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) - 65 bytes, last modified: 3/25/2025 - 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
```

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

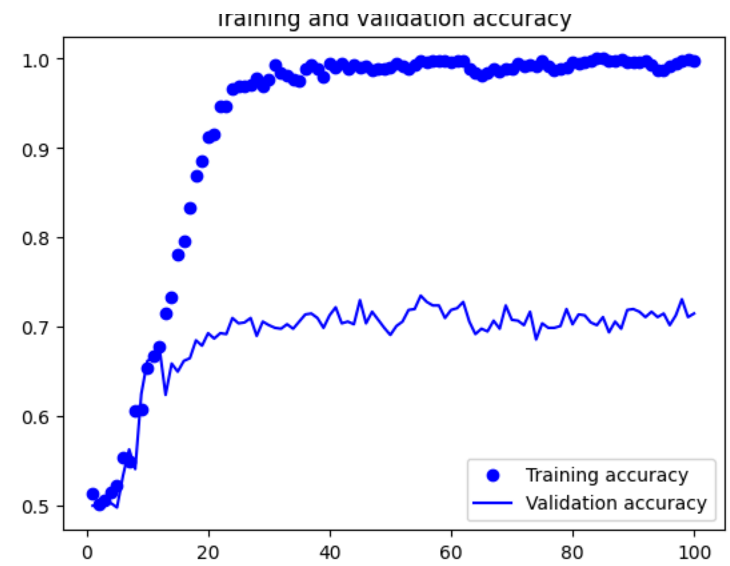
Model Fitting

```
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)
```

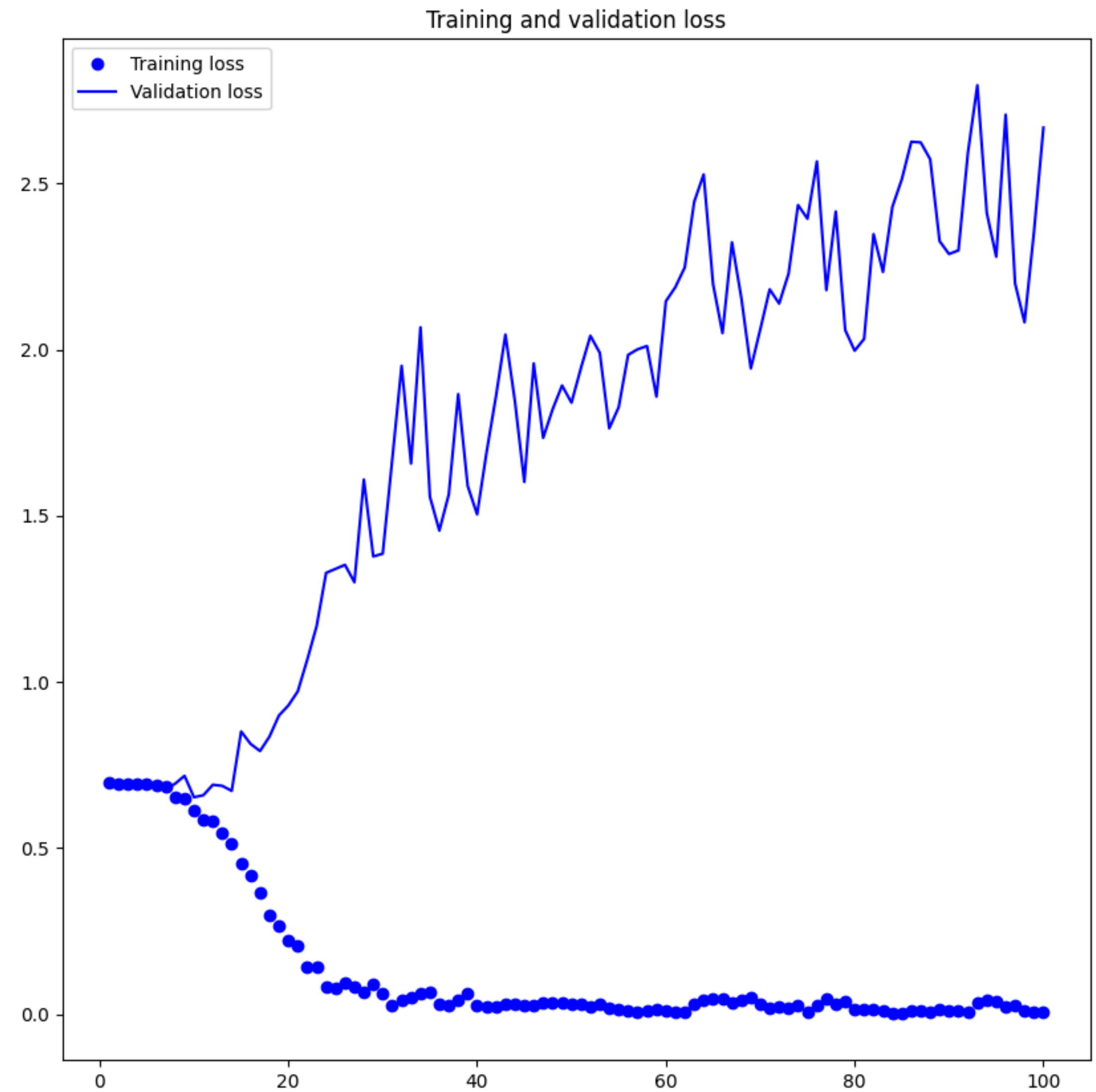
```
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 73ms/step - accuracy: 0.9958 - loss: 0.0203 - val_accuracy: 0.7190 - val_loss: 2.3264
Epoch 90/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - accuracy: 0.9928 - loss: 0.0172 - val_accuracy: 0.7200 - val_loss: 2.2877
Epoch 91/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.9955 - loss: 0.0122 - val_accuracy: 0.7170 - val_loss: 2.2988
Epoch 92/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 64ms/step - accuracy: 0.9964 - loss: 0.0059 - val_accuracy: 0.7110 - val_loss: 2.5922
Epoch 93/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - accuracy: 0.9956 - loss: 0.0191 - val_accuracy: 0.7170 - val_loss: 2.7956
Epoch 94/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 59ms/step - accuracy: 0.9837 - loss: 0.0614 - val_accuracy: 0.7110 - val_loss: 2.4112
Epoch 95/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9885 - loss: 0.0290 - val_accuracy: 0.7150 - val_loss: 2.2796
Epoch 96/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.9899 - loss: 0.0269 - val_accuracy: 0.7020 - val_loss: 2.7071
Epoch 97/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.9922 - loss: 0.0249 - val_accuracy: 0.7130 - val_loss: 2.1988
