

Going modular with



Where can you get help?

- Follow along with the code

```
*writerfile going_modular/model_builder.py
"""
Contains Pytorch model code to instantiate a TinyVGG model.
"""

import torch
from torch import nn

class TinyVGG(nn.Module):
    """Creates the TinyVGG architecture.

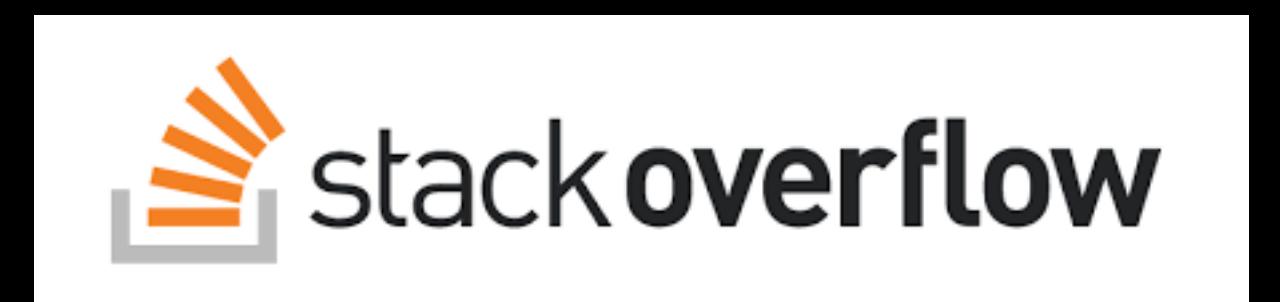
Replicates the TinyVGG architecture from the CNN explainer website in PyTorch.
See the original architecture here: https://poloclub.github.io/cnn-explainer/

Args:
    input_shape: An integer indicating number of input channels.
    hidden_units: An integer indicating number of hidden units between layers.
    output_shape: An integer indicating number of output units.

    """
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int) -> None:
        super().__init__()
        self.conv_block_1 = nn.Sequential(
            nn.Conv2d(in_channels=input_shape,
                      out_channels=hidden_units,
```

- Try it for yourself

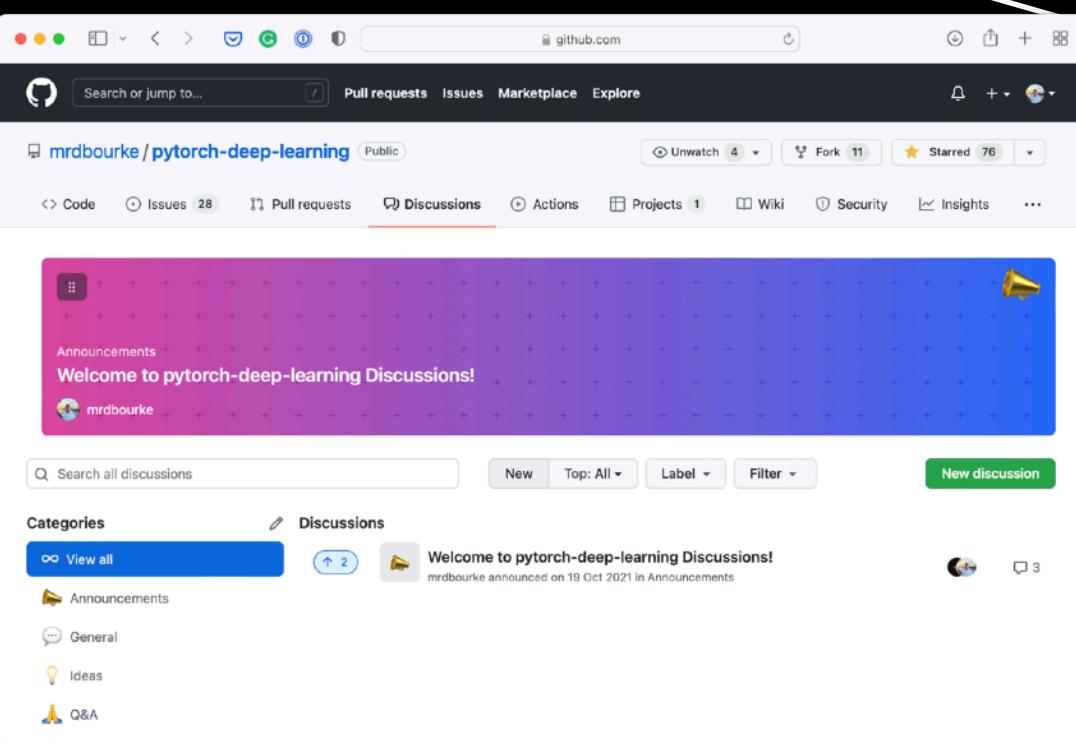
- Press SHIFT + CMD + SPACE to read the docstring



