TensorFlow

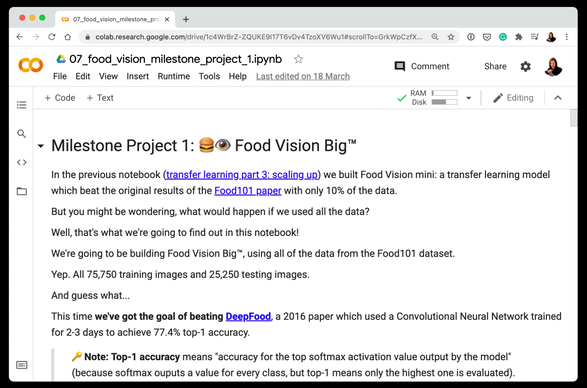
Milestone Project 1

🍔👁 Food Vision Big™

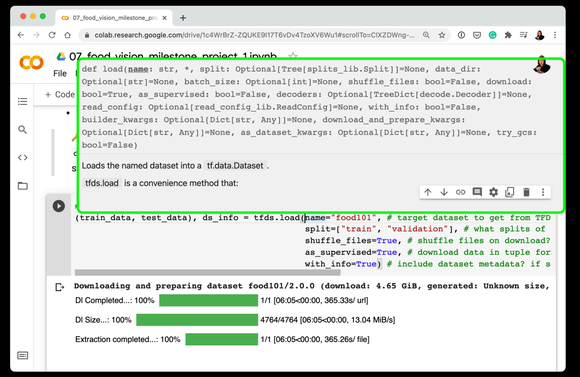
Where can you get help?

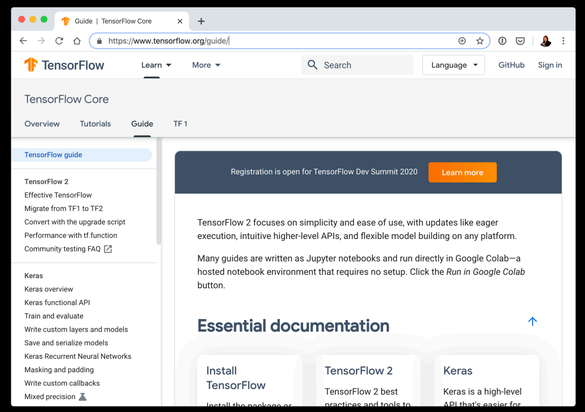
- Follow along with the code
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring
- Search for it
- Try again
- Ask (don't forget the Discord chat!)

(yes, including the "dumb" questions)



"If in doubt, run the code"





A new way of approaching things

What we're going to cover (broadly)

- Using TensorFlow Datasets to download and explore data (all of Food101)
- Creating a preprocessing function for our data
- Batching & preparing datasets for modelling (making them run fast)
- Setting up mixed precision training (faster model training)