Epoch 98/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.9994 - loss: 0.0064 - val_accuracy: 0.7310 - val_loss: 2.0823
Epoch 99/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 48ms/step - accuracy: 0.9993 - loss: 0.0056 - val_accuracy: 0.7110 - val_loss: 2.3543
Epoch 100/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 61ms/step - accuracy: 0.9993 - loss: 0.0031 - val_accuracy: 0.7150 - val_loss: 2.6681
```

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}")
```

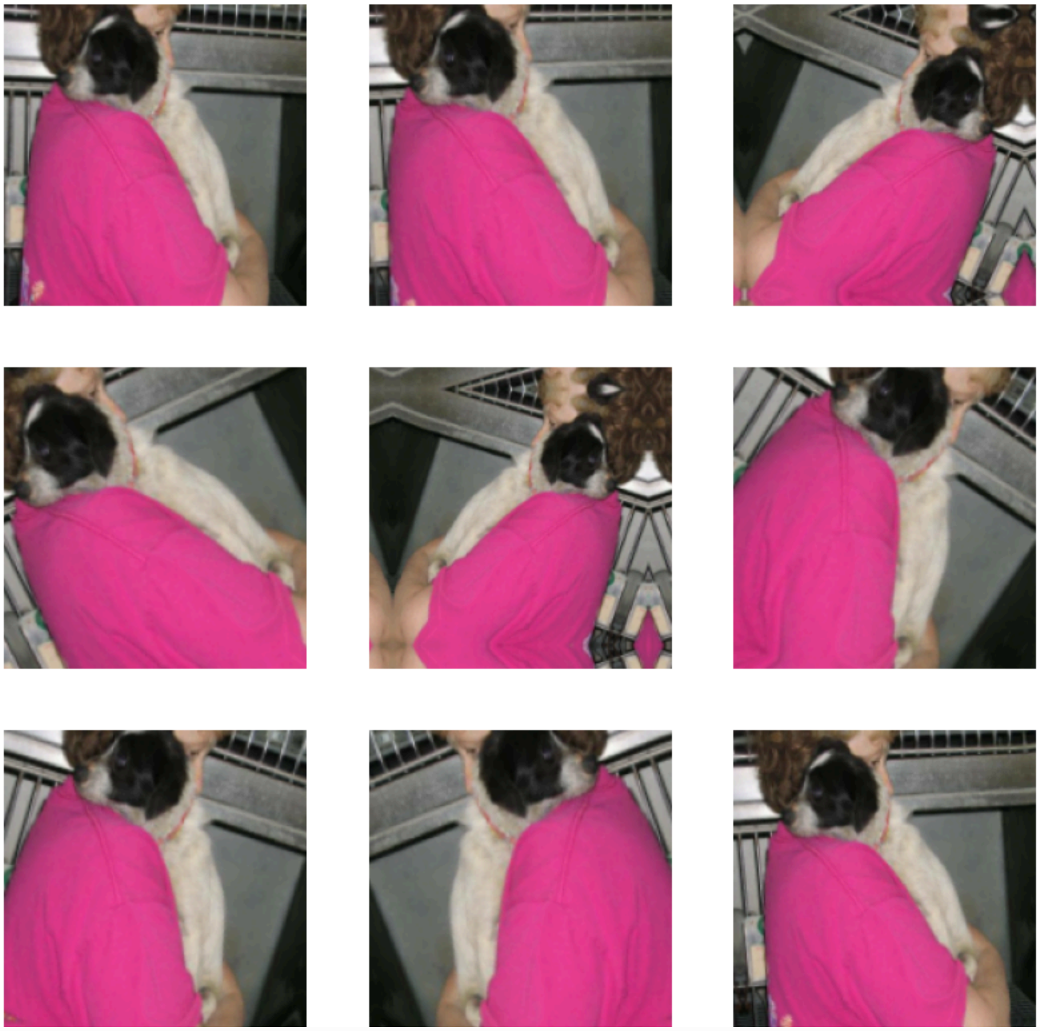
```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.6574 - loss: 0.6460
Test accuracy: 0.655
```

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 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.7755 - loss: 0.4705 - val_accuracy: 0.7730 - val_loss: 0.5060
Epoch 23/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 50ms/step - accuracy: 0.7705 - loss: 0.4665 - val_accuracy: 0.7430 - val_loss: 0.5699
Epoch 24/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.7977 - loss: 0.4321 - val_accuracy: 0.7590 - val_loss: 0.5087
Epoch 25/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 51ms/step - accuracy: 0.7662 - loss: 0.4676 - val_accuracy: 0.7660 - val_loss: 0.4959
Epoch 26/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 50ms/step - accuracy: 0.7875 - loss: 0.4393 - val_accuracy: 0.7760 - val_loss: 0.5118
Epoch 27/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 49ms/step - accuracy: 0.7893 - loss: 0.4253 - val_accuracy: 0.7780 - val_loss: 0.5109
Epoch 28/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 56ms/step - accuracy: 0.8157 - loss: 0.4191 - val_accuracy: 0.7760 - val_loss: 0.5254
