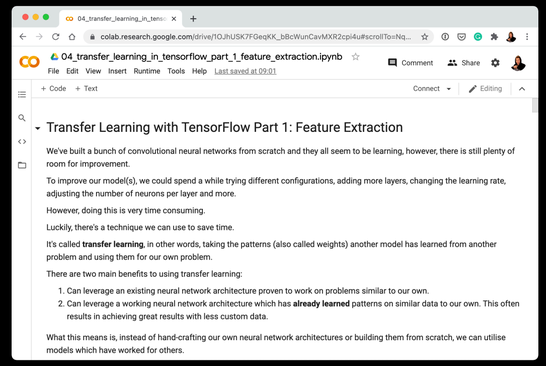
Transfer Learning with

TensorFlow

Part 1: Feature extraction

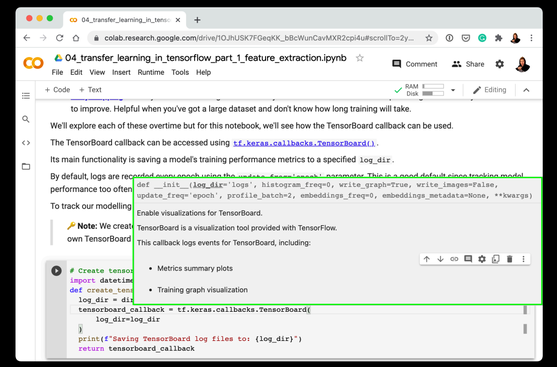
Where can you get help?

- Follow along with the code



"If in doubt, run the code"

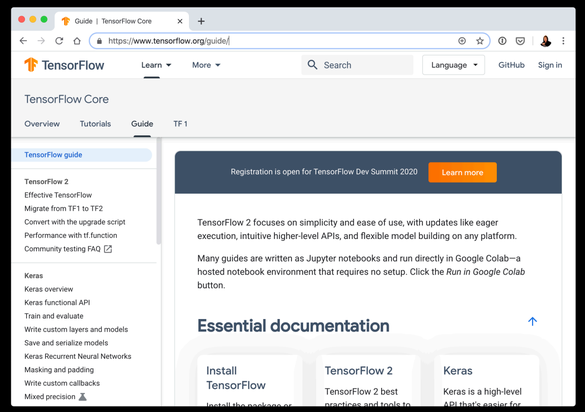
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring



- Search for it

- Try again
- Ask (don't forget the Discord chat!)

(yes, including the "dumb" questions)

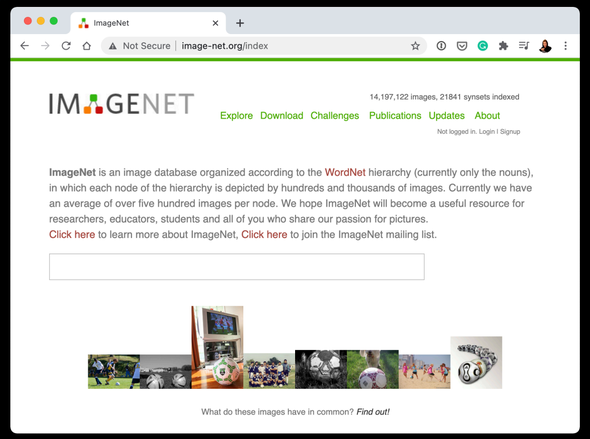


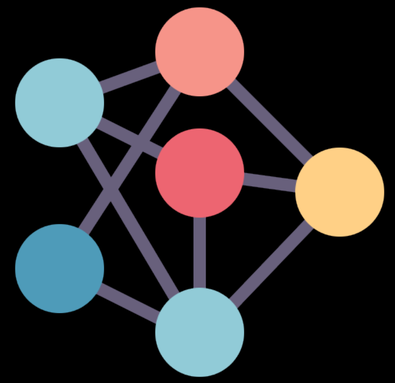
“What is transfer learning?”

