```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!unzip /content/drive/MyDrive/dogs-vs-cats.zip
```

```
Archive:  /content/drive/MyDrive/dogs-vs-cats.zip
  inflating: sampleSubmission.csv
  inflating: test1.zip
  inflating: train.zip
```

```
!unzip /content/train.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: train/dog.5499.jpg
  inflating: train/dog.55.jpg
  inflating: train/dog.550.jpg
  inflating: train/dog.5500.jpg
  inflating: train/dog.5501.jpg
  inflating: train/dog.5502.jpg
  inflating: train/dog.5503.jpg
  inflating: train/dog.5504.jpg
  inflating: train/dog.5505.jpg
  inflating: train/dog.5506.jpg
  inflating: train/dog.5507.jpg
  inflating: train/dog.5508.jpg
  inflating: train/dog.5509.jpg
  inflating: train/dog.551.jpg
  inflating: train/dog.5510.jpg
  inflating: train/dog.5511.jpg
  inflating: train/dog.5512.jpg
  inflating: train/dog.5513.jpg
  inflating: train/dog.5514.jpg
  inflating: train/dog.5515.jpg
  inflating: train/dog.5516.jpg
  inflating: train/dog.5517.jpg
  inflating: train/dog.5518.jpg
  inflating: train/dog.5519.jpg
  inflating: train/dog.552.jpg
  inflating: train/dog.5520.jpg
  inflating: train/dog.5521.jpg
  inflating: train/dog.5522.jpg
  inflating: train/dog.5523.jpg
  inflating: train/dog.5524.jpg
  inflating: train/dog.5525.jpg
  inflating: train/dog.5526.jpg
  inflating: train/dog.5527.jpg
  inflating: train/dog.5528.jpg
  inflating: train/dog.5529.jpg
  inflating: train/dog.553.jpg
  inflating: train/dog.5530.jpg
  inflating: train/dog.5531.jpg
  inflating: train/dog.5532.jpg
  inflating: train/dog.5533.jpg
  inflating: train/dog.5534.jpg
  inflating: train/dog.5535.jpg
  inflating: train/dog.5536.jpg
  inflating: train/dog.5537.jpg
  inflating: train/dog.5538.jpg
  inflating: train/dog.5539.jpg
  inflating: train/dog.554.jpg
  inflating: train/dog.5540.jpg
  inflating: train/dog.5541.jpg
  inflating: train/dog.5542.jpg
  inflating: train/dog.5543.jpg
  inflating: train/dog.5544.jpg
  inflating: train/dog.5545.jpg
  inflating: train/dog.5546.jpg
  inflating: train/dog.5547.jpg
  inflating: train/dog.5548.jpg
  inflating: train/dog.5549.jpg
```

```
!unzip /content/test1.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: test1/5499.jpg
  inflating: test1/55.jpg
  inflating: test1/550.jpg
  inflating: test1/5500.jpg
  inflating: test1/5501.jpg
  inflating: test1/5502.jpg
  inflating: test1/5503.jpg
  inflating: test1/5504.jpg
  inflating: test1/5505.jpg
  inflating: test1/5506.jpg
```

```
inflating: test1/5507.jpg
inflating: test1/5508.jpg
inflating: test1/5509.jpg
inflating: test1/551.jpg
inflating: test1/5510.jpg
inflating: test1/5511.jpg
inflating: test1/5512.jpg
inflating: test1/5513.jpg
inflating: test1/5514.jpg
inflating: test1/5515.jpg
inflating: test1/5516.jpg
inflating: test1/5517.jpg
inflating: test1/5518.jpg
inflating: test1/5519.jpg
inflating: test1/552.jpg
inflating: test1/5520.jpg
inflating: test1/5521.jpg
inflating: test1/5522.jpg
inflating: test1/5523.jpg
inflating: test1/5524.jpg
inflating: test1/5525.jpg
inflating: test1/5526.jpg
inflating: test1/5527.jpg
inflating: test1/5528.jpg
inflating: test1/5529.jpg
inflating: test1/553.jpg
inflating: test1/5530.jpg
inflating: test1/5531.jpg
inflating: test1/5532.jpg
inflating: test1/5533.jpg
inflating: test1/5534.jpg
inflating: test1/5535.jpg
inflating: test1/5536.jpg
inflating: test1/5537.jpg
inflating: test1/5538.jpg
inflating: test1/5539.jpg
inflating: test1/554.jpg
inflating: test1/5540.jpg
inflating: test1/5541.jpg
inflating: test1/5542.jpg
inflating: test1/5543.jpg
inflating: test1/5544.jpg
inflating: test1/5545.jpg
inflating: test1/5546.jpg
inflating: test1/5547.jpg
inflating: test1/5548.jpg
inflating: test1/5549.jpg
```

Q1. 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?

Retriving images to training, validation, and test directories

```
import os, shutil, pathlib

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

```
from tensorflow.keras.utils import image_dataset_from_directory

tra = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
valid = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
tes= image_dataset_from_directory(
    new_base_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 an instance of the dataset using a NumPy array that has 1000 random samples with a vector size of 16.