Epoch 29/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 73ms/step - accuracy: 0.7978 - loss: 0.4124 - val_accuracy: 0.7900 - val_loss: 0.4900
Epoch 30/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 51ms/step - accuracy: 0.8100 - loss: 0.4170 - val_accuracy: 0.8020 - val_loss: 0.4645
Epoch 31/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 78ms/step - accuracy: 0.8198 - loss: 0.3948 - val_accuracy: 0.7810 - val_loss: 0.5008
Epoch 32/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.8316 - loss: 0.3536 - val_accuracy: 0.7880 - val_loss: 0.4543
Epoch 33/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 49ms/step - accuracy: 0.8125 - loss: 0.3923 - val_accuracy: 0.7980 - val_loss: 0.4605
Epoch 34/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 68ms/step - accuracy: 0.8292 - loss: 0.3714 - val_accuracy: 0.8000 - val_loss: 0.4596
Epoch 35/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.8477 - loss: 0.3672 - val_accuracy: 0.8020 - val_loss: 0.4881
Epoch 36/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 50ms/step - accuracy: 0.8476 - loss: 0.3364 - val_accuracy: 0.8110 - val_loss: 0.4604
Epoch 37/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.8578 - loss: 0.3388 - val_accuracy: 0.8000 - val_loss: 0.5019
Epoch 38/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 49ms/step - accuracy: 0.8437 - loss: 0.3472 - val_accuracy: 0.8020 - val_loss: 0.4985
Epoch 39/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 57ms/step - accuracy: 0.8609 - loss: 0.3399 - val_accuracy: 0.7930 - val_loss: 0.4872
Epoch 40/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.8616 - loss: 0.3333 - val_accuracy: 0.8240 - val_loss: 0.4824
