Corn or Maize Leaf Disease Classification

In this Notebook, We have to predict Corn or Maize Leaf Disease Classification. For This Purpose, We used Pretrained Models called Transfer Learning. Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.

Dataset we are Using https://www.kaggle.com/datasets/smaranjitghose/corn-or-maize-leaf-disease-dataset

We used two Different Pretrained Deep learning Models to get maximum accuracy and minumum loss

This Notebook is Designed in Google Colab and Using GPU Runtime Type

1-Import libraries

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2-Load Helper Function

```
!wget https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/extras/helper
```

```
--2022-08-15 08:15:04--  https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-lea
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:443
HTTP request sent, awaiting response... 200 OK
Length: 10246 (10K) [text/plain]
Saving to: ‘helper_functions.py’

helper_functions.py 100%[===================>]  10.01K  --.-KB/s    in 0s

2022-08-15 08:15:04 (112 MB/s) - ‘helper_functions.py’ saved [10246/10246]
```

3-Import Functions Using Helper Functions

```
# Import series of helper functions for our notebook
from helper_functions import create_tensorboard_callback, plot_loss_curves, unzip_data, compa
```

4-Install and Import Split Folders

```
pip install split-folders
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
```

```
import splitfolders
```

5-Mount Data From Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

6-Load Data from Google Drive

```
import zipfile

# unzip the downloaded file
zip_ref = zipfile.ZipFile("/content/drive/MyDrive/corn-or-maize-leaf-disease-dataset (2).zip"
zip_ref.extractall()
zip_ref.close()
```

7-Get Class Names Programmatically

```
# get the class names prgrammatically
import pathlib
import numpy as np
data_dir = pathlib.Path("/content/corn-or-maize-leaf-disease-dataset/balanced_dataset")
```

```
class_names = np.array(sorted([item.name for item in data_dir.glob("*")])) # create a list of
print(class_names)
```

```
['Blight' 'Common_Rust' 'Gray_Leaf_Spot' 'Healthy']
```

8-Visualize Our Images

```
# let's visualize our image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random
import os

def view_random_image(target_dir, target_class):
  # setup the 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) # this line means randomly sampl
  print(random_image)

  # 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 thw shape of the image

  return img
```

9-Split The Folders into Train , Validation and Test Datasets

```
splitfolders.ratio("/content/corn-or-maize-leaf-disease-dataset/balanced_dataset", output="co
    seed=1337, ratio=(.8, .1, .1), group_prefix=None, move=False)
```

```
Copying files: 2296 files [00:00, 4066.79 files/s]
```

10-Set Train, Test and Validation Paths

```
train_dir = "/content/corn-or-maize-dataset-split/train/"
test_dir = "/content/corn-or-maize-dataset-split/test/"
val_dir = '/content/corn-or-maize-dataset-split/val/'
```

11-Checking Number of Images

```
# How many images/classes are there?
walk_through_dir("/content/corn-or-maize-leaf-disease-dataset/balanced_dataset")
```

```
There are 4 directories and 0 images in '/content/corn-or-maize-leaf-disease-dataset/bal
There are 0 directories and 574 images in '/content/corn-or-maize-leaf-disease-dataset/b
There are 0 directories and 574 images in '/content/corn-or-maize-leaf-disease-dataset/b
There are 0 directories and 574 images in '/content/corn-or-maize-leaf-disease-dataset/b
There are 0 directories and 574 images in '/content/corn-or-maize-leaf-disease-dataset/b
```

12-Set Data Inputs

```
# Set up data inputs
import tensorflow as tf
IMG_SIZE = (224, 224)
train_data = tf.keras.preprocessing.image_dataset_from_directory(train_dir,
                                                                 label_mode="c
                                                                 image_size=IM
test_data =  tf.keras.preprocessing.image_dataset_from_directory(test_dir,
                                                                 label_mode="categorical",
                                                                 image_size=IMG_SIZE,
                                                                 shuffle=False) # don't shuff
```

```
Found 1836 files belonging to 4 classes.
Found 232 files belonging to 4 classes.
```

13-Create Checkpoint Callback

```
# Create a checkpoint callback
checkpoint_path = "corn-or-maize-disease-dataset-model"
checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                         save_weights_only=True,
                                                         monitor="val_accuracy",
                                                         save_best_only=True)
```

14-Make Data Augmentation

```
# Create a data augmentation layer to incorporate it right into the model
```

```
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.models import Sequential

# Setup data augmentation
data_augmentation = Sequential([
   preprocessing.RandomFlip("horizontal"),
   preprocessing.RandomRotation(0.2),
   preprocessing.RandomHeight(0.2),
   preprocessing.RandomWidth(0.2),
   preprocessing.RandomZoom(0.2),
  # preprocessing.Rescaling(1/255.)  # rescale inputs of images between 1 & 0, required for m
], name= "data_augmentation")
```

Model No 1 EfficientNet V2B3

15-Setup Base Model (Feature Extraction)

```
# Setup a base model and freeze its layer (this will extract features)
base_model = tf.keras.applications.EfficientNetV2B3(include_top=False)
base_model.trainable = False

# Setup a model architecture with trainable top layers
inputs = layers.Input(shape=(224, 224, 3), name="input_layer")
# x = data_augmentation(inputs) # augment layers (only happens during trainable phase)
x = base_model(inputs, training=False) # put the base model in interface mode so weights whic
x = layers.GlobalAveragePooling2D(name="global_avg_pooling_layer")(x)
outputs = layers.Dense(len(train_data.class_names), activation="softmax", name="output_layer"
model=tf.keras.Model(inputs, outputs)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/effic
52609024/52606240 [==============================] - 0s 0us/step
52617216/52606240 [==============================] - 0s 0us/step
```

```
# Get a summary of model we've been created
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 224, 224, 3)]     0

 efficientnetv2-b3 (Function  (None, None, None, 1536)  12930622
```

```
 al)

 global_avg_pooling_layer (G  (None, 1536)             0
 lobalAveragePooling2D)

 output_layer (Dense)        (None, 4)                 6148

=================================================================
Total params: 12,936,770
Trainable params: 6,148
Non-trainable params: 12,930,622
_________________________________________________________________
```

16-Import Metrics

```
from tensorflow.keras import metrics
```

```
METRICS = [
  metrics.TruePositives(name="tp"),
  metrics.FalsePositives(name="fp"),
  metrics.TrueNegatives(name="tn"),
  metrics.FalseNegatives(name="fn"),
  metrics.CategoricalAccuracy(name="accuracy"),
  metrics.Precision(name="precision"),
  metrics.Recall(name="recall"),
  metrics.AUC(name="auc")
]
```

17-Compile and Fit The Model

```
# Compile
model.compile(loss="categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(),
              metrics=METRICS)

# Fit
history_for_feature_extraction = model.fit(train_data,
                                           epochs=5, # fit to 5 epochs to keep experiment qui
                                           validation_data=test_data,
                                           validation_steps=int(0.15 * len(test_data)), # val
                                           callbacks=[checkpoint_callback])
```

```
Epoch 1/5
58/58 [==============================] - 41s 188ms/step - loss: 0.6697 - tp: 1061.0000 -
Epoch 2/5
58/58 [==============================] - 8s 139ms/step - loss: 0.3460 - tp: 1547.0000 -
Epoch 3/5
```

```
58/58 [==============================] - 7s 119ms/step - loss: 0.2795 - tp: 1624.0000 -
Epoch 4/5
58/58 [==============================] - 7s 120ms/step - loss: 0.2452 - tp: 1644.0000 -
Epoch 5/5
58/58 [==============================] - 7s 121ms/step - loss: 0.2195 - tp: 1677.0000 -
```

