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

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

Choose Files kaggle.json

- **kaggle.json**(application/json) - 72 bytes, last modified: 10/18/2024 - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"tarunkumarkorimi","key":"73169c2fe4d4266b6145222efdb63bee"}'}
```

```
!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 [00:06<00:00, 225MB/s]
100% 812M/812M [00:06<00:00, 131MB/s]
```

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

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

```
!ls train
```

```
cat.1745.jpg   cat.4960.jpg  cat.8175.jpg  dog.1138.jpg   dog.3354.jpg   dog.656.jpg   dog.9785.jpg
cat.1746.jpg   cat.4961.jpg  cat.8176.jpg  dog.11390.jpg  dog.3355.jpg   dog.6570.jpg  dog.9786.jpg
cat.1747.jpg   cat.4962.jpg  cat.8177.jpg  dog.11391.jpg  dog.3356.jpg   dog.6571.jpg  dog.9787.jpg
cat.1748.jpg   cat.4963.jpg  cat.8178.jpg  dog.11392.jpg  dog.3357.jpg   dog.6572.jpg  dog.9788.jpg
cat.1749.jpg   cat.4964.jpg  cat.8179.jpg  dog.11393.jpg  dog.3358.jpg   dog.6573.jpg  dog.9789.jpg
cat.174.jpg    cat.4965.jpg  cat.817.jpg   dog.11394.jpg  dog.3359.jpg   dog.6574.jpg  dog.978.jpg
cat.1750.jpg   cat.4966.jpg  cat.8180.jpg  dog.11395.jpg  dog.335.jpg    dog.6575.jpg  dog.9790.jpg
cat.1751.jpg   cat.4967.jpg  cat.8181.jpg  dog.11396.jpg  dog.3360.jpg   dog.6576.jpg  dog.9791.jpg
cat.1752.jpg   cat.4968.jpg  cat.8182.jpg  dog.11397.jpg  dog.3361.jpg   dog.6577.jpg  dog.9792.jpg
cat.1753.jpg   cat.4969.jpg  cat.8183.jpg  dog.11398.jpg  dog.3362.jpg   dog.6578.jpg  dog.9793.jpg
cat.1754.jpg   cat.496.jpg   cat.8184.jpg  dog.11399.jpg  dog.3363.jpg   dog.6579.jpg  dog.9794.jpg
cat.1755.jpg   cat.4970.jpg  cat.8185.jpg  dog.1139.jpg   dog.3364.jpg   dog.657.jpg   dog.9795.jpg
cat.1756.jpg   cat.4971.jpg  cat.8186.jpg  dog.113.jpg    dog.3365.jpg   dog.6580.jpg  dog.9796.jpg
cat.1757.jpg   cat.4972.jpg  cat.8187.jpg  dog.11400.jpg  dog.3366.jpg   dog.6581.jpg  dog.9797.jpg
cat.1758.jpg   cat.4973.jpg  cat.8188.jpg  dog.11401.jpg  dog.3367.jpg   dog.6582.jpg  dog.9798.jpg
cat.1759.jpg   cat.4974.jpg  cat.8189.jpg  dog.11402.jpg  dog.3368.jpg   dog.6583.jpg  dog.9799.jpg
cat.175.jpg    cat.4975.jpg  cat.818.jpg   dog.11403.jpg  dog.3369.jpg   dog.6584.jpg  dog.979.jpg
cat.1760.jpg   cat.4976.jpg  cat.8190.jpg  dog.11404.jpg  dog.336.jpg    dog.6585.jpg  dog.97.jpg
cat.1761.jpg   cat.4977.jpg  cat.8191.jpg  dog.11405.jpg  dog.3370.jpg   dog.6586.jpg  dog.9800.jpg
cat.1762.jpg   cat.4978.jpg  cat.8192.jpg  dog.11406.jpg  dog.3371.jpg   dog.6587.jpg  dog.9801.jpg
cat.1763.jpg   cat.4979.jpg  cat.8193.jpg  dog.11407.jpg  dog.3372.jpg   dog.6588.jpg  dog.9802.jpg
cat.1764.jpg   cat.497.jpg   cat.8194.jpg  dog.11408.jpg  dog.3373.jpg   dog.6589.jpg  dog.9803.jpg
cat.1765.jpg   cat.4980.jpg  cat.8195.jpg  dog.11409.jpg  dog.3374.jpg   dog.658.jpg   dog.9804.jpg
cat.1766.jpg   cat.4981.jpg  cat.8196.jpg  dog.1140.jpg   dog.3375.jpg   dog.6590.jpg  dog.9805.jpg
```

Question 1: 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?

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)
```

Interpreting 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.
```

Create a dataset instance with 1000 random samples, each with a vector size of 16 using a NumPy array.