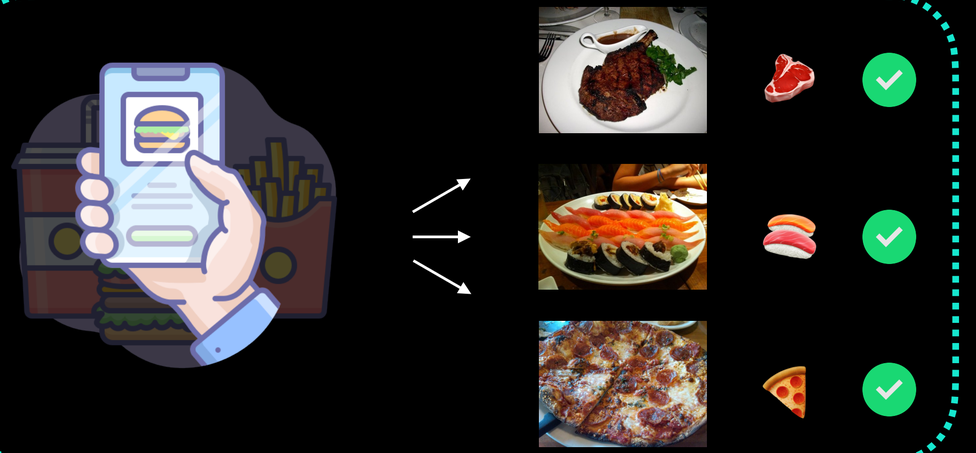
Surely someone has spent the time crafting the right model for the job...

Example transfer learning use cases

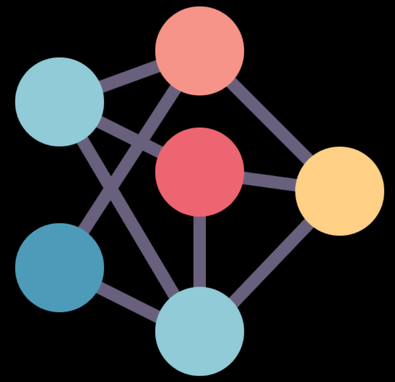
Computer vision

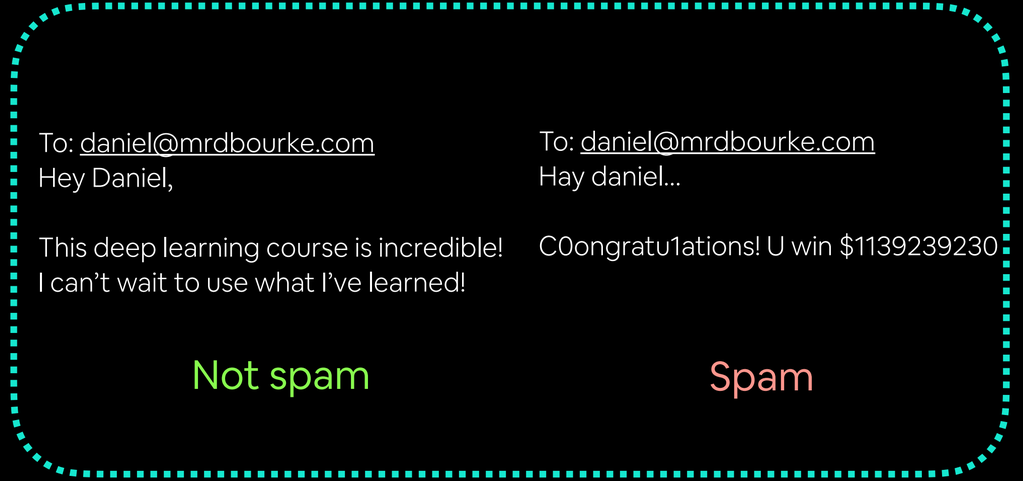






Natural language processing





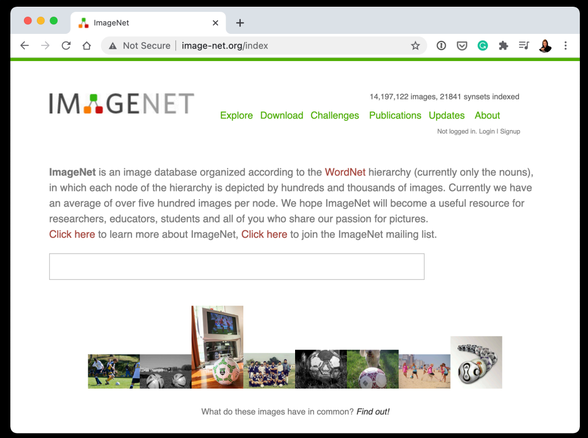
Model learns patterns/weights from similar problem space