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

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

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

```
batched_dataset = dataset.batch(32)
for i, element in enumerate(batched_dataset):
    print(element.shape)
    if i >= 2:
        break
```

```
(32, 16)
(32, 16)
(32, 16)
```

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

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

Creating the model

building a small network to separate dogs from cats

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

Identifying a small convolution for dogs vs. cats categories .

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

input1000 = keras.Input(shape=(180, 180, 3))
din = layers.Rescaling(1./255)(input1000)
din = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(din)
din = layers.MaxPooling2D(pool_size=2)(din)
din = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(din)
din = layers.MaxPooling2D(pool_size=2)(din)
din = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(din)
din = layers.MaxPooling2D(pool_size=2)(din)
din = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(din)
din = layers.MaxPooling2D(pool_size=2)(din)
din = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(din)
din = layers.Flatten()(din)
din = layers.Dropout(0.5)(din)
output1000 = layers.Dense(1, activation="sigmoid")(din)
model10 = keras.Model(inputs=input1000, outputs=output1000)
```

Model training

```
model10.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

The training dataset is used to train the model after it has been built. We use the validation dataset to verify the model's performance at the end of each epoch. I'm utilizing GPU to reduce the time it takes for each epoch to execute.

```
model10.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0

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

 conv2d (Conv2D)             (None, 178, 178, 32)      896

 max_pooling2d (MaxPooling2  (None, 89, 89, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 43, 43, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 20, 20, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_3 (MaxPoolin  (None, 9, 9, 256)         0
 g2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 256)         590080

 flatten (Flatten)           (None, 12544)             0

 dropout (Dropout)           (None, 12544)             0

 dense (Dense)               (None, 1)                 12545

=================================================================
Total params: 991041 (3.78 MB)
Trainable params: 991041 (3.78 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model fitting

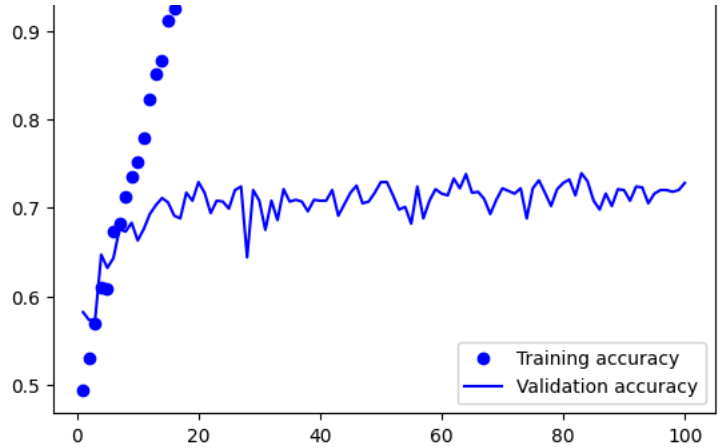
```
call1000 = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
his1000 = model10.fit(
```