Epoch 41/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 49ms/step - accuracy: 0.8589 - loss: 0.3189 - val_accuracy: 0.8010 - val_loss: 0.5552
Epoch 42/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.8628 - loss: 0.3131 - val_accuracy: 0.7900 - val_loss: 0.4840
Epoch 43/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 75ms/step - accuracy: 0.8574 - loss: 0.3048 - val_accuracy: 0.7880 - val_loss: 0.5040
Epoch 44/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.8822 - loss: 0.2870 - val_accuracy: 0.8060 - val_loss: 0.4605
Epoch 45/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.8856 - loss: 0.2763 - val_accuracy: 0.8100 - val_loss: 0.4719
Epoch 46/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.8679 - loss: 0.2960 - val_accuracy: 0.8220 - val_loss: 0.4890
Epoch 47/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.8752 - loss: 0.2916 - val_accuracy: 0.8120 - val_loss: 0.4716
Epoch 48/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 49ms/step - accuracy: 0.8882 - loss: 0.2696 - val_accuracy: 0.8150 - val_loss: 0.4780
Epoch 49/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 60ms/step - accuracy: 0.8806 - loss: 0.2742 - val_accuracy: 0.8070 - val_loss: 0.4786
Epoch 50/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 50ms/step - accuracy: 0.8918 - loss: 0.2632 - val_accuracy: 0.8070 - val_loss: 0.5426
```

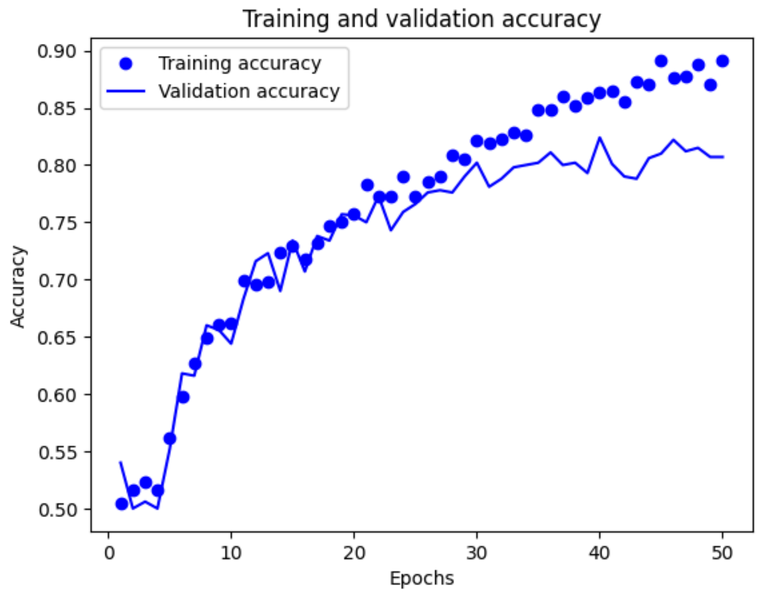
Curves of loss and accuracy during training were constructed

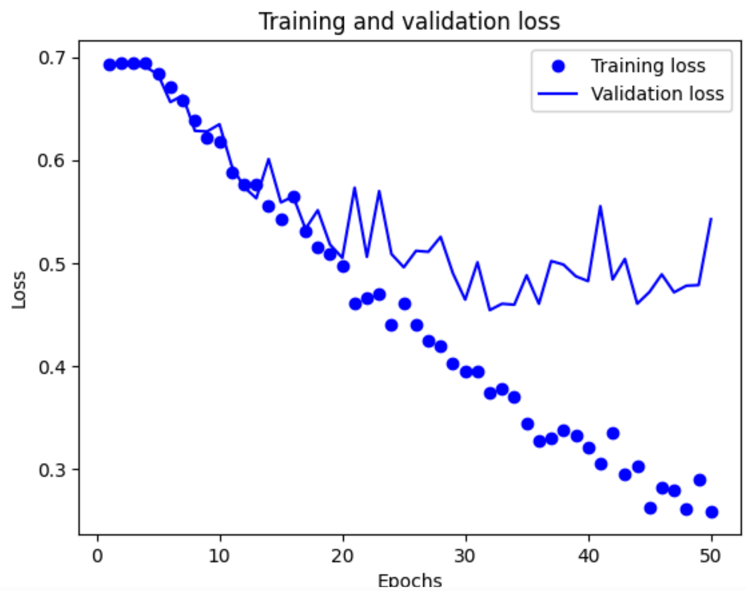
```
accuracy = hist.history["accuracy"]
val = hist.history["val_accuracy"]
loss = hist.history["loss"]
