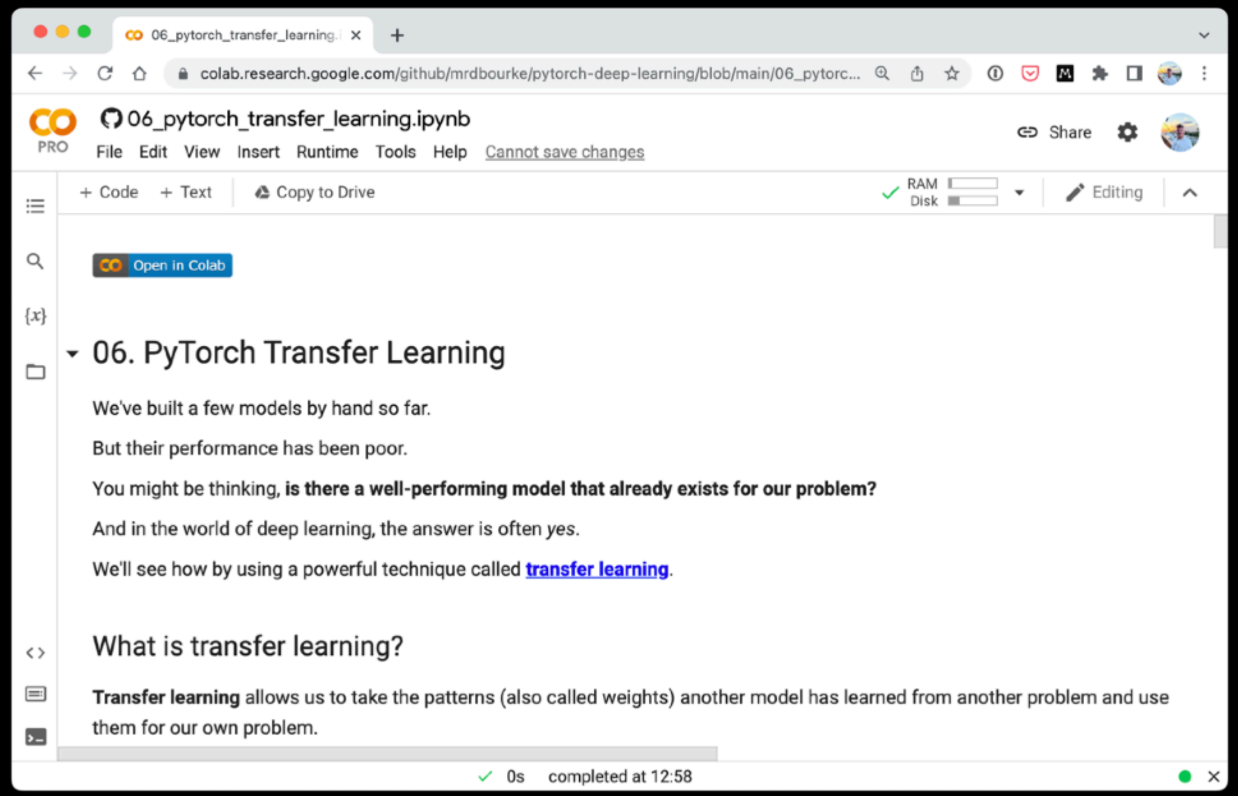
Transfer Learning with

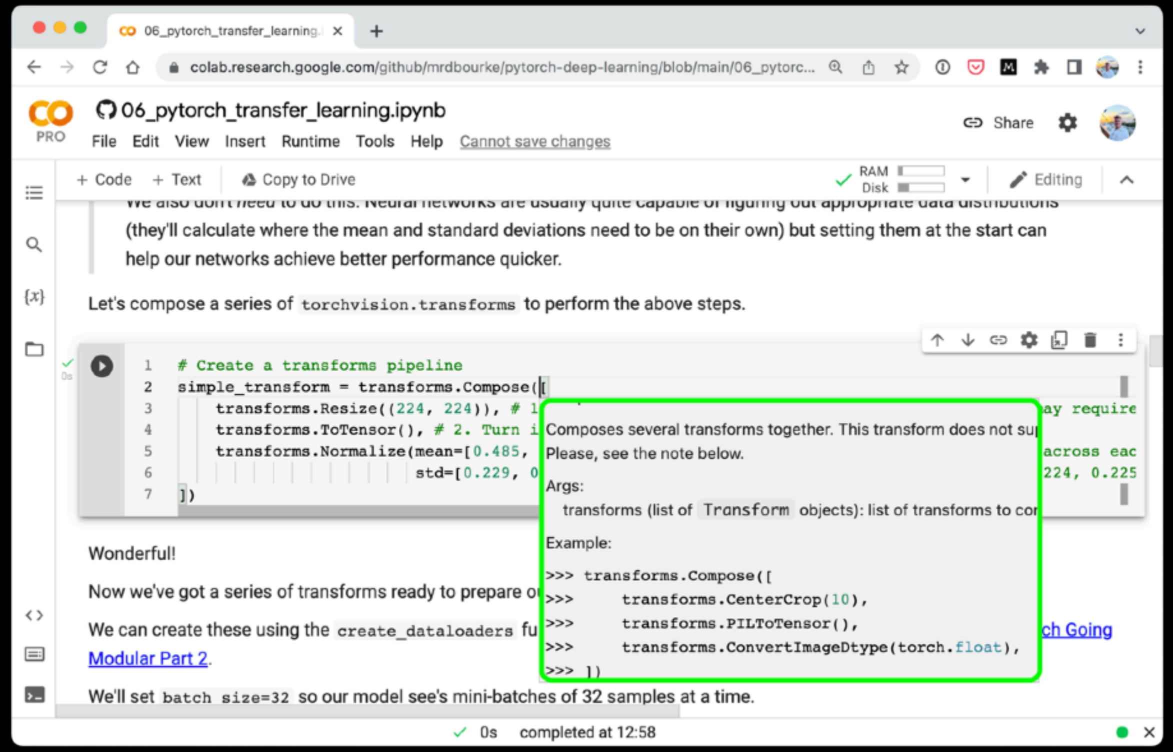
PyTorch

Where can you get help?