```
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)
```

Developing the model

Creating a tiny network for categorizing dogs versus cats

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

preparing model for training

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

The model is constructed at first, then it is then trained using the training dataset. We use the validation dataset to verify the model's performance at the end of each phase. I'm utilizing a GPU to reduce the processing length of each phase.

```
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

Total params: 991,041 (3.78 MB)
Trainable params: 991,041 (3.78 MB)

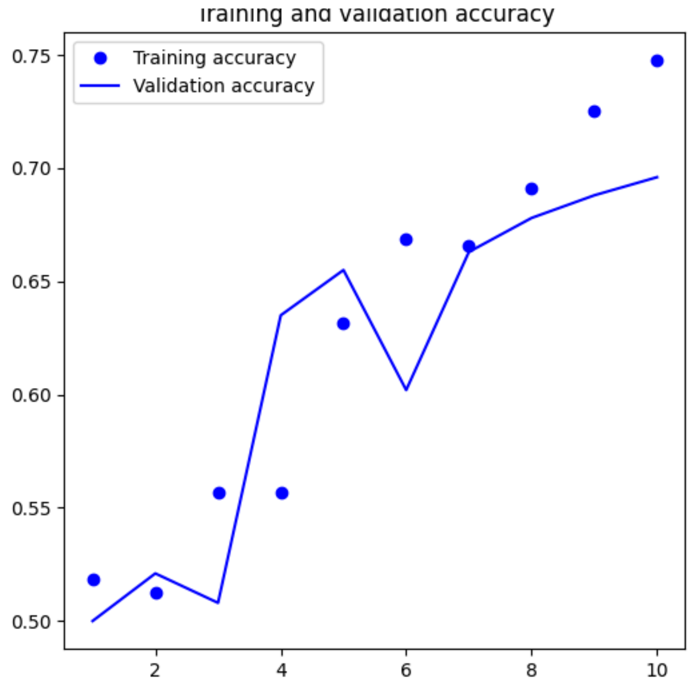
Model fitting follows using the dataset.

```
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)
```

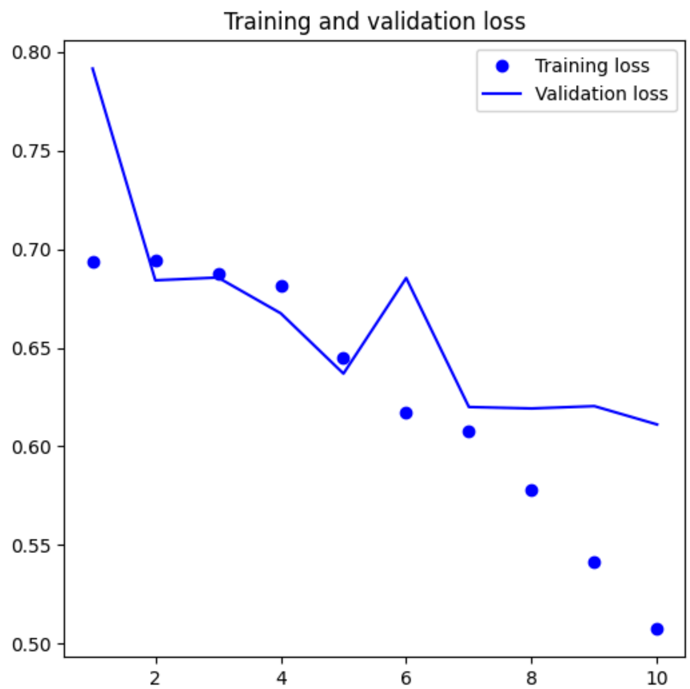
```
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 192ms/step - accuracy: 0.5157 - loss: 0.6956 - val_accuracy: 0.5000 - val_loss: 0.7916
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.5065 - loss: 0.6976 - val_accuracy: 0.5210 - val_loss: 0.6842
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 85ms/step - accuracy: 0.5519 - loss: 0.6865 - val_accuracy: 0.5080 - val_loss: 0.6856
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.5399 - loss: 0.6886 - val_accuracy: 0.6350 - val_loss: 0.6675
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 52ms/step - accuracy: 0.6342 - loss: 0.6491 - val_accuracy: 0.6550 - val_loss: 0.6369
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.6735 - loss: 0.6173 - val_accuracy: 0.6020 - val_loss: 0.6854
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 74ms/step - accuracy: 0.6612 - loss: 0.6070 - val_accuracy: 0.6630 - val_loss: 0.6200
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 52ms/step - accuracy: 0.6830 - loss: 0.5747 - val_accuracy: 0.6780 - val_loss: 0.6192
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - accuracy: 0.7219 - loss: 0.5465 - val_accuracy: 0.6880 - val_loss: 0.6204
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 99ms/step - accuracy: 0.7524 - loss: 0.5031 - val_accuracy: 0.6960 - val_loss: 0.6111
```

