jpg    dog.6785.jpg  dog.99.jpg
cat.1961.jpg   cat.5176.jpg  cat.8391.jpg  dog.11605.jpg  dog.3570.jpg   dog.6786.jpg  dog.9.jpg
cat.1962.jpg   cat.5177.jpg  cat.8392.jpg  dog.11606.jpg  dog.3571.jpg   dog.6787.jpg
cat.1963.jpg   cat.5178.jpg  cat.8393.jpg  dog.11607.jpg  dog.3572.jpg   dog.6788.jpg
cat.1964.jpg   cat.5179.jpg  cat.8394.jpg  dog.11608.jpg  dog.3573.jpg   dog.6789.jpg
cat.1965.jpg   cat.517.jpg   cat.8395.jpg  dog.11609.jpg  dog.3574.jpg   dog.678.jpg
```

Question 1:

copying images to the test, validation, and training directories

```
import os, shutil, pathlib

original_dataset_dir = pathlib.Path("train")
base_dataset_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = base_dataset_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_dataset_dir / fname,
                            dst=dir / fname)


make_subset("train", start_index=667, end_index=1667)
make_subset("validation", start_index=1668, end_index=2168)
make_subset("test", start_index=2169, end_index=2669)
```

Loading and processing images with 'image_dataset_from_directory

```
from tensorflow.keras.utils import image_dataset_from_directory

train = image_dataset_from_directory(
    base_dataset_dir / "train",
    image_size=(180, 180),
    batch_size=32)

validation = image_dataset_from_directory(
    base_dataset_dir / "validation",
    image_size=(180, 180),
    batch_size=32)

test = image_dataset_from_directory(
    base_dataset_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

```
Found 2000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

Construct a dataset with 1000 instances, each having 16 random values.

```
import numpy as np
import tensorflow as tf
random_num = np.random.normal(size=(1000, 16))
data = tf.data.Dataset.from_tensor_slices(random_num)


for i, element in enumerate(data):
    print(element.shape)
```

```
    if i >= 2:
        break
```

```
(16,)
(16,)
(16,)
```

```
for i, element in enumerate(data):
    print(element.shape)
    if i >= 2:
        break
```

```
(16,)
(16,)
(16,)
```

```
reshapedata = data.map(lambda x: tf.reshape(x, (4, 4)))
for i, element in enumerate(reshapedata):
    print(element.shape)
    if i >= 2:
        break
```

```
(4, 4)
(4, 4)
(4, 4)
```

Building a small neural network to differentiate dog and cat images

```
for data_batch, labels_batch in train:
    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,)
```

```
from tensorflow import keras
from tensorflow.keras import layers

input = keras.Input(shape=(180, 180, 3))
l = layers.Rescaling(1./255)(input)
l = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(l)
l = layers.MaxPooling2D(pool_size=2)(l)
l = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(l)
l = layers.MaxPooling2D(pool_size=2)(l)
l = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(l)
l = layers.MaxPooling2D(pool_size=2)(l)
l = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(l)
l = layers.MaxPooling2D(pool_size=2)(l)
l = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(l)
l = layers.Flatten()(l)
l = layers.Dropout(0.5)(l)
output1 = layers.Dense(1, activation="sigmoid")(l)
model1 = keras.Model(inputs=input, outputs=output1)
```

preparing model for training

