jjoy6_Assignment_2

March 23, 2025

Imports and dir setup

[1]:
```
import os, shutil, pathlib, random
new_base_dir = pathlib.Path("/content/drive/MyDrive/Colab Notebooks/
 ↪cats_vs_dogs_small")

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.utils import image_dataset_from_directory
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

Sampling training data to generate different amounts.

[4]:
```
#This function will create the number of samples you want from both cat and dog
#Takes in a subset, a new subset name for organization, and a sample size
'''
def make_subset(subset_name, new_subset_name, sample_size):
    for category in ("cats", "dogs"):
        dir = new_base_dir / new_subset_name / category
        src_dir = new_base_dir / 'cats_vs_dogs_small' / subset_name / category
        os.makedirs(dir, exist_ok=True)
        fnames = [f for f in os.listdir(src_dir)]

        sampled_files = random.sample(fnames, sample_size)
        for sampled_files in sampled_files:
            shutil.copyfile(src=src_dir / sampled_files,
                            dst=dir / sampled_files)

make_subset("train", "train", 500)
make_subset("train", "train_2", 750)
#make_subset("validation", "validation", 250)
#make_subset("test", "test", 250)
make_subset("train", "train_3", 1000)
'''
```

Data preprocessing and batch size settings

[77]:
```
#change the names to change training/validation/test sets
train_dataset = image_dataset_from_directory(
    new_base_dir / "train_3",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)

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

batched_dataset = dataset.batch(32)
for i, element in enumerate(batched_dataset):
    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
```

```
Found 2000 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
(16,)
(16,)
(16,)
(32, 16)
(32, 16)
(32, 16)
(4, 4)
(4, 4)
(4, 4)
```

[78]:
```
for data_batch, labels_batch in train_dataset:
    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,)
```

Building different models without data augmentation

[ ]:
```
#Base Model from book
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

[ ]:
```
#Model and extra layer
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

[4]:
```
#Model with padding
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
```

```
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

[ ]:
```
#Model with dropout
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

[ ]:
```
#Model with batchnormalization
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
```

```
x = layers.Conv2D(filters=64, kernel_size=3, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

[ ]:
```
#Model with BatchNormalization and padding
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

[9]:
```
#Model with padding and dropout after each pooling and flatten
inputs = keras.Input(shape=(180, 180, 3))
```

```
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

[ ]:
```
#Model with padding and additional layer with 512 filters
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

[ ]:
```
#Model with padding, extra layer, batchNormalization
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

[ ]:
```
#Model with BatchNormalization padding and dropuut
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
```

```
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

[2]:
```
#padding dropout and layer
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Compiling Model

[11]:
```
#Compiling model
model.compile(loss="binary_crossentropy",
```

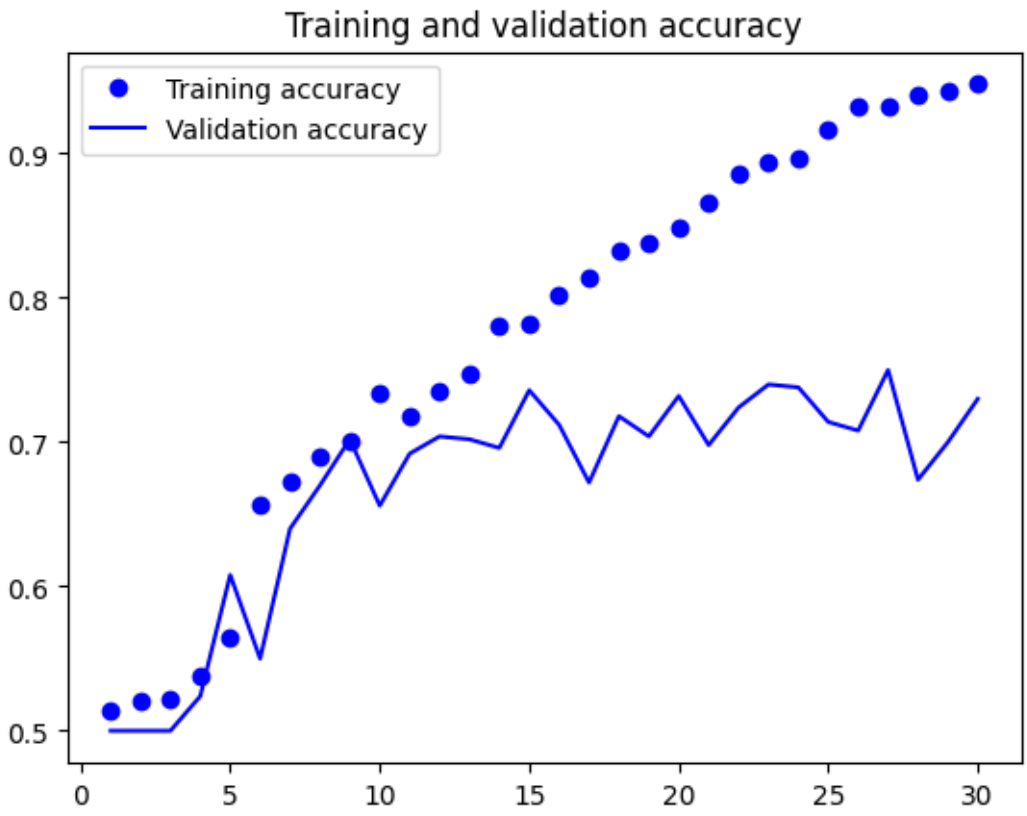
```
              optimizer="rmsprop",
              metrics=["accuracy"])
```

