01 Milestone Project 1: 🍔::👁 Food Vision Big™

In this notebook, We're going to be building Food Vision Big, using all of the data from the Food101 dataset.

All 75,750 training images and 25,250 testing images

What we're going to cover

- 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
- Viewing training results on TensorBoard

Downloading Food101 dataset from Tensorflow Datasets

```
import tensorflow as tf
import tensorflow_datasets as tdfs
```

To get the list of available datasets, use tfds.list_builders()

```
# Check whether our Food101 dataset is present in TDFS or not
Datasets_list = tdfs.list_builders()

target = "food101"
print(f"The {target} dataset is present in {target in Datasets_list}")
```

```
The food101 dataset is present in True
```

The easiest way of loading a dataset is `tfds.load`. It will:

- Download the data and save it as tfrecord files.
- Load the tfrecord and create the tf.data.Dataset.
- split= it splits the data ["train","validation"]

- shuffle_files=: Control whether to shuffle the files between each epoch (TFDS store big datasets in multiple smaller files).
- with_info=True: Returns the tfds.core.DatasetInfo containing dataset metadata

```
(train_data, test_data), ds_info = tdfs.load("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 da
Downloading and preparing dataset Unknown size (download: Unknown size, gener
Dl Completed...: 100% 1/1 [18:40<00:00, 877.98s/ url]
Dl Size...: 100% 4764/4764 [18:40<00:00, 2.44 MiB/s]
Extraction completed...: 100% 101008/101008 [18:40<00:00, 1609.09 file/s]
Dataset food101 downloaded and prepared to /root/tensorflow_datasets/food101/
```

By using `with_info = True` returns the dataset info

```
# check what features does our dataset have
ds_info.features
```

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

```
class_names = ds_info.features['label'].names
class_names[:20]
# class_names[82]
```

```
['apple_pie',
 'baby_back_ribs',
 'baklava',
 'beef_carpaccio',
 'beef_tartare',
 'beet_salad',
 'beignets',
 'bibimbap',
 'bread_pudding',
 'breakfast_burrito',
 'bruschetta',
 'caesar_salad',
 'cannoli',
 'caprese_salad',
 'carrot_cake',
 'ceviche',
 'cheesecake',
 'cheese_plate',
 'chicken_curry',
 'chicken_quesadilla']
```

Create a preprocessing function for our data

```
# first take one sample of our data and know its (shape,dtype)
sample_of_train_data = train_data.take(1)
sample_of_train_data
```

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

```
for image, label in sample_of_train_data:
  print(f'''
  The image has shape: {image.shape},
  The image data type:{image.dtype},
  The label of an image in (tensor form):{label},
  The label of an image in (str form): {class_names[label.numpy()]}
  ''')
```

```

  The image has shape: (512, 512, 3),
  The image data type:<dtype: 'uint8'>,
  The label of an image in (tensor form):30,
  The label of an image in (str form): deviled_eggs
```

```
# let's visualize the image
import matplotlib.pyplot as plt
plt.imshow(image)
plt.title(class_names[label.numpy()])
plt.axis(False)
```

```
(np.float64(-0.5), np.float64(511.5), np.float64(511.5), np.float64(-0.5))
```

I created a function that helps us to make all images convert `shape into (224,224)` and `data type from unit8 to float32`

```
def preprocess_img(image, label, img_shape=224):
  image = tf.image.resize(image,[img_shape,img_shape])
  return tf.cast(image, dtype=tf.float32),label
```

```
preprocessed_img = preprocess_img(image,label)[0]
preprocessed_img
```

```
<tf.Tensor: shape=(224, 224, 3), dtype=float32, numpy=
array([[[ 21.801022,  24.801022,  43.80102 ],
        [ 17.7551  ,  19.7551  ,  40.755104],
        [ 16.16837 ,  18.16837 ,  41.16837 ],
        ...,
        [ 37.61732 ,  46.831585,  75.964294],
        [ 30.214298,  41.2143  ,  72.02044 ],
        [ 26.163052,  36.80591 ,  69.877335]],

       [[ 22.091837,  23.091837,  43.09184 ],
        [ 10.974489,  11.974489,  31.97449 ],
        [ 16.112244,  17.112244,  38.112244],
        ...,
        [ 45.48988 ,  57.48988 ,  81.20421 ],
        [ 42.413116,  54.34169 ,  80.49475 ],
        [ 33.551304,  45.479874,  71.76559 ]],

       [[ 31.57653 ,  30.57653 ,  48.14796 ],
        [ 21.      ,  20.      ,  37.999996],
        [ 31.688778,  29.857143,  50.352043],
```

```
          ...,
          [ 36.87249 ,  52.4235  ,  70.229675],
          [ 42.530632,  57.9592  ,  76.316345],
          [ 45.449043,  60.877613,  79.23476 ]],

         ...,

         [[152.57149 ,  99.571495, 119.14297 ],
          [157.84186 , 106.81125 , 125.398026],
          [152.21431 , 104.59697 , 120.09185 ],
          ...,
          [105.47965 , 143.73982 , 181.03062 ],
          [101.41298 , 139.78543 , 177.14252 ],
          [ 97.933525, 136.93867 , 174.01003 ]],

         [[158.04582 , 104.04582 , 127.04582 ],
          [153.99483 , 102.99482 , 121.99482 ],
          [149.72957 , 102.15814 , 117.40302 ],
          ...,
          [105.44889 , 139.64784 , 177.64784 ],
          [ 96.71934 , 130.78568 , 168.78568 ],
          [ 95.00502 , 129.95909 , 167.95909 ]],

         [[149.08682 ,  95.08683 , 118.08683 ],
          [150.14304 ,  99.14303 , 117.4797  ],
          [150.7756  , 103.20417 , 116.83681 ],
          ...,
          [104.8928  , 136.89279 , 175.89279 ],
          [101.147995, 133.148   , 172.148   ],
          [101.16301 , 133.62221 , 172.39261 ]]], dtype=float32)>
```

```
print(f"{preprocessed_img.shape},{preprocessed_img.dtype}")
plt.imshow(preprocessed_img/255.)
plt.title(class_names[label.numpy()])
plt.axis(False)
```

```
(224, 224, 3),<dtype: 'float32'>
(np.float64(-0.5), np.float64(223.5), np.float64(223.5), np.float64(-0.5))
```

