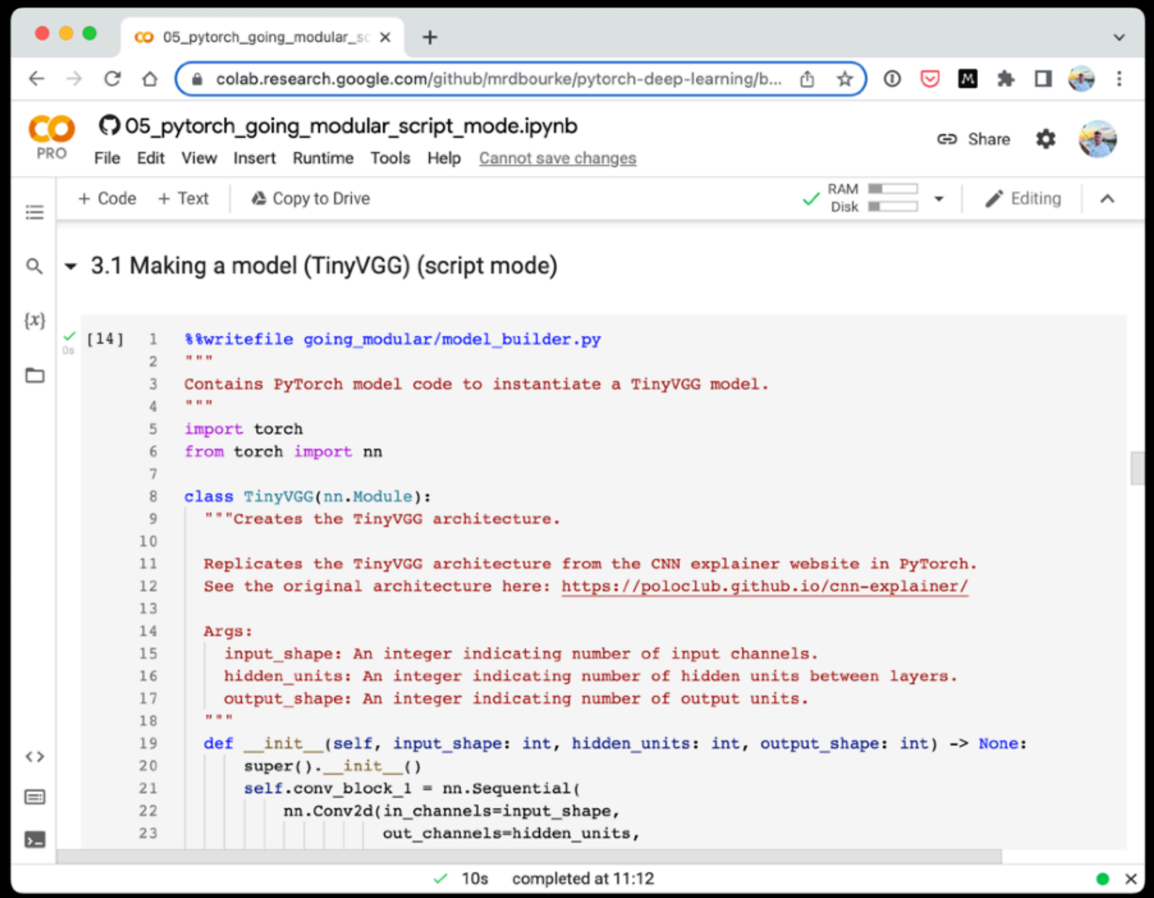
# Going modular with

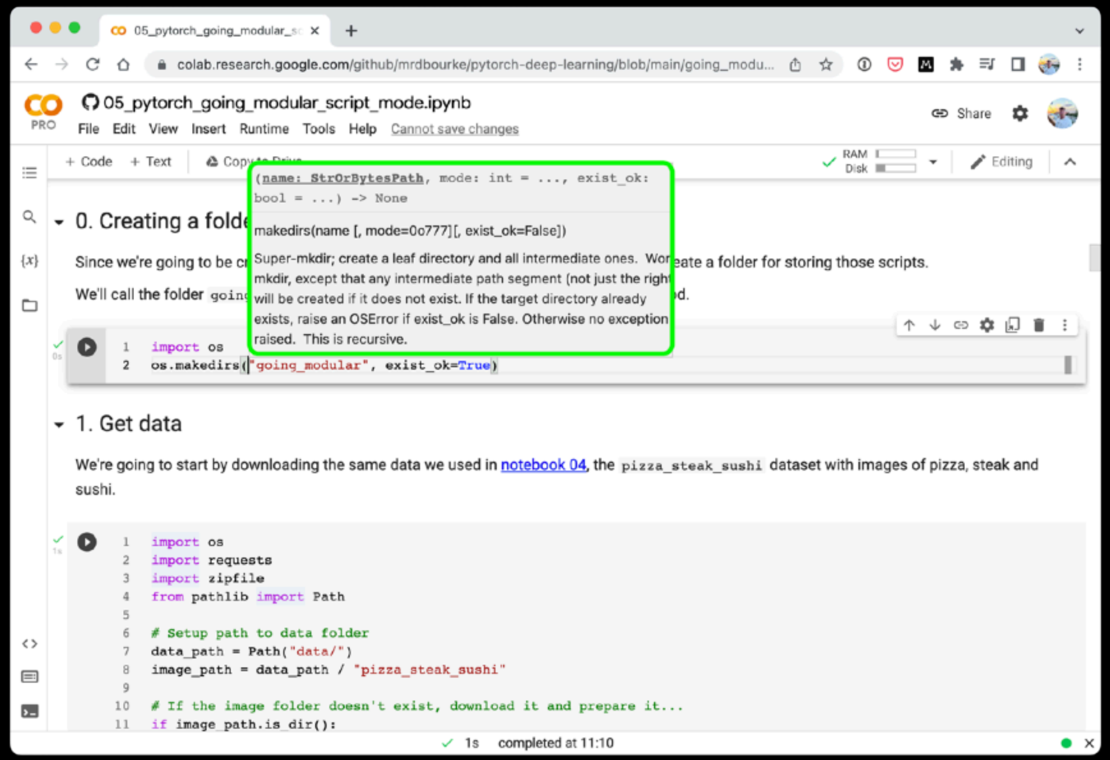
