Convolutional Neural Networks and Computer Vision with TensorFlow

Get the data

```
import zipfile

# Download zip file of pizza_steak images
!wget https://storage.googleapis.com/ztm_tf_course/food_vision/pizza_steak.zip

# Unzip the downloaded file
zip_ref = zipfile.ZipFile("pizza_steak.zip", "r")
zip_ref.extractall()
zip_ref.close()
```

```
--2023-05-11 03:50:16--  https://storage.googleapis.com/ztm_tf_course/food_vision/pizza_steak.zip
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.128.128, 74.125.143.128, 173.194.69.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.128.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 109540975 (104M) [application/zip]
Saving to: ‘pizza_steak.zip’

pizza_steak.zip     100%[===================>] 104.47M  29.4MB/s    in 3.6s

2023-05-11 03:50:20 (29.4 MB/s) - ‘pizza_steak.zip’ saved [109540975/109540975]
```

Inspect the data

```
!ls pizza_steak
```

```
test  train
```

Double-click (or enter) to edit

```
!ls pizza_steak/train/
```

```
pizza  steak
```

```
!ls pizza_steak/train/steak/
```

```
1000205.jpg  1647351.jpg  2238681.jpg  2824680.jpg  3375959.jpg  417368.jpg
100135.jpg   1650002.jpg  2238802.jpg  2825100.jpg  3381560.jpg  4176.jpg
101312.jpg   165639.jpg   2254705.jpg  2826987.jpg  3382936.jpg  42125.jpg
1021458.jpg  1658186.jpg  225990.jpg   2832499.jpg  3386119.jpg  421476.jpg
1032846.jpg  1658443.jpg  2260231.jpg  2832960.jpg  3388717.jpg  421561.jpg
10380.jpg    165964.jpg   2268692.jpg  285045.jpg   3389138.jpg  438871.jpg
1049459.jpg  167069.jpg   2271133.jpg  285147.jpg   3393547.jpg  43924.jpg
1053665.jpg  1675632.jpg  227576.jpg   2855315.jpg  3393688.jpg  440188.jpg
1068516.jpg  1678108.jpg  2283057.jpg  2856066.jpg  3396589.jpg  442757.jpg
1068975.jpg  168006.jpg   2286639.jpg  2859933.jpg  339891.jpg   443210.jpg
1081258.jpg  1682496.jpg  2287136.jpg  286219.jpg   3417789.jpg  444064.jpg
1090122.jpg  1684438.jpg  2291292.jpg  2862562.jpg  3425047.jpg  444709.jpg
1093966.jpg  168775.jpg   229323.jpg   2865730.jpg  3434983.jpg  447557.jpg
1098844.jpg  1697339.jpg  2300534.jpg  2878151.jpg  3435358.jpg  461187.jpg
1100074.jpg  1710569.jpg  2300845.jpg  2880035.jpg  3438319.jpg  461689.jpg
1105280.jpg  1714605.jpg  231296.jpg   2881783.jpg  3444407.jpg  465494.jpg
1117936.jpg  1724387.jpg  2315295.jpg  2884233.jpg  345734.jpg   468384.jpg
1126126.jpg  1724717.jpg  2323132.jpg  2890573.jpg  3460673.jpg  477486.jpg
114601.jpg   172936.jpg   2324994.jpg  2893832.jpg  3465327.jpg  482022.jpg
1147047.jpg  1736543.jpg  2327701.jpg  2893892.jpg  3466159.jpg  482465.jpg
1147883.jpg  1736968.jpg  2331076.jpg  2907177.jpg  3469024.jpg  483788.jpg
1155665.jpg  1746626.jpg  233964.jpg   290850.jpg   3470083.jpg  493029.jpg
1163977.jpg  1752330.jpg  2344227.jpg  2909031.jpg  3476564.jpg  503589.jpg
1190233.jpg  1761285.jpg  234626.jpg   2910418.jpg  3478318.jpg  510757.jpg
1208405.jpg  176508.jpg   234704.jpg   2912290.jpg  3488748.jpg  513129.jpg
1209120.jpg  1772039.jpg  2357281.jpg  2916448.jpg  3492328.jpg  513842.jpg
1212161.jpg  1777107.jpg  2361812.jpg  2916967.jpg  3518960.jpg  523535.jpg
1213988.jpg  1787505.jpg  2365287.jpg  2927833.jpg  3522209.jpg  525041.jpg
1219039.jpg  179293.jpg   2374582.jpg  2928643.jpg  3524429.jpg  534560.jpg
1225762.jpg  1816235.jpg  239025.jpg   2929179.jpg  3528458.jpg  534633.jpg
1230968.jpg  1822407.jpg  2390628.jpg  2936477.jpg  3531805.jpg  536535.jpg
```

```
1236155.jpg  1823263.jpg  2392910.jpg  2938012.jpg  3536023.jpg  541410.jpg
1241193.jpg  1826066.jpg  2394465.jpg  2938151.jpg  3538682.jpg  543691.jpg
1248337.jpg  1828502.jpg  2395127.jpg  2939678.jpg  3540750.jpg  560503.jpg
1257104.jpg  1828969.jpg  2396291.jpg  2940544.jpg  354329.jpg   561972.jpg
126345.jpg   1829045.jpg  2400975.jpg  2940621.jpg  3547166.jpg  56240.jpg
1264050.jpg  1829088.jpg  2403776.jpg  2949079.jpg  3553911.jpg  56409.jpg
1264154.jpg  1836332.jpg  2403907.jpg  295491.jpg   3556871.jpg  564530.jpg
1264858.jpg  1839025.jpg  240435.jpg   296268.jpg   355715.jpg   568972.jpg
127029.jpg   1839481.jpg  2404695.jpg  2964732.jpg  356234.jpg   576725.jpg
1289900.jpg  183995.jpg   2404884.jpg  2965021.jpg  3571963.jpg  588739.jpg
1290362.jpg  184110.jpg   2407770.jpg  2966859.jpg  3576078.jpg  590142.jpg
1295457.jpg  184226.jpg   2412263.jpg  2977966.jpg  3577618.jpg  60633.jpg
1312841.jpg  1846706.jpg  2425062.jpg  2979061.jpg  3577732.jpg  60655.jpg
1313316.jpg  1849364.jpg  2425389.jpg  2983260.jpg  3578934.jpg  606820.jpg
1324791.jpg  1849463.jpg  2435316.jpg  2984311.jpg  358042.jpg   612551.jpg
1327567.jpg  1849542.jpg  2437268.jpg  2988960.jpg  358045.jpg   614975.jpg
1327667.jpg  1853564.jpg  2437843.jpg  2989882.jpg  3591821.jpg  616809.jpg
1333055.jpg  1869467.jpg  2440131.jpg  2995169.jpg  359330.jpg   628628.jpg
1334054.jpg  1870942.jpg  2443168.jpg  2996324.jpg  3601483.jpg  632427.jpg
1335556.jpg  187303.jpg   2446660.jpg  3000131.jpg  3606642.jpg  636594.jpg
1337814.jpg  187521.jpg   2455944.jpg  3002350.jpg  3609394.jpg  637374.jpg
1340977.jpg  1888450.jpg  2458401.jpg  3007772.jpg  361067.jpg   640539.jpg
1343209.jpg  1889336.jpg  2487306.jpg  3008192.jpg  3613455.jpg  644777.jpg
134369.jpg   1907039.jpg  248841.jpg   3009617.jpg  3621464.jpg  644867.jpg
1344105.jpg  1925230.jpg  2489716.jpg  3011642.jpg  3621562.jpg  658189.jpg
134598.jpg   1927984.jpg  2490489.jpg  3020591.jpg  3621565.jpg  660900.jpg
```

```
import os

for dirpath, dirnames, filenames in os.walk("pizza_steak"):
  print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

```
There are 2 directories and 0 images in 'pizza_steak'.
There are 2 directories and 0 images in 'pizza_steak/test'.
There are 0 directories and 250 images in 'pizza_steak/test/steak'.
There are 0 directories and 250 images in 'pizza_steak/test/pizza'.
There are 2 directories and 0 images in 'pizza_steak/train'.
There are 0 directories and 750 images in 'pizza_steak/train/steak'.
There are 0 directories and 750 images in 'pizza_steak/train/pizza'.
```

```
# Another way to find out how many images are in a file
num_steak_images_train = len(os.listdir("pizza_steak/train/steak"))

num_steak_images_train
```

```
750
```

```
# Get the class names
import pathlib
import numpy as np
data_dir = pathlib.Path("pizza_steak/train/")
class_names = np.array(sorted([item.name for item in data_dir.glob('*')])) # created a list of class_names from the subdirectories
print(class_names)
```

```
['pizza' 'steak']
```

```
# View an image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random

def view_random_image(target_dir, target_class):
  # Setup target directory (we'll view images from here)
  target_folder = target_dir+target_class

  # Get a random image path
  random_image = random.sample(os.listdir(target_folder), 1)

  # Read in the image and plot it using matplotlib
  img = mpimg.imread(target_folder + "/" + random_image[0])
  plt.imshow(img)
  plt.title(target_class)
  plt.axis("off");

  print(f"Image shape: {img.shape}") # show the shape of the image

  return img
```

```
# View a random image from the training dataset
img = view_random_image(target_dir="pizza_steak/train/",
                        target_class="steak")
```

```
Image shape: (512, 512, 3)
```


```
# View the img (actually just a big array/tensor)
img
```

```
array([[[ 80,  32,  18],
        [ 77,  29,  15],
        [ 75,  27,  13],
        ...,
        [189, 163, 130],
        [196, 170, 137],
        [192, 166, 133]],

       [[ 79,  31,  17],
        [ 76,  28,  14],
        [ 74,  26,  12],
        ...,
        [155, 129,  96],
        [177, 151, 118],
        [192, 166, 133]],

       [[ 78,  30,  16],
        [ 76,  28,  14],
        [ 74,  26,  12],
        ...,
        [135, 109,  76],
        [149, 123,  90],
        [170, 144, 111]],

       ...,

       [[ 52,  18,  17],
        [ 63,  29,  28],
        [ 53,  19,  17],
        ...,
        [253, 237, 203],
        [253, 237, 203],
        [253, 237, 203]],

       [[ 53,  22,  20],
        [ 63,  32,  30],
        [ 56,  22,  21],
        ...,
        [253, 237, 203],
        [253, 237, 203],
        [252, 236, 202]],