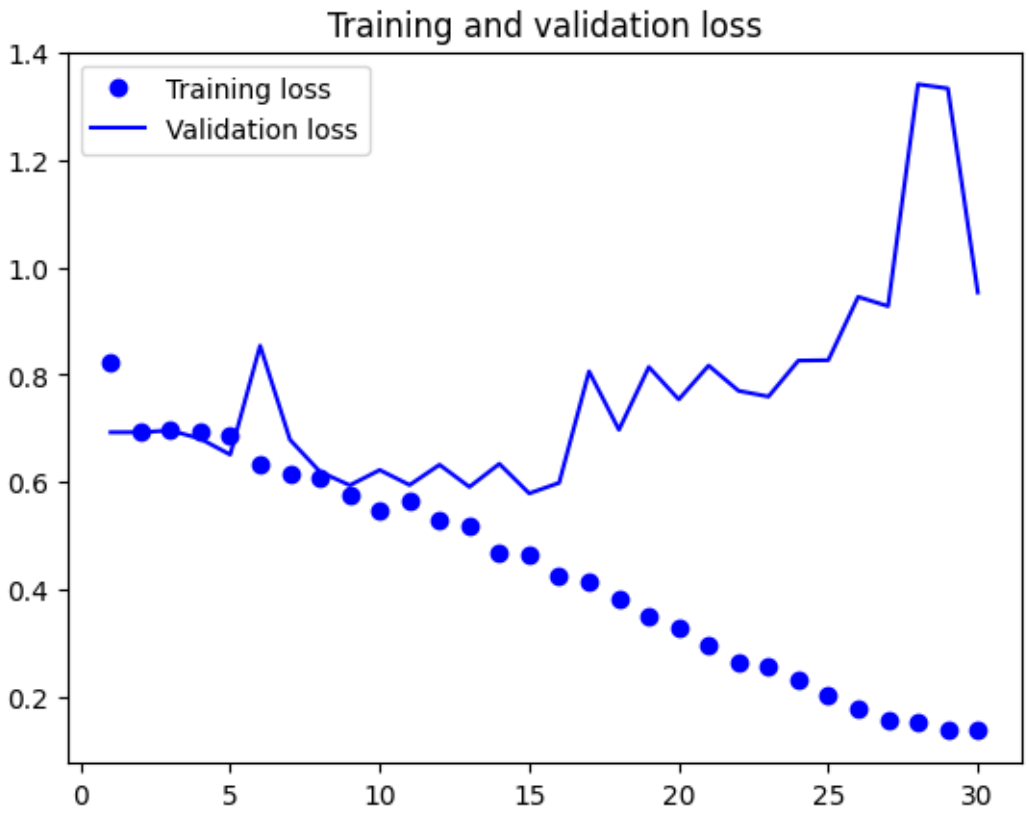
Fitting the model and using a callback

[ ]:
```
#callback to best epoch based on val_loss and fitting the model
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath= new_base_dir / "convnet_from_scratch_{epoch:02d}.keras",
        save_best_only=True,
        monitor="val_loss")

]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

Graphing the fitted model Training loss and acc Vs Validation loss and acc.

[13]:
```
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.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()
```


Training and validation accuracy
Training accuracy
Validation accuracy
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30



Using model on test data

[14]:
```
test_model = keras.models.load_model(new_base_dir / "convnet_from_scratch_15.
↪keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test Loss: {test_loss:.3f}")
```

```
16/16              1s 50ms/step -
accuracy: 0.7252 - loss: 0.5990
Test accuracy: 0.726
Test Loss: 0.597
```

Add Data Augmentation

[94]:
```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

Models using Data Augmentation and additional optimizations

[9]:
```
#Example improvements from book Data Augmentation and Dropout
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

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

[ ]:
```
#Adding Padding with one dropout
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

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

[17]:
```
#just padding
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

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

[60]:
```
#Adding Padding and droupot after each pooling and flatten
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
```

```
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

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

[ ]:
```
#Adding Padding everywhere
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

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

[10]:
```
#Extra layer
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
```

```
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)


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

[ ]:
```
#Multiple layers
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
```

```
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

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

[95]:
```
#Multiple layers and residuals
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
residuals = x
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
residuals = layers.Conv2D(32, 1, strides = 2)(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
residuals = layers.Conv2D(64, 1, strides = 2)(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
residuals = layers.Conv2D(128, 1, strides = 2)(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
```

```
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
residuals = layers.Conv2D(256, 1, strides = 2)(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same',␣
 ↪activation="relu")(x)
residuals = layers.Conv2D(256, 1)(residuals)
x = layers.add([x, residuals])
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

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

[ ]:
```
#Layers residuals batch
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
residuals = x
x = layers.Conv2D(filters=32, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
residuals = layers.Conv2D(32, 1, strides = 2, use_bias=False)(residuals)
residuals = layers.BatchNormalization()(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=64, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
```

```
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
residuals = layers.Conv2D(64, 1, strides = 2, use_bias=False)(residuals)
residuals = layers.BatchNormalization()(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=128, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
residuals = layers.Conv2D(128, 1, strides = 2, use_bias=False)(residuals)
residuals = layers.BatchNormalization()(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
residuals = layers.Conv2D(256, 1, strides = 2, use_bias=False)(residuals)
residuals = layers.BatchNormalization()(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
residuals = layers.Conv2D(256, 1, use_bias=False)(residuals)
residuals = layers.BatchNormalization()(residuals)
x = layers.add([x, residuals])
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

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

[21]:
```
#Layers residuals batch and dropout
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
residuals = x
x = layers.Conv2D(filters=32, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.2)(x)
residuals = layers.Conv2D(32, 1, strides = 2, use_bias=False)(residuals)
residuals = layers.BatchNormalization()(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=64, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.2)(x)
residuals = layers.Conv2D(64, 1, strides = 2, use_bias=False)(residuals)
residuals = layers.BatchNormalization()(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=128, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.2)(x)
residuals = layers.Conv2D(128, 1, strides = 2, use_bias=False)(residuals)
residuals = layers.BatchNormalization()(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
```

```
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.2)(x)
residuals = layers.Conv2D(256, 1, strides = 2, use_bias=False)(residuals)
residuals = layers.BatchNormalization()(residuals)
x = layers.add([x, residuals])
residuals = x
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
residuals = layers.Conv2D(256, 1, use_bias=False)(residuals)
residuals = layers.BatchNormalization()(residuals)
x = layers.add([x, residuals])
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

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

[25]:
```
#dropout
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
```

```
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

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

[22]:
```
#Dropout and Batch
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

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

Callback and model fit with increase to max number of epochs

[96]:
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="1convnet_from_scratch_with_augmentation_{epoch:02d}.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=100,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/100
63/63             12s 108ms/step -
accuracy: 0.4891 - loss: 0.7031 - val_accuracy: 0.4860 - val_loss: 0.6944
Epoch 2/100
63/63             5s 74ms/step -
accuracy: 0.5069 - loss: 0.6958 - val_accuracy: 0.5060 - val_loss: 0.6908
Epoch 3/100
63/63             5s 76ms/step -
accuracy: 0.5383 - loss: 0.6859 - val_accuracy: 0.5440 - val_loss: 0.6883
Epoch 4/100
63/63             5s 75ms/step -
accuracy: 0.5265 - loss: 0.6933 - val_accuracy: 0.5300 - val_loss: 0.6853
Epoch 5/100
63/63             5s 75ms/step -
accuracy: 0.5895 - loss: 0.6771 - val_accuracy: 0.5420 - val_loss: 0.6821
Epoch 6/100
63/63             5s 75ms/step -
accuracy: 0.5818 - loss: 0.6758 - val_accuracy: 0.5800 - val_loss: 0.6793
Epoch 7/100
63/63             5s 75ms/step -
accuracy: 0.5792 - loss: 0.6802 - val_accuracy: 0.5880 - val_loss: 0.6756
Epoch 8/100
63/63             5s 75ms/step -
accuracy: 0.5958 - loss: 0.6671 - val_accuracy: 0.5600 - val_loss: 0.6753
Epoch 9/100
63/63             5s 75ms/step -
accuracy: 0.5890 - loss: 0.6705 - val_accuracy: 0.5920 - val_loss: 0.6710
Epoch 10/100
63/63             5s 75ms/step -
accuracy: 0.6000 - loss: 0.6621 - val_accuracy: 0.6020 - val_loss: 0.6667
```

```
Epoch 11/100
63/63             5s 73ms/step -
accuracy: 0.6370 - loss: 0.6490 - val_accuracy: 0.5540 - val_loss: 0.6714
Epoch 12/100
63/63             5s 76ms/step -
accuracy: 0.6185 - loss: 0.6501 - val_accuracy: 0.6180 - val_loss: 0.6604
Epoch 13/100
63/63             5s 75ms/step -
accuracy: 0.6091 - loss: 0.6560 - val_accuracy: 0.6140 - val_loss: 0.6584
Epoch 14/100
63/63             5s 72ms/step -
accuracy: 0.6136 - loss: 0.6556 - val_accuracy: 0.5900 - val_loss: 0.6617
Epoch 15/100
63/63             5s 75ms/step -
accuracy: 0.6291 - loss: 0.6450 - val_accuracy: 0.6280 - val_loss: 0.6523
Epoch 16/100
63/63             5s 73ms/step -
accuracy: 0.6394 - loss: 0.6414 - val_accuracy: 0.5580 - val_loss: 0.6785
Epoch 17/100
63/63             5s 74ms/step -
accuracy: 0.6215 - loss: 0.6356 - val_accuracy: 0.6220 - val_loss: 0.6507
Epoch 18/100
63/63             5s 74ms/step -
accuracy: 0.6437 - loss: 0.6281 - val_accuracy: 0.6440 - val_loss: 0.6437
Epoch 19/100
63/63             5s 75ms/step -
accuracy: 0.6569 - loss: 0.6194 - val_accuracy: 0.6480 - val_loss: 0.6409
Epoch 20/100
63/63             5s 71ms/step -
accuracy: 0.6744 - loss: 0.6184 - val_accuracy: 0.5820 - val_loss: 0.6646
Epoch 21/100
63/63             5s 72ms/step -
accuracy: 0.6359 - loss: 0.6246 - val_accuracy: 0.6100 - val_loss: 0.6474
Epoch 22/100
63/63             5s 74ms/step -
accuracy: 0.6680 - loss: 0.6073 - val_accuracy: 0.6660 - val_loss: 0.6343
Epoch 23/100
63/63             5s 74ms/step -
accuracy: 0.6463 - loss: 0.6194 - val_accuracy: 0.6480 - val_loss: 0.6329
Epoch 24/100
63/63             5s 76ms/step -
accuracy: 0.6608 - loss: 0.6081 - val_accuracy: 0.6480 - val_loss: 0.6325
Epoch 25/100
63/63             5s 72ms/step -
accuracy: 0.6615 - loss: 0.6064 - val_accuracy: 0.6440 - val_loss: 0.6346
Epoch 26/100
63/63             5s 71ms/step -
accuracy: 0.6807 - loss: 0.6009 - val_accuracy: 0.6420 - val_loss: 0.6388
```