In order to improve visualization and understanding, training curves for accuracy and loss were created.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 6))
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=(6, 6))
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 38ms/step - accuracy: 0.6967 - loss: 0.5816
Test accuracy: 0.685
```

According to the above result, the test accuracy without data augmentation is about 69.7%, while the training accuracy is about 92%.

Question 2: 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?

Using data augmentation

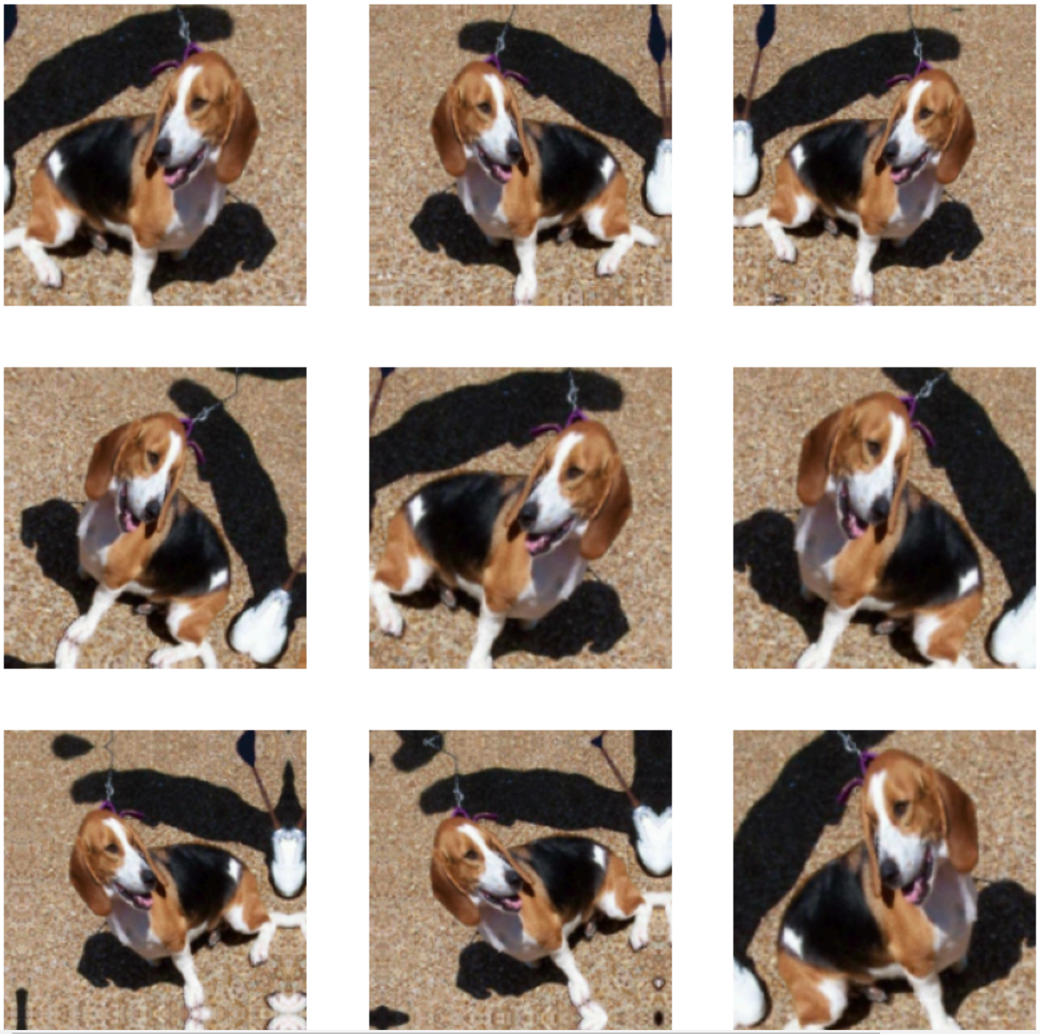
Define a data augmentation stage to add to an image model