```
    tra,
    epochs=100,
    validation_data=valid,
    callbacks=call1000)
```

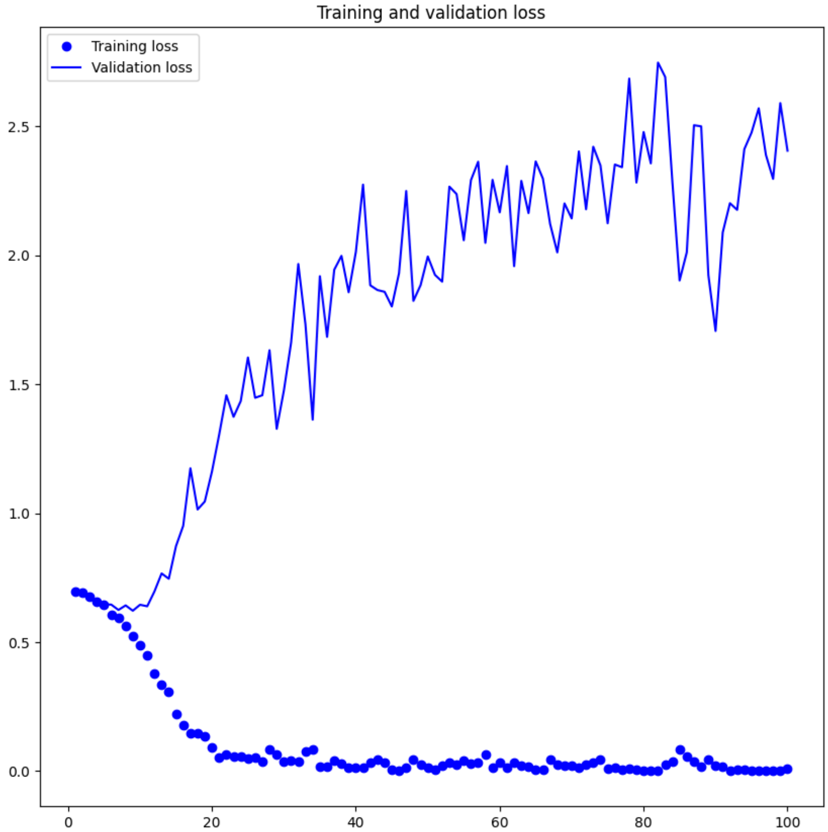
```
63/63 [==============================] - 1s 15ms/step - loss: 0.0234 - accuracy: 0.9940 - val_loss: 2.1780 - val_accuracy: 0.716
Epoch 73/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0322 - accuracy: 0.9885 - val_loss: 2.4205 - val_accuracy: 0.722
Epoch 74/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0431 - accuracy: 0.9875 - val_loss: 2.3477 - val_accuracy: 0.688
Epoch 75/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0096 - accuracy: 0.9965 - val_loss: 2.1238 - val_accuracy: 0.722
Epoch 76/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0117 - accuracy: 0.9960 - val_loss: 2.3516 - val_accuracy: 0.731
Epoch 77/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0059 - accuracy: 0.9980 - val_loss: 2.3410 - val_accuracy: 0.717
Epoch 78/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0077 - accuracy: 0.9965 - val_loss: 2.6847 - val_accuracy: 0.702
Epoch 79/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0058 - accuracy: 0.9985 - val_loss: 2.2815 - val_accuracy: 0.721
Epoch 80/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0030 - accuracy: 0.9990 - val_loss: 2.4776 - val_accuracy: 0.728
Epoch 81/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0036 - accuracy: 0.9985 - val_loss: 2.3559 - val_accuracy: 0.732
Epoch 82/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 2.7467 - val_accuracy: 0.714
Epoch 83/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0267 - accuracy: 0.9925 - val_loss: 2.6908 - val_accuracy: 0.739
Epoch 84/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0360 - accuracy: 0.9855 - val_loss: 2.2834 - val_accuracy: 0.730
Epoch 85/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0857 - accuracy: 0.9760 - val_loss: 1.9020 - val_accuracy: 0.708
Epoch 86/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0570 - accuracy: 0.9775 - val_loss: 2.0111 - val_accuracy: 0.698
Epoch 87/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0374 - accuracy: 0.9885 - val_loss: 2.5039 - val_accuracy: 0.716
Epoch 88/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0161 - accuracy: 0.9945 - val_loss: 2.4995 - val_accuracy: 0.702
Epoch 89/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0431 - accuracy: 0.9885 - val_loss: 1.9231 - val_accuracy: 0.721
Epoch 90/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0220 - accuracy: 0.9945 - val_loss: 1.7062 - val_accuracy: 0.720
Epoch 91/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0170 - accuracy: 0.9945 - val_loss: 2.0883 - val_accuracy: 0.708
Epoch 92/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0029 - accuracy: 0.9990 - val_loss: 2.2015 - val_accuracy: 0.724
Epoch 93/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0076 - accuracy: 0.9970 - val_loss: 2.1757 - val_accuracy: 0.723
Epoch 94/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0056 - accuracy: 0.9990 - val_loss: 2.4112 - val_accuracy: 0.705
Epoch 95/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 2.4752 - val_accuracy: 0.716
Epoch 96/100
63/63 [==============================] - 1s 15ms/step - loss: 7.3363e-04 - accuracy: 1.0000 - val_loss: 2.5696 - val_accuracy: 0
Epoch 97/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0012 - accuracy: 0.9995 - val_loss: 2.3886 - val_accuracy: 0.720
Epoch 98/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0013 - accuracy: 0.9995 - val_loss: 2.2957 - val_accuracy: 0.718
Epoch 99/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0015 - accuracy: 0.9995 - val_loss: 2.5896 - val_accuracy: 0.720
Epoch 100/100
63/63 [==============================] - 1s 15ms/step - loss: 0.0096 - accuracy: 0.9970 - val_loss: 2.4050 - val_accuracy: 0.728
```

curves of loss and accuracy during training were constructed

```
import matplotlib.pyplot as plt
accuracy = his1000.history["accuracy"]
val_accuracy = his1000.history["val_accuracy"]
loss = his1000.history["loss"]
val_loss = his1000.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.figure(figsize=(10, 10))
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```




```
<Figure size 640x480 with 0 Axes>
```