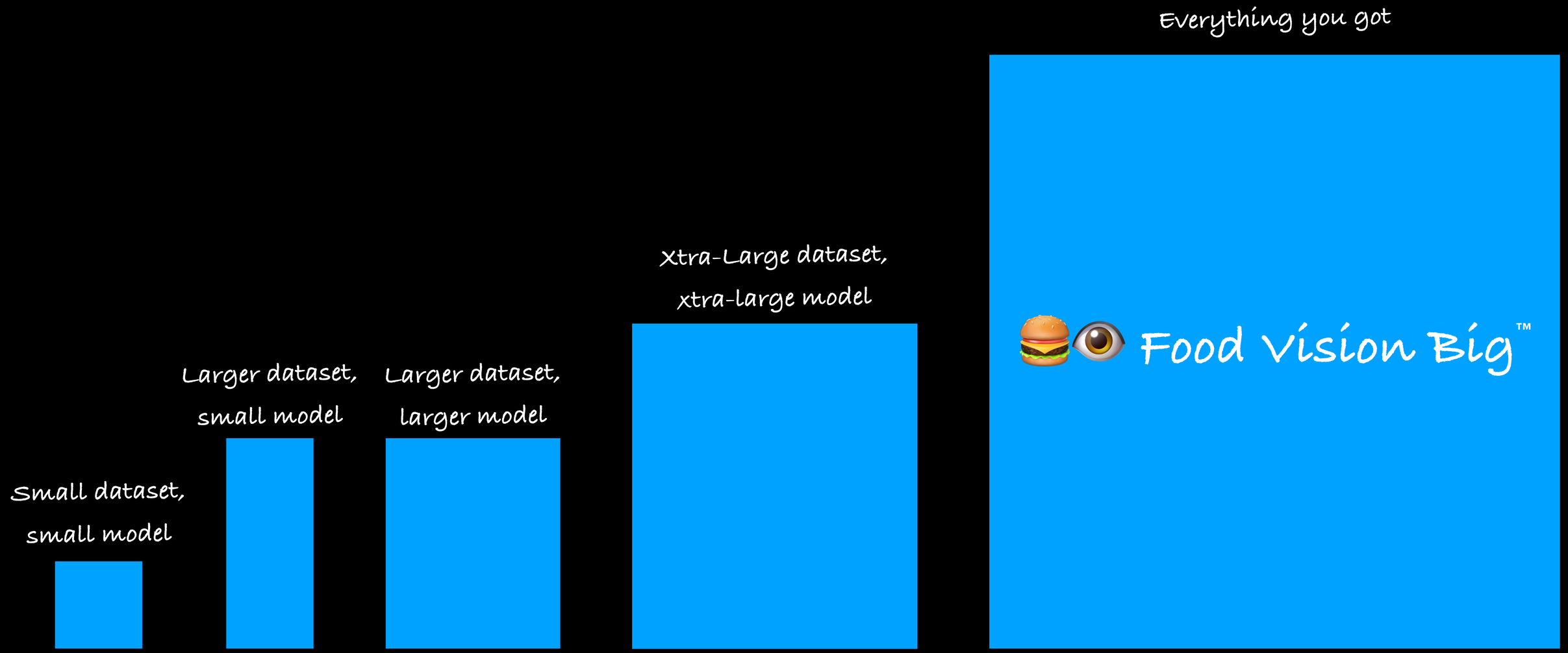
What you're going to cover...

- Building and training a feature extraction model
- Fine-tuning your feature extraction model to the beat the DeepFood paper 📈
- Evaluating your model results on TensorBoard
- Evaluating your model results by making and plotting predictions

(cook up lots of code!)

How:

Serial experimentation



🍔👁 Food Vision: Dataset(s) we're using

Note: For randomly selected data, the Food101 dataset was downloaded and modified using the Image Data Modification Notebook

Dataset Name	Source	Classes	Training data	Testing data
pizza_steak	Food101	Pizza, steak (2)	750 images of pizza and steak (same as original Food101 dataset)	250 images of pizza and steak (same as original Food101 dataset)
10_food_classes_1_percent	Same as above	Chicken curry, chicken wings, fried rice, grilled salmon, hamburger, ice cream, pizza, ramen, steak, sushi (10)	7 randomly selected images of each class (**1%** of original training data)	250 images of each class (same as original Food101 dataset)
10_food_classes_10_percent	Same as above	Same as above	75 randomly selected images of each class (**10%** of original training data)	Same as above
10_food_classes_100_percent	Same as above	Same as above	750 images of each class (**100%** of original training data)	Same as above
101_food_classes_10_percent	Same as above	All classes from Food101 (101)	75 images of each class (10% of the original Food101 training dataset)	250 images of each class (same as original Food101 dataset)
"food101"	Same as above (from TensorFlow Datasets)	All classes from Food101 (101)	75,750 images (750 images per class)	250 images of each class (same as original Food101 dataset)

Let’s code!

What is TensorFlow Datasets (TFDS)?

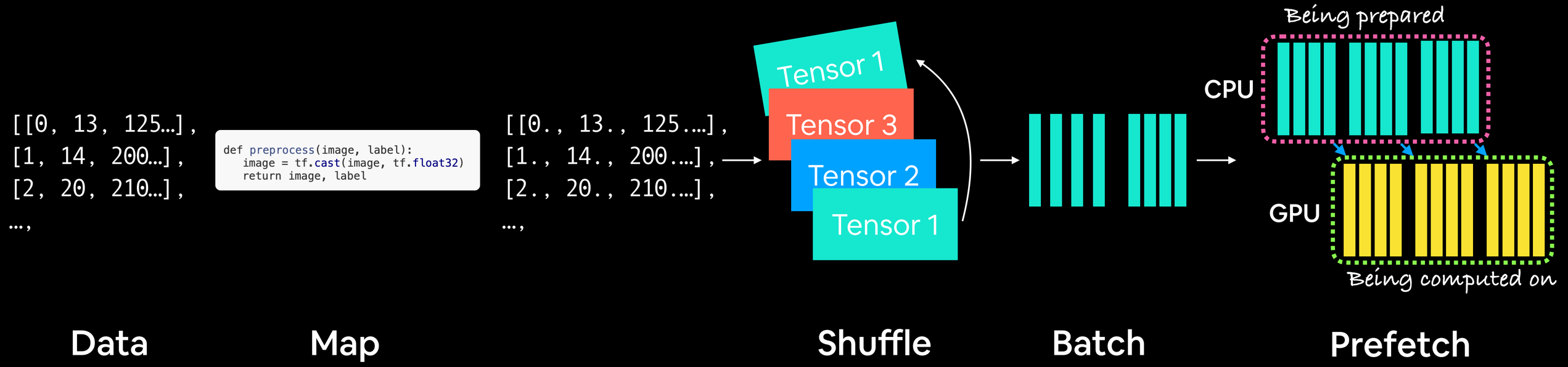
TensorFlow Datasets is a place for prepared and ready-to-use machine learning datasets.

