

Transfer Learning with

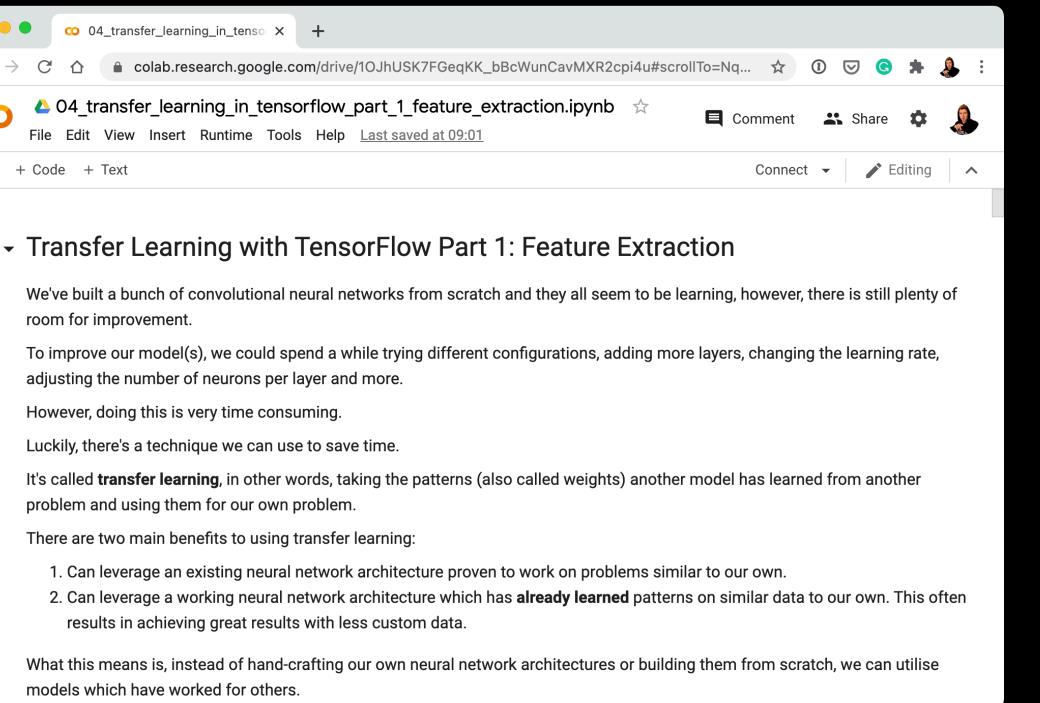


TensorFlow

Part 1: Feature extraction

Where can you get help?

- Follow along with the code



```
Transfer Learning with TensorFlow Part 1: Feature Extraction

We've built a bunch of convolutional neural networks from scratch and they all seem to be learning, however, there is still plenty of room for improvement.

To improve our model(s), we could spend a while trying different configurations, adding more layers, changing the learning rate, adjusting the number of neurons per layer and more.

However, doing this is very time consuming.

Luckily, there's a technique we can use to save time.

It's called transfer learning, in other words, taking the patterns (also called weights) another model has learned from another problem and using them for our own problem.

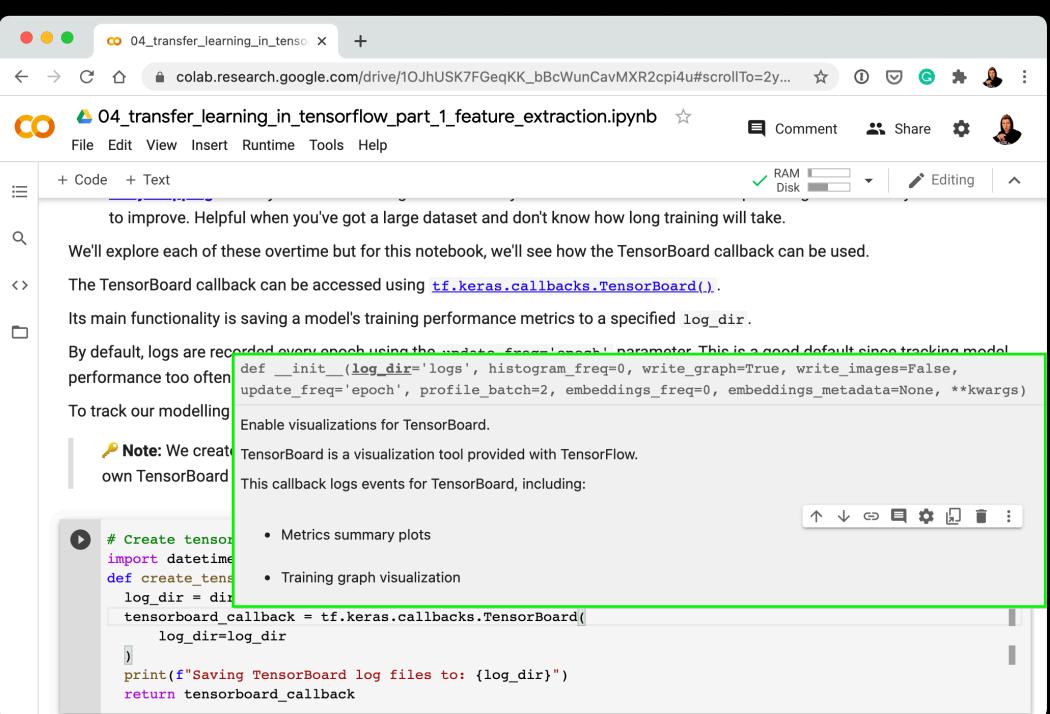
There are two main benefits to using transfer learning:  
1. Can leverage an existing neural network architecture proven to work on problems similar to our own.  
2. Can leverage a working neural network architecture which has already learned patterns on similar data to our own. This often results in achieving great results with less custom data.

What this means is, instead of hand-crafting our own neural network architectures or building them from scratch, we can utilise models which have worked for others.
```

"If in doubt, run the code"