       [[ 41,  10,   8],
        [ 62,  31,  29],
        [ 53,  19,  18],
        ...,
        [253, 237, 203],
        [252, 236, 202],
        [252, 236, 202]]], dtype=uint8)
```

```
# View the image shape
img.shape # returns (width, height, colour channels)
```

```
(512, 512, 3)
```

```
# Get all the pixel values between 0 & 1
img/255.
```

```
array([[[0.31372549, 0.1254902 , 0.07058824],
        [0.30196078, 0.11372549, 0.05882353],
        [0.29411765, 0.10588235, 0.05098039],
        ...,
        [0.74117647, 0.63921569, 0.50980392],
        [0.76862745, 0.66666667, 0.5372549 ],
        [0.75294118, 0.65098039, 0.52156863]],

       [[0.30980392, 0.12156863, 0.06666667],
        [0.29803922, 0.10980392, 0.05490196],
        [0.29019608, 0.10196078, 0.04705882],
        ...,
        [0.60784314, 0.50588235, 0.37647059],
        [0.69411765, 0.59215686, 0.4627451 ],
        [0.75294118, 0.65098039, 0.52156863]],

       [[0.30588235, 0.11764706, 0.0627451 ],
        [0.29803922, 0.10980392, 0.05490196],
        [0.29019608, 0.10196078, 0.04705882],
        ...,
        [0.52941176, 0.42745098, 0.29803922],
        [0.58431373, 0.48235294, 0.35294118],
        [0.66666667, 0.56470588, 0.43529412]],

       ...,

       [[0.20392157, 0.07058824, 0.06666667],
        [0.24705882, 0.11372549, 0.10980392],
        [0.20784314, 0.0745098 , 0.06666667],
        ...,
        [0.99215686, 0.92941176, 0.79607843],
        [0.99215686, 0.92941176, 0.79607843],
        [0.99215686, 0.92941176, 0.79607843]],

       [[0.20784314, 0.08627451, 0.07843137],
        [0.24705882, 0.1254902 , 0.11764706],
        [0.21960784, 0.08627451, 0.08235294],
        ...,
        [0.99215686, 0.92941176, 0.79607843],
        [0.99215686, 0.92941176, 0.79607843],
        [0.98823529, 0.9254902 , 0.79215686]],

       [[0.16078431, 0.03921569, 0.03137255],
        [0.24313725, 0.12156863, 0.11372549],
        [0.20784314, 0.0745098 , 0.07058824],
        ...,
        [0.99215686, 0.92941176, 0.79607843],
        [0.98823529, 0.9254902 , 0.79215686],
        [0.98823529, 0.9254902 , 0.79215686]]])
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Set the seed
tf.random.set_seed(42)

# Preprocess data
train_datagen = ImageDataGenerator(rescale=1./255)
valid_datagen = ImageDataGenerator(rescale=1./255)

# Setup the train and test directories
train_dir = "pizza_steak/train/"
test_dir = "pizza_steak/test/"

# Import data from directories and turn it into batches
train_data = train_datagen.flow_from_directory(train_dir,
                                               batch_size=32, # number of images to process at a time
                                               target_size=(224, 224), # convert all images to be 224 x 224
                                               class_mode="binary", # type of problem
                                               seed=42)
```

```
valid_data = valid_datagen.flow_from_directory(test_dir,
                                               batch_size=32,
                                               target_size=(224, 224),
                                               class_mode="binary",
                                               seed=42)

model_1 = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(filters=10,
                         kernel_size=3, # can also be (3, 3)
                         activation="relu",
                         input_shape=(224, 224, 3)), # first layer specifies input shape (height, width, colour channels)
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.MaxPool2D(pool_size=2, # pool_size can also be (2, 2)
                            padding="valid"), # padding can also be 'same'
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.MaxPool2D(2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(1, activation="sigmoid") # binary activation output
])

# Compile the model
model_1.compile(loss="binary_crossentropy",
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# Fit the model
history_1 = model_1.fit(train_data,
                        epochs=5,
                        steps_per_epoch=len(train_data),
                        validation_data=valid_data,
                        validation_steps=len(valid_data))
```

```
Found 1500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Epoch 1/5
47/47 [==============================] - 23s 209ms/step - loss: 0.5981 - accuracy: 0.6773 - val_loss: 0.4158 - val_accuracy: 0.8060
Epoch 2/5
47/47 [==============================] - 9s 194ms/step - loss: 0.4584 - accuracy: 0.7967 - val_loss: 0.3675 - val_accuracy: 0.8500
Epoch 3/5
47/47 [==============================] - 9s 192ms/step - loss: 0.4656 - accuracy: 0.7893 - val_loss: 0.4424 - val_accuracy: 0.7940
Epoch 4/5
47/47 [==============================] - 9s 192ms/step - loss: 0.4098 - accuracy: 0.8380 - val_loss: 0.3485 - val_accuracy: 0.8780
Epoch 5/5
47/47 [==============================] - 9s 191ms/step - loss: 0.3708 - accuracy: 0.8453 - val_loss: 0.3162 - val_accuracy: 0.8820
```

```
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 10)      280

 conv2d_1 (Conv2D)           (None, 220, 220, 10)      910

 max_pooling2d (MaxPooling2D  (None, 110, 110, 10)     0
 )

 conv2d_2 (Conv2D)           (None, 108, 108, 10)      910

 conv2d_3 (Conv2D)           (None, 106, 106, 10)      910

 max_pooling2d_1 (MaxPooling  (None, 53, 53, 10)       0
 2D)

 flatten (Flatten)           (None, 28090)             0

 dense (Dense)               (None, 1)                 28091

=================================================================
Total params: 31,101
Trainable params: 31,101
Non-trainable params: 0
_________________________________________________________________
```

```
# Set random seed
tf.random.set_seed(42)
```

```
# Create a model to replicate the TensorFlow model
model_2 = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(224, 224, 3)), # dense layers expect a 1-dimensional vector as input
  tf.keras.layers.Dense(4, activation='relu'),
  tf.keras.layers.Dense(4, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model_2.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# Fit the model
history_2 = model_2.fit(train_data, # use same training data created above
                        epochs=5,
                        steps_per_epoch=len(train_data),
                        validation_data=valid_data, # use same validation data created above
                        validation_steps=len(valid_data))
```

```
Epoch 1/5
47/47 [==============================] - 11s 197ms/step - loss: 1.7892 - accuracy: 0.4920 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 2/5
47/47 [==============================] - 9s 191ms/step - loss: 0.6932 - accuracy: 0.5000 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 3/5
47/47 [==============================] - 9s 189ms/step - loss: 0.6932 - accuracy: 0.5000 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 4/5
47/47 [==============================] - 9s 199ms/step - loss: 0.6932 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5/5
47/47 [==============================] - 9s 196ms/step - loss: 0.6932 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
```

```
# second model's architecture
model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 150528)            0

 dense_1 (Dense)             (None, 4)                 602116

 dense_2 (Dense)             (None, 4)                 20

 dense_3 (Dense)             (None, 1)                 5

=================================================================
Total params: 602,141
Trainable params: 602,141
Non-trainable params: 0
_________________________________________________________________
```

```
# Set random seed
tf.random.set_seed(42)

# Create a model similar to model_1 but add an extra layer and increase the number of hidden units in each layer
model_3 = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(224, 224, 3)), # dense layers expect a 1-dimensional vector as input
  tf.keras.layers.Dense(100, activation='relu'), # increase number of neurons from 4 to 100 (for each layer)
  tf.keras.layers.Dense(100, activation='relu'),
  tf.keras.layers.Dense(100, activation='relu'), # add an extra layer
  tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model_3.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# Fit the model
history_3 = model_3.fit(train_data,
                        epochs=5,
                        steps_per_epoch=len(train_data),
                        validation_data=valid_data,
                        validation_steps=len(valid_data))
```

```
Epoch 1/5
47/47 [==============================] - 11s 201ms/step - loss: 2.8247 - accuracy: 0.6413 - val_loss: 0.6214 - val_accuracy: 0.7720
```

```
Epoch 2/5
47/47 [==============================] - 9s 195ms/step - loss: 0.6881 - accuracy: 0.7300 - val_loss: 0.5762 - val_accuracy: 0.7140
Epoch 3/5
47/47 [==============================] - 9s 196ms/step - loss: 1.2195 - accuracy: 0.6860 - val_loss: 0.6234 - val_accuracy: 0.7320
Epoch 4/5
47/47 [==============================] - 9s 194ms/step - loss: 0.5172 - accuracy: 0.7893 - val_loss: 0.5061 - val_accuracy: 0.7600
Epoch 5/5
47/47 [==============================] - 9s 192ms/step - loss: 0.5041 - accuracy: 0.7833 - val_loss: 0.4380 - val_accuracy: 0.8000
```

Woah! Looks like our model is learning again. It got ~70% accuracy on the training set and ~70% accuracy on the validation set.

How does the architecute look?

```
# Check out model_3 architecture
model_3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 150528)            0

 dense_4 (Dense)             (None, 100)               15052900

 dense_5 (Dense)             (None, 100)               10100

 dense_6 (Dense)             (None, 100)               10100

 dense_7 (Dense)             (None, 1)                 101

=================================================================
Total params: 15,073,201
Trainable params: 15,073,201
Non-trainable params: 0
_________________________________________________________________
```

```
# import zipfile

# # Download zip file of pizza_steak images
# !wget https://storage.googleapis.com/ztm_tf_course/food_vision/pizza_steak.zip

# # Unzip the downloaded file
# zip_ref = zipfile.ZipFile("pizza_steak.zip", "r")
# zip_ref.extractall()
# zip_ref.close()
```

```
plt.figure()
plt.subplot(1, 2, 1)
steak_img = view_random_image("pizza_steak/train/", "steak")
plt.subplot(1, 2, 2)
pizza_img = view_random_image("pizza_steak/train/", "pizza")
```

```
Image shape: (512, 512, 3)
Image shape: (512, 512, 3)
```


```
# Define training and test directory paths
train_dir = "pizza_steak/train/"
test_dir = "pizza_steak/test/"
```

```
# Create train and test data generators and rescale the data
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
train_datagen = ImageDataGenerator(rescale=1/255.)
test_datagen = ImageDataGenerator(rescale=1/255.)


