🍔👁 Food Vision Big™

Project adapted from Udemy Course: TensorFlow Developer Certificate in 2023: Zero to Mastery by Daniel Bourke Objective: Perform better than DeepFood**, a 2016 paper which used a Convolutional Neural Network trained for 2-3 days to achieve 77.4% top-1 accuracy.

Goal: Accuracy > 77.4%

Contents:

- Using TensorFlow Datasets to download and explore data
- Creating preprocessing function for our data
- Batching & preparing datasets for modelling (**making our datasets run fast**)
- Creating modelling callbacks
- Setting up **mixed precision training**
- Building a feature extraction model
- Fine-tuning the feature extraction model

```
# Check GPU
!nvidia-smi -L
```

```
GPU 0: Tesla T4 (UUID: GPU-969b4cd4-41dc-2131-71a6-52395e6c6110)
```

```
import tensorflow as tf
print(tf.__version__)
```

```
2.12.0
```

```
# Get helper functions file
import os
```

Importing from helper functions

https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/extras/helper_functions.py

```
import datetime
import matplotlib.pyplot as plt

def create_tensorboard_callback(dir_name, experiment_name):
  """
  Creates a TensorBoard callback instand to store log files.

  Stores log files with the filepath:
    "dir_name/experiment_name/current_datetime/"

  Args:
    dir_name: target directory to store TensorBoard log files
    experiment_name: name of experiment directory (e.g. efficientnet_model_1)
  """
  log_dir = dir_name + "/" + experiment_name + "/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
  tensorboard_callback = tf.keras.callbacks.TensorBoard(
      log_dir=log_dir
  )
  print(f"Saving TensorBoard log files to: {log_dir}")
  return tensorboard_callback

def plot_loss_curves(history):
  """
  Returns separate loss curves for training and validation metrics.

  Args:
    history: TensorFlow model History object (see: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/History)
  """
  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();


def compare_historys(original_history, new_history, initial_epochs=5):
    """
    Compares two TensorFlow model History objects.

    Args:
      original_history: History object from original model (before new_history)
      new_history: History object from continued model training (after original_history)
      initial_epochs: Number of epochs in original_history (new_history plot starts from here)
    """

    # Get original history measurements
    acc = original_history.history["accuracy"]
    loss = original_history.history["loss"]

    val_acc = original_history.history["val_accuracy"]
    val_loss = original_history.history["val_loss"]

    # Combine original history with new history
    total_acc = acc + new_history.history["accuracy"]
    total_loss = loss + new_history.history["loss"]

    total_val_acc = val_acc + new_history.history["val_accuracy"]
    total_val_loss = val_loss + new_history.history["val_loss"]

    # Make plots
    plt.figure(figsize=(8, 8))
    plt.subplot(2, 1, 1)
    plt.plot(total_acc, label='Training Accuracy')
    plt.plot(total_val_acc, label='Validation Accuracy')
    plt.plot([initial_epochs-1, initial_epochs-1],
              plt.ylim(), label='Start Fine Tuning') # reshift plot around epochs
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(2, 1, 2)
    plt.plot(total_loss, label='Training Loss')
    plt.plot(total_val_loss, label='Validation Loss')
    plt.plot([initial_epochs-1, initial_epochs-1],
              plt.ylim(), label='Start Fine Tuning') # reshift plot around epochs
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.xlabel('epoch')
    plt.show()

def create_tensorboard_callback(dir_name, experiment_name):
  """
  Creates a TensorBoard callback instand to store log files.

  Stores log files with the filepath:
    "dir_name/experiment_name/current_datetime/"

  Args:
    dir_name: target directory to store TensorBoard log files
    experiment_name: name of experiment directory (e.g. efficientnet_model_1)
  """
  log_dir = dir_name + "/" + experiment_name + "/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
  tensorboard_callback = tf.keras.callbacks.TensorBoard(
      log_dir=log_dir
```

```
  )
  print(f"Saving TensorBoard log files to: {log_dir}")
  return tensorboard_callback
```

Using TensorFlow Datasets to download and explore data

```
# Get TensorFlow Datasets
import tensorflow_datasets as tfds
```

```
# List available datasets
datasets_list = tfds.list_builders()
print("food101" in datasets_list)
```

```
True
```