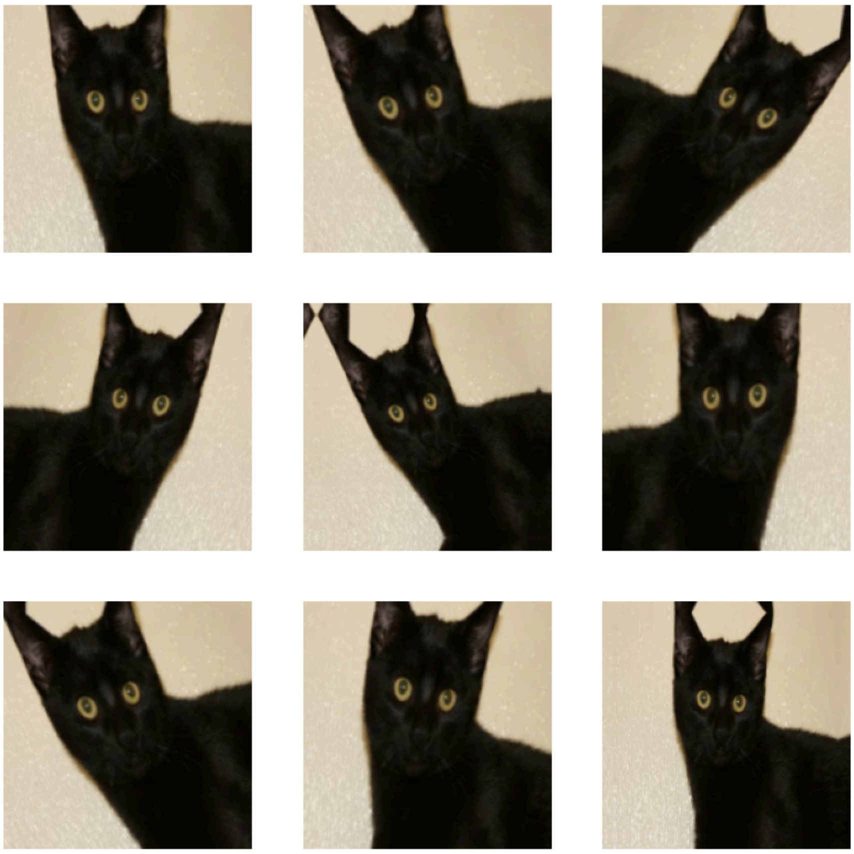

```
test1000 = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = test1000.evaluate(tes)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 1s 7ms/step - loss: 0.5956 - accuracy: 0.7030
Test accuracy: 0.703
```

Q2. 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

```
import os, shutil, pathlib

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

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_Q2")

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 1500 samples, test has 500 samples and validation has 500 samples.
make_subset("train", start_index=667, end_index=2167)
make_subset("validation", start_index=2168, end_index=2668)
make_subset("test", start_index=2669, end_index=3168)


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


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



convolutional neural network with dropout and picture augmentation

```
input15 = keras.Input(shape=(180, 180, 3))
din2 = augmentation_info(input15)
din2 = layers.Rescaling(1./255)(din2)
din2 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(din2)
din2 = layers.MaxPooling2D(pool_size=2)(din2)
din2 = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(din2)
din2 = layers.MaxPooling2D(pool_size=2)(din2)
din2 = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(din2)
din2 = layers.MaxPooling2D(pool_size=2)(din2)
din2 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(din2)
din2 = layers.MaxPooling2D(pool_size=2)(din2)
din2 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(din2)
din2 = layers.Flatten()(din2)
din2 = layers.Dropout(0.5)(din2)
output15 = layers.Dense(1, activation="sigmoid")(din2)
model15 = keras.Model(inputs=input15, outputs=output15)

model15.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])


callback15 = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation_info.keras",
        save_best_only=True,
        monitor="val_loss")
]
hist15 = model15.fit(
    tra,
    epochs=200,
    validation_data=valid,
    callbacks=callback15)
```

```
63/63 [==============================] - 1s 16ms/step - loss: 0.0987 - accuracy: 0.9670 - val_loss: 0.7637 - val_accuracy: 0.827
Epoch 198/200
63/63 [==============================] - 1s 16ms/step - loss: 0.0674 - accuracy: 0.9745 - val_loss: 0.7195 - val_accuracy: 0.838
Epoch 199/200
63/63 [==============================] - 1s 16ms/step - loss: 0.0790 - accuracy: 0.9675 - val_loss: 0.7772 - val_accuracy: 0.825
Epoch 200/200
63/63 [==============================] - 1s 17ms/step - loss: 0.0940 - accuracy: 0.9665 - val_loss: 0.7728 - val_accuracy: 0.822
```

Test Accuracy of model

```
test15ac = keras.models.load_model(
    "convnet_from_scratch_with_augmentation_info.keras")
test_loss, test_acc = test15ac.evaluate(tes)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 1s 7ms/step - loss: 0.4779 - accuracy: 0.8220
Test accuracy: 0.822
```

Q3. 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.

Increasing the training sample to 2000, keeping the Validation and test sets the same as before(500 samples)

```
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)
```

```
input20 = keras.Input(shape=(180, 180, 3))
din3 = augmentation_info(input20)
din3 = layers.Rescaling(1./255)(din3)
din3 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(din3)
din3 = layers.MaxPooling2D(pool_size=2)(din3)
din3 = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(din3)
din3 = layers.MaxPooling2D(pool_size=2)(din3)
din3 = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(din3)
din3 = layers.MaxPooling2D(pool_size=2)(din3)
din3 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(din3)
din3 = layers.MaxPooling2D(pool_size=2)(din3)
din3 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(din3)
din3 = layers.Flatten()(din3)
din3 = layers.Dropout(0.5)(din3)
output20 = layers.Dense(1, activation="sigmoid")(din3)
mode20 = keras.Model(inputs=input20, outputs=output20)

mode20.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])


callback20 = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation_info.keras",
        save_best_only=True,
        monitor="val_loss")
]
hist20 = mode20.fit(
    tra,
    epochs=300,
    validation_data=valid,
    callbacks=callback20)
```

