Transfer Learning with

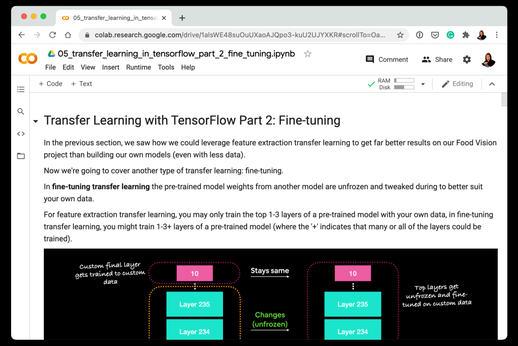
TensorFlow

Part 2: Fine-tuning

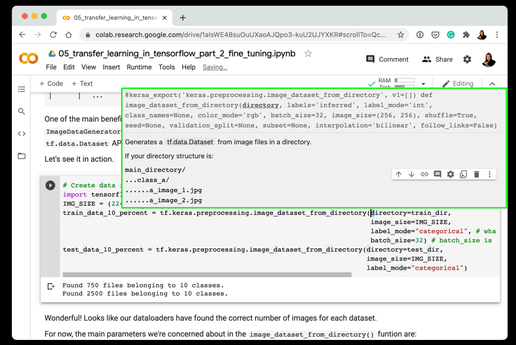
Where can you get help?

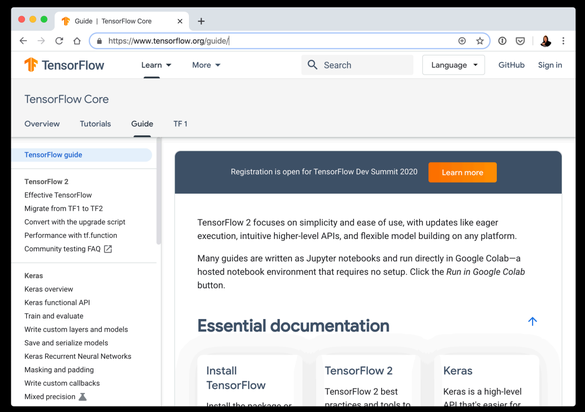
- Follow along with the code
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring
- Search for it
- Try again
- Ask (don't forget the Discord chat!)

(yes, including the "dumb" questions)



"If in doubt, run the code"



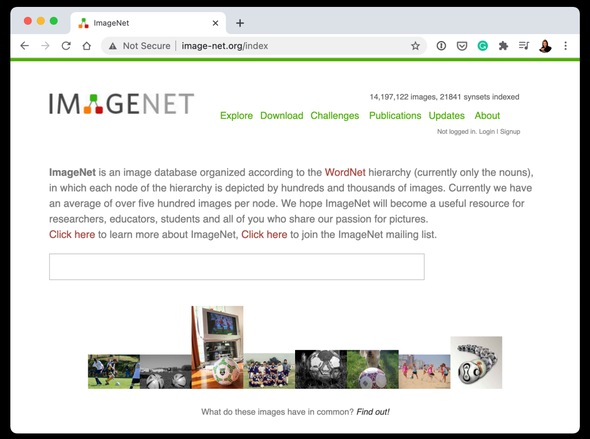


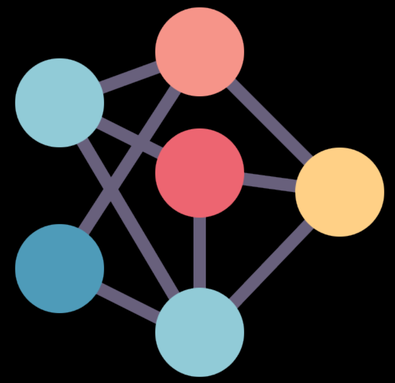
“What is transfer learning?”

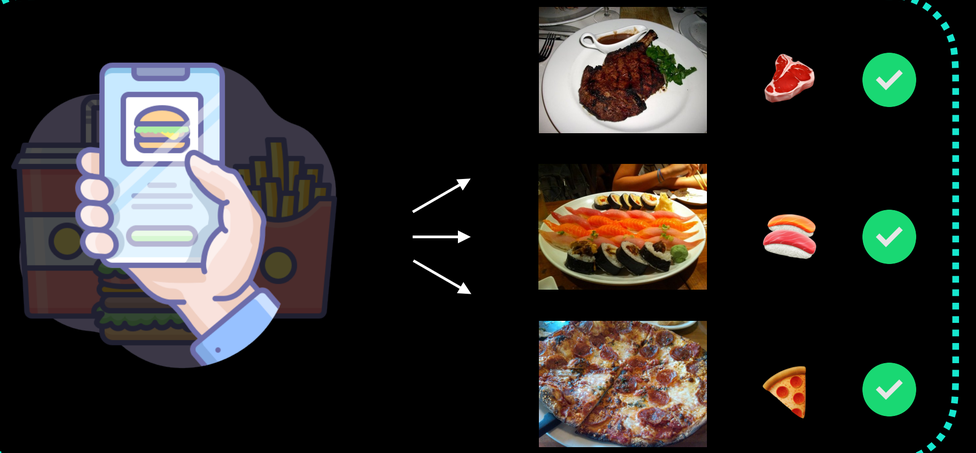
Surely someone has spent the time crafting the right model for the job...

