Convolution Assignment

Uploading Kaggle API File and Downloading Dogs vs Cats dataset from Kaggle

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

Choose Files kaggle.json
- **kaggle.json**(application/json) - 64 bytes, last modified: 10/20/2024 - 100% done

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

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

```
!kaggle competitions download -c dogs-vs-cats
!unzip -qq dogs-vs-cats.zip
!unzip -qq train.zip
```

```
Downloading dogs-vs-cats.zip to /content
 98% 793M/812M [00:06<00:00, 164MB/s]
100% 812M/812M [00:06<00:00, 133MB/s]
```

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?

Creating and Copying dataset to test, train and validation directory

```
import os, shutil, pathlib
d_dir = pathlib.Path("train")
n_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = n_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:
            src = d_dir / fname
            dst = dir / fname
            shutil.copyfile(src, dst)

make_subset("train", start_index=500, end_index=1500)
make_subset("validation", start_index=1500, end_index=2000)
make_subset("test", start_index=2000, end_index=2500)
```

Building a basic model to classify dogs and cats using convolutional neural networks

```
from tensorflow.keras.utils import image_dataset_from_directory

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

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

test_data= image_dataset_from_directory(n_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

run_num = np.random.normal(size=(1000, 16))
dataset = tf.data.Dataset.from_tensor_slices(run_num)
for i, element in enumerate(dataset):
    print(element.shape)
    if i >= 2:
       break

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

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
```

```
(16,)
(16,)
(16,)
(32, 16)
(32, 16)
(32, 16)
(4, 4)
(4, 4)
(4, 4)
```

Displaying the shapes of the data and labels yielded by the Dataset

```
for dataset_batch, label_batch in train_data:
    print("data batch shape:", dataset_batch.shape)
    print("labels batch shape:", label_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

input_1000 = keras.Input(shape=(180, 180, 3))
dat = layers.Rescaling(1./255)(input_1000)
dat = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(dat)
dat = layers.MaxPooling2D(pool_size=2)(dat)
dat = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(dat)
dat = layers.MaxPooling2D(pool_size=2)(dat)
dat = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(dat)
dat = layers.MaxPooling2D(pool_size=2)(dat)
dat = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(dat)
dat = layers.MaxPooling2D(pool_size=2)(dat)
dat = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(dat)
dat = layers.Flatten()(dat)
dat = layers.Dropout(0.5)(dat)
output_1000 = layers.Dense(1, activation="sigmoid")(dat)
model = keras.Model(inputs=input_1000, outputs=output_1000)
```

Model Training

```
model.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 T4 GPU to reduce the time it takes for each epoch to execute

```
model.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)

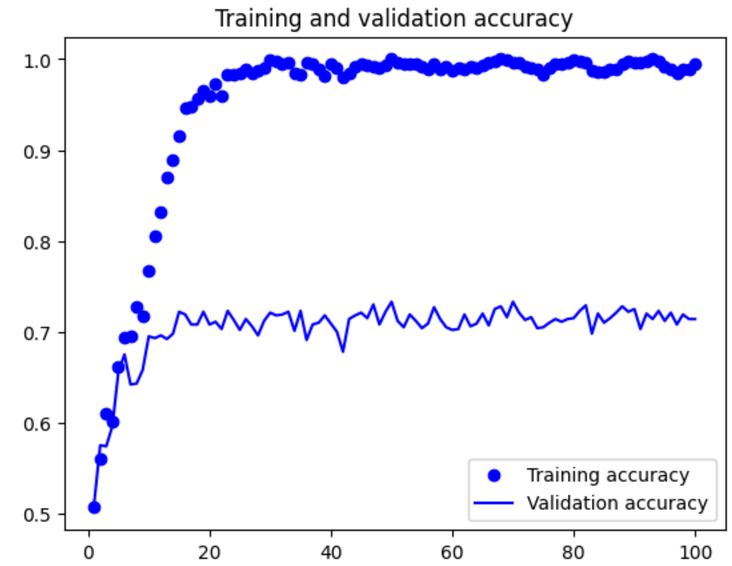
Model Fitting

```
callbacks = [
keras.callbacks.ModelCheckpoint(
filepath="convnet_from_scratch.keras",
save_best_only=True,
monitor="val_loss")
]
history = model.fit(train_data,
epochs=100,
validation_data=valid_data,
callbacks=callbacks)
```

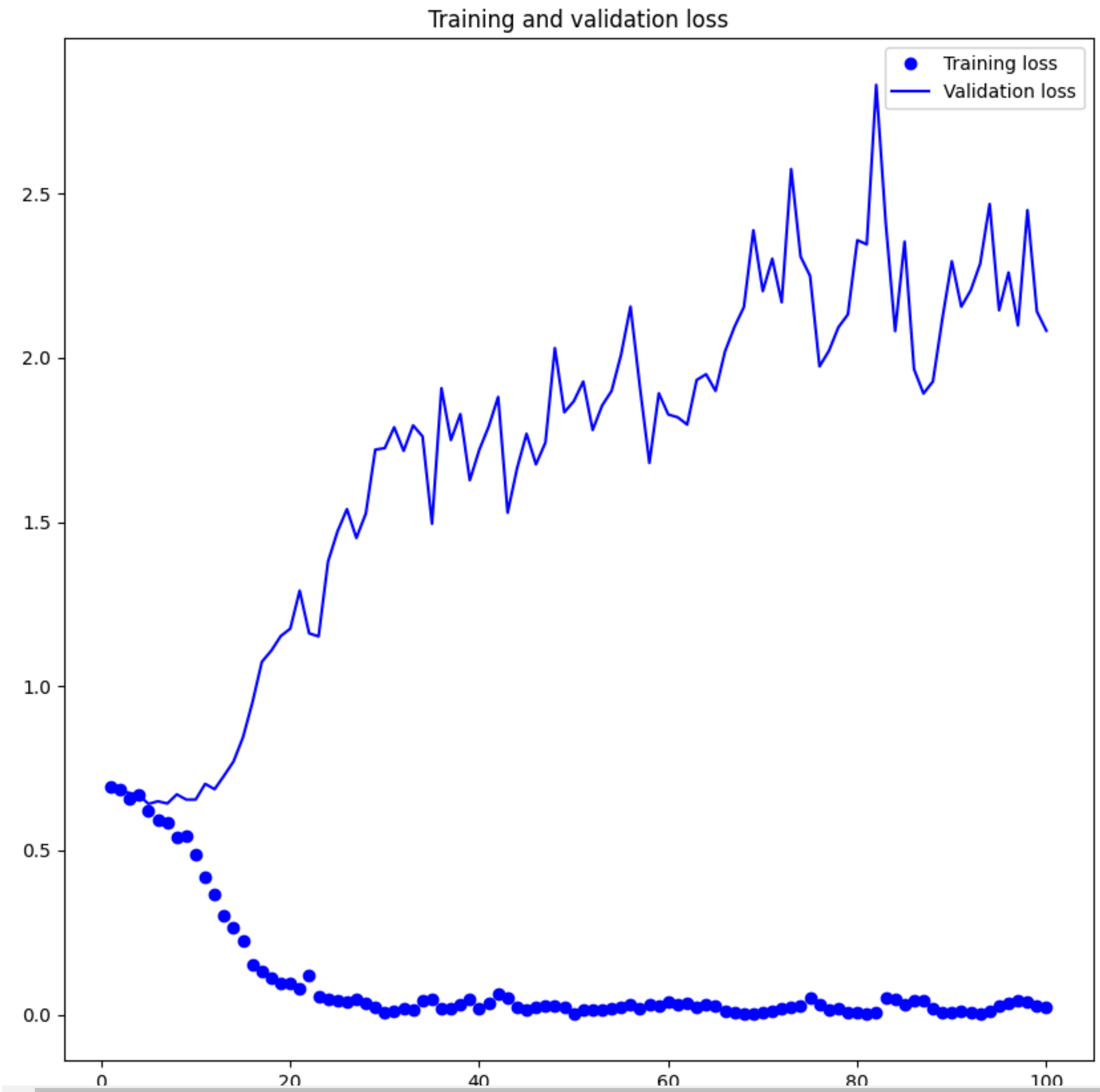
```
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 87ms/step - accuracy: 0.9969 - loss: 0.0065 - val_accuracy: 0.7220 - val_loss: 2.1197
Epoch 90/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 54ms/step - accuracy: 0.9958 - loss: 0.0124 - val_accuracy: 0.7250 - val_loss: 2.2940
Epoch 91/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 81ms/step - accuracy: 0.9936 - loss: 0.0134 - val_accuracy: 0.7030 - val_loss: 2.1556
Epoch 92/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - accuracy: 0.9980 - loss: 0.0044 - val_accuracy: 0.7200 - val_loss: 2.2056
Epoch 93/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 59ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.7140 - val_loss: 2.2867
Epoch 94/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 74ms/step - accuracy: 0.9980 - loss: 0.0094 - val_accuracy: 0.7230 - val_loss: 2.4682
Epoch 95/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 67ms/step - accuracy: 0.9921 - loss: 0.0239 - val_accuracy: 0.7120 - val_loss: 2.1449
Epoch 96/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.9924 - loss: 0.0246 - val_accuracy: 0.7210 - val_loss: 2.2599
Epoch 97/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.9815 - loss: 0.0502 - val_accuracy: 0.7080 - val_loss: 2.0987
Epoch 98/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 105ms/step - accuracy: 0.9933 - loss: 0.0251 - val_accuracy: 0.7190 - val_loss: 2.4498
Epoch 99/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 59ms/step - accuracy: 0.9874 - loss: 0.0323 - val_accuracy: 0.7140 - val_loss: 2.1410
Epoch 100/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9942 - loss: 0.0259 - val_accuracy: 0.7140 - val_loss: 2.0825
```