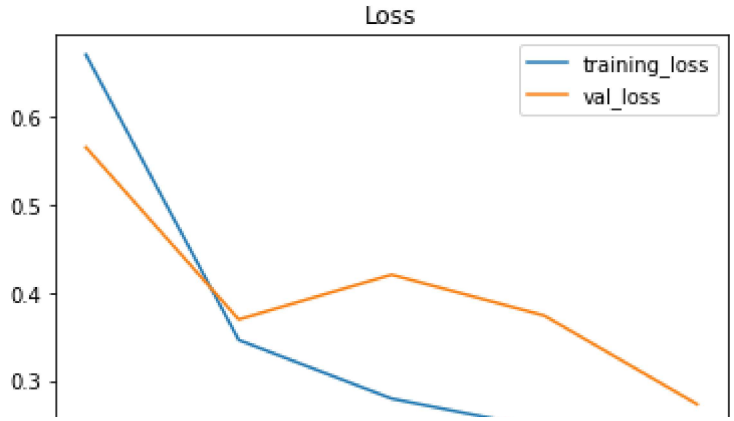
18-Evaluate The Model

```
# Evaluate on the whole test dataset  ##its a feature extraction instead of fine tune
feature_extraction_results = model.evaluate(test_data)
feature_extraction_results
```

```
8/8 [==============================] - 1s 178ms/step - loss: 0.1769 - tp: 220.0000 - fp
[0.17694979906082153,
 220.0,
 7.0,
 689.0,
 12.0,
 0.9568965435028076,
 0.9691630005836487,
 0.9482758641242981,
 0.9970459938049316]
```

19-Plot The Graph

```
plot_loss_curves(history_for_feature_extraction)
```



20-Unfreeze Some Layers

```
# Unfreeze all of the layers in base model
base_model.trainable=True

# Refreeze every layer except the last 5 layer
for layer in base_model.layers[:-5]:
  layer.trainable=False
```

21-Recompile and Fit The Model

```
# Recompile model with lower learning rate (it's typically best practice to lower the learnin
model.compile(loss="categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001), # learning rate lower
              metrics=METRICS)
```

```
# What layers in the model are trainable?
for layer in model.layers:
  print(layer.name, layer.trainable)
```

```
input_layer True
efficientnetv2-b3 True
global_avg_pooling_layer True
output_layer True
```

```
# Check which layers in our model is trainable
for layer_number, layer in enumerate(model.layers[1].layers):
  print(layer_number, layer.name, layer.trainable )
```

```
322 block6h_project_bn False
323 block6h_add False
324 block6i_expand_conv False
325 block6i_expand_bn False
326 block6i_expand_activation False
327 block6i_dwconv2 False
```

```
328 block6i_bn False
329 block6i_activation False
330 block6i_se_squeeze False
331 block6i_se_reshape False
332 block6i_se_reduce False
333 block6i_se_expand False
334 block6i_se_excite False
335 block6i_project_conv False
336 block6i_project_bn False
337 block6i_add False
338 block6j_expand_conv False
339 block6j_expand_bn False
340 block6j_expand_activation False
341 block6j_dwconv2 False
342 block6j_bn False
343 block6j_activation False
344 block6j_se_squeeze False
345 block6j_se_reshape False
346 block6j_se_reduce False
347 block6j_se_expand False
348 block6j_se_excite False
349 block6j_project_conv False
350 block6j_project_bn False
351 block6j_add False
352 block6k_expand_conv False
353 block6k_expand_bn False
354 block6k_expand_activation False
355 block6k_dwconv2 False
356 block6k_bn False
357 block6k_activation False
358 block6k_se_squeeze False
359 block6k_se_reshape False
360 block6k_se_reduce False
361 block6k_se_expand False
362 block6k_se_excite False
363 block6k_project_conv False
364 block6k_project_bn False
365 block6k_add False
366 block6l_expand_conv False
367 block6l_expand_bn False
368 block6l_expand_activation False
369 block6l_dwconv2 False
370 block6l_bn False
371 block6l_activation False
372 block6l_se_squeeze False
373 block6l_se_reshape False
374 block6l_se_reduce False
375 block6l_se_expand False
376 block6l_se_excite False
377 block6l_project_conv False
378 block6l_project_bn True
379 block6l_add True
380 top_conv True
```

```
# Fine-tune for more 5 epochs
fine_tune_epochs = 10 # model has already done the 5 epochs (feature extraction), this is the
```

```
# Fine-tune our model
history_for_fine_tune = model.fit(train_data,
                                                    epochs=fine_tune_epochs,
                                                    validation_data=test_data,
                                                    validation_steps=int(0.15 * len(test_dat
                                                    initial_epoch=history_for_feature_extrac
```

```
Epoch 5/10
58/58 [==============================] - 21s 182ms/step - loss: 0.1997 - tp: 1910.0000 -
Epoch 6/10
58/58 [==============================] - 8s 124ms/step - loss: 0.1706 - tp: 1711.0000 -
Epoch 7/10
58/58 [==============================] - 7s 123ms/step - loss: 0.1472 - tp: 1736.0000 -
Epoch 8/10
58/58 [==============================] - 7s 123ms/step - loss: 0.1317 - tp: 1751.0000 -
Epoch 9/10
58/58 [==============================] - 8s 124ms/step - loss: 0.1177 - tp: 1765.0000 -
Epoch 10/10
58/58 [==============================] - 8s 125ms/step - loss: 0.1071 - tp: 1769.0000 -
```

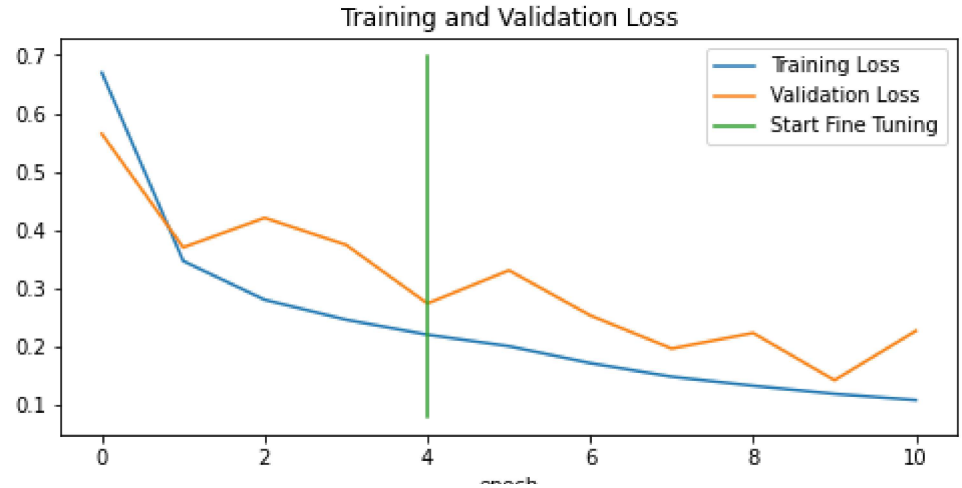
22-Re_Evaluate The Model

```
# Evaluate on the whole test dataset
fine_tune_results =model.evaluate(test_data)
fine_tune_results
```

```
8/8 [==============================] - 1s 118ms/step - loss: 0.1140 - tp: 224.0000 - fp
[0.11403942853212357,
 224.0,
 7.0,
 689.0,
 8.0,
 0.9655172228813171,
 0.9696969985961914,
 0.9655172228813171,
 0.9983804225921631]
```

23-Compare Histories

```
# Compare the historys of feature extraction model with fine-tuning model
compare_historys(original_history=history_for_feature_extraction,
                 new_history=history_for_fine_tune,
                 initial_epochs=5)
```



24-Save The Model

```
# Save our fine-tuning model
model.save("corn-or-maize-leaf-model.h5")
model.save("corn-or-maize-leaf-model.hdf5")
```