Creating preprocessing function for our data

```
# Load in the data (takes about 5-6 minutes in Google Colab)
(train_data, test_data), ds_info = tfds.load(name="food101", # target dataset to get from TFDS
                                             split=["train", "validation"], # what splits of data should we get? note: not all datasets have
                                             shuffle_files=True, # shuffle files on download?
                                             as_supervised=True, # download data in tuple format (sample, label), e.g. (image, label)
                                             with_info=True) # include dataset metadata? if so, tfds.load() returns tuple (data, ds_info)
```

```
# Features of Food101 TFDS
ds_info.features
```

```
FeaturesDict({
    'image': Image(shape=(None, None, 3), dtype=uint8),
    'label': ClassLabel(shape=(), dtype=int64, num_classes=101),
})
```

```
# Get class names
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']
```

Exploring the Food101 data from TensorFlow Datasets

```
# Sample of the training data
train_one_sample = train_data.take(1) # samples are in format (image_tensor, label)
```

```
# What does one sample of our training data look like?
train_one_sample
```

```
<_TakeDataset element_spec=(TensorSpec(shape=(None, None, 3), dtype=tf.uint8, name=None), TensorSpec(shape=(), dtype=tf.int64,
name=None))>
```

```
# Output info about our training sample
for image, label in train_one_sample:
  print(f"""
  Image shape: {image.shape}
  Image dtype: {image.dtype}
  Target class from Food101 (tensor form): {label}
  Class name (str form): {class_names[label.numpy()]}
        """)
```

```
Image shape: (512, 512, 3)
Image dtype: <dtype: 'uint8'>
Target class from Food101 (tensor form): 90
Class name (str form): spaghetti_bolognese
```

```
# What does an image tensor from TFDS's Food101 look like?
image
```

```
<tf.Tensor: shape=(512, 512, 3), dtype=uint8, numpy=
array([[[ 12,  13,   7],
        [ 12,  13,   7],
        [ 13,  14,   8],
        ...,
        [ 21,  11,   0],
        [ 21,  11,   0],
        [ 21,  11,   0]],

       [[ 12,  13,   7],
        [ 11,  12,   6],
        [ 11,  12,   6],
        ...,
        [ 21,  11,   0],
        [ 21,  11,   0],
        [ 21,  11,   0]],

       [[  7,   8,   2],
        [  7,   8,   2],
        [  7,   8,   2],
        ...,
        [ 22,  12,   2],
        [ 21,  11,   1],
        [ 20,  10,   0]],

       ...,

       [[188, 191, 184],
        [188, 191, 184],
        [188, 191, 184],
        ...,
        [243, 248, 244],
        [243, 248, 244],
        [242, 247, 243]],

       [[187, 190, 183],
        [189, 192, 185],
        [190, 193, 186],
        ...,
        [241, 245, 244],
        [241, 245, 244],
        [241, 245, 244]],

       [[186, 189, 182],
        [189, 192, 185],
        [191, 194, 187],
        ...,
        [238, 242, 241],
        [239, 243, 242],
        [239, 243, 242]]], dtype=uint8)>
```

```
# What are the min and max values?
tf.reduce_min(image), tf.reduce_max(image)
```

```
(<tf.Tensor: shape=(), dtype=uint8, numpy=0>,
 <tf.Tensor: shape=(), dtype=uint8, numpy=255>)
```

```
plt.imshow(image)
plt.title(class_names[label.numpy()]) # add title to image by indexing on class_names list
plt.axis(False);
```