# Turn it into batches
train_data = train_datagen.flow_from_directory(directory=train_dir,
                                               target_size=(224, 224),
                                               class_mode='binary',
                                               batch_size=32)

test_data = test_datagen.flow_from_directory(directory=test_dir,
                                             target_size=(224, 224),
                                             class_mode='binary',
                                             batch_size=32)
```

```
Found 1500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
```

```
# Get a sample of the training data batch
images, labels = train_data.next() # get the 'next' batch of images/labels
len(images), len(labels)
```

```
(32, 32)
```

```
# Get the first two images
images[:2], images[0].shape
```

```
(array([[[[0.47058827, 0.40784317, 0.34509805],
          [0.4784314 , 0.427451  , 0.3647059 ],
          [0.48627454, 0.43529415, 0.37254903],
          ...,
          [0.8313726 , 0.70980394, 0.48627454],
          [0.8431373 , 0.73333335, 0.5372549 ],
          [0.87843144, 0.7725491 , 0.5882353 ]],

         [[0.50980395, 0.427451  , 0.36078432],
          [0.5058824 , 0.42352945, 0.35686275],
          [0.5137255 , 0.4431373 , 0.3647059 ],
          ...,
          [0.82745105, 0.7058824 , 0.48235297],
          [0.82745105, 0.70980394, 0.5058824 ],
          [0.8431373 , 0.73333335, 0.5372549 ]],

         [[0.5254902 , 0.427451  , 0.34901962],
          [0.5372549 , 0.43921572, 0.36078432],
          [0.5372549 , 0.45098042, 0.36078432],
          ...,
          [0.82745105, 0.7019608 , 0.4784314 ],
          [0.82745105, 0.7058824 , 0.49411768],
          [0.8352942 , 0.7176471 , 0.5137255 ]],

         ...,

         [[0.77647066, 0.5647059 , 0.2901961 ],
          [0.7803922 , 0.53333336, 0.22352943],
          [0.79215693, 0.5176471 , 0.18039216],
          ...,
          [0.30588236, 0.2784314 , 0.24705884],
          [0.24705884, 0.23137257, 0.19607845],
          [0.2784314 , 0.27450982, 0.25490198]],

         [[0.7843138 , 0.57254905, 0.29803923],
          [0.79215693, 0.54509807, 0.24313727],
          [0.8000001 , 0.5254902 , 0.18823531],
          ...,
          [0.2627451 , 0.23529413, 0.20392159],
          [0.24313727, 0.227451  , 0.19215688],
          [0.26666668, 0.2627451 , 0.24313727]],

         [[0.7960785 , 0.59607846, 0.3372549 ],
          [0.7960785 , 0.5647059 , 0.26666668],
          [0.81568635, 0.54901963, 0.22352943],
          ...,
          [0.23529413, 0.19607845, 0.16078432],
          [0.3019608 , 0.26666668, 0.24705884],
          [0.26666668, 0.2509804 , 0.24705884]]],


        [[[0.38823533, 0.4666667 , 0.36078432],
          [0.3921569 , 0.46274513, 0.36078432],
          [0.38431376, 0.454902  , 0.36078432],
```

```
               ...,
               [0.5294118 , 0.627451  , 0.54509807],
               [0.5294118 , 0.627451  , 0.54509807],
               [0.5411765 , 0.6392157 , 0.5568628 ]],
```

```
# View the first batch of labels
labels
```

```
array([1., 1., 0., 1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0., 1.,
       1., 0., 1., 0., 1., 1., 1., 0., 0., 0., 0., 0., 1., 0., 1.],
      dtype=float32)
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Activation
from tensorflow.keras import Sequential
```

```
# Create the model
model_4 = Sequential([
  Conv2D(filters=10,
         kernel_size=3,
         strides=1,
         padding='valid',
         activation='relu',
         input_shape=(224, 224, 3)), # input layer
  Conv2D(10, 3, activation='relu'),
  Conv2D(10, 3, activation='relu'),
  Flatten(),
  Dense(1, activation='sigmoid') # output layer
])
```

```
model_4.compile(loss='binary_crossentropy',
                optimizer=Adam(),
                metrics=['accuracy'])
```

```
len(train_data), len(test_data)
```

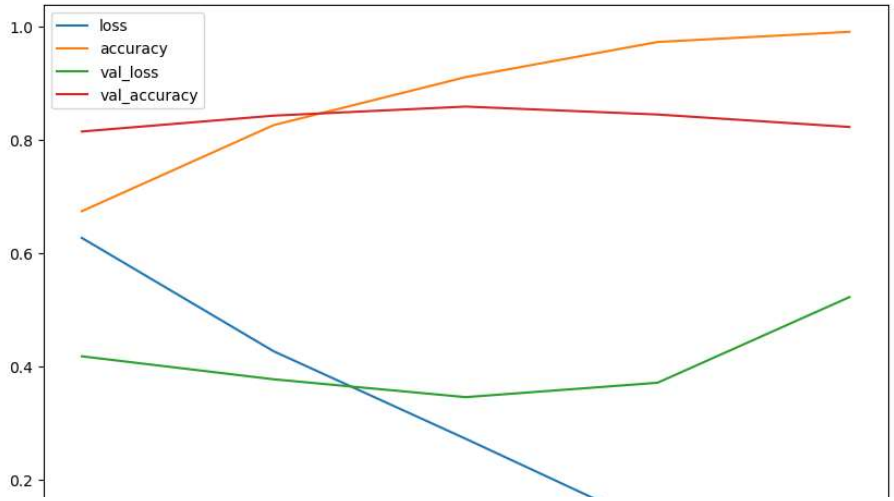
```
(47, 16)
```

```
# Fit the model
history_4 = model_4.fit(train_data,
                        epochs=5,
                        steps_per_epoch=len(train_data),
                        validation_data=test_data,
                        validation_steps=len(test_data))
```

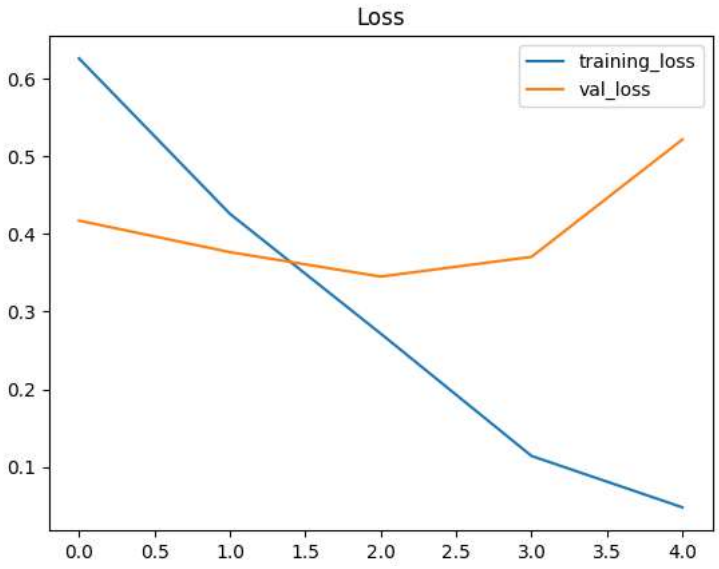
```
Epoch 1/5
47/47 [==============================] - 11s 198ms/step - loss: 0.6262 - accuracy: 0.6733 - val_loss: 0.4172 - val_accuracy: 0.8140
Epoch 2/5
47/47 [==============================] - 9s 192ms/step - loss: 0.4260 - accuracy: 0.8253 - val_loss: 0.3766 - val_accuracy: 0.8420
Epoch 3/5
47/47 [==============================] - 9s 191ms/step - loss: 0.2717 - accuracy: 0.9100 - val_loss: 0.3451 - val_accuracy: 0.8580
Epoch 4/5
47/47 [==============================] - 9s 191ms/step - loss: 0.1141 - accuracy: 0.9720 - val_loss: 0.3705 - val_accuracy: 0.8440
Epoch 5/5
47/47 [==============================] - 9s 191ms/step - loss: 0.0478 - accuracy: 0.9900 - val_loss: 0.5217 - val_accuracy: 0.8220
```

5. Evaluate the model

```
# Plot the training curves
import pandas as pd
pd.DataFrame(history_4.history).plot(figsize=(10, 7));
```




```
# Plot the validation and training data separately
def plot_loss_curves(history):
  """
  Returns separate loss curves for training and validation metrics.
  """
  loss = history.history['loss']
  val_loss = history.history['val_loss']

  accuracy = history.history['accuracy']
  val_accuracy = history.history['val_accuracy']

  epochs = range(len(history.history['loss']))

  # Plot loss
  plt.plot(epochs, loss, label='training_loss')
  plt.plot(epochs, val_loss, label='val_loss')
  plt.title('Loss')
  plt.xlabel('Epochs')
  plt.legend()

  # Plot accuracy
  plt.figure()
  plt.plot(epochs, accuracy, label='training_accuracy')
  plt.plot(epochs, val_accuracy, label='val_accuracy')
  plt.title('Accuracy')
  plt.xlabel('Epochs')
  plt.legend();


plot_loss_curves(history_4)
```




```
# model's architecture
model_4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 222, 222, 10)      280

 conv2d_5 (Conv2D)           (None, 220, 220, 10)      910

 conv2d_6 (Conv2D)           (None, 218, 218, 10)      910

 flatten_3 (Flatten)         (None, 475240)            0

 dense_8 (Dense)             (None, 1)                 475241

=================================================================
Total params: 477,341
Trainable params: 477,341
Non-trainable params: 0
_________________________________________________________________
```

```
# Create the model
model_5 = Sequential([
  Conv2D(10, 3, activation='relu', input_shape=(224, 224, 3)),
  MaxPool2D(pool_size=2), # reduce number of features by half
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Flatten(),
  Dense(1, activation='sigmoid')
])


# Compile model
model_5.compile(loss='binary_crossentropy',
                optimizer=Adam(),
                metrics=['accuracy'])


# Fit the model
history_5 = model_5.fit(train_data,
                        epochs=5,
                        steps_per_epoch=len(train_data),
                        validation_data=test_data,
                        validation_steps=len(test_data))
```

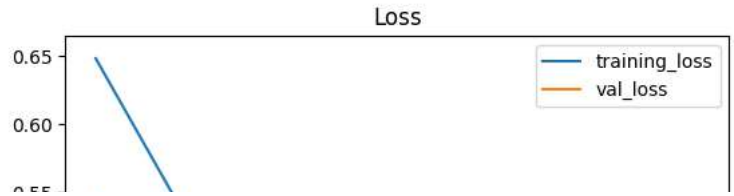
```
Epoch 1/5
47/47 [==============================] - 11s 196ms/step - loss: 0.6482 - accuracy: 0.6240 - val_loss: 0.5451 - val_accuracy: 0.7100
Epoch 2/5
47/47 [==============================] - 9s 190ms/step - loss: 0.4537 - accuracy: 0.8013 - val_loss: 0.3536 - val_accuracy: 0.8380
Epoch 3/5
47/47 [==============================] - 9s 192ms/step - loss: 0.4161 - accuracy: 0.8233 - val_loss: 0.3578 - val_accuracy: 0.8460
Epoch 4/5
```