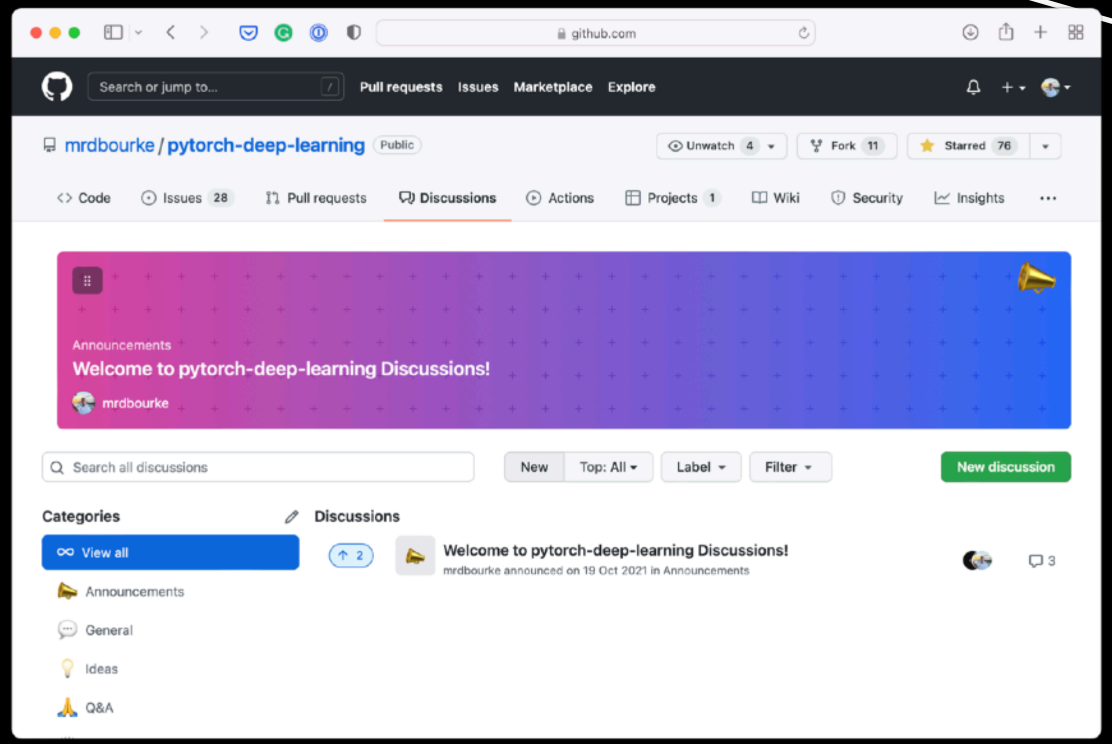


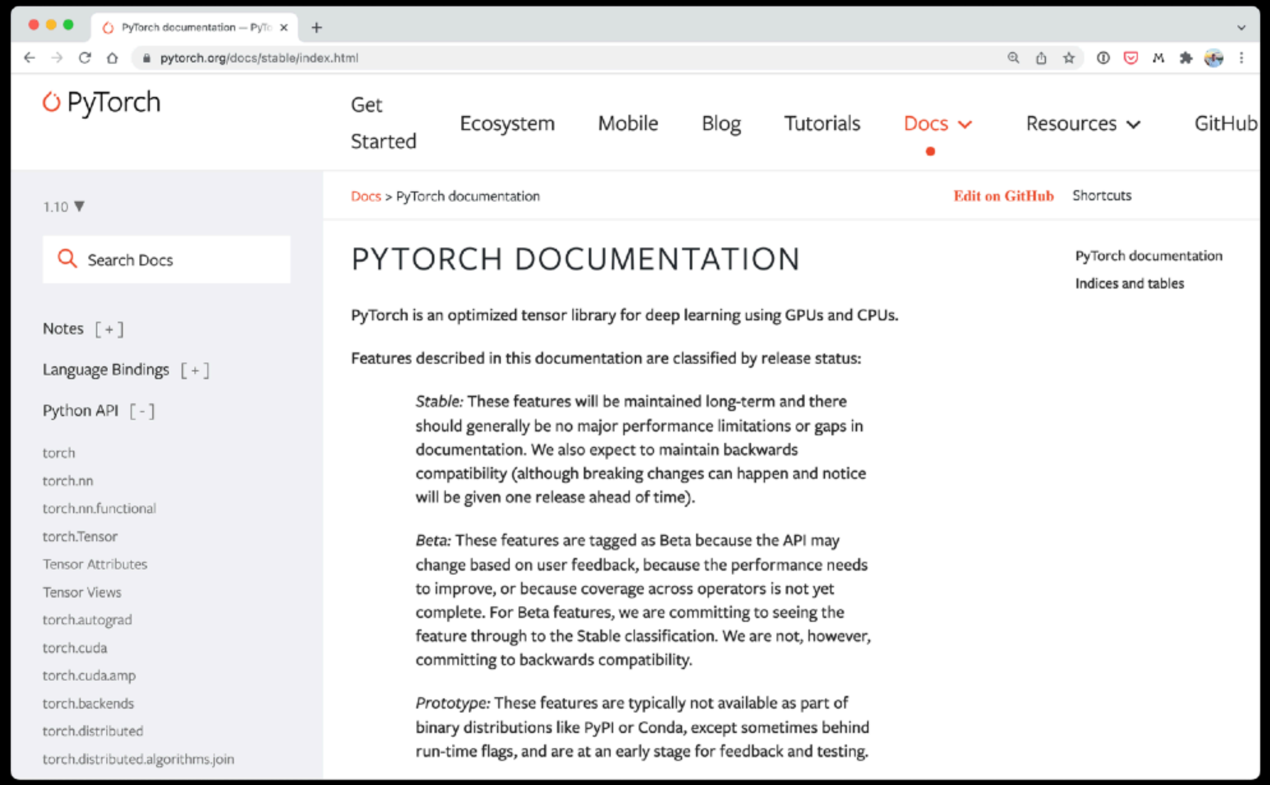
"If in doubt, run the code"

- Follow along with the code
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring
- Search for it
- Try again
- Ask



stackoverflow





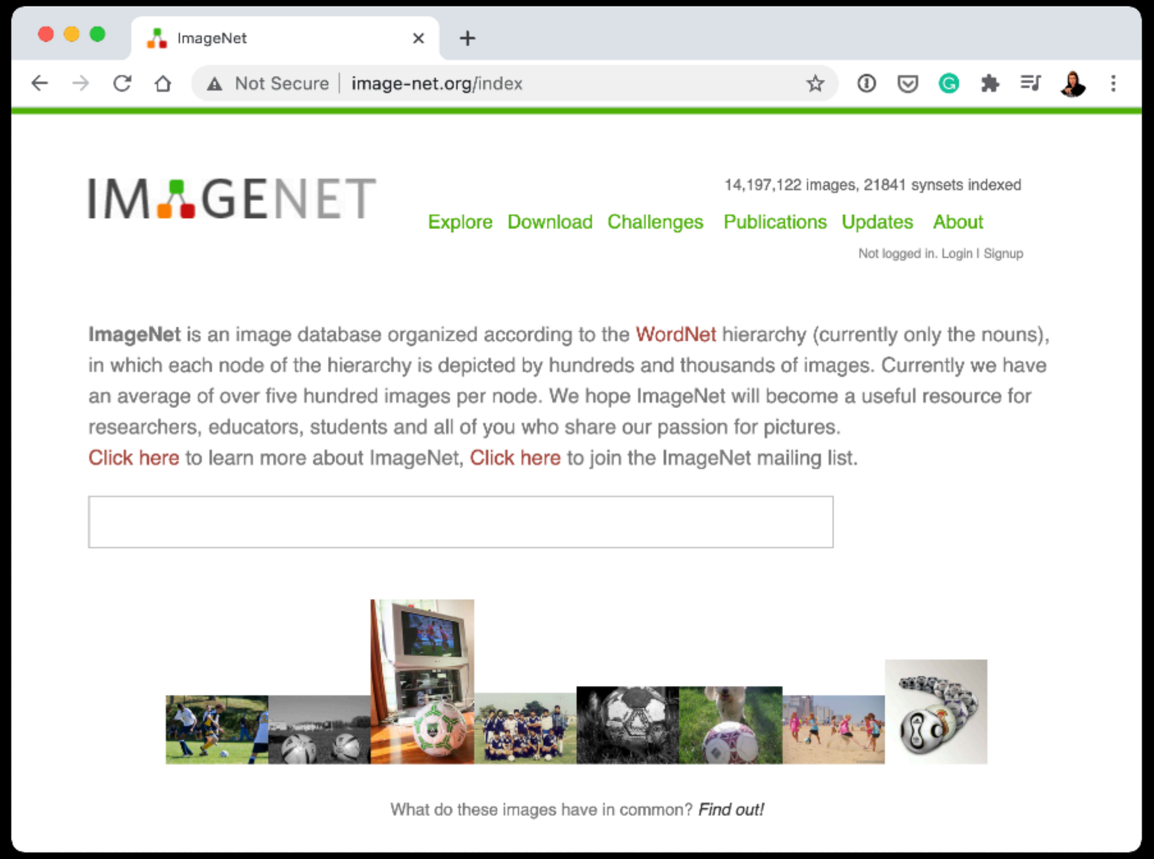
https://www.github.com/mrdbourke/pytorch-deep-learning/discussions