```
63/63 [==============================] - 1s 17ms/step - loss: 0.0708 - accuracy: 0.9725 - val_loss: 0.9134 - val_accuracy: 0.841
Epoch 276/300
63/63 [==============================] - 1s 17ms/step - loss: 0.0676 - accuracy: 0.9745 - val_loss: 0.8878 - val_accuracy: 0.818
Epoch 277/300
63/63 [==============================] - 1s 16ms/step - loss: 0.1004 - accuracy: 0.9635 - val_loss: 0.8262 - val_accuracy: 0.822
Epoch 278/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0525 - accuracy: 0.9810 - val_loss: 0.7807 - val_accuracy: 0.840
Epoch 279/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0689 - accuracy: 0.9750 - val_loss: 0.7374 - val_accuracy: 0.840
Epoch 280/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0725 - accuracy: 0.9745 - val_loss: 0.7454 - val_accuracy: 0.838
Epoch 281/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0765 - accuracy: 0.9740 - val_loss: 0.7784 - val_accuracy: 0.847
Epoch 282/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0604 - accuracy: 0.9760 - val_loss: 0.7916 - val_accuracy: 0.841
Epoch 283/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0625 - accuracy: 0.9785 - val_loss: 0.6756 - val_accuracy: 0.846
Epoch 284/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0735 - accuracy: 0.9760 - val_loss: 0.6660 - val_accuracy: 0.843
Epoch 285/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0602 - accuracy: 0.9795 - val_loss: 0.7859 - val_accuracy: 0.838
Epoch 286/300
63/63 [==============================] - 1s 17ms/step - loss: 0.0515 - accuracy: 0.9800 - val_loss: 0.6867 - val_accuracy: 0.856
Epoch 287/300
63/63 [==============================] - 1s 17ms/step - loss: 0.0756 - accuracy: 0.9760 - val_loss: 0.8012 - val_accuracy: 0.849
Epoch 288/300
63/63 [==============================] - 1s 17ms/step - loss: 0.0728 - accuracy: 0.9690 - val_loss: 0.8095 - val_accuracy: 0.842
Epoch 289/300
63/63 [==============================] - 1s 17ms/step - loss: 0.0742 - accuracy: 0.9685 - val_loss: 0.8434 - val_accuracy: 0.811
Epoch 290/300
63/63 [==============================] - 1s 17ms/step - loss: 0.0591 - accuracy: 0.9795 - val_loss: 0.8041 - val_accuracy: 0.820
Epoch 291/300
63/63 [==============================] - 1s 17ms/step - loss: 0.0927 - accuracy: 0.9665 - val_loss: 0.8297 - val_accuracy: 0.819
Epoch 292/300
63/63 [==============================] - 1s 17ms/step - loss: 0.0759 - accuracy: 0.9715 - val_loss: 0.6929 - val_accuracy: 0.849
Epoch 293/300
63/63 [==============================] - 1s 17ms/step - loss: 0.0647 - accuracy: 0.9770 - val_loss: 0.7925 - val_accuracy: 0.833
Epoch 294/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0434 - accuracy: 0.9825 - val_loss: 0.7727 - val_accuracy: 0.840
Epoch 295/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0744 - accuracy: 0.9735 - val_loss: 0.7936 - val_accuracy: 0.832
Epoch 296/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0696 - accuracy: 0.9745 - val_loss: 0.9461 - val_accuracy: 0.832
Epoch 297/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0587 - accuracy: 0.9780 - val_loss: 0.8504 - val_accuracy: 0.827
Epoch 298/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0652 - accuracy: 0.9725 - val_loss: 0.8918 - val_accuracy: 0.834
Epoch 299/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0568 - accuracy: 0.9745 - val_loss: 0.8409 - val_accuracy: 0.835
Epoch 300/300
63/63 [==============================] - 1s 16ms/step - loss: 0.0671 - accuracy: 0.9765 - val_loss: 0.7958 - val_accuracy: 0.834
```

```
tesacc20 = keras.models.load_model(
    "convnet_from_scratch_with_augmentation_info.keras")
test_loss, test_acc = tesacc20.evaluate(tes)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 1s 7ms/step - loss: 0.4901 - accuracy: 0.8040
Test accuracy: 0.804
```

Q4. 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

```
convoluted_b = 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 [==============================] - 1s 0us/step
```

```
convoluted_b.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (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: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

pretrained model for feature extraction without data augmentation

```
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 = convoluted_b.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(tra)
val_features, val_labels =  get_features_and_labels(valid)
test_features, test_labels =  get_features_and_labels(tes)
```

```
1/1 [==============================] - 1s 547ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/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 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
```

```
train_features.shape
```