- Try it for yourself

- Press SHIFT + CMD + SPACE to read the docstring



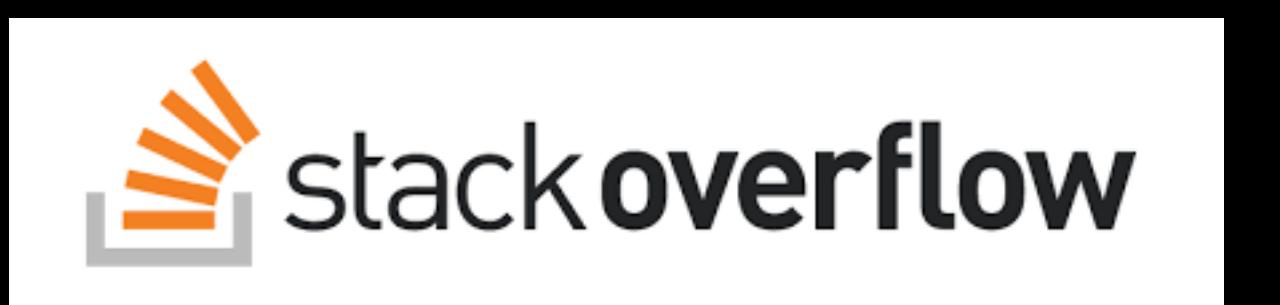
```
to improve. Helpful when you've got a large dataset and don't know how long training will take.

Well explore each of these overtime but for this notebook, we'll see how the TensorBoard callback can be used.

The TensorBoard callback can be accessed using tf.keras.callbacks.TensorBoard(). Its main functionality is saving a model's training performance metrics to a specified log_dir. By default, logs are recorded over time using the tf.summary.FileWriter class. This is a good default since tracking model performance too often can be resource intensive. We can update the log_dir parameter. This is a good default since tracking model performance too often can be resource intensive. To track our modelling progress, we can use the TensorBoard callback. This callback logs events for TensorBoard, including:  
• Metrics summary plots  
• Training graph visualization
```

```
# Create tensorboard
import tensorflow as tf
def create_tensorboard(log_dir):
    log_dir = os.path.join(os.getcwd(), log_dir)
    tensorboard_callback = tf.keras.callbacks.TensorBoard(
        log_dir=log_dir,
        histogram_freq=0, write_graph=True, write_images=False,
        update_freq='epoch', profile_batch=2, embeddings_freq=0, embeddings_metadata=None, **kwargs)
    return tensorboard_callback
```

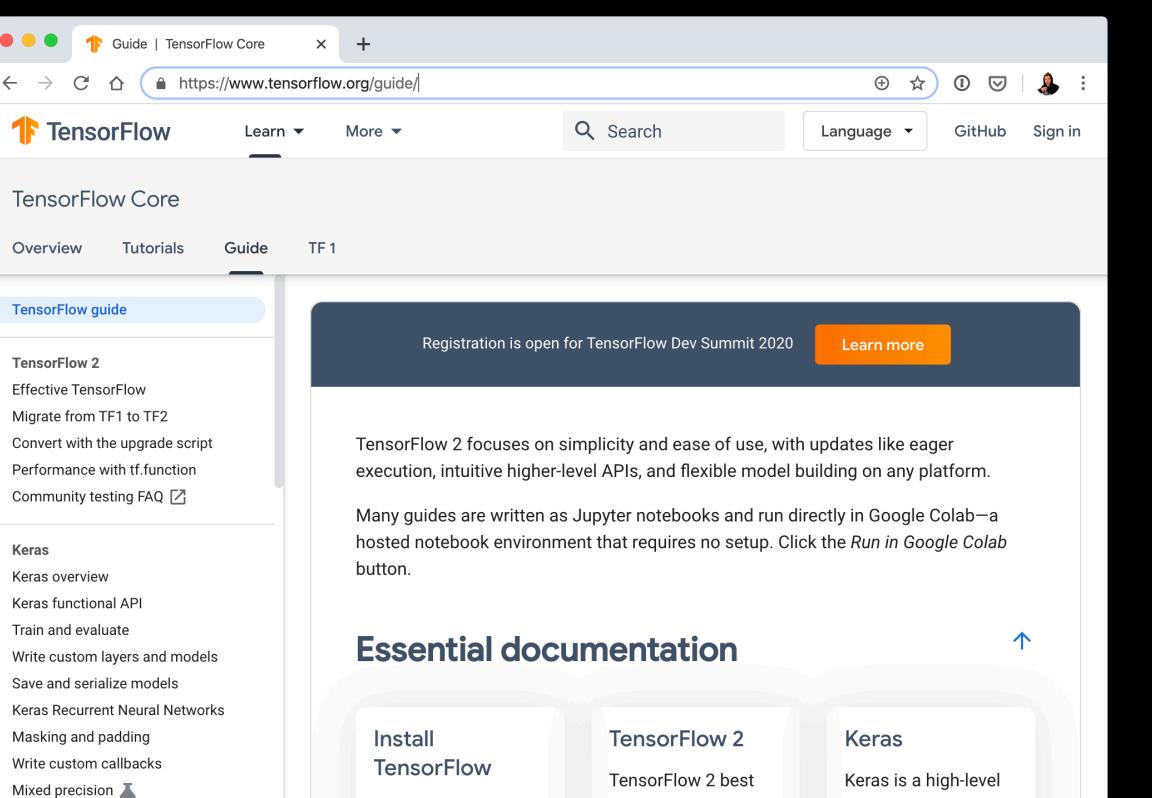
- Search for it



- Try again

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

(yes, including the "dumb" questions)

