Transfer Learning with

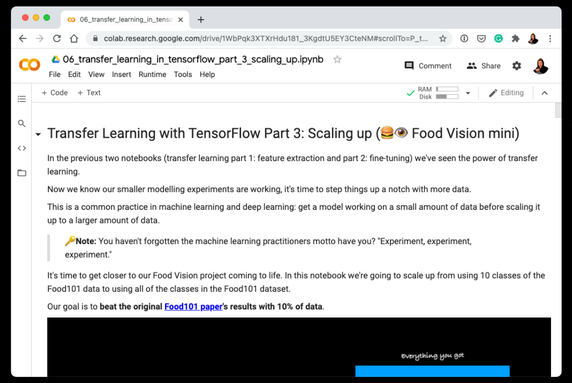
TensorFlow

Part 3: Scaling up

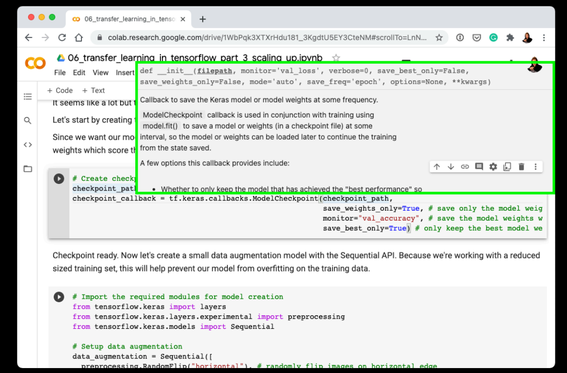
Where can you get help?

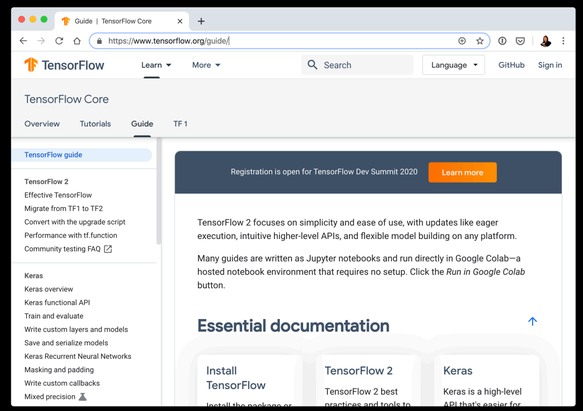
- Follow along with the code
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring
- Search for it
- Try again
- Ask (don't forget the Discord chat!)

(yes, including the "dumb" questions)



"If in doubt, run the code"



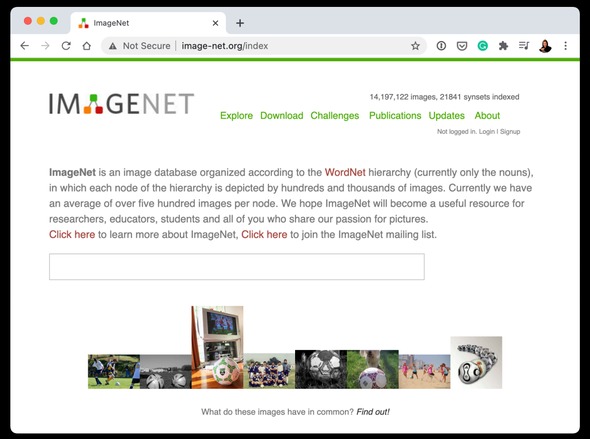


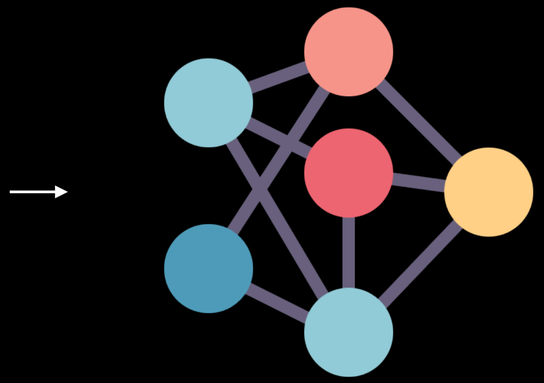
“What is transfer learning?”

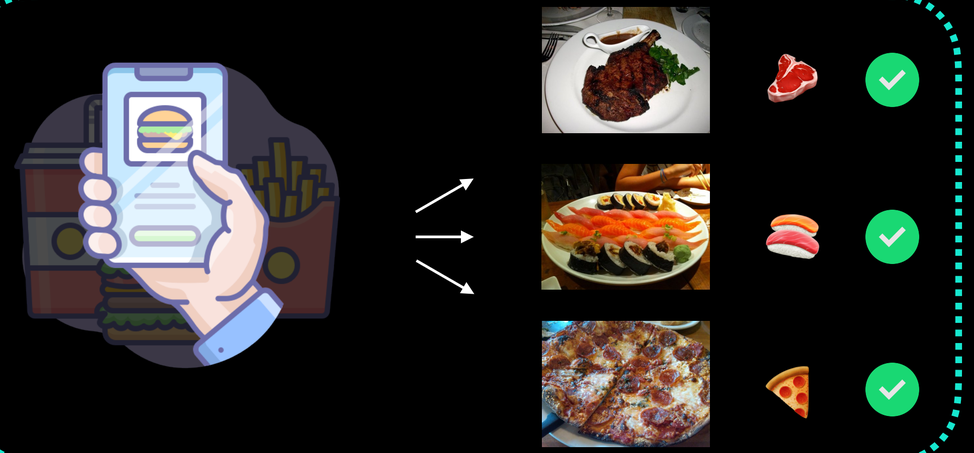
Surely someone has spent the time crafting the right model for the job...

Example transfer learning use cases

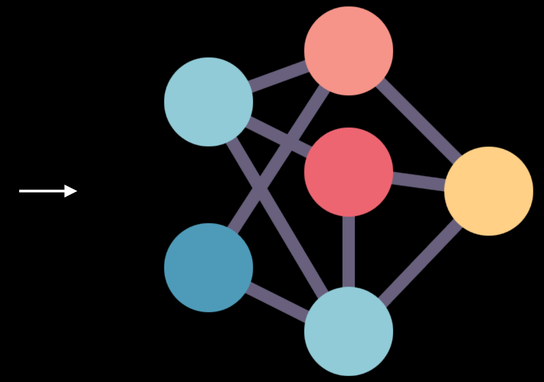
Computer vision







Natural language processing



To: daniel@mrdbourke.com
Hey Daniel,

This deep learning course is incredible!
I can't wait to use what I've learned!