```
model1.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

The model is initially developed, followed by training on the training set. To evaluate the model's performance at each stage, we utilize the validation set.

```
model1.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 180, 180, 3)	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_1 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_2 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_3 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_4 (Conv2D)	(None, 7, 7, 256)	590,080
flatten (Flatten)	(None, 12544)	0
dropout (Dropout)	(None, 12544)	0
dense (Dense)	(None, 1)	12,545

The dataset is used to refine the model's parameters.

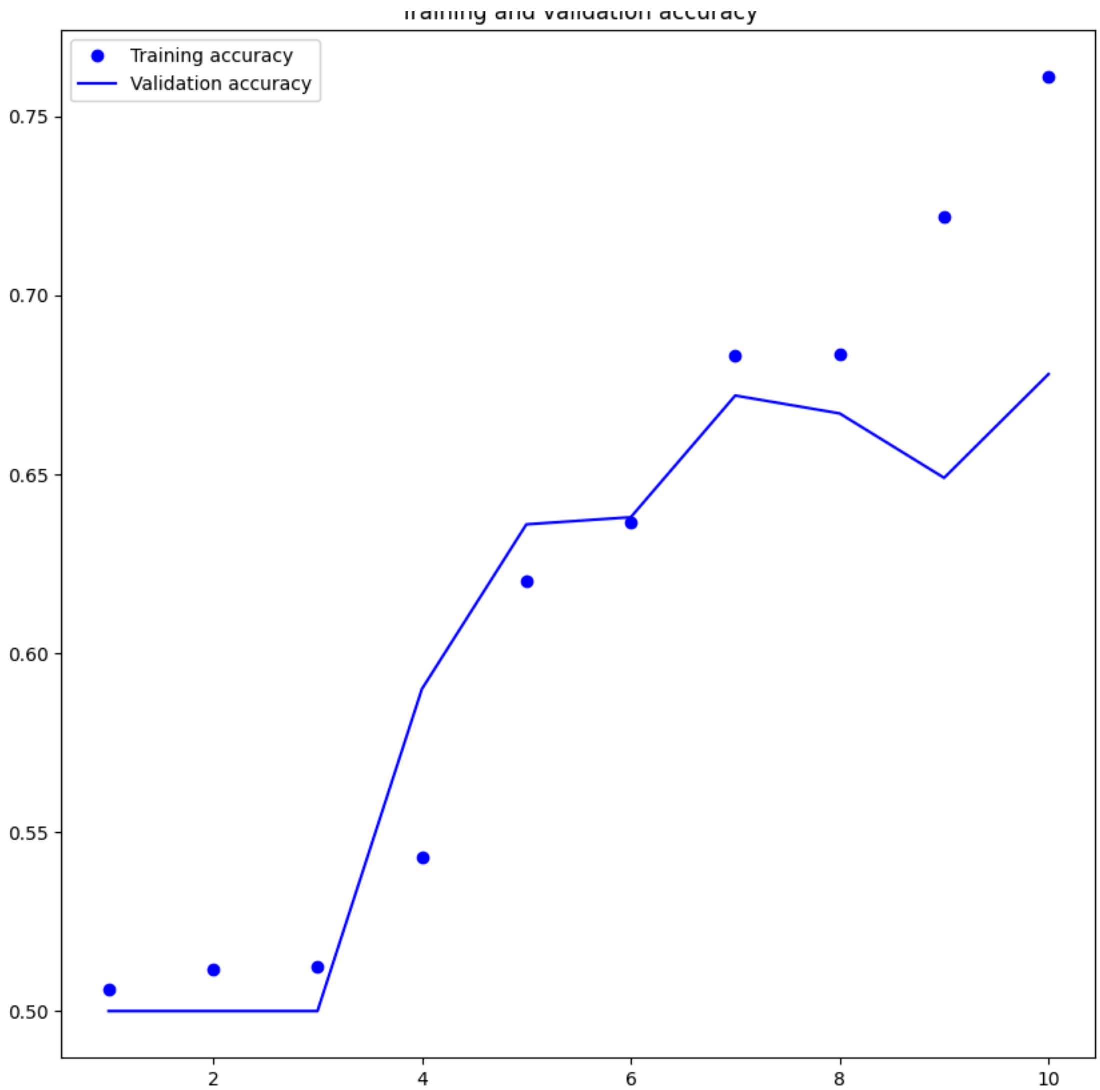
```
from keras.callbacks import ModelCheckpoint, EarlyStopping

callback1 = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history1 = model1.fit(
    train,
    epochs=10,
    validation_data=validation,
    callbacks=callback1)
```