26-Make Prediction Probabilites

```
# Make predictions with model
preds_probs = model.predict(test_data, verbose=1) # set verbosity to see how long it left
```

```
8/8 [==============================] - 3s 128ms/step
```

```
len(test_data)
```

```
8
```

```
# How many predictions are there?
len(preds_probs)
```

```
232
```

```
# What's the shape of our predictions?
preds_probs.shape
```

```
(232, 4)
```

```
# Let's see whats the first 10 predictions looks like
preds_probs[:10]
```

```
array([[6.8684304e-01, 6.0789060e-02, 2.5087541e-01, 1.4924821e-03],
       [9.3977726e-01, 1.6788630e-03, 5.8524404e-02, 1.9446699e-05],
       [9.0393925e-01, 9.8118670e-03, 8.1733346e-02, 4.5155152e-03],
       [9.8427355e-01, 8.9026109e-04, 1.4813614e-02, 2.2608190e-05],
       [9.3496406e-01, 5.9055117e-05, 6.4963147e-02, 1.3761132e-05],
       [7.2537094e-01, 1.9993656e-03, 2.7257553e-01, 5.4135166e-05],
       [9.7174406e-01, 8.3288336e-07, 2.8254962e-02, 9.4631076e-08],
       [7.8735894e-01, 3.9426163e-03, 2.0842887e-01, 2.6959908e-04],
       [7.7056450e-01, 1.1389566e-03, 2.2827837e-01, 1.8195495e-05],
       [6.9635439e-01, 4.4325409e-03, 2.9918507e-01, 2.8025222e-05]],
      dtype=float32)
```

```
# What does the first prediction probability array look like?
preds_probs[0], len(preds_probs[0]), sum(preds_probs[0])
```

```
(array([0.68684304, 0.06078906, 0.2508754 , 0.00149248], dtype=float32),
 4,
 0.9999999930150807)
```

```
# We get one prediction probability per class(in our case there's 101 prediction probabilitie
print(f"Number of prediction probabilites for sample 0: {len(preds_probs[0])}")
print(f"What prediction probabilites sample 0 looks like:\n {preds_probs[0]}")
print(f"The class with highest predicted probability by the model for sample 0: {preds_probs[
```

```
Number of prediction probabilites for sample 0: 4
What prediction probabilites sample 0 looks like:
 [0.68684304 0.06078906 0.2508754  0.00149248]
The class with highest predicted probability by the model for sample 0: 0
```

```
# Get the pred classes of each model
pred_classes = preds_probs.argmax(axis=1)

# How do they look like?
pred_classes[:10]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
# How many pred classes we have?
```

```
len(pred_classes)
```

```
232
```

```
# To get our test dataset labels we need to unravel our test_data BatchDataset
y_labels = []
for images, labels in test_data.unbatch():
  y_labels.append(labels.numpy().argmax()) # currently test labels look like: [0, 0, 0, 1,...
y_labels[:10] # look at the first 10
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
# How many y_labels are there?
len(y_labels)
```

```
232
```

```
test_data
```

```
<BatchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32,
name=None), TensorSpec(shape=(None, 4), dtype=tf.float32, name=None))>
```

```
len(test_data)
```

```
8
```

27-Make Confusion Matrix

```
from helper_functions import make_confusion_matrix
```

```
# Get a list of class names
class_names =test_data.class_names
class_names[:10]
```

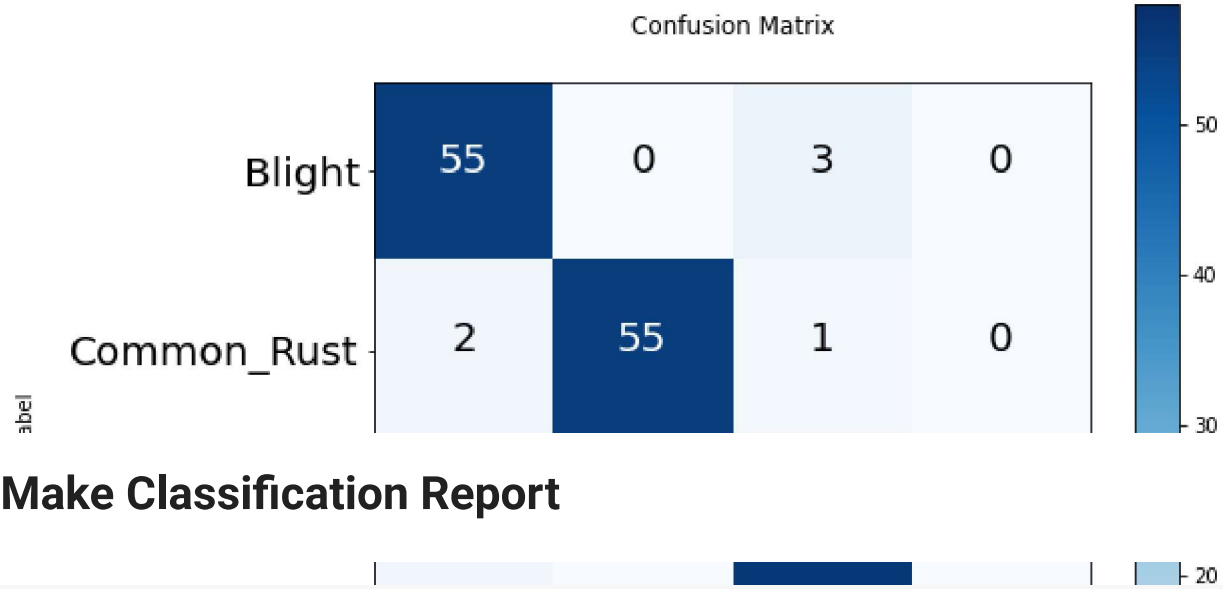
```
['Blight', 'Common_Rust', 'Gray_Leaf_Spot', 'Healthy']
```

```
import itertools
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix

# We need to make some changes to our make_confusion_matrix function to ensure the x-label pr
def make_confusion_matrix(y_true, y_pred, classes=None, figsize=(10, 10), text_size=15, norm=
  """Makes a labelled confusion matrix comparing predictions and ground truth labels.
```

```
  If classes is passed, confusion matrix will be labelled, if not, integer class values
  will be used.

  Args:
    y_true: Array of truth labels (must be same shape as y_pred).
    y_pred: Array of predicted labels (must be same shape as y_true).
    classes: Array of class labels (e.g. string form). If `None`, integer labels are used.
    figsize: Size of output figure (default=(10, 10)).
    text_size: Size of output figure text (default=15).
    norm: normalize values or not (default=False).
    savefig: save confusion matrix to file (default=False).

  Returns:
    A labelled confusion matrix plot comparing y_true and y_pred.

  Example usage:
    make_confusion_matrix(y_true=test_labels, # ground truth test labels
                          y_pred=y_preds, # predicted labels
                          classes=class_names, # array of class label names
                          figsize=(15, 15),
                          text_size=10)
  """
  # Create the confustion matrix
  cm = confusion_matrix(y_true, y_pred)
  cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis] # normalize it
  n_classes = cm.shape[0] # find the number of classes we're dealing with

  # Plot the figure and make it pretty
  fig, ax = plt.subplots(figsize=figsize)
  cax = ax.matshow(cm, cmap=plt.cm.Blues) # colors will represent how 'correct' a class is, d
  fig.colorbar(cax)

  # Are there a list of classes?
  if classes:
    labels = classes
  else:
    labels = np.arange(cm.shape[0])

  # Label the axes
  ax.set(title="Confusion Matrix",
         xlabel="Predicted label",
         ylabel="True label",
         xticks=np.arange(n_classes), # create enough axis slots for each class
         yticks=np.arange(n_classes),
         xticklabels=labels, # axes will labeled with class names (if they exist) or ints
         yticklabels=labels)

  # Make x-axis labels appear on bottom
  ax.xaxis.set_label_position("bottom")
  ax.xaxis.tick_bottom()
```

```
  ### changes (x-labels vertically) ###
  plt.xticks(rotation=70, fontsize=text_size)
  plt.yticks(fontsize=text_size)

  # Set the threshold for different colors
  threshold = (cm.max() + cm.min()) / 2.

  # Plot the text on each cell
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    if norm:
      plt.text(j, i, f"{cm[i, j]} ({cm_norm[i, j]*100:.1f}%)",
              horizontalalignment="center",
              color="white" if cm[i, j] > threshold else "black",
              size=text_size)
    else:
      plt.text(j, i, f"{cm[i, j]}",
              horizontalalignment="center",
              color="white" if cm[i, j] > threshold else "black",
              size=text_size)

  # Save the figure to the current working directory
  if savefig:
    fig.savefig("confusion_matrix.png")
```

```
make_confusion_matrix(y_true=y_labels,
                      y_pred=pred_classes,
                      classes=class_names,
                     figsize=(8, 8),
                     text_size=20,
                     savefig=True)
```