```
47/47 [==============================] - 9s 192ms/step - loss: 0.3903 - accuracy: 0.8353 - val_loss: 0.3509 - val_accuracy: 0.8480
Epoch 5/5
47/47 [==============================] - 9s 192ms/step - loss: 0.3567 - accuracy: 0.8513 - val_loss: 0.3200 - val_accuracy: 0.8700
```

```
model_5.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 222, 222, 10)      280

 max_pooling2d_2 (MaxPooling  (None, 111, 111, 10)     0
 2D)

 conv2d_8 (Conv2D)           (None, 109, 109, 10)      910

 max_pooling2d_3 (MaxPooling  (None, 54, 54, 10)       0
 2D)

 conv2d_9 (Conv2D)           (None, 52, 52, 10)        910

 max_pooling2d_4 (MaxPooling  (None, 26, 26, 10)       0
 2D)

 flatten_4 (Flatten)         (None, 6760)              0

 dense_9 (Dense)             (None, 1)                 6761

=================================================================
Total params: 8,861
Trainable params: 8,861
Non-trainable params: 0
_________________________________________________________________
```

```
# Plot loss curves
plot_loss_curves(history_5)
```



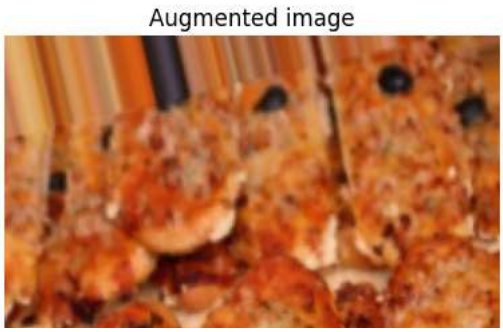

```
# Create ImageDataGenerator training instance with data augmentation
train_datagen_augmented = ImageDataGenerator(rescale=1/255.,
                                             rotation_range=20, # rotate the image slightly between 0 and 20 degrees (note: this is an int not
                                             shear_range=0.2, # shear the image
                                             zoom_range=0.2, # zoom into the image
                                             width_shift_range=0.2, # shift the image width ways
                                             height_shift_range=0.2, # shift the image height ways
                                             horizontal_flip=True) # flip the image on the horizontal axis

# Create ImageDataGenerator training instance without data augmentation
train_datagen = ImageDataGenerator(rescale=1/255.)

# Create ImageDataGenerator test instance without data augmentation
test_datagen = ImageDataGenerator(rescale=1/255.)
```

epochs

```
# Import data and augment it from training directory
print("Augmented training images:")
train_data_augmented = train_datagen_augmented.flow_from_directory(train_dir,
                                                                   target_size=(224, 224),
                                                                   batch_size=32,
                                                                   class_mode='binary',
                                                                   shuffle=False)

# Create non-augmented data batches
print("Non-augmented training images:")
train_data = train_datagen.flow_from_directory(train_dir,
                                               target_size=(224, 224),
                                               batch_size=32,
                                               class_mode='binary',
                                               shuffle=False)

print("Unchanged test images:")
test_data = test_datagen.flow_from_directory(test_dir,
                                             target_size=(224, 224),
                                             batch_size=32,
                                             class_mode='binary')
```

```
Augmented training images:
Found 1500 images belonging to 2 classes.
Non-augmented training images:
Found 1500 images belonging to 2 classes.
Unchanged test images:
Found 500 images belonging to 2 classes.
```

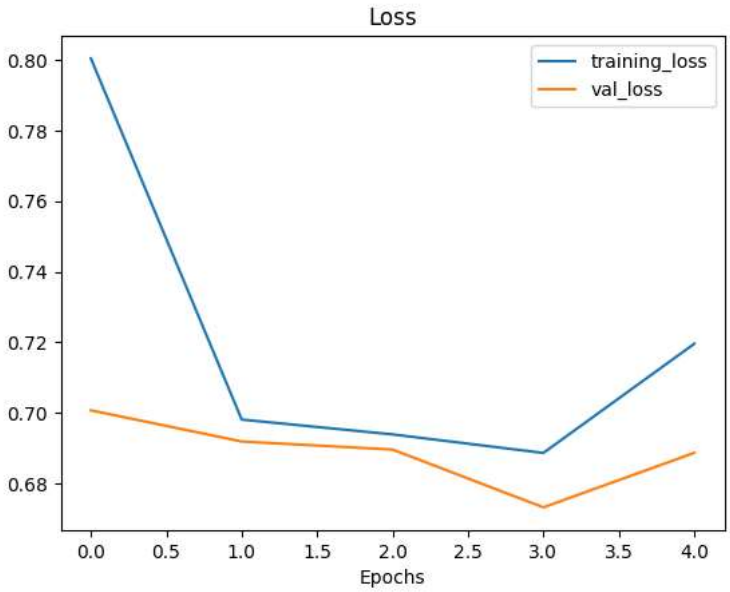
```
# Get data batch samples
images, labels = train_data.next()
augmented_images, augmented_labels = train_data_augmented.next()


# Show original image and augmented image
random_number = random.randint(0, 31)
plt.imshow(images[random_number])
plt.title(f"Original image")
plt.axis(False)
plt.figure()
plt.imshow(augmented_images[random_number])
plt.title(f"Augmented image")
plt.axis(False);
```

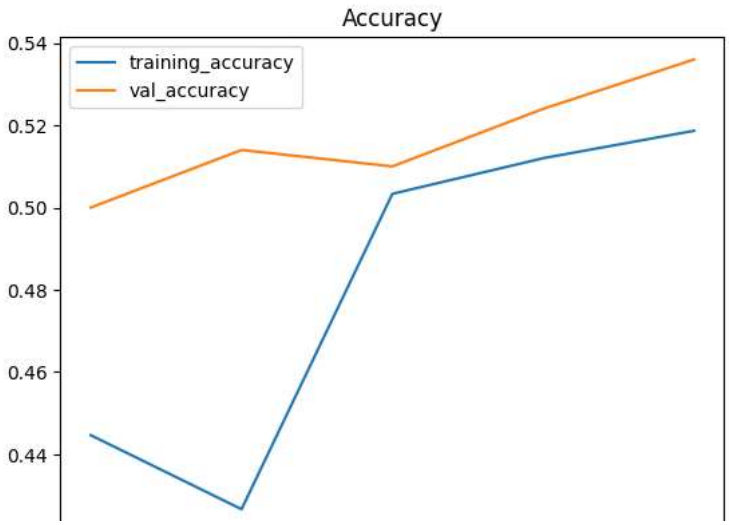



```
# Create the model
model_6 = Sequential([
  Conv2D(10, 3, activation='relu', input_shape=(224, 224, 3)),
  MaxPool2D(pool_size=2), # reduce number of features by half
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Flatten(),
  Dense(1, activation='sigmoid')
])

# Compile the model
model_6.compile(loss='binary_crossentropy',
                optimizer=Adam(),
                metrics=['accuracy'])

# Fit the model
history_6 = model_6.fit(train_data_augmented,
                        epochs=5,
                        steps_per_epoch=len(train_data_augmented),
                        validation_data=test_data,
                        validation_steps=len(test_data))
```

```
Epoch 1/5
47/47 [==============================] - 24s 472ms/step - loss: 0.8005 - accuracy: 0.4447 - val_loss: 0.7007 - val_accuracy: 0.5000
Epoch 2/5
47/47 [==============================] - 22s 475ms/step - loss: 0.6981 - accuracy: 0.4267 - val_loss: 0.6918 - val_accuracy: 0.5140
Epoch 3/5
47/47 [==============================] - 22s 474ms/step - loss: 0.6938 - accuracy: 0.5033 - val_loss: 0.6896 - val_accuracy: 0.5100
Epoch 4/5
47/47 [==============================] - 22s 477ms/step - loss: 0.6886 - accuracy: 0.5120 - val_loss: 0.6732 - val_accuracy: 0.5240
Epoch 5/5
47/47 [==============================] - 22s 477ms/step - loss: 0.7195 - accuracy: 0.5187 - val_loss: 0.6886 - val_accuracy: 0.5360
```

```
# Check model's performance history training on augmented data
plot_loss_curves(history_6)
```