val_loss = hist.history["val_loss"]
epochs = range(1, len(accuracy) + 1)

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 49ms/step - accuracy: 0.7880 - loss: 0.4521
Test accuracy: 0.788
```

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)
```

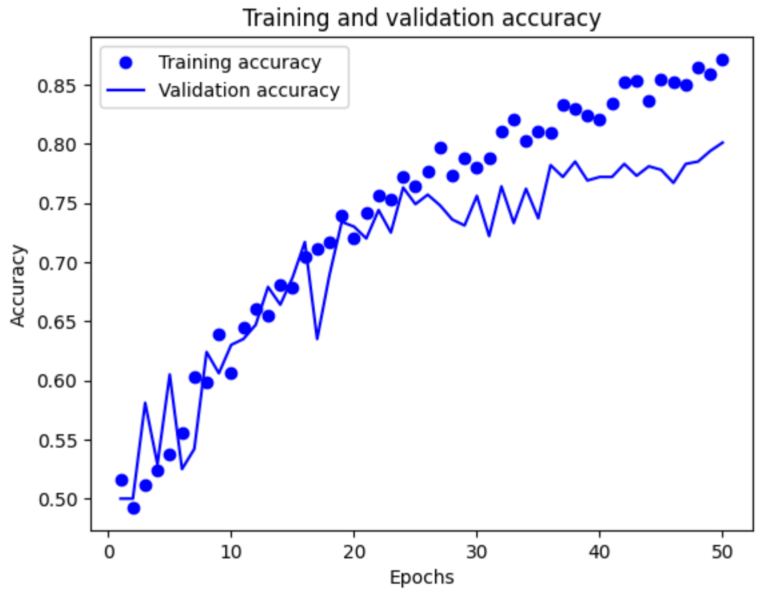
```
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.8302 - loss: 0.3779 - val_accuracy: 0.7720 - val_loss: 0.5337
Epoch 42/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.8537 - loss: 0.3495 - val_accuracy: 0.7830 - val_loss: 0.4943
Epoch 43/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 76ms/step - accuracy: 0.8537 - loss: 0.3413 - val_accuracy: 0.7730 - val_loss: 0.5388
Epoch 44/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.8443 - loss: 0.3463 - val_accuracy: 0.7810 - val_loss: 0.5185
Epoch 45/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 56ms/step - accuracy: 0.8504 - loss: 0.3377 - val_accuracy: 0.7780 - val_loss: 0.4776
Epoch 46/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 71ms/step - accuracy: 0.8501 - loss: 0.3495 - val_accuracy: 0.7670 - val_loss: 0.4898
Epoch 47/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 49ms/step - accuracy: 0.8535 - loss: 0.3129 - val_accuracy: 0.7830 - val_loss: 0.5044
Epoch 48/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 56ms/step - accuracy: 0.8566 - loss: 0.3196 - val_accuracy: 0.7850 - val_loss: 0.4779
Epoch 49/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.8545 - loss: 0.3218 - val_accuracy: 0.7940 - val_loss: 0.4837
Epoch 50/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 49ms/step - accuracy: 0.8670 - loss: 0.3162 - val_accuracy: 0.8010 - val_loss: 0.4924
