DL in Applied Mathematics

Lecture 9: CONVOLUTIONAL NEURAL NETWORK (CNN) & COMPUTER VISION

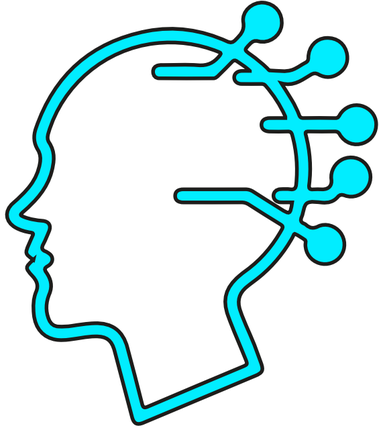
CNN

CNNS, A CLASS OF DEEP LEARNING ALGORITHMS, HAVE REVOLUTIONIZED HOW MACHINES PERCEIVE AND INTERPRET VISUAL INFORMATION, ENABLING ADVANCEMENTS IN VARIOUS APPLICATIONS FROM AUTONOMOUS VEHICLES TO MEDICAL IMAGE ANALYSIS. THIS ARTICLE DELVES INTO THE INTRICACIES OF CNN ARCHITECTURES, THEIR OPERATIONAL MECHANISMS, AND HOW THEY EXTRACT AND LEARN FEATURES FROM VISUAL INPUTS. BY COMPREHENDING THESE ASPECTS, WE CAN APPRECIATE THE COMPLEXITIES AND THE POTENTIAL OF CNNS IN TRANSFORMING OUR INTERACTION WITH THE DIGITAL WORLD.

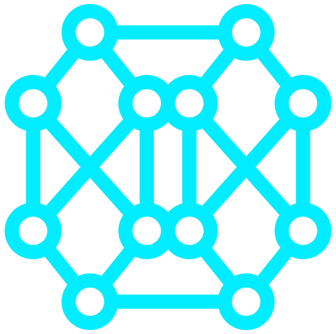
AT FIRST, A ZIP FILE CONTAINING PIZZA AND STEAK IMAGES IS FIRST DOWNLOADED USING THE WGET COMMAND FROM A GOOGLE CLOUD STORAGE URL. THIS FILE IS INTENDED FOR USE IN A FOOD VISION PROJECT. FOLLOWING THE DOWNLOAD, THE PYTHON ZIPFILE MODULE IS UTILIZED TO OPEN THE PIZZA_STEAK.ZIP FILE IN READ MODE, EXTRACT ITS CONTENTS TO THE CURRENT WORKING DIRECTORY, AND THEN CLOSE THE FILE TO EFFICIENTLY MANAGE SYSTEM RESOURCES.

```
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()

# Saving to: ‘pizza_steak.zip’
# pizza_steak.zip     100%[===================>] 104.47M  33.5MB/s    in 3.1s
```

```
!ls pizza_steak
#test  train
!ls pizza_steak/train/
#pizza  steak
!ls pizza_steak/train/steak/
```



```
1000205.jpg  1598345.jpg  2062248.jpg  2548974.jpg  3030578.jpg  3571963.jpg  510757.jpg
100135.jpg   1598885.jpg  2081995.jpg  2549316.jpg  3047807.jpg  3576078.jpg  513129.jpg
101312.jpg   1600179.jpg  2087958.jpg  2561199.jpg  3059843.jpg  3577618.jpg  513842.jpg
1021458.jpg  1600794.jpg  2088030.jpg  2563233.jpg  3074367.jpg  3577732.jpg  523535.jpg
1032846.jpg  160552.jpg   2088195.jpg  256592.jpg   3082120.jpg  3578934.jpg  525041.jpg
10380.jpg    1606596.jpg  2090493.jpg  2568848.jpg  3094354.jpg  358042.jpg   534560.jpg
1049459.jpg  1615395.jpg  2090504.jpg  2573392.jpg  3095301.jpg  358045.jpg   534633.jpg
1053665.jpg  1618011.jpg  2125877.jpg  2592401.jpg  3099645.jpg  3591821.jpg  536535.jpg
1068516.jpg  1619357.jpg  2129685.jpg  2599817.jpg  3100476.jpg  359330.jpg   541410.jpg
1068975.jpg  1621763.jpg  2133717.jpg  2603058.jpg  3110387.jpg  3601483.jpg  543691.jpg
1081258.jpg  1623325.jpg  2136662.jpg  2606444.jpg  3113772.jpg  3606642.jpg  560503.jpg
1090122.jpg  1624450.jpg  213765.jpg   2614189.jpg  3116018.jpg  3609394.jpg  561972.jpg
1093966.jpg  1624747.jpg  2138335.jpg  2614649.jpg  3128952.jpg  361067.jpg   56240.jpg
1098844.jpg  1628861.jpg  2140776.jpg  2615718.jpg  3130412.jpg  3613455.jpg  56409.jpg
1100074.jpg  1632774.jpg  214320.jpg   2619625.jpg  3136.jpg     3621464.jpg  564530.jpg
1105280.jpg  1636831.jpg  2146963.jpg  2622140.jpg  313851.jpg   3621562.jpg  568972.jpg
1117936.jpg  1645470.jpg  215222.jpg   262321.jpg   3140083.jpg  3621565.jpg  576725.jpg
1126126.jpg  1647351.jpg  2154126.jpg  2625330.jpg  3140147.jpg  3623556.jpg  588739.jpg
114601.jpg   1650002.jpg  2154779.jpg  2628106.jpg  3142045.jpg  3640915.jpg  590142.jpg
1147047.jpg  165639.jpg   2159975.jpg  2629750.jpg  3142618.jpg  3643951.jpg  60633.jpg
1147883.jpg  1658186.jpg  2163079.jpg  2643906.jpg  3142674.jpg  3653129.jpg  60655.jpg
1155665.jpg  1658443.jpg  217250.jpg   2644457.jpg  3143192.jpg  3656752.jpg  606820.jpg
1163977.jpg  165964.jpg   2172600.jpg  2648423.jpg  314359.jpg   3663518.jpg  612551.jpg
1190233.jpg  167069.jpg   2173084.jpg  2651300.jpg  3157832.jpg  3663800.jpg  614975.jpg
1208405.jpg  1675632.jpg  217996.jpg   2653594.jpg  3159818.jpg  3664376.jpg  616809.jpg
1209120.jpg  1678108.jpg  2193684.jpg  2661577.jpg  3162376.jpg  3670607.jpg  628628.jpg
1212161.jpg  168006.jpg   220341.jpg   2668916.jpg  3168620.jpg  3671021.jpg  632427.jpg
1213988.jpg  1682496.jpg  22080.jpg    268444.jpg   3171085.jpg  3671877.jpg  636594.jpg
1219039.jpg  1684438.jpg  2216146.jpg  2691461.jpg  317206.jpg   368073.jpg   637374.jpg
1225762.jpg  168775.jpg   2222018.jpg  2706403.jpg  3173444.jpg  368162.jpg   640539.jpg
1230968.jpg  1697339.jpg  2223787.jpg  270687.jpg   3180182.jpg  368170.jpg   644777.jpg
1236155.jpg  1710569.jpg  2230959.jpg  2707522.jpg  31881.jpg    3693649.jpg  644867.jpg
1241193.jpg  1714605.jpg  2232310.jpg  2711806.jpg  3191589.jpg  3700079.jpg  658189.jpg
1248337.jpg  1724387.jpg  2233395.jpg  2716993.jpg  3204977.jpg  3704103.jpg  660900.jpg
1257104.jpg  1724717.jpg  2238681.jpg  2724554.jpg  320658.jpg   3707493.jpg  663014.jpg
126345.jpg   172936.jpg   2238802.jpg  2738227.jpg  3209173.jpg  3716881.jpg  664545.jpg
```



```
import os

# Walk through pizza_steak directory and list number of files
for dirpath, dirnames, filenames in os.walk("pizza_steak"):
  print(f"There are {len(dirnames)} directories and {len(filenames)} imag

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