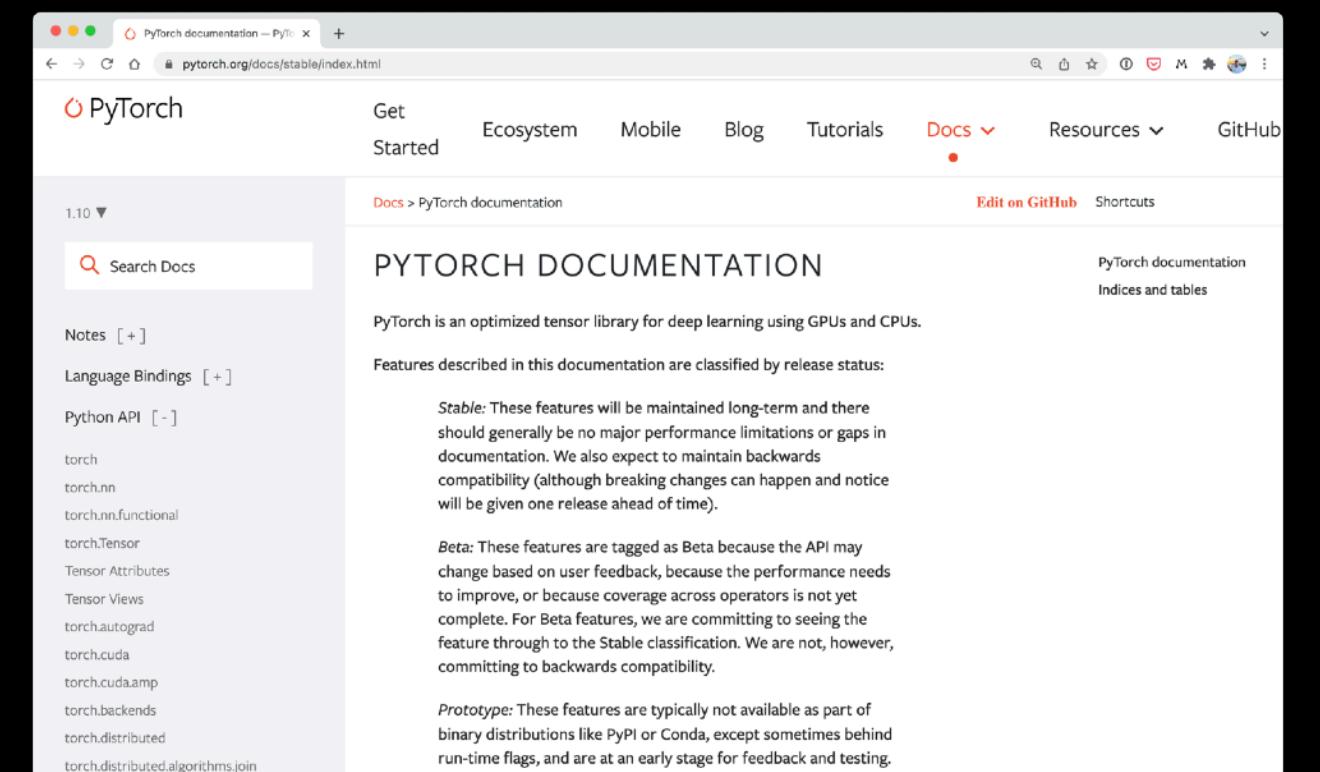
- Search for it

- Try again

- Ask



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



"If in doubt, run the code"

```
name: StrOrBytesPath, mode: int = ..., exist_ok: bool = ...)
```

