```
!wget https://raw.githubusercontent.com/soumikdalei/Food_Vision/refs/heads/main/helper_function.py
```

```
--2025-09-14 05:27:28--  https://raw.githubusercontent.com/soumikdalei/Food_Vision/refs/heads/main/helper_function.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5105 (5.0K) [text/plain]
Saving to: ‘helper_function.py’

helper_function.py  100%[===================>]   4.99K  --.-KB/s    in 0s

2025-09-14 05:27:28 (58.8 MB/s) - ‘helper_function.py’ saved [5105/5105]
```

```
from helper_function import create_tensorboard_callback, plot_loss_curves, compare_historys
```

```
import tensorflow_datasets as tfds
(train_data, test_data), ds_info = tfds.load(name="food101",
                                             split=["train", "validation"],
                                             shuffle_files=True,
                                             as_supervised=True,
                                             with_info=True)
```

```
WARNING:absl:Variant folder /root/tensorflow_datasets/food101/2.0.0 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/
Dl Completed...: 100%  1/1 [07:11<00:00, 185.27s/ url]

Dl Size...: 100%  4764/4764 [07:11<00:00, 22.96 MiB/s]

Extraction completed...: 100%  101008/101008 [07:11<00:00, 1653.68 file/s]
```

```
class_names = ds_info.features["label"].names
class_names[:10]
```

```
['apple_pie',
 'baby_back_ribs',
 'baklava',
 'beef_carpaccio',
 'beef_tartare',
 'beet_salad',
 'beignets',
 'bibimbap',
 'bread_pudding',
 'breakfast_burrito']
```

```
def preprocess_img(image, label, img_shape=224):

    image = tf.image.resize(image, [img_shape, img_shape]) #
    return tf.cast(image, tf.float32), label
```

```
import tensorflow as tf
train_data = train_data.map(map_func=preprocess_img, num_parallel_calls=tf.data.AUTOTUNE)
train_data = train_data.shuffle(buffer_size=1000).batch(batch_size=32).prefetch(buffer_size=tf.data.AUTOTUNE)
test_data = test_data.map(preprocess_img, num_parallel_calls=tf.data.AUTOTUNE)
test_data = test_data.batch(32).prefetch(tf.data.AUTOTUNE)
```

```
from helper_function import create_tensorboard_callback


checkpoint_path = "model_checkpoints/cp.weights.h5"
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                      monitor="val_accuracy",
                                                      save_best_only=True,
                                                      save_weights_only=True,
                                                      verbose=0)
```

```
from tensorflow.keras import mixed_precision
mixed_precision.set_global_policy(policy="mixed_float16")
```

```
mixed_precision.global_policy()
```

```
<DTypePolicy "mixed_float16">
```

```
from tensorflow.keras import layers


input_shape = (224, 224, 3)
base_model = tf.keras.applications.EfficientNetB0(include_top=False)
base_model.trainable = False


inputs = layers.Input(shape=input_shape, name="input_layer")

x = base_model(inputs, training=False)
x = layers.GlobalAveragePooling2D(name="pooling_layer")(x)
x = layers.Dense(len(class_names))(x)

outputs = layers.Activation("softmax", dtype=tf.float32, name="softmax_float32")(x)
model = tf.keras.Model(inputs, outputs)


model.compile(loss="sparse_categorical_crossentropy",
              optimizer=tf.keras.optimizers.Lion(learning_rate=0.0001),
              metrics=["accuracy"])
```

```
model.summary()
```

**Model: "functional_3"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_layer (InputLayer) | (None, 224, 224, 3) | 0 |
| efficientnetb0 (Functional) | (None, 7, 7, 1280) | 4,049,571 |
| pooling_layer (GlobalAveragePooling2D) | (None, 1280) | 0 |
| dense_3 (Dense) | (None, 101) | 129,381 |
| softmax_float32 (Activation) | (None, 101) | 0 |