```
Epoch 27/100
63/63             5s 71ms/step -
accuracy: 0.6870 - loss: 0.5910 - val_accuracy: 0.6580 - val_loss: 0.6333
Epoch 28/100
63/63             5s 74ms/step -
accuracy: 0.6825 - loss: 0.5904 - val_accuracy: 0.6580 - val_loss: 0.6301
Epoch 29/100
63/63             5s 73ms/step -
accuracy: 0.6868 - loss: 0.5914 - val_accuracy: 0.6480 - val_loss: 0.6294
Epoch 30/100
63/63             5s 73ms/step -
accuracy: 0.7099 - loss: 0.5888 - val_accuracy: 0.6640 - val_loss: 0.6244
Epoch 31/100
63/63             5s 74ms/step -
accuracy: 0.7086 - loss: 0.5781 - val_accuracy: 0.6740 - val_loss: 0.6232
Epoch 32/100
63/63             4s 71ms/step -
accuracy: 0.6826 - loss: 0.5976 - val_accuracy: 0.6500 - val_loss: 0.6257
Epoch 33/100
63/63             5s 71ms/step -
accuracy: 0.6891 - loss: 0.5891 - val_accuracy: 0.6440 - val_loss: 0.6253
Epoch 34/100
63/63             5s 73ms/step -
accuracy: 0.6938 - loss: 0.5776 - val_accuracy: 0.6840 - val_loss: 0.6127
Epoch 35/100
63/63             4s 71ms/step -
accuracy: 0.7112 - loss: 0.5776 - val_accuracy: 0.5800 - val_loss: 0.6878
Epoch 36/100
63/63             5s 73ms/step -
accuracy: 0.6814 - loss: 0.5914 - val_accuracy: 0.6860 - val_loss: 0.6096
Epoch 37/100
63/63             4s 70ms/step -
accuracy: 0.7198 - loss: 0.5731 - val_accuracy: 0.6180 - val_loss: 0.6510
Epoch 38/100
63/63             4s 71ms/step -
accuracy: 0.6847 - loss: 0.5917 - val_accuracy: 0.6360 - val_loss: 0.6327
Epoch 39/100
63/63             4s 70ms/step -
accuracy: 0.7114 - loss: 0.5767 - val_accuracy: 0.6220 - val_loss: 0.6411
Epoch 40/100
63/63             5s 73ms/step -
accuracy: 0.7138 - loss: 0.5783 - val_accuracy: 0.6680 - val_loss: 0.6056
Epoch 41/100
63/63             4s 70ms/step -
accuracy: 0.7099 - loss: 0.5693 - val_accuracy: 0.6720 - val_loss: 0.6262
Epoch 42/100
63/63             4s 70ms/step -
accuracy: 0.7173 - loss: 0.5622 - val_accuracy: 0.6600 - val_loss: 0.6058
```