Curves of loss and accuracy during training

```
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.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>
```



Test Accuracy of model

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

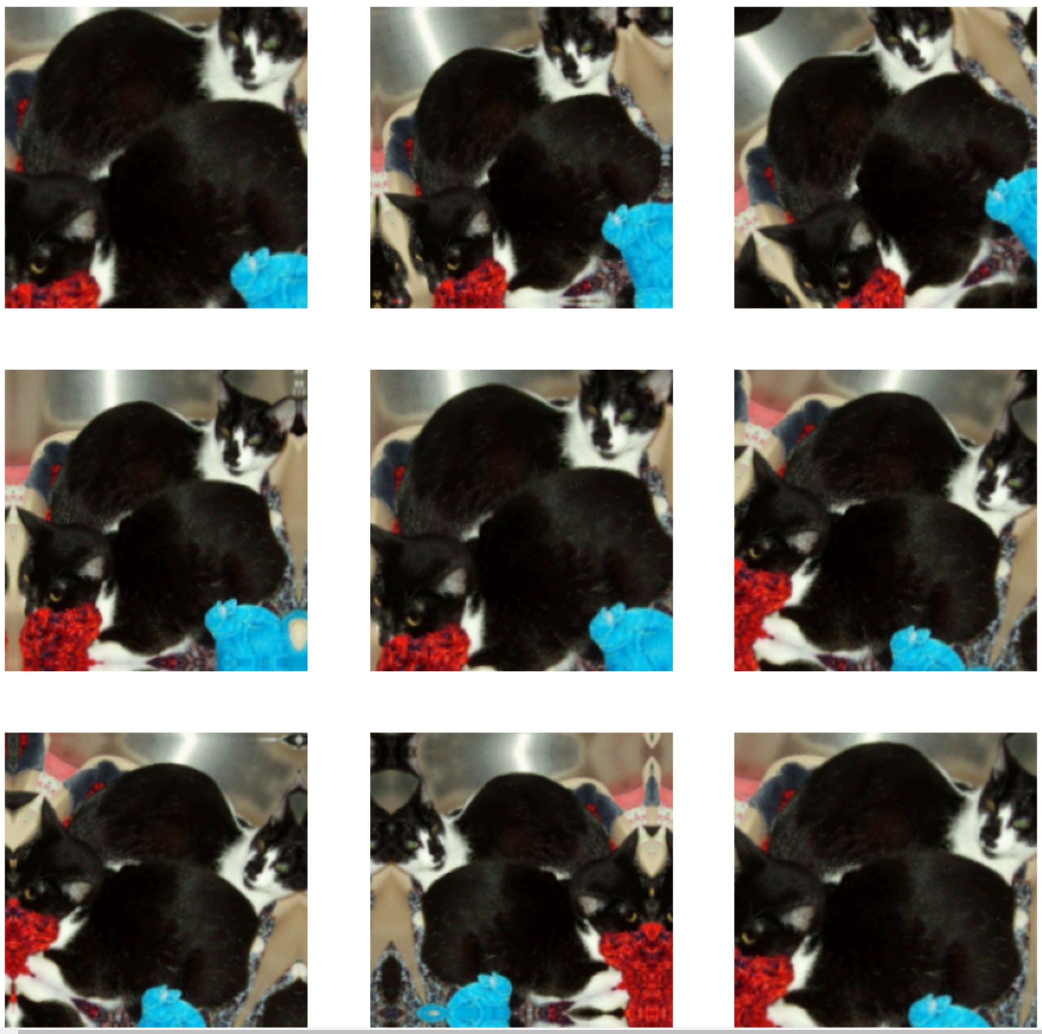
```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.6679 - loss: 0.6161
Test accuracy: 0.656
```

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

```
shutil.rmtree("./cats_vs_dogs_small_Q2", ignore_errors=True)
org_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=org_dir / fname,
            dst=dir / fname)

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 train_data.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