spaghetti_bolognese

```
# Make a function for preprocessing images
def preprocess_img(image, label, img_shape=224):
    """
    Converts image datatype from 'uint8' -> 'float32' and reshapes image to
    [img_shape, img_shape, color_channels]
    """
    image = tf.image.resize(image, [img_shape, img_shape]) # reshape to img_shape
    return tf.cast(image, tf.float32), label # return (float32_image, label) tuple
```

```
# Preprocess a single sample image and check the outputs
preprocessed_img = preprocess_img(image, label)[0]
print(f"Image before preprocessing:\n {image[:2]}...,\nShape: {image.shape},\nDatatype: {image.dtype}\n")
print(f"Image after preprocessing:\n {preprocessed_img[:2]}...,\nShape: {preprocessed_img.shape},\nDatatype: {preprocessed_img.dtype}")
```

```
Image before preprocessing:
 [[[12 13  7]
  [12 13  7]
  [13 14  8]
  ...
  [21 11  0]
  [21 11  0]
  [21 11  0]]

 [[12 13  7]
  [11 12  6]
  [11 12  6]
  ...
  [21 11  0]
  [21 11  0]
  [21 11  0]]]...,
Shape: (512, 512, 3),
Datatype: <dtype: 'uint8'>

Image after preprocessing:
 [[[11.586735   12.586735    6.586735  ]
  [11.714286   12.714286    6.714286  ]
  [ 8.857142    9.857142    4.8571424 ]
  ...
  [20.714308   11.142836    1.2857144 ]
  [20.668371   10.668372    0.        ]
  [21.         11.          0.        ]]

 [[ 2.3571415   3.3571415   0.1428566 ]
  [ 3.1530607   4.153061    0.07653028]
  [ 3.0561223   4.0561223   0.        ]
  ...
  [26.071407   18.071407    7.0714073 ]
  [24.785702   14.785702    4.7857018 ]
  [22.499966   12.499966    2.4999657 ]]]...,
Shape: (224, 224, 3),
Datatype: <dtype: 'float32'>
```

```
# We can still plot our preprocessed image as long as we
# divide by 255 (for matplotlib capability)
plt.imshow(preprocessed_img/255.)
plt.title(class_names[label])
plt.axis(False);
```

spaghetti_bolognese

Batching & preparing datasets for modelling

```
# Map preprocessing function to training data (and paralellize)
train_data = train_data.map(map_func=preprocess_img, num_parallel_calls=tf.data.AUTOTUNE)
# Shuffle train_data and turn it into batches and prefetch it (load it faster)
train_data = train_data.shuffle(buffer_size=1000).batch(batch_size=32).prefetch(buffer_size=tf.data.AUTOTUNE)

# Map prepreprocessing function to test data
test_data = test_data.map(preprocess_img, num_parallel_calls=tf.data.AUTOTUNE)
# Turn test data into batches (don't need to shuffle)
test_data = test_data.batch(32).prefetch(tf.data.AUTOTUNE)
```

```
train_data, test_data
```

```
(<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,),
dtype=tf.int64, name=None))>,
 <_PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,),
dtype=tf.int64, name=None))>)
```

Creating modelling callbacks

```
# Create ModelCheckpoint callback to save model's progress
checkpoint_path = "model_checkpoints/cp.ckpt" # saving weights requires ".ckpt" extension
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                      monitor="val_accuracy",
                                                      save_best_only=True,
                                                      save_weights_only=True,
                                                      verbose=0)
```

Setup mixed precision training

```
# Turn on mixed precision training
from tensorflow.keras import mixed_precision
mixed_precision.set_global_policy(policy="mixed_float16")
```

```
mixed_precision.global_policy()
```

```
<Policy "mixed_float16">
```

Building a feature extraction model

```
from tensorflow.keras import layers

# Create base model
input_shape = (224, 224, 3)
base_model = tf.keras.applications.EfficientNetB0(include_top=False)
base_model.trainable = False # freeze base model layers
```

```
# Create Functional model
inputs = layers.Input(shape=input_shape, name="input_layer")
# Note: EfficientNetBX models have rescaling built-in but if your model didn't you could have a layer like below
# x = layers.Rescaling(1./255)(x)
x = base_model(inputs, training=False) # set base_model to inference mode only
x = layers.GlobalAveragePooling2D(name="pooling_layer")(x)
x = layers.Dense(len(class_names))(x) # want one output neuron per class
# Separate activation of output layer so we can output float32 activations
outputs = layers.Activation("softmax", dtype=tf.float32, name="softmax_float32")(x)
model = tf.keras.Model(inputs, outputs)

# Compile the model
model.compile(loss="sparse_categorical_crossentropy", # Use sparse_categorical_crossentropy when labels are *not* one-hot
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 [==============================] - 2s 0us/step
```

```
# Check out our model
model.summary()
```