```
import os, shutil, pathlib

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

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

# Function 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))
b = augmentation(input2)
b = layers.Rescaling(1./255)(b)
b = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(b)
b = layers.MaxPooling2D(pool_size=2)(b)
b = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(b)
b = layers.MaxPooling2D(pool_size=2)(b)
b = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(b)
b = layers.MaxPooling2D(pool_size=2)(b)
b = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(b)
b = layers.MaxPooling2D(pool_size=2)(b)
b = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(b)
b = layers.Flatten()(b)
b = layers.Dropout(0.5)(b)
output2 = layers.Dense(1, activation="sigmoid")(b)
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 ━━━━━━━━━━━━━━━━━━━━ 8s 85ms/step - accuracy: 0.5091 - loss: 0.6931 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.5137 - loss: 0.6931 - val_accuracy: 0.5850 - val_loss: 0.6930
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 61ms/step - accuracy: 0.4659 - loss: 0.6934 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 93ms/step - accuracy: 0.4992 - loss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6919
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 54ms/step - accuracy: 0.4911 - loss: 0.6944 - val_accuracy: 0.5070 - val_loss: 0.6930
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 84ms/step - accuracy: 0.5000 - loss: 0.6952 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 54ms/step - accuracy: 0.4751 - loss: 0.6940 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 81ms/step - accuracy: 0.5186 - loss: 0.6931 - val_accuracy: 0.5000 - val_loss: 0.6930
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.5265 - loss: 0.6924 - val_accuracy: 0.5080 - val_loss: 0.6930
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 55ms/step - accuracy: 0.4855 - loss: 0.6937 - val_accuracy: 0.5220 - val_loss: 0.6879
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 66ms/step - accuracy: 0.5354 - loss: 0.6858 - val_accuracy: 0.5580 - val_loss: 0.6846
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 87ms/step - accuracy: 0.5550 - loss: 0.6854 - val_accuracy: 0.5140 - val_loss: 0.6910
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.5044 - loss: 0.6894 - val_accuracy: 0.5010 - val_loss: 0.6937
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.5189 - loss: 0.6939 - val_accuracy: 0.5000 - val_loss: 0.6933
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.4874 - loss: 0.6938 - val_accuracy: 0.5480 - val_loss: 0.6886
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 67ms/step - accuracy: 0.5370 - loss: 0.6862 - val_accuracy: 0.5300 - val_loss: 0.6867
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.5559 - loss: 0.6860 - val_accuracy: 0.5350 - val_loss: 0.6855
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 82ms/step - accuracy: 0.5664 - loss: 0.6864 - val_accuracy: 0.5320 - val_loss: 0.6862
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 71ms/step - accuracy: 0.5665 - loss: 0.6800 - val_accuracy: 0.5730 - val_loss: 0.6773
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 60ms/step - accuracy: 0.5831 - loss: 0.6809 - val_accuracy: 0.5010 - val_loss: 0.6965
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 66ms/step - accuracy: 0.5128 - loss: 0.6901 - val_accuracy: 0.5640 - val_loss: 0.6682
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 76ms/step - accuracy: 0.6137 - loss: 0.6637 - val_accuracy: 0.6380 - val_loss: 0.6436
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 61ms/step - accuracy: 0.6475 - loss: 0.6302 - val_accuracy: 0.5950 - val_loss: 0.6573
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 83ms/step - accuracy: 0.6370 - loss: 0.6427 - val_accuracy: 0.6280 - val_loss: 0.6514
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 56ms/step - accuracy: 0.6459 - loss: 0.6317 - val_accuracy: 0.6410 - val_loss: 0.6371
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 84ms/step - accuracy: 0.6710 - loss: 0.6090 - val_accuracy: 0.6870 - val_loss: 0.6129
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.6640 - loss: 0.6261 - val_accuracy: 0.6030 - val_loss: 0.7129
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 55ms/step - accuracy: 0.6841 - loss: 0.6004 - val_accuracy: 0.6400 - val_loss: 0.6628
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 87ms/step - accuracy: 0.6710 - loss: 0.6029 - val_accuracy: 0.6370 - val_loss: 0.6772
```

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: 0.6929 - loss: 0.5904
Test accuracy: 0.684
```