```
input = keras.Input(shape=(180, 180, 3))
data = augmentation_info(input)
data = layers.Rescaling(1./255)(data)
data= layers.Conv2D(filters=32, kernel_size=3, activation="relu")(data)
data = layers.MaxPooling2D(pool_size=2)(data)
data = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(data)
data = layers.MaxPooling2D(pool_size=2)(data)
data = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(data)
data = layers.MaxPooling2D(pool_size=2)(data)
data= layers.Conv2D(filters=256, kernel_size=3, activation="relu")(data)
data = layers.MaxPooling2D(pool_size=2)(data)
data = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(data)
data= layers.Flatten()(data)
data = layers.Dropout(0.5)(data)
output = layers.Dense(1, activation="sigmoid")(data)
model = keras.Model(inputs=input, outputs=output)
model.compile(loss="binary_crossentropy",
optimizer="adam",
metrics=["accuracy"])
callbacks= [
keras.callbacks.ModelCheckpoint(
filepath="convnet_from_scratch_with_augmentation_info.keras",
save_best_only=True,
monitor="val_loss")
]
hist = model.fit(
train_data,
epochs=50,
validation_data=valid_data,
callbacks=callbacks)
```

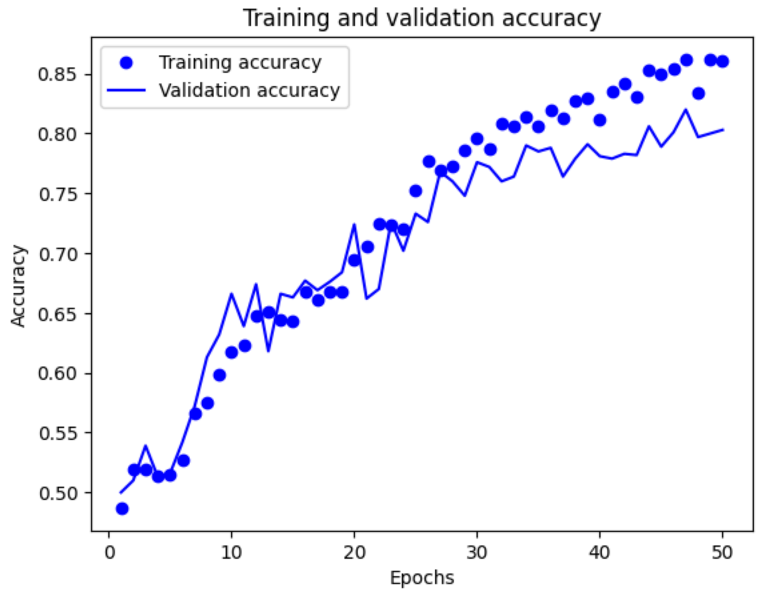
```
Epoch 22/50
63/63 [==============================] - 6s 95ms/step - loss: 0.5398 - accuracy: 0.7245 - val_loss: 0.6362 - val_accuracy: 0.6700
Epoch 23/50
63/63 [==============================] - 4s 59ms/step - loss: 0.5493 - accuracy: 0.7235 - val_loss: 0.5597 - val_accuracy: 0.7260
Epoch 24/50
63/63 [==============================] - 4s 58ms/step - loss: 0.5354 - accuracy: 0.7205 - val_loss: 0.6122 - val_accuracy: 0.7020
Epoch 25/50
63/63 [==============================] - 5s 80ms/step - loss: 0.5104 - accuracy: 0.7530 - val_loss: 0.5933 - val_accuracy: 0.7330
Epoch 26/50
63/63 [==============================] - 4s 61ms/step - loss: 0.4716 - accuracy: 0.7775 - val_loss: 0.5593 - val_accuracy: 0.7260
Epoch 27/50
63/63 [==============================] - 4s 65ms/step - loss: 0.4901 - accuracy: 0.7690 - val_loss: 0.5320 - val_accuracy: 0.7680
Epoch 28/50
63/63 [==============================] - 7s 112ms/step - loss: 0.4694 - accuracy: 0.7730 - val_loss: 0.5145 - val_accuracy: 0.7600
Epoch 29/50
63/63 [==============================] - 4s 58ms/step - loss: 0.4520 - accuracy: 0.7860 - val_loss: 0.5454 - val_accuracy: 0.7480
Epoch 30/50
63/63 [==============================] - 4s 59ms/step - loss: 0.4512 - accuracy: 0.7960 - val_loss: 0.5086 - val_accuracy: 0.7760
Epoch 31/50
63/63 [==============================] - 6s 96ms/step - loss: 0.4589 - accuracy: 0.7875 - val_loss: 0.5040 - val_accuracy: 0.7720
Epoch 32/50
63/63 [==============================] - 4s 60ms/step - loss: 0.4360 - accuracy: 0.8085 - val_loss: 0.5003 - val_accuracy: 0.7600
Epoch 33/50
63/63 [==============================] - 4s 58ms/step - loss: 0.4200 - accuracy: 0.8060 - val_loss: 0.5523 - val_accuracy: 0.7640
Epoch 34/50
63/63 [==============================] - 7s 113ms/step - loss: 0.4079 - accuracy: 0.8135 - val_loss: 0.4882 - val_accuracy: 0.7900
Epoch 35/50
63/63 [==============================] - 4s 61ms/step - loss: 0.4114 - accuracy: 0.8065 - val_loss: 0.4815 - val_accuracy: 0.7850
Epoch 36/50
63/63 [==============================] - 4s 58ms/step - loss: 0.3989 - accuracy: 0.8195 - val_loss: 0.4969 - val_accuracy: 0.7880
Epoch 37/50
63/63 [==============================] - 5s 78ms/step - loss: 0.4180 - accuracy: 0.8130 - val_loss: 0.4887 - val_accuracy: 0.7640
Epoch 38/50
63/63 [==============================] - 4s 59ms/step - loss: 0.3844 - accuracy: 0.8275 - val_loss: 0.5206 - val_accuracy: 0.7790
Epoch 39/50
63/63 [==============================] - 4s 65ms/step - loss: 0.3844 - accuracy: 0.8295 - val_loss: 0.5073 - val_accuracy: 0.7910
Epoch 40/50
63/63 [==============================] - 8s 115ms/step - loss: 0.3934 - accuracy: 0.8120 - val_loss: 0.4967 - val_accuracy: 0.7810
Epoch 41/50
63/63 [==============================] - 4s 58ms/step - loss: 0.3607 - accuracy: 0.8350 - val_loss: 0.5204 - val_accuracy: 0.7790
Epoch 42/50
63/63 [==============================] - 5s 83ms/step - loss: 0.3670 - accuracy: 0.8415 - val_loss: 0.5229 - val_accuracy: 0.7830
Epoch 43/50
63/63 [==============================] - 4s 60ms/step - loss: 0.3791 - accuracy: 0.8310 - val_loss: 0.5528 - val_accuracy: 0.7820
Epoch 44/50
63/63 [==============================] - 4s 59ms/step - loss: 0.3321 - accuracy: 0.8525 - val_loss: 0.4920 - val_accuracy: 0.8060
Epoch 45/50
63/63 [==============================] - 7s 97ms/step - loss: 0.3496 - accuracy: 0.8495 - val_loss: 0.5370 - val_accuracy: 0.7890
Epoch 46/50
63/63 [==============================] - 4s 59ms/step - loss: 0.3204 - accuracy: 0.8545 - val_loss: 0.5037 - val_accuracy: 0.8010
Epoch 47/50
63/63 [==============================] - 4s 57ms/step - loss: 0.3250 - accuracy: 0.8615 - val_loss: 0.4898 - val_accuracy: 0.8200
Epoch 48/50
63/63 [==============================] - 6s 97ms/step - loss: 0.3597 - accuracy: 0.8340 - val_loss: 0.4856 - val_accuracy: 0.7970
Epoch 49/50
63/63 [==============================] - 4s 58ms/step - loss: 0.3160 - accuracy: 0.8620 - val_loss: 0.5120 - val_accuracy: 0.8000
Epoch 50/50
63/63 [==============================] - 4s 59ms/step - loss: 0.3116 - accuracy: 0.8605 - val_loss: 0.5059 - val_accuracy: 0.8030
```

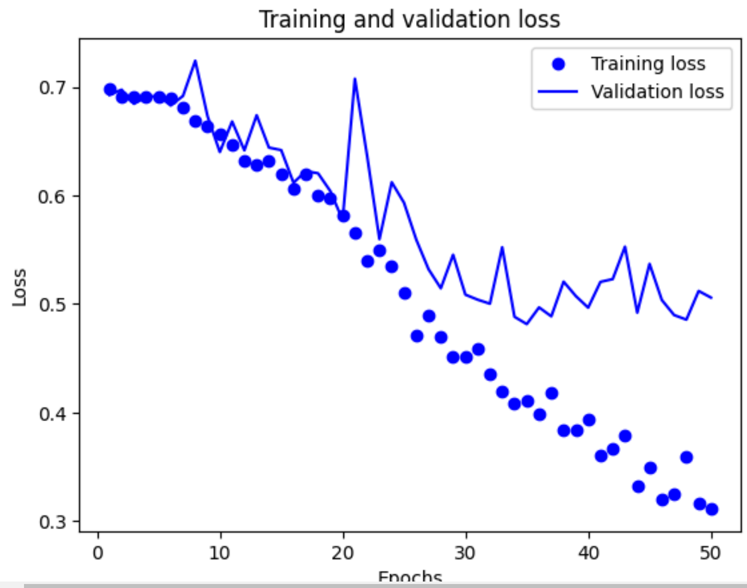
Curves of loss and accuracy during training were constructed