```
# Import data and augment it from directories
train_data_augmented_shuffled = train_datagen_augmented.flow_from_directory(train_dir,
                                                                            target_size=(224, 224),
                                                                            batch_size=32,
                                                                            class_mode='binary',
                                                                            shuffle=True) # Shuffle data (default)
```

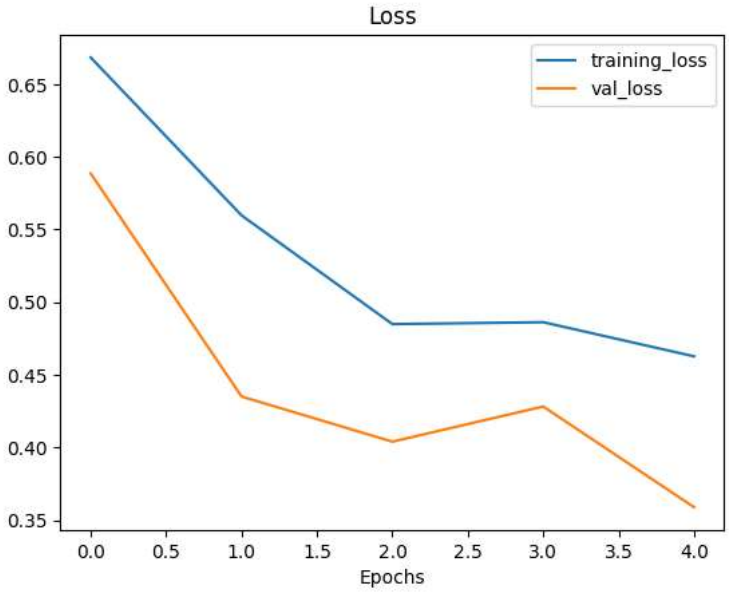
```
Found 1500 images belonging to 2 classes.
```

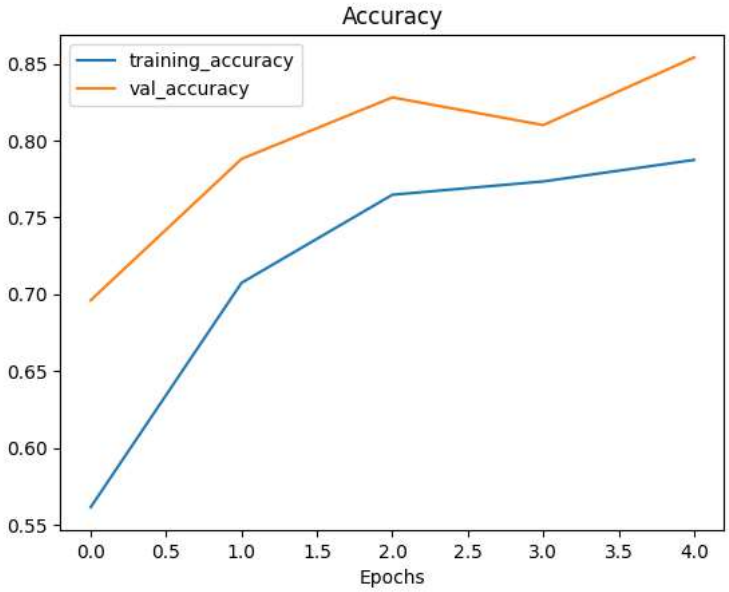
```
# Create the model (same as model_5 and model_6)
model_7 = Sequential([
  Conv2D(10, 3, activation='relu', input_shape=(224, 224, 3)),
  MaxPool2D(),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Flatten(),
  Dense(1, activation='sigmoid')
])

# Compile the model
model_7.compile(loss='binary_crossentropy',
                optimizer=Adam(),
                metrics=['accuracy'])

# Fit the model
history_7 = model_7.fit(train_data_augmented_shuffled, # now the augmented data is shuffled
                        epochs=5,
                        steps_per_epoch=len(train_data_augmented_shuffled),
```

```
                    validation_data=test_data,
                    validation_steps=len(test_data))
```

```
Epoch 1/5
47/47 [==============================] - 24s 474ms/step - loss: 0.6685 - accuracy: 0.5613 - val_loss: 0.5886 - val_accuracy: 0.6960
Epoch 2/5
47/47 [==============================] - 22s 478ms/step - loss: 0.5598 - accuracy: 0.7073 - val_loss: 0.4351 - val_accuracy: 0.7880
Epoch 3/5
47/47 [==============================] - 22s 477ms/step - loss: 0.4849 - accuracy: 0.7647 - val_loss: 0.4040 - val_accuracy: 0.8280
Epoch 4/5
47/47 [==============================] - 23s 487ms/step - loss: 0.4862 - accuracy: 0.7733 - val_loss: 0.4282 - val_accuracy: 0.8100
Epoch 5/5
47/47 [==============================] - 23s 486ms/step - loss: 0.4627 - accuracy: 0.7873 - val_loss: 0.3589 - val_accuracy: 0.8540
```

```
# Check model's performance history training on augmented data
plot_loss_curves(history_7)
```






```
# Create a CNN model
model_8 = Sequential([
  Conv2D(10, 3, activation='relu', input_shape=(224, 224, 3)),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Conv2D(10, 3, activation='relu'),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Flatten(),
  Dense(1, activation='sigmoid')
])
```

```
# Compile the model
model_8.compile(loss="binary_crossentropy",
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])

# Fit the model
history_8 = model_8.fit(train_data_augmented_shuffled,
                        epochs=5,
                        steps_per_epoch=len(train_data_augmented_shuffled),
                        validation_data=test_data,
                        validation_steps=len(test_data))
```

```
Epoch 1/5
47/47 [==============================] - 25s 492ms/step - loss: 0.6345 - accuracy: 0.6267 - val_loss: 0.4843 - val_accuracy: 0.7500
Epoch 2/5
47/47 [==============================] - 23s 495ms/step - loss: 0.5293 - accuracy: 0.7327 - val_loss: 0.3829 - val_accuracy: 0.8400
Epoch 3/5
47/47 [==============================] - 23s 490ms/step - loss: 0.5296 - accuracy: 0.7360 - val_loss: 0.5177 - val_accuracy: 0.7160
Epoch 4/5
47/47 [==============================] - 22s 475ms/step - loss: 0.4971 - accuracy: 0.7560 - val_loss: 0.3477 - val_accuracy: 0.8600
Epoch 5/5
47/47 [==============================] - 22s 476ms/step - loss: 0.4670 - accuracy: 0.7927 - val_loss: 0.3511 - val_accuracy: 0.8540
```

```
# Check model_1 architecture
model_1.summary()
```

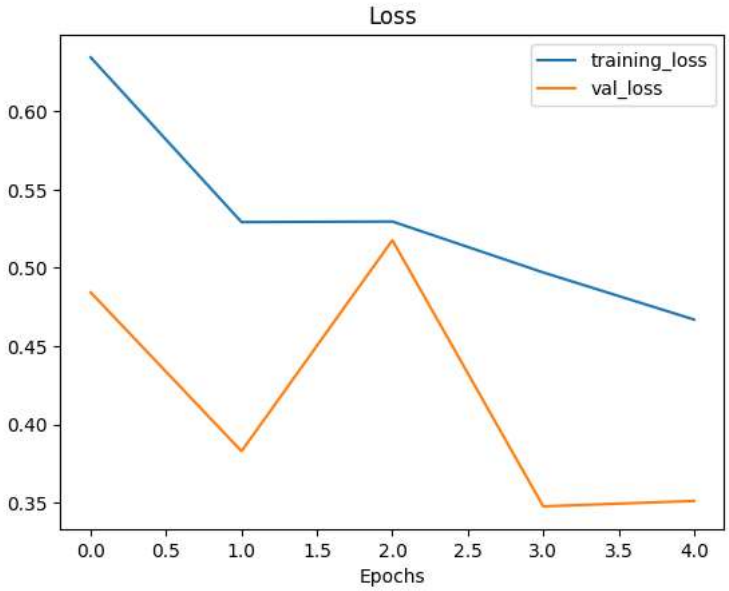
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 10)      280

 conv2d_1 (Conv2D)           (None, 220, 220, 10)      910

 max_pooling2d (MaxPooling2D  (None, 110, 110, 10)     0
 )

 conv2d_2 (Conv2D)           (None, 108, 108, 10)      910

 conv2d_3 (Conv2D)           (None, 106, 106, 10)      910

 max_pooling2d_1 (MaxPooling  (None, 53, 53, 10)       0
 2D)

 flatten (Flatten)           (None, 28090)             0

 dense (Dense)               (None, 1)                 28091

=================================================================
Total params: 31,101
Trainable params: 31,101
Non-trainable params: 0
_________________________________________________________________
```

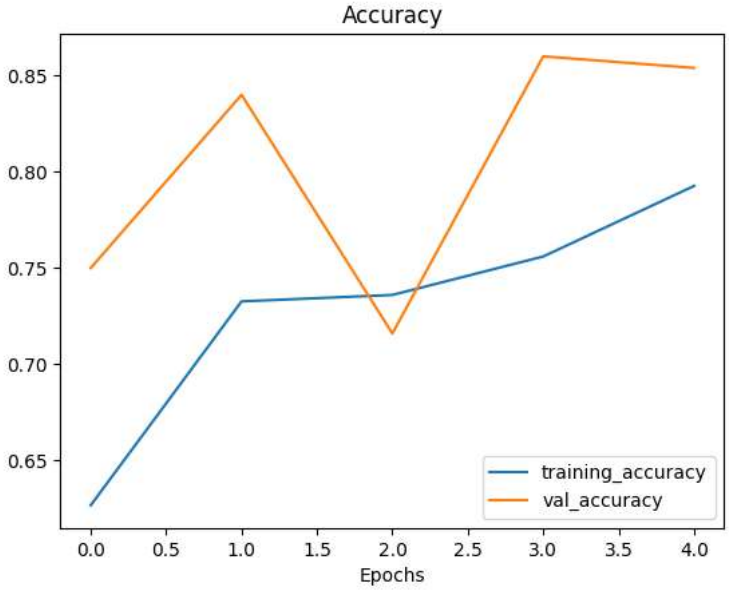
```
# Check model_8 architecture
model_8.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 222, 222, 10)      280

 conv2d_17 (Conv2D)          (None, 220, 220, 10)      910

 max_pooling2d_11 (MaxPoolin  (None, 110, 110, 10)     0
 g2D)

 conv2d_18 (Conv2D)          (None, 108, 108, 10)      910

 conv2d_19 (Conv2D)          (None, 106, 106, 10)      910

 max_pooling2d_12 (MaxPoolin  (None, 53, 53, 10)       0
 g2D)

 flatten_7 (Flatten)         (None, 28090)             0

 dense_12 (Dense)            (None, 1)                 28091

=================================================================
```

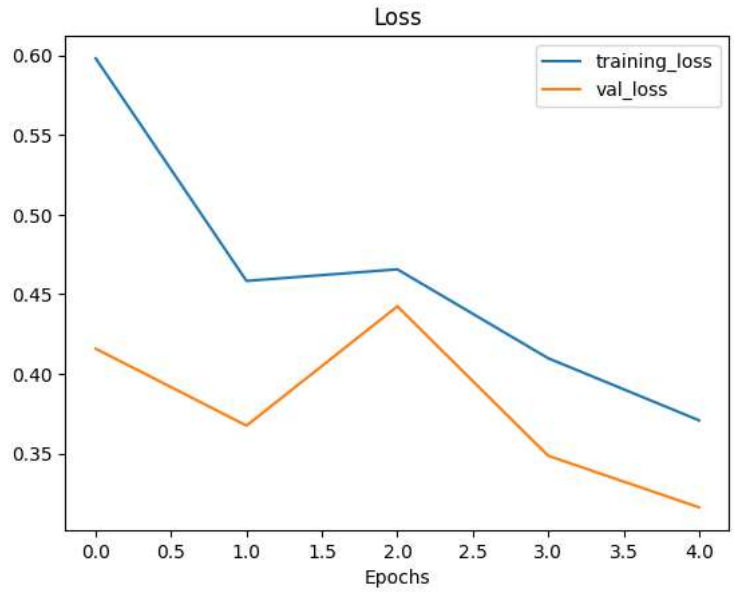
```
Total params: 31,101
Trainable params: 31,101
Non-trainable params: 0
_________________________________________________________________
```