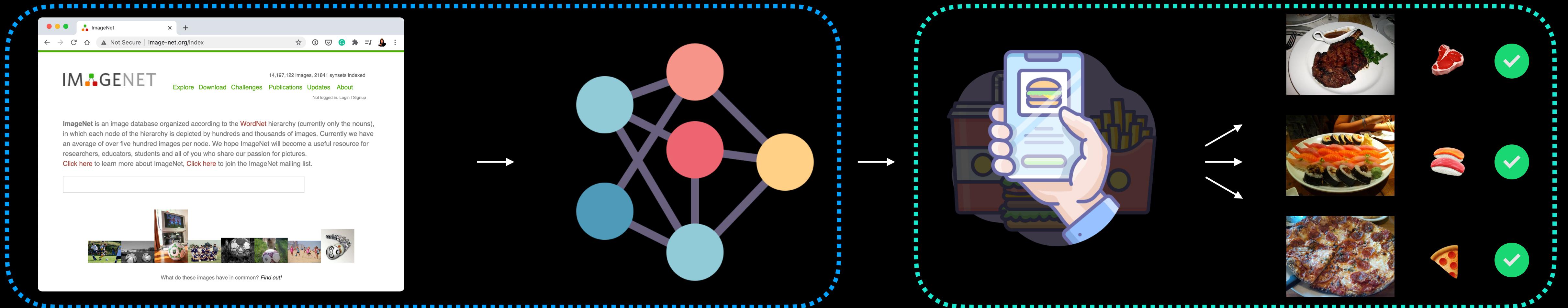


“What is transfer learning?”

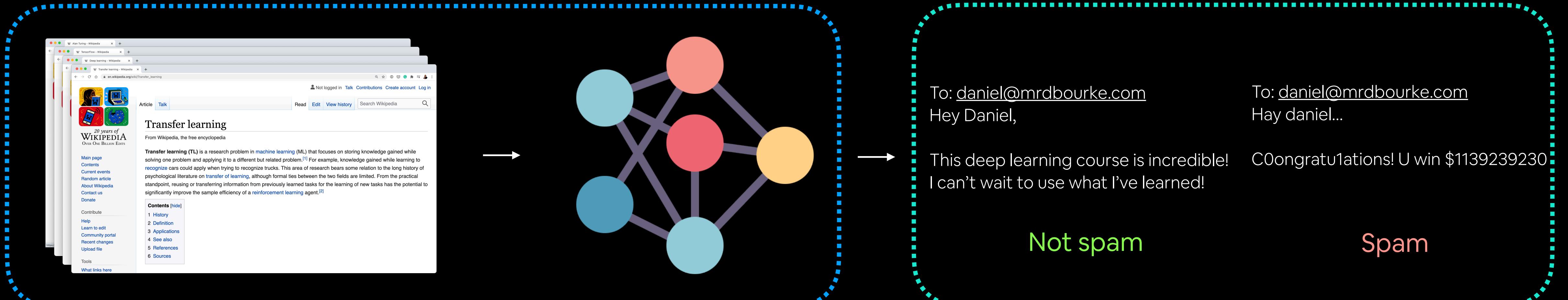
Surely someone has spent the time crafting the right model for the job...

Example transfer learning use cases

Computer vision



Natural language processing



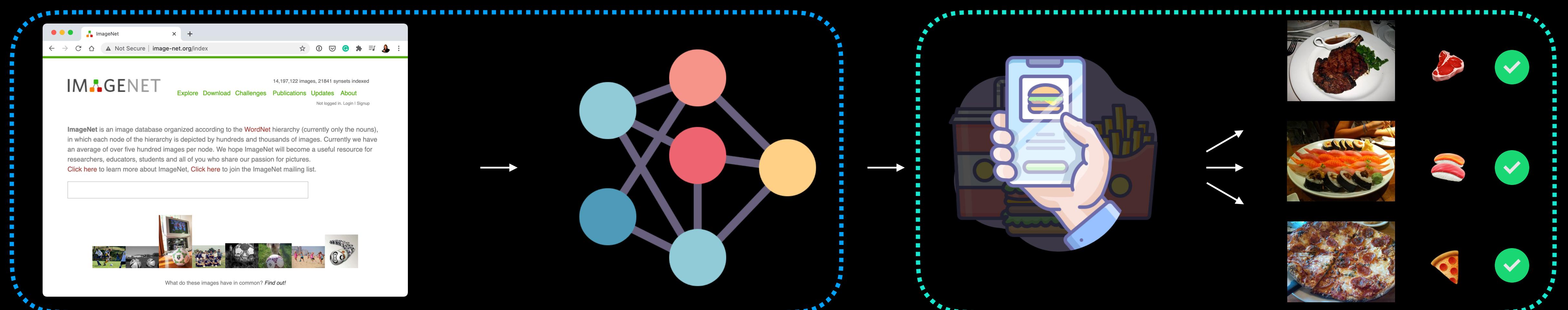
Model learns patterns/weights from similar problem space

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 architecture (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?