Since we're going to be creating a leaf directory and all intermediate ones. We generate a folder for storing those scripts.

```
mkdir(exist_ok=True)
```

Super-mkdir: create a leaf directory and all intermediate ones. Work like mkdir, except that any intermediate path segment (not just the rightmost) will be created if it does not exist. If the target directory already exists, raise an OSError if exist_ok is False. Otherwise no exception is raised. This is recursive.

```
os.makedirs('going_modular', exist_ok=True)
```

1. Get data

We're going to start by downloading the same data we used in [notebook 04](#), the pizza_steak_sushi dataset with images of pizza, steak and sushi.

```
1 Import
2 import requests
3 Import zipfile
4 From pathlib import Path
5
6 # Set up path to data folder
7 data_path = Path('data/')
8 image_path = data_path / 'pizza_steak_sushi'
9
10 # If the image folder doesn't exist, download it and prepare it...
11 if image_path.is_dir():
```

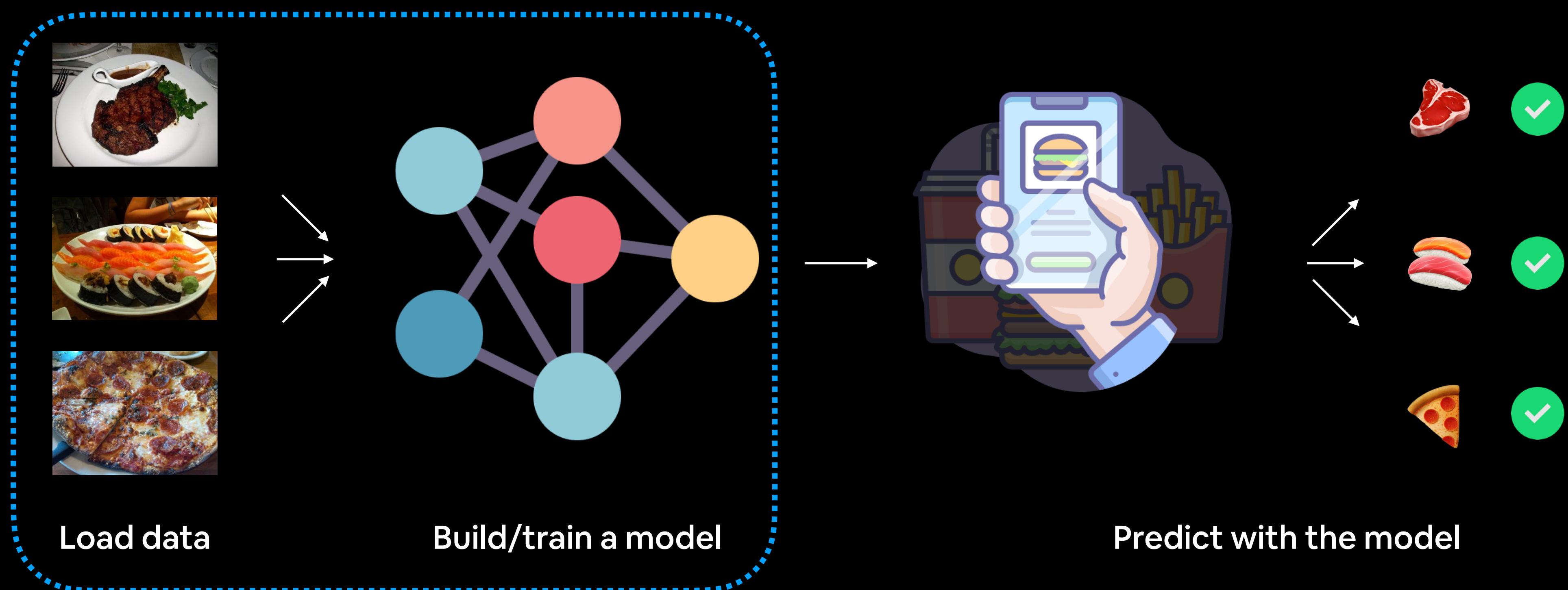
“What is going modular?”

“I’ve written some nice code in a notebook, can I reuse it elsewhere?”