Example transfer learning use cases

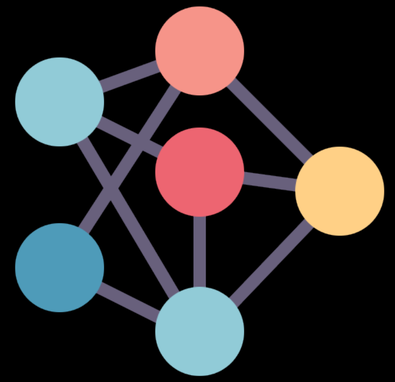
Computer vision

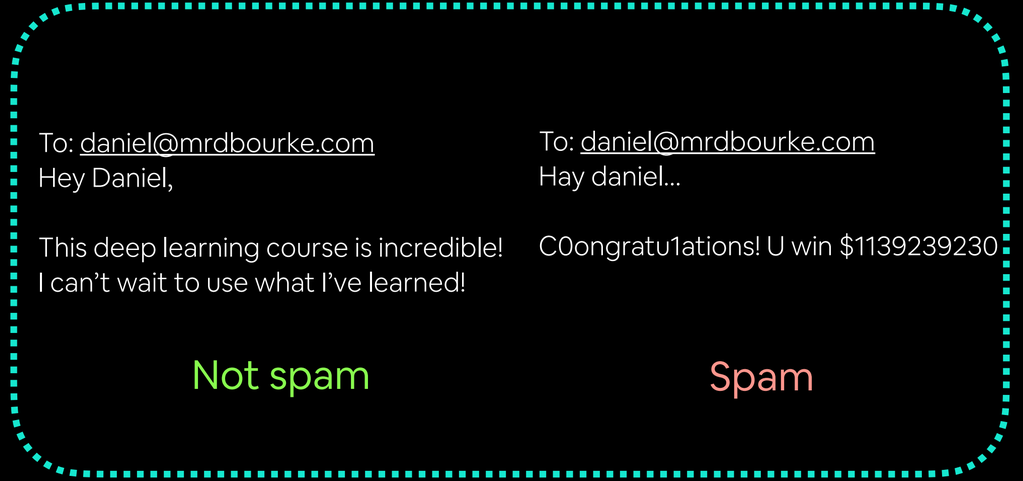






Natural language processing





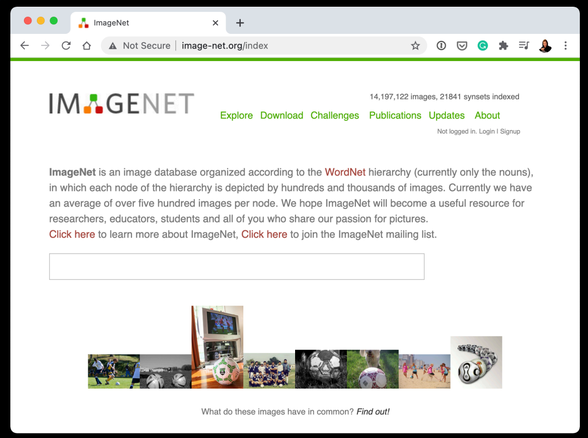
Model learns patterns/weights from similar problem space

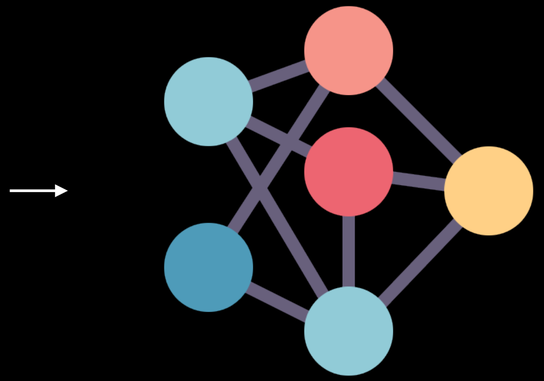
Patterns get used/tuned to specific problem

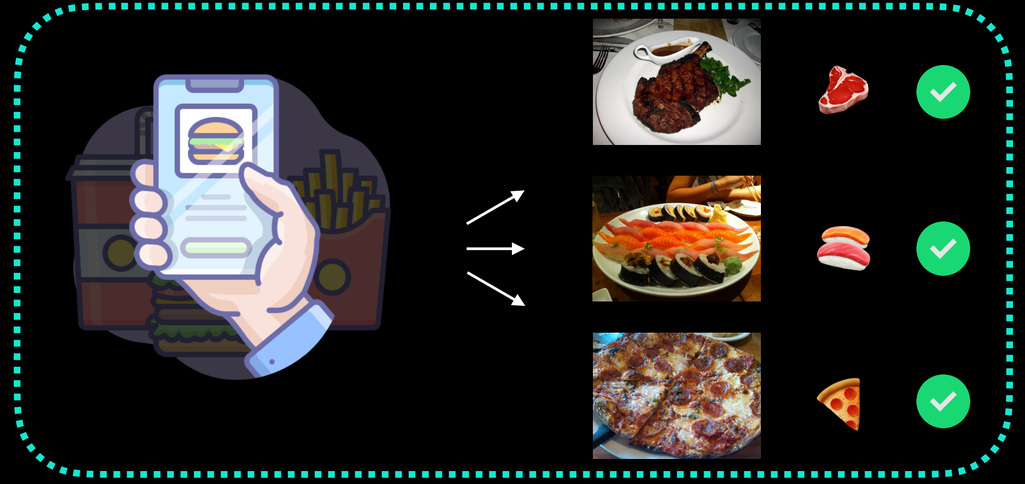
“Why use transfer learning?”

Why use transfer learning?