**Total params:** 4,178,952 (15.94 MB)
**Trainable params:** 129,381 (505.39 KB)

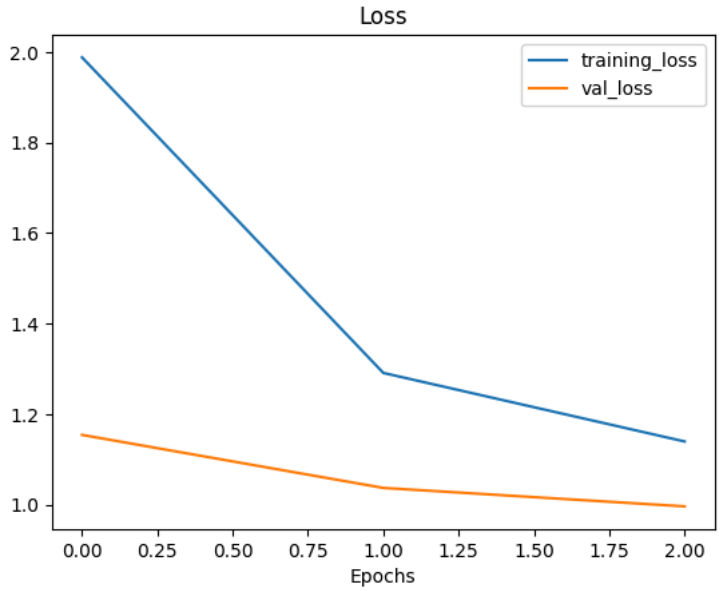
```
history_101_food_classes_feature_extract = model.fit(train_data,
                                                     epochs=3,
                                                     steps_per_epoch=len(train_data),
                                                     validation_data=test_data,
                                                     validation_steps=int(0.15 * len(test_data)),
                                                     callbacks=[create_tensorboard_callback("training_logs",
                                                                                            "efficientnetb0_101_classes_all_da
                                                                model_checkpoint])
```

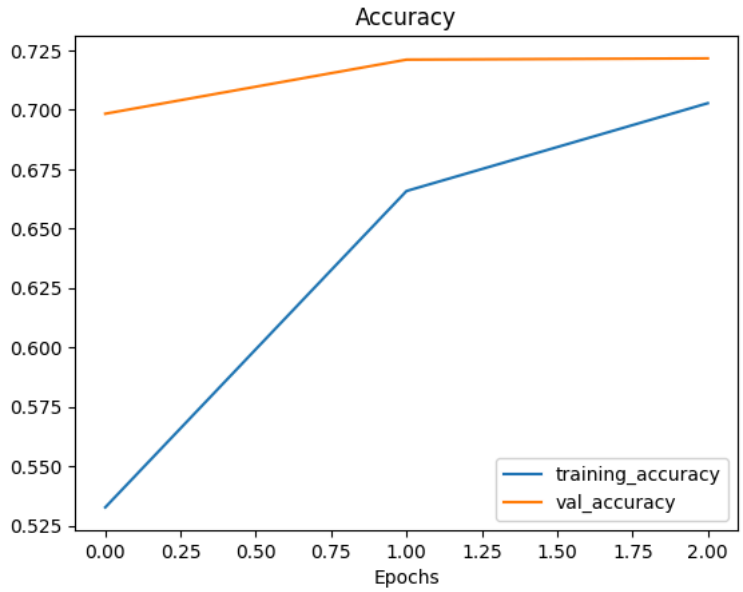
```
Saving TensorBoard log files to: training_logs/efficientnetb0_101_classes_all_data_feature_extract/20250914-055456
Epoch 1/3
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 222s 79ms/step - accuracy: 0.4038 - loss: 2.7217 - val_accuracy: 0.6984 - val_loss: 1.1540
Epoch 2/3
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 154s 64ms/step - accuracy: 0.6575 - loss: 1.3297 - val_accuracy: 0.7211 - val_loss: 1.0368
Epoch 3/3
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 263s 90ms/step - accuracy: 0.6997 - loss: 1.1553 - val_accuracy: 0.7217 - val_loss: 0.9961
```

```
results_feature_extract_model = model.evaluate(test_data)
results_feature_extract_model
```

```
790/790 ━━━━━━━━━━━━━━━━━━━━ 60s 76ms/step - accuracy: 0.7249 - loss: 0.9961
[0.9921460151672363, 0.7296633720397949]
```

```
plot_loss_curves(history_101_food_classes_feature_extract)
```




```
base_model.trainable = True

for layer in base_model.layers[:-20]:
    layer.trainable = False


model.compile(loss="sparse_categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001/10),
              metrics=["accuracy"])

fine_tune_epochs = 10
history_101_food_classes_fine_tune = model.fit(train_data,
                                               epochs=fine_tune_epochs,
                                               steps_per_epoch=len(train_data),
                                               validation_data=test_data,
                                               validation_steps=int(0.15 * len(test_data)),
                                               initial_epoch=history_101_food_classes_feature_extract.epoch[-1],
                                               callbacks=[create_tensorboard_callback("training_logs",
                                                                                      "efficientnetb0_101_classes_all_data_fi
                                                          model_checkpoint])
```