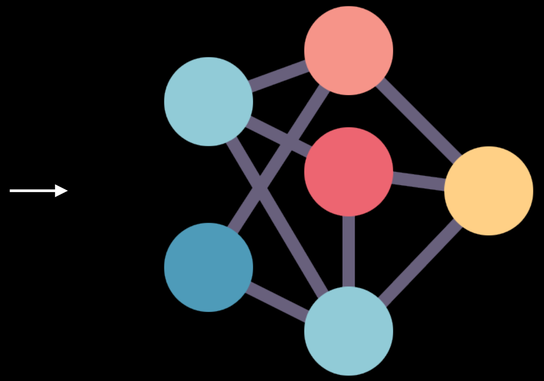
Patterns get used/tuned to specific problem

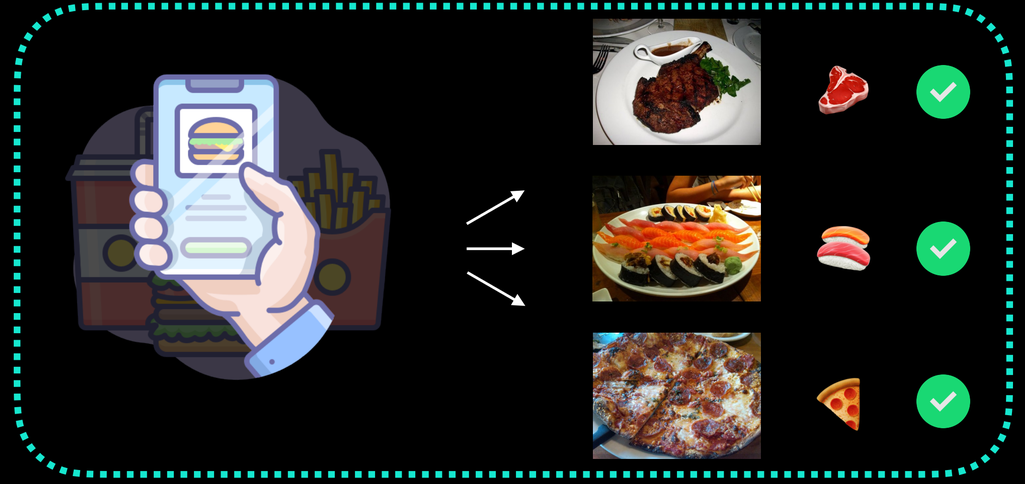
“Why use transfer learning?”

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

What we're going to cover

(broadly)

- Introduce transfer learning with TensorFlow
- Using a small dataset to experiment faster (10% of training samples)
- Building a transfer learning feature extraction model with TensorFlow Hub
- Use TensorBoard to track modelling experiments and results

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

How:

Let’s code!

What are callbacks?

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

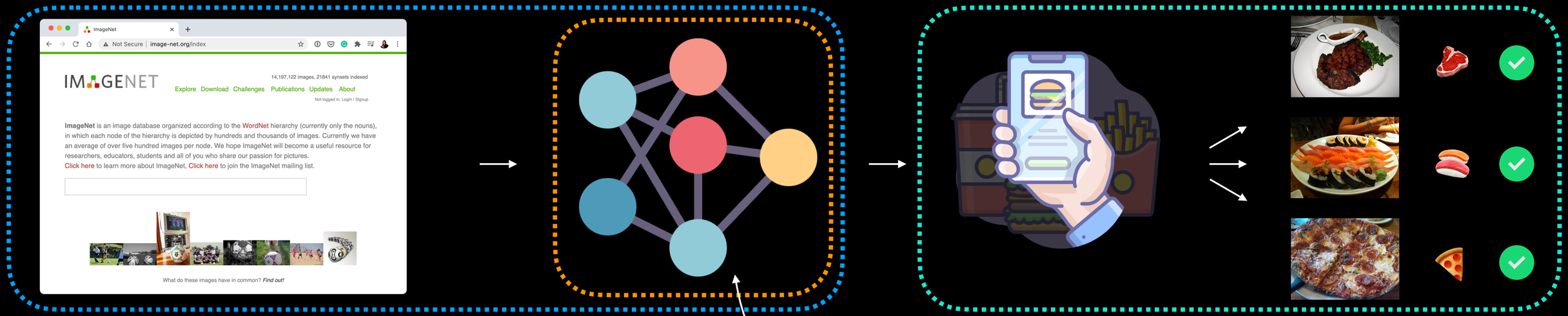
Callback name	Use case	Code
TensorBoard	Log the performance of multiple models and then view and compare these models in a visual way on TensorBoard (a dashboard for inspecting neural network parameters). Helpful to compare the results of different models on your data.	`tf.keras.callbacks.TensorBoard()`
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 TensorFlow Hub?