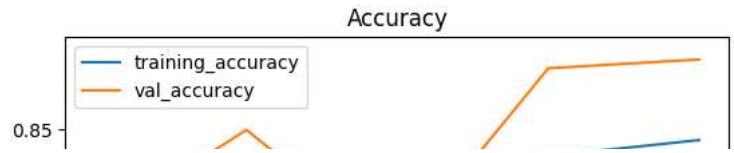
```
plot_loss_curves(history_8)
```






```
plot_loss_curves(history_1)
```






```
print(class_names)
```

```
['pizza' 'steak']
```

```
!wget https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-steak.jpeg
steak = mpimg.imread("03-steak.jpeg")
plt.imshow(steak)
plt.axis(False);
```

```
--2023-05-11 04:00:27--  https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-lea
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443
HTTP request sent, awaiting response... 200 OK
Length: 1978213 (1.9M) [image/jpeg]
Saving to: '03-steak.jpeg'

03-steak.jpeg       100%[===================>]   1.89M  --.-KB/s    in 0.02s

2023-05-11 04:00:27 (90.1 MB/s) - '03-steak.jpeg' saved [1978213/1978213]
```

```
# Check the shape of our image
steak.shape
```

```
(4032, 3024, 3)
```

```
def load_and_prep_image(filename, img_shape=224):
  """
  Reads an image from filename, turns it into a tensor
  and reshapes it to (img_shape, img_shape, colour_channel).
  """
  # Read in target file (an image)
  img = tf.io.read_file(filename)

  # Decode the read file into a tensor & ensure 3 colour channels
  # (our model is trained on images with 3 colour channels and sometimes images have 4 colour channels)
  img = tf.image.decode_image(img, channels=3)

  # Resize the image (to the same size our model was trained on)
  img = tf.image.resize(img, size = [img_shape, img_shape])

  # Rescale the image (get all values between 0 and 1)
  img = img/255.
  return img


# Load in and preprocess our custom image
steak = load_and_prep_image("03-steak.jpeg")
steak
```

```
<tf.Tensor: shape=(224, 224, 3), dtype=float32, numpy=
array([[[0.6377451 , 0.6220588 , 0.57892156],
        [0.6504902 , 0.63186276, 0.5897059 ],
        [0.63186276, 0.60833335, 0.5612745 ],
        ...,
        [0.52156866, 0.05098039, 0.09019608],
        [0.49509802, 0.04215686, 0.07058824],
        [0.52843136, 0.07745098, 0.10490196]],

       [[0.6617647 , 0.6460784 , 0.6107843 ],
        [0.6387255 , 0.6230392 , 0.57598037],
        [0.65588236, 0.63235295, 0.5852941 ],
        ...,
        [0.5352941 , 0.06862745, 0.09215686],
        [0.529902  , 0.05931373, 0.09460784],
        [0.5142157 , 0.05539216, 0.08676471]],

       [[0.6519608 , 0.6362745 , 0.5892157 ],
        [0.6392157 , 0.6137255 , 0.56764704],
        [0.65637255, 0.6269608 , 0.5828431 ],
        ...,
        [0.53137255, 0.06470589, 0.08039216],
        [0.527451  , 0.06862745, 0.1       ],
        [0.52254903, 0.05196078, 0.0872549 ]],

       ...,

       [[0.49313724, 0.42745098, 0.31029412],
        [0.05441177, 0.01911765, 0.        ],
        [0.2127451 , 0.16176471, 0.09509804],
        ...,
        [0.6132353 , 0.59362745, 0.57009804],
        [0.65294117, 0.6333333 , 0.6098039 ],
        [0.64166665, 0.62990195, 0.59460783]],

       [[0.65392154, 0.5715686 , 0.45      ],
        [0.6367647 , 0.54656863, 0.425     ],
        [0.04656863, 0.01372549, 0.        ],
        ...,
        [0.6372549 , 0.61764705, 0.59411764],
        [0.63529414, 0.6215686 , 0.5892157 ],
        [0.6401961 , 0.62058824, 0.59705883]],

       [[0.1       , 0.05539216, 0.        ],
        [0.48333332, 0.40882352, 0.29117647],
        [0.65      , 0.5686275 , 0.44019607],
        ...,
        [0.6308824 , 0.6161765 , 0.5808824 ],
        [0.6519608 , 0.63186276, 0.5901961 ],
        [0.6338235 , 0.6259804 , 0.57892156]]], dtype=float32)>
```

```
model_8.predict(steak)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-61-fd7eef5274d1> in <cell line: 2>()
      1 # Make a prediction on our custom image (spoiler: this won't work)
----> 2 model_8.predict(steak)

 1 frames
/usr/local/lib/python3.10/dist-packages/keras/engine/training.py in
tf__predict_function(iterator)
     13                 try:
     14                     do_return = True
---> 15                     retval_ = ag__.converted_call(ag__.ld(step_function),
(ag__.ld(self), ag__.ld(iterator)), None, fscope)
     16                 except:
     17                     do_return = False

ValueError: in user code:

    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py", line 2169,
in predict_function  *
        return step_function(self, iterator)
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py", line 2155,
in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py", line 2143,
in run_step  **
        outputs = model.predict_step(data)
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py", line 2111,
in predict_step
        return self(x, training=False)
```

```
# Add an extra axis
print(f"Shape before new dimension: {steak.shape}")
steak = tf.expand_dims(steak, axis=0) # add an extra dimension at axis 0
#steak = steak[tf.newaxis, ...] # alternative to the above, '...' is short for 'every other dimension'
print(f"Shape after new dimension: {steak.shape}")
steak
```

```
Shape before new dimension: (224, 224, 3)
Shape after new dimension: (1, 224, 224, 3)
<tf.Tensor: shape=(1, 224, 224, 3), dtype=float32, numpy=
array([[[[0.6377451 , 0.6220588 , 0.57892156],
         [0.6504902 , 0.63186276, 0.5897059 ],
         [0.63186276, 0.60833335, 0.5612745 ],
         ...,
         [0.52156866, 0.05098039, 0.09019608],
         [0.49509802, 0.04215686, 0.07058824],
         [0.52843136, 0.07745098, 0.10490196]],

        [[0.6617647 , 0.6460784 , 0.6107843 ],
         [0.6387255 , 0.6230392 , 0.57598037],
         [0.65588236, 0.63235295, 0.5852941 ],
         ...,
         [0.5352941 , 0.06862745, 0.09215686],
         [0.529902  , 0.05931373, 0.09460784],
         [0.5142157 , 0.05539216, 0.08676471]],

        [[0.6519608 , 0.6362745 , 0.5892157 ],
         [0.6392157 , 0.6137255 , 0.56764704],
         [0.65637255, 0.6269608 , 0.5828431 ],
         ...,
         [0.53137255, 0.06470589, 0.08039216],
         [0.527451  , 0.06862745, 0.1       ],
         [0.52254903, 0.05196078, 0.0872549 ]],

        ...,

        [[0.49313724, 0.42745098, 0.31029412],
         [0.05441177, 0.01911765, 0.        ],
         [0.2127451 , 0.16176471, 0.09509804],
         ...,
         [0.6132353 , 0.59362745, 0.57009804],
         [0.65294117, 0.6333333 , 0.6098039 ],
         [0.64166665, 0.62990195, 0.59460783]],

        [[0.65392154, 0.5715686 , 0.45      ],
         [0.6367647 , 0.54656863, 0.425     ],
         [0.04656863, 0.01372549, 0.        ],
         ...,
         [0.6372549 , 0.61764705, 0.59411764],
         [0.63529414, 0.6215686 , 0.5892157 ],
         [0.6401961 , 0.62058824, 0.59705883]],
```

```
[[0.1       , 0.05539216, 0.        ],
 [0.48333332, 0.40882352, 0.29117647],
 [0.65      , 0.5686275 , 0.44019607],
 ...,
 [0.6308824 , 0.6161765 , 0.5808824 ],
 [0.6519608 , 0.63186276, 0.5901961 ],
 [0.6338235 , 0.6259804 , 0.57892156]]], dtype=float32)>
```

```
# Make a prediction on custom image tensor
pred = model_8.predict(steak)
pred
```

```
1/1 [==============================] - 0s 401ms/step
array([[0.7315909]], dtype=float32)
```

```
class_names
```

```
array(['pizza', 'steak'], dtype='<U5')
```

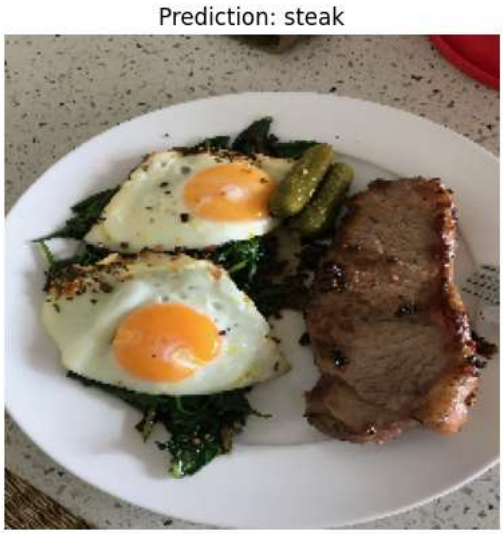
```
pred_class = class_names[int(tf.round(pred)[0][0])]
pred_class
```

```
'steak'
```

```
def pred_and_plot(model, filename, class_names):
  """
  Imports an image located at filename, makes a prediction on it with
  a trained model and plots the image with the predicted class as the title.
  """
  # Import the target image and preprocess it
  img = load_and_prep_image(filename)

  # Make a prediction
  pred = model.predict(tf.expand_dims(img, axis=0))

  # Get the predicted class
  pred_class = class_names[int(tf.round(pred)[0][0])]

  # Plot the image and predicted class
  plt.imshow(img)
  plt.title(f"Prediction: {pred_class}")
  plt.axis(False);
```

```
pred_and_plot(model_8, "03-steak.jpeg", class_names)
```

```
1/1 [==============================] - 0s 23ms/step
```



1. Import and become one with the data

Again, we've got a subset of the Food101 dataset. In addition to the pizza and steak images, we've pulled out another eight classes.

```
import os

# Walk through 10_food_classes directory and list number of files
for dirpath, dirnames, filenames in os.walk("10_food_classes_all_data"):
  print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