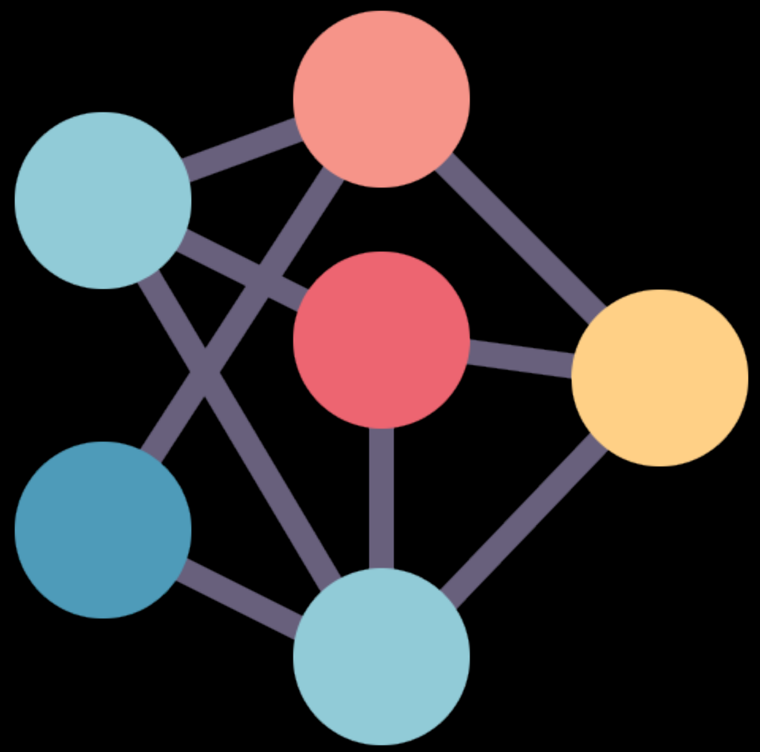
“What is transfer learning?”

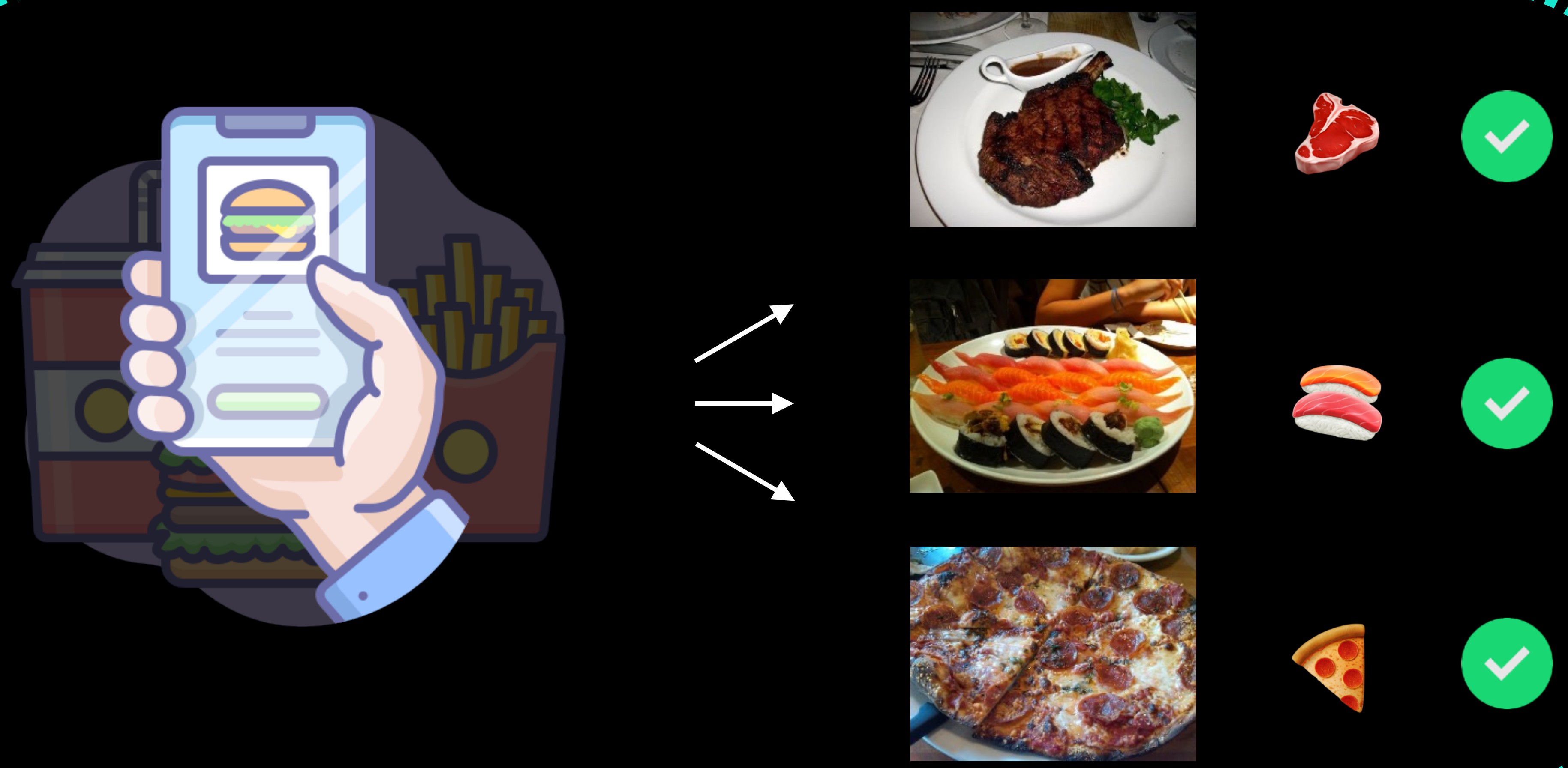
Surely someone has spent the time crafting the right model for the job...

Example transfer learning use cases

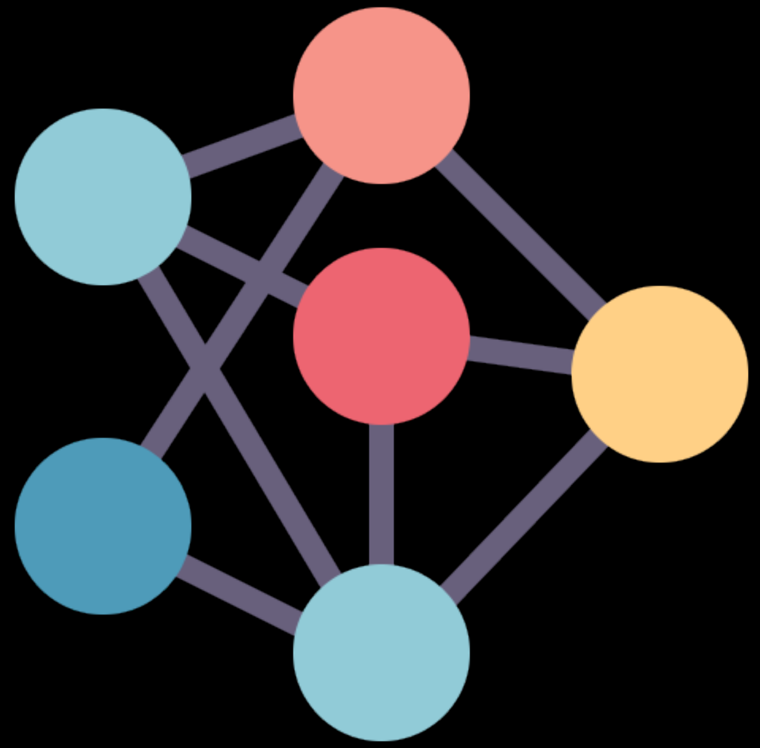
Computer vision







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!	To: daniel@mrdbourke.com Hay daniel... C0ongratu1ations! U win $1139239230
Not spam	Spam