- A place to find a plethora of pre-trained machine learning models (ready to be applied and fine-tuned for your own problems)

TensorFlow Hub

🤔 "Does my problem exist on TensorFlow Hub?"



https://tfhub.dev/tensorflow/efficientnet/b0/feature-vector/1

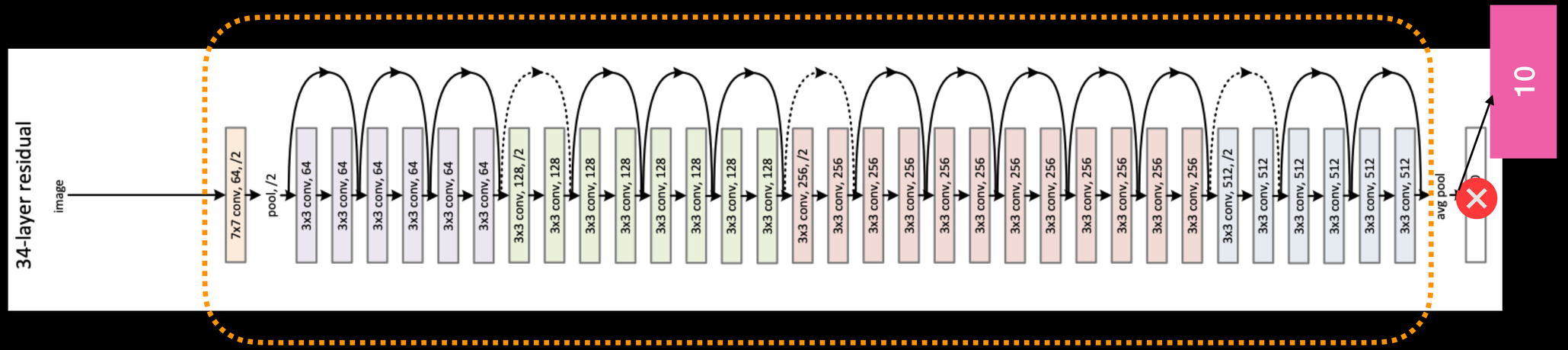
TensorFlow Hub makes using a pre-trained model as simple as calling a URL

ResNet50* feature extractor

Input data
(10 classes of Food101)

