```
# Unzip the file
import zipfile
zip_ref1 = zipfile.ZipFile("/content/Test.zip", "r")
zip_ref2 = zipfile.ZipFile("/content/Train.zip", "r")
zip_ref3 = zipfile.ZipFile("/content/Validation.zip", "r")

zip_ref1.extractall()
zip_ref1.close()
zip_ref2.extractall()
zip_ref2.close()
zip_ref3.extractall()
zip_ref3.close()
```

```
train_dir = "/content/Train/"
test_dir = "/content/Test/"
val_dir = "/content/Validation/"
```

```
# Get the class names for our multi-class dataset
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)
```

```
['Healthy' 'Powdery' 'Rust']
```

```
# View an image
import os
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
import random
img = view_random_image(target_dir=train_dir,
                        target_class=random.choice(class_names))
```

```
Image shape: (3456, 5184, 3)
```


```
!pip show tensorflow
```

```
Name: tensorflow
Version: 2.18.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.11/dist-packages
Requires: absl-py, astunparse, flatbuffers, gast, google-pasta, grpcio, h5py, keras, libclang, ml-dtypes, numpy, opt-ei
Required-by: dopamine_rl, tensorflow-text, tf_keras
```

```
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')

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

```
Found 1322 images belonging to 3 classes.
Found 150 images belonging to 3 classes.
```

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

# Create our model
model = 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(),
  Conv2D(10, 3, activation='relu'),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Flatten(),
  Dense(3, activation='softmax')
])

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