```
Epoch 43/100
63/63             5s 72ms/step -
accuracy: 0.7169 - loss: 0.5576 - val_accuracy: 0.6560 - val_loss: 0.6198
Epoch 44/100
63/63             5s 71ms/step -
accuracy: 0.7124 - loss: 0.5581 - val_accuracy: 0.6900 - val_loss: 0.6159
Epoch 45/100
63/63             5s 71ms/step -
accuracy: 0.7208 - loss: 0.5502 - val_accuracy: 0.6420 - val_loss: 0.6264
Epoch 46/100
63/63             5s 73ms/step -
accuracy: 0.7177 - loss: 0.5548 - val_accuracy: 0.6800 - val_loss: 0.5916
Epoch 47/100
63/63             4s 70ms/step -
accuracy: 0.7474 - loss: 0.5362 - val_accuracy: 0.6120 - val_loss: 0.6645
Epoch 48/100
63/63             5s 71ms/step -
accuracy: 0.7247 - loss: 0.5588 - val_accuracy: 0.6440 - val_loss: 0.6315
Epoch 49/100
63/63             5s 73ms/step -
accuracy: 0.7222 - loss: 0.5504 - val_accuracy: 0.6980 - val_loss: 0.5863
Epoch 50/100
63/63             5s 71ms/step -
accuracy: 0.7068 - loss: 0.5456 - val_accuracy: 0.6920 - val_loss: 0.6145
Epoch 51/100
63/63             5s 71ms/step -
accuracy: 0.7425 - loss: 0.5406 - val_accuracy: 0.6220 - val_loss: 0.6450
Epoch 52/100
63/63             5s 73ms/step -
accuracy: 0.7180 - loss: 0.5577 - val_accuracy: 0.6960 - val_loss: 0.5808
Epoch 53/100
63/63             4s 70ms/step -
accuracy: 0.7404 - loss: 0.5399 - val_accuracy: 0.6520 - val_loss: 0.6129
Epoch 54/100
63/63             5s 73ms/step -
accuracy: 0.7310 - loss: 0.5469 - val_accuracy: 0.6980 - val_loss: 0.5749
Epoch 55/100
63/63             4s 70ms/step -
accuracy: 0.7347 - loss: 0.5465 - val_accuracy: 0.6980 - val_loss: 0.5774
Epoch 56/100
63/63             5s 73ms/step -
accuracy: 0.7483 - loss: 0.5265 - val_accuracy: 0.6920 - val_loss: 0.5715
Epoch 57/100
63/63             4s 70ms/step -
accuracy: 0.7293 - loss: 0.5358 - val_accuracy: 0.6400 - val_loss: 0.6272
Epoch 58/100
63/63             4s 71ms/step -
accuracy: 0.7368 - loss: 0.5345 - val_accuracy: 0.6960 - val_loss: 0.5817
```

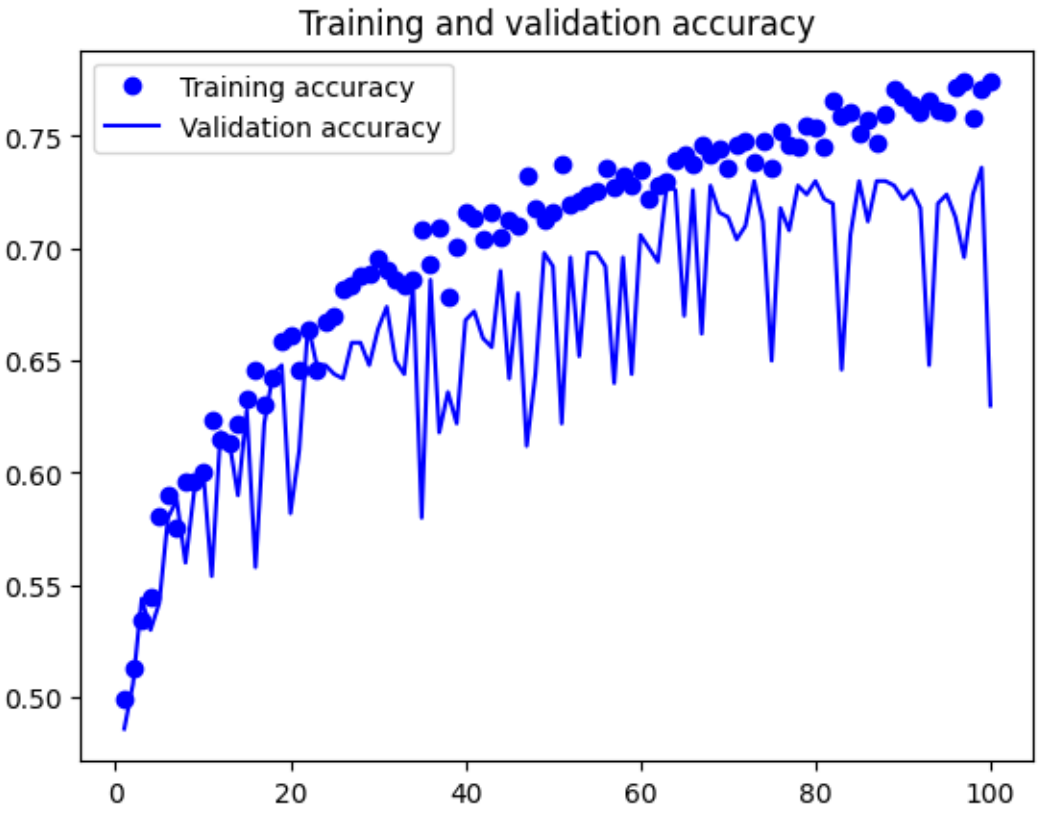
```
Epoch 59/100
63/63             5s 71ms/step -
accuracy: 0.7436 - loss: 0.5277 - val_accuracy: 0.6440 - val_loss: 0.6249
Epoch 60/100
63/63             4s 71ms/step -
accuracy: 0.7333 - loss: 0.5352 - val_accuracy: 0.7060 - val_loss: 0.5792
Epoch 61/100
63/63             4s 70ms/step -
accuracy: 0.7294 - loss: 0.5264 - val_accuracy: 0.7000 - val_loss: 0.5895
Epoch 62/100
63/63             5s 71ms/step -
accuracy: 0.7315 - loss: 0.5248 - val_accuracy: 0.6940 - val_loss: 0.5726
Epoch 63/100
63/63             5s 74ms/step -
accuracy: 0.7351 - loss: 0.5256 - val_accuracy: 0.7280 - val_loss: 0.5627
Epoch 64/100
63/63             4s 70ms/step -
accuracy: 0.7477 - loss: 0.5217 - val_accuracy: 0.7260 - val_loss: 0.5692
Epoch 65/100
63/63             4s 70ms/step -
accuracy: 0.7421 - loss: 0.5279 - val_accuracy: 0.6700 - val_loss: 0.5923
Epoch 66/100
63/63             5s 71ms/step -
accuracy: 0.7477 - loss: 0.5072 - val_accuracy: 0.7260 - val_loss: 0.5641
Epoch 67/100
63/63             5s 71ms/step -
accuracy: 0.7629 - loss: 0.5081 - val_accuracy: 0.6620 - val_loss: 0.5970
Epoch 68/100
63/63             5s 73ms/step -
accuracy: 0.7407 - loss: 0.5265 - val_accuracy: 0.7280 - val_loss: 0.5608
Epoch 69/100
63/63             4s 70ms/step -
accuracy: 0.7664 - loss: 0.5047 - val_accuracy: 0.7160 - val_loss: 0.5678
Epoch 70/100
63/63             5s 74ms/step -
accuracy: 0.7387 - loss: 0.5210 - val_accuracy: 0.7140 - val_loss: 0.5541
Epoch 71/100
63/63             5s 71ms/step -
accuracy: 0.7398 - loss: 0.5193 - val_accuracy: 0.7040 - val_loss: 0.5649
Epoch 72/100
63/63             5s 75ms/step -
accuracy: 0.7513 - loss: 0.5093 - val_accuracy: 0.7100 - val_loss: 0.5478
Epoch 73/100
63/63             4s 70ms/step -
accuracy: 0.7519 - loss: 0.5205 - val_accuracy: 0.7300 - val_loss: 0.5558
Epoch 74/100
63/63             4s 70ms/step -
accuracy: 0.7629 - loss: 0.4952 - val_accuracy: 0.7120 - val_loss: 0.5807
```

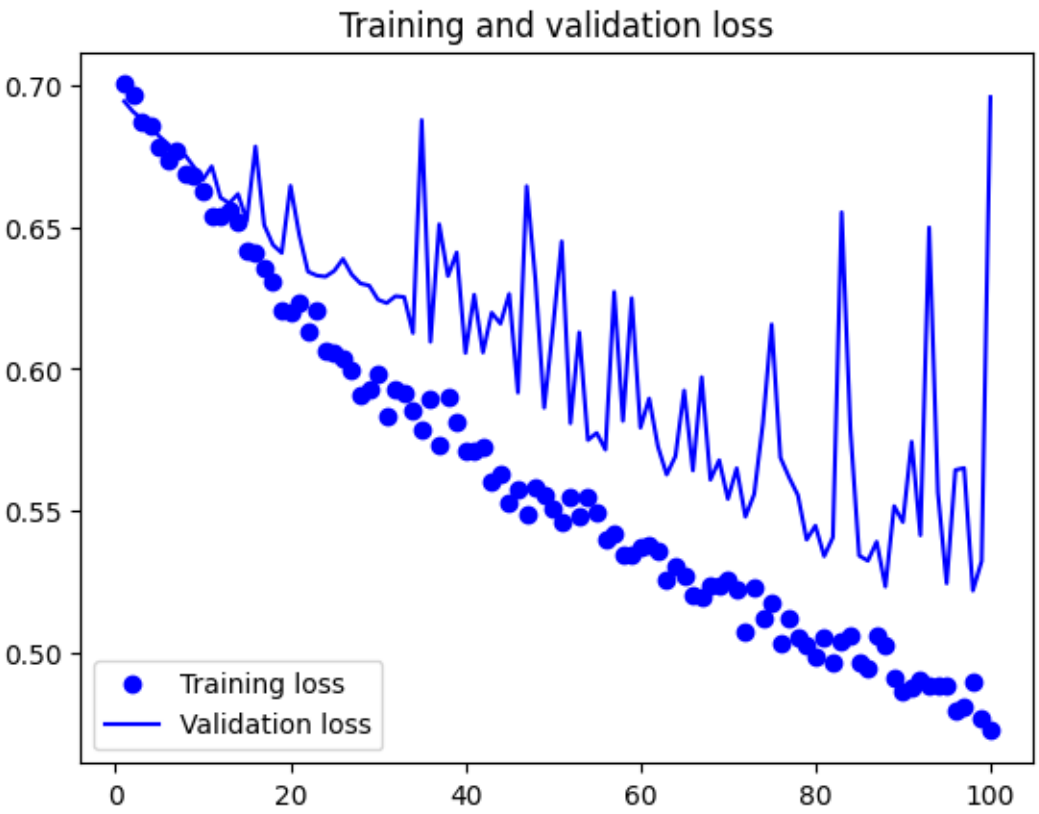
```
Epoch 75/100
63/63             4s 71ms/step -
accuracy: 0.7365 - loss: 0.5130 - val_accuracy: 0.6500 - val_loss: 0.6157
Epoch 76/100
63/63             4s 70ms/step -
accuracy: 0.7463 - loss: 0.5071 - val_accuracy: 0.7180 - val_loss: 0.5687
Epoch 77/100
63/63             5s 71ms/step -
accuracy: 0.7375 - loss: 0.5130 - val_accuracy: 0.7080 - val_loss: 0.5616
Epoch 78/100
63/63             5s 71ms/step -
accuracy: 0.7337 - loss: 0.5133 - val_accuracy: 0.7280 - val_loss: 0.5554
Epoch 79/100
63/63             5s 74ms/step -
accuracy: 0.7594 - loss: 0.4985 - val_accuracy: 0.7240 - val_loss: 0.5397
Epoch 80/100
63/63             5s 71ms/step -
accuracy: 0.7580 - loss: 0.4939 - val_accuracy: 0.7300 - val_loss: 0.5446
Epoch 81/100
63/63             5s 74ms/step -
accuracy: 0.7503 - loss: 0.4994 - val_accuracy: 0.7220 - val_loss: 0.5339
Epoch 82/100
63/63             4s 71ms/step -
accuracy: 0.7667 - loss: 0.4956 - val_accuracy: 0.7200 - val_loss: 0.5406
Epoch 83/100
63/63             4s 71ms/step -
accuracy: 0.7693 - loss: 0.4977 - val_accuracy: 0.6460 - val_loss: 0.6552
Epoch 84/100
63/63             5s 71ms/step -
accuracy: 0.7506 - loss: 0.5165 - val_accuracy: 0.7060 - val_loss: 0.5786
Epoch 85/100
63/63             4s 70ms/step -
accuracy: 0.7582 - loss: 0.4923 - val_accuracy: 0.7300 - val_loss: 0.5341
Epoch 86/100
63/63             5s 74ms/step -
accuracy: 0.7661 - loss: 0.4906 - val_accuracy: 0.7120 - val_loss: 0.5323
Epoch 87/100
63/63             5s 71ms/step -
accuracy: 0.7473 - loss: 0.5076 - val_accuracy: 0.7300 - val_loss: 0.5390
Epoch 88/100
63/63             5s 73ms/step -
accuracy: 0.7467 - loss: 0.5064 - val_accuracy: 0.7300 - val_loss: 0.5232
Epoch 89/100
63/63             5s 72ms/step -
accuracy: 0.7713 - loss: 0.4778 - val_accuracy: 0.7280 - val_loss: 0.5516
Epoch 90/100
63/63             5s 71ms/step -
accuracy: 0.7676 - loss: 0.4832 - val_accuracy: 0.7220 - val_loss: 0.5459
```