```

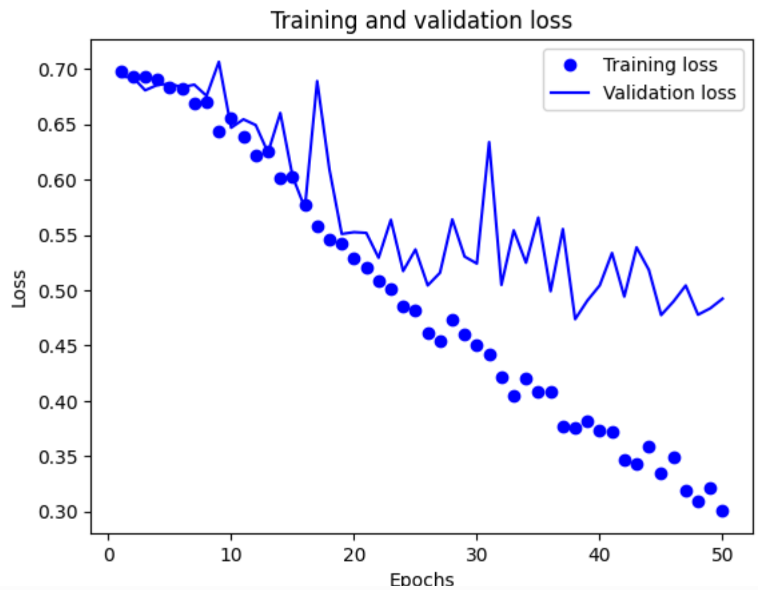
Curves of loss and accuracy during training

```
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 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.8076 - loss: 0.4412
Test accuracy: 0.807
```

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.
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
convoluted.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

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 167ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 168ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 163ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 208ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 171ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 168ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 168ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 163ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 163ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
```

```
train_features.shape
```

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

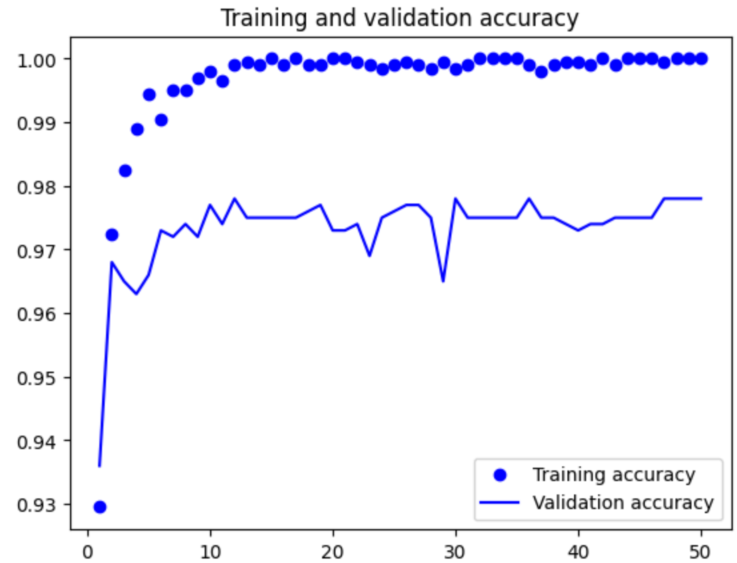
Model Fitting

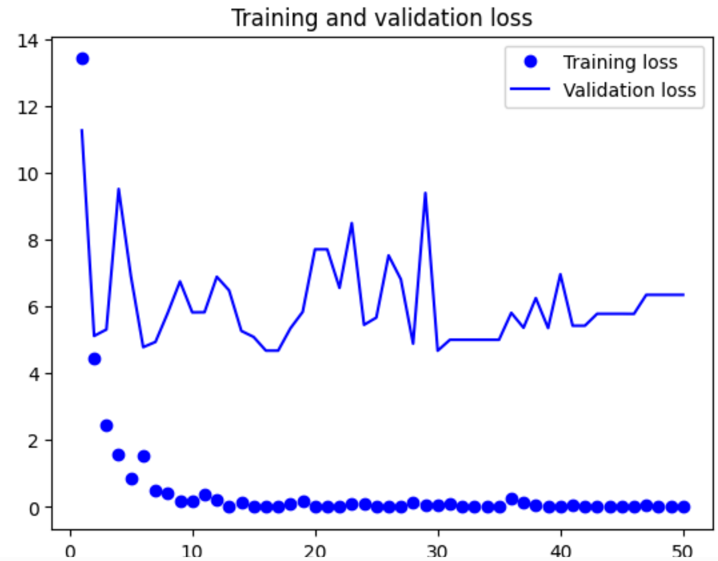
```
input = keras.Input(shape=(5, 5, 512))
data_2 = layers.Flatten()(input)
data_2 = layers.Dense(256)(data_2)
data_2 = layers.Dropout(0.5)(data_2)
out = layers.Dense(1, activation="sigmoid")(data_2)
model = keras.Model(input, out)
model.compile(loss="binary_crossentropy",
optimizer="rmsprop",
metrics=["accuracy"])
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 27/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9992 - loss: 0.0022 - val_accuracy: 0.9770 - val_loss: 6.8087