```
history = model.fit(train_data,
                    epochs=5,
                    steps_per_epoch=len(train_data),
                    validation_data=test_data,
                    validation_steps=len(test_data))
```

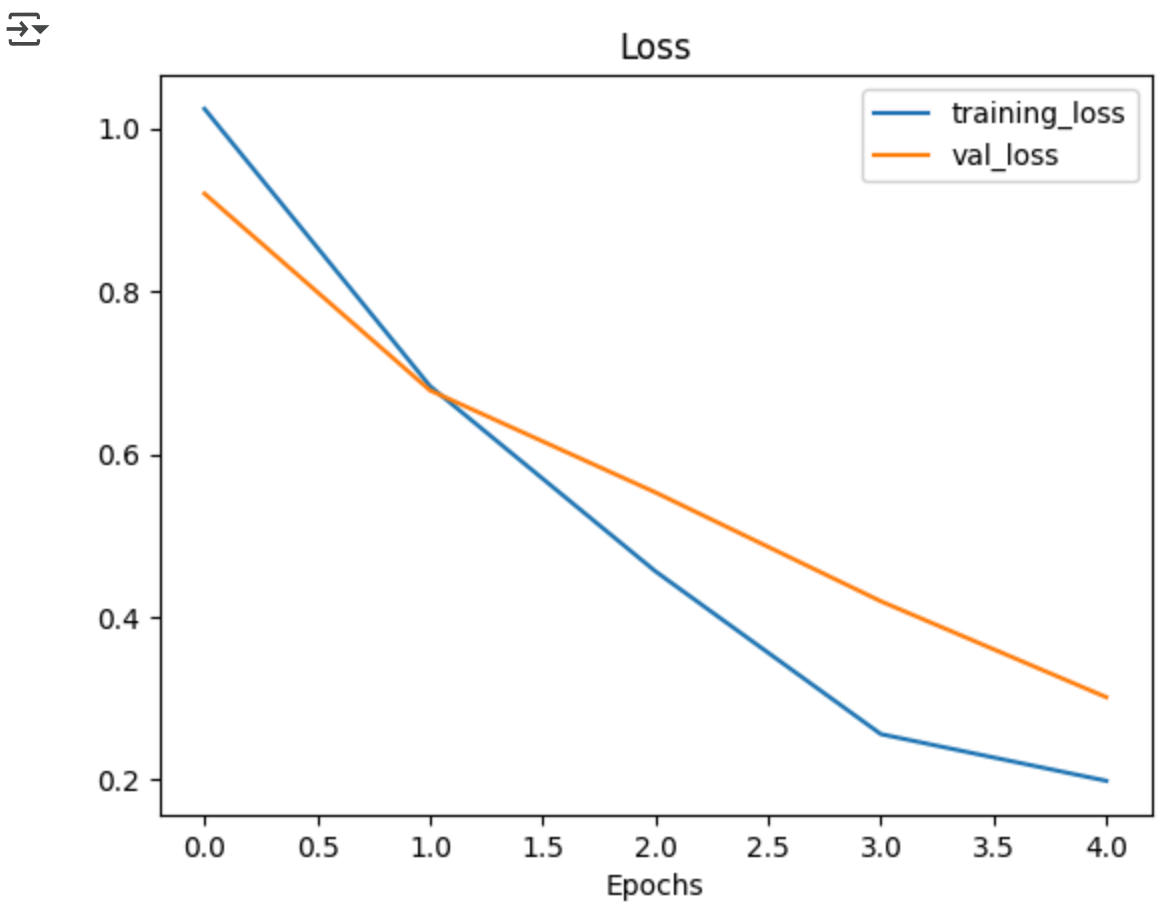
```
Epoch 1/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.3666 - loss: 1.0860 - val_accuracy: 0.5400 - val_loss: 0.9200
Epoch 2/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 134s 2s/step - accuracy: 0.6836 - loss: 0.7075 - val_accuracy: 0.6867 - val_loss: 0.6784
Epoch 3/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.8122 - loss: 0.4586 - val_accuracy: 0.8200 - val_loss: 0.5529
Epoch 4/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 78s 2s/step - accuracy: 0.9064 - loss: 0.2756 - val_accuracy: 0.8733 - val_loss: 0.4192
Epoch 5/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.9227 - loss: 0.2004 - val_accuracy: 0.9200 - val_loss: 0.3013
```

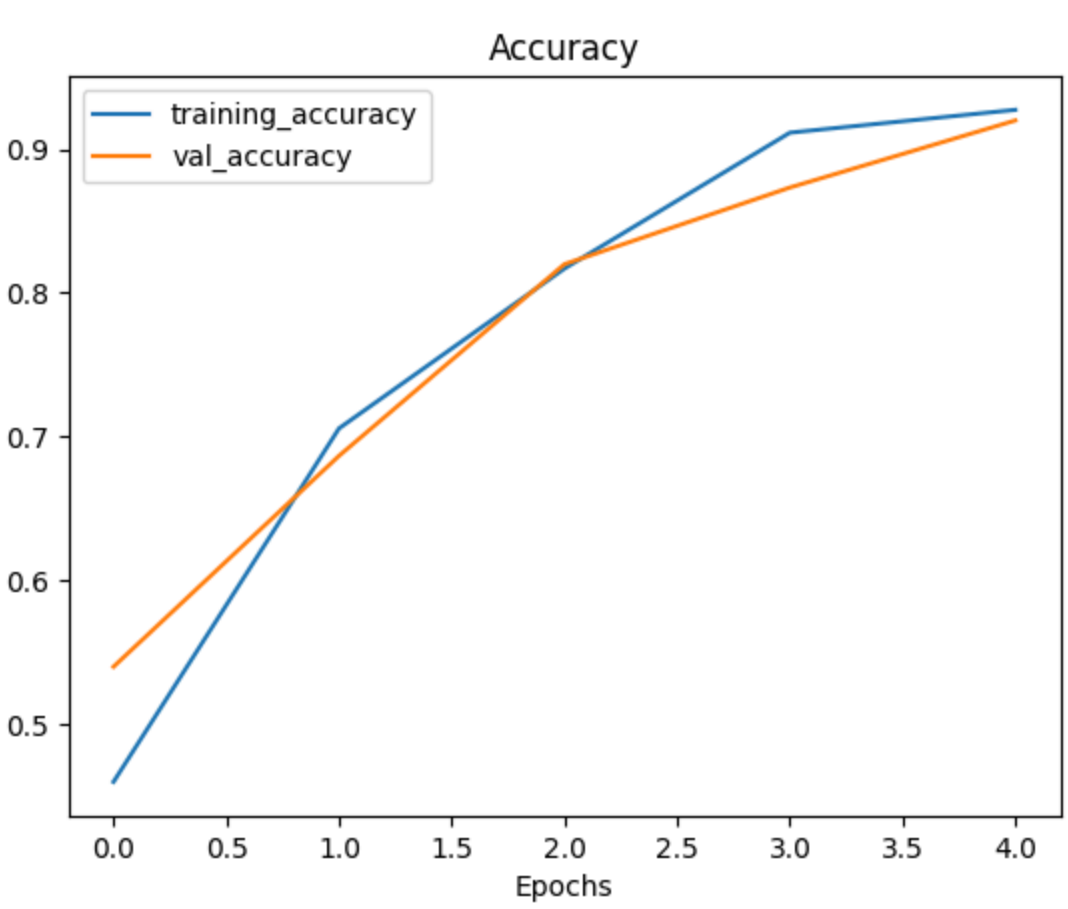
```
# Evaluate on the test data
model.evaluate(test_data)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 1s/step - accuracy: 0.9295 - loss: 0.3096
[0.30129313468933105, 0.9200000166893005]
```