Not spam

To: daniel@mrdbourke.com
Hay daniel...

C0ongratu1ations! U win $1139239230

Spam

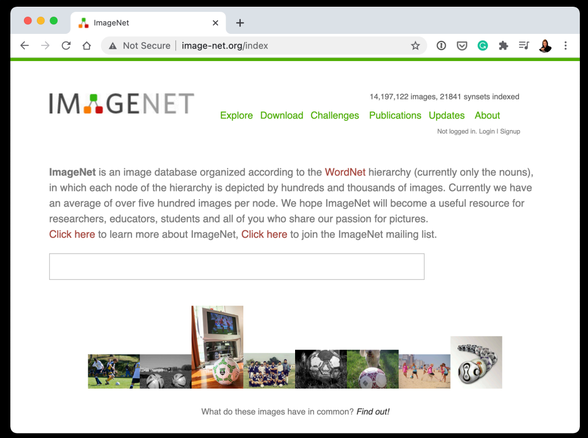
Model learns patterns/weights from similar problem space

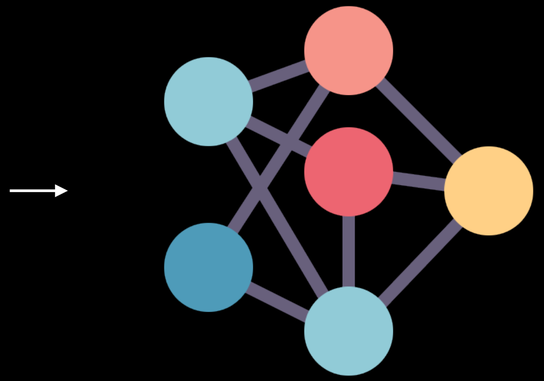
Patterns get used/tuned to specific problem

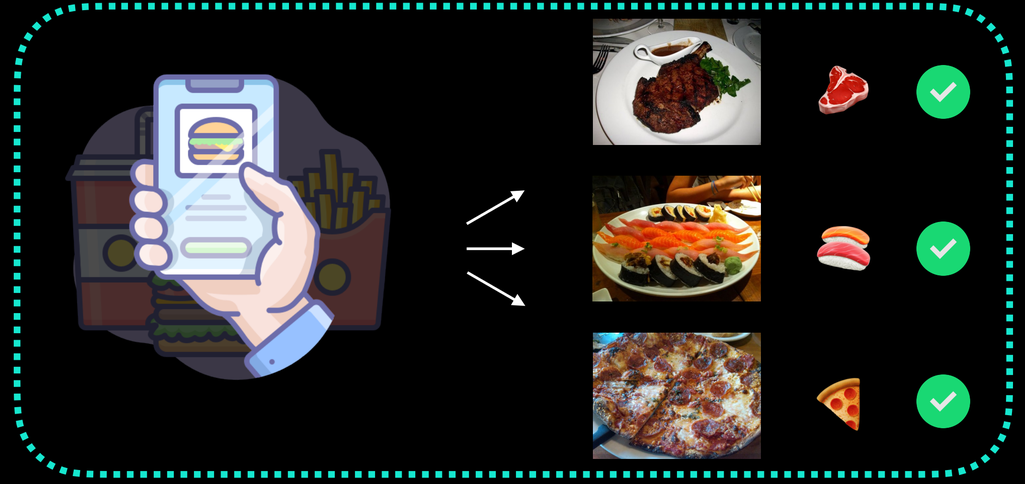
“Why use transfer learning?”

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

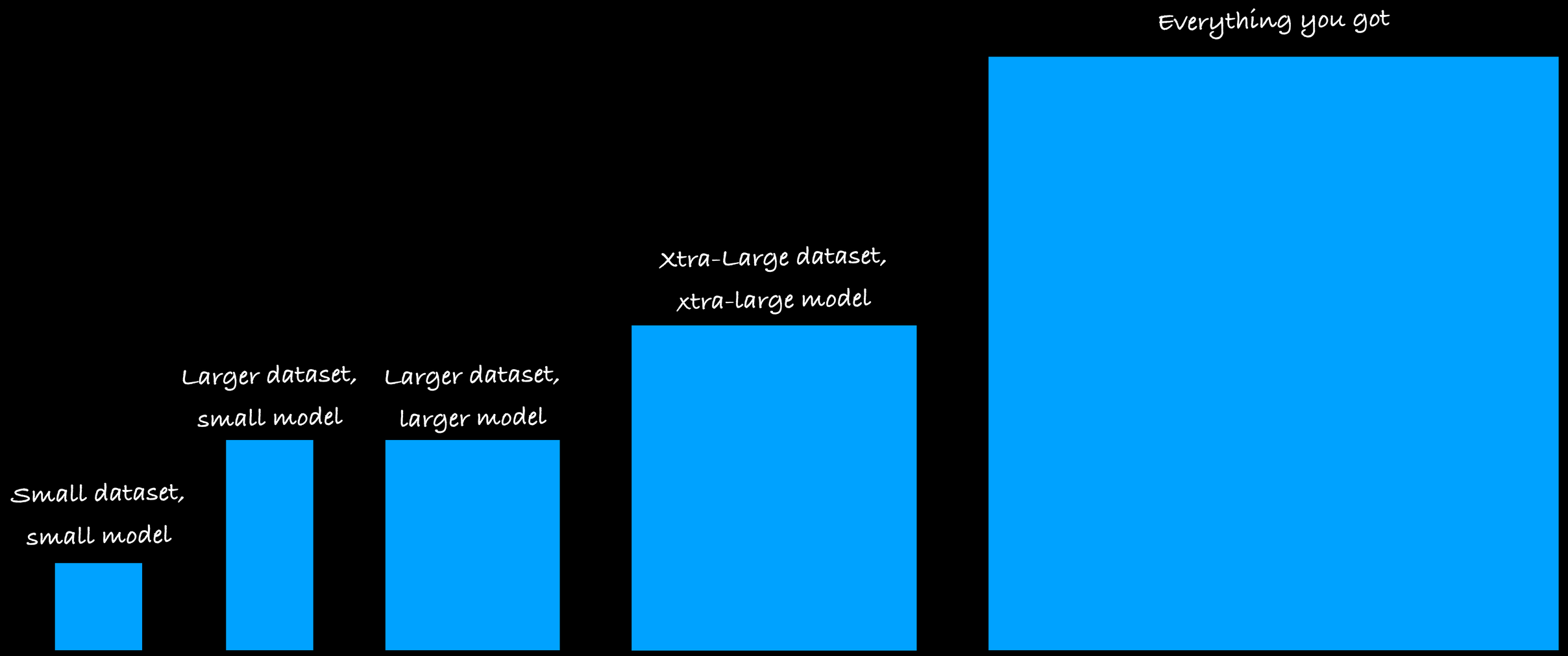
(broadly)