- Can leverage an existing neural network architecture proven to work on problems similar to our own
- Can leverage a working network architecture which has already learned patterns on similar data to our own (often results in great results with less data)







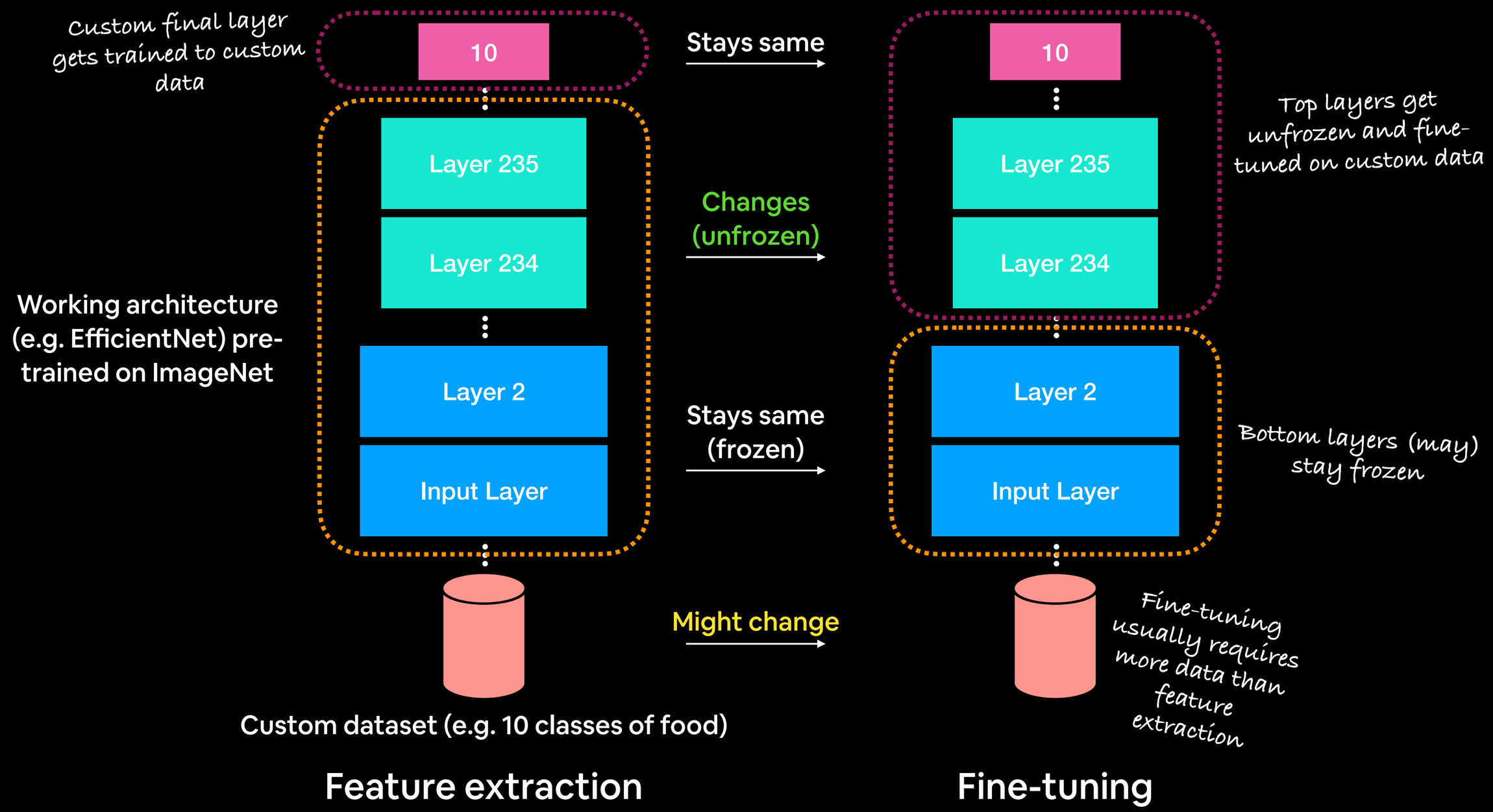
Learn patterns in a wide variety of images (using ImageNet)

EfficientNet archiecture (already works really well on computer vision tasks)

Tune patterns/weights to our own problem (Food Vision)

Model performs better than from scratch

Feature extraction vs. Fine-tuning



What we're going to cover

(broadly)

- Introduce fine-tuning transfer learning with TensorFlow
- Introduce the Keras Functional API to build models
- Using a small dataset to experiment faster (e.g. 10% of training samples)
- Data augmentation (making your training set more diverse without adding samples)
- Running a series of experiments on our Food Vision data
- Introduce the ModelCheckpoint callback to save intermediate training results

(we'll be cooking up lots of code!)

How:

Let’s code!

Dataset shapes

"Create batches of 32 images of size 224x224 split into red, green, blue colour channels."

```
train_data_10_percent = tf.keras.preprocessing.image_dataset_from_directory(directory=train_dir,
                                                                            image_size=(224, 224),
                                                                            label_mode="categorical",
                                                                            batch_size=32)

train_data_10_percent
```

```
Found 750 files belonging to 10 classes.
<BatchDataset shapes: ((None, 224, 224, 3), (None, 10)), types: (tf.float32, tf.float32)>
```

- Number of total samples (750 images, 75 per class)
- Number of classes (10 types of food)
- Batch size (default is 32)
- Image size (height, width)
- Number of colour channels (red, green, blue)
- Number of classes in label tensors (10 types of food)

Keras Sequential vs Functional API

Sequential API

```python
# Creating a model with the Sequential API
sequential_model = tf.keras.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation="relu"),
  tf.keras.layers.Dense(64, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
], name="sequential_model")

sequential_model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(),
    metrics=["accuracy"]
)

sequential_model.fit(x_train, y_train,
                     batch_size=32,
                     epochs=5)
```