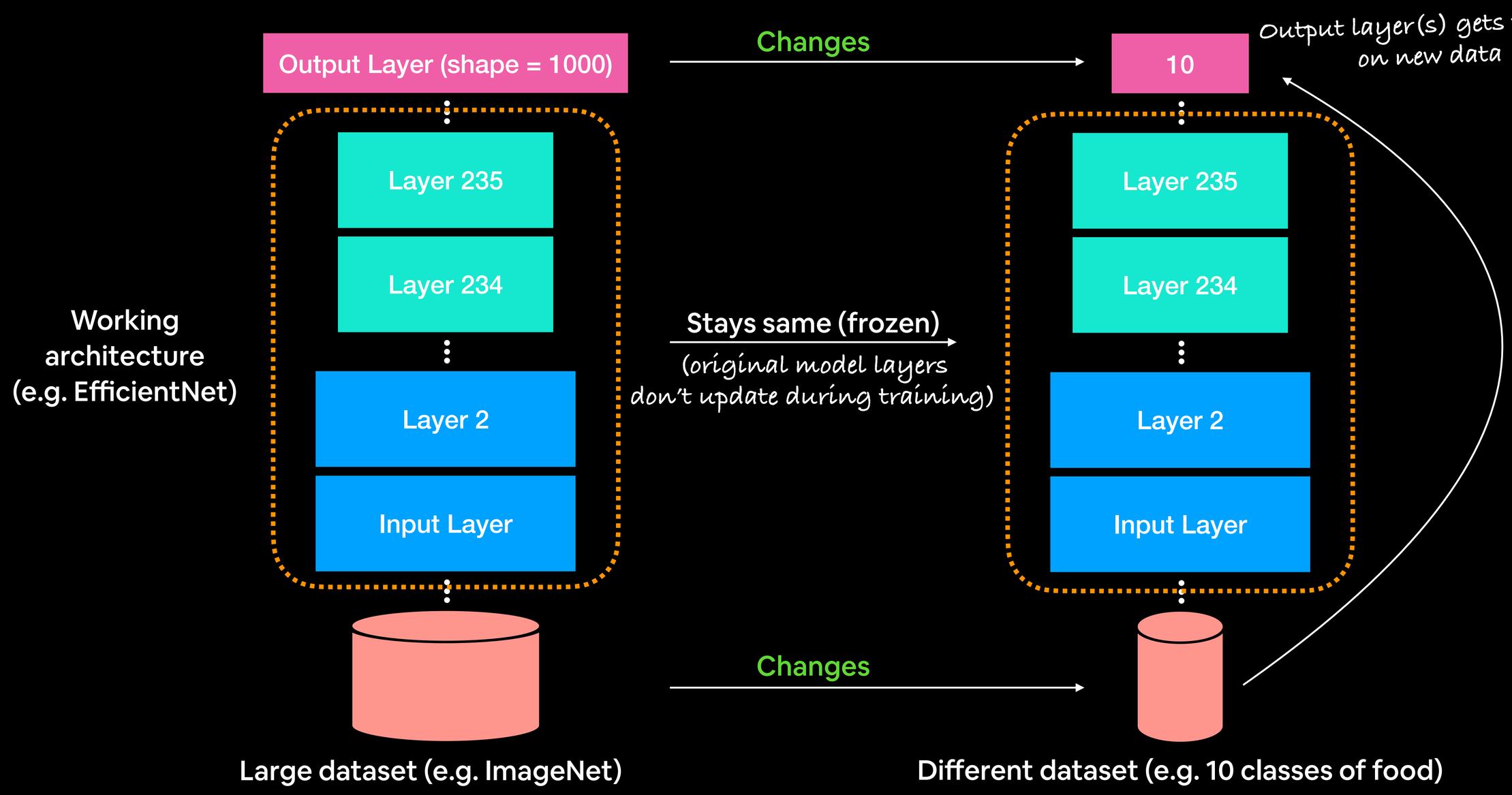
Stays same
(frozen, pre-trained on ImageNet)

Changes
(same shape as number of classes)