28-Make Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(y_true=y_labels,
                            y_pred=pred_classes))
```

```
                  precision    recall  f1-score   support

               0       0.93      0.95      0.94        58
               1       1.00      0.95      0.97        58
               2       0.93      0.97      0.95        58
               3       1.00      1.00      1.00        58

        accuracy                           0.97       232
       macro avg       0.97      0.97      0.97       232
    weighted avg       0.97      0.97      0.97       232
```

```
# Get a dictionary of the classification report
classification_report_dict = classification_report(y_labels, pred_classes, output_dict=True)
classification_report_dict
```


```
{'0': {'f1-score': 0.94017094017094,
  'precision': 0.9322033898305084,
  'recall': 0.9482758620689655,
  'support': 58},
 '1': {'f1-score': 0.9734513274336283,
  'precision': 1.0,
  'recall': 0.9482758620689655,
  'support': 58},
 '2': {'f1-score': 0.9491525423728815,
  'precision': 0.9333333333333333,
  'recall': 0.9655172413793104,
  'support': 58},
 '3': {'f1-score': 1.0, 'precision': 1.0, 'recall': 1.0, 'support': 58},
 'accuracy': 0.9655172413793104,
 'macro avg': {'f1-score': 0.9656937024943625,
  'precision': 0.9663841807909604,
```

```
         'recall': 0.9655172413793104,
         'support': 232},
        'weighted avg': {'f1-score': 0.9656937024943624,
         'precision': 0.9663841807909604,
         'recall': 0.9655172413793104,
         'support': 232}}
```

```
class_names[3]
```

```
'Healthy'
```

```
classification_report_dict["3"]["f1-score"]
```

```
1.0
```

```
# Create empty dictionary
class_f1_scores = {}
# Loop through classification report dictionary items
for k, v in classification_report_dict.items():
  if k == "accuracy": # stop once we get to accuracy key
    break
  else:
     # Add names and f1-scores to new dictionary
     class_f1_scores[class_names[int(k)]] = v["f1-score"]
class_f1_scores
```

```
{'Blight': 0.94017094017094,
 'Common_Rust': 0.9734513274336283,
 'Gray_Leaf_Spot': 0.9491525423728815,
 'Healthy': 1.0}
```

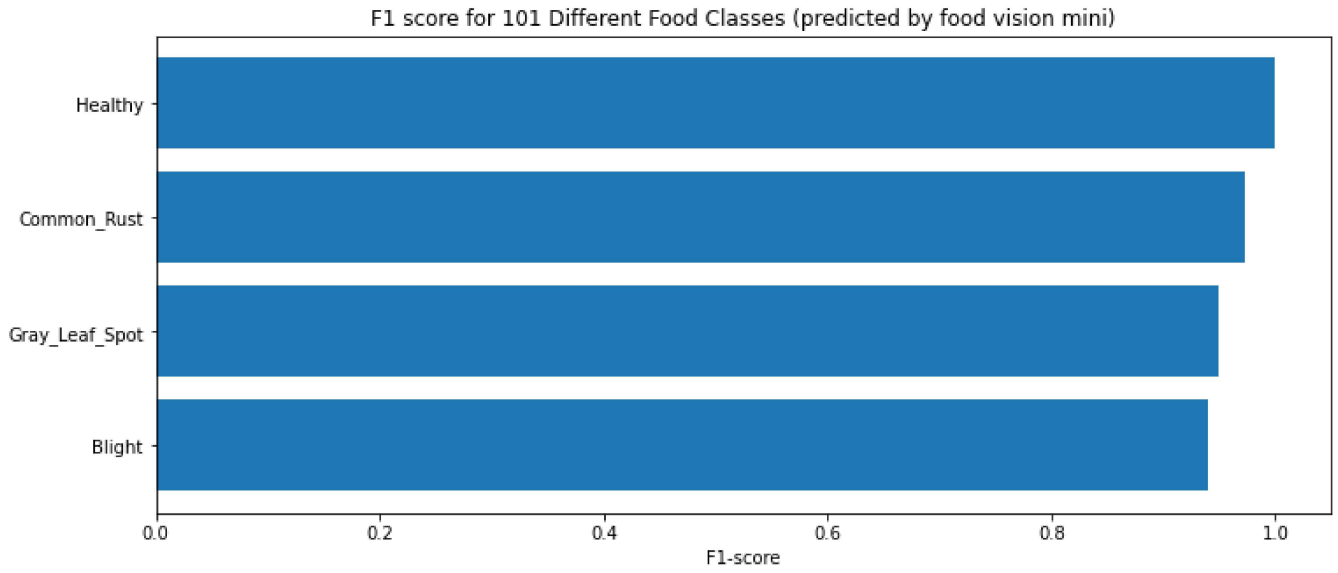
```
# Turn f1 scores into Dataframe visualization
import pandas as pd
f1_scores =pd.DataFrame({"class_names": list(class_f1_scores.keys()),
                         "f1-score": list(class_f1_scores.values())}).sort_values("f1-score",
f1_scores
```

	class_names	f1-score
3	Healthy	1.000000
1	Common_Rust	0.973451
2	Gray_Leaf_Spot	0.949153
0	Blight	0.940171

```
f1_scores[:2]
```

	class_names	f1-score
3	Healthy	1.000000
1	Common_Rust	0.973451

```
import matplotlib.pyplot as plt


import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(12, 5))
scores = ax.barh(range(len(f1_scores)), f1_scores["f1-score"].values) # get f1-score value
ax.set_yticks(range(len(f1_scores)))
ax.set_yticklabels(f1_scores["class_names"])
ax.set_xlabel("F1-score")
ax.set_title("F1 score for 101 Different Food Classes (predicted by food vision mini)")
ax.invert_yaxis(); # reverse the order of our plot


# Challenge: add value to the end of each bar of what the actual f1-score is
# (hint: use the "autolabel" function from here: https://matplotlib.org/2.0.2/examples/api/ba
```



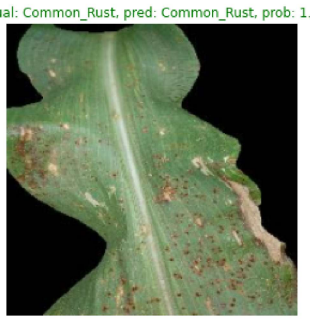
29-Load and Prepare Images Function

```
# Create a function to load and create a images
def load_and_prep_image(filename, img_shape=224, scale=True):
  """
  Read in an image from filename, turn it into a tensor and reshapes into
  specified shape (img_shape, img_shape, color_channels=3).
```

```
  Args:
    filename(str): path to target image
    image_shape(int): height/width dimension of target image size
    scale(bool): scale pixel values from 0-255 to 0-1 or not

  Returns:
    image tensor of shape (img_shape, img_shape, 3)
  """