```
# 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 (programmatically, this is much more helpful with a longer
import pathlib
import numpy as np
data_dir = pathlib.Path("pizza_steak/train/") # turn our training path into a Py
class_names = np.array(sorted([item.name for item in data_dir.glob('*')])) # cre
print(class_names)
#['pizza' 'steak']
```

THE OS MODULE IS USED TO NAVIGATE THROUGH THE CONTENTS OF THE "PIZZA_STEAK" DIRECTORY. THE OS.WALK("PIZZA_STEAK") FUNCTION IS EMPLOYED TO TRAVERSE THE DIRECTORY STRUCTURE, RETURNING A TUPLE CONTAINING THE DIRECTORY PATH, DIRECTORY NAMES, AND FILENAMES FOR EACH ITERATION. FOR EACH DIRECTORY IN THE "PIZZA_STEAK" STRUCTURE, A FORMATTED PRINT STATEMENT REPORTS THE NUMBER OF SUBDIRECTORIES (DIRNAMES) AND IMAGES (FILENAMES) PRESENT, PROVIDING A CLEAR OVERVIEW OF THE DIRECTORY'S STRUCTURE AND CONTENTS.

THE FUNCTION TAKES TWO PARAMETERS: TARGET_DIR, THE MAIN DIRECTORY, AND TARGET_CLASS, THE SUBCATEGORY WITHIN THAT DIRECTORY. INITIALLY, THE FUNCTION CONSTRUCTS THE PATH TO THE TARGET FOLDER BY COMBINING THESE PARAMETERS. THEN, IT SELECTS A RANDOM IMAGE FROM THIS FOLDER USING THE OS.LISTDIR AND RANDOM.SAMPLE METHODS. THE SELECTED IMAGE IS READ AND DISPLAYED USING THE MATPLOTLIB LIBRARY, WITH THE TITLE SET TO THE TARGET CLASS AND THE AXIS TURNED OFF FOR A CLEANER VIEW. ADDITIONALLY, THE SHAPE OF THE IMAGE IS PRINTED, PROVIDING DETAILS ABOUT ITS DIMENSIONS.

```
# 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")

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

# View the image shape
img.shape # returns (width, height, colour channels)
(512, 512, 3)
```

```
array([[[ 92,  33,  27],
        [ 98,  39,  33],
        [ 98,  39,  33],
        ...,
        [ 27,  19,  17],
        [ 28,  19,  20],
        [ 27,  18,  19]],

       [[ 98,  39,  33],
        [100,  41,  35],
        [ 99,  40,  34],
        ...,
        [ 31,  23,  21],
        [ 30,  20,  21],
        [ 27,  18,  19]],

       [[102,  45,  38],
        [102,  45,  38],
        [101,  44,  37],
        ...,
        [ 33,  23,  21],
        [ 32,  20,  20],
        [ 28,  18,  17]],

       ...,

       [[232, 208, 174],
        [228, 204, 170],
        [225, 202, 170],
        ...,
        [109,  91,  67],
        [111,  93,  71],
        [111,  93,  71]],

       [[232, 208, 172],
        [230, 206, 170],
        [227, 204, 170],
        ...,
        [114,  96,  72],
        [120, 102,  80],
        [122, 104,  82]],