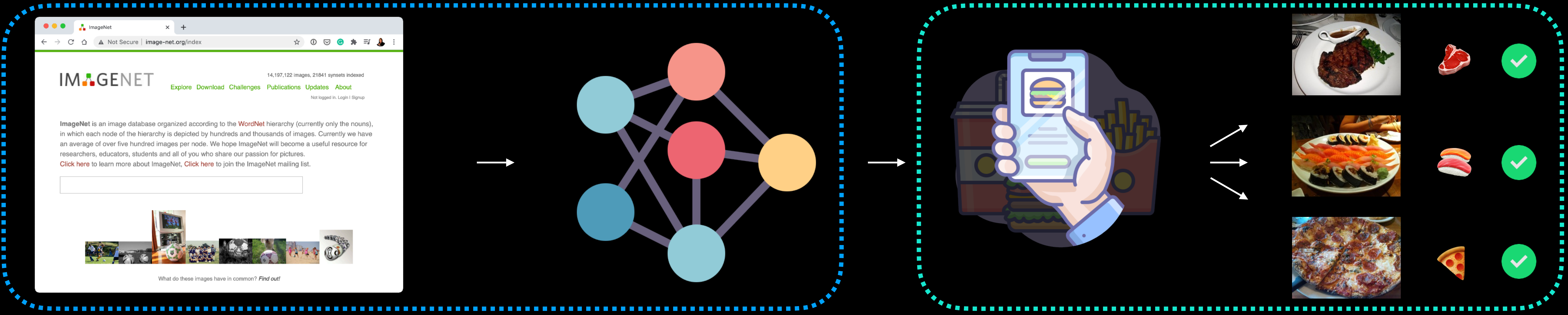
Model learns patterns/weights from similar problem space

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)

Pretrained EfficientNet architecture (already works really well on computer vision tasks)

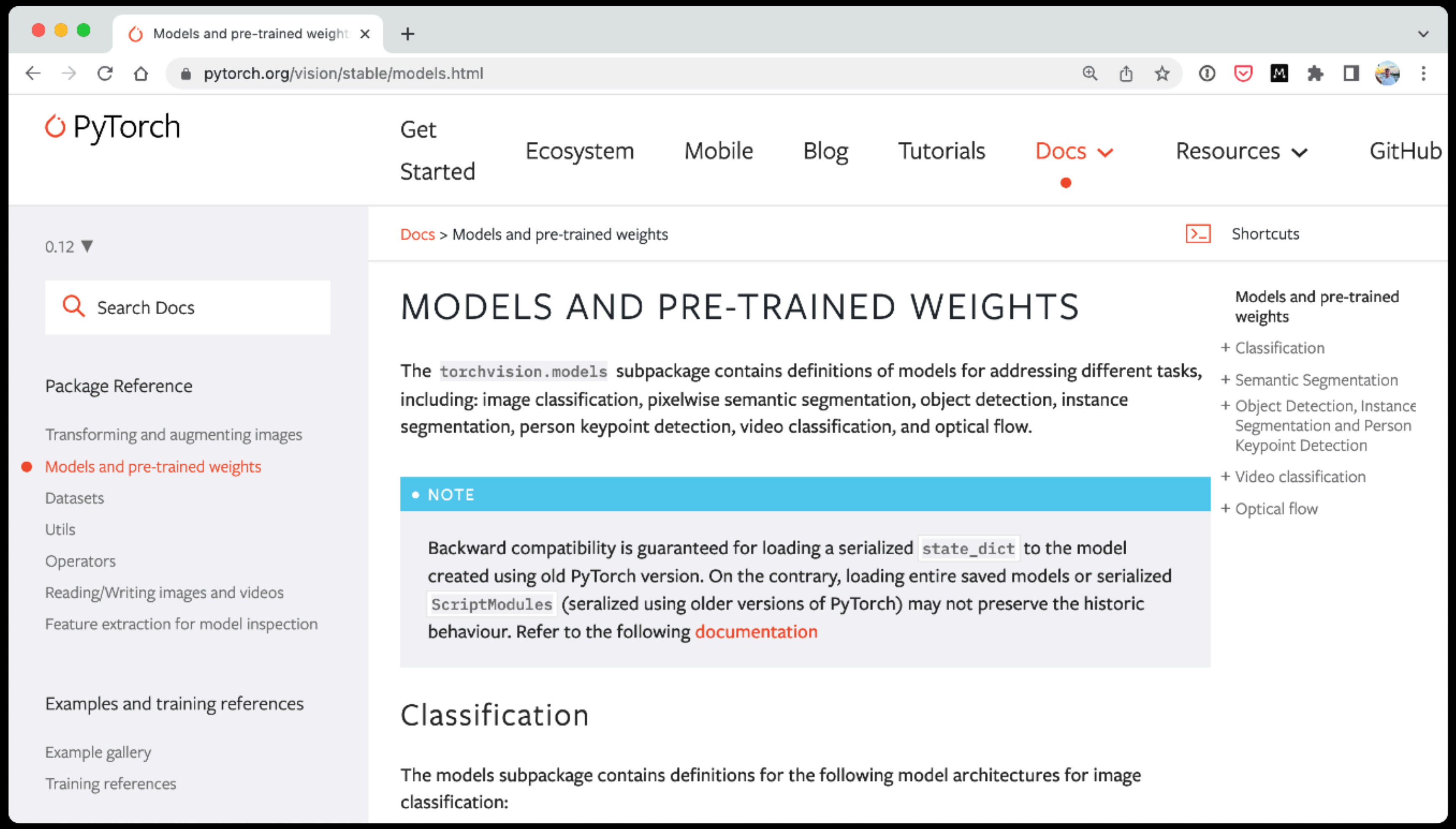
Extract/tune patterns/weights to suit our own problem (FoodVision Mini)