- Why use TensorFlow Datasets?
  - Load data already in tensor format
  - Practice on well established datasets (for many different problem types)
  - Experiment with different modelling techniques on a consistent dataset
- Why not use TensorFlow Datasets?
  - Datasets are static (do not change like real-world datasets)

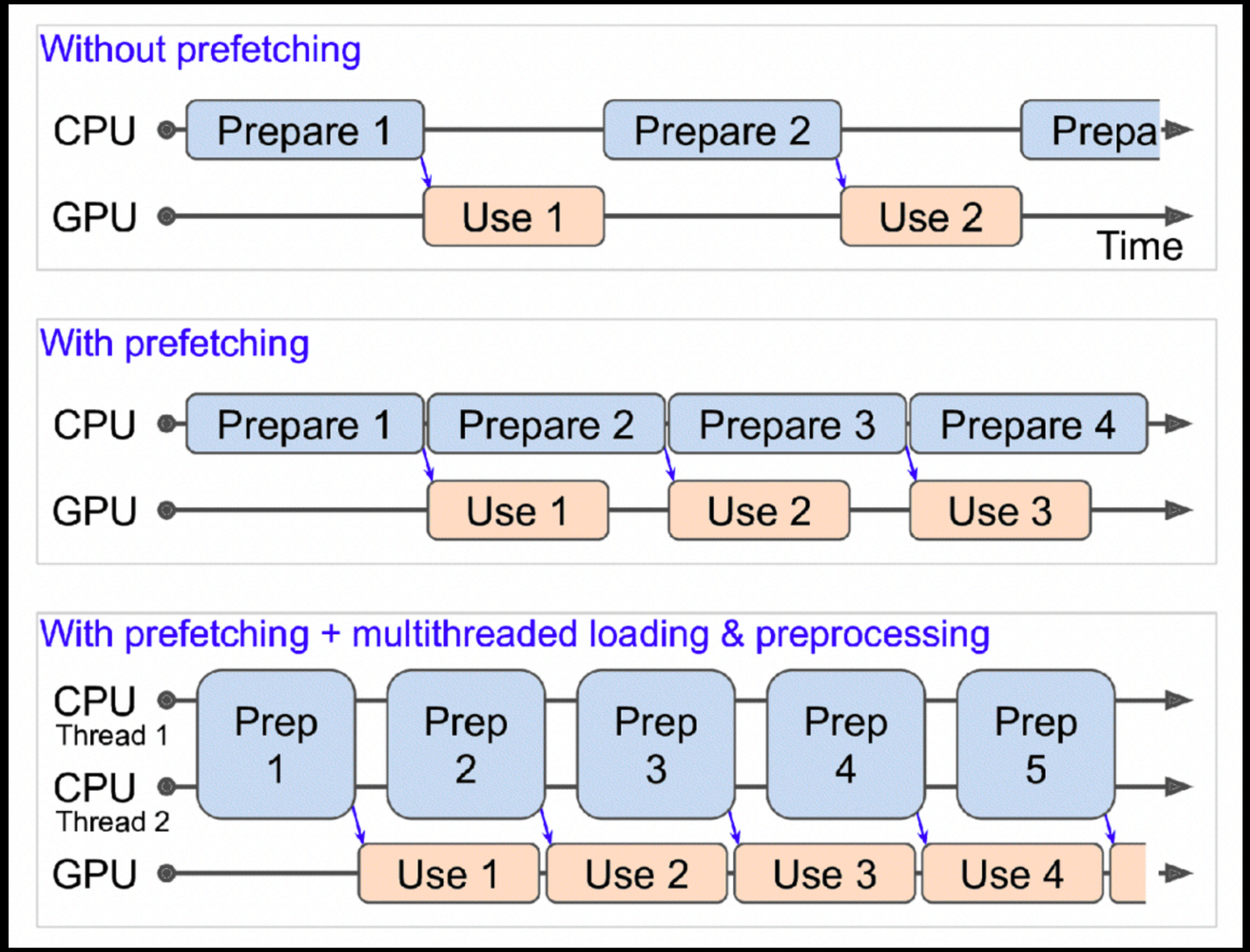
```
import tensorflow_datasets as tfds

# Load in Food101 data from TFDS (takes about 5-6 minutes in Google Colab)
(train_data, test_data), ds_info = tfds.load(name="food101",
                                             split=["train", "validation"],
                                             shuffle_files=True,
                                             as_supervised=True,
                                             with_info=True)
```