       [[232, 208, 172],
        [230, 206, 170],
        [228, 205, 171],
        ...,
        [118, 100,  76],
        [127, 109,  87],
        [131, 113,  91]]], dtype=uint8)
```

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

steak

As we know, many machine learning models, including neural networks prefer the values they work with to be between 0 and 1. Knowing this, one of the most common preprocessing steps for working with images is to scale (also referred to as normalize) their pixel values by dividing the image arrays by 255

[[0.90980392, 0.81568627, 0.68235294],
[0.89411765, 0.8 , 0.66666667],
[0.88235294, 0.79215686, 0.66666667],
...,
[0.42745098, 0.35686275, 0.2627451],
[0.43529412, 0.36470588, 0.27843137],
[0.43529412, 0.36470588, 0.27843137]],

[[0.90980392, 0.81568627, 0.6745098],
[0.90196078, 0.80784314, 0.66666667],
[0.89019608, 0.8 , 0.66666667],
...,
[0.44705882, 0.37647059, 0.28235294],
[0.47058824, 0.4 , 0.31372549],
[0.47843137, 0.40784314, 0.32156863]],

[[0.90980392, 0.81568627, 0.6745098],
[0.90196078, 0.80784314, 0.66666667],
[0.89411765, 0.80392157, 0.67058824],
...,
[0.4627451 , 0.39215686, 0.29803922],
[0.49803922, 0.42745098, 0.34117647],
[0.51372549, 0.44313725, 0.35686275]]])

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

Output:

array([[[0.36078431, 0.12941176,
0.10588235],
[0.38431373, 0.15294118, 0.12941176],
[0.38431373, 0.15294118, 0.12941176],
...,
[0.10588235, 0.0745098 , 0.06666667],
[0.10980392, 0.0745098 , 0.07843137],
[0.10588235, 0.07058824, 0.0745098]],

[[0.38431373, 0.15294118, 0.12941176],
[0.39215686, 0.16078431, 0.1372549],
[0.38823529, 0.15686275, 0.13333333],
...,
[0.12156863, 0.09019608, 0.08235294],
[0.11764706, 0.07843137, 0.08235294],
[0.10588235, 0.07058824, 0.0745098]],

[[0.4 , 0.17647059, 0.14901961],
[0.4 , 0.17647059, 0.14901961],
[0.39607843, 0.17254902, 0.14509804],
...,
[0.12941176, 0.09019608, 0.08235294],
[0.1254902 , 0.07843137, 0.07843137],
[0.10980392, 0.07058824, 0.06666667]],

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

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

# Preprocess data (get all of the pixel values between 1 and 0, also called scaling/normalization)
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 we're working on
                                               seed=42)

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

# Create a CNN model (same as Tiny VGG - https://poloclub.github.io/cnn-explainer/)
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"), # activation='relu' == tf.keras.layers.Activations(tf.nn.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))
```

Output:

```
Found 1500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Epoch 1/5
47/47 [==============================] - 18s 134ms/step - loss: 0.6059 - accuracy: 0.6800 - val_loss: 0.4634 - val_accuracy: 0.8040
Epoch 2/5
47/47 [==============================] - 7s 158ms/step - loss: 0.4751 - accuracy: 0.7793 - val_loss: 0.4584 - val_accuracy: 0.7880
Epoch 3/5
47/47 [==============================] - 6s 122ms/step - loss: 0.4439 - accuracy: 0.8013 - val_loss: 0.3878 - val_accuracy: 0.8340
Epoch 4/5
47/47 [==============================] - 7s 149ms/step - loss: 0.4140 - accuracy: 0.8260 - val_loss: 0.3776 - val_accuracy: 0.8300
Epoch 5/5
47/47 [==============================] - 6s 118ms/step - loss: 0.3715 - accuracy: 0.8520 - val_loss: 0.3449 - val_accuracy: 0.8540
```

```
# Check out the layers in our model
model_1.summary()
```

A convolutional neural network (CNN) is set up and trained using TensorFlow and the Keras API to classify images into two categories: pizza and steak. Initially, the random seed is set for reproducibility. ImageDataGenerators are created for training and validation datasets to preprocess the images by scaling their pixel values between 0 and 1 (normalization). The generators load images from specified directories (train_dir and test_dir), converting them into batches of 32 images of size 224x224 pixels, and setting the class mode to 'binary' for binary classification.

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

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

 max_pooling2d (MaxPooling2  (None, 110, 110, 10)      0
 D)

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

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

 max_pooling2d_1 (MaxPoolin  (None, 53, 53, 10)        0
 g2D)

 flatten (Flatten)           (None, 28090)             0

 dense (Dense)               (None, 1)                 28091

=================================================================
Total params: 31101 (121.49 KB)
Trainable params: 31101 (121.49 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

It initializes and trains a simpler TensorFlow neural network model for binary image classification. The model, structured with dense layers and ‘relu’ activations, is compiled with binary cross-entropy loss and Adam optimizer, and then trained on the same pizza and steak image dataset for 5 epochs.

```
odel: "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

=================================================================
otal params: 602141 (2.30 MB)
rainable params: 602141 (2.30 MB)
on-trainable params: 0 (0.00 Byte)
```