Yes.

What we're going to build

FoodVision Mini

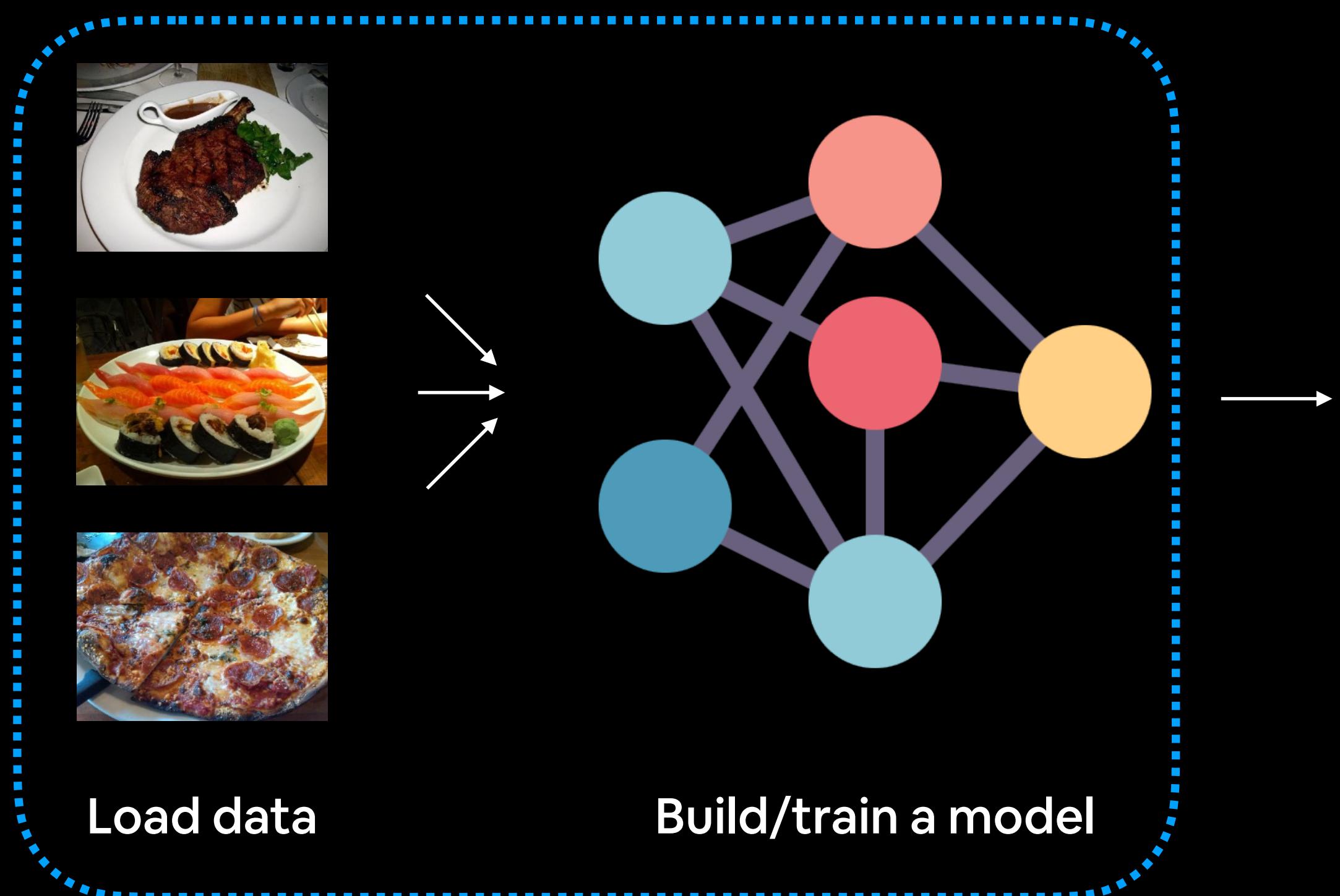


*we're going to turn the code to do
this from notebook cell code into a
series of Python scripts*