Batching & preparing datasets

```
# Map preprocessing function to training data (and paralellize)
train_data = train_data.map(map_func=preprocess, 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)
```



Prefetching



Source: Page 422 of Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow Book by Aurélien Géron

What are callbacks?

- Callbacks are a tool which can add helpful functionality to your models during training, evaluation or inference
- Some popular callbacks include:

Callback name	Use case	Code
TensorBoard	Log the performance of multiple models and then view and compare these models in a visual way on TensorBoard (a dashboard for inspecting neural network parameters). Helpful to compare the results of different models on your data.	`tf.keras.callbacks.TensorBoard()`
Model checkpointing	Save your model as it trains so you can stop training if needed and come back to continue off where you left. Helpful if training takes a long time and can't be done in one sitting.	`tf.keras.callbacks.ModelCheckpoint()`
Early stopping	Leave your model training for an arbitrary amount of time and have it stop training automatically when it ceases to improve. Helpful when you've got a large dataset and don't know how long training will take.	`tf.keras.callbacks.EarlyStopping()`

What is mixed precision training?

Mixed precision training uses the combination of float32 and float16 data types to speed up model training (can result in a 3x speed up on modern GPUs).

No mixed precision

With mixed precision*

*3x speedup = maybe

Before and after mixed precision

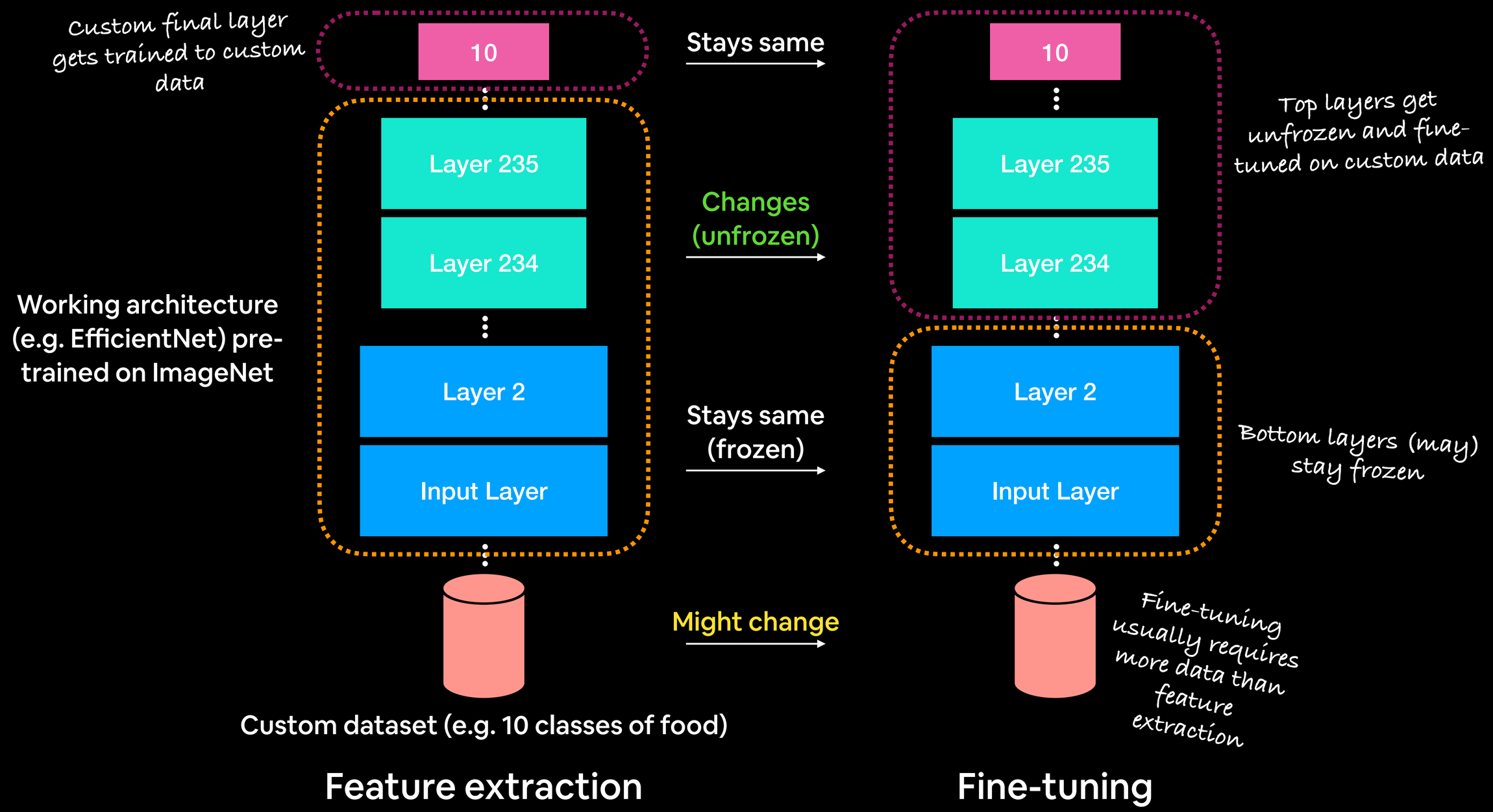
```
import tensorflow as tf
from tensorflow.keras import layers

# Download base model and freeze underlying layers
base_model = tf.keras.applications.EfficientNetB0(include_top=False)
base_model.trainable = False

# Create functional model
inputs = layers.Input(shape=INPUT_SHAPE)
x = base_model(inputs, training=False)
x = layers.GlobalAveragePooling2D()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)

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

No mixed precision

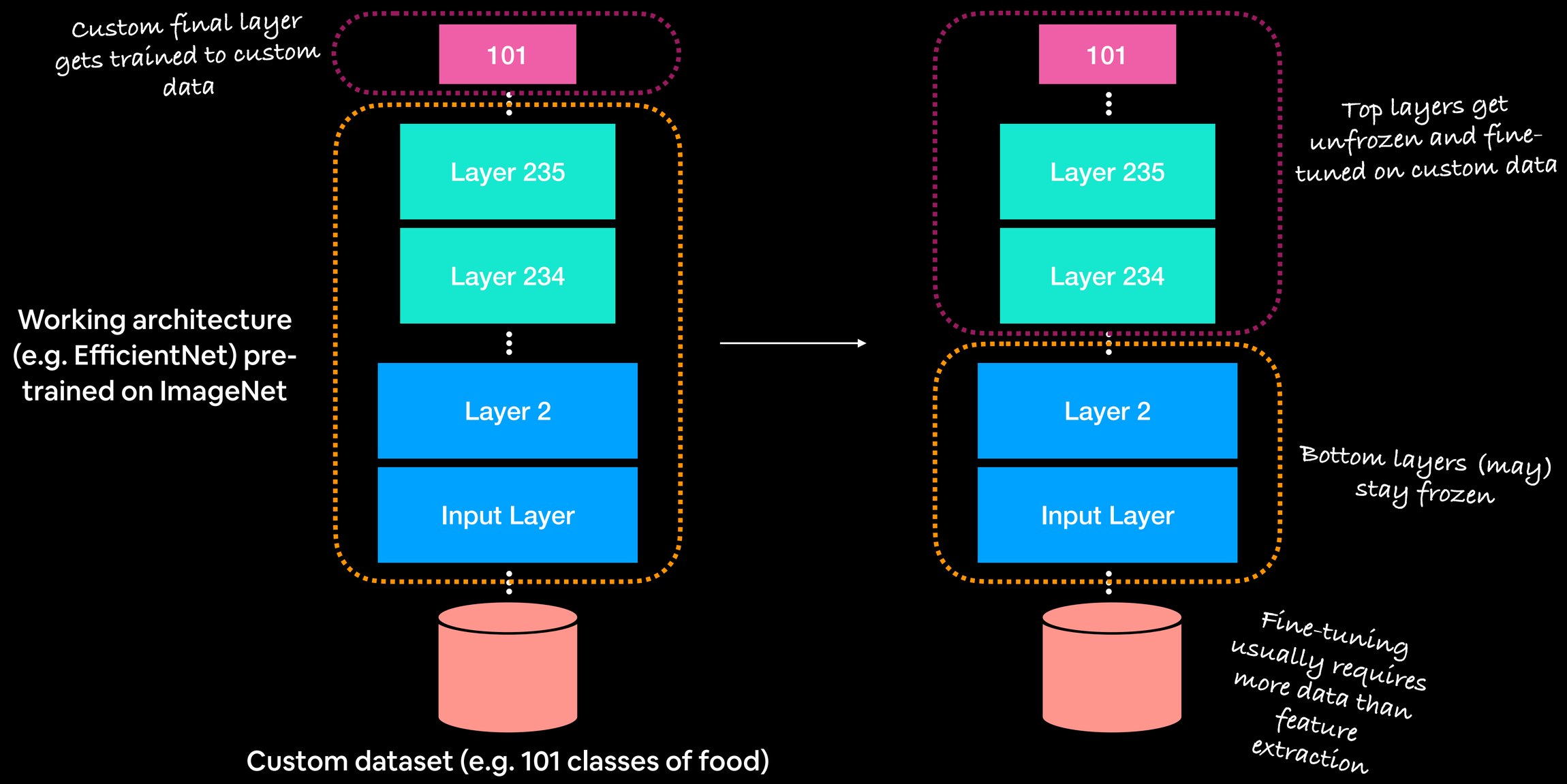
```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import mixed_precision  1