Batching & preparing datasets for modelling (making our datasets run fast)

In this section we are going to make our images to form a batch of size 32 so that it can instead of going to run all at once it will run one batche of images at a time.

```
train_data, test_data
```

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

By using `map()` we are going to map all images to the associated function, `num_parallel_calls=tf.data.AUTOTNE` it allocate all the available CPU to this

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

```
# Map prepreprocessing function to test data
test_data = test_data.map(preprocess_img, num_parallel_calls=tf.data.AUTOTUN
# 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))>)
```

Create modelling callbacks

```
# import helper functions
import os
import pathlib
if not os.path.exists("helper_functions.py"):
  !wget https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning
else:
  print("The helper_functions.py already exists skipping download...")
```

```
--2025-11-16 11:06:09--  https://raw.githubusercontent.com/mrdbourke/tensorfl
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.10
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.1
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

2025-11-16 11:06:09 (97.5 MB/s) - ‘helper_functions.py’ saved [10246/10246]
```

```
from helper_functions import create_tensorboard_callback
```

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

Setting up mixed precision training

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

```
mixed_precision.global_policy() # should output "mixed_float16" (if your GPU
```

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

Build feature extraction model using EfficientNetV2B0

Callbacks: ready to roll.

Mixed precision: turned on.

Let's build a model.

Because our dataset is quite large, we're going to move towards fine-tuning an existing pretrained model (EfficienetNetV20).

But before we get into fine-tuning, let's set up a feature-extraction model.

Recall, the typical order for using transfer learning is:

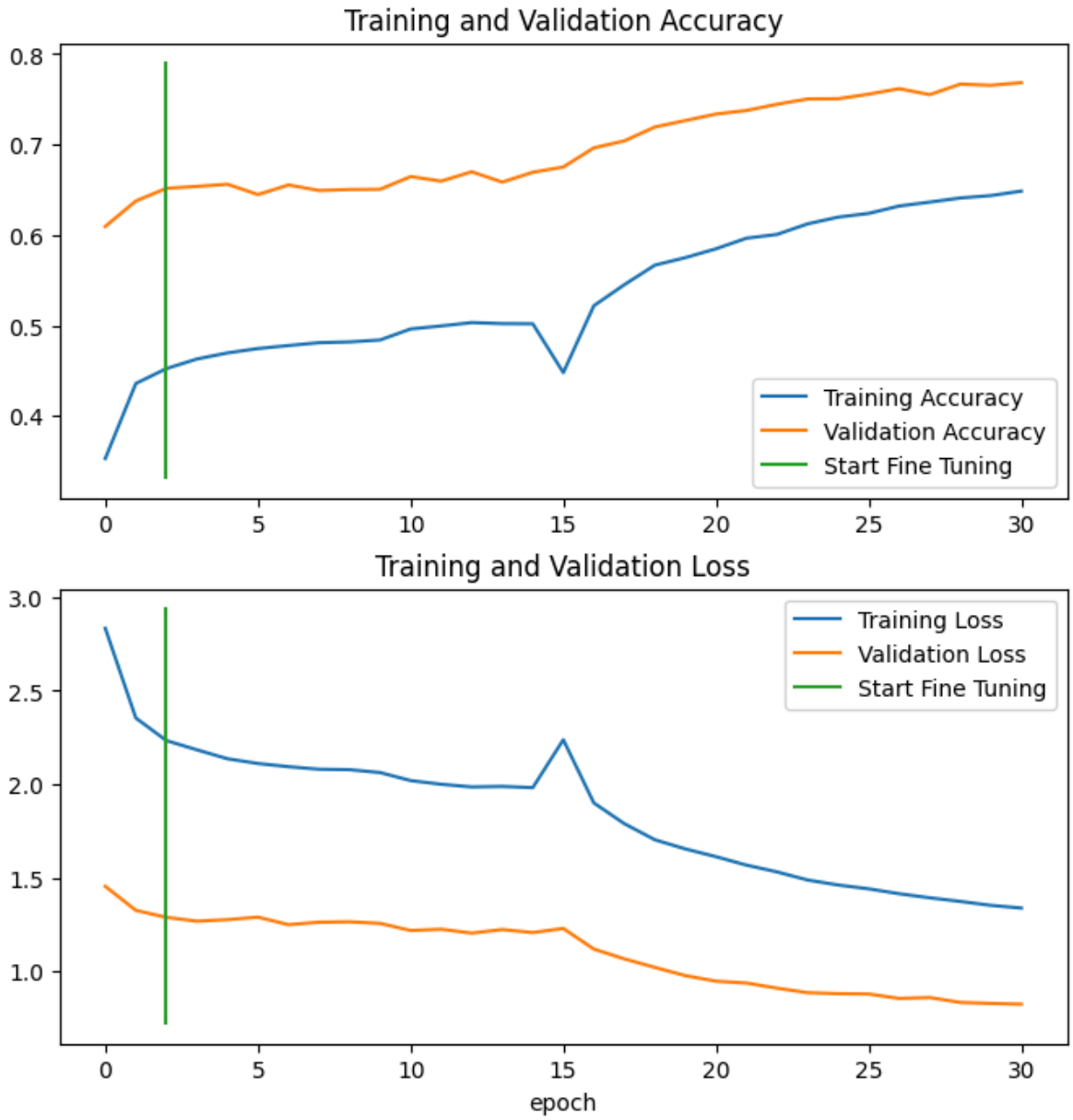
1. Build a feature extraction model (replace the top few layers of a pretrained model)
2. Train for a few epochs with lower layers frozen
3. Fine-tune if necessary with multiple layers unfrozen