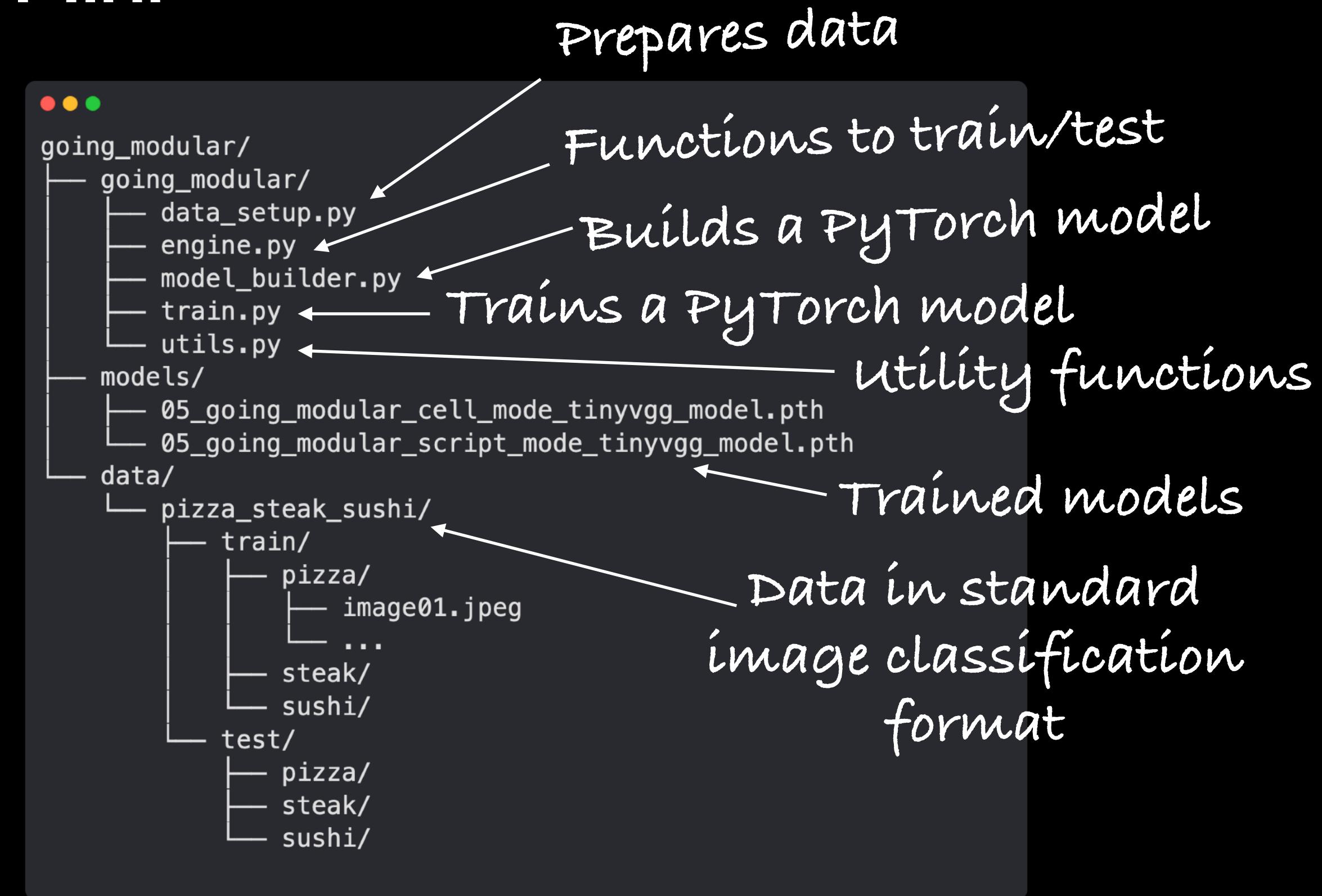


What we're going to build

FoodVision Mini

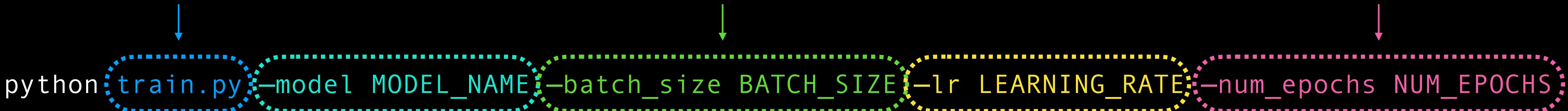


*we're going to turn the code to do
this from notebook cell code into a
series of Python scripts*



PyTorch from the command line

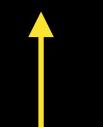
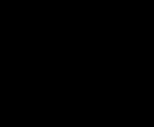
Target Python script



Model to train

How big should the batch size be?