- Downloading & preparing 10% of all Food101 classes (7500+ training images)
- Training a transfer learning feature extraction model
- Fine-tuning our feature extraction model (🍔👁️Food Vision mini) to beat the original Food101 paper with only 10% of the data
- Evaluating Food Vision mini's predictions
  - Finding the most wrong predictions (on the test dataset)
- Making predictions with Food Vision mini on our own custom images

(we'll be cooking 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)

The dataset we're using in Transfer Learning with TensorFlow Part 3: Scaling up

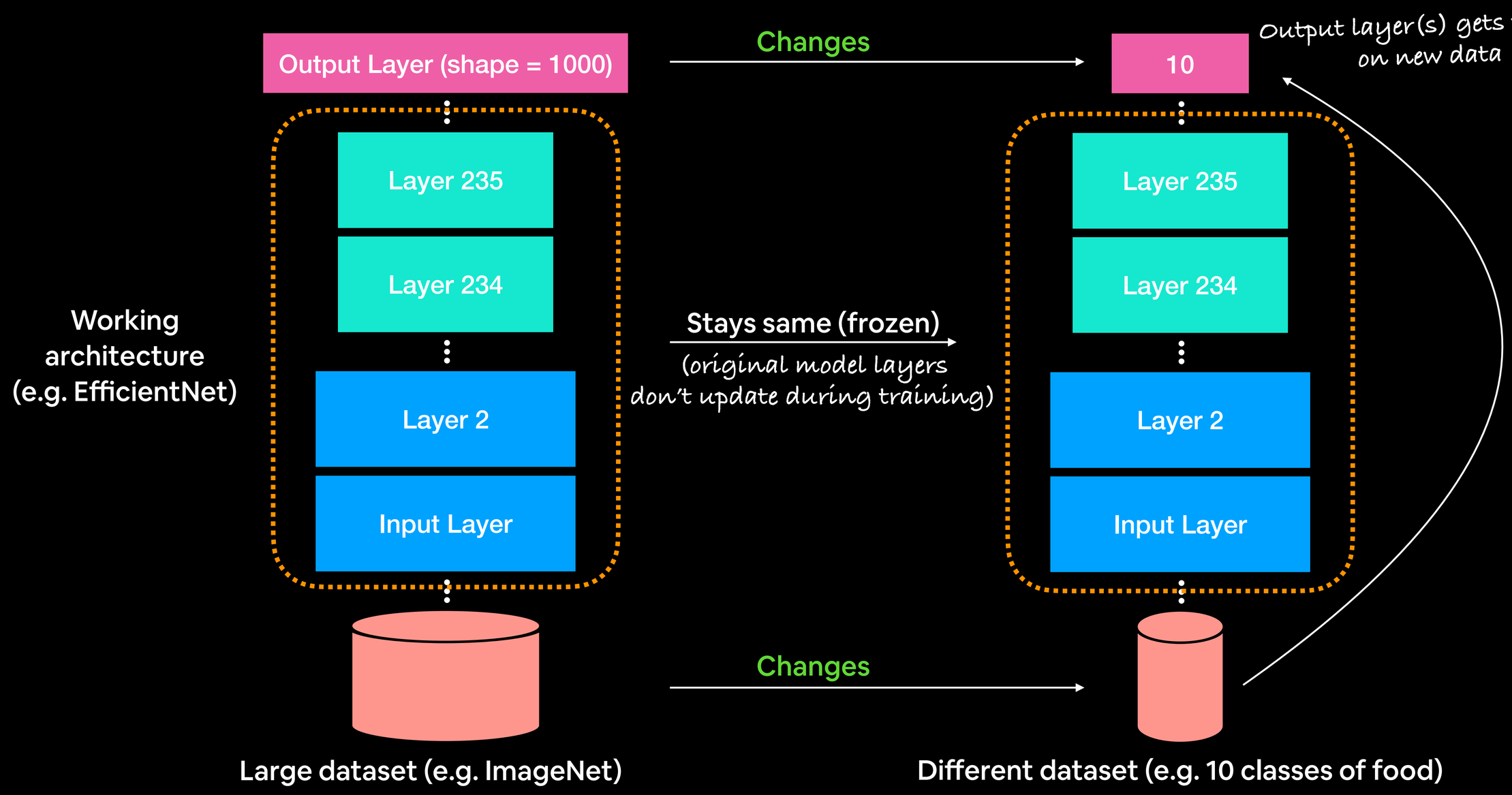
Let’s code!

What are callbacks?