```
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 184ms/step - accuracy: 0.5031 - loss: 0.6973 - val_accuracy: 0.5000 - val_loss: 0.6930
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 54ms/step - accuracy: 0.5136 - loss: 0.6933 - val_accuracy: 0.5000 - val_loss: 0.6934
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 116ms/step - accuracy: 0.4992 - loss: 0.6936 - val_accuracy: 0.5000 - val_loss: 0.6870
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 54ms/step - accuracy: 0.5295 - loss: 0.6878 - val_accuracy: 0.5900 - val_loss: 0.6750
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.6143 - loss: 0.6604 - val_accuracy: 0.6360 - val_loss: 0.7417
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 91ms/step - accuracy: 0.6240 - loss: 0.6626 - val_accuracy: 0.6380 - val_loss: 0.6314
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.6721 - loss: 0.6090 - val_accuracy: 0.6720 - val_loss: 0.6209
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 57ms/step - accuracy: 0.6850 - loss: 0.5729 - val_accuracy: 0.6670 - val_loss: 0.6366
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 77ms/step - accuracy: 0.7079 - loss: 0.5611 - val_accuracy: 0.6490 - val_loss: 0.6265
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 68ms/step - accuracy: 0.7556 - loss: 0.4970 - val_accuracy: 0.6780 - val_loss: 0.5893
```

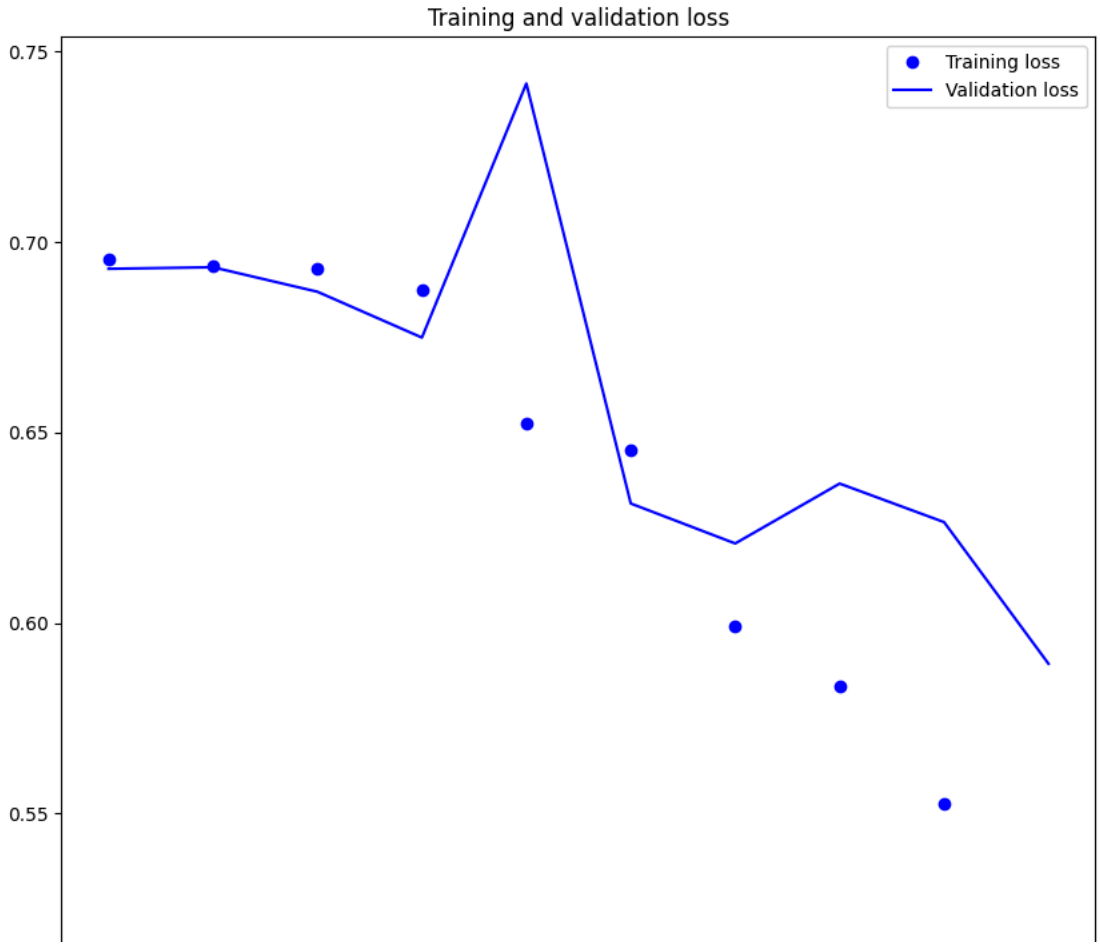
To visualize the model's performance over time, training curves for accuracy and loss were created.

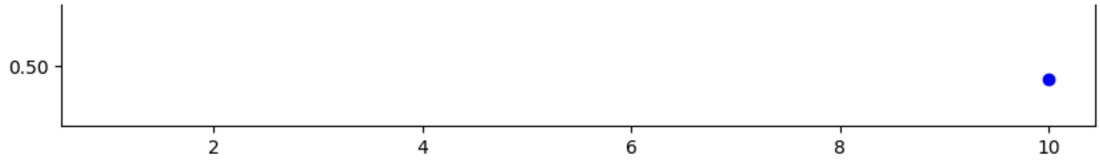
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
accuracy1 = history1.history["accuracy"]
val_accuracy1 = history1.history["val_accuracy"]
loss1 = history1.history["loss"]
val_loss1 = history1.history["val_loss"]
epochs = range(1, len(accuracy1) + 1)
plt.plot(epochs, accuracy1, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy1, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
```

```
plt.legend()
plt.figure()
plt.figure(figsize=(10, 10))
plt.plot(epochs, loss1, "bo", label="Training loss")
plt.plot(epochs, val_loss1, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```



<Figure size 640x480 with 0 Axes>






```
testacc1 = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = testacc1.evaluate(test)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.6912 - loss: 0.6092
Test accuracy: 0.696
```

According to the above result, the test accuracy without data augmentation is about 69.3%, while the training accuracy is about 92%.

Question 2:

Expanding image dataset: Adding data augmentation to an image model.