```
accuracy = hist.history["accuracy"]
val = hist.history["val_accuracy"]
loss = hist.history["loss"]
val_loss = hist.history["val_loss"]
epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```





Test Accuracy of model

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

```
32/32 [==============================] - 2s 46ms/step - loss: 0.4406 - accuracy: 0.8020
Test accuracy: 0.802
```

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)

```
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,exist_ok=True)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=org_dir / fname,
            dst=dir / fname)
make_subset("train", start_index=500, end_index=2500)
make_subset("validation", start_index=2500, end_index=3000)
make_subset("test", start_index=3000, end_index=3500)
input= keras.Input(shape=(180, 180, 3))
data_1 = augmentation_info(input)
data_1 = layers.Rescaling(1./255)(data_1)
data_1= layers.Conv2D(filters=32, kernel_size=3, activation="relu")(data_1)
data_1 = layers.MaxPooling2D(pool_size=2)(data_1)
data_1 = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(data_1)
data_1= layers.MaxPooling2D(pool_size=2)(data_1)
data_1= layers.Conv2D(filters=128, kernel_size=3, activation="relu")(data_1)
data_1= layers.MaxPooling2D(pool_size=2)(data_1)
data_1= layers.Conv2D(filters=256, kernel_size=3, activation="relu")(data_1)
data_1= layers.MaxPooling2D(pool_size=2)(data_1)
data_1= layers.Conv2D(filters=256, kernel_size=3, activation="relu")(data_1)
data_1 = layers.Flatten()(data_1)
data_1= layers.Dropout(0.5)(data_1)
output = layers.Dense(1, activation="sigmoid")(data_1)
model = keras.Model(inputs=input, outputs=output)
model.compile(loss="binary_crossentropy",
optimizer="adam",
metrics=["accuracy"])
callback = [
keras.callbacks.ModelCheckpoint(
filepath="convnet_from_scratch_with_augmentation_info.keras",
save_best_only=True,
monitor="val_loss")
]
hist = model.fit(
train_data,
epochs=50,
validation_data=valid_data,
callbacks=callback)
```

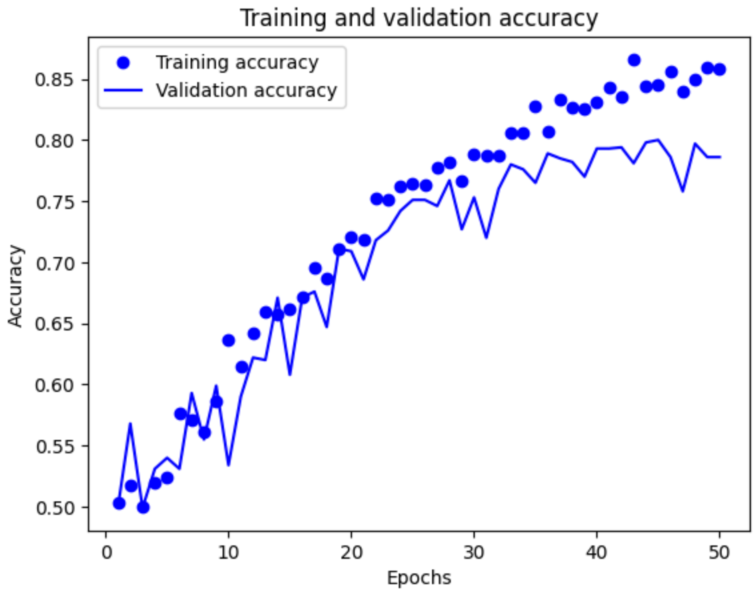
```
63/63 [==============================] - 6s 85ms/step - loss: 0.5021 - accuracy: 0.7625 - val_loss: 0.5429 - val_accuracy: 0.7420
Epoch 25/50
63/63 [==============================] - 4s 61ms/step - loss: 0.4969 - accuracy: 0.7645 - val_loss: 0.5399 - val_accuracy: 0.7510
Epoch 26/50
63/63 [==============================] - 7s 112ms/step - loss: 0.4950 - accuracy: 0.7635 - val_loss: 0.5212 - val_accuracy: 0.7510
Epoch 27/50
63/63 [==============================] - 4s 59ms/step - loss: 0.4648 - accuracy: 0.7770 - val_loss: 0.5080 - val_accuracy: 0.7460
Epoch 28/50
63/63 [==============================] - 4s 59ms/step - loss: 0.4600 - accuracy: 0.7815 - val_loss: 0.5107 - val_accuracy: 0.7670
Epoch 29/50
63/63 [==============================] - 7s 103ms/step - loss: 0.4760 - accuracy: 0.7665 - val_loss: 0.5413 - val_accuracy: 0.7270
Epoch 30/50
63/63 [==============================] - 4s 58ms/step - loss: 0.4522 - accuracy: 0.7880 - val_loss: 0.5096 - val_accuracy: 0.7530
Epoch 31/50
63/63 [==============================] - 5s 84ms/step - loss: 0.4522 - accuracy: 0.7875 - val_loss: 0.5851 - val_accuracy: 0.7200
Epoch 32/50
63/63 [==============================] - 4s 58ms/step - loss: 0.4414 - accuracy: 0.7875 - val_loss: 0.5486 - val_accuracy: 0.7600
Epoch 33/50
63/63 [==============================] - 5s 74ms/step - loss: 0.4184 - accuracy: 0.8055 - val_loss: 0.5026 - val_accuracy: 0.7800
Epoch 34/50
63/63 [==============================] - 6s 90ms/step - loss: 0.4202 - accuracy: 0.8060 - val_loss: 0.4930 - val_accuracy: 0.7760
Epoch 35/50
63/63 [==============================] - 4s 58ms/step - loss: 0.3862 - accuracy: 0.8275 - val_loss: 0.5570 - val_accuracy: 0.7650
Epoch 36/50
63/63 [==============================] - 7s 103ms/step - loss: 0.4006 - accuracy: 0.8065 - val_loss: 0.4778 - val_accuracy: 0.7890
Epoch 37/50
63/63 [==============================] - 5s 68ms/step - loss: 0.3603 - accuracy: 0.8330 - val_loss: 0.5639 - val_accuracy: 0.7850
Epoch 38/50
63/63 [==============================] - 4s 58ms/step - loss: 0.3872 - accuracy: 0.8270 - val_loss: 0.5236 - val_accuracy: 0.7820
Epoch 39/50
63/63 [==============================] - 6s 94ms/step - loss: 0.3802 - accuracy: 0.8255 - val_loss: 0.5038 - val_accuracy: 0.7700
Epoch 40/50
63/63 [==============================] - 6s 81ms/step - loss: 0.3746 - accuracy: 0.8315 - val_loss: 0.5284 - val_accuracy: 0.7930
Epoch 41/50
63/63 [==============================] - 4s 61ms/step - loss: 0.3510 - accuracy: 0.8435 - val_loss: 0.5113 - val_accuracy: 0.7930
Epoch 42/50
63/63 [==============================] - 7s 111ms/step - loss: 0.3602 - accuracy: 0.8355 - val_loss: 0.4871 - val_accuracy: 0.7940
Epoch 43/50
63/63 [==============================] - 4s 58ms/step - loss: 0.3251 - accuracy: 0.8660 - val_loss: 0.6773 - val_accuracy: 0.7810
Epoch 44/50
63/63 [==============================] - 5s 77ms/step - loss: 0.3522 - accuracy: 0.8445 - val_loss: 0.4857 - val_accuracy: 0.7980
Epoch 45/50
63/63 [==============================] - 4s 59ms/step - loss: 0.3418 - accuracy: 0.8450 - val_loss: 0.5075 - val_accuracy: 0.8000
Epoch 46/50
63/63 [==============================] - 5s 84ms/step - loss: 0.3254 - accuracy: 0.8560 - val_loss: 0.5616 - val_accuracy: 0.7860
Epoch 47/50
63/63 [==============================] - 4s 62ms/step - loss: 0.3494 - accuracy: 0.8395 - val_loss: 0.5858 - val_accuracy: 0.7580
Epoch 48/50
63/63 [==============================] - 5s 69ms/step - loss: 0.3306 - accuracy: 0.8500 - val_loss: 0.5257 - val_accuracy: 0.7970
Epoch 49/50
63/63 [==============================] - 7s 97ms/step - loss: 0.3178 - accuracy: 0.8595 - val_loss: 0.5404 - val_accuracy: 0.7860
Epoch 50/50
63/63 [==============================] - 4s 58ms/step - loss: 0.3129 - accuracy: 0.8585 - val_loss: 0.5153 - val_accuracy: 0.7860
```