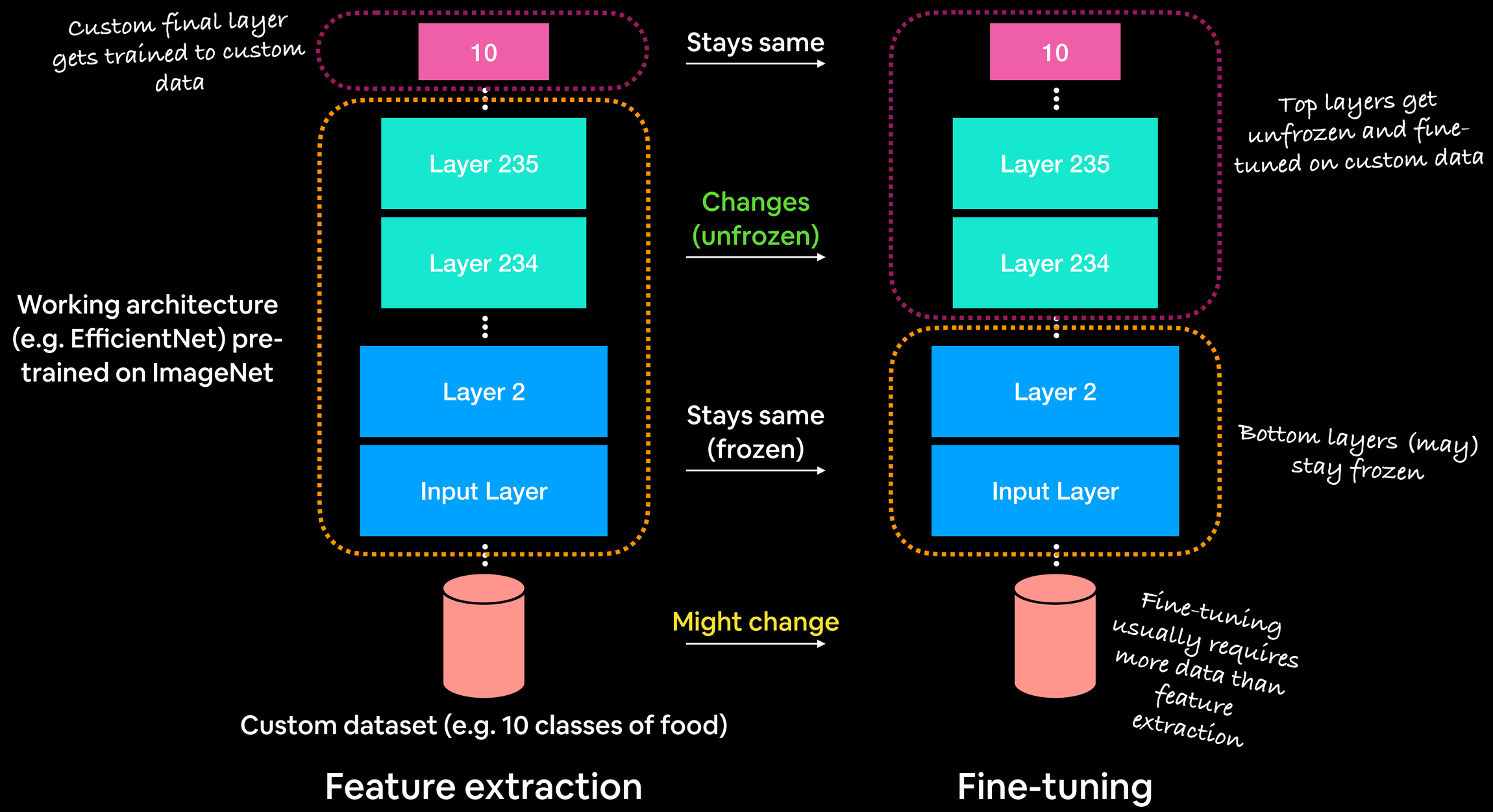
- Callbacks are a tool which can add helpful functionality to your models during training, evaluation or inference
- Some popular callbacks include:

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()`

Original Model vs. Feature Extraction



Feature extraction vs. Fine-tuning

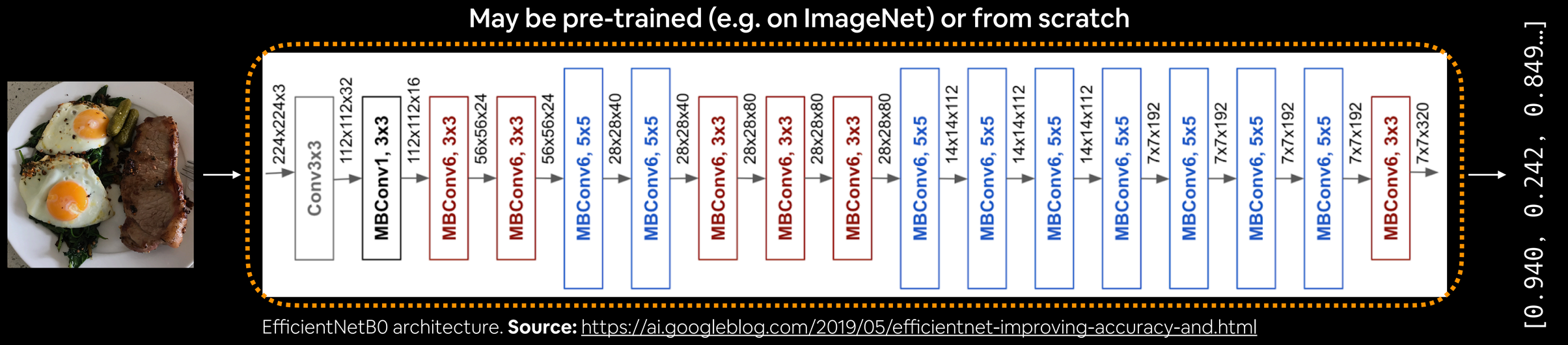


Kinds of Transfer Learning

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 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

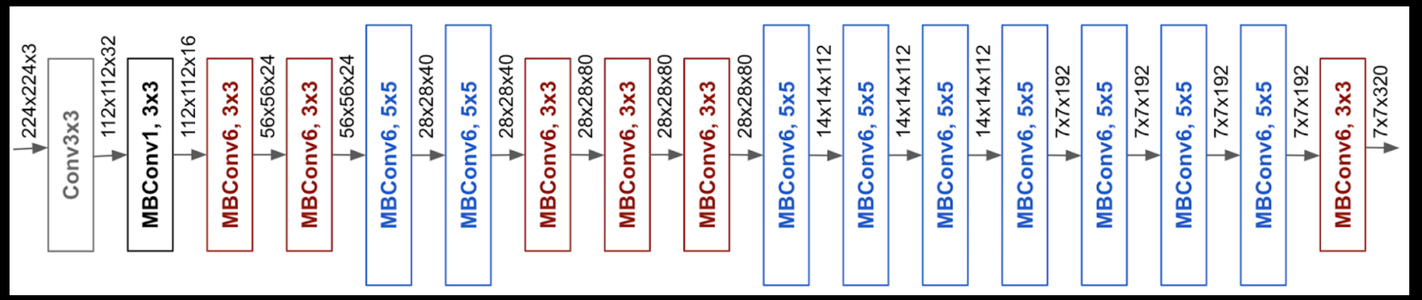
Output (feature vector)

Model we've created

Input data

Data augmentation

Model



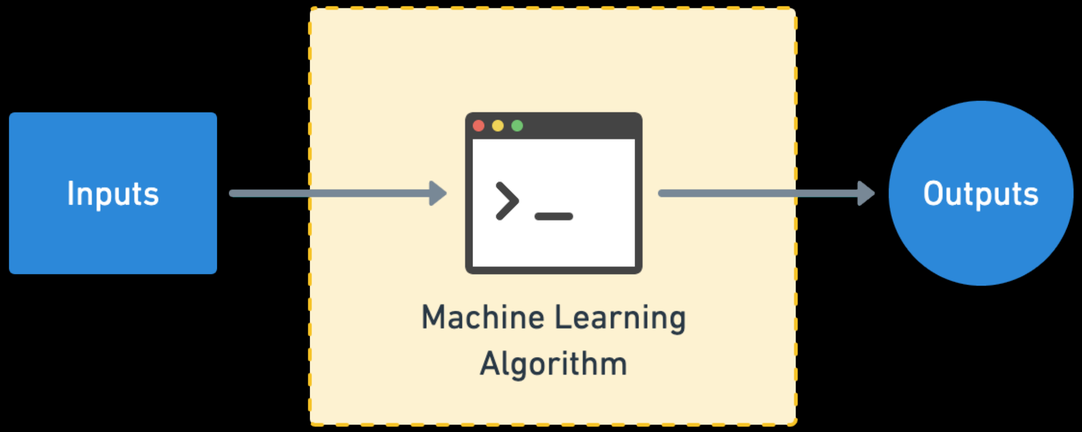
EfficientNetB0 architecture.
Source: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