  # Read in the image
  img = tf.io.read_file(filename)

  # Decode image into tensor
  img = tf.io.decode_image(img, channels=3)

  # Resize the image
  img = tf.image.resize(img, [img_shape, img_shape])

  # Scale Yes/no?
  if scale:
    # reshape the image (get all value between 0 and 1)
    return img/255.
  else:
    return img # don't need to rescale image for EffiecienNet model in TensorFlow
```

```
# Make preds on series of images
import os
import random

plt.figure(figsize=(24,5))
for i in range(3):
  # Choose a random image(s) from a random class(es)
  class_name =random.choice(class_names)
  filename = random.choice(os.listdir(val_dir + "/" + class_name))
  filepath = val_dir + class_name + "/" + filename


  # Load the image and make predictions
  img = load_and_prep_image(filepath, scale=False)
  # print(img.shape)
  img_expanded = tf.expand_dims(img, axis=0)
  # print(img_expanded.shape)
  pred_prob = model.predict(img_expanded) # get prediction probability array
  pred_class = class_names[pred_prob.argmax()] # get heighst prediction probability index and
  # print(pred_prob)
   # print(pred_class)

    # Plot the image(s)
```

```
  plt.subplot(1, 3, i+1)
  plt.imshow(img/255.)
  if class_name == pred_class: # if predicted class matches truth class, make text green
    title_color = "g"
  else:
    title_color ="r"
  plt.title(f"actual: {class_name}, pred: {pred_class}, prob: {pred_prob.argmax():.2f}", c=ti
  plt.axis(False);
```




```
images = ['/content/common-rust.jpg', '/content/grey leaf spot.jpg', '/content/blight.jpg', '
```

```
# Make prediction on and plot the custom food images
for img in images:
  img = load_and_prep_image(img, scale=False) # don't need to do scale for our EfficienNetB0
  pred_prob = model.predict(tf.expand_dims(img, axis=0)) # make prediction on the image with
  pred_class = class_names[pred_prob.argmax()] # get the index with heighest prediction proba
  # plot the appropriate information
  plt.figure()
  plt.imshow(img/255.)
  plt.title(f"pred: {pred_class},   prob: {pred_prob.max():.2f}")
  plt.axis(False)
```

pred: Common_Rust, prob: 0.98

pred: Gray_Leaf_Spot, prob: 0.72

pred: Blight, prob: 0.69

pred: Healthy, prob: 0.92

Model For EfficientNetB4

```
# Setup a base model and freeze its layer (this will extract features)
base_model = tf.keras.applications.EfficientNetB4(include_top=False)
base_model.trainable = False

# Setup a model architecture with trainable top layers
```

```
inputs = layers.Input(shape=(224, 224, 3), name="input_layer")
# x = data_augmentation(inputs) # augment layers (only happens during trainable phase)
x = base_model(inputs, training=False) # put the base model in interface mode so weights whic
x = layers.GlobalAveragePooling2D(name="global_avg_pooling_layer")(x)
outputs = layers.Dense(len(train_data.class_names), activation="softmax", name="output_layer"
model=tf.keras.Model(inputs, outputs)
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb4_r
71688192/71686520 [==============================] - 0s 0us/step
71696384/71686520 [==============================] - 0s 0us/step
```

```
from tensorflow.keras import metrics
```

```
METRICS = [
  metrics.TruePositives(name="tp"),
  metrics.FalsePositives(name="fp"),
  metrics.TrueNegatives(name="tn"),
  metrics.FalseNegatives(name="fn"),
  metrics.CategoricalAccuracy(name="accuracy"),
  metrics.Precision(name="precision"),
  metrics.Recall(name="recall"),
  metrics.AUC(name="auc")
]
```

```
# Compile
model.compile(loss="categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(),
              metrics=METRICS)
```

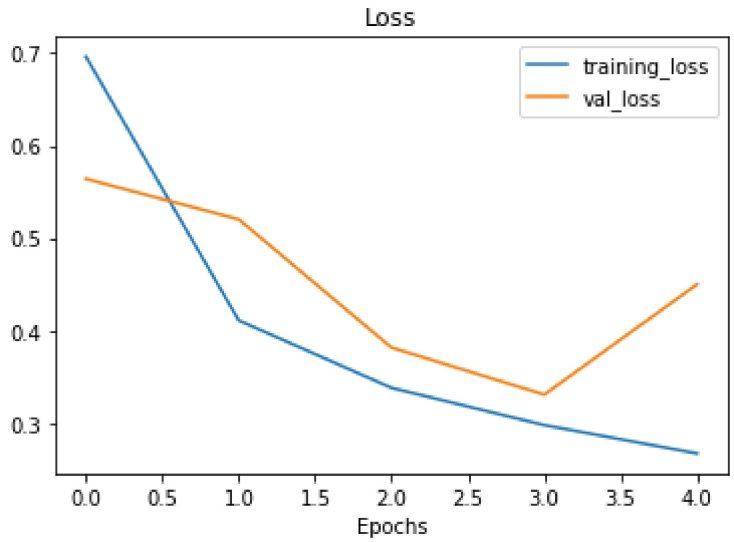
```
# Fit
history_for_feature_extraction = model.fit(train_data,
                                           epochs=5, # fit to 5 epochs to keep experiment qui
                                           validation_data=test_data,
                                           validation_steps=int(0.15 * len(test_data)), # val
                                           callbacks=[checkpoint_callback])
```

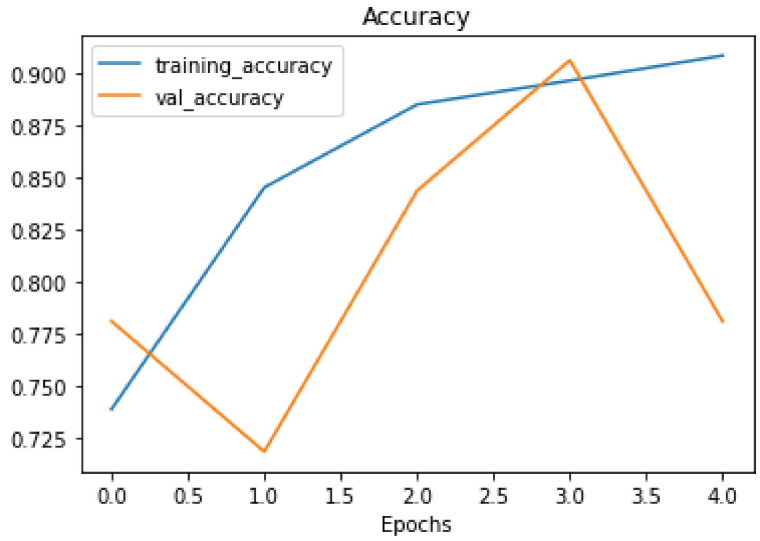
```
Epoch 1/5
58/58 [==============================] - 27s 242ms/step - loss: 0.6962 - tp: 1033.0000 -
Epoch 2/5
58/58 [==============================] - 11s 187ms/step - loss: 0.4110 - tp: 1463.0000 -
Epoch 3/5
58/58 [==============================] - 11s 177ms/step - loss: 0.3381 - tp: 1554.0000 -
Epoch 4/5
58/58 [==============================] - 11s 178ms/step - loss: 0.2978 - tp: 1589.0000 -
Epoch 5/5
58/58 [==============================] - 11s 177ms/step - loss: 0.2672 - tp: 1631.0000 -
```

```
# Evaluate on the whole test dataset  ##its a feature extraction instead of fine tune
feature_extraction_results = model.evaluate(test_data)
feature_extraction_results
```

```
8/8 [==============================] - 2s 262ms/step - loss: 0.2740 - tp: 204.0000 - fp
[0.2739827036857605,
 204.0,
 17.0,
 679.0,
 28.0,
 0.9051724076271057,
 0.9230769276618958,
 0.8793103694915771,
 0.9897661209106445]
```