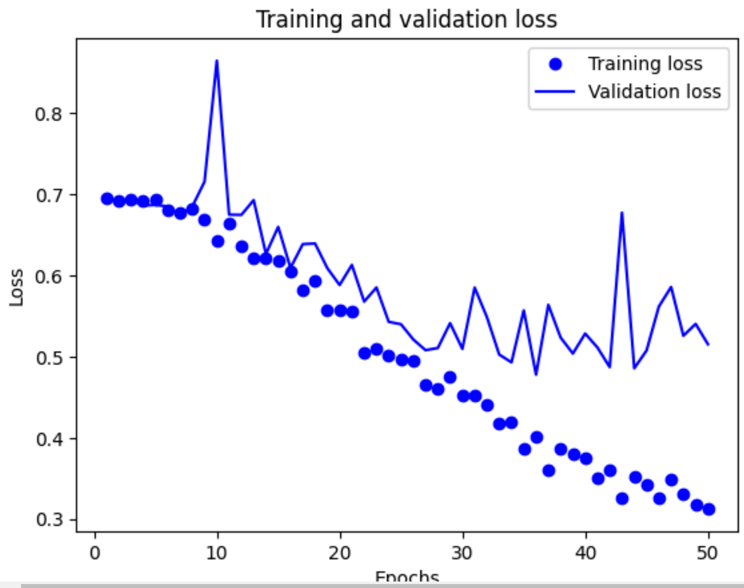
Curves of loss and accuracy during training

```
accuracy = hist.history["accuracy"]
validation = hist.history["val_accuracy"]
loss = hist.history["loss"]
valloss = hist.history["val_loss"]
epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, validation, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, valloss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```





Test Accuracy of model

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

```
32/32 [==============================] - 2s 30ms/step - loss: 0.4511 - accuracy: 0.7940
Test accuracy: 0.794
```

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

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

```
convoluted.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

```
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.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_data)
val_features, val_labels = get_features_and_labels(valid_data)
test_features, test_labels = get_features_and_labels(test_data)
```

```
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 18ms/step
```

```
train_features.shape
```