Functional API

```python
# Creating a model with the Functional API
inputs = tf.keras.layers.Input(shape=(28, 28))
x = tf.keras.layers.Flatten()(inputs)
x = tf.keras.layers.Dense(64, activation="relu")(x)
x = tf.keras.layers.Dense(64, activation="relu")(x)
outputs = tf.keras.layers.Dense(10, activation="softmax")(x)
functional_model = tf.keras.Model(inputs, outputs, name="functional_model")

functional_model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(),
    metrics=["accuracy"]
)

functional_model.fit(x_train, y_train,
                     batch_size=32,
                     epochs=5)
```

compiling and fitting stays the same

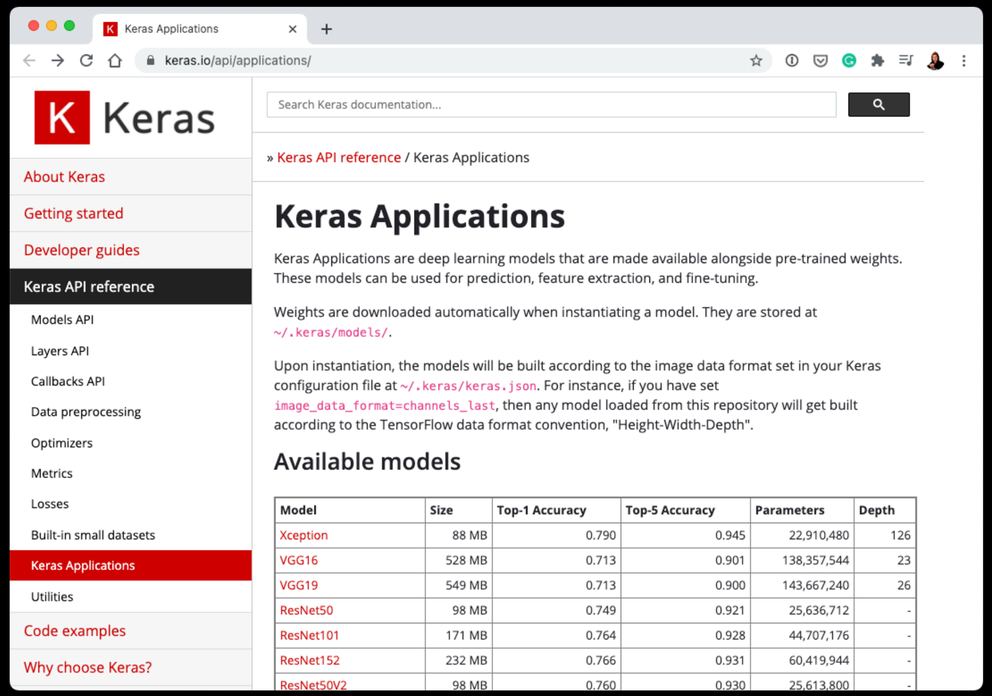
- Similarities: compiling, fitting, evaluating
- Differences: model construction (the Functional API is more flexible and able to produce more sophisticated models)

Building a feature extraction model with the Keras Functional API

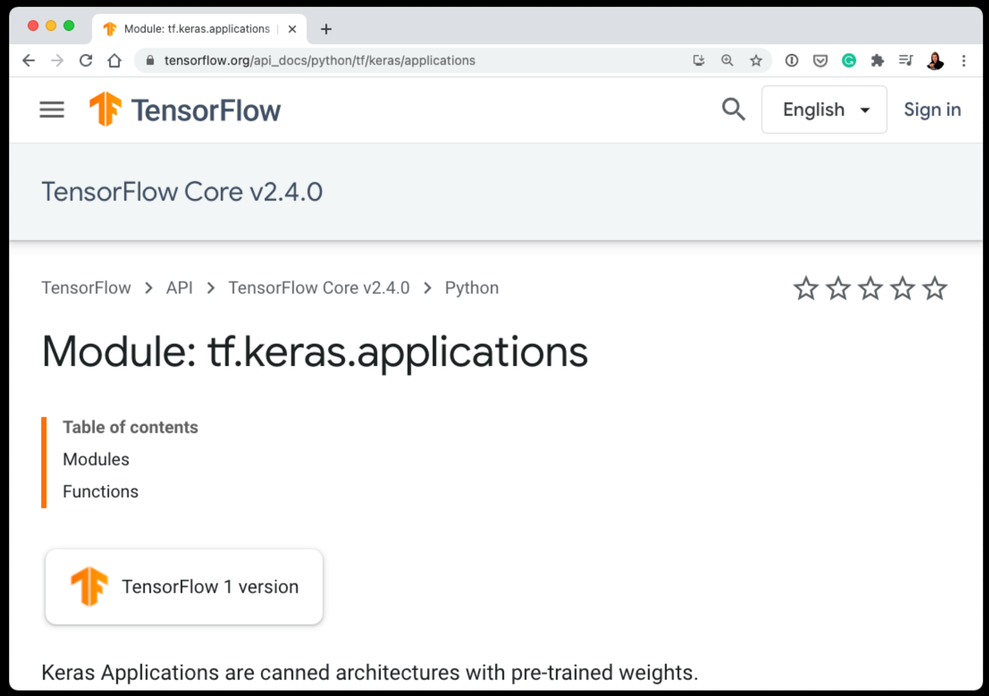
```python
# 1. Create base model with tf.keras.applications
base_model = tf.keras.applications.EfficientNetB0(include_top=False)

# 2. Freeze the base model (so the pre-learned patterns remain)
base_model.trainable = False

# 3. Create inputs into the base model
inputs = tf.keras.layers.Input(shape=(224, 224, 3), name="input_layer")

# 4. If using ResNet50V2, add this to speed up convergence, remove for EfficientNet
# x = tf.keras.layers.experimental.preprocessing.Rescaling(1./255)(inputs)

# 5. Pass the inputs to the base_model (note: using tf.keras.applications, EfficientNet inputs don't have to be normalized)
x = base_model(inputs)
# Check data shape after passing it to base_model
print(f"Shape after base_model: {x.shape}")

# 6. Average pool the outputs of the base model (aggregate all the most important information)
x = tf.keras.layers.GlobalAveragePooling2D(name="global_average_pooling_layer")(x)
print(f"After GlobalAveragePooling2D(): {x.shape}")

# 7. Create the output activation layer
outputs = tf.keras.layers.Dense(10, activation="softmax", name="output_layer")(x)

# 8. Combine the inputs with the outputs into a model
model_0 = tf.keras.Model(inputs, outputs)

# 9. Compile the model
model_0.compile(loss='categorical_crossentropy',
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])

# 10. Fit the model (we use less steps for validation so it's faster)
history_10_percent = model_0.fit(train_data_10_percent,
                                 epochs=5,
                                 steps_per_epoch=len(train_data_10_percent),
                                 validation_data=test_data_10_percent,
                                 # Go through less of the validation data so epochs are faster (we want faster experiments!)
                                 validation_steps=int(0.25 * len(test_data_10_percent)),
                                 # Track our model's training logs for visualization later
                                 callbacks=[create_tensorboard_callback("transfer_learning", "10_percent_feature_extract")])
```