```
plot_loss_curves(history_for_feature_extraction)
```





```
# Unfreeze all of the layers in base model
base_model.trainable=True

# Refreeze every layer except the last 5 layer
for layer in base_model.layers[:-5]:
  layer.trainable=False
```

```
# Recompile model with lower learning rate (it's typically best practice to lower the learnin
model.compile(loss="categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001), # learning rate lower
              metrics=METRICS)


# What layers in the model are trainable?
for layer in model.layers:
  print(layer.name, layer.trainable)



# Check which layers in our model is trainable
for layer_number, layer in enumerate(model.layers[1].layers):
  print(layer_number, layer.name, layer.trainable )
```

```
413 block6g_expand_conv False
414 block6g_expand_bn False
415 block6g_expand_activation False
416 block6g_dwconv False
417 block6g_bn False
418 block6g_activation False
419 block6g_se_squeeze False
420 block6g_se_reshape False
421 block6g_se_reduce False
422 block6g_se_expand False
423 block6g_se_excite False
424 block6g_project_conv False
425 block6g_project_bn False
426 block6g_drop False
427 block6g_add False
428 block6h_expand_conv False
429 block6h_expand_bn False
430 block6h_expand_activation False
431 block6h_dwconv False
432 block6h_bn False
433 block6h_activation False
434 block6h_se_squeeze False
435 block6h_se_reshape False
436 block6h_se_reduce False
437 block6h_se_expand False
438 block6h_se_excite False
439 block6h_project_conv False
440 block6h_project_bn False
441 block6h_drop False
442 block6h_add False
443 block7a_expand_conv False
```

```
444 block7a_expand_bn False
445 block7a_expand_activation False
446 block7a_dwconv False
447 block7a_bn False
448 block7a_activation False
449 block7a_se_squeeze False
450 block7a_se_reshape False
451 block7a_se_reduce False
452 block7a_se_expand False
453 block7a_se_excite False
454 block7a_project_conv False
455 block7a_project_bn False
456 block7b_expand_conv False
457 block7b_expand_bn False
458 block7b_expand_activation False
459 block7b_dwconv False
460 block7b_bn False
461 block7b_activation False
462 block7b_se_squeeze False
463 block7b_se_reshape False
464 block7b_se_reduce False
465 block7b_se_expand False
466 block7b_se_excite False
467 block7b_project_conv False
468 block7b_project_bn False
469 block7b_drop True
470 block7b_add True
```

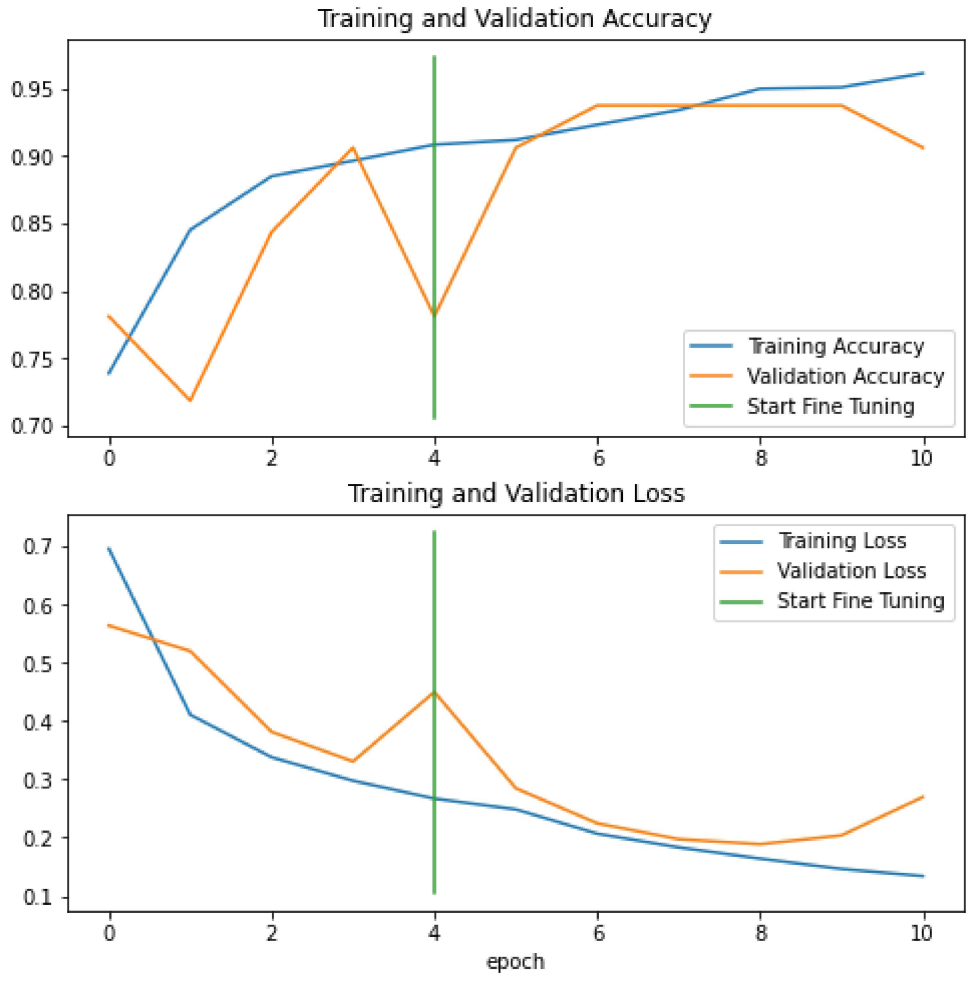
```
# Fine-tune for more 5 epochs
fine_tune_epochs = 10 # model has already done the 5 epochs (feature extraction), this is the

# Fine-tune our model
history_for_fine_tune = model.fit(train_data,
                                                    epochs=fine_tune_epochs,
                                                    validation_data=test_data,
                                                    validation_steps=int(0.15 * len(test_dat
                                                    initial_epoch=history_for_feature_extrac
```

```
Epoch 5/10
58/58 [==============================] - 26s 232ms/step - loss: 0.2486 - tp: 1837.0000 -
Epoch 6/10
58/58 [==============================] - 11s 178ms/step - loss: 0.2069 - tp: 1665.0000 -
Epoch 7/10
58/58 [==============================] - 11s 180ms/step - loss: 0.1834 - tp: 1698.0000 -
Epoch 8/10
58/58 [==============================] - 11s 181ms/step - loss: 0.1641 - tp: 1719.0000 -
Epoch 9/10
58/58 [==============================] - 11s 185ms/step - loss: 0.1463 - tp: 1731.0000 -
Epoch 10/10
58/58 [==============================] - 11s 178ms/step - loss: 0.1341 - tp: 1746.0000 -
```

```
# Evaluate on the whole test dataset
fine_tune_results =model.evaluate(test_data)
fine_tune_results
```

```
8/8 [==============================] - 1s 168ms/step - loss: 0.1935 - tp: 217.0000 - fp
[0.19352512061595917,
 217.0,
 14.0,
 682.0,
 15.0,
 0.9353448152542114,
 0.939393937587738,
 0.9353448152542114,
 0.9937666058540344]
```

```
compare_historys(history_for_feature_extraction,
                 history_for_fine_tune)
```




```
model.save("corn-maize-b4-model.h5")
model.save("corn-maize-b4-model.hdf5")
```

```
# Make predictions with model
```

```
preds_probs = model.predict(test_data, verbose=1) # set verbosity to see how long it left


len(test_data)