Train for how long?



What should the learning rate be?

```
python train.py -model tinyvgg -batch_size 32 -lr 0.001 -num_epochs 10
```

Note: there are many more
hyperparameters you could add here

“Train the TinyVGG model with a batch size of 32
and a learning rate of 0.001 for 10 epochs.”

PyTorch in the wild

(examples of Python scripts)

Training & Evaluation in Command Line

We provide two scripts in “tools/plain_train_net.py” and “tools/train_net.py”, that are made to train all the configs provided in detectron2. You may want to use it as a reference to write your own training script.

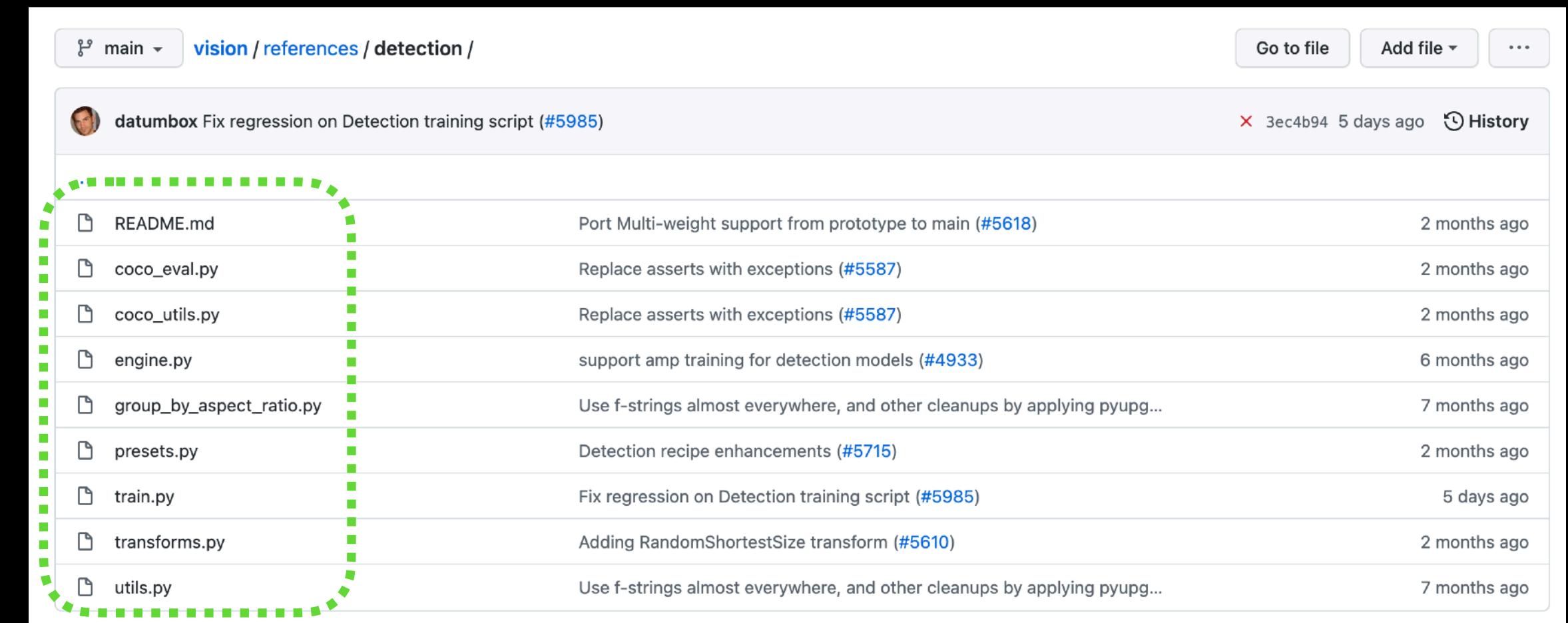
Compared to “train_net.py”, “plain_train_net.py” supports fewer default features. It also includes fewer abstraction, therefore is easier to add custom logic.