```
Epoch 91/100
63/63             4s 70ms/step -
accuracy: 0.7491 - loss: 0.5048 - val_accuracy: 0.7260 - val_loss: 0.5743
Epoch 92/100
63/63             5s 71ms/step -
accuracy: 0.7588 - loss: 0.4819 - val_accuracy: 0.7180 - val_loss: 0.5413
Epoch 93/100
63/63             5s 72ms/step -
accuracy: 0.7672 - loss: 0.4918 - val_accuracy: 0.6480 - val_loss: 0.6499
Epoch 94/100
63/63             5s 71ms/step -
accuracy: 0.7594 - loss: 0.4996 - val_accuracy: 0.7200 - val_loss: 0.5564
Epoch 95/100
63/63             5s 71ms/step -
accuracy: 0.7590 - loss: 0.4977 - val_accuracy: 0.7240 - val_loss: 0.5243
Epoch 96/100
63/63             4s 71ms/step -
accuracy: 0.7789 - loss: 0.4753 - val_accuracy: 0.7140 - val_loss: 0.5643
Epoch 97/100
63/63             5s 81ms/step -
accuracy: 0.7696 - loss: 0.4781 - val_accuracy: 0.6960 - val_loss: 0.5650
Epoch 98/100
63/63             5s 74ms/step -
accuracy: 0.7568 - loss: 0.4893 - val_accuracy: 0.7240 - val_loss: 0.5217
Epoch 99/100
63/63             4s 70ms/step -
accuracy: 0.7817 - loss: 0.4693 - val_accuracy: 0.7360 - val_loss: 0.5322
Epoch 100/100
63/63             4s 70ms/step -
accuracy: 0.7822 - loss: 0.4688 - val_accuracy: 0.6300 - val_loss: 0.6959
```

Graphing of Training Loss and Acc Vs. Validation Loss and Acc.

[97]:
```
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.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()
```


Training and validation accuracy
Training accuracy
Validation accuracy
0.75
0.70
0.65
0.60
0.55
0.50
0
20
40
60
80
100



Using model and seeing results when used on test data.

[98]:
```
test_model = keras.models.load_model(
    "1convnet_from_scratch_with_augmentation_98.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test Loss: {test_loss:.3f}")
```

```
16/16            2s 41ms/step -
accuracy: 0.7148 - loss: 0.5525
Test accuracy: 0.726
Test Loss: 0.548
```

Pretrained Models

[63]:
```
#Using Pretrained Model
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

[64]:
```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer_20 (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


Total params: 14,714,688 (56.13 MB)

Trainable params: 14,714,688 (56.13 MB)

Non-trainable params: 0 (0.00 B)
```

Extracting features from VGG16 and seperating labels to work correctly in VGG16

[65]:
```
def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = conv_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

train_features, train_labels =  get_features_and_labels(train_dataset)
val_features, val_labels =  get_features_and_labels(validation_dataset)
test_features, test_labels =  get_features_and_labels(test_dataset)
```

```
2/2             1s 63ms/step
2/2             0s 64ms/step
2/2             0s 64ms/step
2/2             0s 63ms/step
```

```
2/2             0s 64ms/step
2/2             0s 64ms/step
2/2             0s 64ms/step
2/2             0s 63ms/step
2/2             0s 64ms/step
2/2             0s 68ms/step
2/2             0s 64ms/step
2/2             0s 63ms/step
2/2             0s 63ms/step
2/2             0s 63ms/step
2/2             0s 64ms/step
2/2             0s 65ms/step
2/2             0s 62ms/step
2/2             0s 64ms/step
2/2             0s 64ms/step
2/2             0s 65ms/step
2/2             0s 64ms/step
2/2             0s 64ms/step
2/2             0s 63ms/step
2/2             0s 63ms/step
2/2             0s 63ms/step
2/2             0s 63ms/step
2/2             0s 64ms/step
2/2             0s 63ms/step
2/2             0s 63ms/step
2/2             0s 64ms/step
2/2             0s 63ms/step
1/1             0s 467ms/step
2/2             0s 72ms/step
2/2             0s 64ms/step
2/2             0s 66ms/step
2/2             0s 65ms/step
2/2             0s 63ms/step
2/2             0s 63ms/step
2/2             0s 64ms/step
2/2             0s 287ms/step
2/2             0s 67ms/step
2/2             0s 66ms/step
2/2             0s 65ms/step
2/2             0s 66ms/step
2/2             0s 64ms/step
2/2             0s 63ms/step
2/2             0s 64ms/step
2/2             0s 46ms/step
```