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

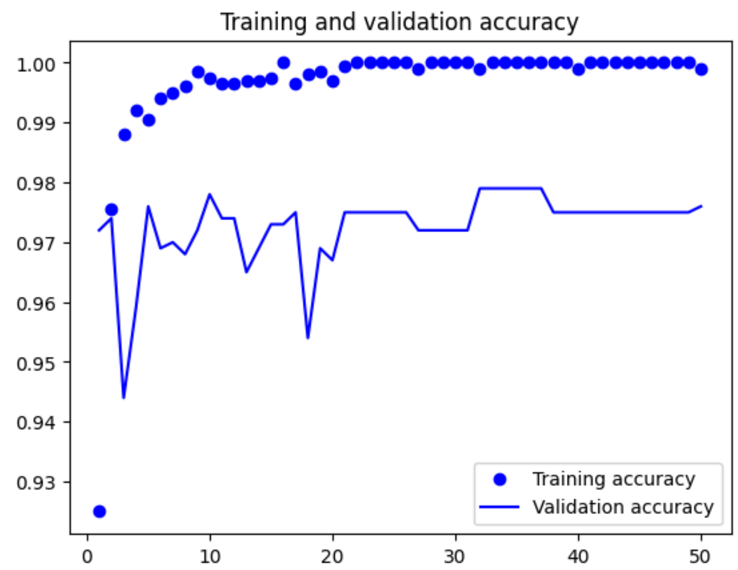
Model Fitting

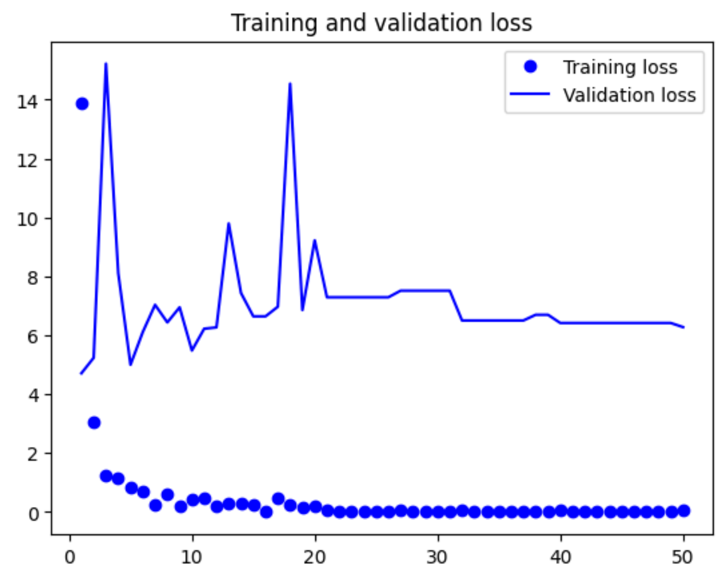
```
input = keras.Input(shape=(5, 5, 512))
data_2 = layers.Flatten()(input)
data_2 = layers.Dense(256)(data_2)
data_2 = layers.Dropout(0.5)(data_2)
out = layers.Dense(1, activation="sigmoid")(data_2)
model = keras.Model(input, out)
model.compile(loss="binary_crossentropy",
optimizer="rmsprop",
metrics=["accuracy"])
callback= [
keras.callbacks.ModelCheckpoint(
filepath="feature_extraction.keras",
save_best_only=True,
monitor="val_loss")
]
history = model.fit(
train_features, train_labels,
epochs=50,
validation_data=(val_features, val_labels),
callbacks=callback)
```

```
Epoch 29/50
63/63 [==============================] - 1s 9ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 7.5133 - val_accuracy: 0.9720
Epoch 30/50
63/63 [==============================] - 0s 8ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 7.5133 - val_accuracy: 0.9720
Epoch 31/50
63/63 [==============================] - 0s 7ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 7.5133 - val_accuracy: 0.9720
Epoch 32/50
63/63 [==============================] - 1s 8ms/step - loss: 0.0537 - accuracy: 0.9990 - val_loss: 6.5020 - val_accuracy: 0.9790
Epoch 33/50
63/63 [==============================] - 1s 8ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 6.5020 - val_accuracy: 0.9790
Epoch 34/50
63/63 [==============================] - 1s 8ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 6.5020 - val_accuracy: 0.9790
Epoch 35/50
63/63 [==============================] - 1s 8ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 6.5020 - val_accuracy: 0.9790
Epoch 36/50
63/63 [==============================] - 0s 8ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 6.5020 - val_accuracy: 0.9790
Epoch 37/50
63/63 [==============================] - 0s 8ms/step - loss: 4.4554e-27 - accuracy: 1.0000 - val_loss: 6.5020 - val_accuracy: 0.9790
Epoch 38/50
63/63 [==============================] - 0s 8ms/step - loss: 6.3486e-07 - accuracy: 1.0000 - val_loss: 6.6957 - val_accuracy: 0.9750
Epoch 39/50
63/63 [==============================] - 1s 8ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 6.6957 - val_accuracy: 0.9750
Epoch 40/50
63/63 [==============================] - 1s 9ms/step - loss: 0.0821 - accuracy: 0.9990 - val_loss: 6.4174 - val_accuracy: 0.9750
Epoch 41/50
63/63 [==============================] - 1s 8ms/step - loss: 4.4931e-23 - accuracy: 1.0000 - val_loss: 6.4174 - val_accuracy: 0.9750
Epoch 42/50
63/63 [==============================] - 0s 6ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 6.4174 - val_accuracy: 0.9750
Epoch 43/50
63/63 [==============================] - 0s 5ms/step - loss: 5.6599e-24 - accuracy: 1.0000 - val_loss: 6.4174 - val_accuracy: 0.9750
Epoch 44/50
63/63 [==============================] - 0s 5ms/step - loss: 1.8735e-19 - accuracy: 1.0000 - val_loss: 6.4174 - val_accuracy: 0.9750
Epoch 45/50
63/63 [==============================] - 0s 5ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 6.4174 - val_accuracy: 0.9750
Epoch 46/50
63/63 [==============================] - 0s 5ms/step - loss: 2.6489e-35 - accuracy: 1.0000 - val_loss: 6.4174 - val_accuracy: 0.9750
Epoch 47/50
63/63 [==============================] - 0s 7ms/step - loss: 2.0297e-33 - accuracy: 1.0000 - val_loss: 6.4174 - val_accuracy: 0.9750
Epoch 48/50
63/63 [==============================] - 0s 5ms/step - loss: 9.6952e-33 - accuracy: 1.0000 - val_loss: 6.4174 - val_accuracy: 0.9750
Epoch 49/50
63/63 [==============================] - 0s 6ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 6.4174 - val_accuracy: 0.9750
Epoch 50/50
63/63 [==============================] - 0s 6ms/step - loss: 0.0835 - accuracy: 0.9990 - val_loss: 6.2767 - val_accuracy: 0.9760
```

Curves of loss and accuracy during training

```
accur = history.history["accuracy"]
valac= history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accur) + 1)
plt.plot(epochs, accur, "bo", label="Training accuracy")
plt.plot(epochs, valac, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
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()
```





Freezing and Unfreezing the Pre-trained Convolutional Base

```
convoluted = keras.applications.vgg16.VGG16(
weights="imagenet",
include_top=False)
convoluted.trainable = False
convoluted.trainable = True
print("This is the number of trainable weights "
"before freezing the conv base:", len(convoluted.trainable_weights))
convoluted.trainable = False
```

```
print("This is the number of trainable weights "
"after freezing the conv base:", len(convoluted.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