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

# Create a model to replicate the TensorFlow Playground model
model_2 = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(224, 224, 3)), # dense layers expect a 1-
  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 c
                        validation_steps=len(valid_data))
```

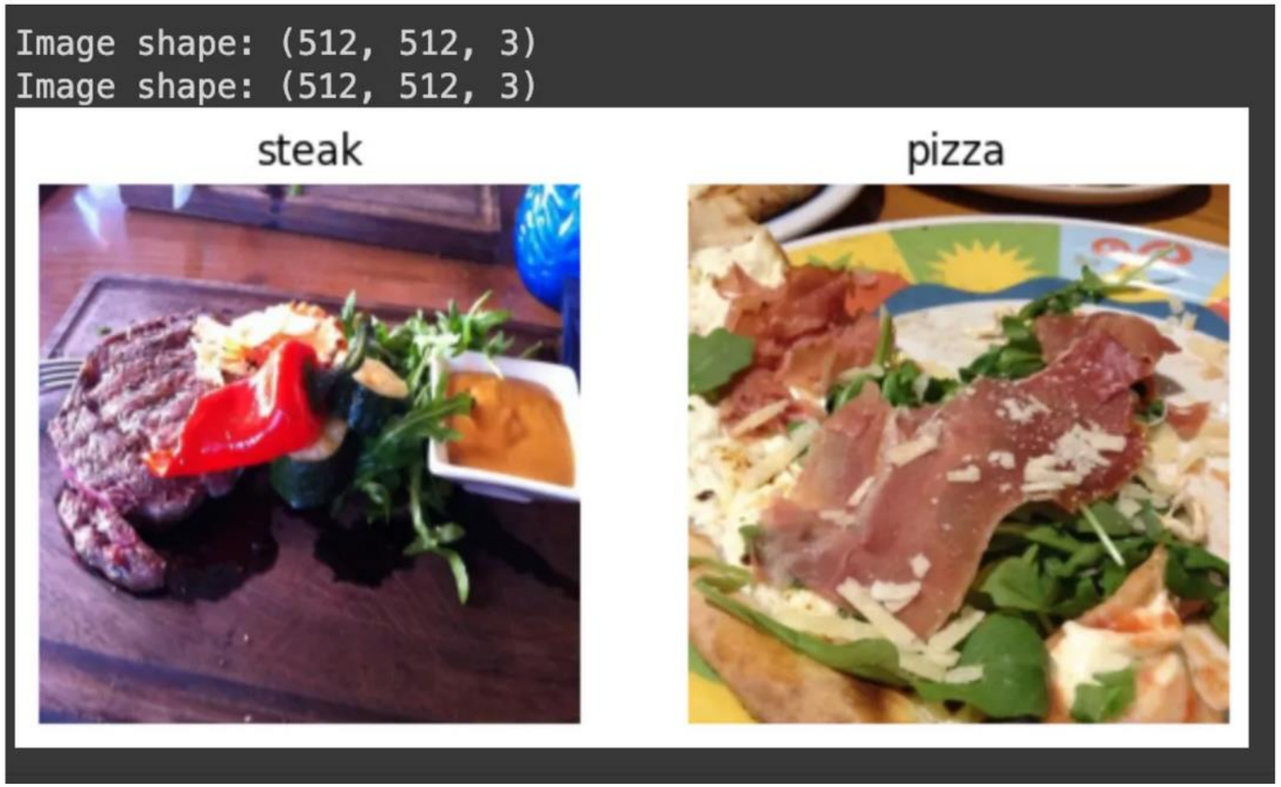
Output:

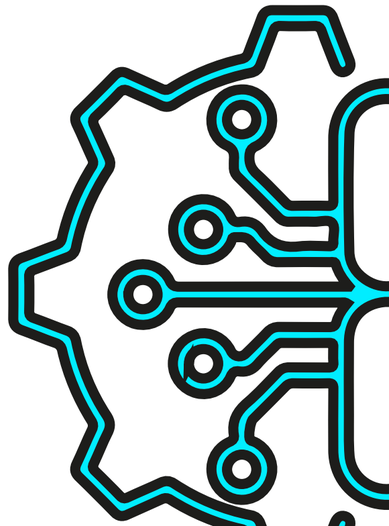
```
Epoch 1/5
47/47 [==============================] - 8s 110ms/step - loss: 0.8635 - accuracy
Epoch 2/5
47/47 [==============================] - 5s 106ms/step - loss: 0.6932 - accuracy
Epoch 3/5
47/47 [==============================] - 7s 140ms/step - loss: 0.6932 - accuracy
Epoch 4/5
47/47 [==============================] - 5s 107ms/step - loss: 0.6932 - accuracy
Epoch 5/5
47/47 [==============================] - 6s 137ms/step - loss: 0.6932 - accuracy
```

1.Import and become one with data

```
# Visualize data (requires function 'view_random_image' above)
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)
```





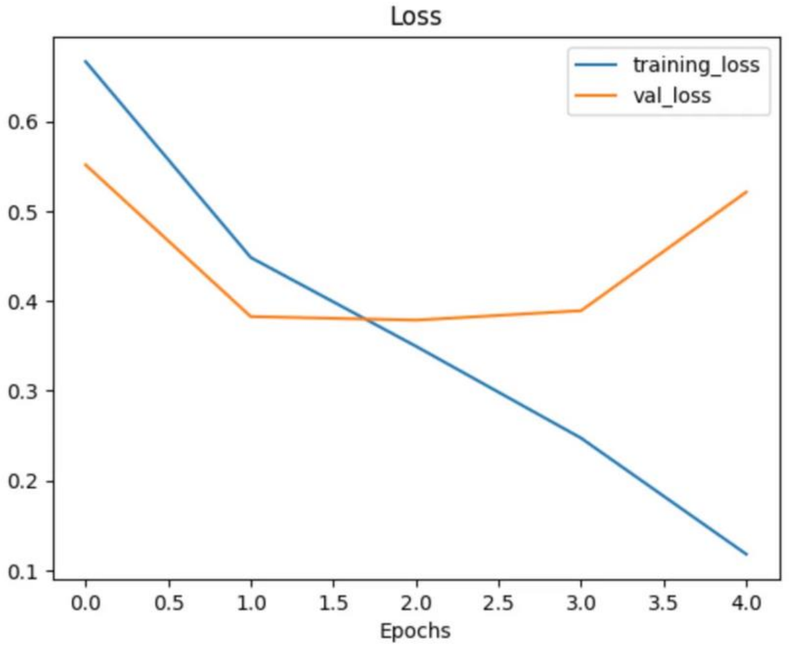
2. PREPROCESS THE DATA (PREPARING IT FOR A MODEL)

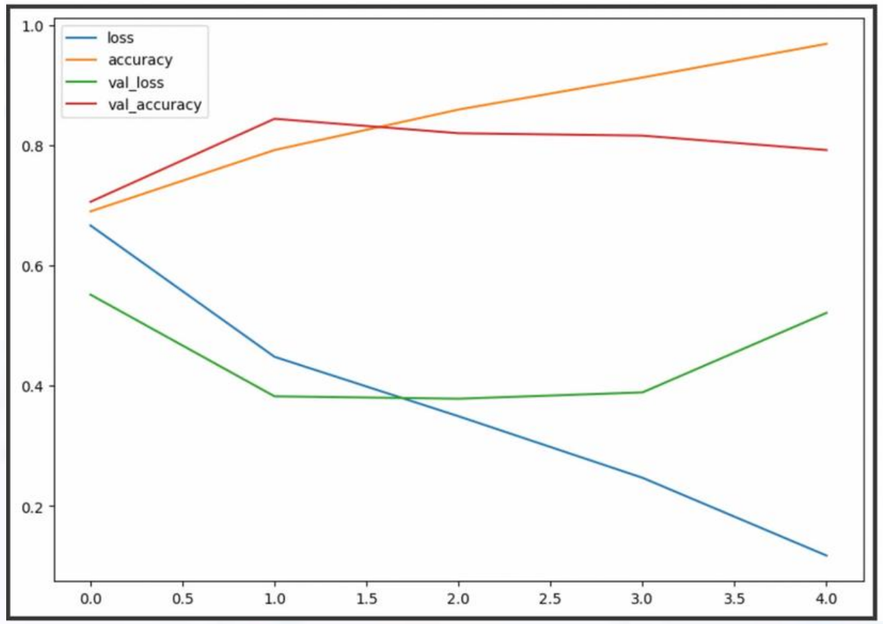
```
# Define training and test directory paths
train_dir = "pizza_steak/train/"
test_dir = "pizza_steak/test/"