↳ 20 cells hidden

Comparing both accuracy and loss before and after fine tuning

```
from helper_functions import compare_historys
```

```
compare_historys(original_history=history_1,
                 new_history=history_2,
                 initial_epochs=3)
```



Prediction

```
import numpy as np
y_true = []
y_pred = []
for batch_x, batch_y in test_data:
  preds = model_1.predict(batch_x, verbose=0)
  y_pred.extend(np.argmax(preds, axis=1))
  y_true.extend(batch_y.numpy())
y_true = np.array(y_true)
y_pred = np.array(y_pred)
```

```
import matplotlib.pyplot as plt
import numpy as np

def plot_random_images(model, test_data, class_names):
  # Get one batch from test_data
  for images_batch, labels_batch in test_data.take(1):
    break # We only need the first batch

  plt.figure(figsize=(10,10))
  # Iterate over individual images in the batch
  num_images_to_plot = min(8, images_batch.shape[0]) # Plot up to 8 images,
  for i in range(num_images_to_plot):
    image = images_batch[i]
    label = labels_batch[i]

    plt.subplot(2, 4, i+1)
    # Expand dims for prediction since model expects a batch of inputs
    pred_prob = model.predict(tf.expand_dims(image, axis=0), verbose=0)
    pred_class = tf.argmax(pred_prob, axis=1)[0]
    pred_score = tf.reduce_max(pred_prob) # Corrected typo from redcue_max t

    plt.imshow(image/255.)
    true_label = class_names[label.numpy()]
    predicted_label = class_names[pred_class.numpy()]
    color = "green" if true_label == predicted_label else "red"
    plt.title(f"True: {true_label}\nPred: {predicted_label} ({pred_score.num
    plt.axis("off")
  plt.tight_layout()
  plt.show()
```

```
plot_random_images(model_1, test_data, class_names=class_names)
```

calculating accuracy, precision, recall, f1-score

```
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
def calculate_results(y_true, y_pred):

  # Calculate model accuracy
  model_accuracy = accuracy_score(y_true, y_pred) * 100
  # Calculate model precision, recall and f1 score using "weighted average
  model_precision, model_recall, model_f1, _ = precision_recall_fscore_suppo
  model_results = {"accuracy": model_accuracy,
                   "precision": model_precision,
                   "recall": model_recall,
                   "f1": model_f1}
```

```
  return model_results
```

```
calculate_results(y_true = y_true,
                  y_pred= y_pred)
```

```
{'accuracy': 76.609900990099,
 'precision': 0.7737069277036254,
 'recall': 0.7660990099009901,
 'f1': 0.7653619932424867}
```

Confusion matrix

```
from helper_functions import make_confusion_matrix
# Plot a confusion matrix with all 25250 predictions, ground truth labels an
make_confusion_matrix(y_true=y_true,
                      y_pred=y_pred,
                      classes=class_names,
                      figsize=(200, 200),
                      text_size=20,
                      norm=True,
                      savefig=True)
```