To train a model with “train_net.py”, first setup the corresponding datasets following [datasets/README.md](#), then run:

```
cd tools/
./train_net.py --num-gpus 8 \
--config-file ../configs/COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_1x.yaml
```

The configs are made for 8-GPU training. To train on 1 GPU, you may need to [change some parameters](#), e.g.:

```
./train_net.py \
--config-file ../configs/COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_1x.yaml \
--num-gpus 1 SOLVER.IMS_PER_BATCH 2 SOLVER.BASE_LR 0.0025
```



Source: [torchvision object detection GitHub](#).

Source: [Detectron2 documentation](#).

Using our standard [training reference script](#), we can train a ResNet50 using the following command:

```
torchrun --nproc_per_node=8 train.py --model resnet50 --batch-size 128 --lr 0.5 \
--lr-scheduler cosineannealinglr --lr-warmup-epochs 5 --lr-warmup-method linear \
--auto-augment ta_wide --epochs 600 --random-erase 0.1 --weight-decay 0.00002 \
--norm-weight-decay 0.0 --label-smoothing 0.1 --mixup-alpha 0.2 --cutmix-alpha 1.0 \
--train-crop-size 176 --model-ema --val-resize-size 232 --ra-sampler --ra-reps 4
```

Fine-tuning

Download the pretrained model from [here](#).

To finetune with multi-node distributed training, run the following on 4 nodes with 8 GPUs each:

```
python submitit_finetune.py \
--job_dir ${JOB_DIR} \
--nodes 4 \
--batch_size 32 \
--model convvit_base_patch16 \
--finetune ${PRETRAIN_CHKPT} \
--epochs 100 \
--blr 5e-4 --layer_decay 0.65 \
--weight_decay 0.05 --drop_path 0.1 --reprob 0.25 --mixup 0.8 --cutmix 1.0 \
--dist_eval --data_path ${IMAGENET_DIR}
```

Source: [ConvMAE paper GitHub](#).

Source: Training state-of-the-art computer vision models with [torchvision](#) from the PyTorch blog.

My workflow

(one of many options)

(experiment, experiment, experiment!)



A screenshot of a Google Colab notebook titled "04_PyTorch_Custom_Datasets.ipynb". The notebook content discusses building custom datasets in PyTorch, specifically for computer vision tasks like food classification. It includes code snippets and examples from the PyTorch Domain Libraries. A progress bar at the bottom indicates the notebook is completed at 10:40.

Start with Jupyter/Google Colab notebooks



data_setup.py

```
import os

from torchvision import datasets, transforms
from torch.utils.data import DataLoader

NUM_WORKERS = os.cpu_count()

def create_dataloaders(train_dir: str, test_dir: str, transform: transforms.Compose,
                      batch_size: int, num_workers: int=NUM_WORKERS):
    """Creates training and testing DataLoaders.

    Args:
        train_dir: Path to training directory.
        test_dir: Path to testing directory.
        transform: torchvision transforms to perform on training and testing data.
        batch_size: Number of samples per batch in each of the DataLoaders.
        num_workers: An integer for number of workers per DataLoader.

    Returns:
        A tuple of (train_dataloader, test_dataloader, class_names).
        Where class_names is a list of the target classes.
    Example usage:
        train_dataloader, test_dataloader, class_names = \
            = create_dataloaders(train_dir=path/to/train_dir, test_dir=path/to/test_dir,
                               transform=some_transform, batch_size=32, num_workers=4)
    """
    # Use ImageFolder to create dataset(s)
    train_data = datasets.ImageFolder(train_dir, transform=transform)
    test_data = datasets.ImageFolder(test_dir, transform=transform)

    # Get class names
    class_names = train_data.classes

    # Turn images into data loaders
    train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True,
                                 num_workers=num_workers, pin_memory=True)
    test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=False,
                                num_workers=num_workers, pin_memory=True)

    return train_dataloader, test_dataloader, class_names
```