Question 3: Now change your training sample so that you achieve better performance than those from Steps 1 and 2. This sample size may be larger, or smaller than those in the previous steps. The objective is to find the ideal training sample size to get best prediction results.

In step three, test sets of 2000 training samples with validation and 500 samples were used. I've discovered that test accuracy is higher with 1500 photos than with training samples of 1000 and 2000 photos.

Training accuracy increases with 1000 training samples.

Increasing the training sample to 2000 while keeping the test and validation sets at 500

```
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)
```

Double-click (or enter) to edit

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

```
Epoch 31/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 87ms/step - accuracy: 0.7721 - loss: 0.4790 - val_accuracy: 0.7870 - val_loss: 0.4716
Epoch 32/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 55ms/step - accuracy: 0.7882 - loss: 0.4321 - val_accuracy: 0.7710 - val_loss: 0.4903
Epoch 33/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 111ms/step - accuracy: 0.7900 - loss: 0.4503 - val_accuracy: 0.8020 - val_loss: 0.4528
Epoch 34/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 55ms/step - accuracy: 0.8026 - loss: 0.4253 - val_accuracy: 0.7550 - val_loss: 0.5129
Epoch 35/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 88ms/step - accuracy: 0.8108 - loss: 0.4189 - val_accuracy: 0.8060 - val_loss: 0.4630
Epoch 36/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 56ms/step - accuracy: 0.8078 - loss: 0.4076 - val_accuracy: 0.7940 - val_loss: 0.4573
Epoch 37/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 95ms/step - accuracy: 0.8198 - loss: 0.3999 - val_accuracy: 0.7650 - val_loss: 0.5137
Epoch 38/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 57ms/step - accuracy: 0.8096 - loss: 0.4074 - val_accuracy: 0.7740 - val_loss: 0.5006
Epoch 39/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 55ms/step - accuracy: 0.8197 - loss: 0.3848 - val_accuracy: 0.7750 - val_loss: 0.5096
Epoch 40/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 85ms/step - accuracy: 0.8185 - loss: 0.4106 - val_accuracy: 0.8000 - val_loss: 0.4725
Epoch 41/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 62ms/step - accuracy: 0.8359 - loss: 0.3728 - val_accuracy: 0.7970 - val_loss: 0.4713
Epoch 42/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 81ms/step - accuracy: 0.8419 - loss: 0.3774 - val_accuracy: 0.7940 - val_loss: 0.4776
Epoch 43/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 70ms/step - accuracy: 0.8369 - loss: 0.3775 - val_accuracy: 0.7850 - val_loss: 0.4998
Epoch 44/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.8343 - loss: 0.3679 - val_accuracy: 0.7820 - val_loss: 0.4750
Epoch 45/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 91ms/step - accuracy: 0.8469 - loss: 0.3486 - val_accuracy: 0.7680 - val_loss: 0.5072
Epoch 46/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 61ms/step - accuracy: 0.8434 - loss: 0.3724 - val_accuracy: 0.7920 - val_loss: 0.4925
Epoch 47/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.8479 - loss: 0.3405 - val_accuracy: 0.8010 - val_loss: 0.4456
Epoch 48/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 68ms/step - accuracy: 0.8636 - loss: 0.3138 - val_accuracy: 0.7210 - val_loss: 0.8223
Epoch 49/50
```

```
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 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.8024 - loss: 0.4282
Test accuracy: 0.807
```

Question 4: Repeat Steps 1-3, but now using a pretrained network. The sample sizes you use in Steps 2 and 3 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.