# How many predictions are there?
len(preds_probs)


# What's the shape of our predictions?
preds_probs.shape


# Let's see whats the first 10 predictions looks like
preds_probs[:10]



# What does the first prediction probability array look like?
preds_probs[0], len(preds_probs[0]), sum(preds_probs[0])



# We get one prediction probability per class(in our case there's 101 prediction probabilitie
print(f"Number of prediction probabilites for sample 0: {len(preds_probs[0])}")
print(f"What prediction probabilites sample 0 looks like:\n {preds_probs[0]}")
print(f"The class with highest predicted probability by the model for sample 0: {preds_probs[




# Get the pred classes of each model
pred_classes = preds_probs.argmax(axis=1)

# How do they look like?
pred_classes[:10]




# How many pred classes we have?
len(pred_classes)




# To get our test dataset labels we need to unravel our test_data BatchDataset
y_labels = []
for images, labels in test_data.unbatch():
  y_labels.append(labels.numpy().argmax()) # currently test labels look like: [0, 0, 0, 1,...
```

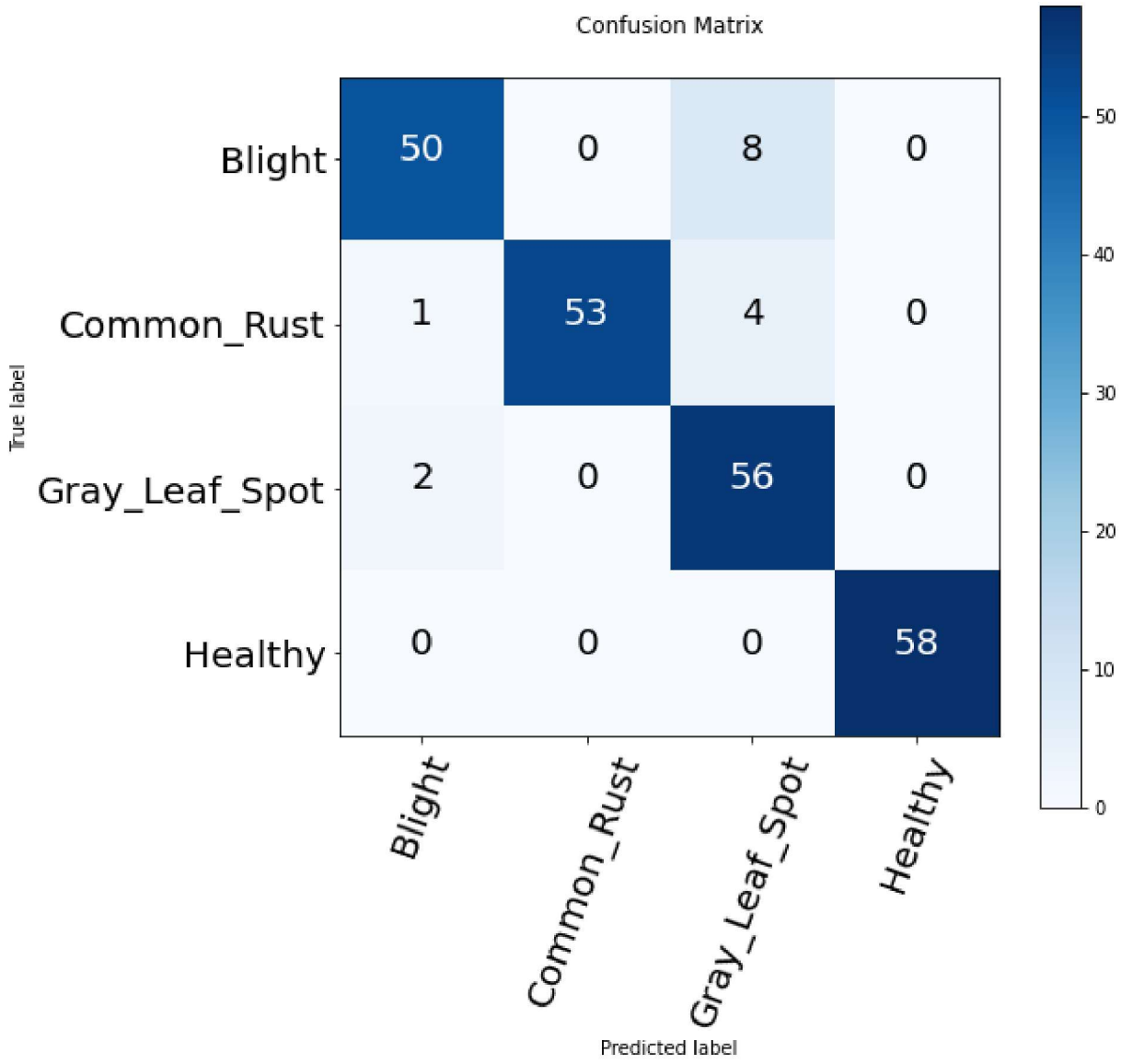
```
y_labels[:10] # look at the first 10



# How many y_labels are there?
len(y_labels)


test_data


len(test_data)
```

```
8/8 [==============================] - 4s 176ms/step
Number of prediction probabilites for sample 0: 4
What prediction probabilites sample 0 looks like:
 [0.6883736  0.02707887 0.28179568 0.00275171]
The class with highest predicted probability by the model for sample 0: 0
8
```

```
from helper_functions import make_confusion_matrix



# Get a list of class names
class_names =test_data.class_names
class_names[:10]



import itertools
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix

# We need to make some changes to our make_confusion_matrix function to ensure the x-label pr
def make_confusion_matrix(y_true, y_pred, classes=None, figsize=(10, 10), text_size=15, norm=
  """Makes a labelled confusion matrix comparing predictions and ground truth labels.

  If classes is passed, confusion matrix will be labelled, if not, integer class values
  will be used.

  Args:
    y_true: Array of truth labels (must be same shape as y_pred).
    y_pred: Array of predicted labels (must be same shape as y_true).
    classes: Array of class labels (e.g. string form). If `None`, integer labels are used.
    figsize: Size of output figure (default=(10, 10)).
    text_size: Size of output figure text (default=15).
    norm: normalize values or not (default=False).
    savefig: save confusion matrix to file (default=False).

  Returns:
```

```
  A labelled confusion matrix plot comparing y_true and y_pred.

Example usage:
  make_confusion_matrix(y_true=test_labels, # ground truth test labels
                        y_pred=y_preds, # predicted labels
                        classes=class_names, # array of class label names
                        figsize=(15, 15),
                        text_size=10)
"""
# Create the confustion matrix
cm = confusion_matrix(y_true, y_pred)
cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis] # normalize it
n_classes = cm.shape[0] # find the number of classes we're dealing with

# Plot the figure and make it pretty
fig, ax = plt.subplots(figsize=figsize)
cax = ax.matshow(cm, cmap=plt.cm.Blues) # colors will represent how 'correct' a class is, d
fig.colorbar(cax)

# Are there a list of classes?
if classes:
  labels = classes
else:
  labels = np.arange(cm.shape[0])

# Label the axes
ax.set(title="Confusion Matrix",
       xlabel="Predicted label",
       ylabel="True label",
       xticks=np.arange(n_classes), # create enough axis slots for each class
       yticks=np.arange(n_classes),
       xticklabels=labels, # axes will labeled with class names (if they exist) or ints
       yticklabels=labels)

# Make x-axis labels appear on bottom
ax.xaxis.set_label_position("bottom")
ax.xaxis.tick_bottom()

### changes (x-labels vertically) ###
plt.xticks(rotation=70, fontsize=text_size)
plt.yticks(fontsize=text_size)

# Set the threshold for different colors
threshold = (cm.max() + cm.min()) / 2.

# Plot the text on each cell
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
  if norm:
    plt.text(j, i, f"{cm[i, j]} ({cm_norm[i, j]*100:.1f}%)",
            horizontalalignment="center",
            color="white" if cm[i, j] > threshold else "black",
```