[79]:
```
train_features.shape
```

[79]:
```
(2000, 5, 5, 512)
```

Using the pretrained model

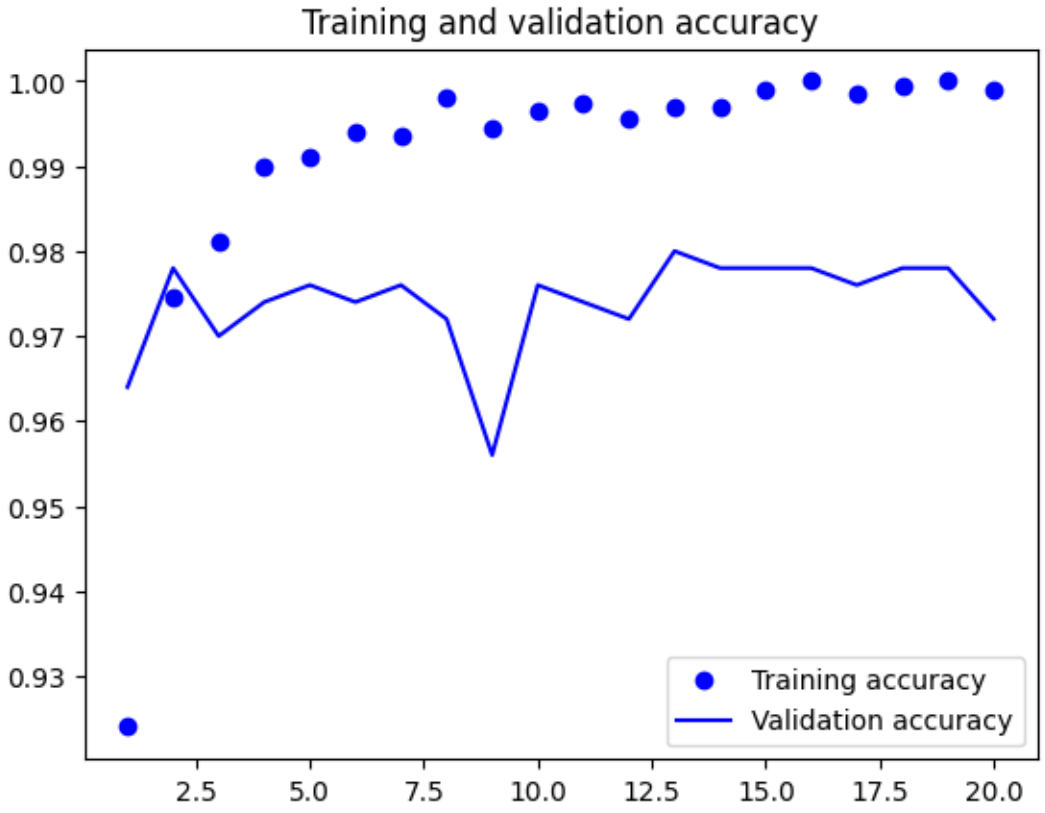
[67]:
```
inputs = keras.Input(shape=(5,5,512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)

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

callbacks = [
    keras.callbacks.ModelCheckpoint(
      filepath="642000_feature_extraction_{epoch:02d}.keras",
      save_best_only=True,
      monitor="val_loss")
]
history = model.fit(
    train_features, train_labels,
    epochs=20,
    validation_data=(val_features, val_labels),
    callbacks=callbacks)
```

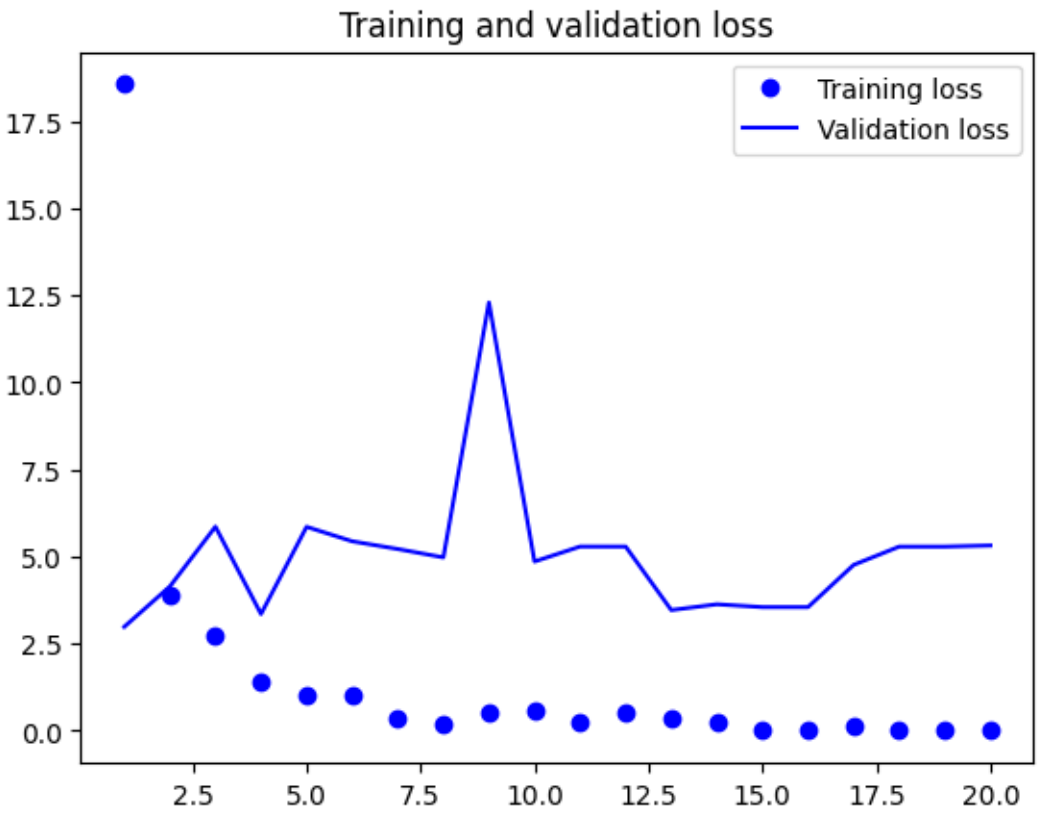
```
Epoch 1/20
63/63             3s 24ms/step -
accuracy: 0.8613 - loss: 47.2641 - val_accuracy: 0.9640 - val_loss: 2.9842
Epoch 2/20
63/63             0s 4ms/step -
accuracy: 0.9790 - loss: 3.3239 - val_accuracy: 0.9780 - val_loss: 4.1421
Epoch 3/20
63/63             0s 4ms/step -
accuracy: 0.9841 - loss: 2.4549 - val_accuracy: 0.9700 - val_loss: 5.8592
Epoch 4/20
63/63             0s 4ms/step -
accuracy: 0.9922 - loss: 1.0603 - val_accuracy: 0.9740 - val_loss: 3.3525
Epoch 5/20
63/63             0s 4ms/step -
accuracy: 0.9961 - loss: 0.3429 - val_accuracy: 0.9760 - val_loss: 5.8556
Epoch 6/20
63/63             0s 4ms/step -
accuracy: 0.9934 - loss: 1.4208 - val_accuracy: 0.9740 - val_loss: 5.4403
Epoch 7/20
63/63             0s 4ms/step -
accuracy: 0.9940 - loss: 0.3522 - val_accuracy: 0.9760 - val_loss: 5.2180
```

```
Epoch 8/20
63/63             0s 4ms/step -
accuracy: 0.9995 - loss: 0.0674 - val_accuracy: 0.9720 - val_loss: 4.9779
Epoch 9/20
63/63             0s 4ms/step -
accuracy: 0.9934 - loss: 0.7205 - val_accuracy: 0.9560 - val_loss: 12.2967
Epoch 10/20
63/63             0s 4ms/step -
accuracy: 0.9950 - loss: 1.0411 - val_accuracy: 0.9760 - val_loss: 4.8591
Epoch 11/20
63/63             0s 4ms/step -
accuracy: 0.9965 - loss: 0.3608 - val_accuracy: 0.9740 - val_loss: 5.2890
Epoch 12/20
63/63             0s 4ms/step -
accuracy: 0.9945 - loss: 0.7502 - val_accuracy: 0.9720 - val_loss: 5.2879
Epoch 13/20
63/63             0s 4ms/step -
accuracy: 0.9973 - loss: 0.2699 - val_accuracy: 0.9800 - val_loss: 3.4650
Epoch 14/20
63/63             0s 4ms/step -
accuracy: 0.9973 - loss: 0.1773 - val_accuracy: 0.9780 - val_loss: 3.6315
Epoch 15/20
63/63             0s 4ms/step -
accuracy: 0.9989 - loss: 0.0188 - val_accuracy: 0.9780 - val_loss: 3.5504
Epoch 16/20
63/63             0s 4ms/step -
accuracy: 1.0000 - loss: 1.3603e-08 - val_accuracy: 0.9780 - val_loss: 3.5501
Epoch 17/20
63/63             0s 4ms/step -
accuracy: 0.9990 - loss: 0.0698 - val_accuracy: 0.9760 - val_loss: 4.7625
Epoch 18/20
63/63             0s 4ms/step -
accuracy: 0.9999 - loss: 0.0058 - val_accuracy: 0.9780 - val_loss: 5.2871
Epoch 19/20
63/63             0s 4ms/step -
accuracy: 1.0000 - loss: 1.3203e-11 - val_accuracy: 0.9780 - val_loss: 5.2871
Epoch 20/20
63/63             0s 4ms/step -
accuracy: 0.9992 - loss: 0.0070 - val_accuracy: 0.9720 - val_loss: 5.3197
```