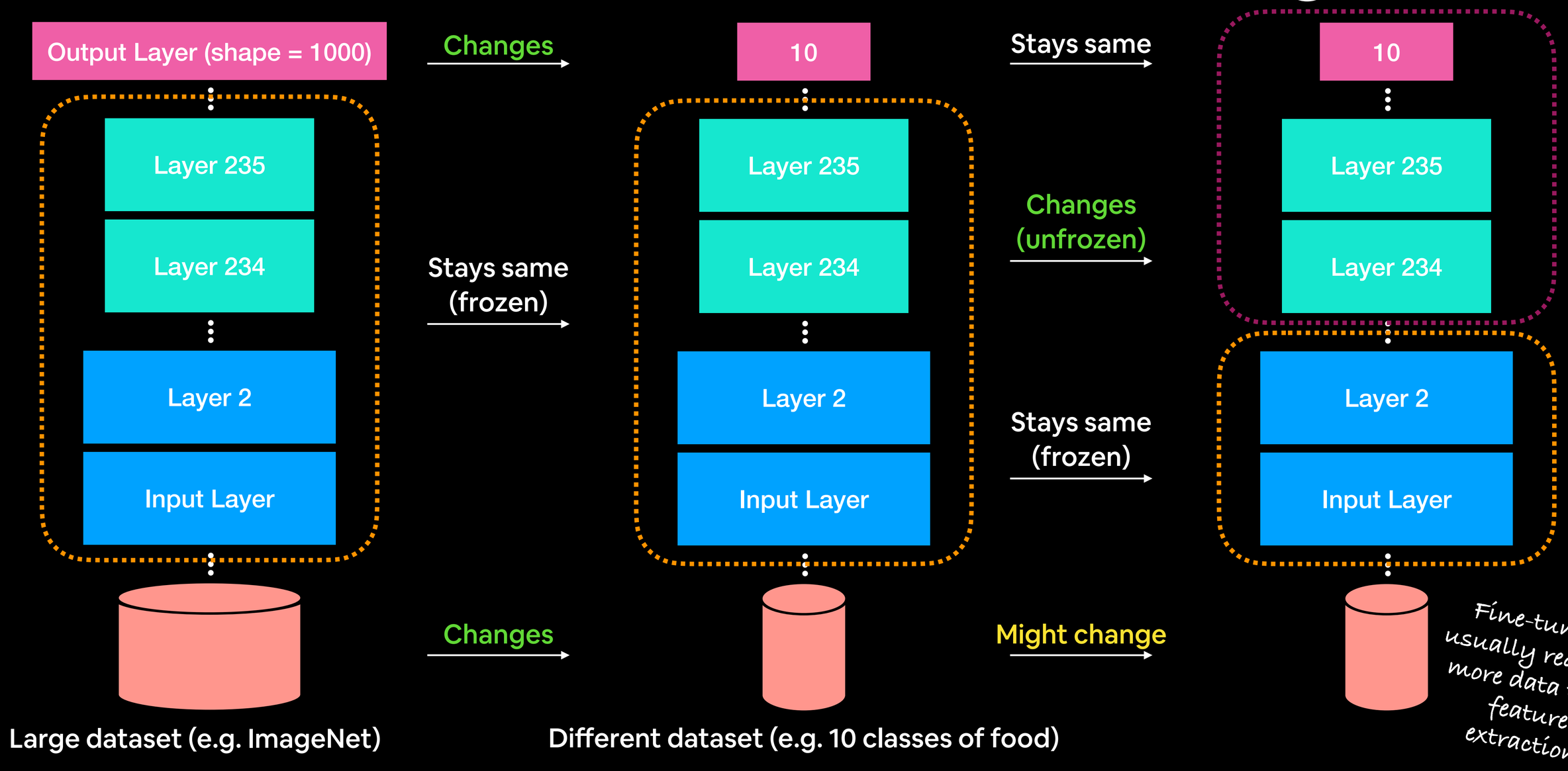


*__Note:__ In the code, we're actually using ResNet50, a slightly larger architecture than ResNet34.
__Image source:__ https://arxiv.org/abs/1512.03385