# 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)
```

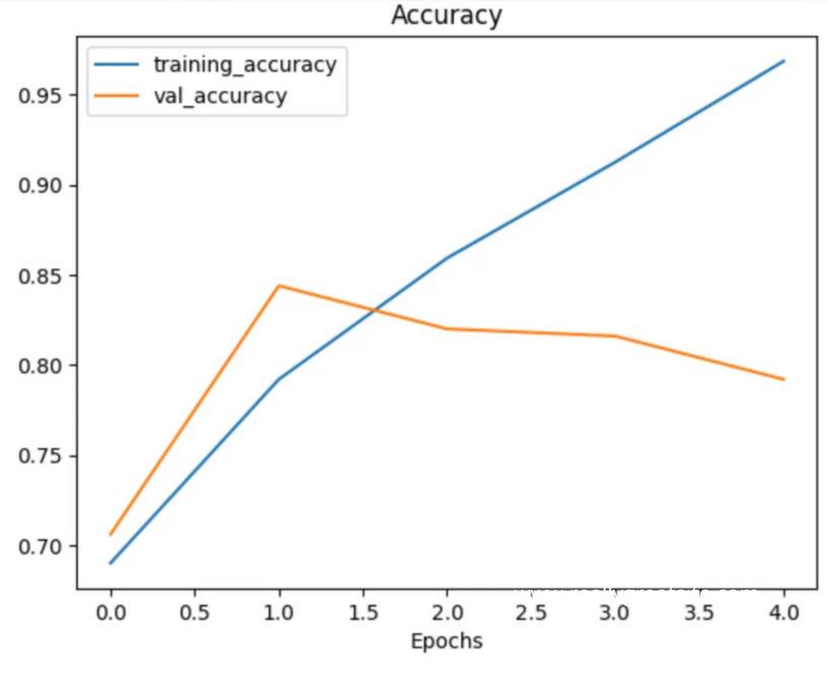
Output:

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

3. Creating a model (starting with a baseline)



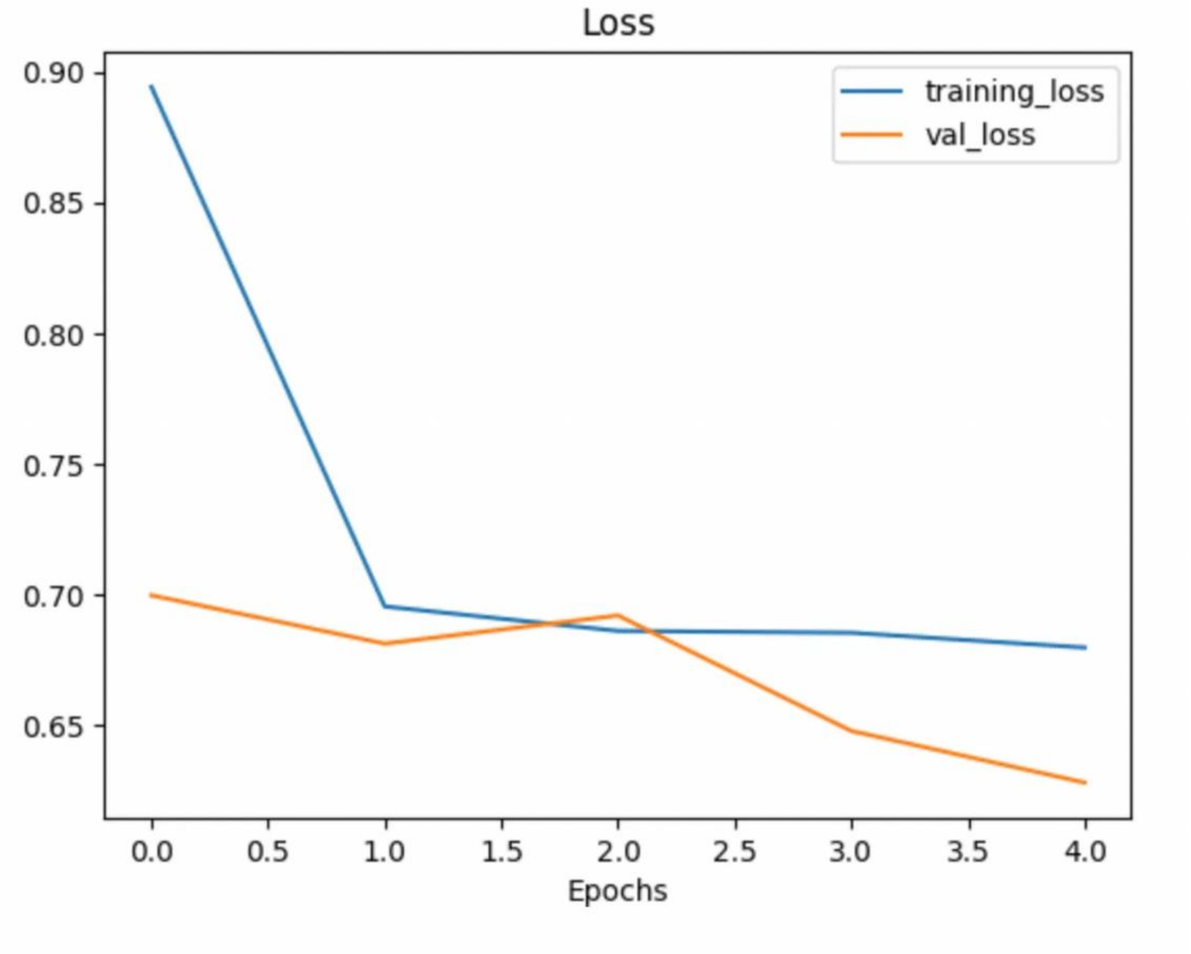


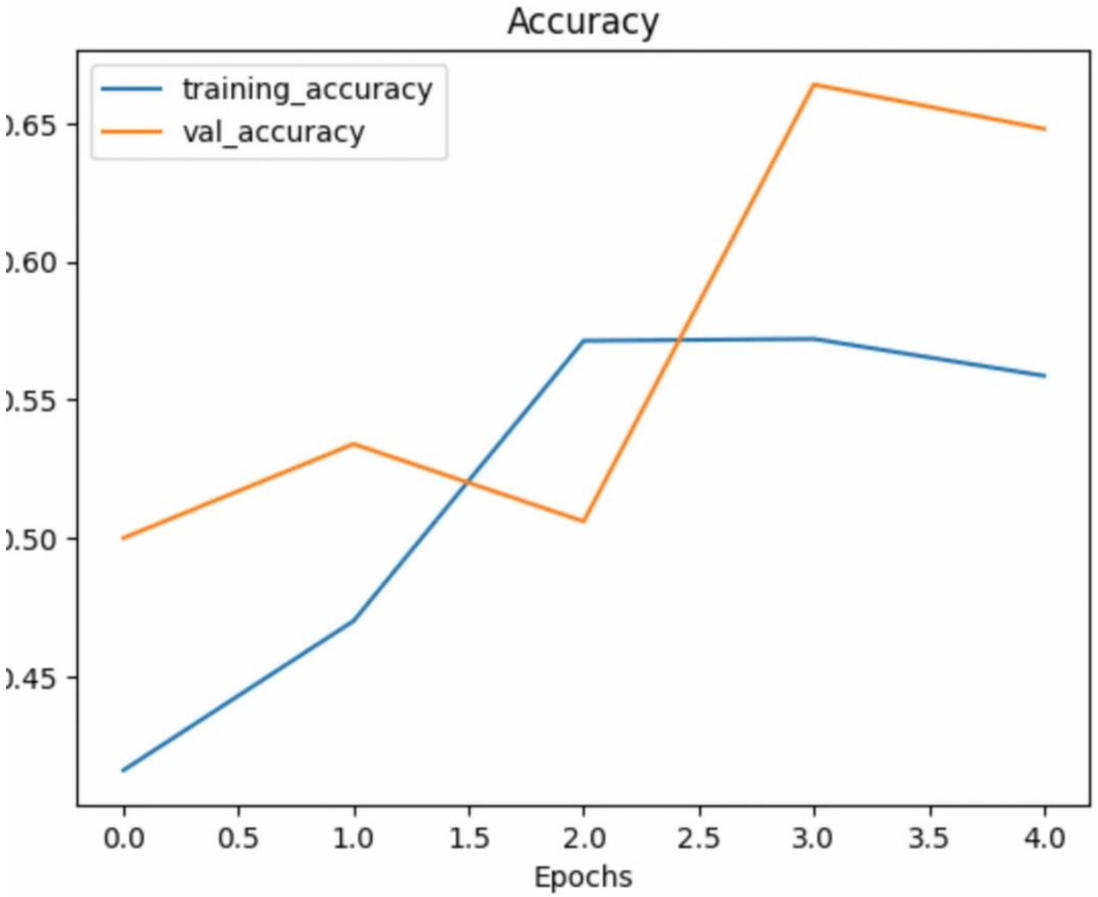