Move most useful code to Python scripts

Cell mode vs. Script mode

This screenshot shows a Google Colab notebook titled "05_pytorch_going_modular_cell_mode.ipynb". The notebook is in "Cell mode", indicated by the blue dashed box around the code cell. The code defines a "TinyVGG" class that replicates the architecture from the CNN explainer website. The code includes imports, a docstring, and a constructor method. A note at the bottom of the cell specifies the kernel size as 3.

```
1 import torch
2 from torch import nn
3
4 class TinyVGG(nn.Module):
5     """Creates the TinyVGG architecture.
6
7     Replicates the TinyVGG architecture from the CNN explainer website in PyTorch.
8     See the original architecture here: https://poloclub.github.io/cnn-explainer/
9
10 Args:
11     input_shape: An integer indicating number of input channels.
12     hidden_units: An integer indicating number of hidden units between layers.
13     output_shape: An integer indicating number of output units.
14 """
15 def __init__(self, input_shape: int, hidden_units: int, output_shape: int) -> None:
16     super().__init__()
17     self.conv_block_1 = nn.Sequential(
18         nn.Conv2d(in_channels=input_shape,
19                  out_channels=hidden_units,
20                  kernel_size=3, # how big is the square that's going over the image?
```

Notebook 05 Part 1: Cell mode

This screenshot shows the same Google Colab notebook as the previous one, but in "Script mode". The code is now part of a single Python script named "model_builder.py". The code is identical to the cell mode version, defining the "TinyVGG" class and its constructor. A note at the top of the script indicates it contains PyTorch model code to instantiate a TinyVGG model.

```
1 %%writefile going_modular/model_builder.py
2 """
3 Contains PyTorch model code to instantiate a TinyVGG model.
4 """
5 import torch
6 from torch import nn
7
8 class TinyVGG(nn.Module):
9     """Creates the TinyVGG architecture.
10
11     Replicates the TinyVGG architecture from the CNN explainer website in PyTorch.
12     See the original architecture here: https://poloclub.github.io/cnn-explainer/
13
14 Args:
15     input_shape: An integer indicating number of input channels.
16     hidden_units: An integer indicating number of hidden units between layers.
17     output_shape: An integer indicating number of output units.
18 """
19 def __init__(self, input_shape: int, hidden_units: int, output_shape: int) -> None:
20     super().__init__()
21     self.conv_block_1 = nn.Sequential(
22         nn.Conv2d(in_channels=input_shape,
23                  out_channels=hidden_units,
```

Notebook 05 Part 2: Script mode
(turns useful code into Python scripts)

`torchvision.transforms`

`torch.utils.data.Dataset`

`torch.utils.data.DataLoader`

`torchmetrics`

`torch.save`
`torch.load`



Each of these could be turned
into a Python script!



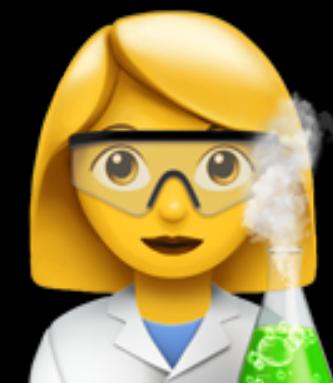
What we're going to cover

(broadly)

- Transforming data for use with a model
- Loading custom data with pre-built functions
- Building FoodVision Mini to classify  images
- Turning useful notebook code (all of the above) into Python scripts
- Training a PyTorch model from the command line

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

How:



Let's code!

Standard image classification data format

Your own data format
will depend on what
you're working

```
pizza_steak_sushi/ # <- overall dataset folder
  train/ # <- training images
    pizza/ # <- class name as folder name
      image01.jpeg
      image02.jpeg
      ...
    steak/
      image24.jpeg
      image25.jpeg
      ...
    sushi/
      image37.jpeg
      ...
  test/ # <- testing images
    pizza/
      image101.jpeg
      image102.jpeg
      ...
    steak/
      image154.jpeg
      image155.jpeg
      ...
    sushi/
      image167.jpeg
      ...
```

The premise remains:
write code to get your
data into tensors for
use with PyTorch