Model performs better than from scratch

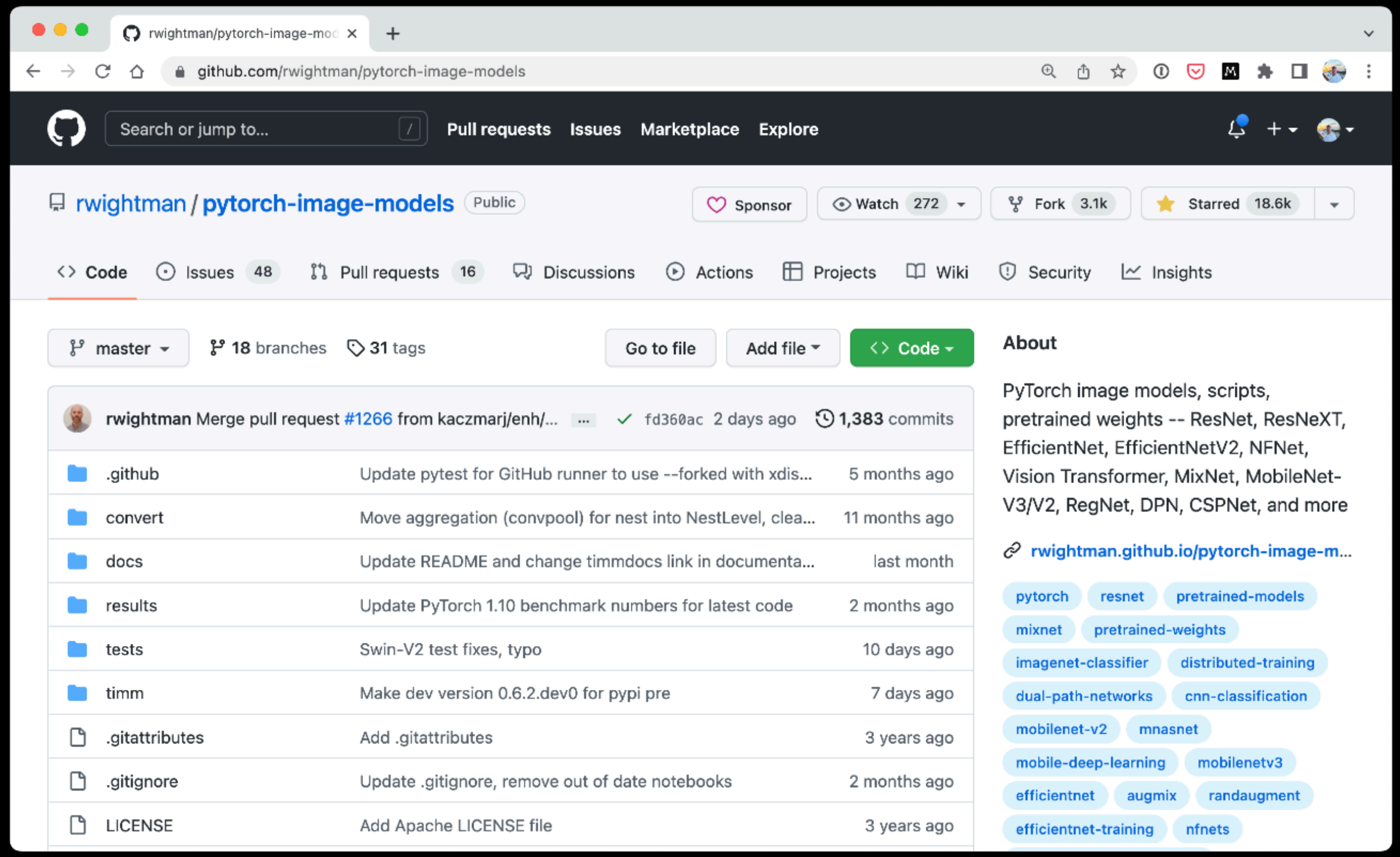
Improving a model

Method to improve a model (reduce overfitting)	What does it do?
More data	Gives a model more of a chance to learn patterns between samples (e.g. if a model is performing poorly on images of pizza, show it more images of pizza).
Data augmentation	Increase the diversity of your training dataset without collecting more data (e.g. take your photos of pizza and randomly rotate them 30°). Increased diversity forces a model to learn more generalisation patterns.
Better data	Not all data samples are created equally. Removing poor samples from or adding better samples to your dataset can improve your model's performance.
Use transfer learning	Take an existing model's pre-learned patterns from one problem and tweak them to suit your own problem. For example, take a model trained on pictures of cars to recognise pictures of trucks.

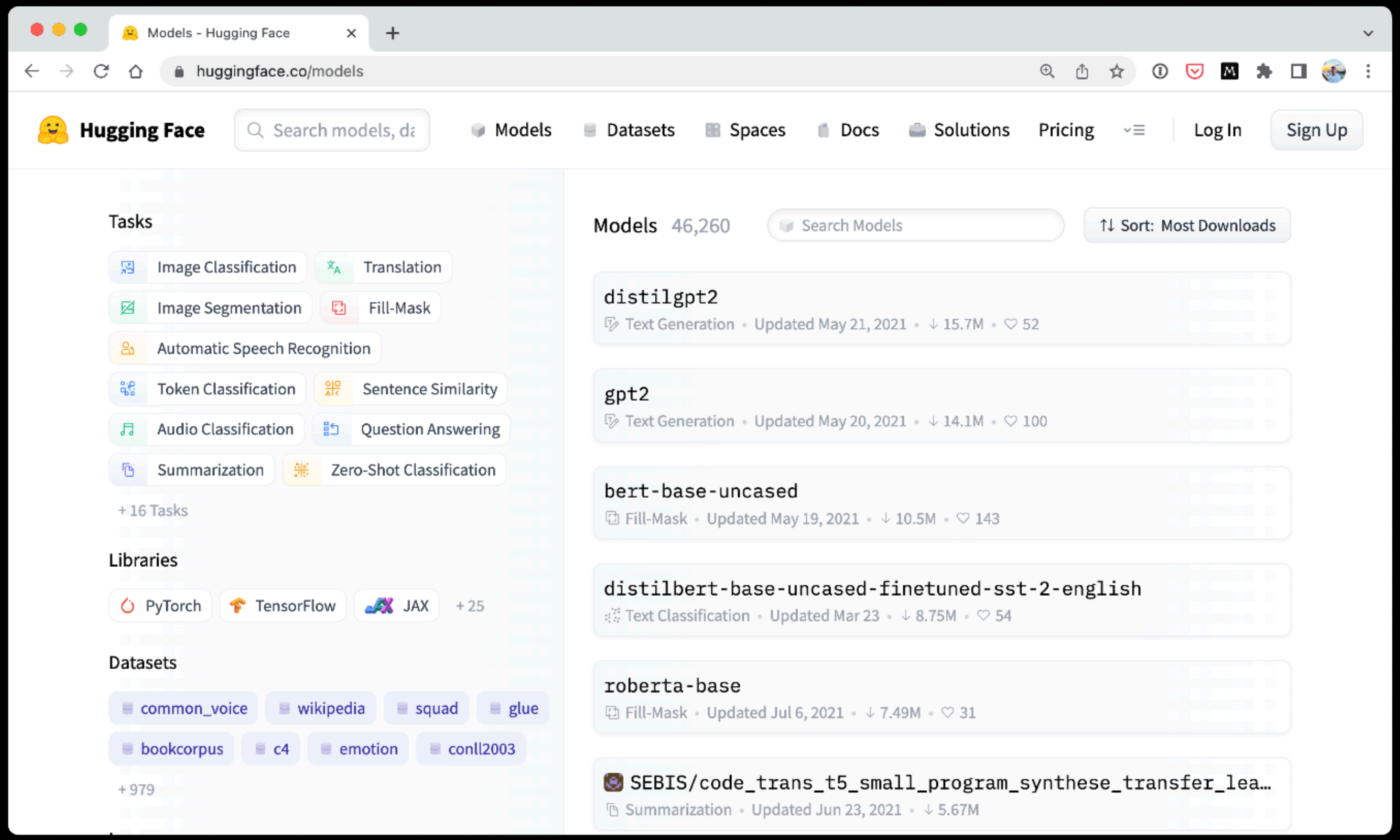
Where to find pretrained models



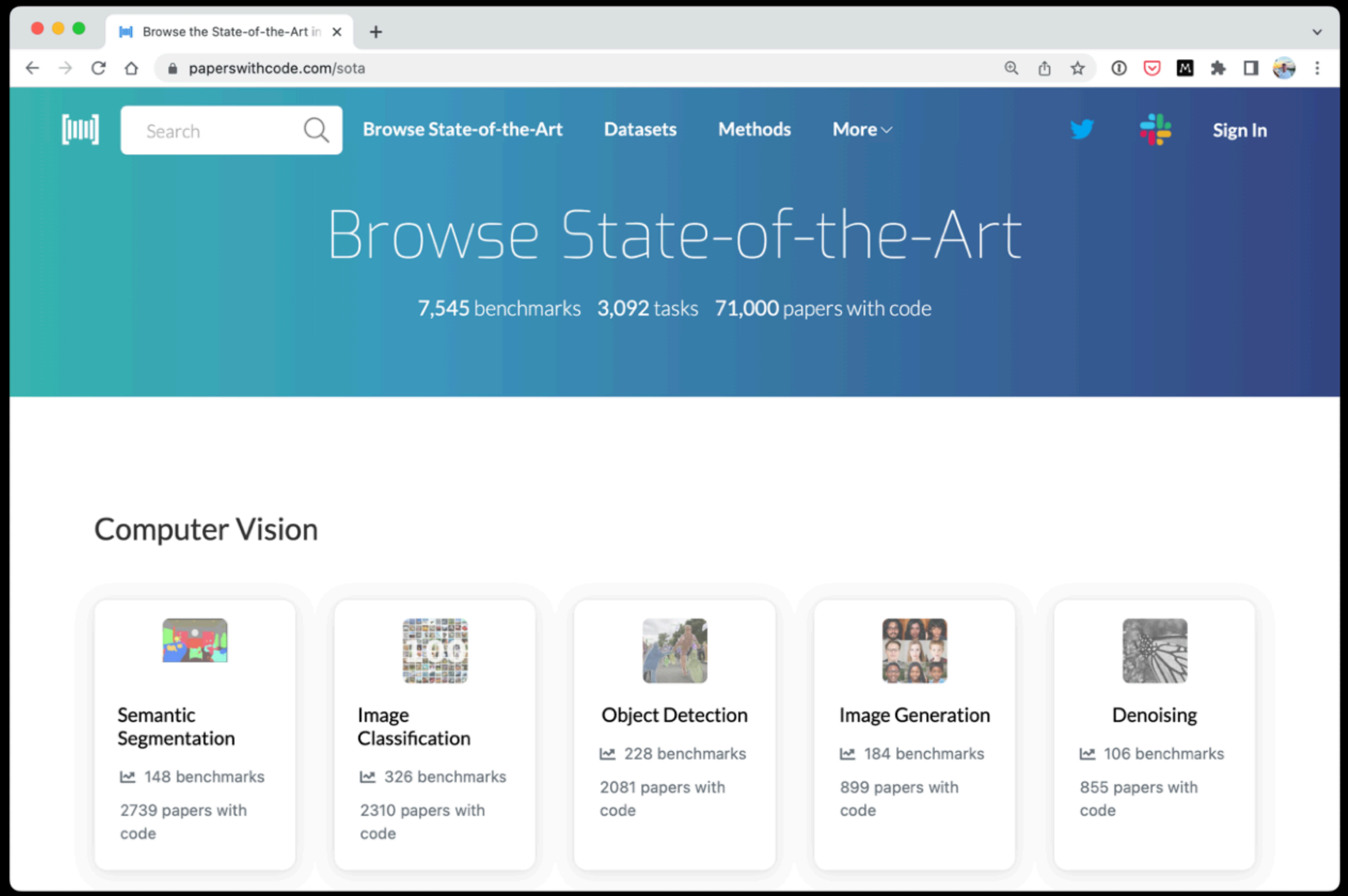
PyTorch domains libraries (`torchvision`, `torchtext`, `torchaudio`, `torchrec`). **Source:** https://pytorch.org/vision/stable/models.html