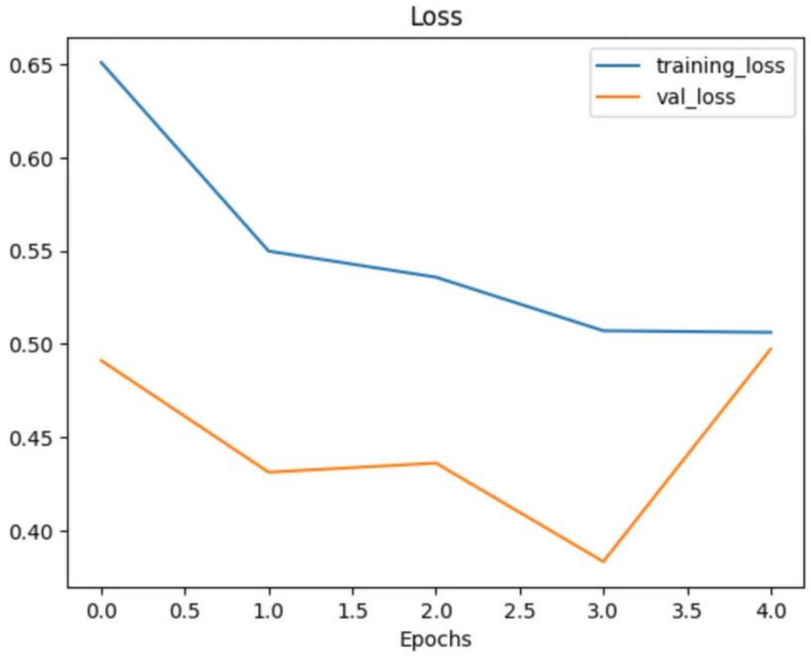
Original image

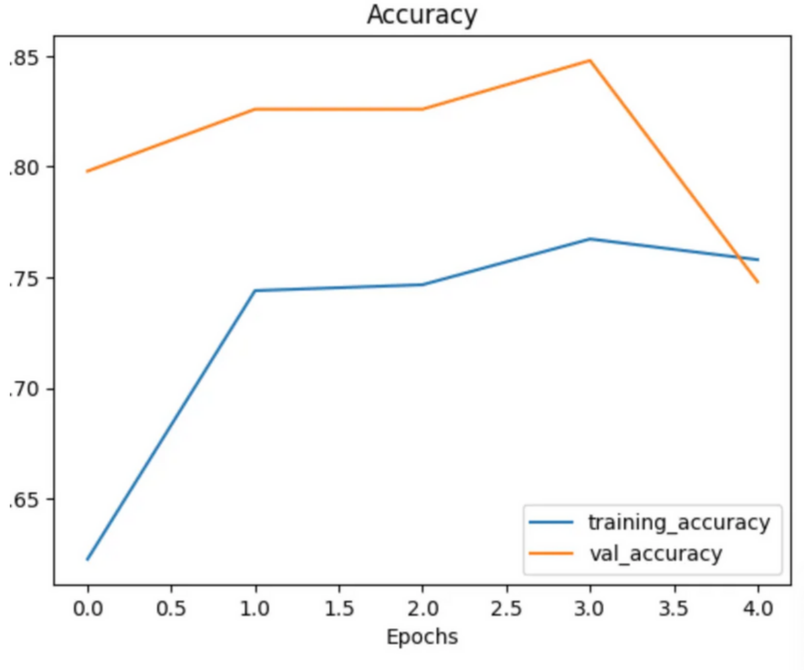
Augmented image

Shuffling Augmented Training Data: Shuffling the augmented training data can have a significant impact on the training process and the behavior of the loss curve.