PyTorch

# Where can you get help?

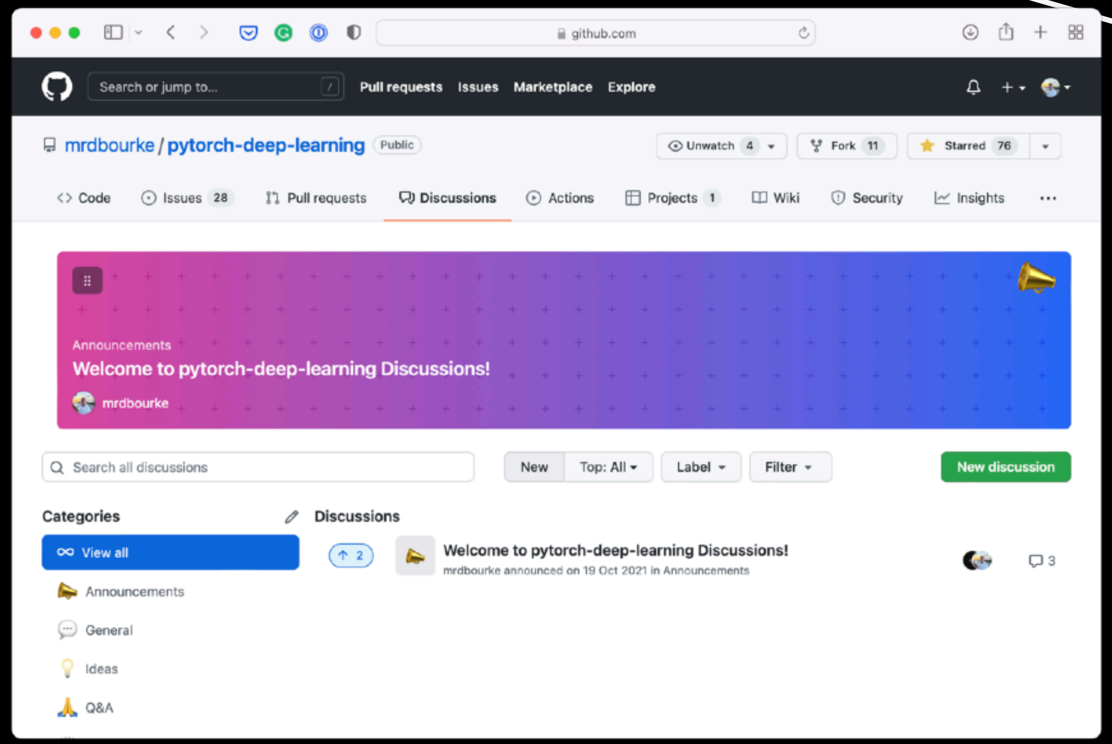
"If in doubt, run the code"