Useful computer vision architectures

- `tf.keras.applications` **and** `keras.applications` **have many of the most popular and best performing computer vision architectures built-in & pre-trained, ready to use for your own problems**



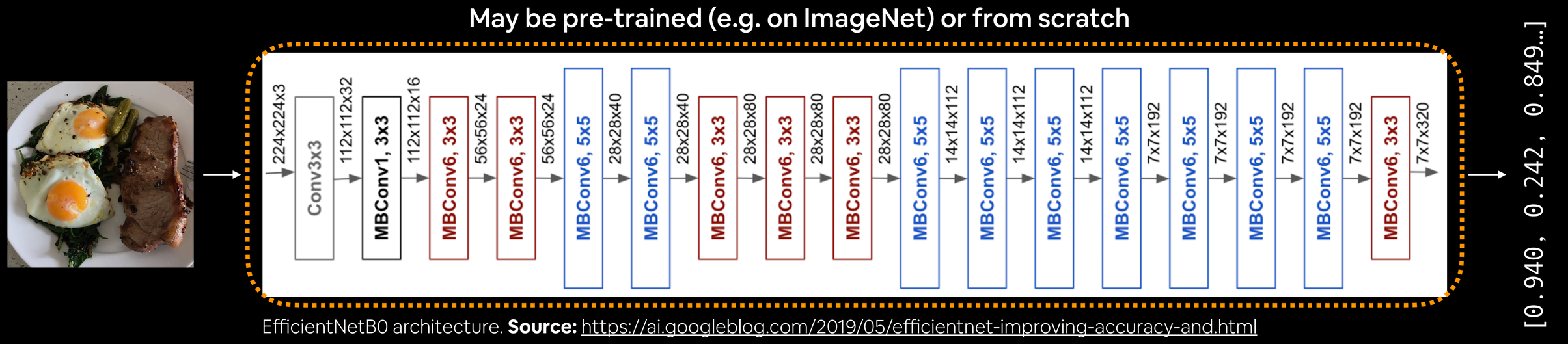
Source: https://keras.io/api/applications/



Source: https://www.tensorflow.org/api_docs/python/tf/keras/applications

What is a feature vector?

- A feature vector is a learned representation of the input data (a compressed form of the input data based on how the model see's it)



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

Input data

Model

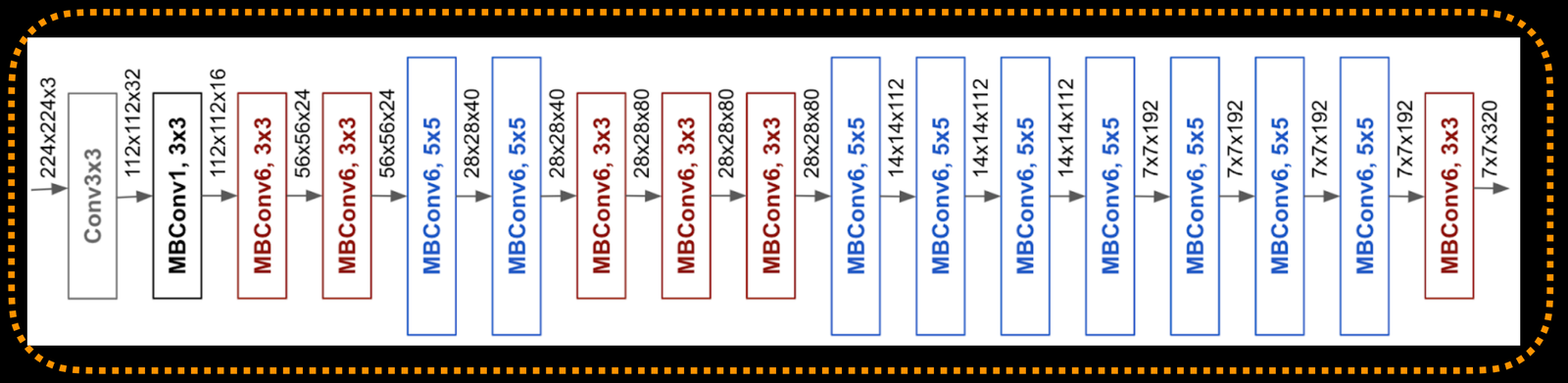
Learns feature representation of input data

Output (feature vector)

EfficientNet feature extractor

Input data
(10 classes of Food101)

Stays same
(frozen, pre-trained on ImageNet)



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

Changes
(same shape
as number
of classes)

Pooling

[0.940, 0.242, 0.849...]