```
There are 2 directories and 0 images in '10_food_classes_all_data'.
There are 10 directories and 0 images in '10_food_classes_all_data/test'.
There are 0 directories and 250 images in '10_food_classes_all_data/test/chicken_wings'.
There are 0 directories and 250 images in '10_food_classes_all_data/test/fried_rice'.
There are 0 directories and 250 images in '10_food_classes_all_data/test/ice_cream'.
There are 0 directories and 250 images in '10_food_classes_all_data/test/grilled_salmon'.
There are 0 directories and 250 images in '10_food_classes_all_data/test/chicken_curry'.
There are 0 directories and 250 images in '10_food_classes_all_data/test/hamburger'.
There are 0 directories and 250 images in '10_food_classes_all_data/test/steak'.
There are 0 directories and 250 images in '10_food_classes_all_data/test/pizza'.
There are 0 directories and 250 images in '10_food_classes_all_data/test/sushi'.
There are 0 directories and 250 images in '10_food_classes_all_data/test/ramen'.
There are 10 directories and 0 images in '10_food_classes_all_data/train'.
There are 0 directories and 750 images in '10_food_classes_all_data/train/chicken_wings'.
There are 0 directories and 750 images in '10_food_classes_all_data/train/fried_rice'.
There are 0 directories and 750 images in '10_food_classes_all_data/train/ice_cream'.
There are 0 directories and 750 images in '10_food_classes_all_data/train/grilled_salmon'.
There are 0 directories and 750 images in '10_food_classes_all_data/train/chicken_curry'.
There are 0 directories and 750 images in '10_food_classes_all_data/train/hamburger'.
There are 0 directories and 750 images in '10_food_classes_all_data/train/steak'.
There are 0 directories and 750 images in '10_food_classes_all_data/train/pizza'.
There are 0 directories and 750 images in '10_food_classes_all_data/train/sushi'.
There are 0 directories and 750 images in '10_food_classes_all_data/train/ramen'.
```

```
train_dir = "10_food_classes_all_data/train/"
test_dir = "10_food_classes_all_data/test/"
```

```
# Get the class names
import pathlib
import numpy as np
data_dir = pathlib.Path(train_dir)
class_names = np.array(sorted([item.name for item in data_dir.glob('*')]))
print(class_names)
```

```
['chicken_curry' 'chicken_wings' 'fried_rice' 'grilled_salmon' 'hamburger'
 'ice_cream' 'pizza' 'ramen' 'steak' 'sushi']
```

```
# View a random image from the training dataset
import random
img = view_random_image(target_dir=train_dir,
                        target_class=random.choice(class_names)) # get a random class name
```

```
Image shape: (512, 384, 3)
```


```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Rescale the data and create data generator instances
train_datagen = ImageDataGenerator(rescale=1/255.)
test_datagen = ImageDataGenerator(rescale=1/255.)
```

```
# Load data in from directories and turn it into batches
train_data = train_datagen.flow_from_directory(train_dir,
                                               target_size=(224, 224),
                                               batch_size=32,
                                               class_mode='categorical') # changed to categorical

test_data = train_datagen.flow_from_directory(test_dir,
                                              target_size=(224, 224),
                                              batch_size=32,
                                              class_mode='categorical')
```

```
Found 7500 images belonging to 10 classes.
Found 2500 images belonging to 10 classes.
```

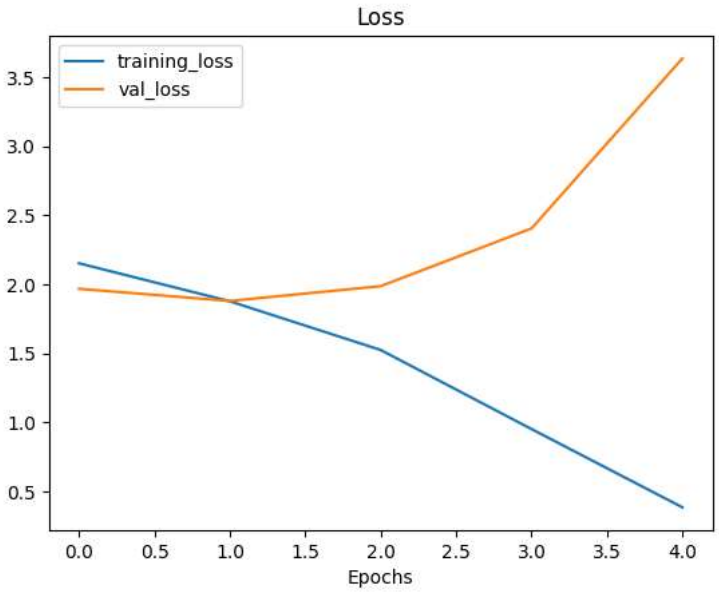
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense

model_9 = Sequential([
  Conv2D(10, 3, activation='relu', input_shape=(224, 224, 3)),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Conv2D(10, 3, activation='relu'),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Flatten(),
  Dense(10, activation='softmax') # changed to have 10 neurons (same as number of classes) and 'softmax' activation
])

# Compile the model
model_9.compile(loss="categorical_crossentropy", # changed to categorical_crossentropy
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])


# Fit the model
history_9 = model_9.fit(train_data, # now 10 different classes
                        epochs=5,
                        steps_per_epoch=len(train_data),
                        validation_data=test_data,
                        validation_steps=len(test_data))
```

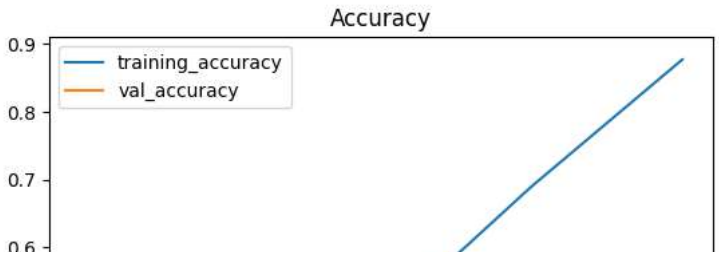
```
Epoch 1/5
235/235 [==============================] - 48s 194ms/step - loss: 2.1520 - accuracy: 0.2133 - val_loss: 1.9674 - val_accuracy: 0.2964
Epoch 2/5
235/235 [==============================] - 45s 193ms/step - loss: 1.8770 - accuracy: 0.3489 - val_loss: 1.8792 - val_accuracy: 0.3424
Epoch 3/5
235/235 [==============================] - 45s 190ms/step - loss: 1.5239 - accuracy: 0.4860 - val_loss: 1.9865 - val_accuracy: 0.3184
Epoch 4/5
235/235 [==============================] - 45s 193ms/step - loss: 0.9507 - accuracy: 0.6897 - val_loss: 2.4049 - val_accuracy: 0.2824
Epoch 5/5
235/235 [==============================] - 45s 190ms/step - loss: 0.3835 - accuracy: 0.8772 - val_loss: 3.6373 - val_accuracy: 0.2760
```

```
# Evaluate on the test data
model_9.evaluate(test_data)
```

```
79/79 [==============================] - 11s 139ms/step - loss: 3.6373 - accuracy: 0.2760
[3.6373109817504883, 0.2759999930858612]
```

```
# Check out the model's loss curves on the 10 classes of data (note: this function comes from above in the notebook)
plot_loss_curves(history_9)
```

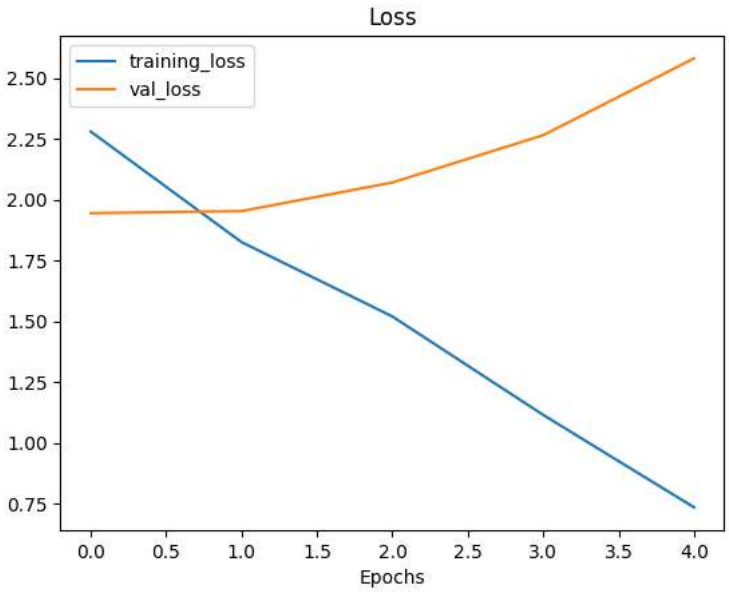


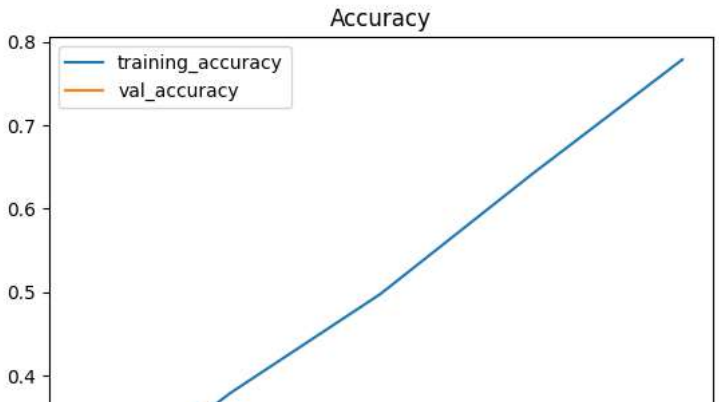



```
# Try a simplified model (removed two layers)
model_10 = Sequential([
  Conv2D(10, 3, activation='relu', input_shape=(224, 224, 3)),
  MaxPool2D(),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Flatten(),
  Dense(10, activation='softmax')
])

model_10.compile(loss='categorical_crossentropy',
                 optimizer=tf.keras.optimizers.Adam(),
                 metrics=['accuracy'])

history_10 = model_10.fit(train_data,
                          epochs=5,
                          steps_per_epoch=len(train_data),
                          validation_data=test_data,
                          validation_steps=len(test_data))
```

```
Epoch 1/5
235/235 [==============================] - 46s 193ms/step - loss: 2.2804 - accuracy: 0.2397 - val_loss: 1.9445 - val_accuracy: 0.3024
Epoch 2/5
235/235 [==============================] - 45s 192ms/step - loss: 1.8253 - accuracy: 0.3785 - val_loss: 1.9529 - val_accuracy: 0.3172
Epoch 3/5
235/235 [==============================] - 44s 186ms/step - loss: 1.5193 - accuracy: 0.4979 - val_loss: 2.0705 - val_accuracy: 0.3160
Epoch 4/5
235/235 [==============================] - 44s 187ms/step - loss: 1.1143 - accuracy: 0.6408 - val_loss: 2.2653 - val_accuracy: 0.3064
Epoch 5/5
235/235 [==============================] - 44s 188ms/step - loss: 0.7340 - accuracy: 0.7787 - val_loss: 2.5814 - val_accuracy: 0.2944
```

```
# Check out the loss curves of model_10
plot_loss_curves(history_10)
```






```
# Create augmented data generator instance
train_datagen_augmented = ImageDataGenerator(rescale=1/255.,
                                             rotation_range=20, # note: this is an int not a float
                                             width_shift_range=0.2,
                                             height_shift_range=0.2,
                                             zoom_range=0.2,
                                             horizontal_flip=True)

train_data_augmented = train_datagen_augmented.flow_from_directory(train_dir,
                                                                   target_size=(224, 224),
                                                                   batch_size=32,
                                                                   class_mode='categorical')
```