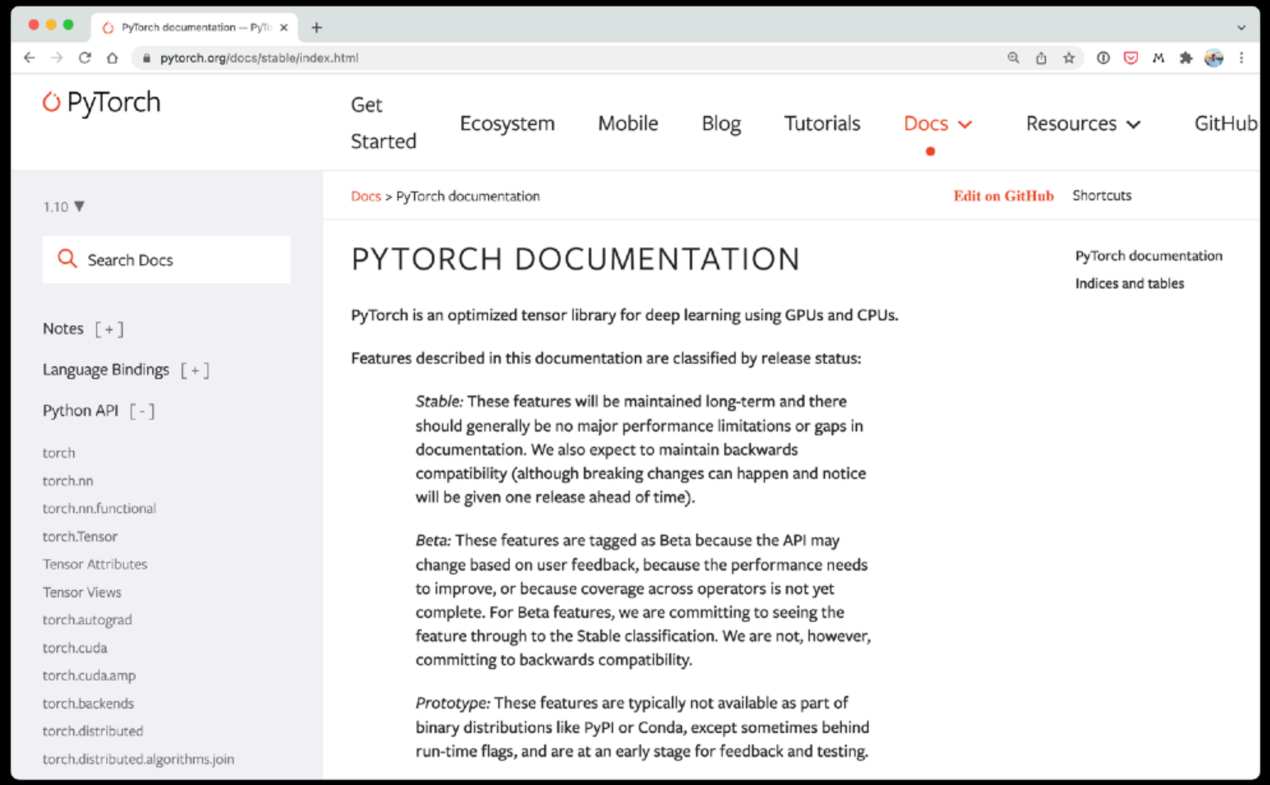
- Follow along with the code
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring
- Search for it
- Try again
- Ask