10

Fully-connected
(dense) classifier layer

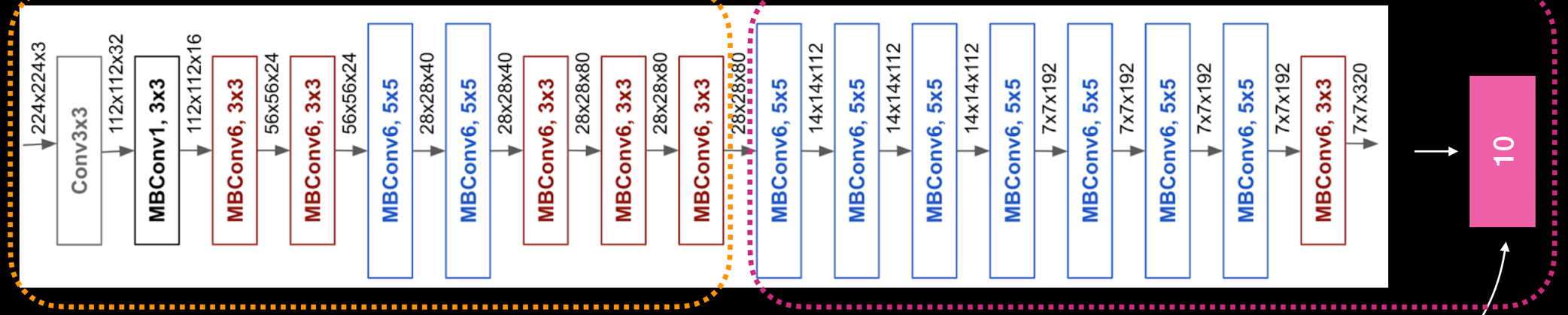
EfficientNet fine-tuning

Input data
(10 classes of Food101)

Stays same
(frozen, pre-trained on ImageNet)

Layers closer to the output layer get unfrozen/fine-tuned first

Changes
(unfrozen, gets fine-tuned on custom data)



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

Bottom layers tend to stay frozen (or are last to get unfrozen)

Fully-connected (dense) classifier layer

Modelling experiments we're running

Experiment	Data	Preprocessing	Model
Model 0 (baseline)	10 classes of Food101 data (**random 10%** training data only)	None	**Feature Extractor:** EfficientNetB0 (pre-trained on ImageNet, all layers frozen) with no top
Model 1	10 classes of Food101 data (**random 1%** training data only)	Random Flip, Rotation, Zoom, Height, Width data augmentation	Same as Model 0
Model 2	Same as Model 0	Same as Model 1	Same as Model 0
Model 3	Same as Model 0	Same as Model 1	**Fine-tuning:** Model 2 (EfficientNetB0 pre-trained on ImageNet) **with top layer trained on custom data, top 10 layers unfrozen**
Model 4	10 classes of Food101 data (**100%** training data)	Same as Model 1	Same as Model 3

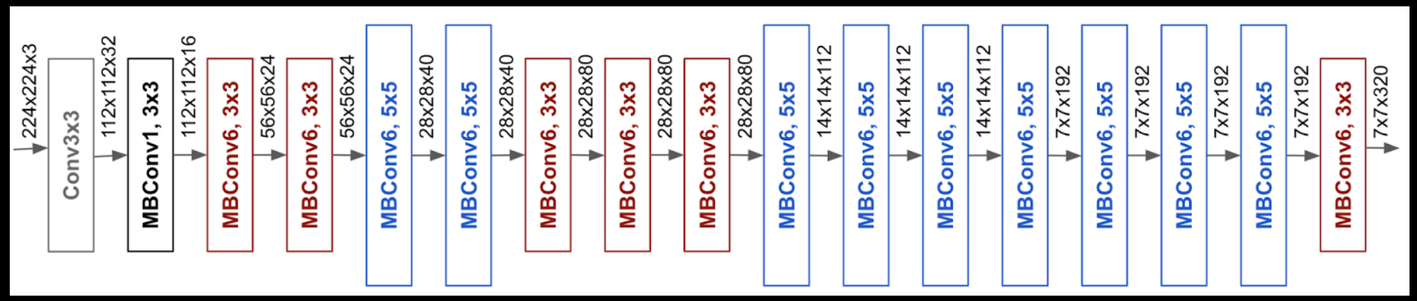
🍔👁 Food Vision: Dataset(s) we're using

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 original Food101 dataset)	250 images of each class (same as original Food101 dataset)

Data augmentation as a layer

Data augmentation



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

```
[0.940, 0.242, 0.849...]
```

Input data

Model

Output (feature vector)

```
# Build data augmentation layer
data_augmentation = tf.keras.Sequential([
  preprocessing.RandomFlip('horizontal'),
  preprocessing.RandomHeight(0.2),
  preprocessing.RandomWidth(0.2),
  preprocessing.RandomZoom(0.2),
  preprocessing.RandomRotation(0.2),
], name="data_augmentation")
```