Torch Image Models (`timm` library).
Source: https://github.com/rwightman/pytorch-image-models



🤗 **HuggingFace Hub.**
Source: https://huggingface.co/models



Paperswithcode SOTA.
Source: https://paperswithcode.com/sota

What we're going to cover

(broadly)

- Getting setup (importing previously written code)
- Introduce transfer learning with PyTorch
- Customise a pretrained model for our own use case

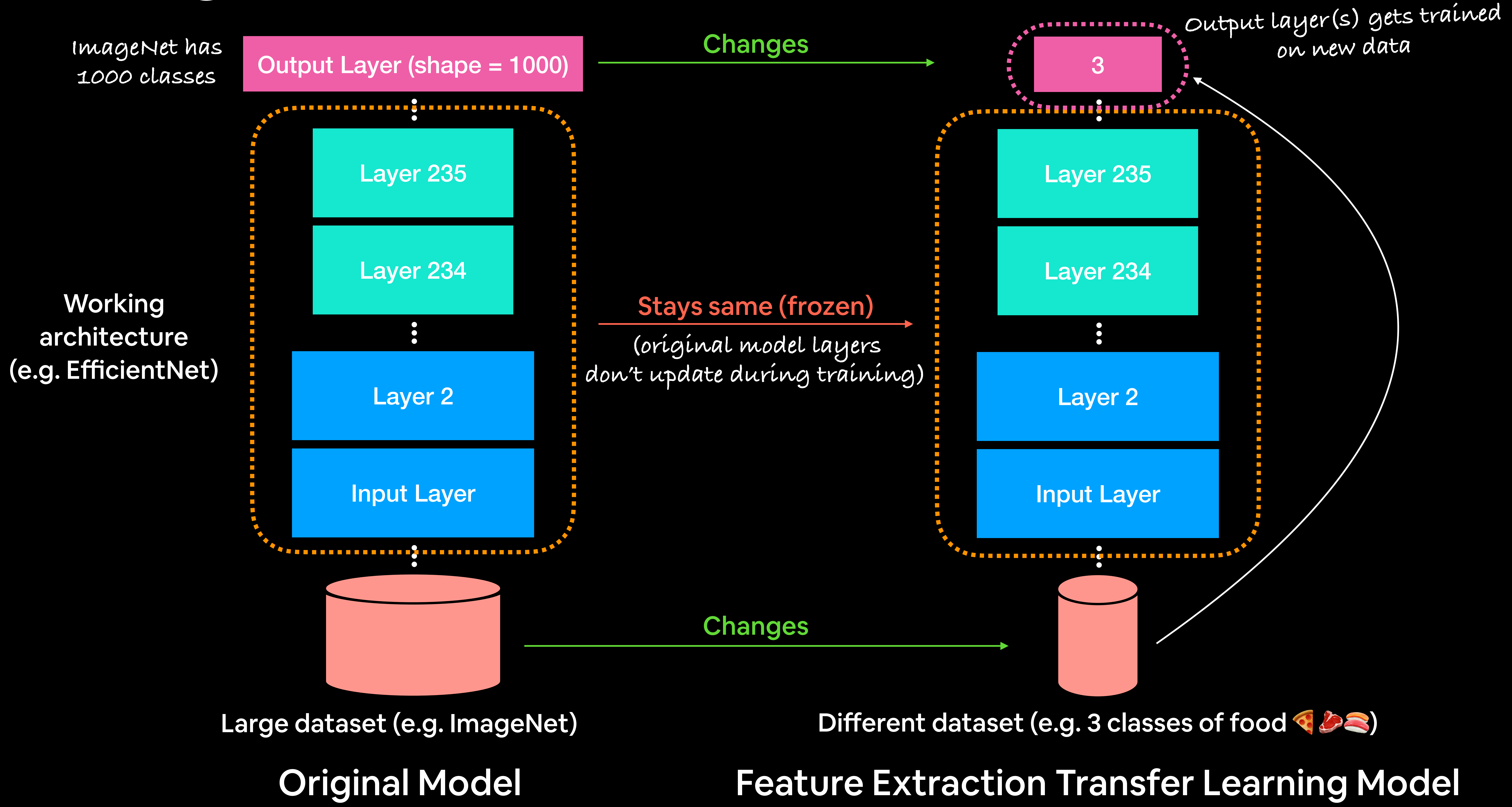
  (FoodVision Mini 🍕🥩🍣)
- Evaluating a transfer learning model
- Making predictions on our own custom data