```
augmented= keras.Sequential(
[
layers.RandomFlip("horizontal"),
layers.RandomRotation(0.1),
layers.RandomZoom(0.2),
]
)
input = keras.Input(shape=(180, 180, 3))
data_3= augmented(input)
data_3=keras.layers.Lambda(
lambda x: keras.applications.vgg16.preprocess_input(x))(data_3)
data_3= convoluted(data_3)
data_3 = layers.Flatten()(data_3)
data_3 = layers.Dense(256)(data_3)
data_3= layers.Dropout(0.5)(data_3)
outputs = layers.Dense(1, activation="sigmoid")(data_3)
model = keras.Model(input, outputs)
model.compile(loss="binary_crossentropy",
optimizer="rmsprop",
metrics=["accuracy"])
callback = [
keras.callbacks.ModelCheckpoint(
filepath="features_extraction_with_augmentation2.keras",
save_best_only=True,
monitor="val_loss"
)
]
history= model.fit(
train_data,
epochs=100,
validation_data=valid_data,
callbacks=callback
)
```

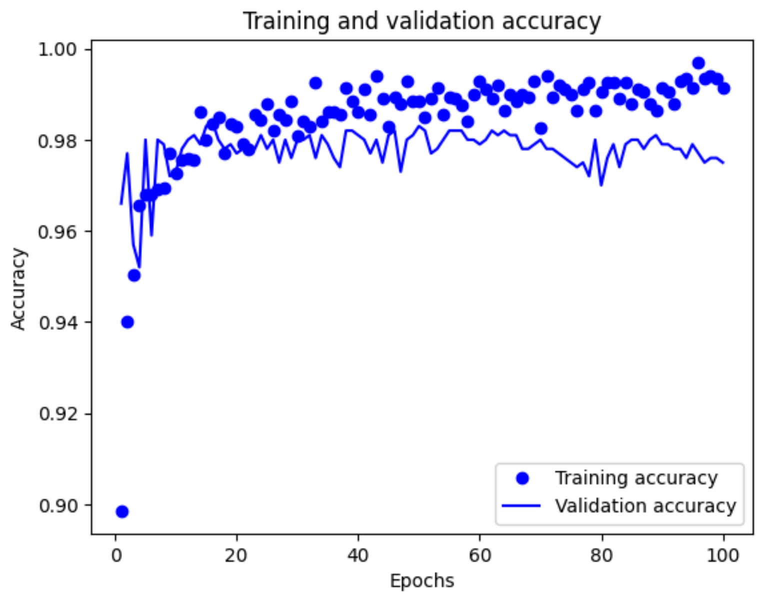
```
Epoch 1/100
63/63 [==============================] - 14s 196ms/step - loss: 15.6539 - accuracy: 0.8985 - val_loss: 4.5115 - val_accuracy: 0.9660
Epoch 2/100
63/63 [==============================] - 11s 172ms/step - loss: 6.7755 - accuracy: 0.9400 - val_loss: 4.0749 - val_accuracy: 0.9770
Epoch 3/100
63/63 [==============================] - 10s 159ms/step - loss: 6.4133 - accuracy: 0.9505 - val_loss: 7.9471 - val_accuracy: 0.9570
Epoch 4/100
63/63 [==============================] - 10s 158ms/step - loss: 4.7811 - accuracy: 0.9655 - val_loss: 9.6755 - val_accuracy: 0.9520
Epoch 5/100
63/63 [==============================] - 12s 191ms/step - loss: 3.3019 - accuracy: 0.9680 - val_loss: 3.5963 - val_accuracy: 0.9800
Epoch 6/100
63/63 [==============================] - 10s 152ms/step - loss: 3.5062 - accuracy: 0.9680 - val_loss: 8.2014 - val_accuracy: 0.9590
Epoch 7/100
63/63 [==============================] - 11s 161ms/step - loss: 3.4003 - accuracy: 0.9690 - val_loss: 3.5877 - val_accuracy: 0.9800
Epoch 8/100
63/63 [==============================] - 11s 166ms/step - loss: 3.4171 - accuracy: 0.9695 - val_loss: 3.5269 - val_accuracy: 0.9790
Epoch 9/100
63/63 [==============================] - 10s 159ms/step - loss: 2.9296 - accuracy: 0.9770 - val_loss: 4.8720 - val_accuracy: 0.9720
Epoch 10/100
63/63 [==============================] - 12s 184ms/step - loss: 3.6618 - accuracy: 0.9725 - val_loss: 4.0819 - val_accuracy: 0.9730
Epoch 11/100
63/63 [==============================] - 11s 169ms/step - loss: 2.4631 - accuracy: 0.9755 - val_loss: 3.4436 - val_accuracy: 0.9780
Epoch 12/100
63/63 [==============================] - 12s 194ms/step - loss: 1.9736 - accuracy: 0.9760 - val_loss: 2.1654 - val_accuracy: 0.9800
Epoch 13/100
63/63 [==============================] - 10s 158ms/step - loss: 1.9307 - accuracy: 0.9755 - val_loss: 2.4559 - val_accuracy: 0.9810
Epoch 14/100
63/63 [==============================] - 12s 184ms/step - loss: 1.4026 - accuracy: 0.9860 - val_loss: 2.5853 - val_accuracy: 0.9790
Epoch 15/100
63/63 [==============================] - 12s 190ms/step - loss: 1.7425 - accuracy: 0.9800 - val_loss: 1.8625 - val_accuracy: 0.9830
Epoch 16/100
63/63 [==============================] - 12s 184ms/step - loss: 1.1059 - accuracy: 0.9835 - val_loss: 2.1287 - val_accuracy: 0.9840
Epoch 17/100
63/63 [==============================] - 10s 161ms/step - loss: 1.0701 - accuracy: 0.9850 - val_loss: 2.4242 - val_accuracy: 0.9800
Epoch 18/100
63/63 [==============================] - 12s 187ms/step - loss: 1.7785 - accuracy: 0.9770 - val_loss: 2.4853 - val_accuracy: 0.9780
Epoch 19/100
63/63 [==============================] - 10s 160ms/step - loss: 1.6377 - accuracy: 0.9835 - val_loss: 1.9899 - val_accuracy: 0.9790
Epoch 20/100
63/63 [==============================] - 10s 160ms/step - loss: 1.5781 - accuracy: 0.9830 - val_loss: 2.3843 - val_accuracy: 0.9770
Epoch 21/100
63/63 [==============================] - 12s 184ms/step - loss: 1.5453 - accuracy: 0.9790 - val_loss: 2.6491 - val_accuracy: 0.9780
Epoch 22/100
63/63 [==============================] - 10s 161ms/step - loss: 1.5617 - accuracy: 0.9780 - val_loss: 2.3315 - val_accuracy: 0.9790
Epoch 23/100
63/63 [==============================] - 10s 158ms/step - loss: 1.2869 - accuracy: 0.9855 - val_loss: 2.1203 - val_accuracy: 0.9780
Epoch 24/100
63/63 [==============================] - 10s 157ms/step - loss: 0.8061 - accuracy: 0.9845 - val_loss: 2.2546 - val_accuracy: 0.9810
Epoch 25/100
63/63 [==============================] - 11s 166ms/step - loss: 0.5307 - accuracy: 0.9880 - val_loss: 3.2337 - val_accuracy: 0.9780
Epoch 26/100
63/63 [==============================] - 12s 186ms/step - loss: 1.3641 - accuracy: 0.9820 - val_loss: 2.2655 - val_accuracy: 0.9800
Epoch 27/100
63/63 [==============================] - 10s 156ms/step - loss: 0.6379 - accuracy: 0.9855 - val_loss: 3.0918 - val_accuracy: 0.9750
Epoch 28/100
63/63 [==============================] - 10s 154ms/step - loss: 0.8963 - accuracy: 0.9845 - val_loss: 2.6102 - val_accuracy: 0.9800
Epoch 29/100
63/63 [==============================] - 10s 156ms/step - loss: 0.5936 - accuracy: 0.9885 - val_loss: 2.3776 - val_accuracy: 0.9760
```

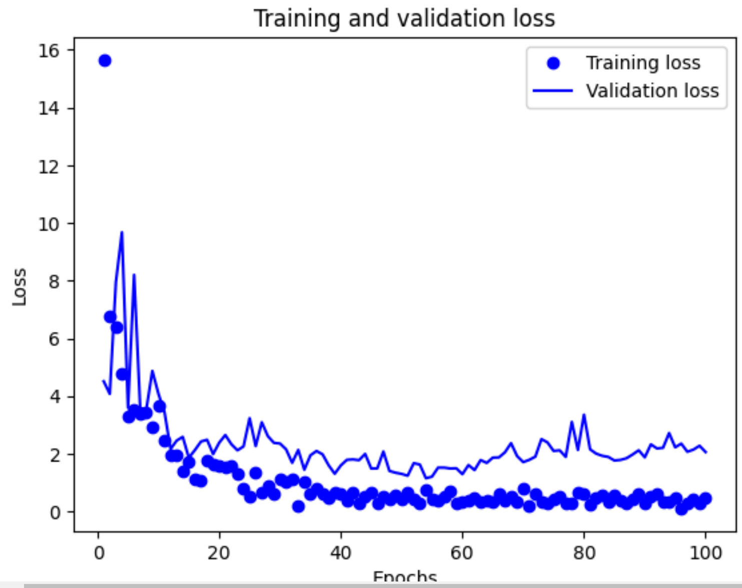
Curves of loss and accuracy during training