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

 efficientnetb0 (Functional)  (None, None, None, 1280)  4049571

 pooling_layer (GlobalAverag  (None, 1280)             0
 ePooling2D)

 dense (Dense)               (None, 101)               129381

 softmax_float32 (Activation  (None, 101)              0
 )

=================================================================
Total params: 4,178,952
Trainable params: 129,381
Non-trainable params: 4,049,571
_________________________________________________________________
```

Checking layer dtype policies

```
# Check the dtype_policy attributes of layers in our model
for layer in model.layers:
    print(layer.name, layer.trainable, layer.dtype, layer.dtype_policy) # Check the dtype policy of layers
```

```
input_layer True float32 <Policy "float32">
efficientnetb0 False float32 <Policy "mixed_float16">
pooling_layer True float32 <Policy "mixed_float16">
dense True float32 <Policy "mixed_float16">
softmax_float32 True float32 <Policy "float32">
```

```
# Check the layers in the base model and see what dtype policy they're using
for layer in model.layers[1].layers[:20]:
    print(layer.name, layer.trainable, layer.dtype, layer.dtype_policy)
```

```
input_1 False float32 <Policy "float32">
rescaling False float32 <Policy "mixed_float16">
normalization False float32 <Policy "mixed_float16">
rescaling_1 False float32 <Policy "mixed_float16">
stem_conv_pad False float32 <Policy "mixed_float16">
stem_conv False float32 <Policy "mixed_float16">
stem_bn False float32 <Policy "mixed_float16">
stem_activation False float32 <Policy "mixed_float16">
block1a_dwconv False float32 <Policy "mixed_float16">
block1a_bn False float32 <Policy "mixed_float16">
block1a_activation False float32 <Policy "mixed_float16">
block1a_se_squeeze False float32 <Policy "mixed_float16">
block1a_se_reshape False float32 <Policy "mixed_float16">
block1a_se_reduce False float32 <Policy "mixed_float16">
block1a_se_expand False float32 <Policy "mixed_float16">
block1a_se_excite False float32 <Policy "mixed_float16">
block1a_project_conv False float32 <Policy "mixed_float16">
```

```
block1a_project_bn False float32 <Policy "mixed_float16">
block2a_expand_conv False float32 <Policy "mixed_float16">
block2a_expand_bn False float32 <Policy "mixed_float16">
```

Fine-tuning the feature extraction model

```
# Turn off all warnings except for errors
tf.get_logger().setLevel('ERROR')

# Fit the model with callbacks
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_data_feature_extr
                                                                model_checkpoint])
```

```
Saving TensorBoard log files to: training_logs/efficientnetb0_101_classes_all_data_feature_extract/20230502-152835
Epoch 1/3
2368/2368 [==============================] - 198s 76ms/step - loss: 1.7178 - accuracy: 0.5822 - val_loss: 1.1188 - val_accuracy: 0.7018
Epoch 2/3
2368/2368 [==============================] - 173s 72ms/step - loss: 1.1989 - accuracy: 0.6897 - val_loss: 1.0225 - val_accuracy: 0.7233
Epoch 3/3
2368/2368 [==============================] - 173s 72ms/step - loss: 1.0545 - accuracy: 0.7225 - val_loss: 1.0017 - val_accuracy: 0.7288
```

```
# Evaluate model (unsaved version) on whole test dataset
results_feature_extract_model = model.evaluate(test_data)
results_feature_extract_model
```

```
790/790 [==============================] - 51s 64ms/step - loss: 1.0019 - accuracy: 0.7267
[1.001875877380371, 0.7266534566879272]
```

Load and evaluate checkpoint weights