[68]:
```python
import matplotlib.pyplot as plt
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, "bo", label="Training accuracy")
```

```
plt.plot(epochs, val_acc, "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()
```





Testing pretrained model on test data.

[69]:
```
#Test the pretrained model as is.
model = keras.models.load_model("642000_feature_extraction_01.keras")
test_loss, test_acc = model.evaluate(test_features, test_labels)
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16            1s 12ms/step -
accuracy: 0.9567 - loss: 6.7265
Test accuracy: 0.962
Test loss: 6.221
```

Using VGG16 as a layer in our own model but not retrainng.

[87]:
```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False)
conv_base.trainable = False
```

Adding Data Augmentation

[88]:
```
#Adding data augmentation
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy", optimizer=keras.optimizers.
 ↪RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])
```

Callback and model fit

[89]:
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="1322000_feature_extraction_with_data_augmentation{epoch:02d}.
 ↪keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/50
63/63            8s 93ms/step -
accuracy: 0.6020 - loss: 9.0093 - val_accuracy: 0.9020 - val_loss: 1.5065
Epoch 2/50
63/63            6s 88ms/step -
accuracy: 0.8207 - loss: 3.6409 - val_accuracy: 0.9320 - val_loss: 1.0950
Epoch 3/50
63/63            6s 88ms/step -
accuracy: 0.8952 - loss: 1.8983 - val_accuracy: 0.9560 - val_loss: 0.9847
Epoch 4/50
63/63            6s 88ms/step -
```

```
accuracy: 0.9030 - loss: 1.8976 - val_accuracy: 0.9480 - val_loss: 0.9647
Epoch 5/50
63/63             6s 94ms/step -
accuracy: 0.9327 - loss: 1.4569 - val_accuracy: 0.9620 - val_loss: 0.8687
Epoch 6/50
63/63             6s 88ms/step -
accuracy: 0.9237 - loss: 1.4165 - val_accuracy: 0.9640 - val_loss: 0.8568
Epoch 7/50
63/63             5s 82ms/step -
accuracy: 0.9359 - loss: 1.0916 - val_accuracy: 0.9620 - val_loss: 0.8897
Epoch 8/50
63/63             6s 93ms/step -
accuracy: 0.9291 - loss: 1.3659 - val_accuracy: 0.9620 - val_loss: 0.8247
Epoch 9/50
63/63             6s 87ms/step -
accuracy: 0.9450 - loss: 0.9952 - val_accuracy: 0.9660 - val_loss: 0.8091
Epoch 10/50
63/63             6s 87ms/step -
accuracy: 0.9483 - loss: 0.7916 - val_accuracy: 0.9660 - val_loss: 0.7857
Epoch 11/50
63/63             6s 87ms/step -
accuracy: 0.9472 - loss: 0.8814 - val_accuracy: 0.9660 - val_loss: 0.7783
Epoch 12/50
63/63             6s 93ms/step -
accuracy: 0.9554 - loss: 0.7005 - val_accuracy: 0.9700 - val_loss: 0.7476
Epoch 13/50
63/63             5s 81ms/step -
accuracy: 0.9536 - loss: 0.7707 - val_accuracy: 0.9680 - val_loss: 0.7866
Epoch 14/50
63/63             5s 81ms/step -
accuracy: 0.9488 - loss: 0.8348 - val_accuracy: 0.9680 - val_loss: 0.7704
Epoch 15/50
63/63             6s 87ms/step -
accuracy: 0.9530 - loss: 0.7041 - val_accuracy: 0.9700 - val_loss: 0.7189
Epoch 16/50
63/63             6s 87ms/step -
accuracy: 0.9502 - loss: 0.9056 - val_accuracy: 0.9720 - val_loss: 0.6875
Epoch 17/50
63/63             6s 87ms/step -
accuracy: 0.9513 - loss: 0.8811 - val_accuracy: 0.9720 - val_loss: 0.7348
Epoch 18/50
63/63             5s 81ms/step -
accuracy: 0.9475 - loss: 0.8171 - val_accuracy: 0.9720 - val_loss: 0.7058
Epoch 19/50
63/63             5s 81ms/step -
accuracy: 0.9657 - loss: 0.6144 - val_accuracy: 0.9700 - val_loss: 0.7813
Epoch 20/50
63/63             6s 87ms/step -
```

```
accuracy: 0.9564 - loss: 0.7739 - val_accuracy: 0.9720 - val_loss: 0.6735
Epoch 21/50
63/63            5s 81ms/step -
accuracy: 0.9697 - loss: 0.4292 - val_accuracy: 0.9760 - val_loss: 0.6749
Epoch 22/50
63/63            5s 81ms/step -
accuracy: 0.9574 - loss: 0.6962 - val_accuracy: 0.9760 - val_loss: 0.7001
Epoch 23/50
63/63            5s 82ms/step -
accuracy: 0.9645 - loss: 0.6015 - val_accuracy: 0.9760 - val_loss: 0.7093
Epoch 24/50
63/63            5s 82ms/step -
accuracy: 0.9578 - loss: 0.7173 - val_accuracy: 0.9740 - val_loss: 0.7107
Epoch 25/50
63/63            5s 82ms/step -
accuracy: 0.9768 - loss: 0.3442 - val_accuracy: 0.9740 - val_loss: 0.7044
Epoch 26/50
63/63            5s 81ms/step -
accuracy: 0.9779 - loss: 0.3919 - val_accuracy: 0.9760 - val_loss: 0.6949
Epoch 27/50
63/63            5s 82ms/step -
accuracy: 0.9692 - loss: 0.3911 - val_accuracy: 0.9760 - val_loss: 0.7295
Epoch 28/50
63/63            5s 81ms/step -
accuracy: 0.9742 - loss: 0.4319 - val_accuracy: 0.9780 - val_loss: 0.8050
Epoch 29/50
63/63            5s 81ms/step -
accuracy: 0.9574 - loss: 0.6925 - val_accuracy: 0.9740 - val_loss: 0.7025
Epoch 30/50
63/63            5s 81ms/step -
accuracy: 0.9734 - loss: 0.3712 - val_accuracy: 0.9780 - val_loss: 0.6922
Epoch 31/50
63/63            5s 81ms/step -
accuracy: 0.9734 - loss: 0.3608 - val_accuracy: 0.9780 - val_loss: 0.7122
Epoch 32/50
63/63            5s 81ms/step -
accuracy: 0.9680 - loss: 0.3769 - val_accuracy: 0.9780 - val_loss: 0.7451
Epoch 33/50
63/63            5s 81ms/step -
accuracy: 0.9684 - loss: 0.3194 - val_accuracy: 0.9780 - val_loss: 0.7115
Epoch 34/50
63/63            5s 81ms/step -
accuracy: 0.9733 - loss: 0.4552 - val_accuracy: 0.9780 - val_loss: 0.6843
Epoch 35/50
63/63            5s 87ms/step -
accuracy: 0.9755 - loss: 0.3574 - val_accuracy: 0.9780 - val_loss: 0.6684
Epoch 36/50
63/63            5s 86ms/step -
```