# Turn on mixed precision training
mixed_precision.set_global_policy("mixed_float16")  2

# Download base model and freeze underlying layers
base_model = tf.keras.applications.EfficientNetB0(include_top=False)
base_model.trainable = False

# Create functional model
inputs = layers.Input(shape=INPUT_SHAPE)
x = base_model(inputs, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(10)(x)
# Mixed precision requires output layer to have dtype=tf.float32
outputs = layers.Activation("softmax", dtype=tf.float32)(x)  3
model = tf.keras.Model(inputs, outputs)

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

With mixed precision

Feature extraction vs. Fine-tuning



Feature extraction -> Fine-tuning



Performance of fast data fetching vs non-data fetching

	Prefetch and mixed precision	No prefetch and no mixed precision
Time per epoch	~280-300s	~1127-1397s

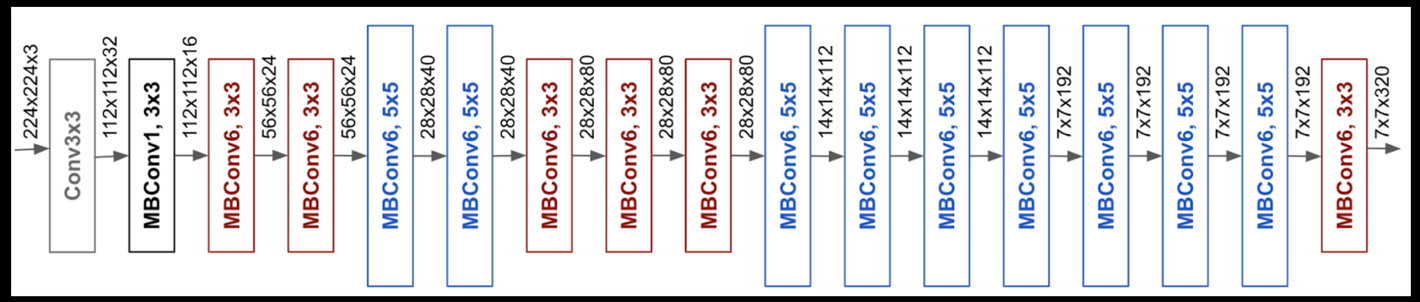
Results from fine-tuning 🍔👁️ Food Vision Big™ on Food101 dataset using an EfficienetNetB0 backbone using a Google Colab Tesla T4 GPU.