Original Model vs. Feature Extraction



Kinds of Transfer Learning



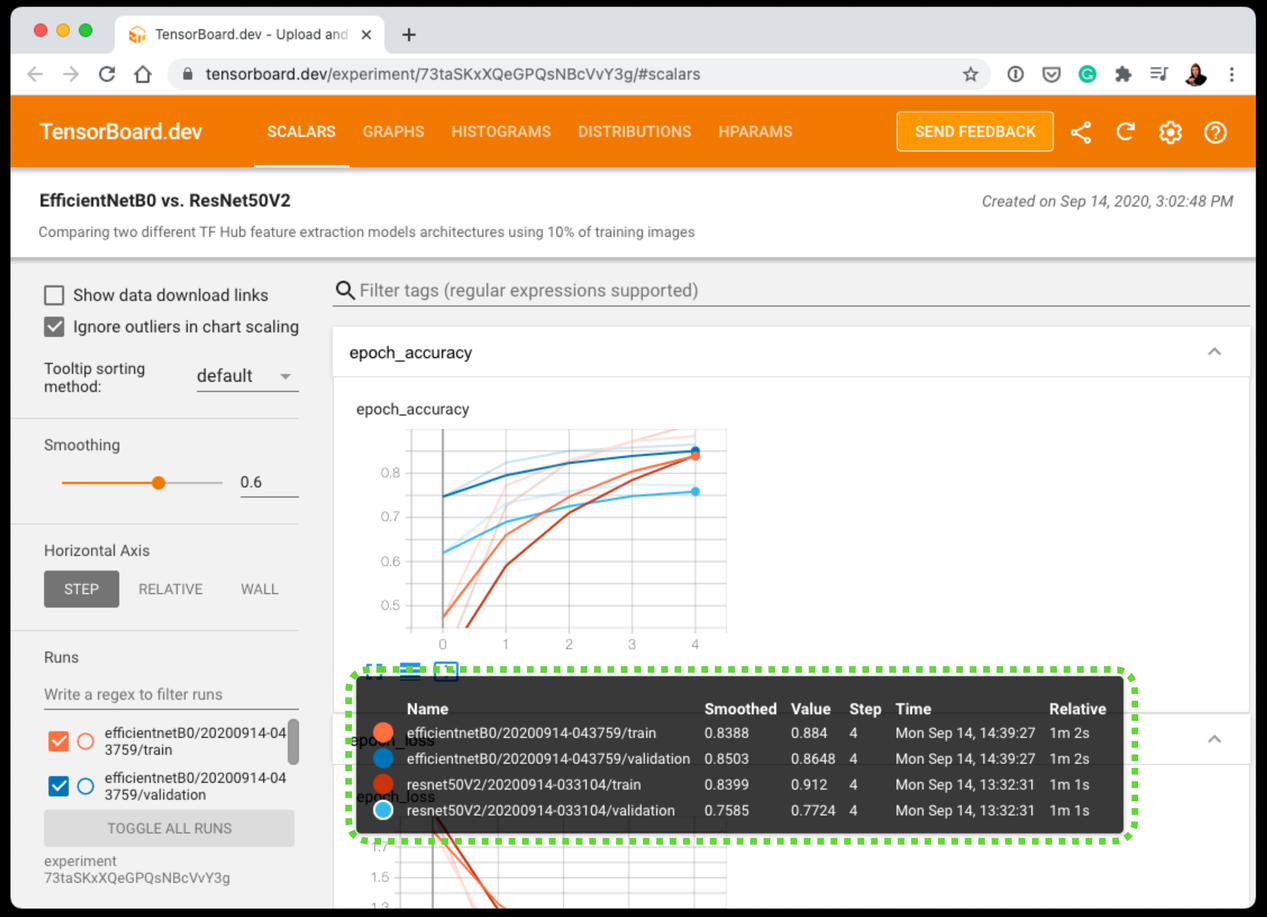
Kinds of Transfer Learning

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, 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