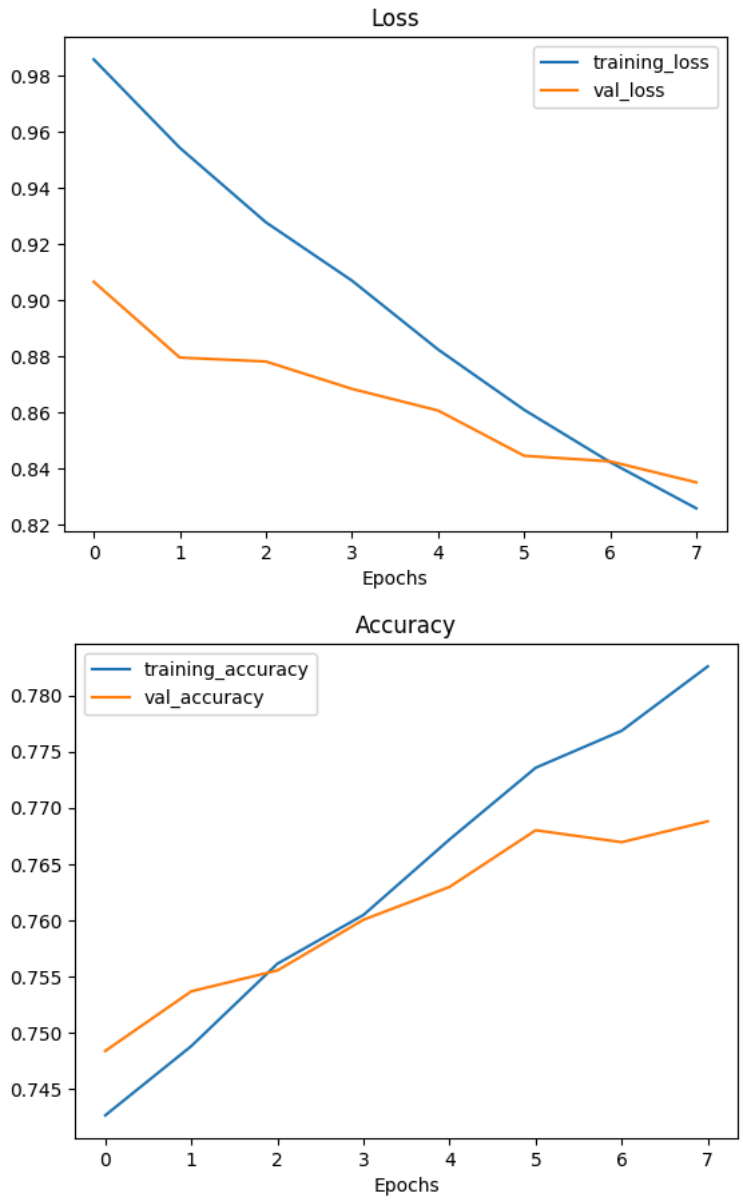
```
Saving TensorBoard log files to: training_logs/efficientnetb0_101_classes_all_data_fine_tuning/20250914-062214
Epoch 3/10
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 245s 83ms/step - accuracy: 0.7374 - loss: 1.0028 - val_accuracy: 0.7484 - val_loss: 0.9066
Epoch 4/10
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 245s 89ms/step - accuracy: 0.7455 - loss: 0.9608 - val_accuracy: 0.7537 - val_loss: 0.8796
Epoch 5/10
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 211s 68ms/step - accuracy: 0.7526 - loss: 0.9390 - val_accuracy: 0.7556 - val_loss: 0.8782
Epoch 6/10
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 199s 66ms/step - accuracy: 0.7582 - loss: 0.9154 - val_accuracy: 0.7601 - val_loss: 0.8684
Epoch 7/10
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 201s 84ms/step - accuracy: 0.7650 - loss: 0.8931 - val_accuracy: 0.7630 - val_loss: 0.8607
Epoch 8/10
```

```
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 171s 70ms/step - accuracy: 0.7715 - loss: 0.8653 - val_accuracy: 0.7680 - val_loss: 0.8446
Epoch 9/10
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 160s 67ms/step - accuracy: 0.7718 - loss: 0.8568 - val_accuracy: 0.7669 - val_loss: 0.8426
Epoch 10/10
2368/2368 ━━━━━━━━━━━━━━━━━━━━ 203s 85ms/step - accuracy: 0.7804 - loss: 0.8347 - val_accuracy: 0.7688 - val_loss: 0.8351
```