Model we've created

(for mixed precision training)

Input data

Model

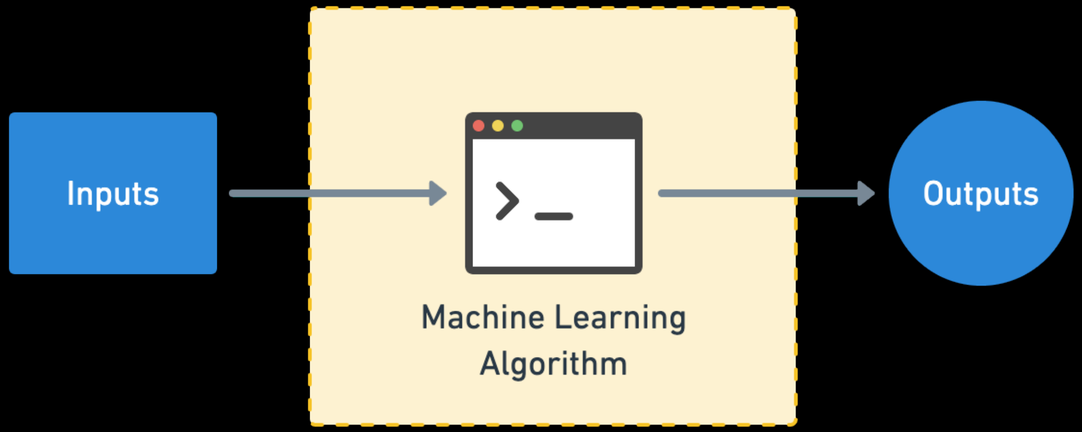


EfficientNetB0 architecture.
Source: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

Output

Softmax activation with dtype=float32

Fully-connected (dense) classifier layer



(some common)

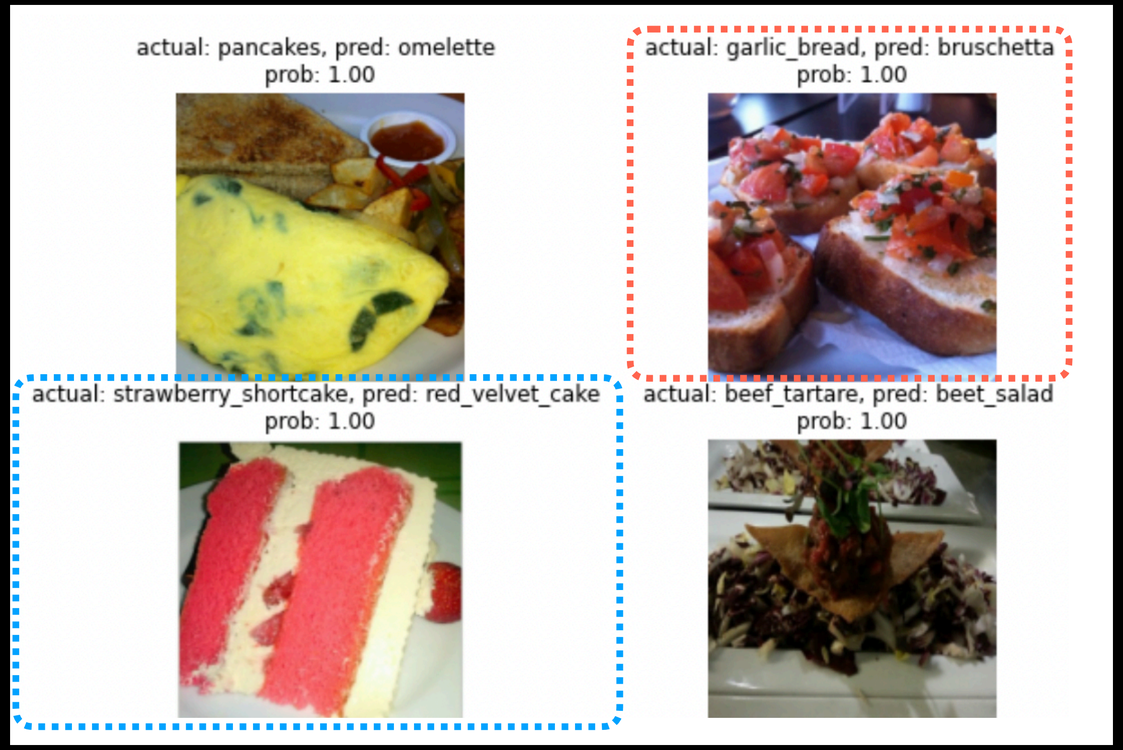
Classification evaluation methods

Key: **tp** = True Positive, **tn** = True Negative, **fp** = False Positive, **fn** = False Negative

Metric Name	Metric Forumla	Code	When to use
Accuracy	$\textbf{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$	`tf.keras.metrics.Accuracy()` or `sklearn.metrics.accuracy_score()`	Default metric for classification problems. Not the best for imbalanced classes.
Precision	$\textbf{Precision} = \frac{tp}{tp + fp}$	`tf.keras.metrics.Precision()` or `sklearn.metrics.precision_score()`	Higher precision leads to less false positives.
Recall	$\textbf{Recall} = \frac{tp}{tp + fn}$	`tf.keras.metrics.Recall()` or `sklearn.metrics.recall_score()`	Higher recall leads to less false negatives.
F1-score	$\textbf{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$	`sklearn.metrics.f1_score()`	Combination of precision and recall, usually a good overall metric for a classification model.
Confusion matrix	NA	`Custom function` or `sklearn.metrics.confusion_matrix()`	When comparing predictions to truth labels to see where model gets confused. Can be hard to use with large numbers of classes.