Instantiating the VGG16 convolutional base

```
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_notop.
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
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

using a pretrained model for feature extraction without data augmentation

obtaining the labels that correlate 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 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```

```
train_features.shape
```

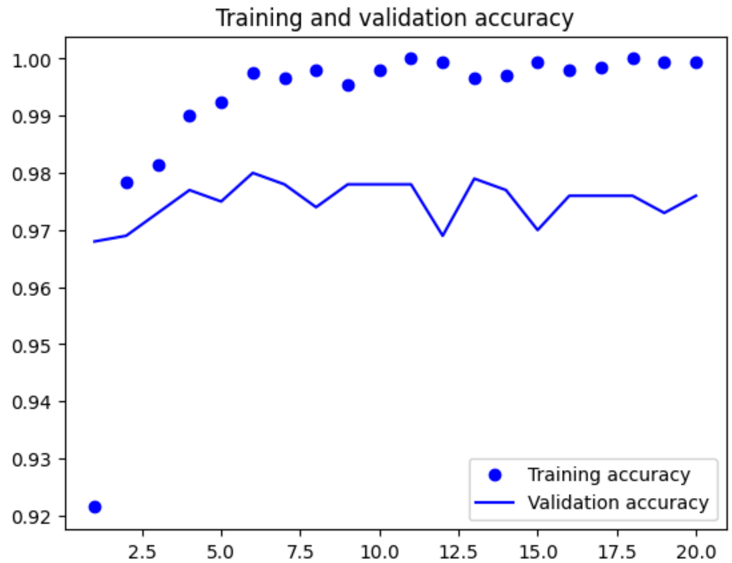
```
(2000, 5, 5, 512)
```

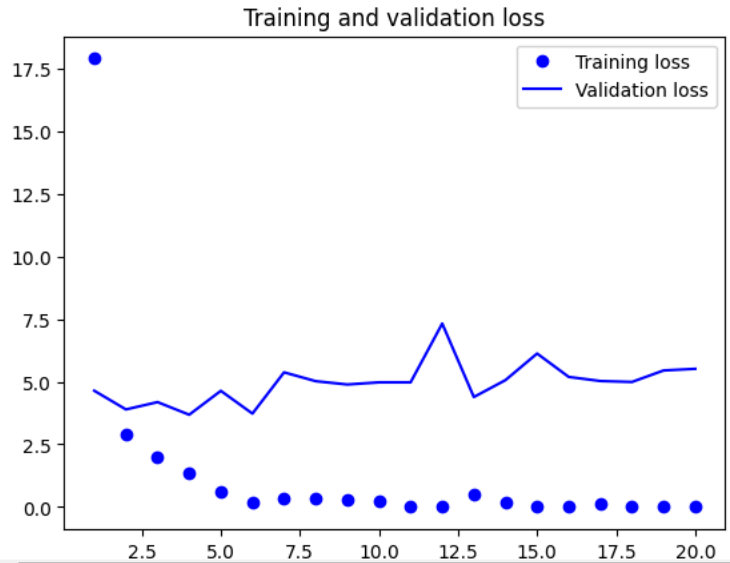
```
i6 = keras.Input(shape=(5, 5, 512))
d = layers.Flatten()(i6)
d = layers.Dense(256)(d)
d = layers.Dropout(0.5)(d)
out6 = layers.Dense(1, activation="sigmoid")(d)
m6 = keras.Model(i6, out6)
m6.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])