Output

GlobalAvgPool2D

10

Fully-connected (dense) classifier layer

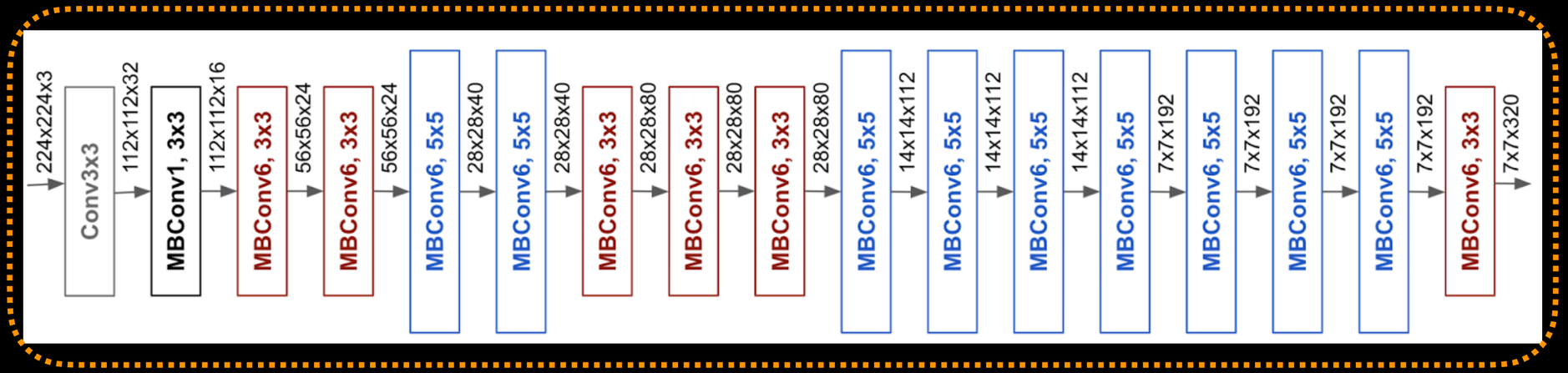


EfficientNet feature extractor

Input data
(10 classes of Food101)

Stays same
(frozen, pre-trained on ImageNet)

Changes
(same shape as number of classes)



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

Fully-connected
(dense) classifier layer

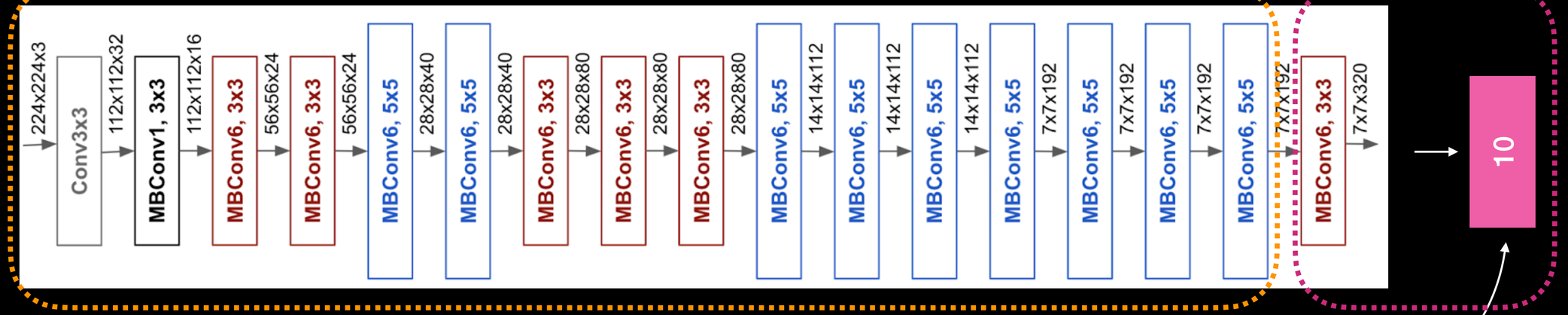
EfficientNet fine-tuning

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)



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

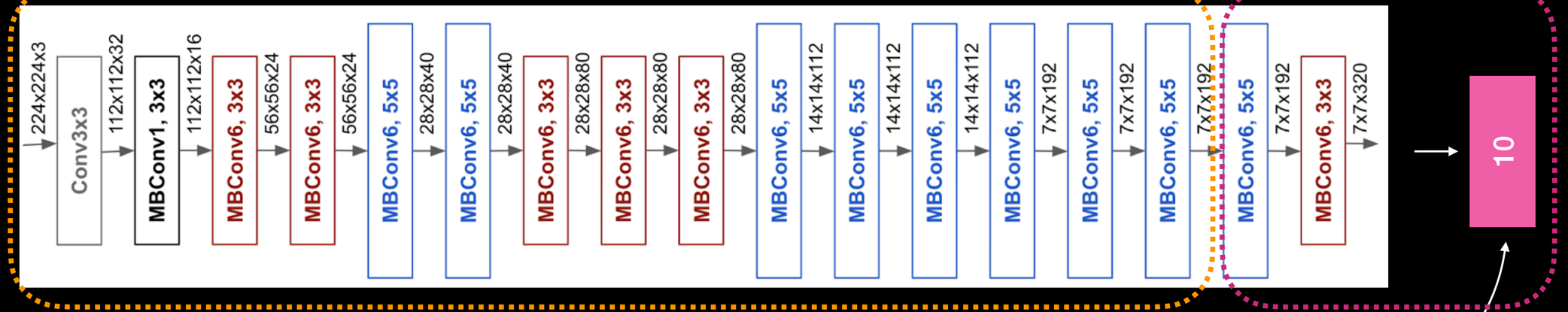
EfficientNet fine-tuning

Input data
(10 classes of Food101)

Learning rate: 0.0001 ⬇️

Stays same
(frozen, pre-trained on ImageNet)

Changes
(unfrozen)