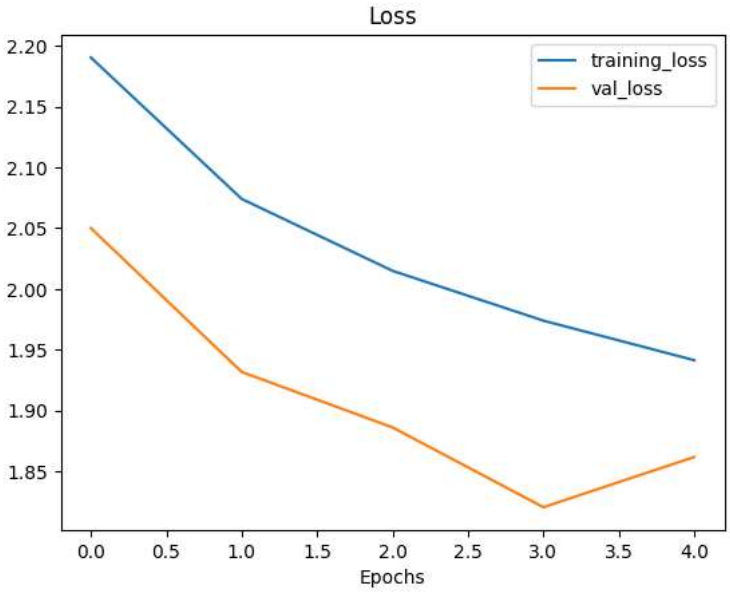
```
Found 7500 images belonging to 10 classes.
```

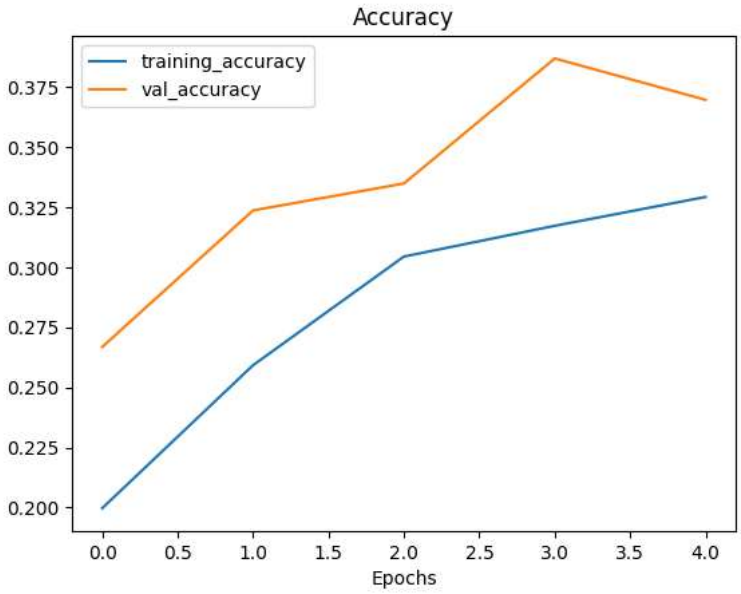
```
# Clone the model
model_11 = tf.keras.models.clone_model(model_10)

# Compile the cloned model
model_11.compile(loss="categorical_crossentropy",
                 optimizer=tf.keras.optimizers.Adam(),
                 metrics=["accuracy"])

# Fit the model
history_11 = model_11.fit(train_data_augmented, # use augmented data
                          epochs=5,
                          steps_per_epoch=len(train_data_augmented),
                          validation_data=test_data,
                          validation_steps=len(test_data))
```

```
Epoch 1/5
235/235 [==============================] - 112s 473ms/step - loss: 2.1903 - accuracy: 0.1997 - val_loss: 2.0500 - val_accuracy: 0.2668
Epoch 2/5
235/235 [==============================] - 114s 483ms/step - loss: 2.0740 - accuracy: 0.2592 - val_loss: 1.9318 - val_accuracy: 0.3236
Epoch 3/5
235/235 [==============================] - 114s 485ms/step - loss: 2.0149 - accuracy: 0.3044 - val_loss: 1.8862 - val_accuracy: 0.3348
```

```
Epoch 4/5
235/235 [==============================] - 114s 487ms/step - loss: 1.9739 - accuracy: 0.3172 - val_loss: 1.8206 - val_accuracy: 0.3868
Epoch 5/5
235/235 [==============================] - 115s 489ms/step - loss: 1.9415 - accuracy: 0.3292 - val_loss: 1.8619 - val_accuracy: 0.3696
```

```
# Check model's performance with augmented data
plot_loss_curves(history_11)
```






```
# What classes has our model been trained on?
class_names
```

```
array(['chicken_curry', 'chicken_wings', 'fried_rice', 'grilled_salmon',
       'hamburger', 'ice_cream', 'pizza', 'ramen', 'steak', 'sushi'],
      dtype='<U14')
```

```
# -q is for "quiet"
!wget -q https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-pizza-dad.jpeg
!wget -q https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-steak.jpeg
!wget -q https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-hamburger.jpeg
!wget -q https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-sushi.jpeg
```

```
# Make a prediction using model_11
pred_and_plot(model=model_11,
              filename="03-steak.jpeg",
              class_names=class_names)
```

```
1/1 [==============================] - 0s 81ms/step
```


```
pred_and_plot(model_11, "03-sushi.jpeg", class_names)
```

```
1/1 [==============================] - 0s 23ms/step
```


```
# Load in and preprocess our custom image
img = load_and_prep_image("03-steak.jpeg")

# Make a prediction
pred = model_11.predict(tf.expand_dims(img, axis=0))

# Match the prediction class to the highest prediction probability
pred_class = class_names[pred.argmax()]
plt.imshow(img)
plt.title(pred_class)
plt.axis(False);
```

```
1/1 [==============================] - 0s 22ms/step
```


```
# Check the output of the predict function
pred = model_11.predict(tf.expand_dims(img, axis=0))
pred
```

```
1/1 [==============================] - 0s 22ms/step
array([[0.00413127, 0.00698275, 0.00470221, 0.14550138, 0.0163848 ,
        0.00341261, 0.00258247, 0.2162087 , 0.5952408 , 0.00485306]],
      dtype=float32)
```

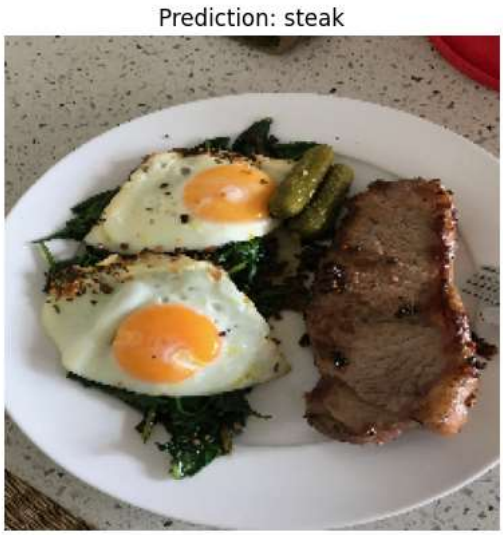
```
# Find the predicted class name
class_names[pred.argmax()]
```

```
'steak'
```

```
# Adjust function to work with multi-class
def pred_and_plot(model, filename, class_names):
  """
  Imports an image located at filename, makes a prediction on it with
  a trained model and plots the image with the predicted class as the title.
  """
  # Import the target image and preprocess it
  img = load_and_prep_image(filename)

  # Make a prediction
  pred = model.predict(tf.expand_dims(img, axis=0))

  # Get the predicted class
  if len(pred[0]) > 1: # check for multi-class
    pred_class = class_names[pred.argmax()] # if more than one output, take the max
  else:
    pred_class = class_names[int(tf.round(pred)[0][0])] # if only one output, round

  # Plot the image and predicted class
  plt.imshow(img)
  plt.title(f"Prediction: {pred_class}")
  plt.axis(False);
```

```
pred_and_plot(model_11, "03-steak.jpeg", class_names)
```

```
1/1 [==============================] - 0s 23ms/step
```




```
pred_and_plot(model_11, "03-sushi.jpeg", class_names)
```

```
1/1 [==============================] - 0s 22ms/step
```


```
pred_and_plot(model_11, "03-hamburger.jpeg", class_names)
```

```
1/1 [==============================] - 0s 22ms/step
```


```
# Save a model
model_11.save("saved_trained_model")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 2 of 2).
```

```
# Load in a model and evaluate it
loaded_model_11 = tf.keras.models.load_model("saved_trained_model")
loaded_model_11.evaluate(test_data)
```

```
79/79 [==============================] - 11s 141ms/step - loss: 1.8619 - accuracy: 0.3696
[1.8618727922439575, 0.36959999799728394]
```

```
model_11.evaluate(test_data)
```

```
79/79 [==============================] - 11s 140ms/step - loss: 1.8619 - accuracy: 0.3696
[1.8618724346160889, 0.36959999799728394]
```