```
              size=text_size)
    else:
      plt.text(j, i, f"{cm[i, j]}",
              horizontalalignment="center",
              color="white" if cm[i, j] > threshold else "black",
              size=text_size)

  # Save the figure to the current working directory
  if savefig:
    fig.savefig("confusion_matrix.png")



make_confusion_matrix(y_true=y_labels,
                      y_pred=pred_classes,
                      classes=class_names,
                     figsize=(8, 8),
                     text_size=20,
                     savefig=True)
```



```
from sklearn.metrics import classification_report
print(classification_report(y_true=y_labels,
                            y_pred=pred_classes))



# Get a dictionary of the classification report
classification_report_dict = classification_report(y_labels, pred_classes, output_dict=True)
classification_report_dict



classification_report_dict["3"]["f1-score"]
```

```
                  precision    recall  f1-score   support

               0       0.94      0.86      0.90        58
               1       1.00      0.91      0.95        58
               2       0.82      0.97      0.89        58
               3       1.00      1.00      1.00        58

        accuracy                           0.94       232
       macro avg       0.94      0.94      0.94       232
    weighted avg       0.94      0.94      0.94       232

    1.0
```

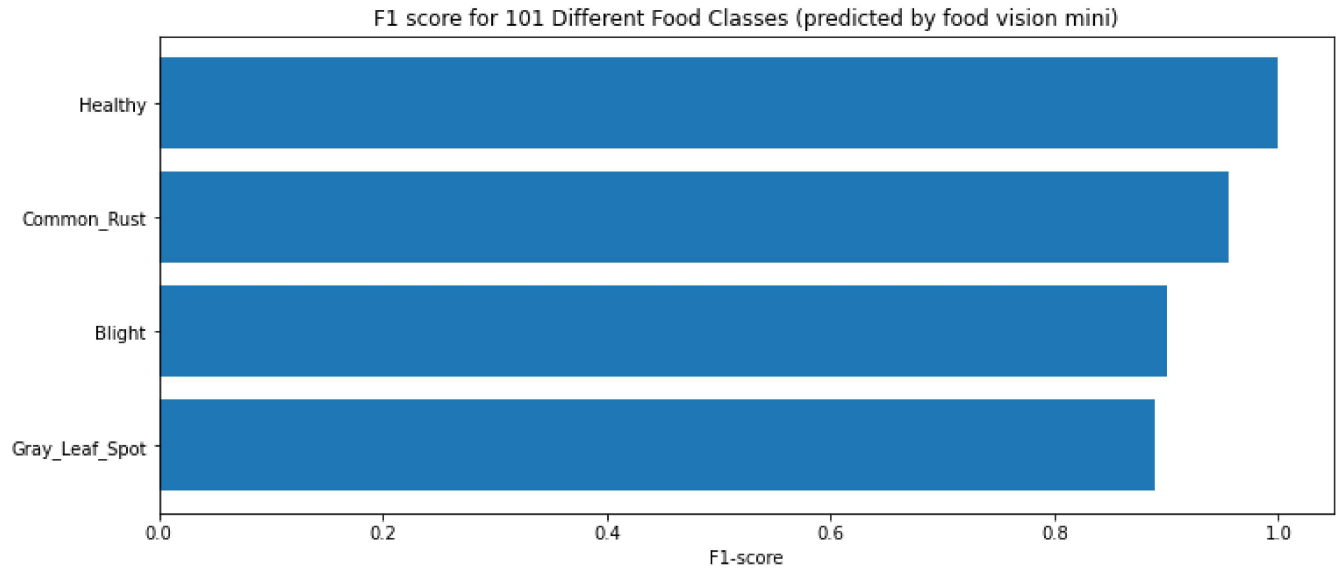
```
# Create empty dictionary
class_f1_scores = {}
# Loop through classification report dictionary items
for k, v in classification_report_dict.items():
  if k == "accuracy": # stop once we get to accuracy key
    break
  else:
     # Add names and f1-scores to new dictionary
     class_f1_scores[class_names[int(k)]] = v["f1-score"]
class_f1_scores




# Turn f1 scores into Dataframe visualization
import pandas as pd
f1_scores =pd.DataFrame({"class_names": list(class_f1_scores.keys()),
                         "f1-score": list(class_f1_scores.values())}).sort_values("f1-score",
f1_scores
```

	class_names	f1-score
3	Healthy	1.000000
1	Common_Rust	0.954955
0	Blight	0.900901

```
import matplotlib.pyplot as plt


import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(12, 5))
scores = ax.barh(range(len(f1_scores)), f1_scores["f1-score"].values) # get f1-score value
ax.set_yticks(range(len(f1_scores)))
ax.set_yticklabels(f1_scores["class_names"])
ax.set_xlabel("F1-score")
ax.set_title("F1 score for 101 Different Food Classes (predicted by food vision mini)")
ax.invert_yaxis(); # reverse the order of our plot


# Challenge: add value to the end of each bar of what the actual f1-score is
# (hint: use the "autolabel" function from here: https://matplotlib.org/2.0.2/examples/api/ba




# Create a function to load and create a images
def load_and_prep_image(filename, img_shape=224, scale=True):
  """
  Read in an image from filename, turn it into a tensor and reshapes into
  specified shape (img_shape, img_shape, color_channels=3).

  Args:
    filename(str): path to target image
    image_shape(int): height/width dimension of target image size
    scale(bool): scale pixel values from 0-255 to 0-1 or not

  Returns:
    image tensor of shape (img_shape, img_shape, 3)
  """

  # Read in the image
  img = tf.io.read_file(filename)

  # Decode image into tensor
  img = tf.io.decode_image(img, channels=3)

  # Resize the image
```

```
  img = tf.image.resize(img, [img_shape, img_shape])

  # Scale Yes/no?
  if scale:
    # reshape the image (get all value between 0 and 1)
    return img/255.
  else:
    return img # don't need to rescale image for EffiecienNet model in TensorFlow
```

```
# Make preds on series of images
import os
import random

plt.figure(figsize=(24,5))
for i in range(3):
  # Choose a random image(s) from a random class(es)
  class_name =random.choice(class_names)
  filename = random.choice(os.listdir(val_dir + "/" + class_name))
  filepath = val_dir + class_name + "/" + filename


  # Load the image and make predictions
  img = load_and_prep_image(filepath, scale=False)
  # print(img.shape)
  img_expanded = tf.expand_dims(img, axis=0)
  # print(img_expanded.shape)
  pred_prob = model.predict(img_expanded) # get prediction probability array
  pred_class = class_names[pred_prob.argmax()] # get heighst prediction probability index and
  # print(pred_prob)
   # print(pred_class)

    # Plot the image(s)
  plt.subplot(1, 3, i+1)
  plt.imshow(img/255.)
  if class_name == pred_class: # if predicted class matches truth class, make text green
    title_color = "g"
  else:
    title_color ="r"
  plt.title(f"actual: {class_name}, pred: {pred_class}, prob: {pred_prob.argmax():.2f}", c=ti
  plt.axis(False);
```




```
images = ['/content/common-rust.jpg', '/content/grey leaf spot.jpg', '/content/blight.jpg', '
```

```
# Make prediction on and plot the custom food images
for img in images:
  img = load_and_prep_image(img, scale=False) # don't need to do scale for our EfficienNetB0
  pred_prob = model.predict(tf.expand_dims(img, axis=0)) # make prediction on the image with
  pred_class = class_names[pred_prob.argmax()] # get the index with heighest prediction proba
  # plot the appropriate information
  plt.figure()
  plt.imshow(img/255.)
  plt.title(f"pred: {pred_class}, prob: {pred_prob.max():.2f}")
  plt.axis(False)
```

pred: Common_Rust, prob: 0.94

pred: Gray_Leaf_Spot, prob: 0.80

pred: Blight, prob: 0.85

pred: Healthy, prob: 0.93

✓ 0s completed at 1:21 PM