stackoverflow




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

# “What is going modular?”

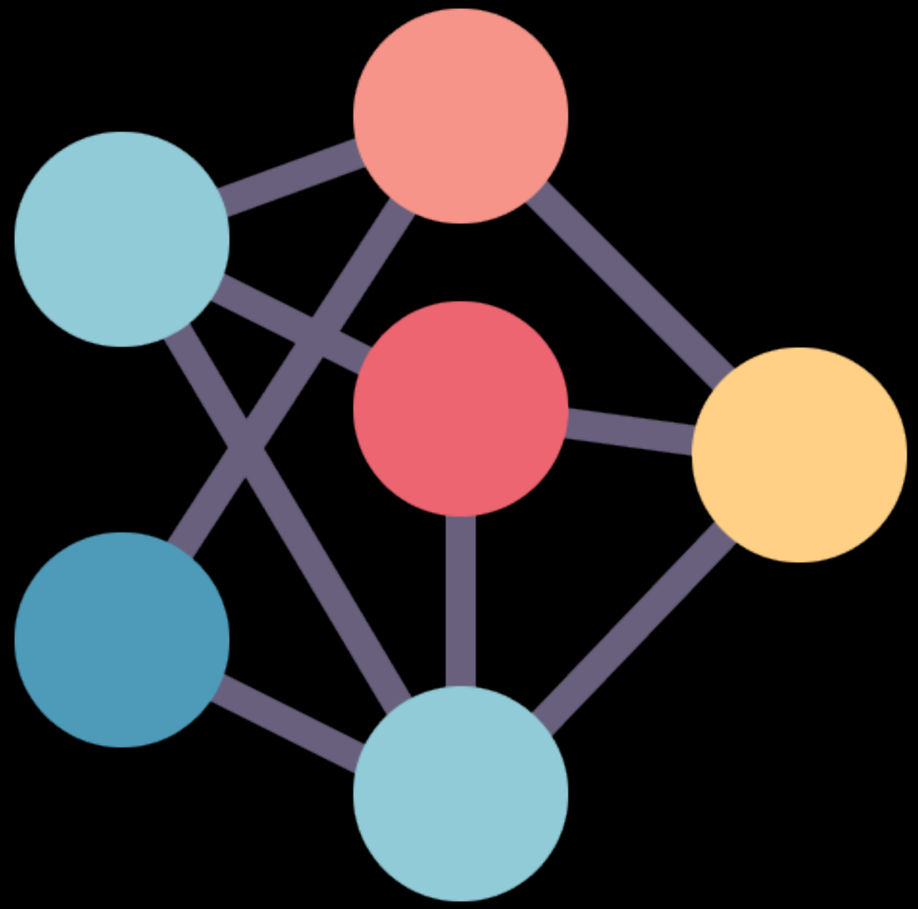
“I’ve written some nice code in a notebook, can I reuse it elsewhere?”

Yes.