Epoch 28/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9982 - loss: 0.1298 - val_accuracy: 0.9750 - val_loss: 4.8855
Epoch 29/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0041 - val_accuracy: 0.9650 - val_loss: 9.3907
Epoch 30/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9969 - loss: 0.1083 - val_accuracy: 0.9780 - val_loss: 4.6738
Epoch 31/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9993 - loss: 0.0799 - val_accuracy: 0.9750 - val_loss: 5.0015
Epoch 32/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 3.2783e-32 - val_accuracy: 0.9750 - val_loss: 5.0015
Epoch 33/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 1.6062e-29 - val_accuracy: 0.9750 - val_loss: 5.0015
Epoch 34/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9750 - val_loss: 5.0015
Epoch 35/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 6.3657e-41 - val_accuracy: 0.9750 - val_loss: 5.0015
Epoch 36/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9994 - loss: 0.1360 - val_accuracy: 0.9780 - val_loss: 5.8061
Epoch 37/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9958 - loss: 0.2433 - val_accuracy: 0.9750 - val_loss: 5.3610
Epoch 38/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9998 - loss: 0.0085 - val_accuracy: 0.9750 - val_loss: 6.2478
Epoch 39/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9999 - loss: 0.0013 - val_accuracy: 0.9740 - val_loss: 5.3519
Epoch 40/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9995 - loss: 0.0122 - val_accuracy: 0.9730 - val_loss: 6.9603