```
# 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)
```





```
# Create a function to import an image and resize it to be able to be used with our model
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
```

```
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[tf.argmax(pred, axis=1).numpy()[0]]

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

```
import os
import random

def get_random_image(dir):
    """
    Randomly selects an image from the validation directory (val_dir).
    Args:
        val_dir (str): Path to the validation data directory containing subdirectories for each class.
    Returns:
        str: Path to the randomly selected image.
    """
    subdirs = os.listdir(dir)

    selected_class_dir = os.path.join(dir, random.choice(subdirs))

    image_files = os.listdir(selected_class_dir)

    selected_image = random.choice(image_files)

    image_path = os.path.join(selected_class_dir, selected_image)

    return image_path
```

```
# Make a prediction using model
pred_and_plot(model=model,
              filename=get_random_image(val_dir),
              class_names=class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```


```
# Make a prediction using model
pred_and_plot(model=model,
              filename=get_random_image(val_dir),
              class_names=class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
```


```
# Make a prediction using model
pred_and_plot(model=model,
              filename=get_random_image(val_dir),
              class_names=class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```

```
# Make a prediction using model
pred_and_plot(model=model,
              filename=get_random_image(val_dir),
              class_names=class_names)
```

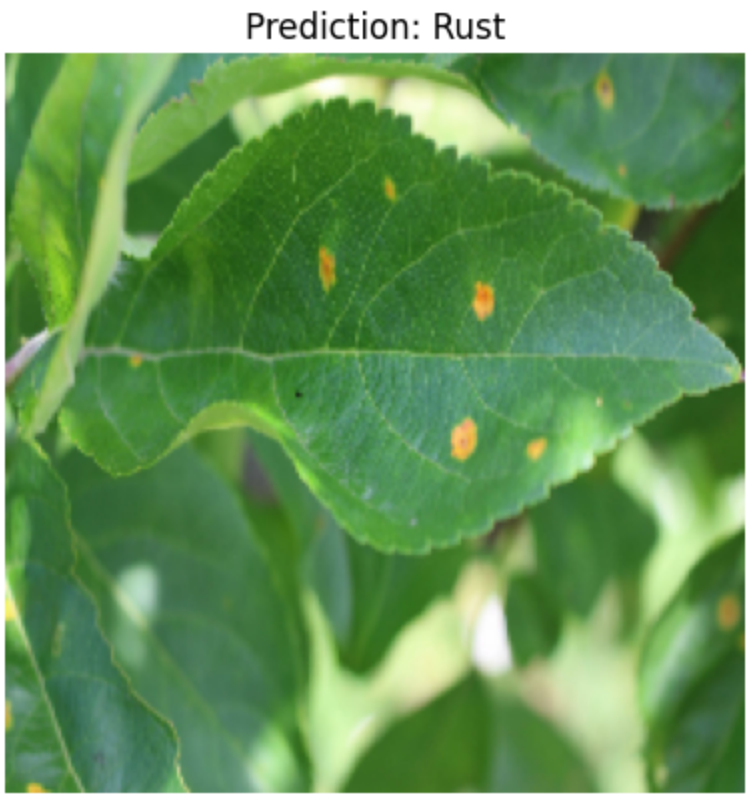
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
```


```
# Make a prediction using model
pred_and_plot(model=model,
              filename=get_random_image(val_dir),
              class_names=class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```


```
# Make a prediction using model
pred_and_plot(model=model,
              filename=get_random_image(val_dir),
              class_names=class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
```




```
# Make a prediction using model
pred_and_plot(model=model,
              filename=get_random_image(val_dir),
              class_names=class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
```


```
# Make a prediction using model
pred_and_plot(model=model,
              filename=get_random_image(val_dir),
              class_names=class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```

```
# Make a prediction using model
pred_and_plot(model=model,
              filename=get_random_image(val_dir),
              class_names=class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```


```
# Make a prediction using model
pred_and_plot(model=model,
              filename=get_random_image(val_dir),
              class_names=class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```