```
accuracy: 0.9720 - loss: 0.5070 - val_accuracy: 0.9760 - val_loss: 0.6611
Epoch 37/50
63/63             5s 87ms/step -
accuracy: 0.9689 - loss: 0.5244 - val_accuracy: 0.9760 - val_loss: 0.6494
Epoch 38/50
63/63             5s 81ms/step -
accuracy: 0.9762 - loss: 0.2679 - val_accuracy: 0.9780 - val_loss: 0.7134
Epoch 39/50
63/63             5s 85ms/step -
accuracy: 0.9711 - loss: 0.3882 - val_accuracy: 0.9760 - val_loss: 0.7264
Epoch 40/50
63/63             5s 81ms/step -
accuracy: 0.9789 - loss: 0.3118 - val_accuracy: 0.9740 - val_loss: 0.7639
Epoch 41/50
63/63             5s 81ms/step -
accuracy: 0.9704 - loss: 0.4261 - val_accuracy: 0.9740 - val_loss: 0.7664
Epoch 42/50
63/63             5s 81ms/step -
accuracy: 0.9744 - loss: 0.3053 - val_accuracy: 0.9740 - val_loss: 0.7137
Epoch 43/50
63/63             5s 81ms/step -
accuracy: 0.9760 - loss: 0.3431 - val_accuracy: 0.9760 - val_loss: 0.7212
Epoch 44/50
63/63             5s 81ms/step -
accuracy: 0.9766 - loss: 0.3696 - val_accuracy: 0.9780 - val_loss: 0.6681
Epoch 45/50
63/63             5s 81ms/step -
accuracy: 0.9755 - loss: 0.3679 - val_accuracy: 0.9760 - val_loss: 0.6736
Epoch 46/50
63/63             5s 81ms/step -
accuracy: 0.9742 - loss: 0.4545 - val_accuracy: 0.9760 - val_loss: 0.6524
Epoch 47/50
63/63             5s 87ms/step -
accuracy: 0.9697 - loss: 0.4184 - val_accuracy: 0.9740 - val_loss: 0.6266
Epoch 48/50
63/63             5s 81ms/step -
accuracy: 0.9721 - loss: 0.3573 - val_accuracy: 0.9800 - val_loss: 0.6876
Epoch 49/50
63/63             5s 81ms/step -
accuracy: 0.9753 - loss: 0.3615 - val_accuracy: 0.9780 - val_loss: 0.6740
Epoch 50/50
63/63             5s 81ms/step -
accuracy: 0.9773 - loss: 0.3294 - val_accuracy: 0.9760 - val_loss: 0.6652
```

Testing model on testdata.

[90]:
```
test_model = keras.models.load_model(
    "1322000_feature_extraction_with_data_augmentation47.keras")
```

```
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16             2s 66ms/step -
accuracy: 0.9870 - loss: 0.4069
Test accuracy: 0.978
Test loss: 0.671
```

Training of unfrozen layers

[91]:
```
#Allow for layers to be fine tuned from model above
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```

Compile, callback, and model fit

[92]:
```
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-10),
              metrics=["accuracy"])

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="21322000_fine_tuning_{epoch:02d}.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/30
63/63             9s 108ms/step -
accuracy: 0.9823 - loss: 0.2050 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 2/30
63/63             6s 92ms/step -
accuracy: 0.9700 - loss: 0.3247 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 3/30
63/63             6s 92ms/step -
accuracy: 0.9742 - loss: 0.3228 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 4/30
63/63             6s 92ms/step -
accuracy: 0.9803 - loss: 0.2751 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 5/30
63/63             6s 91ms/step -
```

```
accuracy: 0.9699 - loss: 0.4445 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 6/30
63/63             6s 91ms/step -
accuracy: 0.9736 - loss: 0.2974 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 7/30
63/63             6s 102ms/step -
accuracy: 0.9815 - loss: 0.3328 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 8/30
63/63             6s 91ms/step -
accuracy: 0.9828 - loss: 0.2922 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 9/30
63/63             6s 90ms/step -
accuracy: 0.9862 - loss: 0.1600 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 10/30
63/63             6s 90ms/step -
accuracy: 0.9730 - loss: 0.3811 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 11/30
63/63             6s 90ms/step -
accuracy: 0.9763 - loss: 0.2965 - val_accuracy: 0.9760 - val_loss: 0.6654
Epoch 12/30
63/63             6s 90ms/step -
accuracy: 0.9710 - loss: 0.5529 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 13/30
63/63             6s 92ms/step -
accuracy: 0.9811 - loss: 0.2981 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 14/30
63/63             6s 90ms/step -
accuracy: 0.9794 - loss: 0.3642 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 15/30
63/63             6s 90ms/step -
accuracy: 0.9858 - loss: 0.2131 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 16/30
63/63             6s 90ms/step -
accuracy: 0.9867 - loss: 0.1641 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 17/30
63/63             6s 90ms/step -
accuracy: 0.9715 - loss: 0.3963 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 18/30
63/63             6s 91ms/step -
accuracy: 0.9802 - loss: 0.2570 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 19/30
63/63             6s 90ms/step -
accuracy: 0.9837 - loss: 0.1861 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 20/30
63/63             6s 91ms/step -
accuracy: 0.9752 - loss: 0.2942 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 21/30
63/63             6s 91ms/step -
```

```
accuracy: 0.9791 - loss: 0.3428 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 22/30
63/63             6s 91ms/step -
accuracy: 0.9799 - loss: 0.2012 - val_accuracy: 0.9760 - val_loss: 0.6652
Epoch 23/30
63/63             6s 91ms/step -
accuracy: 0.9739 - loss: 0.3511 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 24/30
63/63             6s 91ms/step -
accuracy: 0.9765 - loss: 0.3838 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 25/30
63/63             6s 91ms/step -
accuracy: 0.9786 - loss: 0.3681 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 26/30
63/63             6s 91ms/step -
accuracy: 0.9797 - loss: 0.3278 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 27/30
63/63             6s 91ms/step -
accuracy: 0.9772 - loss: 0.3178 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 28/30
63/63             6s 90ms/step -
accuracy: 0.9757 - loss: 0.3319 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 29/30
63/63             6s 90ms/step -
accuracy: 0.9819 - loss: 0.1955 - val_accuracy: 0.9760 - val_loss: 0.6653
Epoch 30/30
63/63             6s 90ms/step -
accuracy: 0.9766 - loss: 0.3354 - val_accuracy: 0.9760 - val_loss: 0.6654
```

Testing model with test data

[93]:
```
model = keras.models.load_model("21322000_fine_tuning_07.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16             2s 62ms/step -
accuracy: 0.9777 - loss: 0.6267
Test accuracy: 0.976
Test loss: 0.622
```