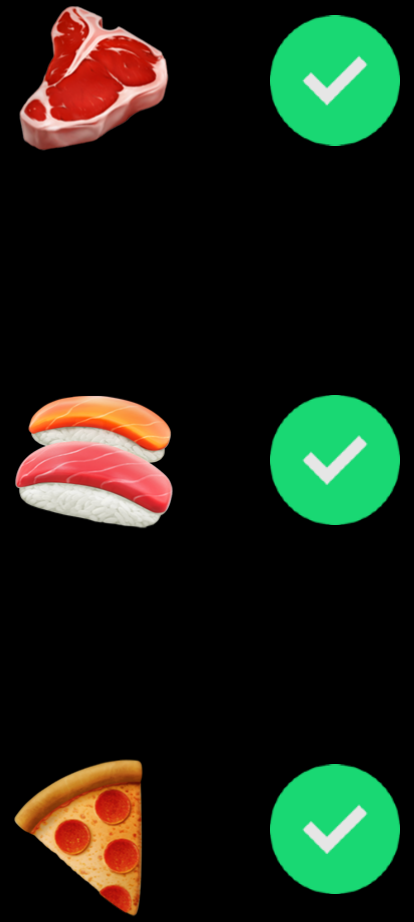
# What we're going to build

## FoodVision Mini

Load data



Build/train a model



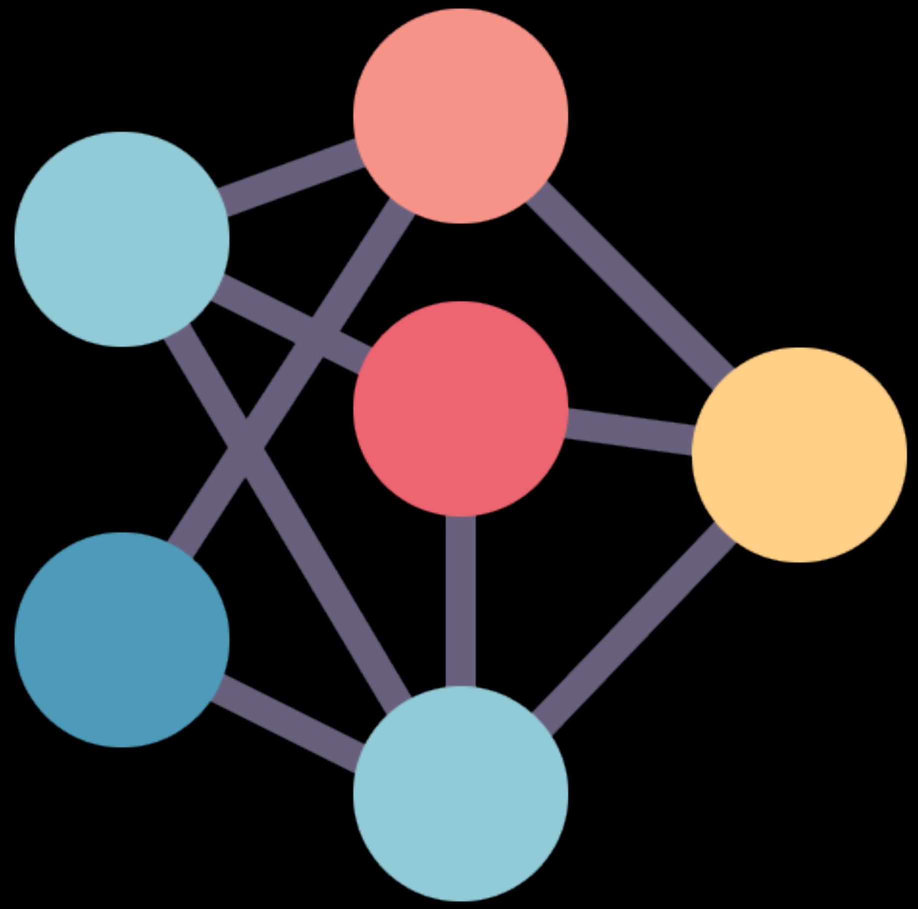
Predict with the model

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

Load data



Build/train a model

```
going_modular/
├── going_modular/
│   ├── data_setup.py
│   ├── engine.py
│   ├── model_builder.py
│   ├── train.py
│   └── utils.py
├── models/
│   ├── 05_going_modular_cell_mode_tinyvgg_model.pth
│   └── 05_going_modular_script_mode_tinyvgg_model.pth
└── data/
    └── pizza_steak_sushi/
        ├── train/
        │   ├── pizza/
        │   │   ├── image01.jpeg
        │   │   └── ...
        │   ├── steak/
        │   └── sushi/
        └── test/
            ├── pizza/
            ├── steak/
            └── sushi/
```

Prepares data

Functions to train/test

Builds a PyTorch model

Trains a PyTorch model

Utility functions

Trained models

Data in standard image classification format

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

How big should the batch size be?

Train for how long?

```
python train.py —model MODEL_NAME —batch_size BATCH_SIZE —lr LEARNING_RATE —num_epochs NUM_EPOCHS
```

Model to train

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:** Detectron2 documentation.

main ▾ vision / references / detection / Go to file Add file ▾ ···

datumbox Fix regression on Detection training script (#5985) ✕ 3ec4b94 5 days ago History

| | | |
|---|---|---|
| README.md | Port Multi-weight support from prototype to main (#5618) | 2 months ago |
| coco_eval.py | Replace asserts with exceptions (#5587) | 2 months ago |
| coco_utils.py | Replace asserts with exceptions (#5587) | 2 months ago |
| engine.py | support amp training for detection models (#4933) | 6 months ago |
| group_by_aspect_ratio.py | Use f-strings almost everywhere, and other cleanups by applying pyupg... | 7 months ago |
| presets.py | Detection recipe enhancements (#5715) | 2 months ago |
| train.py | Fix regression on Detection training script (#5985) | 5 days ago |
| transforms.py | Adding RandomShortestSize transform (#5610) | 2 months ago |
| utils.py | Use f-strings almost everywhere, and other cleanups by applying pyupg... | 7 months ago |