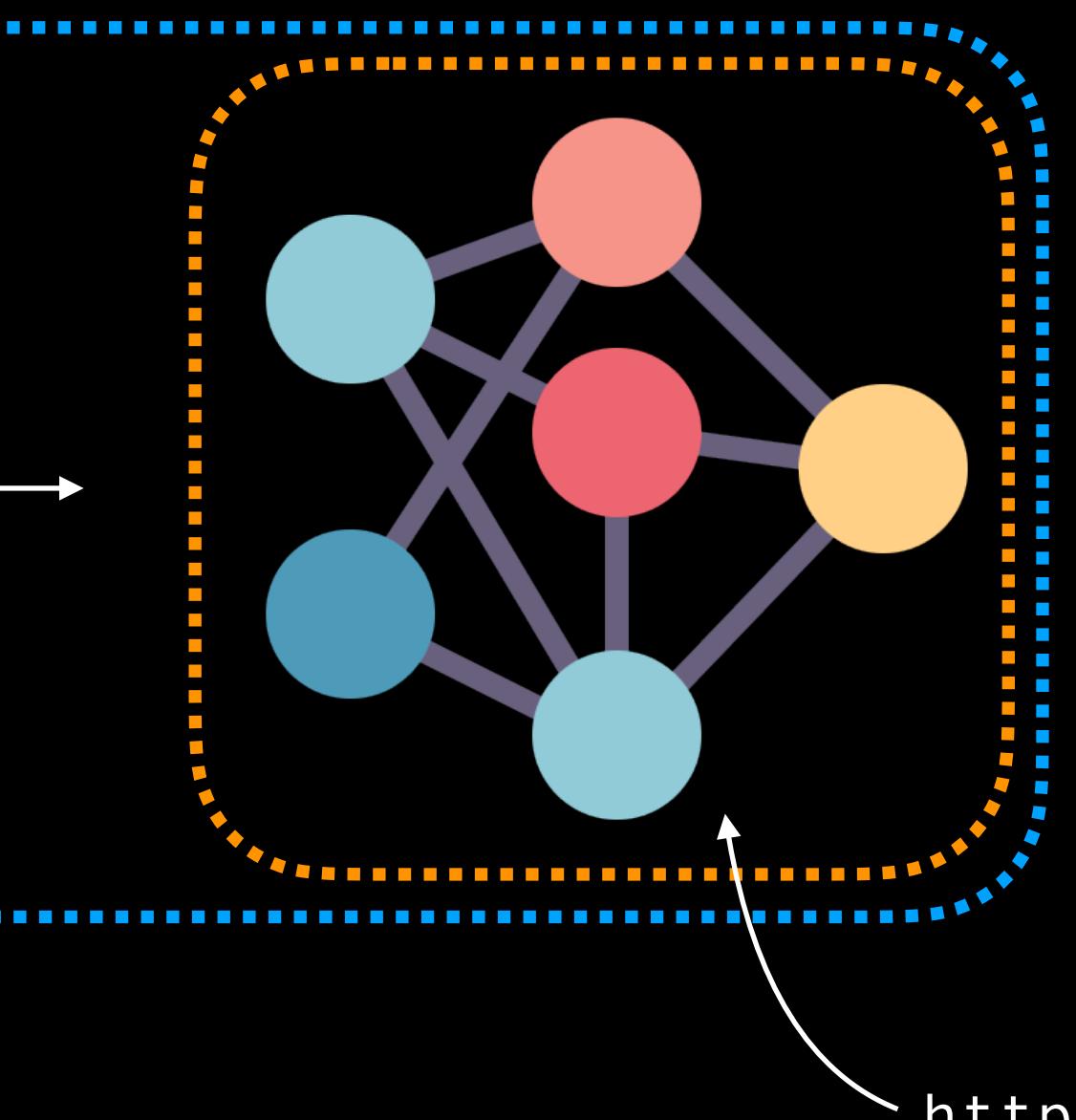
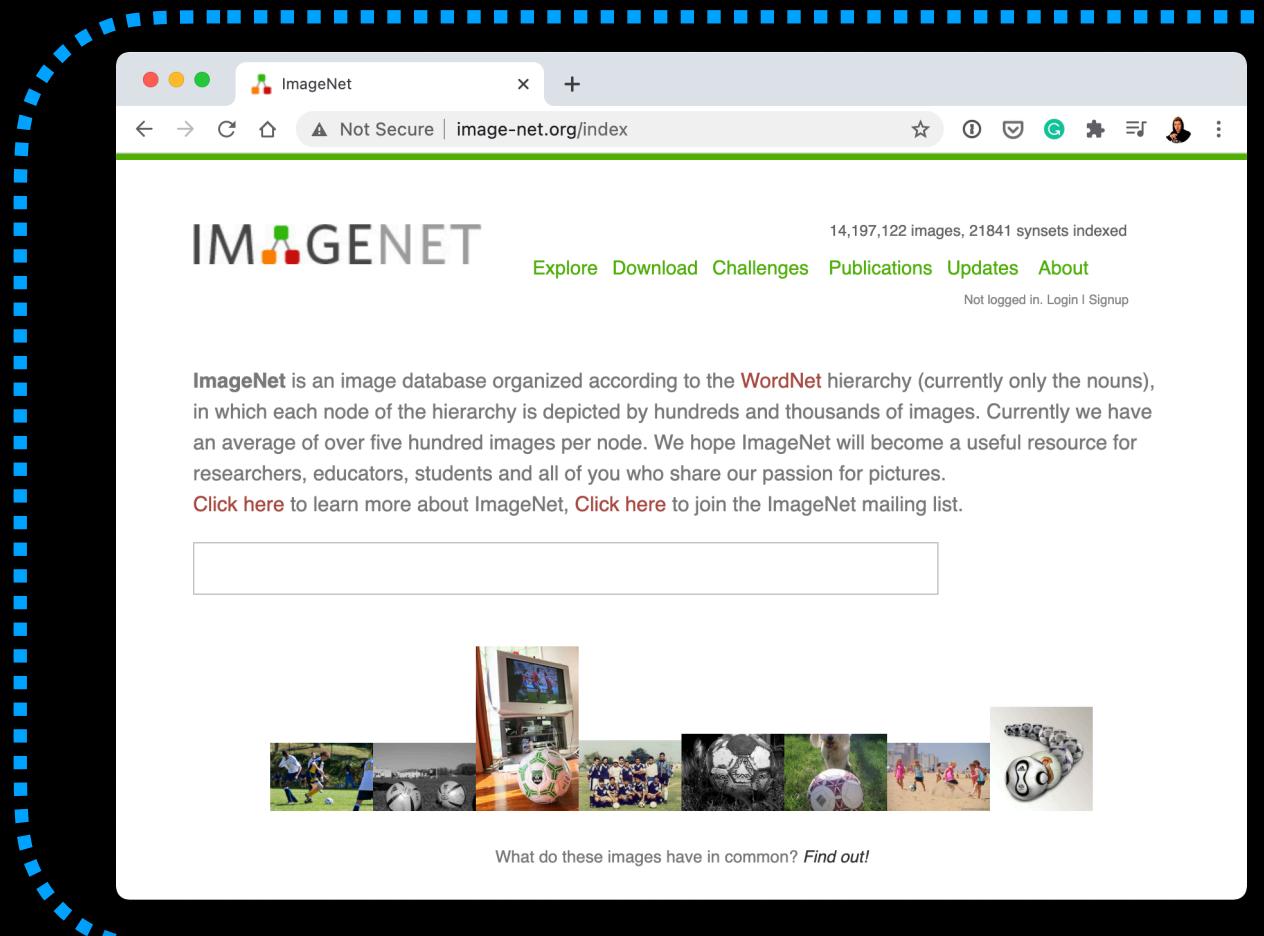
- Callbacks are a tool which can **add helpful functionality** to your models during training, evaluation or inference
- Some popular callbacks include:

Callback name	Use case	Code
<u>TensorBoard</u>	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.	<code>tf.keras.callbacks.TensorBoard()</code>
<u>Model checkpointing</u>	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.	<code>tf.keras.callbacks.ModelCheckpoint()</code>
<u>Early stopping</u>	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.	<code>tf.keras.callbacks.EarlyStopping()</code>

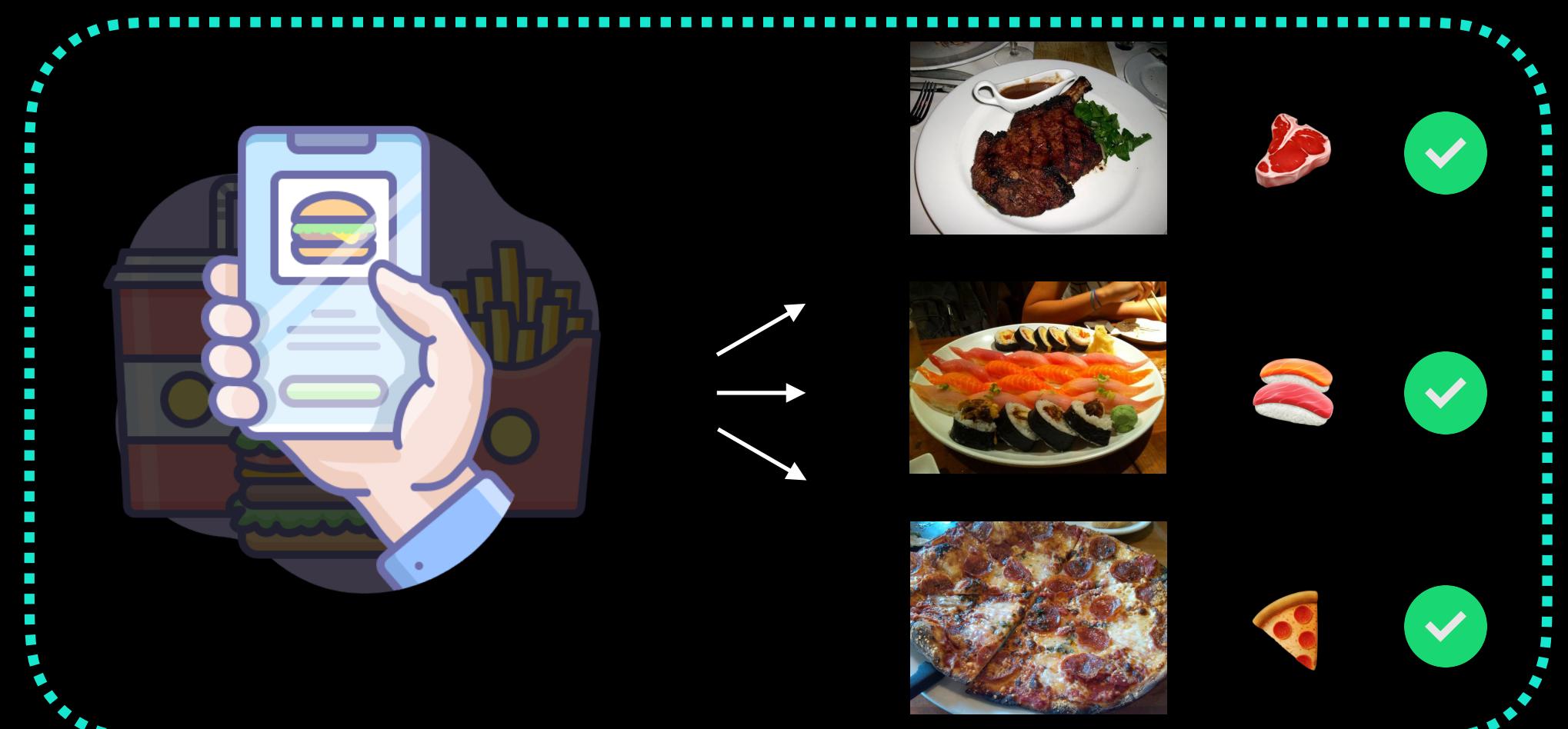
What is TensorFlow Hub?

- 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?”