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

How: 👩‍🍳 👩‍🔬

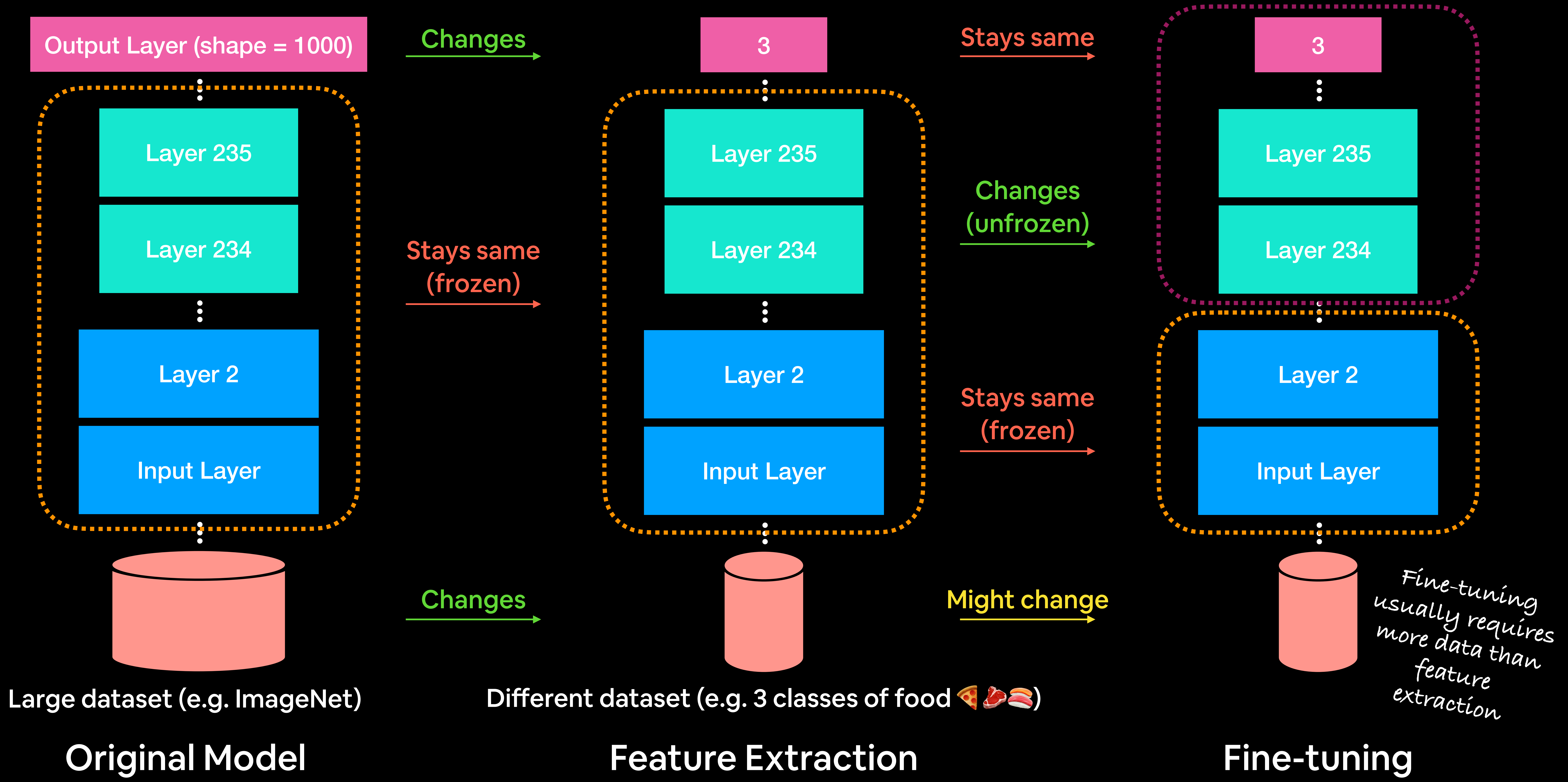
Let’s code!

Original Model vs. Feature Extraction



Kinds of Transfer Learning

Top layers get trained on new data



Kinds of Transfer Learning

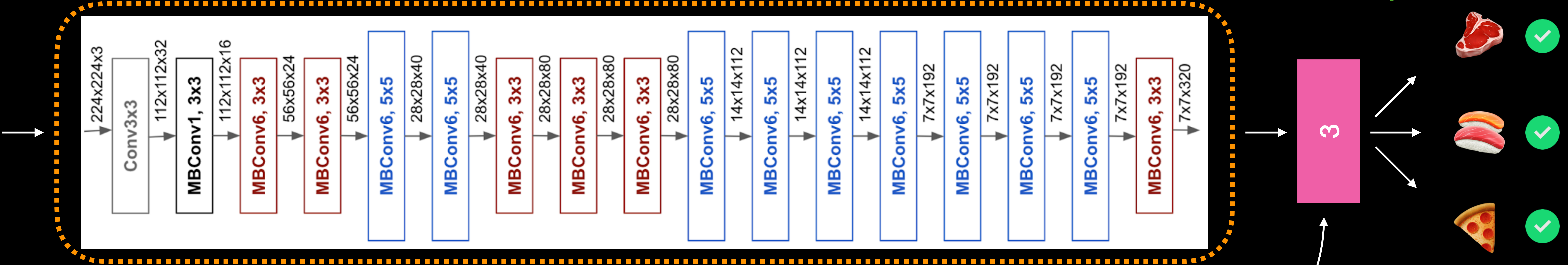
Type	Description	What happens	When to use
Original model ("As is")	Take a pretrained model as it is and apply it to your task without any changes.	The original model **remains unchanged**.	Helpful if you have the **exact same kind of data** the original model was trained on.
Feature extraction	Take the 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.

EfficientNet feature extractor

Input data
(Pizza, Steak, Sushi)

Stays same
(frozen, pretrained on ImageNet)

Changes
(same shape as number of classes)



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

EfficientNetB0 Backbone
(torchvision.models.efficientnet_b0)

Linear classifier layer
(torch.nn.Linear)

EfficientNet feature extractor

EfficientNetB0 Backbone

(torchvision.models.efficientnet_b0(prertained=True)

```
EfficientNet(
  (features): Sequential(
    (0): ConvNormActivation(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True)
      (2): SiLU(inplace=True)
    )
    (1): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): ConvNormActivation(
            (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True)
            (2): SiLU(inplace=True)
  ...
  ...
  ...
  (avgpool): AdaptiveAvgPool2d(output_size=1)
  (classifier): Sequential(
    (0): Dropout(p=0.2, inplace=True)
    (1): Linear(in_features=1280, out_features=1000, bias=True)
  )
)
```

Extracts features from image

Turns features into a feature vector (by taking the average)

Turns feature vector into prediction logits

Can adjust depending on the number of classes you have

EfficientNet feature extractor — changing the classifier head

EfficientNetB0 Backbone

(torchvision.models.efficientnet_b0(pretained=True)

```
EfficientNet(
  (features): Sequential(
    (0): ConvNormActivation(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True)
      (2): SiLU(inplace=True)
    )
    (1): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): ConvNormActivation(
            (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True)
            (2): SiLU(inplace=True)
  ...
  ...
  ...
  (avgpool): AdaptiveAvgPool2d(output_size=1)
  (classifier): Sequential(
    (0): Dropout(p=0.2, inplace=True)
    (1): Linear(in_features=1280, out_features=1000, bias=True)
  )
)
```

Same

Changed

```
EfficientNet(
  (features): Sequential(
    (0): ConvNormActivation(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True)
      (2): SiLU(inplace=True)
    )
    (1): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): ConvNormActivation(
            (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True)
            (2): SiLU(inplace=True)
  ...
  ...
  ...
  (avgpool): AdaptiveAvgPool2d(output_size=1)
  (classifier): Sequential(
    (0): Dropout(p=0.2, inplace=True)
    (1): Linear(in_features=1280, out_features=3, bias=True)
  )
)
```

Original Model

(1000 output classes for ImageNet)

Original Model + Changed Classifier Head

(3 output classes for 🍕, 🥩, 🍣)