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

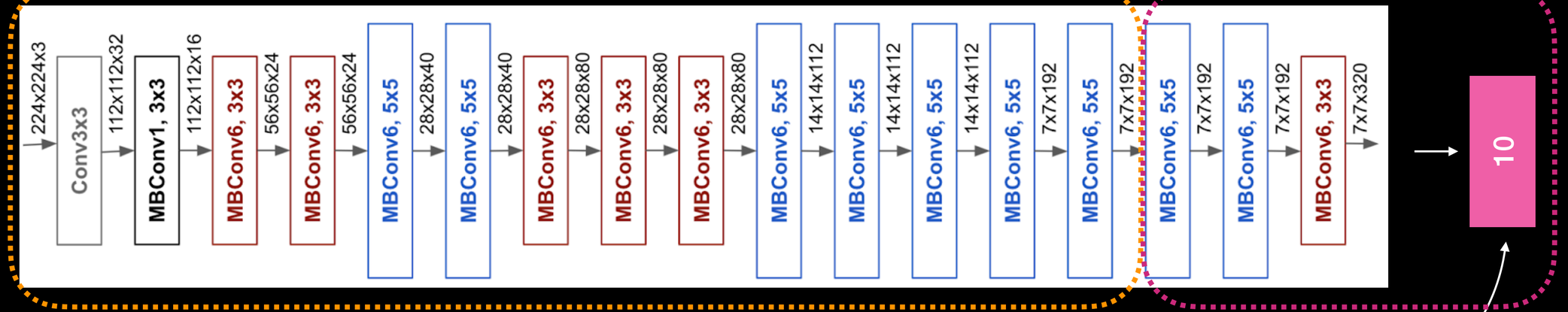
EfficientNet fine-tuning

Input data
(10 classes of Food101)

Learning rate: 0.000025 ⬇️

Stays same
(frozen, pre-trained on ImageNet)

Changes
(unfrozen)



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

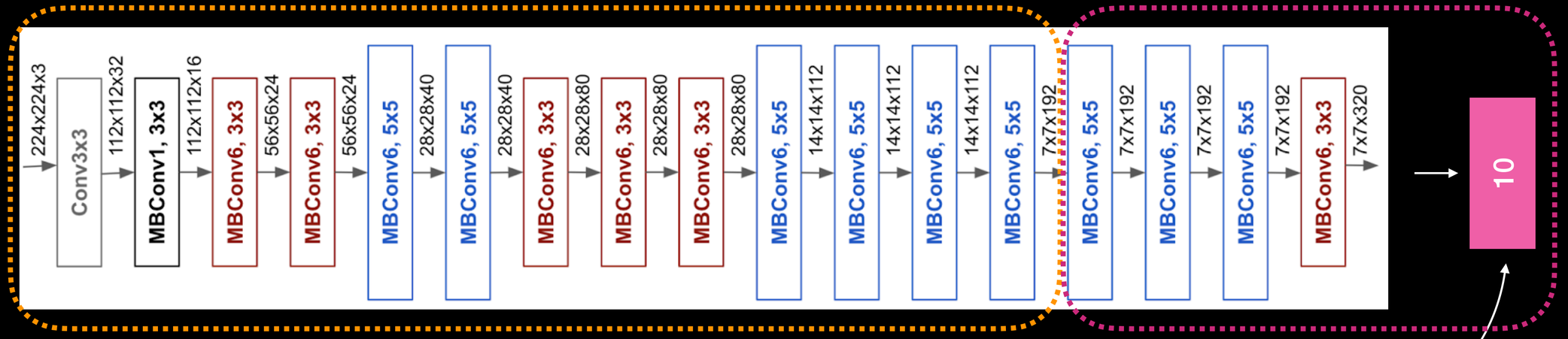
EfficientNet fine-tuning

Input data
(10 classes of Food101)

Learning rate: 0.00001 ⬇️

Stays same
(frozen, pre-trained on ImageNet)

Changes
(unfrozen)



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

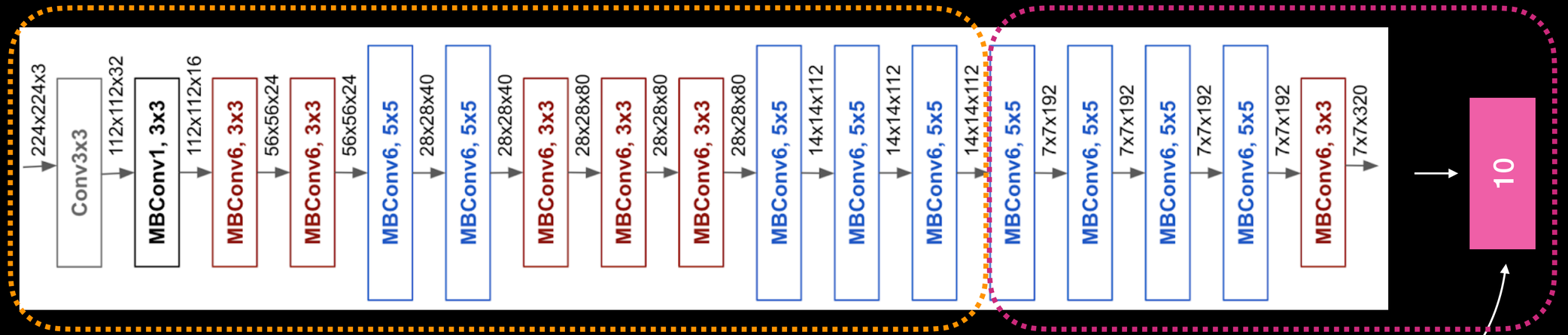
EfficientNet fine-tuning

Input data
(10 classes of Food101)

Learning rate: 0.0000025 ⬇️

Stays same
(frozen, pre-trained on ImageNet)

Changes
(unfrozen)



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

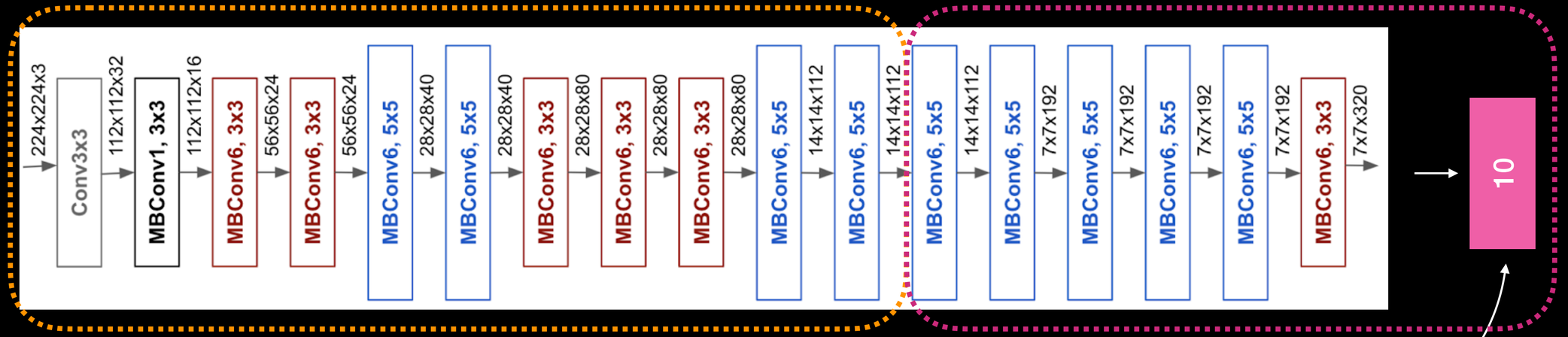
EfficientNet fine-tuning

Input data
(10 classes of Food101)