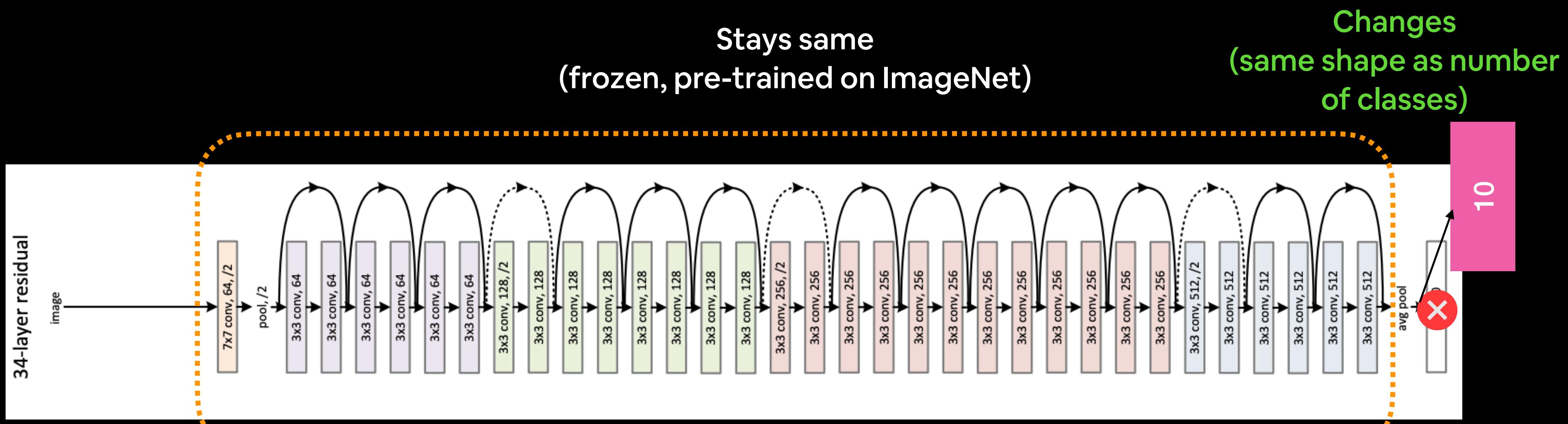
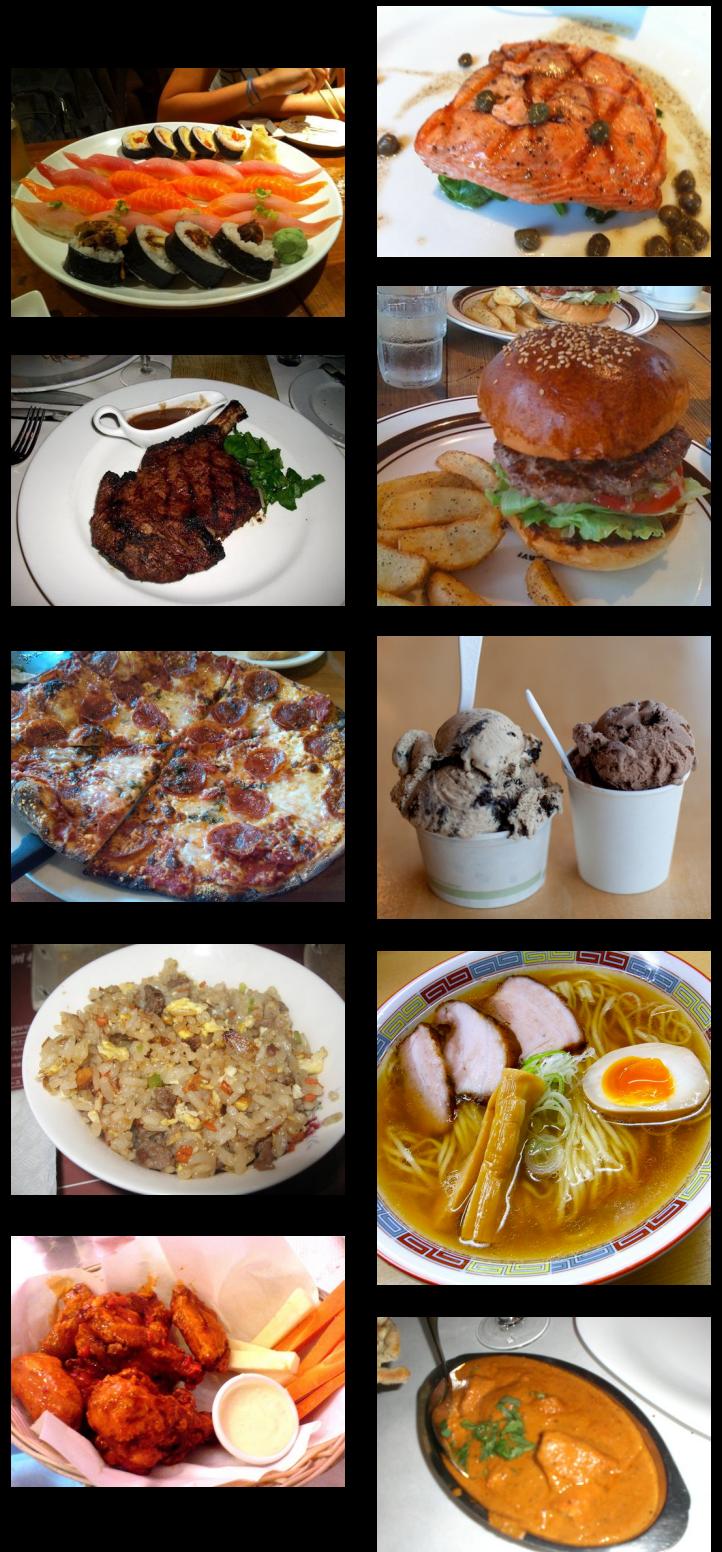