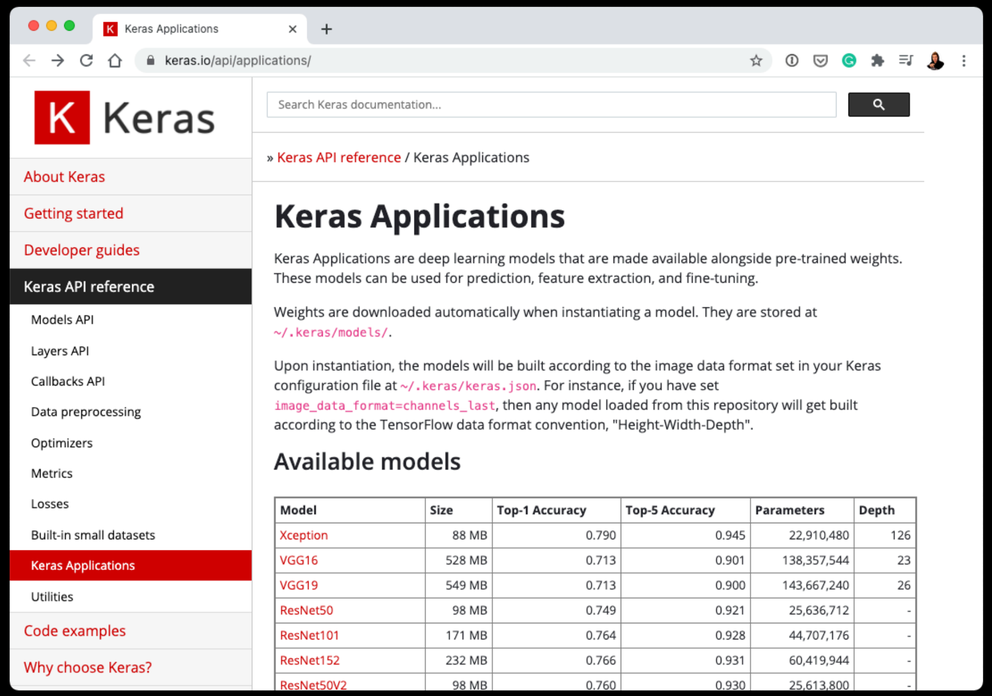
🍔👁 Food Vision: Dataset(s) we're using

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

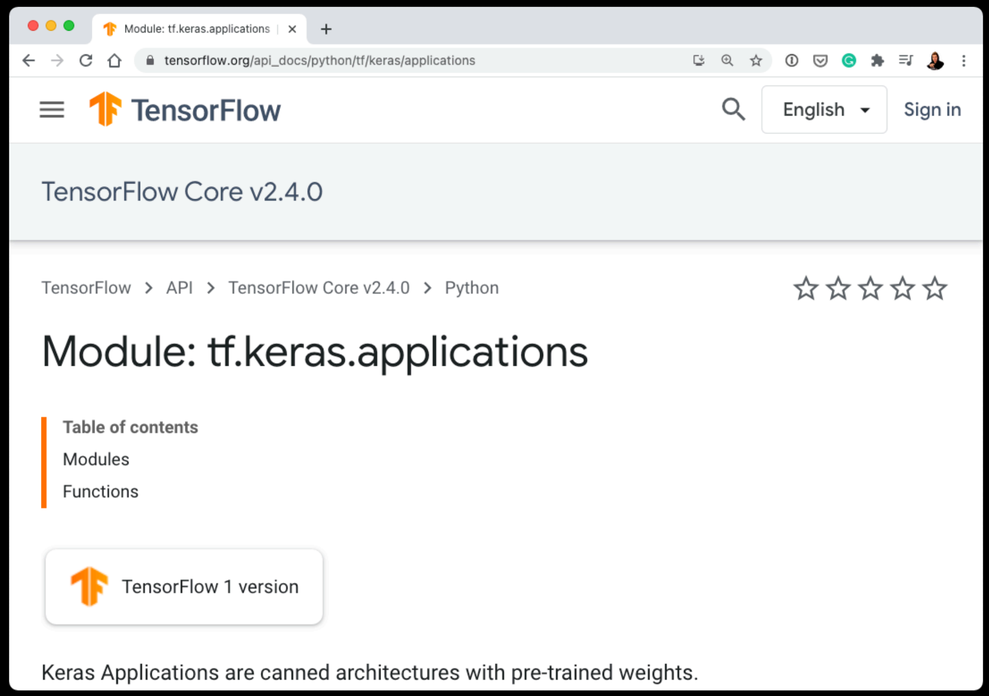
Dataset Name	Source	Classes	Training data	Testing data
pizza_steak	Food101	Pizza, steak (2)	750 images of pizza and steak (same as original Food101 dataset)	250 images of pizza and steak (same as original Food101 dataset)
10_food_classes_1_percent	Same as above	Chicken curry, chicken wings, fried rice, 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 training dataset)	250 images of each class (same as original Food101 dataset)

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

Improving a model

(from a model's perspective)

```python
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(4, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_subset, y_train_subset, epochs=5)
```

Smaller model


```python
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_full, y_train_full, epochs=100)
```

Larger model

Common ways to improve a deep model:

- Adding layers
- Increase the number of hidden units
- Change the activation functions
- Change the optimization function
- Change the learning rate
- Fitting on more data
- Fitting for longer

(because you can alter each of these, they're hyperparameters)