```
import os, shutil, pathlib

shutil.rmtree("./cats_vs_dogs_small_Q2", ignore_errors=True)

# Defining the original directory and the new base directory
original_dataset_dir = pathlib.Path("train")
base_dataset_dir = pathlib.Path("cats_vs_dogs_small_Q2")

# Functions to create subsets
def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = base_dataset_dir / subset_name / category
        os.makedirs(dir, exist_ok=True)  # Create directory, if it doesn't exist
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dataset_dir / fname,
                            dst=dir / fname)

# Creating subsets for training, validation, and testing
make_subset("train", start_index=667, end_index=2167)  # 1500 samples
make_subset("validation", start_index=2168, end_index=2668)  # 500 samples
make_subset("test", start_index=2669, end_index=3168)  # 500 samples
```

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

showing the training augmented pictures

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

Developing a new convolutional neural network that includes picture augmentation and dropout

```
input2 = keras.Input(shape=(180, 180, 3))
m = augmentation(input2)
m = layers.Rescaling(1./255)(m)
m = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(m)
m = layers.MaxPooling2D(pool_size=2)(m)
m = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(m)
m = layers.MaxPooling2D(pool_size=2)(m)
m = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(m)
m = layers.MaxPooling2D(pool_size=2)(m)
m = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(m)
m = layers.MaxPooling2D(pool_size=2)(m)
m = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(m)
m = layers.Flatten()(m)
m = layers.Dropout(0.5)(m)
output2 = layers.Dense(1, activation="sigmoid")(m)
model2 = keras.Model(inputs=input2, outputs=output2)

model2.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])


from keras.callbacks import ModelCheckpoint, EarlyStopping
callback2 = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history2 = model2.fit(
    train,
    epochs=30,
    validation_data=validation,
    callbacks=callback2)
```

```
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - accuracy: 0.5872 - loss: 0.6876 - val_accuracy: 0.5860 - val_loss: 0.6727
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 79ms/step - accuracy: 0.5848 - loss: 0.6783 - val_accuracy: 0.5700 - val_loss: 0.6988
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 76ms/step - accuracy: 0.5855 - loss: 0.6694 - val_accuracy: 0.5890 - val_loss: 0.6667
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 57ms/step - accuracy: 0.6083 - loss: 0.6659 - val_accuracy: 0.6210 - val_loss: 0.6334
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 66ms/step - accuracy: 0.6216 - loss: 0.6427 - val_accuracy: 0.5840 - val_loss: 0.7851
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 90ms/step - accuracy: 0.6165 - loss: 0.6676 - val_accuracy: 0.6360 - val_loss: 0.6398
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 56ms/step - accuracy: 0.6641 - loss: 0.6296 - val_accuracy: 0.5910 - val_loss: 0.6497
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 100ms/step - accuracy: 0.6538 - loss: 0.6308 - val_accuracy: 0.6670 - val_loss: 0.6010
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.6763 - loss: 0.6117 - val_accuracy: 0.6930 - val_loss: 0.5917
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.6813 - loss: 0.6052 - val_accuracy: 0.6810 - val_loss: 0.5914
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 110ms/step - accuracy: 0.6939 - loss: 0.5980 - val_accuracy: 0.6860 - val_loss: 0.6072
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.6832 - loss: 0.5964 - val_accuracy: 0.7400 - val_loss: 0.5555
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.6992 - loss: 0.5724 - val_accuracy: 0.7200 - val_loss: 0.5565
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 79ms/step - accuracy: 0.7026 - loss: 0.5705 - val_accuracy: 0.7210 - val_loss: 0.5645
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 74ms/step - accuracy: 0.7186 - loss: 0.5491 - val_accuracy: 0.7280 - val_loss: 0.5445
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.7306 - loss: 0.5460 - val_accuracy: 0.7080 - val_loss: 0.5614
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 57ms/step - accuracy: 0.7608 - loss: 0.5176 - val_accuracy: 0.7370 - val_loss: 0.5366
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 86ms/step - accuracy: 0.7403 - loss: 0.5325 - val_accuracy: 0.7180 - val_loss: 0.5812
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.7578 - loss: 0.5031 - val_accuracy: 0.7070 - val_loss: 0.6345
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.7689 - loss: 0.4817 - val_accuracy: 0.7600 - val_loss: 0.5171
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 104ms/step - accuracy: 0.7666 - loss: 0.4876 - val_accuracy: 0.7020 - val_loss: 0.6346
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 57ms/step - accuracy: 0.7409 - loss: 0.5121 - val_accuracy: 0.7570 - val_loss: 0.5096
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.7731 - loss: 0.4641 - val_accuracy: 0.7510 - val_loss: 0.5242
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 86ms/step - accuracy: 0.7701 - loss: 0.4972 - val_accuracy: 0.7740 - val_loss: 0.4980
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.7715 - loss: 0.4768 - val_accuracy: 0.7770 - val_loss: 0.4627
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.7867 - loss: 0.4424 - val_accuracy: 0.7780 - val_loss: 0.4951
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 60ms/step - accuracy: 0.7972 - loss: 0.4311 - val_accuracy: 0.7790 - val_loss: 0.5013
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 108ms/step - accuracy: 0.8128 - loss: 0.4023 - val_accuracy: 0.7850 - val_loss: 0.4623
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.8118 - loss: 0.3877 - val_accuracy: 0.7770 - val_loss: 0.4906
```

Model evaluated based on test set