```
# 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)

Output:
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 [==============================] - 23s 458ms/step - loss: 0.6510 - accuracy: 0.6227 - val_loss: 0.4911 - val_accuracy: 0.7980
Epoch 2/5
47/47 [==============================] - 23s 492ms/step - loss: 0.5498 - accuracy: 0.7440 - val_loss: 0.4312 - val_accuracy: 0.8260
Epoch 3/5
47/47 [==============================] - 22s 461ms/step - loss: 0.5358 - accuracy: 0.7467 - val_loss: 0.4361 - val_accuracy: 0.8260
Epoch 4/5
47/47 [==============================] - 23s 499ms/step - loss: 0.5071 - accuracy: 0.7673 - val_loss: 0.3832 - val_accuracy: 0.8480
Epoch 5/5
47/47 [==============================] - 27s 574ms/step - loss: 0.5062 - accuracy: 0.7580 - val_loss: 0.4974 - val_accuracy: 0.7480
# Check model's performance history training on augmented data
plot_loss_curves(history_7)
```

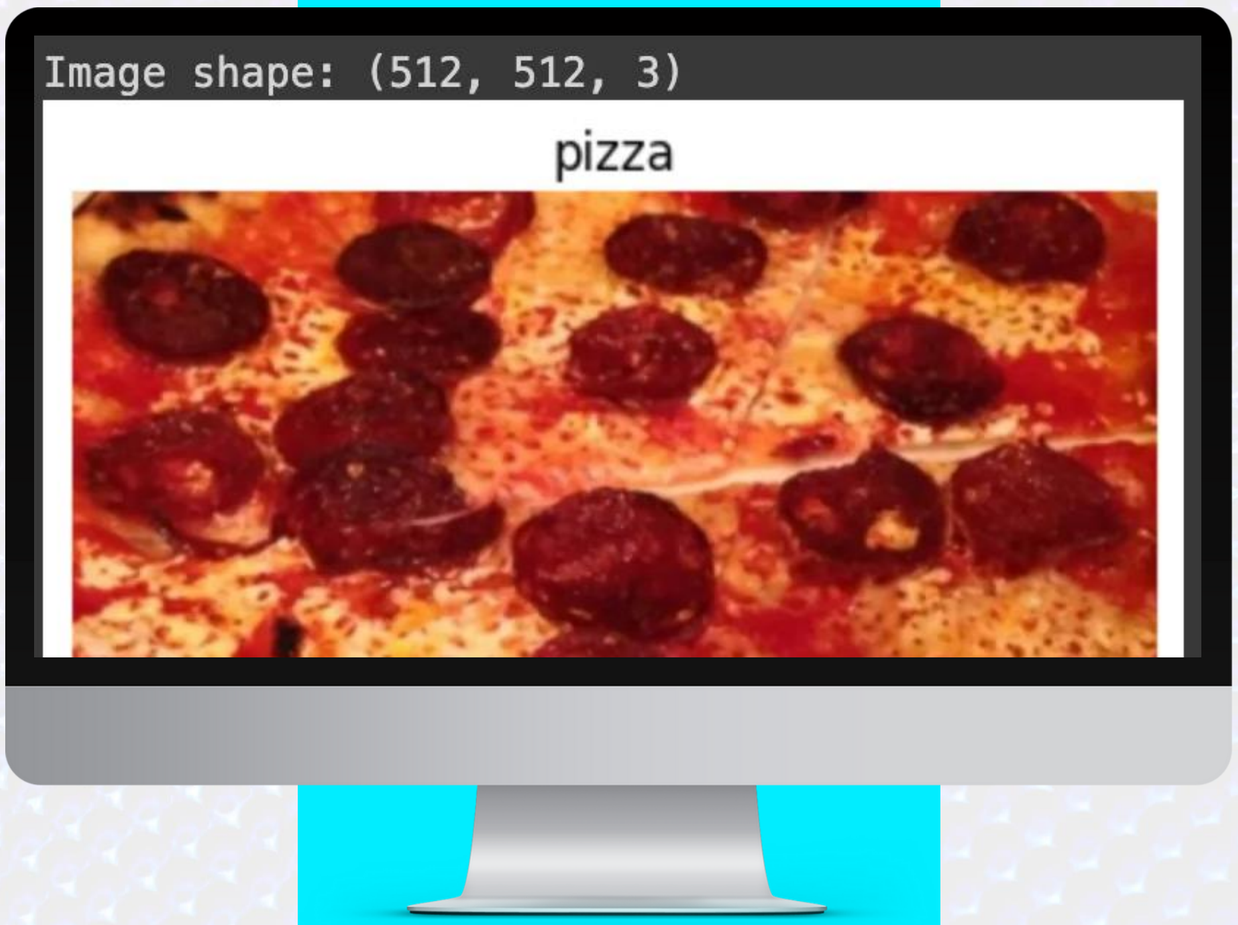
Making a prediction with our trained model

To test it out, we'll upload a couple of our own images and see how the model goes.