```
# Clone the model we created (this resets all weights)
cloned_model = tf.keras.models.clone_model(model)
cloned_model.summary()
```

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

 efficientnetb0 (Functional)  (None, None, None, 1280)  4049571

 pooling_layer (GlobalAverag  (None, 1280)             0
 ePooling2D)

 dense (Dense)               (None, 101)               129381

 softmax_float32 (Activation  (None, 101)              0
 )

=================================================================
Total params: 4,178,952
Trainable params: 129,381
Non-trainable params: 4,049,571
_________________________________________________________________
```

```
# Where are our checkpoints stored?
checkpoint_path
```

```
'model_checkpoints/cp.ckpt'
```

```
# Load checkpointed weights into cloned_model
cloned_model.load_weights(checkpoint_path)
```

```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7fc88d7b88b0>
```

```
# Compile cloned_model (with same parameters as original model)
cloned_model.compile(loss="sparse_categorical_crossentropy",
                     optimizer=tf.keras.optimizers.Adam(),
                     metrics=["accuracy"])
```

```
# Evalaute cloned model with loaded weights (should be same score as trained model)
results_cloned_model_with_loaded_weights = cloned_model.evaluate(test_data)
```

```
790/790 [==============================] - 52s 63ms/step - loss: 1.3772 - accuracy: 0.6312
```

```
# Check the layers in the base model and see what dtype policy they're using
for layer in cloned_model.layers[1].layers[:20]: # check only the first 20 layers to save printing space
    print(layer.name, layer.trainable, layer.dtype, layer.dtype_policy)
```

```
input_1 True float32 <Policy "float32">
rescaling False float32 <Policy "mixed_float16">
normalization False float32 <Policy "mixed_float16">
rescaling_1 False float32 <Policy "mixed_float16">
stem_conv_pad False float32 <Policy "mixed_float16">
stem_conv False float32 <Policy "mixed_float16">
stem_bn False float32 <Policy "mixed_float16">
stem_activation False float32 <Policy "mixed_float16">
block1a_dwconv False float32 <Policy "mixed_float16">
block1a_bn False float32 <Policy "mixed_float16">
block1a_activation False float32 <Policy "mixed_float16">
block1a_se_squeeze False float32 <Policy "mixed_float16">
block1a_se_reshape False float32 <Policy "mixed_float16">
block1a_se_reduce False float32 <Policy "mixed_float16">
block1a_se_expand False float32 <Policy "mixed_float16">
block1a_se_excite False float32 <Policy "mixed_float16">
block1a_project_conv False float32 <Policy "mixed_float16">
block1a_project_bn False float32 <Policy "mixed_float16">
block2a_expand_conv False float32 <Policy "mixed_float16">
block2a_expand_bn False float32 <Policy "mixed_float16">
```