```
testacc2 = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = testacc2.evaluate(test)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.7836 - loss: 0.4762
Test accuracy: 0.804
```

Question 3:

In the third step, training sets of 2000 samples were employed, with 500 samples reserved for validation and testing. I discovered that the test accuracy was superior with 1500 training samples compared to 1000 or 2000. Moreover, training accuracy improved with 1000 training samples. Increasing the training set to 2000 while maintaining the same validation and testing sets resulted in these findings.

```
original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_Q3")

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)
#Creating training, Test and validation sets.
#Training has 2000 samples, test has 500 samples and validation has 500 samples.
make_subset("train", start_index=667, end_index=2667)
make_subset("validation", start_index=2668, end_index=3168)
make_subset("test", start_index=3169, end_index=3669)
```

Click enter to edit the data

```
i3 = keras.Input(shape=(180, 180, 3))
n = augmentation(i3)
n = layers.Rescaling(1./255)(n)
n = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(n)
n = layers.MaxPooling2D(pool_size=2)(n)
n = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(n)
n = layers.MaxPooling2D(pool_size=2)(n)
n = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(n)
n = layers.MaxPooling2D(pool_size=2)(n)
n = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(n)
n = layers.MaxPooling2D(pool_size=2)(n)
n = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(n)
n = layers.Flatten()(n)
n = layers.Dropout(0.5)(n)
out3 = layers.Dense(1, activation="sigmoid")(n)
mod3 = keras.Model(inputs=i3, outputs=out3)

mod3.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])


callback3 = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation1.keras",
        save_best_only=True,
        monitor="val_loss")
]
hist3 = mod3.fit(
    train,
    epochs=50,
    validation_data=validation,
    callbacks=callback3)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.8102 - loss: 0.4077 - val_accuracy: 0.7780 - val_loss: 0.4875
Epoch 42/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 91ms/step - accuracy: 0.8120 - loss: 0.4207 - val_accuracy: 0.7660 - val_loss: 0.6103
Epoch 43/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 55ms/step - accuracy: 0.7999 - loss: 0.4209 - val_accuracy: 0.7660 - val_loss: 0.5525
Epoch 44/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 87ms/step - accuracy: 0.7978 - loss: 0.4250 - val_accuracy: 0.7970 - val_loss: 0.4709
Epoch 45/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.8144 - loss: 0.3998 - val_accuracy: 0.7780 - val_loss: 0.4901
Epoch 46/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.8202 - loss: 0.4176 - val_accuracy: 0.7800 - val_loss: 0.4884
Epoch 47/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.8189 - loss: 0.3940 - val_accuracy: 0.7680 - val_loss: 0.5483
Epoch 48/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 87ms/step - accuracy: 0.8140 - loss: 0.3998 - val_accuracy: 0.7740 - val_loss: 0.4856
Epoch 49/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 58ms/step - accuracy: 0.8286 - loss: 0.3950 - val_accuracy: 0.7920 - val_loss: 0.4691
Epoch 50/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 86ms/step - accuracy: 0.8364 - loss: 0.3605 - val_accuracy: 0.7960 - val_loss: 0.4677
```

```python
acc_test3 = keras.models.load_model(
    "convnet_from_scratch_with_augmentation1.keras")
test_loss, test_acc = acc_test3.evaluate(test)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.8035 - loss: 0.4668
Test accuracy: 0.805
```

Question 4:

Creating the VGG16 convolutional base