**Source:** `torchvision` object detection GitHub.

### 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.

Using our standard training reference script, we can train a ResNet50 using the following command:

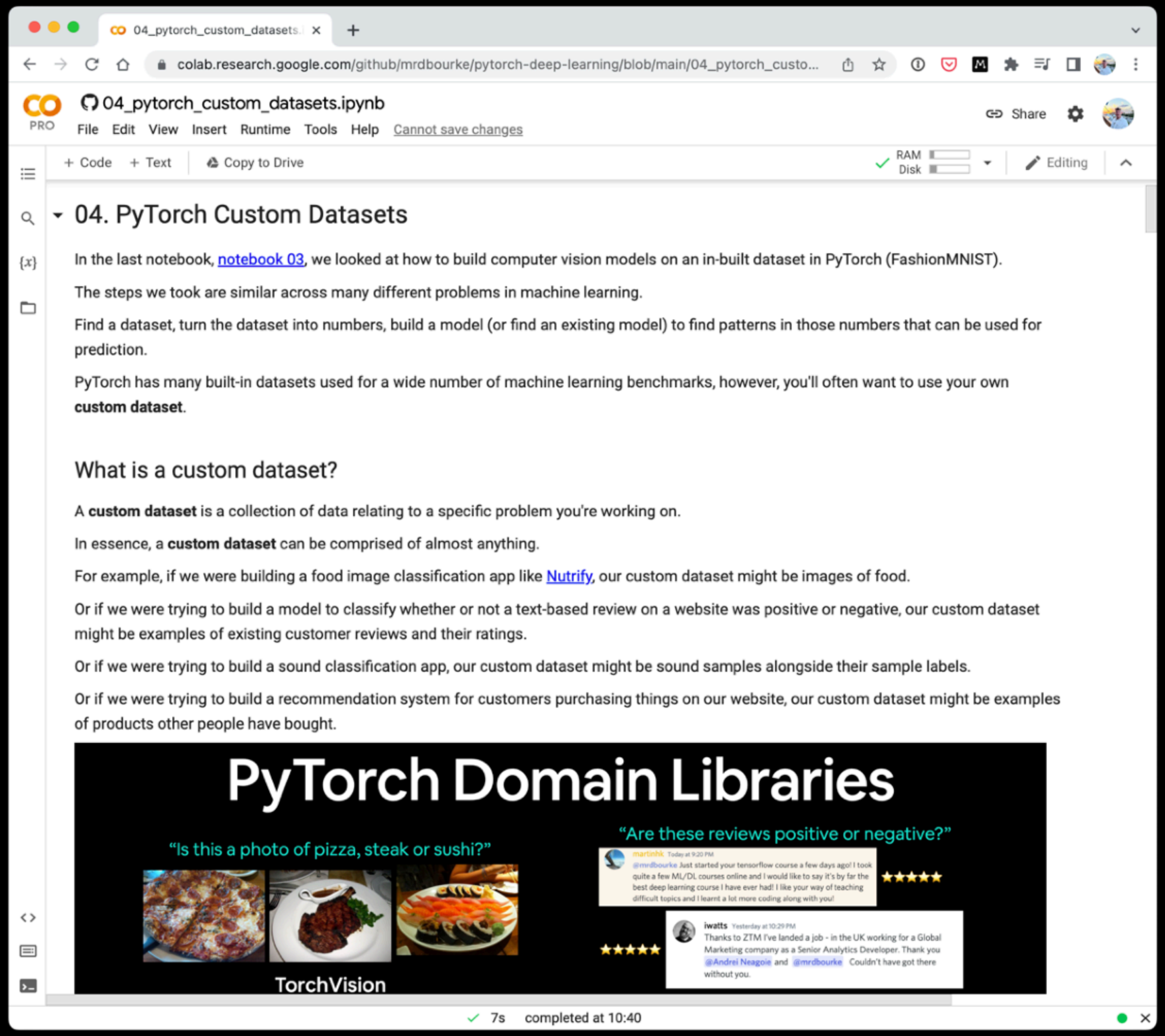
```
torchrun --nproc_per_node=8 train.py --model resnet50 --batch-size 128 --lr 0.5 \