<https://tfhub.dev/tensorflow/efficientnet/b0/feature-vector/1>

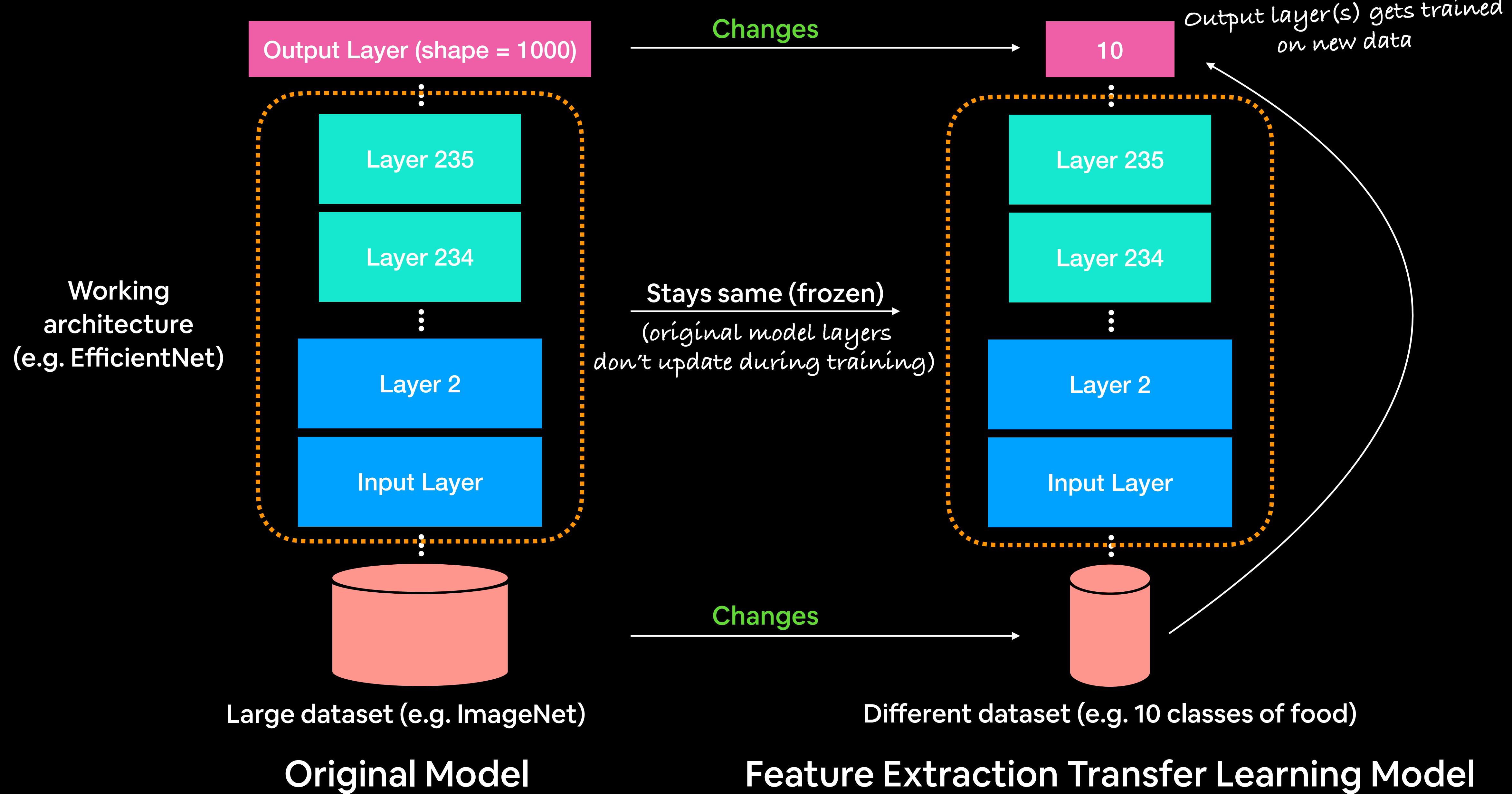
TensorFlow Hub makes using a pre-trained model as
simple as calling a URL

ResNet50* feature extractor

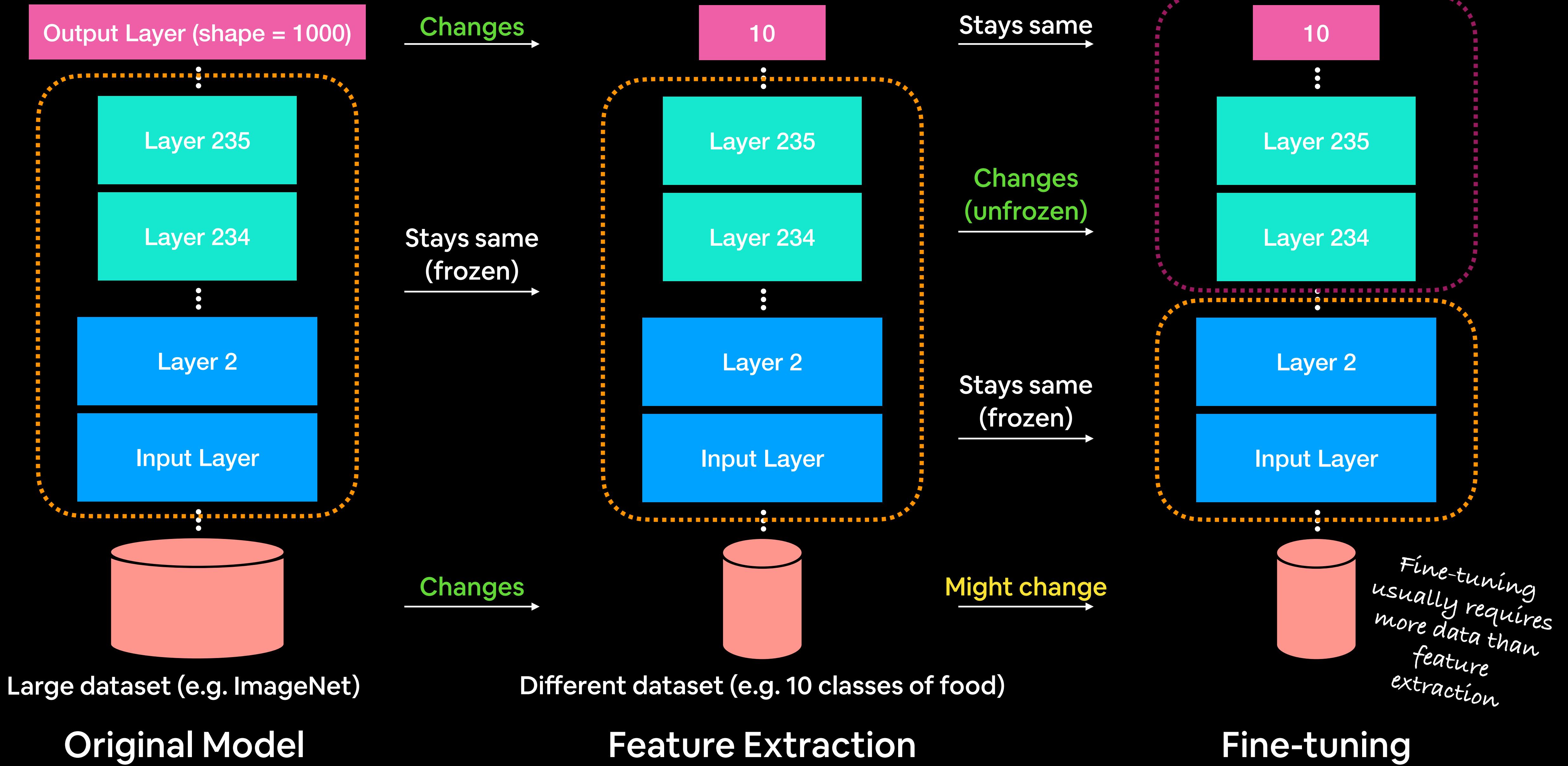
Input data
(10 classes of Food101)