```python
convolution_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_no
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```python
convolution_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 180, 180, 3)	0
block1_conv1 (Conv2D)	(None, 180, 180, 64)	1,792
block1_conv2 (Conv2D)	(None, 180, 180, 64)	36,928
block1_pool (MaxPooling2D)	(None, 90, 90, 64)	0
block2_conv1 (Conv2D)	(None, 90, 90, 128)	73,856
block2_conv2 (Conv2D)	(None, 90, 90, 128)	147,584
block2_pool (MaxPooling2D)	(None, 45, 45, 128)	0
block3_conv1 (Conv2D)	(None, 45, 45, 256)	295,168
block3_conv2 (Conv2D)	(None, 45, 45, 256)	590,080
block3_conv3 (Conv2D)	(None, 45, 45, 256)	590,080
block3_pool (MaxPooling2D)	(None, 22, 22, 256)	0
block4_conv1 (Conv2D)	(None, 22, 22, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 11, 11, 512)	0
block5_conv1 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0

Utilizing a pre-trained model for feature extraction without data augmentation: Obtaining the labels associated with the VGG16 characteristics

```
import numpy as np

def get_features_and_labels(dataset):
    all_feature = []
    all_label = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = convolution_base.predict(preprocessed_images)
        all_feature.append(features)
        all_label.append(labels)
    return np.concatenate(all_feature), np.concatenate(all_label)

train_features, train_labels =  get_features_and_labels(train)
val_features, val_labels =  get_features_and_labels(validation)
test_features, test_labels =  get_features_and_labels(test)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
```

```
train_features.shape
```