Finding the most wrong predictions

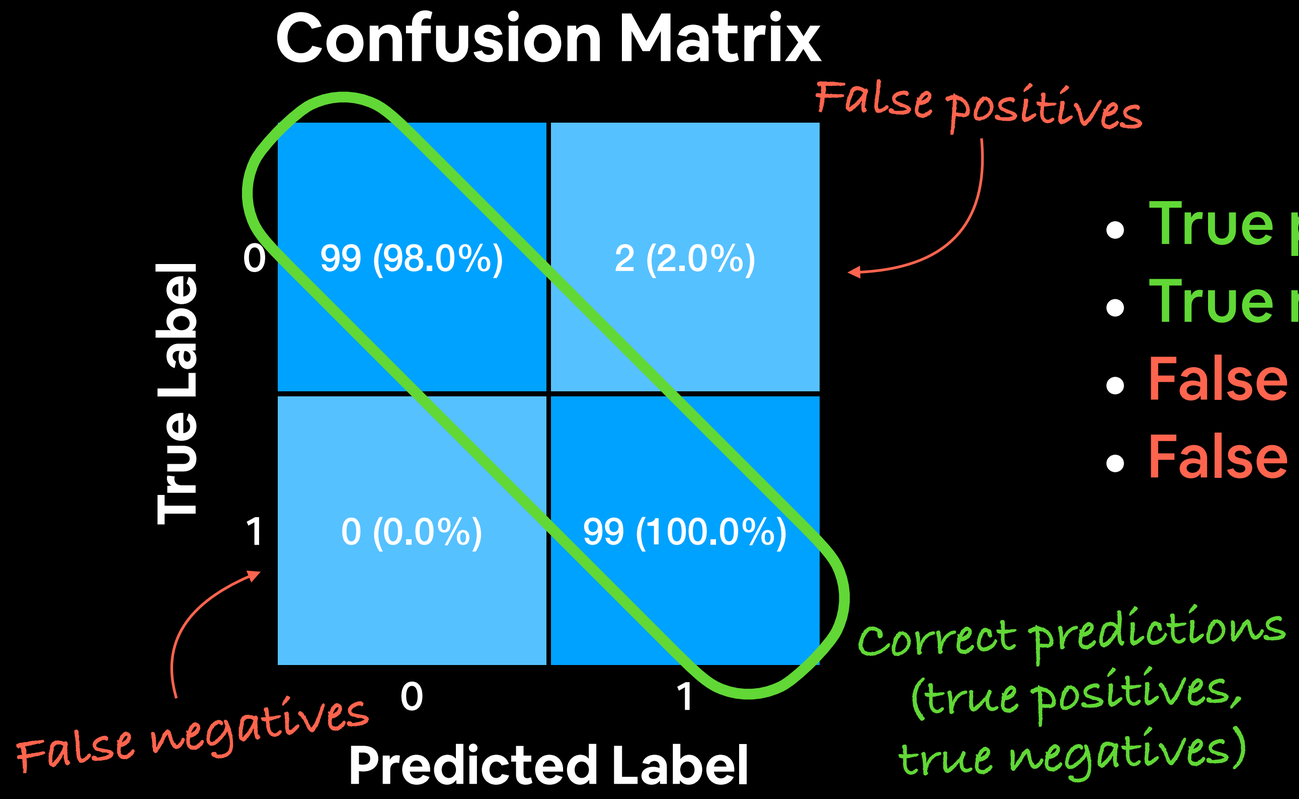
- A good way to inspect your model's performance is to view the wrong predictions with the highest prediction probability (or highest loss)
- Can reveal insights such as:
  - Data issues (wrong labels, e.g. model is right, label is wrong)
  - Confusing classes (get better/more diverse data)



Wrong label?

Confusing class

Anatomy of a confusion matrix



- True positive = model predicts 1 when truth is 1
- True negative = model predicts 0 when truth is 0
- False positive = model predicts 1 when truth is 0
- False negative = model predicts 0 when truth is 1