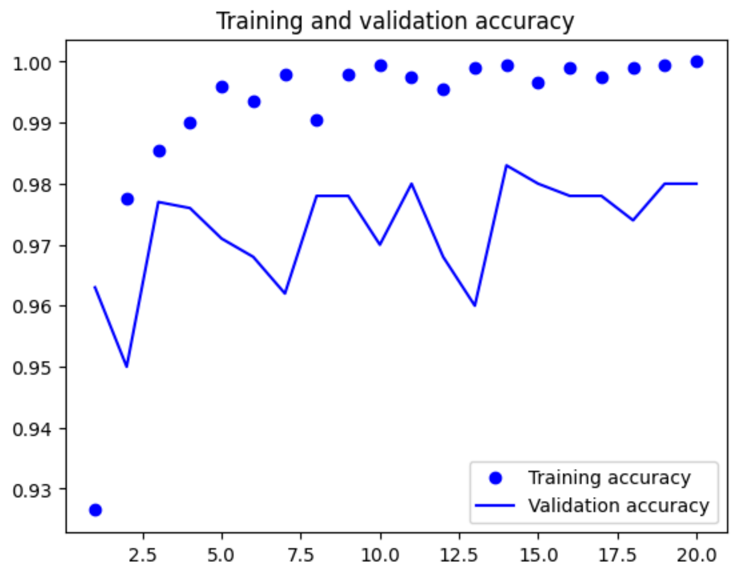
```
(2000, 5, 5, 512)
```

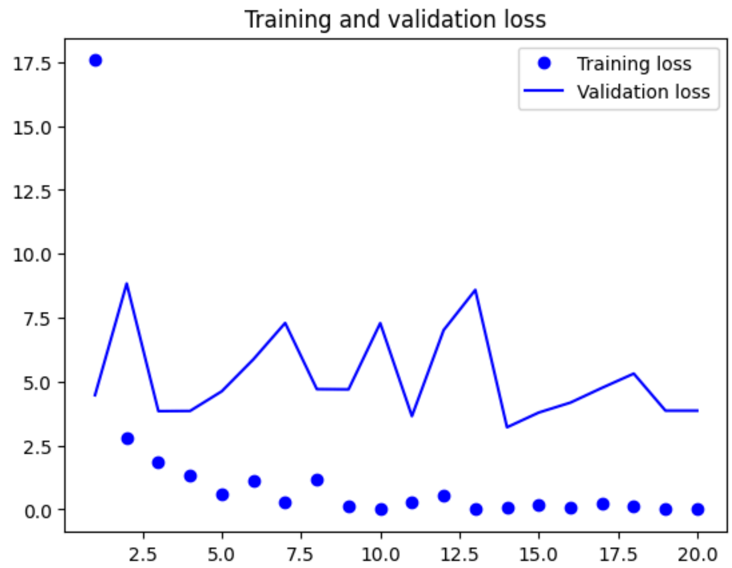
```
inp6 = keras.Input(shape=(5, 5, 512))
din4 = layers.Flatten()(inp6)
din4 = layers.Dense(256)(din4)
din4 = layers.Dropout(0.5)(din4)
out4 = layers.Dense(1, activation="sigmoid")(din4)
model4 = keras.Model(inp6, out4)
model4.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])

callbac4 = [
    keras.callbacks.ModelCheckpoint(
      filepath="feature_extraction.keras",
      save_best_only=True,
      monitor="val_loss")
]
history466 = model4.fit(
    train_features, train_labels,
    epochs=20,
    validation_data=(val_features, val_labels),
    callbacks=callbac4)
```

```
Epoch 1/20
63/63 [==============================] - 1s 9ms/step - loss: 17.5665 - accuracy: 0.9265 - val_loss: 4.4696 - val_accuracy: 0.9630
Epoch 2/20
63/63 [==============================] - 0s 4ms/step - loss: 2.7860 - accuracy: 0.9775 - val_loss: 8.8357 - val_accuracy: 0.9500
Epoch 3/20
63/63 [==============================] - 0s 6ms/step - loss: 1.8621 - accuracy: 0.9855 - val_loss: 3.8460 - val_accuracy: 0.9770
Epoch 4/20
63/63 [==============================] - 0s 5ms/step - loss: 1.3121 - accuracy: 0.9900 - val_loss: 3.8523 - val_accuracy: 0.9760
Epoch 5/20
63/63 [==============================] - 0s 4ms/step - loss: 0.6007 - accuracy: 0.9960 - val_loss: 4.6217 - val_accuracy: 0.9710
Epoch 6/20
63/63 [==============================] - 0s 4ms/step - loss: 1.1346 - accuracy: 0.9935 - val_loss: 5.8753 - val_accuracy: 0.9680
Epoch 7/20
63/63 [==============================] - 0s 4ms/step - loss: 0.2632 - accuracy: 0.9980 - val_loss: 7.2924 - val_accuracy: 0.9620
Epoch 8/20
63/63 [==============================] - 0s 5ms/step - loss: 1.1837 - accuracy: 0.9905 - val_loss: 4.7049 - val_accuracy: 0.9780
Epoch 9/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1455 - accuracy: 0.9980 - val_loss: 4.6968 - val_accuracy: 0.9780
Epoch 10/20
63/63 [==============================] - 0s 4ms/step - loss: 0.0059 - accuracy: 0.9995 - val_loss: 7.2867 - val_accuracy: 0.9700
Epoch 11/20
63/63 [==============================] - 0s 6ms/step - loss: 0.2628 - accuracy: 0.9975 - val_loss: 3.6507 - val_accuracy: 0.9800
Epoch 12/20
63/63 [==============================] - 0s 4ms/step - loss: 0.5302 - accuracy: 0.9955 - val_loss: 7.0121 - val_accuracy: 0.9680
Epoch 13/20
63/63 [==============================] - 0s 4ms/step - loss: 0.0268 - accuracy: 0.9990 - val_loss: 8.5860 - val_accuracy: 0.9600
Epoch 14/20
63/63 [==============================] - 0s 6ms/step - loss: 0.0721 - accuracy: 0.9995 - val_loss: 3.2137 - val_accuracy: 0.9830
```

```
Epoch 15/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1649 - accuracy: 0.9965 - val_loss: 3.7892 - val_accuracy: 0.9800
Epoch 16/20
63/63 [==============================] - 0s 4ms/step - loss: 0.0548 - accuracy: 0.9990 - val_loss: 4.1783 - val_accuracy: 0.9780
Epoch 17/20
63/63 [==============================] - 0s 4ms/step - loss: 0.2411 - accuracy: 0.9975 - val_loss: 4.7576 - val_accuracy: 0.9780
Epoch 18/20
63/63 [==============================] - 0s 4ms/step - loss: 0.1478 - accuracy: 0.9990 - val_loss: 5.3126 - val_accuracy: 0.9740
Epoch 19/20
63/63 [==============================] - 0s 4ms/step - loss: 0.0290 - accuracy: 0.9995 - val_loss: 3.8645 - val_accuracy: 0.9800
Epoch 20/20
63/63 [==============================] - 0s 4ms/step - loss: 1.6025e-17 - accuracy: 1.0000 - val_loss: 3.8645 - val_accuracy: 0.980
```

```
import matplotlib.pyplot as plt
accur4 = history466.history["accuracy"]
valac4 = history466.history["val_accuracy"]
loss4 = history466.history["loss"]
valloss4 = history466.history["val_loss"]
epochs = range(1, len(accur4) + 1)
plt.plot(epochs, accur4, "bo", label="Training accuracy")
plt.plot(epochs, valac4, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss4, "bo", label="Training loss")
plt.plot(epochs, valloss4, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```





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

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

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