Learning rate: 0.000001 ⬇️

Stays same
(frozen, pre-trained on ImageNet)

Changes
(unfrozen)



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

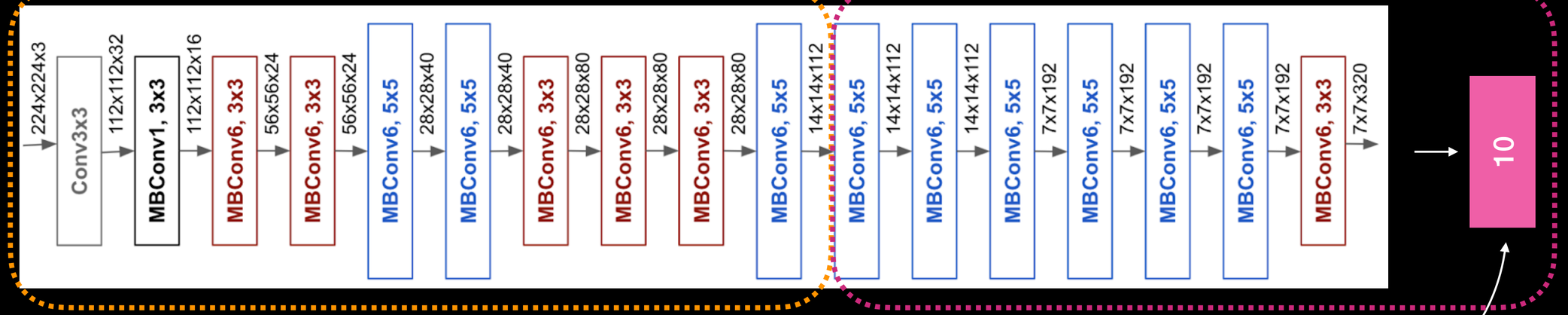
EfficientNet fine-tuning

Input data
(10 classes of Food101)

Learning rate: 0.00000025 ⬇️

Stays same
(frozen, pre-trained on ImageNet)

Changes
(unfrozen)



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

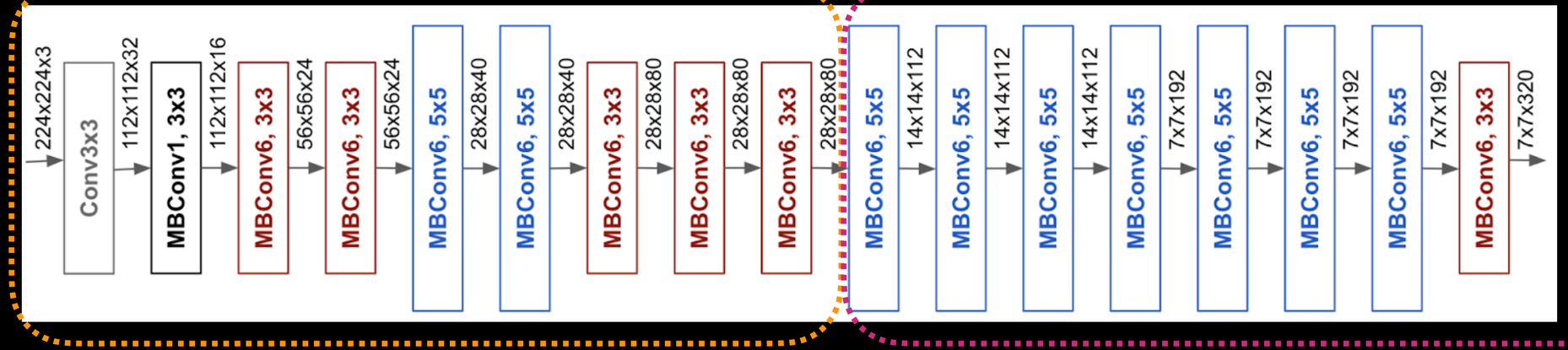
EfficientNet fine-tuning

Input data
(10 classes of Food101)

Learning rate: 0.0000001 ⬇️

Stays same
(frozen, pre-trained on ImageNet)

Changes
(unfrozen)



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

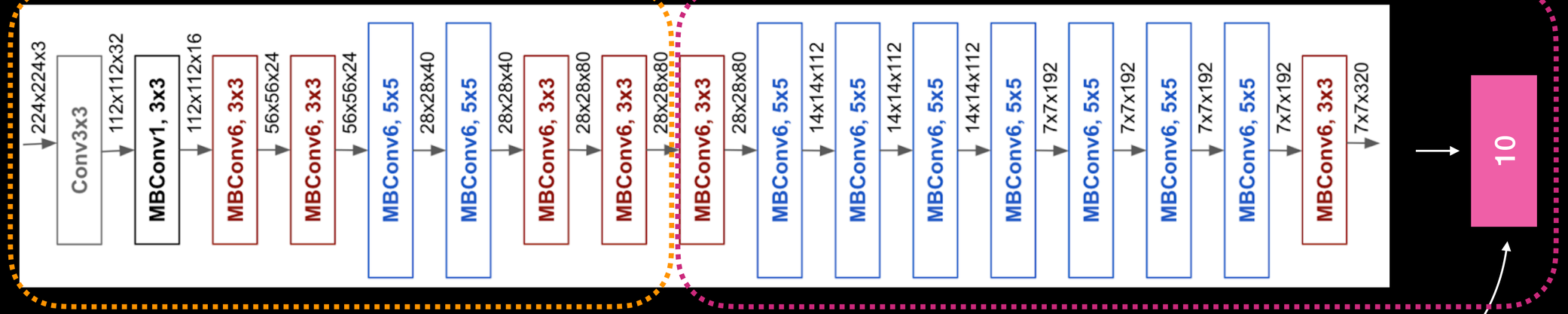
EfficientNet fine-tuning

Input data
(10 classes of Food101)