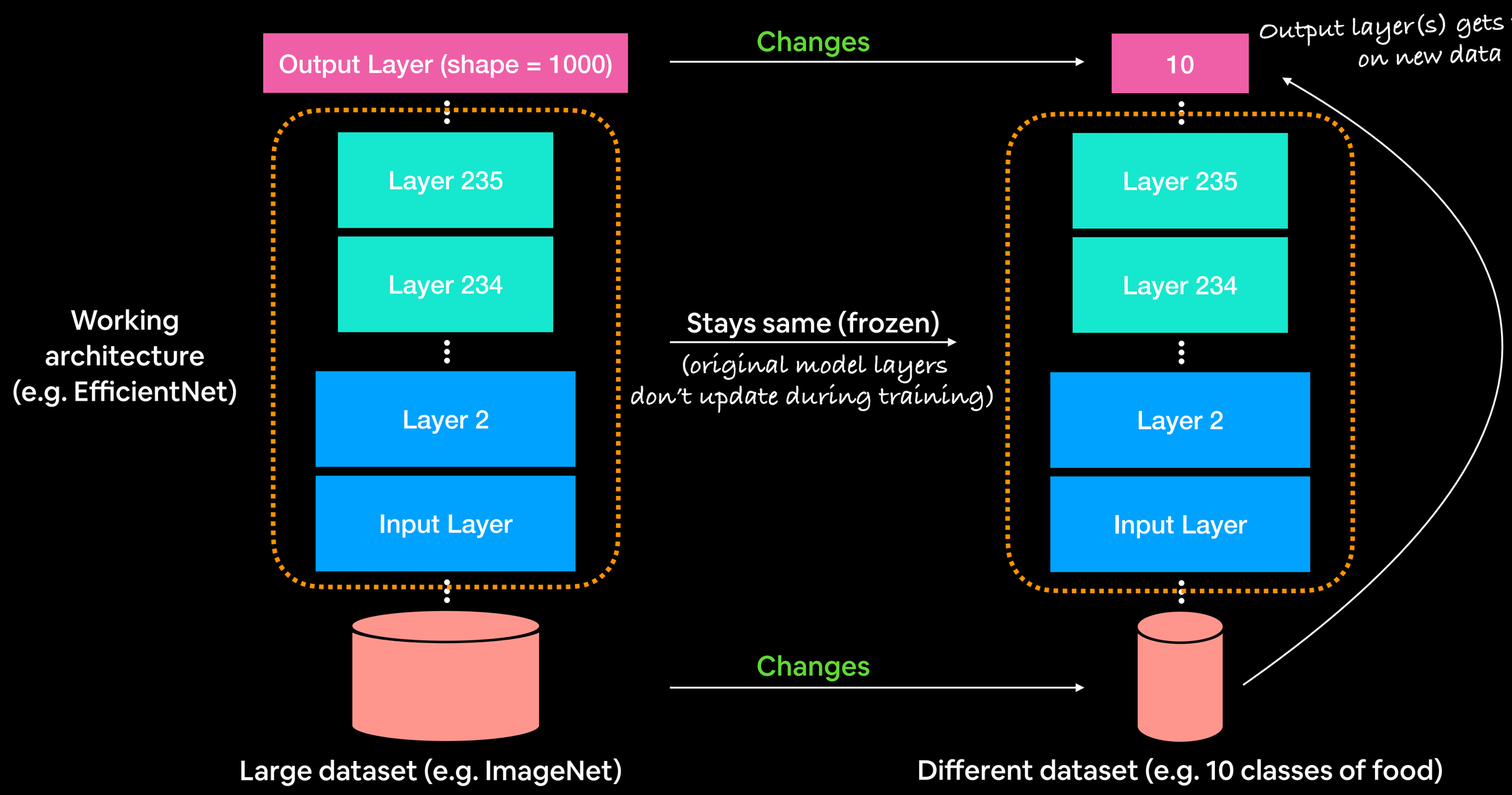
When passed as a layer to a model data augmentation is automatically turned on during training (augments training data) but turned off during inference (does not augment testing data or new unseen data).

What are callbacks?

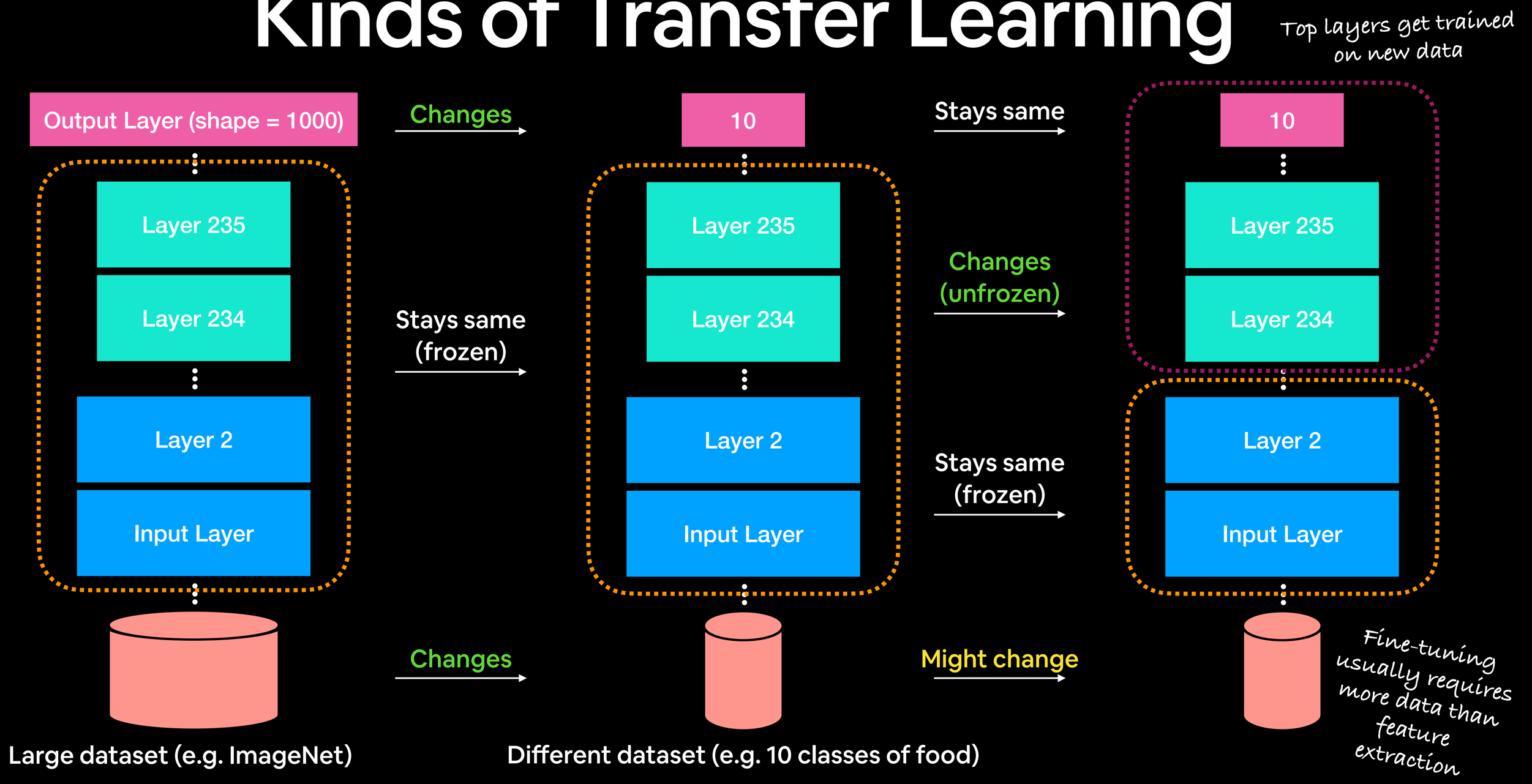
- Callbacks are a tool which can add helpful functionality to your models during training, evaluation or inference
- Some popular callbacks include:

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()`

Original Model vs. Feature Extraction



Kinds of Transfer Learning



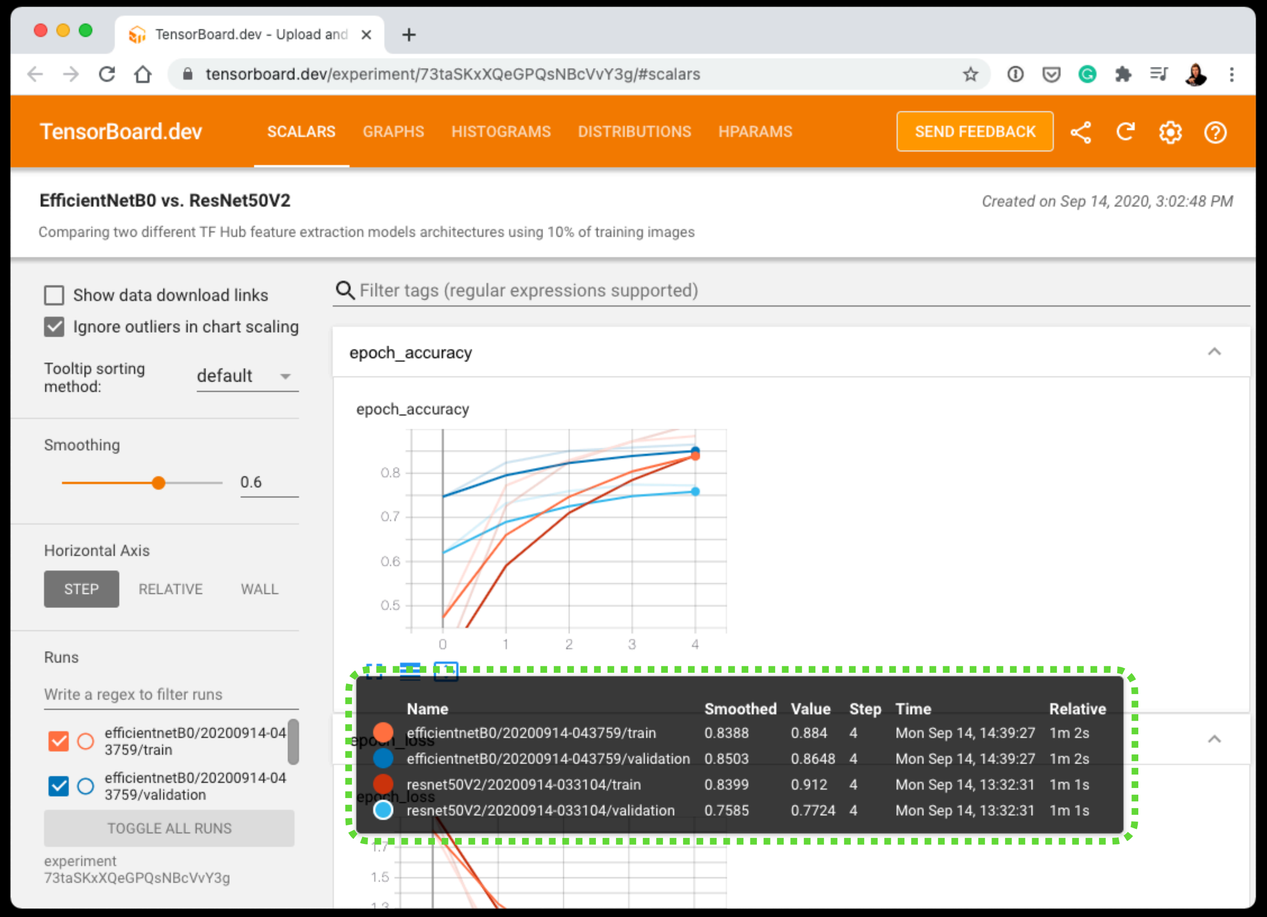
Kinds of Transfer Learning

Transfer Learning Type	Description	What happens	When to use
Original model (“As is”)	Take a pretrained model as it is and apply it to your task without any changes.	The original model **remains unchanged**.	Helpful if you have the **exact same kind of data** the original model was trained on.
Feature extraction	Take the underlying patterns (also called weights) a pretrained model has learned and adjust its outputs to be more suited to your problem.	**Most of the layers** in the original model **remain frozen** during training (only the top 1-3 layers get updated).	Helpful if you have a **small amount of custom data** (similar to what the original model was trained on) and want to utilise a pretrained model to get **better results on your specific problem**.
Fine-tuning	Take the weights of a pretrained model and adjust (fine-tune) them to your own problem.	**Some (1-3+), many or all** of the layers in the pretrained model **are updated** during training.	Helpful if you have a **large amount of custom data** and want to utilise a pretrained model and improve its underlying patterns to your specific problem.

What is TensorBoard?

- A way to visually explore your machine learning models performance and internals
- Host, track and share your machine learning experiments on TensorBoard.dev

(TensorBoard also integrates with websites like Weights & Biases)



Comparing the results of two different model architectures (ResNet50V2 & EfficientNetB0) on the same dataset.

Source: https://tensorboard.dev/experiment/73taSKxXQeGPQsNBcVvY3g/#scalars