```
augmented2 = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

input22 = keras.Input(shape=(180, 180, 3))
dinx = augmented2(input22)
dinx =keras.layers.Lambda(
     lambda x: keras.applications.vgg16.preprocess_input(x))(dinx)
dinx = convoluted_b(dinx)
dinx = layers.Flatten()(dinx)
dinx = layers.Dense(256)(dinx)
dinx = layers.Dropout(0.5)(dinx)
outputs = layers.Dense(1, activation="sigmoid")(dinx)
modelx = keras.Model(input22, outputs)
modelx.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])


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

historyfi = modelx.fit(
    tra,
    epochs=10,
    validation_data=valid,
    callbacks=callbafi
)
```

```
Epoch 1/10
63/63 [==============================] - 4s 37ms/step - loss: 18.1709 - accuracy: 0.8985 - val_loss: 3.6626 - val_accuracy: 0.9720
Epoch 2/10
63/63 [==============================] - 2s 33ms/step - loss: 7.1967 - accuracy: 0.9490 - val_loss: 1.5404 - val_accuracy: 0.9810
Epoch 3/10
63/63 [==============================] - 2s 28ms/step - loss: 6.3302 - accuracy: 0.9515 - val_loss: 7.0458 - val_accuracy: 0.9630
Epoch 4/10
63/63 [==============================] - 2s 27ms/step - loss: 5.5661 - accuracy: 0.9580 - val_loss: 6.5527 - val_accuracy: 0.9660
Epoch 5/10
63/63 [==============================] - 2s 27ms/step - loss: 3.8799 - accuracy: 0.9645 - val_loss: 5.7896 - val_accuracy: 0.9690
Epoch 6/10
63/63 [==============================] - 2s 27ms/step - loss: 3.1075 - accuracy: 0.9695 - val_loss: 3.6678 - val_accuracy: 0.9700
Epoch 7/10
63/63 [==============================] - 2s 27ms/step - loss: 3.1434 - accuracy: 0.9695 - val_loss: 4.6109 - val_accuracy: 0.9730
Epoch 8/10
63/63 [==============================] - 2s 27ms/step - loss: 3.2794 - accuracy: 0.9700 - val_loss: 4.1807 - val_accuracy: 0.9750
Epoch 9/10
63/63 [==============================] - 2s 27ms/step - loss: 2.9110 - accuracy: 0.9740 - val_loss: 2.7439 - val_accuracy: 0.9780
Epoch 10/10
63/63 [==============================] - 2s 27ms/step - loss: 2.7900 - accuracy: 0.9755 - val_loss: 3.0460 - val_accuracy: 0.9780
```