Preparing our model's layers for fine-tuning

```
# Download the saved model from Google Storage
!wget https://storage.googleapis.com/ztm_tf_course/food_vision/07_efficientnetb0_feature_extract_model_mixed_precision.zip
```

```
--2023-05-02 15:40:27--  https://storage.googleapis.com/ztm_tf_course/food_vision/07_efficientnetb0_feature_extract_model_mixed_precisio
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.200.128, 74.125.68.128, 74.125.24.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.200.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 16976857 (16M) [application/zip]
Saving to: ‘07_efficientnetb0_feature_extract_model_mixed_precision.zip’

07_efficientnetb0_f 100%[===================>]  16.19M  7.26MB/s    in 2.2s

2023-05-02 15:40:30 (7.26 MB/s) - ‘07_efficientnetb0_feature_extract_model_mixed_precision.zip’ saved [16976857/16976857]
```

```
# Unzip the SavedModel downloaded from Google Stroage
!mkdir downloaded_gs_model # create new dir to store downloaded feature extraction model
!unzip 07_efficientnetb0_feature_extract_model_mixed_precision.zip -d downloaded_gs_model
```

```
Archive:  07_efficientnetb0_feature_extract_model_mixed_precision.zip
   creating: downloaded_gs_model/07_efficientnetb0_feature_extract_model_mixed_precision/
   creating: downloaded_gs_model/07_efficientnetb0_feature_extract_model_mixed_precision/variables/
  inflating: downloaded_gs_model/07_efficientnetb0_feature_extract_model_mixed_precision/variables/variables.data-00000-of-00001
  inflating: downloaded_gs_model/07_efficientnetb0_feature_extract_model_mixed_precision/variables/variables.index
  inflating: downloaded_gs_model/07_efficientnetb0_feature_extract_model_mixed_precision/saved_model.pb
   creating: downloaded_gs_model/07_efficientnetb0_feature_extract_model_mixed_precision/assets/
```

```
# Load and evaluate downloaded GS model
loaded_gs_model = tf.keras.models.load_model("downloaded_gs_model/07_efficientnetb0_feature_extract_model_mixed_precision")
```

```
WARNING:absl:Importing a function (__inference_block3a_expand_activation_layer_call_and_return_conditional_losses_192161) with ops wit
WARNING:absl:Importing a function (__inference_block4b_se_reduce_layer_call_and_return_conditional_losses_193305) with ops with unsave
WARNING:absl:Importing a function (__inference_block5a_activation_layer_call_and_return_conditional_losses_160748) with ops with unsav
WARNING:absl:Importing a function (__inference_block5c_activation_layer_call_and_return_conditional_losses_161371) with ops with unsav
WARNING:absl:Importing a function (__inference_block4a_activation_layer_call_and_return_conditional_losses_192937) with ops with unsav
WARNING:absl:Importing a function (__inference_block7a_se_reduce_layer_call_and_return_conditional_losses_196568) with ops with unsave
WARNING:absl:Importing a function (__inference_block2b_expand_activation_layer_call_and_return_conditional_losses_191788) with ops wit
WARNING:absl:Importing a function (__inference_block3a_expand_activation_layer_call_and_return_conditional_losses_159106) with ops wit
WARNING:absl:Importing a function (__inference_block3b_se_reduce_layer_call_and_return_conditional_losses_159497) with ops with unsave
WARNING:absl:Importing a function (__inference_block5c_expand_activation_layer_call_and_return_conditional_losses_161315) with ops wit
WARNING:absl:Importing a function (__inference_efficientnetb0_layer_call_and_return_conditional_losses_184891) with ops with unsaved c
WARNING:absl:Importing a function (__inference_model_layer_call_and_return_conditional_losses_178256) with ops with unsaved custom gra
WARNING:absl:Importing a function (__inference_block6a_activation_layer_call_and_return_conditional_losses_161710) with ops with unsav
WARNING:absl:Importing a function (__inference_block6a_expand_activation_layer_call_and_return_conditional_losses_161653) with ops wit
WARNING:absl:Importing a function (__inference_block3a_se_reduce_layer_call_and_return_conditional_losses_159212) with ops with unsave
WARNING:absl:Importing a function (__inference_stem_activation_layer_call_and_return_conditional_losses_158197) with ops with unsaved
WARNING:absl:Importing a function (__inference_efficientnetb0_layer_call_and_return_conditional_losses_189764) with ops with unsaved c
WARNING:absl:Importing a function (__inference_block3b_se_reduce_layer_call_and_return_conditional_losses_192606) with ops with unsave
WARNING:absl:Importing a function (__inference_block6a_activation_layer_call_and_return_conditional_losses_195081) with ops with unsav
WARNING:absl:Importing a function (__inference_block6c_activation_layer_call_and_return_conditional_losses_162333) with ops with unsav
WARNING:absl:Importing a function (__inference_block5a_se_reduce_layer_call_and_return_conditional_losses_160797) with ops with unsave
WARNING:absl:Importing a function (__inference_block5a_activation_layer_call_and_return_conditional_losses_194009) with ops with unsav
WARNING:absl:Importing a function (__inference_block6c_se_reduce_layer_call_and_return_conditional_losses_195822) with ops with unsave
WARNING:absl:Importing a function (__inference_block5b_activation_layer_call_and_return_conditional_losses_161033) with ops with unsav
WARNING:absl:Importing a function (__inference_block6b_expand_activation_layer_call_and_return_conditional_losses_195330) with ops wit
WARNING:absl:Importing a function (__inference_block3a_activation_layer_call_and_return_conditional_losses_159163) with ops with unsav
WARNING:absl:Importing a function (__inference_block4c_se_reduce_layer_call_and_return_conditional_losses_160459) with ops with unsave
WARNING:absl:Importing a function (__inference_block6b_activation_layer_call_and_return_conditional_losses_195407) with ops with unsav
WARNING:absl:Importing a function (__inference_block7a_se_reduce_layer_call_and_return_conditional_losses_163058) with ops with unsave
WARNING:absl:Importing a function (__inference_block3a_se_reduce_layer_call_and_return_conditional_losses_192280) with ops with unsave
WARNING:absl:Importing a function (__inference_block6d_activation_layer_call_and_return_conditional_losses_162671) with ops with unsav
WARNING:absl:Importing a function (__inference__wrapped_model_152628) with ops with unsaved custom gradients. Will likely fail if a g
WARNING:absl:Importing a function (__inference_block6b_se_reduce_layer_call_and_return_conditional_losses_162044) with ops with unsave
WARNING:absl:Importing a function (__inference_block2b_se_reduce_layer_call_and_return_conditional_losses_158873) with ops with unsave
WARNING:absl:Importing a function (__inference_block4c_activation_layer_call_and_return_conditional_losses_160410) with ops with unsav
WARNING:absl:Importing a function (__inference_block6a_expand_activation_layer_call_and_return_conditional_losses_195004) with ops wit
WARNING:absl:Importing a function (__inference_block3b_activation_layer_call_and_return_conditional_losses_192564) with ops with unsav
WARNING:absl:Importing a function (__inference_block5b_se_reduce_layer_call_and_return_conditional_losses_161082) with ops with unsave
WARNING:absl:Importing a function (__inference_block5c_se_reduce_layer_call_and_return_conditional_losses_161420) with ops with unsave
WARNING:absl:Importing a function (__inference_block4c_activation_layer_call_and_return_conditional_losses_193636) with ops with unsav
WARNING:absl:Importing a function (__inference_top_activation_layer_call_and_return_conditional_losses_196775) with ops with unsaved c
WARNING:absl:Importing a function (__inference_block4b_activation_layer_call_and_return_conditional_losses_160072) with ops with unsav
WARNING:absl:Importing a function (__inference_block6b_expand_activation_layer_call_and_return_conditional_losses_161939) with ops wit
WARNING:absl:Importing a function (__inference_block5a_expand_activation_layer_call_and_return_conditional_losses_193932) with ops wit
WARNING:absl:Importing a function (__inference_block4b_expand_activation_layer_call_and_return_conditional_losses_193186) with ops wit
WARNING:absl:Importing a function (__inference_block1a_se_reduce_layer_call_and_return_conditional_losses_158302) with ops with unsave
WARNING:absl:Importing a function (__inference_block6a_se_reduce_layer_call_and_return_conditional_losses_195123) with ops with unsave
WARNING:absl:Importing a function (__inference_block2a_expand_activation_layer_call_and_return_conditional_losses_191462) with ops wit
```

```
# Get a summary of our downloaded model
loaded_gs_model.summary()
```