```
--2023-12-06 12:03:00--  https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-steak.jpeg
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.110.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
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 0.03s

2023-12-06 12:03:00 (68.6 MB/s) - '03-steak.jpeg' saved [1978213/1978213]
```

Since our model takes in images of shapes (224, 224, 3), we've got to reshape our custom image to use it with our model.

To do so, we can import and decode our image using tf.io.read_file (for readining files) and tf.image (for resizing our image and turning it into a tensor).



MULTICLASS CLASSIFICATION

1.Import and become one with the data

2.Preprocess the data (prepare it for a model)

3.Create a model (start with a baseline)

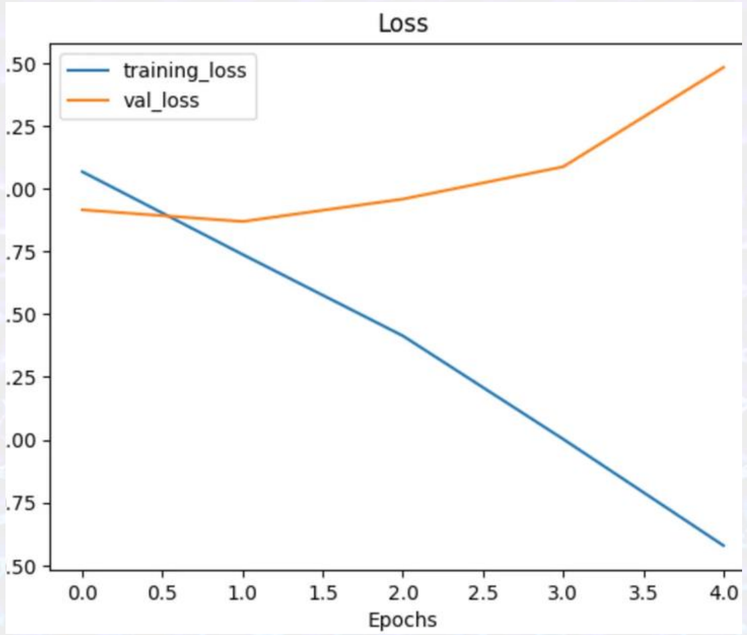
04 FIT A MODEL

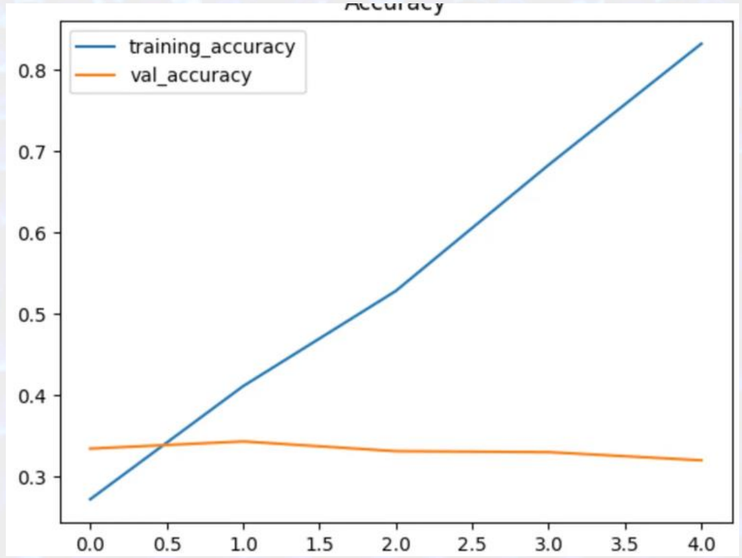
```
# Fit the model
history_9 = model_9.fit(train_data, # now 10 different classes
                        epochs=5,
                        steps_per_epoch=len(train_data),
                        validation_data=test_data,
                        validation_steps=len(test_data))

Epoch 1/5
235/235 [==============================] - 52s 220ms/step - loss: 0.1604 - accur
Epoch 2/5
235/235 [==============================] - 30s 128ms/step - loss: 0.0469 - accur
Epoch 3/5
235/235 [==============================] - 34s 143ms/step - loss: 0.0288 - accur
Epoch 4/5
235/235 [==============================] - 30s 126ms/step - loss: 0.0424 - accur
Epoch 5/5
235/235 [==============================] - 29s 124ms/step - loss: 0.0423 - accur
[ ]
```

Even with a simplifed model, it looks like our model is still dramatically overfitting the training data.Data augmentation?

Data augmentation makes it harder for the model to learn on the training data and in turn, hopefully making the patterns it learns more generalizable to unseen data.

05 EVALUATE THE MODEL





CONCLUSION

To conclude, our journey through the realms of CNNs and computer vision reveals a dynamic interplay of technology and practicality, where theoretical concepts transform into real-world applications. We have seen how CNNs, through their unique architecture and learning capabilities, adeptly handle complex visual data, paving the way for breakthroughs in various sectors.

The versatility and efficiency of CNNs in image recognition, object detection, and beyond signify a significant leap in how machines understand and interact with their visual environment. However, challenges such as overfitting and the need for extensive data highlight areas for future improvement and research. As we continue to advance in the field of AI and machine learning, CNNs stand as a testament to the profound impact of these technologies in shaping our visual and digital experiences.

THANK YOU