Original Model vs. Feature Extraction



Kinds of Transfer Learning

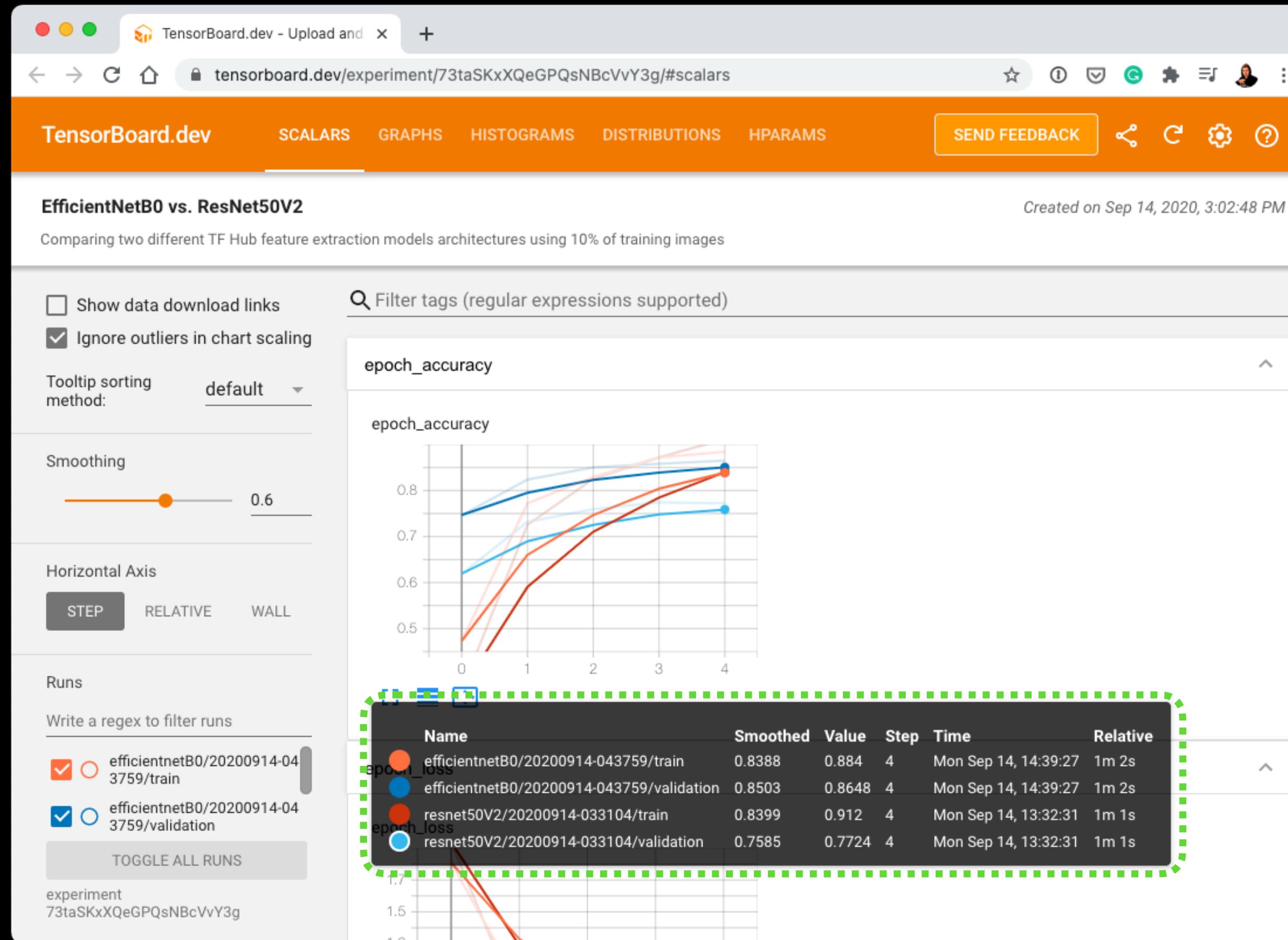


Kinds of Transfer Learning

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?

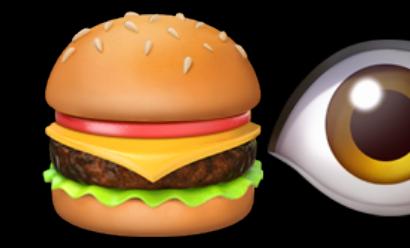
- A way to **visually explore** your machine learning models performance and internals
- Host, track and share your machine learning experiments on TensorBoard.dev



(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

The screenshot shows the Keras Applications page. The left sidebar has a red background for the 'Keras Applications' section. The main content area includes a search bar, a breadcrumb trail ('» Keras API reference / Keras Applications'), and a table of available models.

Keras Applications

Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

Weights are downloaded automatically when instantiating a model. They are stored at `~/.keras/models/`.

Upon instantiation, the models will be built according to the image data format set in your Keras configuration file at `~/.keras/keras.json`. For instance, if you have set `image_data_format=channels_last`, then any model loaded from this repository will get built according to the TensorFlow data format convention, "Height-Width-Depth".

Available models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-

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

The screenshot shows the tf.keras.applications module page. The top navigation bar includes the TensorFlow logo and language selection ('English'). The main content area includes a table of contents, a sidebar with 'TensorFlow 1 version' and 'Keras Applications are canned architectures with pre-trained weights.', and a five-star rating.

Module: tf.keras.applications

TensorFlow Core v2.4.0

TensorFlow > API > TensorFlow Core v2.4.0 > Python

★★★★★

Table of contents

Modules

Functions

TensorFlow 1 version

Keras Applications are canned architectures with pre-trained weights.

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

Improving a model

(from a model's perspective)

```
# 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

```
# 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 (because you can alter each of these, they're hyperparameters)
- Fitting on more data
- Fitting for longer