Epoch 41/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9990 - loss: 0.0752 - val_accuracy: 0.9740 - val_loss: 5.4220
Epoch 42/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 1.9124e-21 - val_accuracy: 0.9740 - val_loss: 5.4220
Epoch 43/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9980 - loss: 0.0165 - val_accuracy: 0.9750 - val_loss: 5.7790
Epoch 44/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9750 - val_loss: 5.7790
Epoch 45/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 1.0000 - loss: 8.8263e-31 - val_accuracy: 0.9750 - val_loss: 5.7790
Epoch 46/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 5.2130e-10 - val_accuracy: 0.9750 - val_loss: 5.7757
Epoch 47/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9995 - loss: 0.0634 - val_accuracy: 0.9780 - val_loss: 6.3452
Epoch 48/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9780 - val_loss: 6.3452
Epoch 49/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 3.4131e-39 - val_accuracy: 0.9780 - val_loss: 6.3452
Epoch 50/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9780 - val_loss: 6.3452
```

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(
```

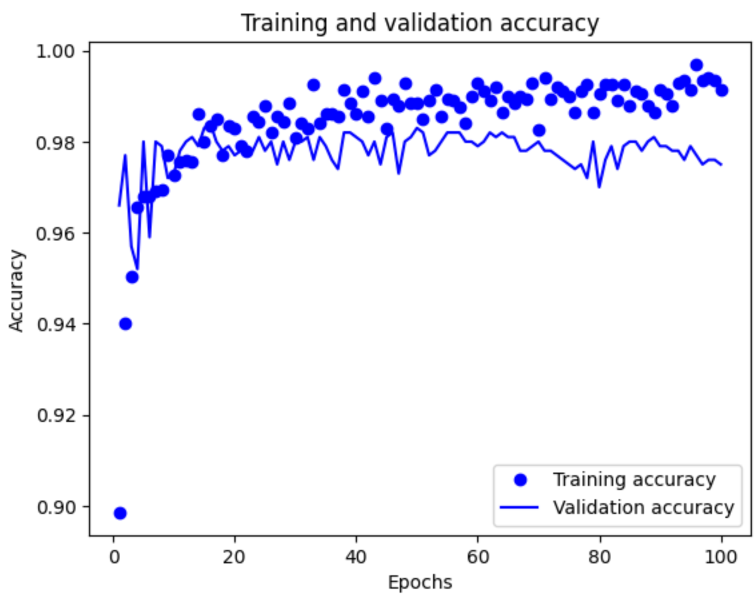
```
data_3=keras.layers.Lambda(
lambda x: keras.applications.vgg16.preprocess_input(x))(data_3)
data_3= convoluted(data_3)
data_3 = layers.Flatten()(data_3)
data_3 = layers.Dense(256)(data_3)
data_3= layers.Dropout(0.5)(data_3)
outputs = layers.Dense(1, activation="sigmoid")(data_3)
model = keras.Model(input, 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
)
```

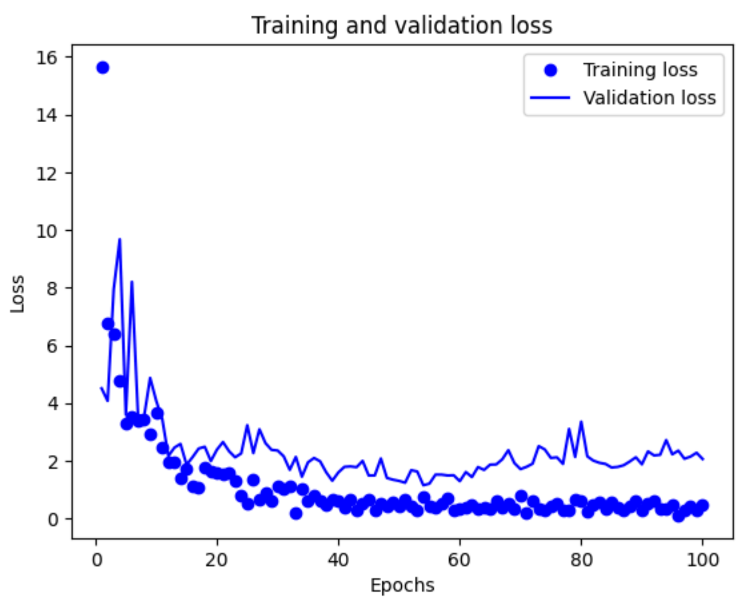
Epoch 74/100

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
```