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

 efficientnetb0 (Functional)  (None, None, None, 1280)  4049571

 pooling_layer (GlobalAverag  (None, 1280)             0
 ePooling2D)

 dense (Dense)               (None, 101)               129381

 softmax_float32 (Activation  (None, 101)              0
 )

=================================================================
Total params: 4,178,952
Trainable params: 129,381
Non-trainable params: 4,049,571
_________________________________________________________________
```

```
# How does the loaded model perform?
results_loaded_gs_model = loaded_gs_model.evaluate(test_data)
results_loaded_gs_model
```

```
790/790 [==============================] - 53s 65ms/step - loss: 1.0881 - accuracy: 0.7066
[1.0881004333496094, 0.7066138386726379]
```

```
# Are any of the layers in our model frozen?
for layer in loaded_gs_model.layers:
    layer.trainable = True # set all layers to trainable
    print(layer.name, layer.trainable, layer.dtype, layer.dtype_policy) # make sure loaded model is using mixed precision dtype_policy ("mixe
```

```
input_layer True float32 <Policy "float32">
efficientnetb0 True float32 <Policy "mixed_float16">
pooling_layer True float32 <Policy "mixed_float16">
dense True float32 <Policy "mixed_float16">
softmax_float32 True float32 <Policy "float32">
```

```
# Check the layers in the base model and see what dtype policy they're using
for layer in loaded_gs_model.layers[1].layers[:20]:
    print(layer.name, layer.trainable, layer.dtype, layer.dtype_policy)
```