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 ━━━━━━━━━━━━━━━━━━━━ 4s 34ms/step - accuracy: 0.8587 - loss: 41.3700 - val_accuracy: 0.9680 - val_loss: 4.6385
Epoch 2/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9768 - loss: 3.8655 - val_accuracy: 0.9690 - val_loss: 3.8934
Epoch 3/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9801 - loss: 1.9199 - val_accuracy: 0.9730 - val_loss: 4.1849
Epoch 4/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9885 - loss: 1.3111 - val_accuracy: 0.9770 - val_loss: 3.6821
Epoch 5/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9930 - loss: 0.4904 - val_accuracy: 0.9750 - val_loss: 4.6349
Epoch 6/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9975 - loss: 0.4181 - val_accuracy: 0.9800 - val_loss: 3.7272
Epoch 7/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9973 - loss: 0.2019 - val_accuracy: 0.9780 - val_loss: 5.3725
```

```
Epoch 8/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9990 - loss: 0.1658 - val_accuracy: 0.9740 - val_loss: 5.0197
Epoch 9/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9976 - loss: 0.1701 - val_accuracy: 0.9780 - val_loss: 4.8875
Epoch 10/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9980 - loss: 0.2144 - val_accuracy: 0.9780 - val_loss: 4.9729
Epoch 11/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 2.8190e-15 - val_accuracy: 0.9780 - val_loss: 4.9729
Epoch 12/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.9690 - val_loss: 7.3161
Epoch 13/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9928 - loss: 1.2824 - val_accuracy: 0.9790 - val_loss: 4.3875
Epoch 14/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9969 - loss: 0.2044 - val_accuracy: 0.9770 - val_loss: 5.0625
Epoch 15/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9999 - loss: 5.5539e-04 - val_accuracy: 0.9700 - val_loss: 6.1221
Epoch 16/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9975 - loss: 0.0892 - val_accuracy: 0.9760 - val_loss: 5.1893
Epoch 17/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9983 - loss: 0.1535 - val_accuracy: 0.9760 - val_loss: 5.0258
Epoch 18/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 1.6421e-08 - val_accuracy: 0.9760 - val_loss: 4.9888
Epoch 19/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9989 - loss: 0.0249 - val_accuracy: 0.9730 - val_loss: 5.4478
Epoch 20/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9988 - loss: 0.1042 - val_accuracy: 0.9760 - val_loss: 5.5123
```

```
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 ━━━━━━━━━━━━━━━━━━━━ 18s 222ms/step - accuracy: 0.8303 - loss: 36.5900 - val_accuracy: 0.9100 - val_loss: 17.0759
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 16s 204ms/step - accuracy: 0.9268 - loss: 11.8919 - val_accuracy: 0.9820 - val_loss: 1.8894
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 163ms/step - accuracy: 0.9601 - loss: 5.1224 - val_accuracy: 0.9770 - val_loss: 2.6429
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 167ms/step - accuracy: 0.9594 - loss: 5.2229 - val_accuracy: 0.9640 - val_loss: 5.5138
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 167ms/step - accuracy: 0.9607 - loss: 4.8999 - val_accuracy: 0.9570 - val_loss: 6.3393
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 163ms/step - accuracy: 0.9732 - loss: 4.5346 - val_accuracy: 0.9690 - val_loss: 5.9684
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 166ms/step - accuracy: 0.9637 - loss: 3.8917 - val_accuracy: 0.9820 - val_loss: 2.9458
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 165ms/step - accuracy: 0.9808 - loss: 1.3837 - val_accuracy: 0.9780 - val_loss: 3.1161
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 166ms/step - accuracy: 0.9661 - loss: 4.5307 - val_accuracy: 0.9780 - val_loss: 3.6173
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 167ms/step - accuracy: 0.9725 - loss: 2.7100 - val_accuracy: 0.9830 - val_loss: 2.2419
```

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

```
-rw-r--r-- 1 root root 82M Oct 20 21:25 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))

x1 = augmentation2(input22)

# Specify output_shape for Lambda layer
x1 = keras.layers.Lambda(
    lambda x: vgg16.preprocess_input(x),
```

```
        output_shape=(180, 180, 3)
    )(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"])

    # Save the model
    callbacks = [
        keras.callbacks.ModelCheckpoint(
```