```
accuracy_1 = history.history["accuracy"]
validation= history.history["val_accuracy"]
loss = history.history["loss"]
valloss = history.history["val_loss"]
epochs = range(1, len(accuracy_1) + 1)
plt.plot(epochs, accuracy_1, "bo", label="Training accuracy")
plt.plot(epochs, validation, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, valloss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
```

```
plt.ylabel("Loss")
plt.legend()
plt.show()
```





Test Accuracy of model

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

```
32/32 [==============================] - 4s 95ms/step - loss: 2.8497 - accuracy: 0.9670
Test accuracy: 0.967
```

Fine-tuning a pretrained model

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

model.compile(loss="binary_crossentropy",
optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
metrics=["accuracy"])
callback = [
keras.callbacks.ModelCheckpoint(
filepath="fine_tuning.keras",
save_best_only=True,
monitor="val_loss")
]
historytuning = model.fit(
train_data,
epochs=50,
validation_data=valid_data,
callbacks=callback)
```

```
Epoch 1/50
63/63 [==============================] - 14s 190ms/step - loss: 0.2360 - accuracy: 0.9925 - val_loss: 2.1644 - val_accuracy: 0.9780
Epoch 2/50
63/63 [==============================] - 12s 188ms/step - loss: 0.2228 - accuracy: 0.9930 - val_loss: 1.7981 - val_accuracy: 0.9770
Epoch 3/50
63/63 [==============================] - 12s 182ms/step - loss: 0.4866 - accuracy: 0.9935 - val_loss: 3.1030 - val_accuracy: 0.9750
Epoch 4/50
63/63 [==============================] - 13s 209ms/step - loss: 0.3216 - accuracy: 0.9940 - val_loss: 1.7541 - val_accuracy: 0.9780
Epoch 5/50
63/63 [==============================] - 13s 196ms/step - loss: 0.1992 - accuracy: 0.9950 - val_loss: 1.8541 - val_accuracy: 0.9780
Epoch 6/50
63/63 [==============================] - 11s 169ms/step - loss: 0.2046 - accuracy: 0.9940 - val_loss: 1.8240 - val_accuracy: 0.9740
Epoch 7/50
63/63 [==============================] - 11s 176ms/step - loss: 0.1288 - accuracy: 0.9960 - val_loss: 1.8414 - val_accuracy: 0.9760
Epoch 8/50
63/63 [==============================] - 11s 168ms/step - loss: 0.1415 - accuracy: 0.9960 - val_loss: 2.1445 - val_accuracy: 0.9760
Epoch 9/50
63/63 [==============================] - 11s 174ms/step - loss: 0.2522 - accuracy: 0.9940 - val_loss: 2.2084 - val_accuracy: 0.9730
Epoch 10/50
63/63 [==============================] - 11s 179ms/step - loss: 0.2158 - accuracy: 0.9960 - val_loss: 2.5019 - val_accuracy: 0.9740
Epoch 11/50
63/63 [==============================] - 11s 170ms/step - loss: 0.0739 - accuracy: 0.9960 - val_loss: 2.4763 - val_accuracy: 0.9710
Epoch 12/50
63/63 [==============================] - 12s 176ms/step - loss: 0.1302 - accuracy: 0.9950 - val_loss: 2.0779 - val_accuracy: 0.9760
Epoch 13/50
63/63 [==============================] - 12s 195ms/step - loss: 0.2289 - accuracy: 0.9950 - val_loss: 2.1873 - val_accuracy: 0.9720
Epoch 14/50
63/63 [==============================] - 11s 173ms/step - loss: 0.0927 - accuracy: 0.9965 - val_loss: 1.9579 - val_accuracy: 0.9760
Epoch 15/50
63/63 [==============================] - 12s 186ms/step - loss: 0.2475 - accuracy: 0.9930 - val_loss: 1.8287 - val_accuracy: 0.9790
Epoch 16/50
63/63 [==============================] - 13s 200ms/step - loss: 0.1318 - accuracy: 0.9945 - val_loss: 2.6297 - val_accuracy: 0.9750
Epoch 17/50
63/63 [==============================] - 13s 199ms/step - loss: 0.2323 - accuracy: 0.9960 - val_loss: 2.0161 - val_accuracy: 0.9790
```

```
Epoch 18/50
63/63 [==============================] - 11s 171ms/step - loss: 0.2604 - accuracy: 0.9950 - val_loss: 1.9839 - val_accuracy: 0.9780
Epoch 19/50
63/63 [==============================] - 12s 180ms/step - loss: 0.1996 - accuracy: 0.9955 - val_loss: 1.5511 - val_accuracy: 0.9780
Epoch 20/50
63/63 [==============================] - 13s 199ms/step - loss: 0.0873 - accuracy: 0.9960 - val_loss: 1.7946 - val_accuracy: 0.9820
Epoch 21/50
63/63 [==============================] - 12s 190ms/step - loss: 0.1611 - accuracy: 0.9955 - val_loss: 1.5405 - val_accuracy: 0.9820
Epoch 22/50
63/63 [==============================] - 13s 200ms/step - loss: 0.0463 - accuracy: 0.9975 - val_loss: 1.7614 - val_accuracy: 0.9850
Epoch 23/50
63/63 [==============================] - 11s 179ms/step - loss: 0.1729 - accuracy: 0.9935 - val_loss: 2.3032 - val_accuracy: 0.9800
Epoch 24/50
63/63 [==============================] - 11s 175ms/step - loss: 0.1969 - accuracy: 0.9970 - val_loss: 1.8171 - val_accuracy: 0.9820
Epoch 25/50
63/63 [==============================] - 11s 180ms/step - loss: 0.0230 - accuracy: 0.9980 - val_loss: 1.9318 - val_accuracy: 0.9750
Epoch 26/50
63/63 [==============================] - 13s 201ms/step - loss: 0.0819 - accuracy: 0.9975 - val_loss: 1.5795 - val_accuracy: 0.9810
Epoch 27/50
63/63 [==============================] - 13s 199ms/step - loss: 0.2035 - accuracy: 0.9960 - val_loss: 1.9160 - val_accuracy: 0.9770
Epoch 28/50
63/63 [==============================] - 11s 177ms/step - loss: 0.1780 - accuracy: 0.9935 - val_loss: 1.9333 - val_accuracy: 0.9800
Epoch 29/50
63/63 [==============================] - 11s 178ms/step - loss: 0.2058 - accuracy: 0.9940 - val_loss: 1.6828 - val_accuracy: 0.9760
```

Curves of loss and accuracy during training

```
accuracy_2= historytuning.history["accuracy"]
val_accuracy_tune = historytuning.history["val_accuracy"]
loss_tune = historytuning.history["loss"]
val_loss = historytuning.history["val_loss"]
epochs = range(1, len(accuracy_2) + 1)

plt.plot(epochs, accuracy_2, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy_tune, "b", label="Validation accuracy")
plt.title("Fine-tuning: Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, loss_tune, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Fine-tuning: Training and validation loss")
plt.xlabel("Epochs")
```