```
input_1 True float32 <Policy "float32">
rescaling True float32 <Policy "mixed_float16">
normalization True float32 <Policy "float32">
stem_conv_pad True float32 <Policy "mixed_float16">
stem_conv True float32 <Policy "mixed_float16">
stem_bn True float32 <Policy "mixed_float16">
stem_activation True float32 <Policy "mixed_float16">
block1a_dwconv True float32 <Policy "mixed_float16">
block1a_bn True float32 <Policy "mixed_float16">
block1a_activation True float32 <Policy "mixed_float16">
block1a_se_squeeze True float32 <Policy "mixed_float16">
block1a_se_reshape True float32 <Policy "mixed_float16">
block1a_se_reduce True float32 <Policy "mixed_float16">
block1a_se_expand True float32 <Policy "mixed_float16">
block1a_se_excite True float32 <Policy "mixed_float16">
block1a_project_conv True float32 <Policy "mixed_float16">
block1a_project_bn True float32 <Policy "mixed_float16">
block2a_expand_conv True float32 <Policy "mixed_float16">
block2a_expand_bn True float32 <Policy "mixed_float16">
block2a_expand_activation True float32 <Policy "mixed_float16">
```

```
# Setup EarlyStopping callback to stop training if model's val_loss doesn't improve for 3 epochs
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_loss", # watch the val loss metric
                                                  patience=3) # if val loss decreases for 3 epochs in a row, stop training

# Create ModelCheckpoint callback to save best model during fine-tuning
checkpoint_path = "fine_tune_checkpoints/"
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                      save_best_only=True,
                                                      monitor="val_loss")
```

```
# Creating learning rate reduction callback
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",
                                                 factor=0.2, # multiply the learning rate by 0.2 (reduce by 5x)
                                                 patience=2,
                                                 verbose=1, # print out when learning rate goes down
                                                 min_lr=1e-7)
```

```
# Compile the model
loaded_gs_model.compile(loss="sparse_categorical_crossentropy", # sparse_categorical_crossentropy for labels that are *not* one-hot
                        optimizer=tf.keras.optimizers.Adam(0.0001), # 10x lower learning rate than the default
                        metrics=["accuracy"])
```

```
# Start to fine-tune (all layers)
history_101_food_classes_all_data_fine_tune = loaded_gs_model.fit(train_data,
                                                        epochs=100, # fine-tune for a maximum of 100 epochs
                                                        steps_per_epoch=len(train_data),
                                                        validation_data=test_data,
                                                        validation_steps=int(0.15 * len(test_data)), # validation during training on 15% of t
                                                        callbacks=[create_tensorboard_callback("training_logs", "efficientb0_101_classes_all_
                                                                   model_checkpoint, # save only the best model during training
                                                                   early_stopping, # stop model after X epochs of no improvements
                                                                   reduce_lr]) # reduce the learning rate after X epochs of no improvements
```

```
logs/efficientb0_101_classes_all_data_fine_tuning/20230502-154136

- ETA: 0s - loss: 0.9197 - accuracy: 0.7529WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_co
- 472s 176ms/step - loss: 0.9197 - accuracy: 0.7529 - val_loss: 0.8199 - val_accuracy: 0.7720 - lr: 1.0000e-04

- 366s 154ms/step - loss: 0.5789 - accuracy: 0.8399 - val_loss: 0.8240 - val_accuracy: 0.7778 - lr: 1.0000e-04
```

```
 - ETA: 0s - loss: 0.3303 - accuracy: 0.9053
ng rate to 1.9999999494757503e-05.
 - 382s 160ms/step - loss: 0.3303 - accuracy: 0.9053 - val_loss: 0.8844 - val_accuracy: 0.7847 - lr: 1.0000e-04

 - 380s 160ms/step - loss: 0.0843 - accuracy: 0.9790 - val_loss: 0.9218 - val_accuracy: 0.8001 - lr: 2.0000e-05
```

Goal Achieved using Transfer Learning

Accuracy at 97.90% > 77.4%

Val Accuracy at 80.01%

✓ 26m 40s completed at 12:08 AM