```
tesaccfi = keras.models.load_model(
    "features_extraction_with_augmentation2.keras",safe_mode=False)
test_loss, test_acc = tesaccfi.evaluate(tes)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 1s 16ms/step - loss: 5.8296 - accuracy: 0.9670
Test accuracy: 0.967
```

Fine-tuning a pretrained model

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

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

callbacktuning = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning.keras",
        save_best_only=True,
        monitor="val_loss")
]
historytuning = modelx.fit(
    tra,
    epochs=30,
    validation_data=valid,
    callbacks=callbacktuning)
```

```
63/63 [==============================] - 2s 34ms/step - loss: 1.0991 - accuracy: 0.9850 - val_loss: 2.4037 - val_accuracy: 0.977
Epoch 3/30
63/63 [==============================] - 2s 34ms/step - loss: 0.9929 - accuracy: 0.9860 - val_loss: 2.2963 - val_accuracy: 0.980
Epoch 4/30
63/63 [==============================] - 2s 26ms/step - loss: 1.0497 - accuracy: 0.9805 - val_loss: 2.4143 - val_accuracy: 0.984
Epoch 5/30
63/63 [==============================] - 2s 34ms/step - loss: 0.6693 - accuracy: 0.9895 - val_loss: 2.1237 - val_accuracy: 0.982
Epoch 6/30
63/63 [==============================] - 2s 34ms/step - loss: 0.5143 - accuracy: 0.9875 - val_loss: 1.9112 - val_accuracy: 0.978
Epoch 7/30
63/63 [==============================] - 2s 26ms/step - loss: 1.0529 - accuracy: 0.9805 - val_loss: 2.3851 - val_accuracy: 0.983
Epoch 8/30
63/63 [==============================] - 2s 26ms/step - loss: 0.6439 - accuracy: 0.9880 - val_loss: 2.4341 - val_accuracy: 0.981
Epoch 9/30
63/63 [==============================] - 2s 26ms/step - loss: 0.1669 - accuracy: 0.9935 - val_loss: 1.9122 - val_accuracy: 0.987
Epoch 10/30
63/63 [==============================] - 2s 27ms/step - loss: 0.6210 - accuracy: 0.9875 - val_loss: 2.1559 - val_accuracy: 0.983
Epoch 11/30
63/63 [==============================] - 2s 34ms/step - loss: 0.2664 - accuracy: 0.9915 - val_loss: 1.7316 - val_accuracy: 0.985
Epoch 12/30
63/63 [==============================] - 2s 26ms/step - loss: 0.3057 - accuracy: 0.9935 - val_loss: 1.9006 - val_accuracy: 0.983
Epoch 13/30
63/63 [==============================] - 2s 35ms/step - loss: 0.5644 - accuracy: 0.9875 - val_loss: 1.5763 - val_accuracy: 0.980
Epoch 14/30
63/63 [==============================] - 2s 26ms/step - loss: 0.2605 - accuracy: 0.9930 - val_loss: 1.8983 - val_accuracy: 0.981
Epoch 15/30
63/63 [==============================] - 2s 26ms/step - loss: 0.3659 - accuracy: 0.9910 - val_loss: 2.0502 - val_accuracy: 0.981
Epoch 16/30
63/63 [==============================] - 2s 26ms/step - loss: 0.2719 - accuracy: 0.9945 - val_loss: 1.8054 - val_accuracy: 0.986
Epoch 17/30
63/63 [==============================] - 2s 34ms/step - loss: 0.2557 - accuracy: 0.9930 - val_loss: 1.3983 - val_accuracy: 0.985
Epoch 18/30
63/63 [==============================] - 2s 26ms/step - loss: 0.2319 - accuracy: 0.9930 - val_loss: 1.7106 - val_accuracy: 0.985
Epoch 19/30
63/63 [==============================] - 2s 26ms/step - loss: 0.4268 - accuracy: 0.9925 - val_loss: 2.0042 - val_accuracy: 0.981
Epoch 20/30
63/63 [==============================] - 2s 26ms/step - loss: 0.2275 - accuracy: 0.9940 - val_loss: 1.8144 - val_accuracy: 0.979
Epoch 21/30
63/63 [==============================] - 2s 26ms/step - loss: 0.1971 - accuracy: 0.9935 - val_loss: 1.7369 - val_accuracy: 0.983
Epoch 22/30
63/63 [==============================] - 2s 26ms/step - loss: 0.1183 - accuracy: 0.9925 - val_loss: 1.7152 - val_accuracy: 0.984
Epoch 23/30
63/63 [==============================] - 2s 26ms/step - loss: 0.0874 - accuracy: 0.9950 - val_loss: 1.4986 - val_accuracy: 0.984
Epoch 24/30
63/63 [==============================] - 2s 26ms/step - loss: 0.1640 - accuracy: 0.9930 - val_loss: 1.5939 - val_accuracy: 0.984
Epoch 25/30
63/63 [==============================] - 2s 26ms/step - loss: 0.2596 - accuracy: 0.9910 - val_loss: 1.5811 - val_accuracy: 0.985
Epoch 26/30
63/63 [==============================] - 2s 26ms/step - loss: 0.1482 - accuracy: 0.9955 - val_loss: 2.0669 - val_accuracy: 0.979
Epoch 27/30
63/63 [==============================] - 2s 26ms/step - loss: 0.1961 - accuracy: 0.9950 - val_loss: 1.5944 - val_accuracy: 0.983
Epoch 28/30
63/63 [==============================] - 2s 27ms/step - loss: 0.2169 - accuracy: 0.9930 - val_loss: 1.4556 - val_accuracy: 0.984
Epoch 29/30
63/63 [==============================] - 2s 34ms/step - loss: 0.1281 - accuracy: 0.9975 - val_loss: 1.3949 - val_accuracy: 0.984
Epoch 30/30
63/63 [==============================] - 2s 26ms/step - loss: 0.1125 - accuracy: 0.9960 - val_loss: 1.9196 - val_accuracy: 0.973
```