Learning rate: 0.000000025 ⬇️

Stays same
(frozen, pre-trained on ImageNet)

Changes
(unfrozen)



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

(some common)

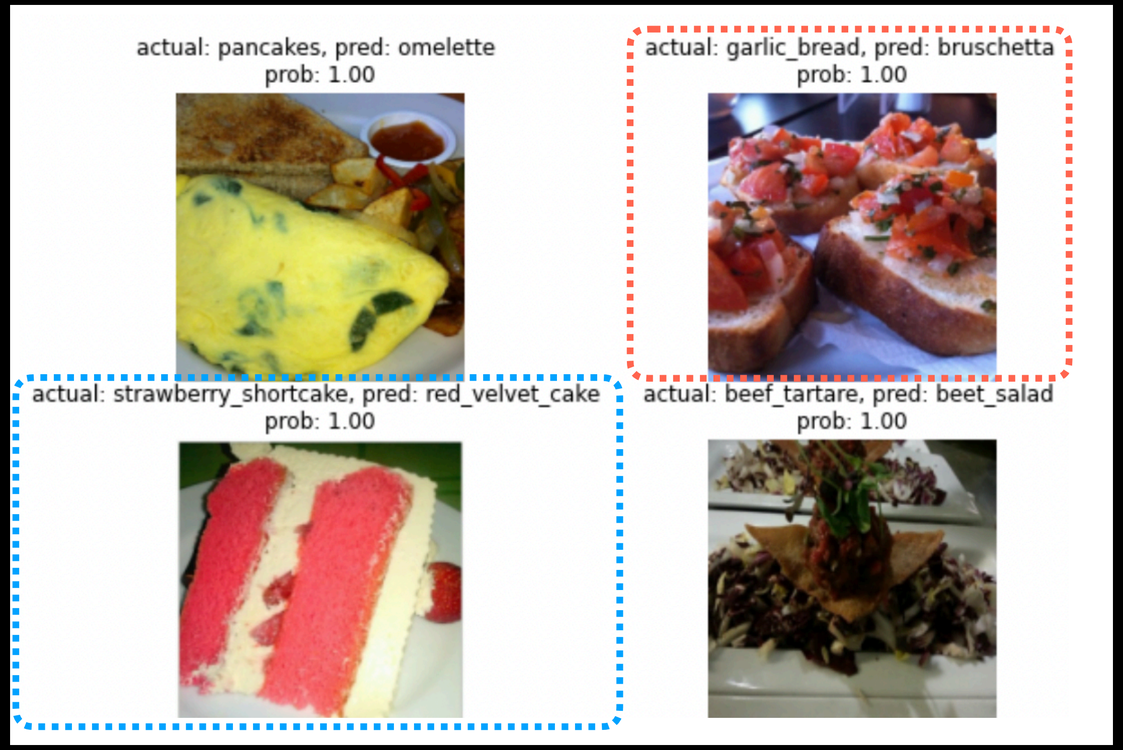
Classification evaluation methods

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

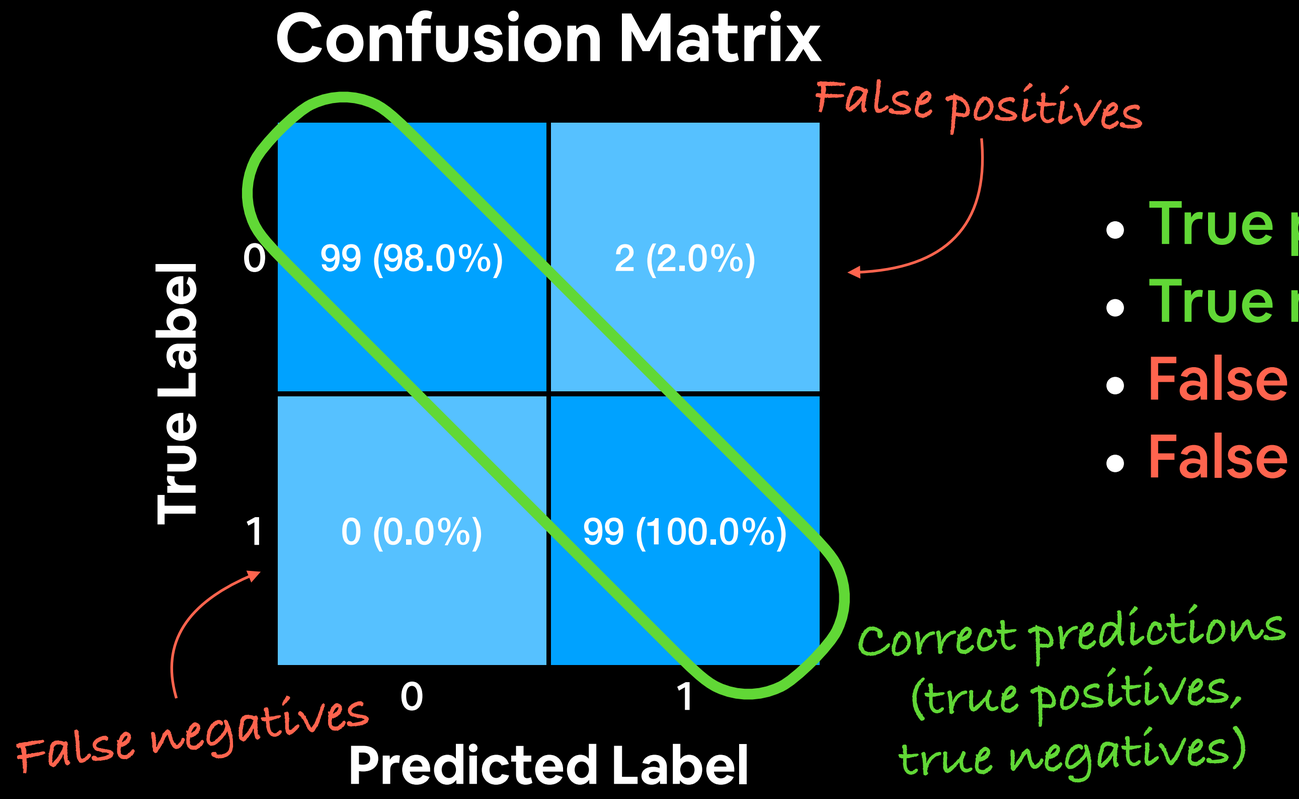
- A good way to inspect your model's performance is to view the wrong predictions with the highest prediction probability (or highest loss)
- Can reveal insights such as:
  - Data issues (wrong labels, e.g. model is right, label is wrong)
  - Confusing classes (get better/more diverse data)



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