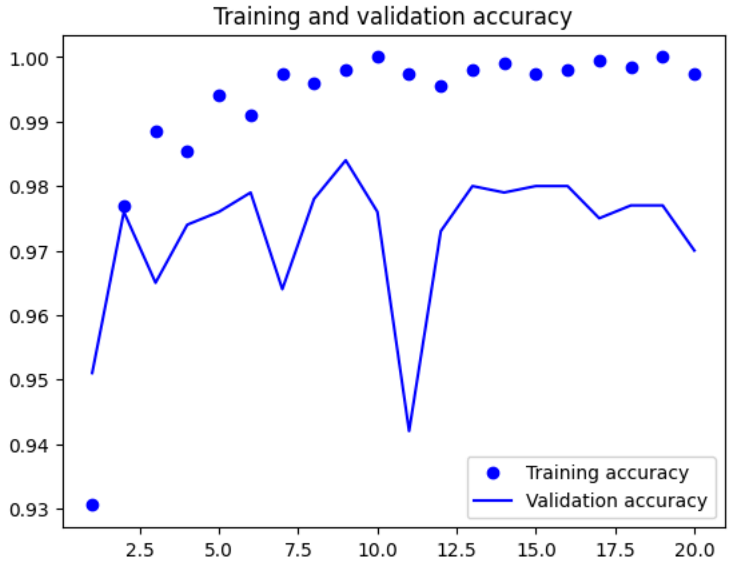
```
(2000, 5, 5, 512)
```

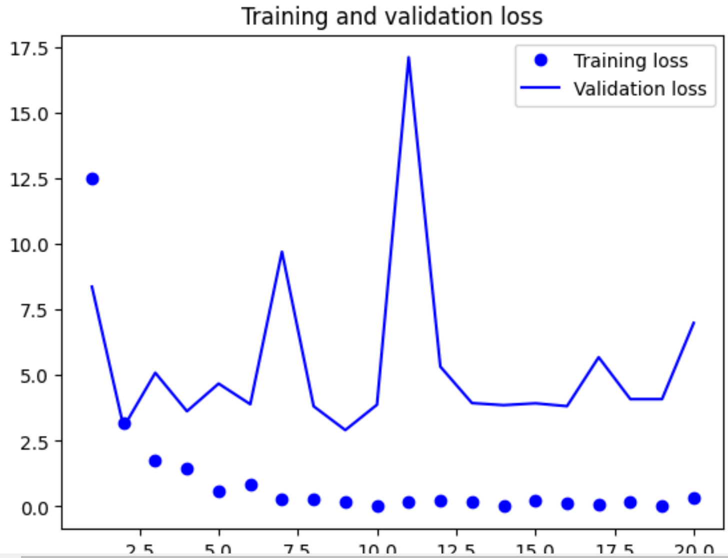
```
i6 = keras.Input(shape=(5, 5, 512))
p = layers.Flatten()(i6)
p = layers.Dense(256)(p)
p = layers.Dropout(0.5)(p)
```

```
out6 = layers.Dense(1, activation="sigmoid")(p)
m6 = keras.Model(i6, out6)
m6.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])

callback6 = [
    keras.callbacks.ModelCheckpoint(
      filepath="feature_extraction.keras",
      save_best_only=True,
      monitor="val_loss")
]
hist6 = m6.fit(
    train_features, train_labels,
    epochs=20,
    validation_data=(val_features, val_labels),
    callbacks=callback6)
```

```
Epoch 1/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 65ms/step - accuracy: 0.8700 - loss: 22.0649 - val_accuracy: 0.9510 - val_loss: 8.3680
Epoch 2/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.9761 - loss: 3.0818 - val_accuracy: 0.9760 - val_loss: 3.0220
Epoch 3/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9888 - loss: 1.9675 - val_accuracy: 0.9650 - val_loss: 5.0891
Epoch 4/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9855 - loss: 1.1799 - val_accuracy: 0.9740 - val_loss: 3.6258
Epoch 5/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9903 - loss: 1.4614 - val_accuracy: 0.9760 - val_loss: 4.6777
Epoch 6/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9953 - loss: 0.4865 - val_accuracy: 0.9790 - val_loss: 3.8926
Epoch 7/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9983 - loss: 0.1459 - val_accuracy: 0.9640 - val_loss: 9.6983
Epoch 8/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9961 - loss: 0.1508 - val_accuracy: 0.9780 - val_loss: 3.8151
Epoch 9/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9989 - loss: 0.0654 - val_accuracy: 0.9840 - val_loss: 2.9081
Epoch 10/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 3.4632e-07 - val_accuracy: 0.9760 - val_loss: 3.8735
Epoch 11/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9977 - loss: 0.0622 - val_accuracy: 0.9420 - val_loss: 17.1012
Epoch 12/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9924 - loss: 0.4070 - val_accuracy: 0.9730 - val_loss: 5.3256
Epoch 13/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9974 - loss: 0.1862 - val_accuracy: 0.9800 - val_loss: 3.9368
Epoch 14/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0014 - val_accuracy: 0.9790 - val_loss: 3.8575
Epoch 15/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9979 - loss: 0.2054 - val_accuracy: 0.9800 - val_loss: 3.9280
Epoch 16/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9993 - loss: 0.0455 - val_accuracy: 0.9800 - val_loss: 3.8181
Epoch 17/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9999 - loss: 0.0176 - val_accuracy: 0.9750 - val_loss: 5.6811
Epoch 18/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9983 - loss: 0.3113 - val_accuracy: 0.9770 - val_loss: 4.0865
Epoch 19/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 1.6221e-09 - val_accuracy: 0.9770 - val_loss: 4.0863
Epoch 20/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9978 - loss: 0.3153 - val_accuracy: 0.9700 - val_loss: 6.9854
```

```
import matplotlib.pyplot as plt
accuracy6 = hist6.history["accuracy"]
valaccuracy6 = hist6.history["val_accuracy"]
los6 = hist6.history["loss"]
vallos6 = hist6.history["val_loss"]
epochs = range(1, len(accuracy6) + 1)
plt.plot(epochs, accuracy6, "bo", label="Training accuracy")
plt.plot(epochs, valaccuracy6, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, los6, "bo", label="Training loss")
plt.plot(epochs, vallos6, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```





VGG16 convolutional base instantiation and freezing

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

convolution_base.trainable = True
print("This is the number of trainable weights "
      "before freezing the conv base:", len(convolution_base.trainable_weights))

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

```
This is the number of trainable weights before freezing the conv base: 26
This is the number of trainable weights after freezing the conv base: 0
```

Model is now performing with a classifier and agumentation to convulation base

```
augmentation2 = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

input22 = keras.Input(shape=(180, 180, 3))
x1 = augmentation2(input22)
x1 =keras.layers.Lambda(
     lambda x: keras.applications.vgg16.preprocess_input(x))(x1)
```

```
x1 = convolution_base(x1)
x1 = layers.Flatten()(x1)
x1 = layers.Dense(256)(x1)
x1 = layers.Dropout(0.5)(x1)
outputs = layers.Dense(1, activation="sigmoid")(x1)
model = keras.Model(input22, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])



callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="features_extraction_with_augmentation2.keras",
        save_best_only=True,
        monitor="val_loss"
    )
]

history = model.fit(
    train,
    epochs=10,
    validation_data=validation,
    callbacks=callbacks
)
```