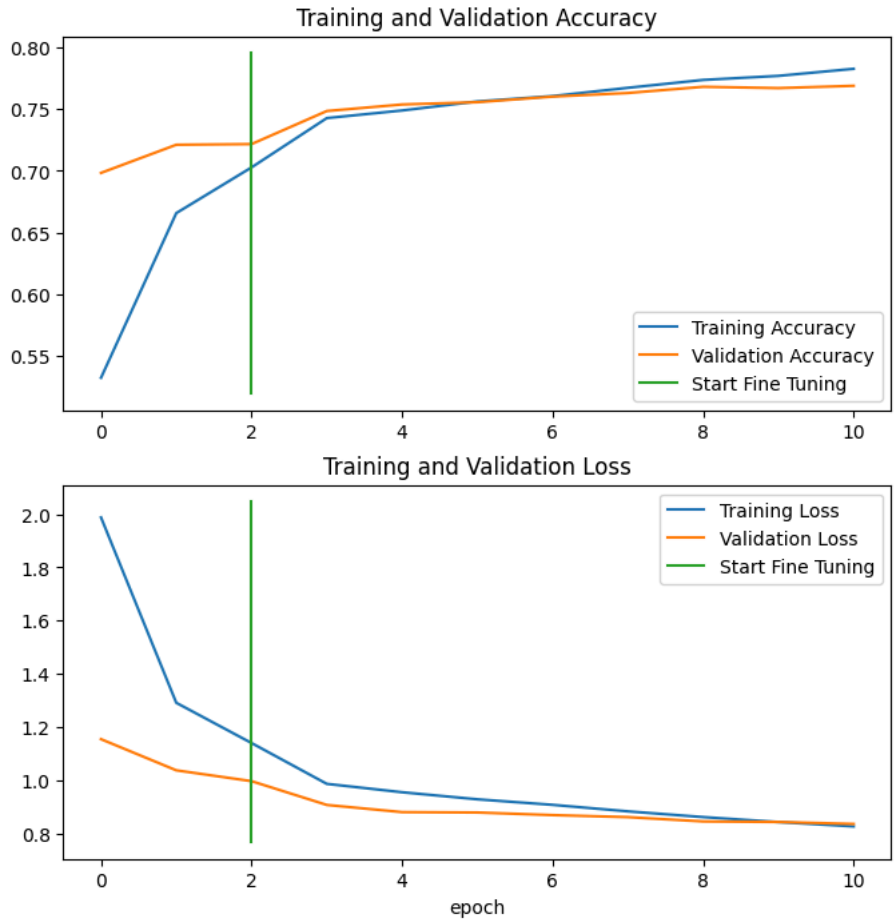
```
model.evaluate(test_data)
```

```
790/790 ━━━━━━━━━━━━━━━━━━━━ 62s 78ms/step - accuracy: 0.7683 - loss: 0.8379
[0.8365280628204346, 0.7691881060600281]
```

```
plot_loss_curves(history_101_food_classes_fine_tune)
```




```
compare_historys(original_history=history_101_food_classes_feature_extract,
                 new_history=history_101_food_classes_fine_tune,
                 initial_epochs=3)
```




```
def predict_on_custom_image(image_path, model, class_names, img_shape=224):

    img = tf.io.read_file(image_path)
    img = tf.image.decode_image(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)
    img = tf.image.resize(img, size=[img_shape, img_shape])
    img = tf.expand_dims(img, axis=0)

    prediction = model.predict(img)
    predicted_class_index = tf.argmax(prediction, axis=1)[0].numpy()
    predicted_class_name = class_names[predicted_class_index]
    prediction_probability = tf.reduce_max(prediction).numpy()

    return predicted_class_name, prediction_probability
```

```
custom_image_path = "/content/images (1).jpeg"
predicted_class, probability = predict_on_custom_image(custom_image_path, model, class_names)
print(f"Predicted class: {predicted_class}, Probability: {probability:.4f}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
Predicted class: grilled_salmon, Probability: 0.0197
```

```
y_pred=model.predict(test_data)
y_pred=tf.argmax(y_pred,axis=1)
y_pred
```

```
790/790 ━━━━━━━━━━━━━━━━━━━━ 82s 78ms/step
<tf.Tensor: shape=(25250,), dtype=int64, numpy=array([ 1, 81, 61, ..., 36, 34, 72])>
```

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

# Our function needs a different name to sklearn's plot_confusion_matrix
def make_confusion_matrix(y_true, y_pred, classes=None, figsize=(100, 100), text_size=15, norm=False, savefig=False):

  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]
  fig, ax = plt.subplots(figsize=figsize)
```

```
  cax = ax.matshow(cm, cmap=plt.cm.Blues)
  fig.colorbar(cax)


  if classes:
    labels = classes
  else:
    labels = np.arange(cm.shape[0])


  ax.set(title="Confusion Matrix",
         xlabel="Predicted label",
         ylabel="True label",
         xticks=np.arange(n_classes),
         yticks=np.arange(n_classes),
         xticklabels=labels,
         yticklabels=labels)


  ax.xaxis.set_label_position("bottom")
  ax.xaxis.tick_bottom()


  plt.xticks(rotation=70, fontsize=text_size)
  plt.yticks(fontsize=text_size)

  threshold = (cm.max() + cm.min()) / 2.


  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)


  if savefig:
    fig.savefig("confusion_matrix.png")
```

```
y_true = [label.numpy() for image, label in test_data.unbatch()]
make_confusion_matrix(y_true=y_true,
                      y_pred=y_pred,
                      classes=class_names)
```