torchinfo.summary(model, input_size=(32, 3, 224, 224))

```
============================================================================================================================
Layer (type (var_name))                                      Input Shape          Output Shape         Param #              Trainable
============================================================================================================================
EfficientNet                                                 --                   --                   --                   True
├─Sequential (features)                                      [32, 3, 224, 224]    [32, 1280, 7, 7]     --                   True
│    └─ConvNormActivation (0)                                [32, 3, 224, 224]    [32, 32, 112, 112]   --                   True
│    │    └─Conv2d (0)                                       [32, 3, 224, 224]    [32, 32, 112, 112]   864                  True
│    │    └─BatchNorm2d (1)                                  [32, 32, 112, 112]   [32, 32, 112, 112]   64                   True
│    │    └─SiLU (2)                                         [32, 32, 112, 112]   [32, 32, 112, 112]   --                   --
│    └─Sequential (1)                                        [32, 32, 112, 112]   [32, 16, 112, 112]   --                   True
│    │    └─MBConv (0)                                       [32, 32, 112, 112]   [32, 16, 112, 112]   1,448                True
│    └─Sequential (2)                                        [32, 16, 112, 112]   [32, 24, 56, 56]     --                   True
│    │    └─MBConv (0)                                       [32, 16, 112, 112]   [32, 24, 56, 56]     6,004                True
│    │    └─MBConv (1)                                       [32, 24, 56, 56]     [32, 24, 56, 56]     10,710               True
│    └─Sequential (3)                                        [32, 24, 56, 56]     [32, 40, 28, 28]     --                   True
│    │    └─MBConv (0)                                       [32, 24, 56, 56]     [32, 40, 28, 28]     15,350               True
│    │    └─MBConv (1)                                       [32, 40, 28, 28]     [32, 40, 28, 28]     31,290               True
│    └─Sequential (4)                                        [32, 40, 28, 28]     [32, 80, 14, 14]     --                   True
│    │    └─MBConv (0)                                       [32, 40, 28, 28]     [32, 80, 14, 14]     37,130               True
│    │    └─MBConv (1)                                       [32, 80, 14, 14]     [32, 80, 14, 14]     102,900              True
│    │    └─MBConv (2)                                       [32, 80, 14, 14]     [32, 80, 14, 14]     102,900              True
│    └─Sequential (5)                                        [32, 80, 14, 14]     [32, 112, 14, 14]    --                   True
│    │    └─MBConv (0)                                       [32, 80, 14, 14]     [32, 112, 14, 14]    126,004              True
│    │    └─MBConv (1)                                       [32, 112, 14, 14]    [32, 112, 14, 14]    208,572              True
│    │    └─MBConv (2)                                       [32, 112, 14, 14]    [32, 112, 14, 14]    208,572              True
│    └─Sequential (6)                                        [32, 112, 14, 14]    [32, 192, 7, 7]      --                   True
│    │    └─MBConv (0)                                       [32, 112, 14, 14]    [32, 192, 7, 7]      262,492              True
│    │    └─MBConv (1)                                       [32, 192, 7, 7]      [32, 192, 7, 7]      587,952              True
│    │    └─MBConv (2)                                       [32, 192, 7, 7]      [32, 192, 7, 7]      587,952              True
│    │    └─MBConv (3)                                       [32, 192, 7, 7]      [32, 192, 7, 7]      587,952              True
│    └─Sequential (7)                                        [32, 192, 7, 7]      [32, 320, 7, 7]      --                   True
│    │    └─MBConv (0)                                       [32, 192, 7, 7]      [32, 320, 7, 7]      717,232              True
│    └─ConvNormActivation (8)                                [32, 320, 7, 7]      [32, 1280, 7, 7]     --                   True
│    │    └─Conv2d (0)                                       [32, 320, 7, 7]      [32, 1280, 7, 7]     409,600              True
│    │    └─BatchNorm2d (1)                                  [32, 1280, 7, 7]     [32, 1280, 7, 7]     2,560                True
│    │    └─SiLU (2)                                         [32, 1280, 7, 7]     [32, 1280, 7, 7]     --                   --
├─AdaptiveAvgPool2d (avgpool)                                [32, 1280, 7, 7]     [32, 1280, 1, 1]     --                   --
├─Sequential (classifier)                                    [32, 1280]           [32, 1000]           --                   True
│    └─Dropout (0)                                           [32, 1280]           [32, 1280]           --                   --
│    └─Linear (1)                                            [32, 1280]           [32, 1000]           1,281,000            True
============================================================================================================================
Total params: 5,288,548
Trainable params: 5,288,548
Non-trainable params: 0
Total mult-adds (G): 12.35
============================================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 3452.35
Params size (MB): 21.15
Estimated Total Size (MB): 3492.77
============================================================================================================================
```

Are the layers trainable? (unfrozen)

Input shape of data per layer

Output shape of data per layer

Total number of parameters and trainable parameters

```
torchinfo.summary(model, input_size=(32, 3, 224, 224))
```

```
============================================================================================================================================
Layer (type (var_name))                                      Input Shape          Output Shape         Param #              Trainable
============================================================================================================================================
EfficientNet                                                 --                   --                   --                   Partial
├─Sequential (features)                                      [32, 3, 224, 224]    [32, 1280, 7, 7]     --                   False
│    └─ConvNormActivation (0)                                [32, 3, 224, 224]    [32, 32, 112, 112]   --                   False
│    │    └─Conv2d (0)                                       [32, 3, 224, 224]    [32, 32, 112, 112]   (864)                False
│    │    └─BatchNorm2d (1)                                  [32, 32, 112, 112]   [32, 32, 112, 112]   (64)                 False
│    │    └─SiLU (2)                                         [32, 32, 112, 112]   [32, 32, 112, 112]   --                   --
│    └─Sequential (1)                                        [32, 32, 112, 112]   [32, 16, 112, 112]   --                   False
│    │    └─MBConv (0)                                       [32, 32, 112, 112]   [32, 16, 112, 112]   (1,448)              False
│    └─Sequential (2)                                        [32, 16, 112, 112]   [32, 24, 56, 56]     --                   False
│    │    └─MBConv (0)                                       [32, 16, 112, 112]   [32, 24, 56, 56]     (6,004)              False
│    │    └─MBConv (1)                                       [32, 24, 56, 56]     [32, 24, 56, 56]     (10,710)             False
│    └─Sequential (3)                                        [32, 24, 56, 56]     [32, 40, 28, 28]     --                   False
│    │    └─MBConv (0)                                       [32, 24, 56, 56]     [32, 40, 28, 28]     (15,350)             False
│    │    └─MBConv (1)                                       [32, 40, 28, 28]     [32, 40, 28, 28]     (31,290)             False
│    └─Sequential (4)                                        [32, 40, 28, 28]     [32, 80, 14, 14]     --                   False
│    │    └─MBConv (0)                                       [32, 40, 28, 28]     [32, 80, 14, 14]     (37,130)             False
│    │    └─MBConv (1)                                       [32, 80, 14, 14]     [32, 80, 14, 14]     (102,900)            False
│    │    └─MBConv (2)                                       [32, 80, 14, 14]     [32, 80, 14, 14]     (102,900)            False
│    └─Sequential (5)                                        [32, 80, 14, 14]     [32, 112, 14, 14]    --                   False
│    │    └─MBConv (0)                                       [32, 80, 14, 14]     [32, 112, 14, 14]    (126,004)            False
│    │    └─MBConv (1)                                       [32, 112, 14, 14]    [32, 112, 14, 14]    (208,572)            False
│    │    └─MBConv (2)                                       [32, 112, 14, 14]    [32, 112, 14, 14]    (208,572)            False
│    └─Sequential (6)                                        [32, 112, 14, 14]    [32, 192, 7, 7]      --                   False
│    │    └─MBConv (0)                                       [32, 112, 14, 14]    [32, 192, 7, 7]      (262,492)            False
│    │    └─MBConv (1)                                       [32, 192, 7, 7]      [32, 192, 7, 7]      (587,952)            False
│    │    └─MBConv (2)                                       [32, 192, 7, 7]      [32, 192, 7, 7]      (587,952)            False
│    │    └─MBConv (3)                                       [32, 192, 7, 7]      [32, 192, 7, 7]      (587,952)            False
│    └─Sequential (7)                                        [32, 192, 7, 7]      [32, 320, 7, 7]      --                   False
│    │    └─MBConv (0)                                       [32, 192, 7, 7]      [32, 320, 7, 7]      (717,232)            False
│    └─ConvNormActivation (8)                                [32, 320, 7, 7]      [32, 1280, 7, 7]     --                   False
│    │    └─Conv2d (0)                                       [32, 320, 7, 7]      [32, 1280, 7, 7]     (409,600)            False
│    │    └─BatchNorm2d (1)                                  [32, 1280, 7, 7]     [32, 1280, 7, 7]     (2,560)              False
│    │    └─SiLU (2)                                         [32, 1280, 7, 7]     [32, 1280, 7, 7]     --                   --
├─AdaptiveAvgPool2d (avgpool)                                [32, 1280, 7, 7]     [32, 1280, 1, 1]     --                   --
├─Sequential (classifier)                                    [32, 1280]           [32, 3]              --                   True
│    └─Dropout (0)                                           [32, 1280]           [32, 1280]           --                   --
│    └─Linear (1)                                            [32, 1280]           [32, 3]              3,843                True
============================================================================================================================================
Total params: 4,011,391
Trainable params: 3,843
Non-trainable params: 4,007,548
Total mult-adds (G): 12.31
============================================================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 3452.09
Params size (MB): 16.05
Estimated Total Size (MB): 3487.41
============================================================================================================================================
```

Many layers untrainable (frozen)

Only last layers are trainable

Final layer output (same as number of classes 🍕🥩🍣)

Less trainable parameters because many layers are frozen