```
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 235ms/step - accuracy: 0.8239 - loss: 51.9657 - val_accuracy: 0.9670 - val_loss: 3.6817
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 188ms/step - accuracy: 0.9517 - loss: 5.4933 - val_accuracy: 0.9740 - val_loss: 2.9366
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 170ms/step - accuracy: 0.9477 - loss: 6.5372 - val_accuracy: 0.9330 - val_loss: 14.2858
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 171ms/step - accuracy: 0.9493 - loss: 5.2048 - val_accuracy: 0.9750 - val_loss: 3.7424
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 173ms/step - accuracy: 0.9598 - loss: 4.7611 - val_accuracy: 0.9760 - val_loss: 4.9396
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 177ms/step - accuracy: 0.9612 - loss: 5.7594 - val_accuracy: 0.9750 - val_loss: 3.6142
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 184ms/step - accuracy: 0.9706 - loss: 2.5897 - val_accuracy: 0.9790 - val_loss: 2.3751
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 175ms/step - accuracy: 0.9781 - loss: 2.2594 - val_accuracy: 0.9690 - val_loss: 6.5185
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 196ms/step - accuracy: 0.9792 - loss: 2.3329 - val_accuracy: 0.9740 - val_loss: 2.5654
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 174ms/step - accuracy: 0.9765 - loss: 2.9532 - val_accuracy: 0.9780 - val_loss: 2.8742
```

```
!ls -lh features_extraction_with_augmentation2.keras
```

```
-rw-r--r-- 1 root root 82M Oct 21 03:11 features_extraction_with_augmentation2.keras
```

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import vgg16

# Define the model
augmentation2 = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

input22 = keras.Input(shape=(180, 180, 3))

a1 = augmentation2(input22)

# Specify output_shape for Lambda layer
a1 = keras.layers.Lambda(
    lambda x: vgg16.preprocess_input(x),
    output_shape=(180, 180, 3)
)(a1)

a1 = convolution_base(a1)
a1 = layers.Flatten()(a1)
a1 = layers.Dense(256)(a1)
a1 = layers.Dropout(0.5)(a1)
outputs = layers.Dense(1, activation="sigmoid")(a1)

model = keras.Model(input22, outputs)
```

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

# Save the model
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="features_extraction_with_augmentation2.keras",
        save_best_only=True,
        monitor="val_loss"
    )
]

history = model.fit(
    train,
    epochs=10,
    validation_data=validation,
    callbacks=callbacks
)
```

```
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 194ms/step - accuracy: 0.8234 - loss: 37.3641 - val_accuracy: 0.8580 - val_loss: 31.6120
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 211ms/step - accuracy: 0.9230 - loss: 9.7153 - val_accuracy: 0.9540 - val_loss: 7.9000
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 182ms/step - accuracy: 0.9567 - loss: 5.6781 - val_accuracy: 0.9600 - val_loss: 6.5357
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 184ms/step - accuracy: 0.9611 - loss: 5.5948 - val_accuracy: 0.9790 - val_loss: 3.2763
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 175ms/step - accuracy: 0.9664 - loss: 3.7052 - val_accuracy: 0.9760 - val_loss: 4.2968
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 212ms/step - accuracy: 0.9685 - loss: 2.8560 - val_accuracy: 0.9820 - val_loss: 2.6743
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 208ms/step - accuracy: 0.9705 - loss: 3.5364 - val_accuracy: 0.9790 - val_loss: 2.5737
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 173ms/step - accuracy: 0.9714 - loss: 2.5616 - val_accuracy: 0.9820 - val_loss: 2.7265
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 186ms/step - accuracy: 0.9678 - loss: 3.2532 - val_accuracy: 0.9850 - val_loss: 2.0829
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 199ms/step - accuracy: 0.9805 - loss: 2.2020 - val_accuracy: 0.9830 - val_loss: 2.1428
```

Fine-tuning a pretrained model

Freezing all layers until the fourth from the last

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


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

callbackstu = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning.keras",
        save_best_only=True,
        monitor="val_loss")
]
historytune = model.fit(
    train,
    epochs=30,
    validation_data=validation,
    callbacks=callbackstu)
```

```
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 204ms/step - accuracy: 0.9825 - loss: 0.8490 - val_accuracy: 0.9810 - val_loss: 1.9390
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 213ms/step - accuracy: 0.9822 - loss: 0.9897 - val_accuracy: 0.9850 - val_loss: 1.8810
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 193ms/step - accuracy: 0.9893 - loss: 0.7837 - val_accuracy: 0.9810 - val_loss: 2.3771
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 194ms/step - accuracy: 0.9859 - loss: 0.7598 - val_accuracy: 0.9790 - val_loss: 1.9564
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 201ms/step - accuracy: 0.9833 - loss: 0.7961 - val_accuracy: 0.9830 - val_loss: 1.6876
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 214ms/step - accuracy: 0.9929 - loss: 0.2419 - val_accuracy: 0.9800 - val_loss: 2.5692
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 199ms/step - accuracy: 0.9874 - loss: 0.5872 - val_accuracy: 0.9830 - val_loss: 1.5809
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 207ms/step - accuracy: 0.9840 - loss: 0.4906 - val_accuracy: 0.9820 - val_loss: 1.4617
```