--lr-scheduler cosineannealinglr --lr-warmup-epochs 5 --lr-warmup-method linear \
--auto-augment ta_wide --epochs 600 --random-erase 0.1 --weight-decay 0.00002 \
--norm-weight-decay 0.0 --label-smoothing 0.1 --mixup-alpha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

**Source:** Training state-of-the-art computer vision models with `torchvision` from the PyTorch blog.

# My workflow

*(one of many options)*

*(experiment, experiment, experiment!)*



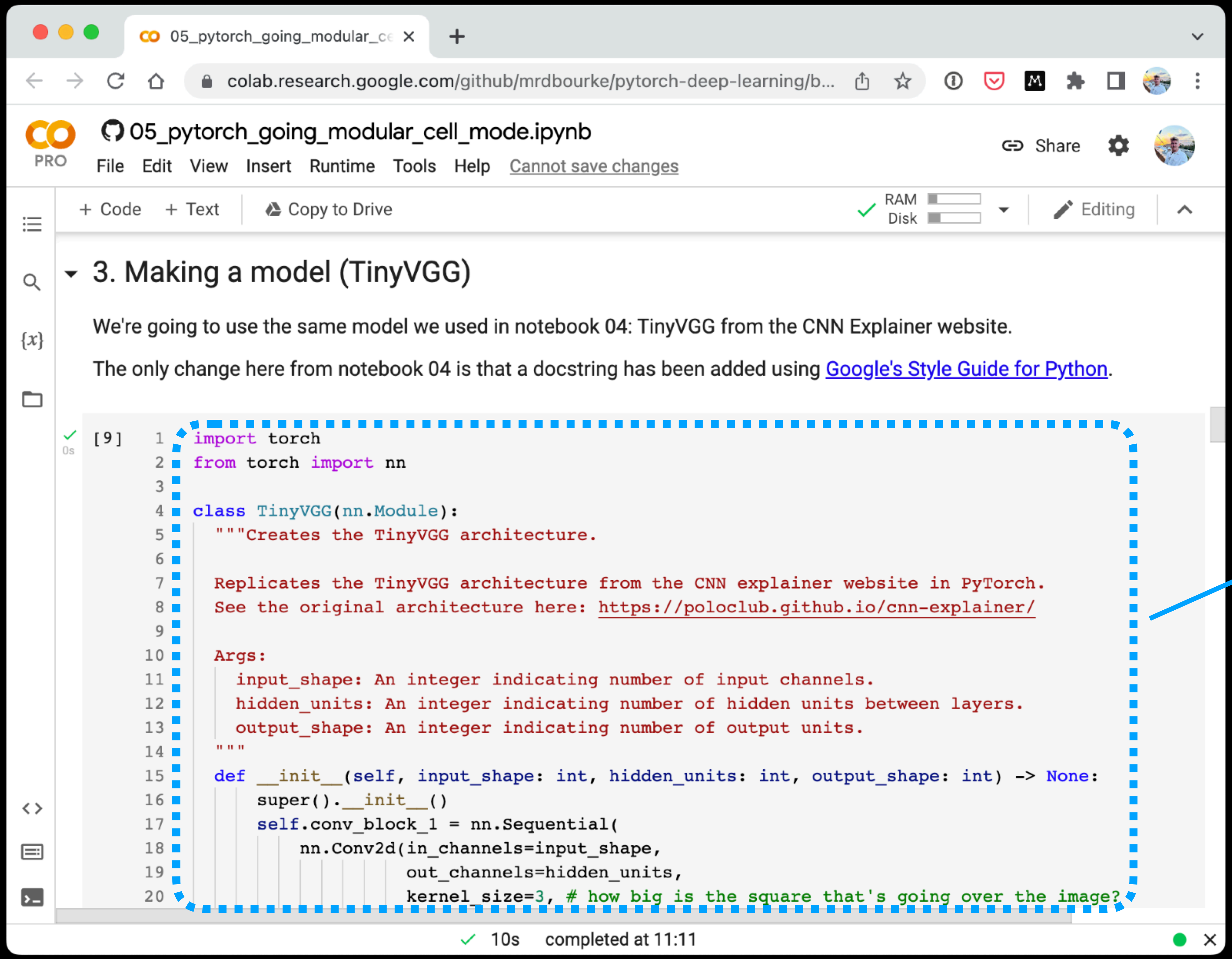

Start with Jupyter/Google Colab notebooks

data_setup.py

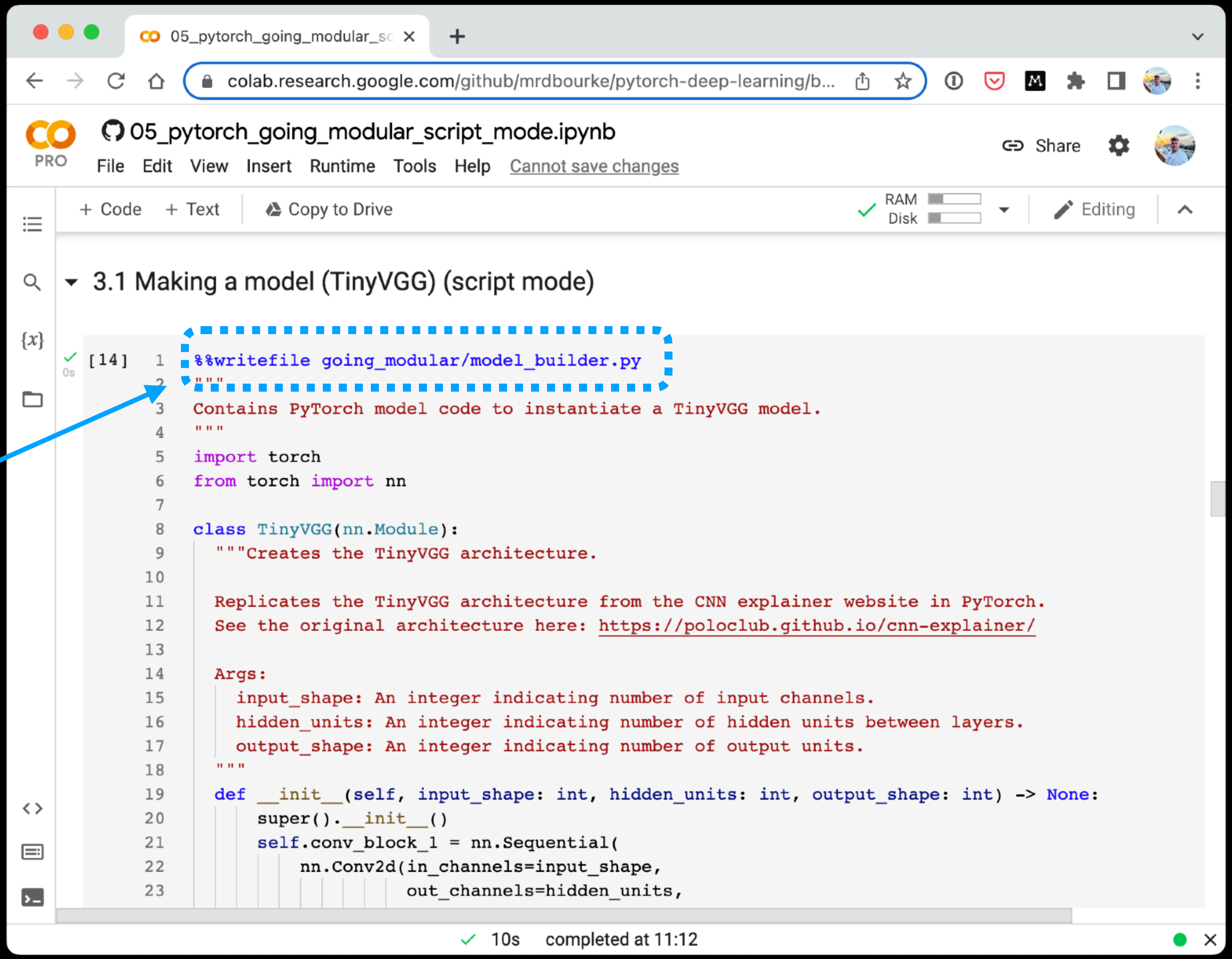
```python
import os

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

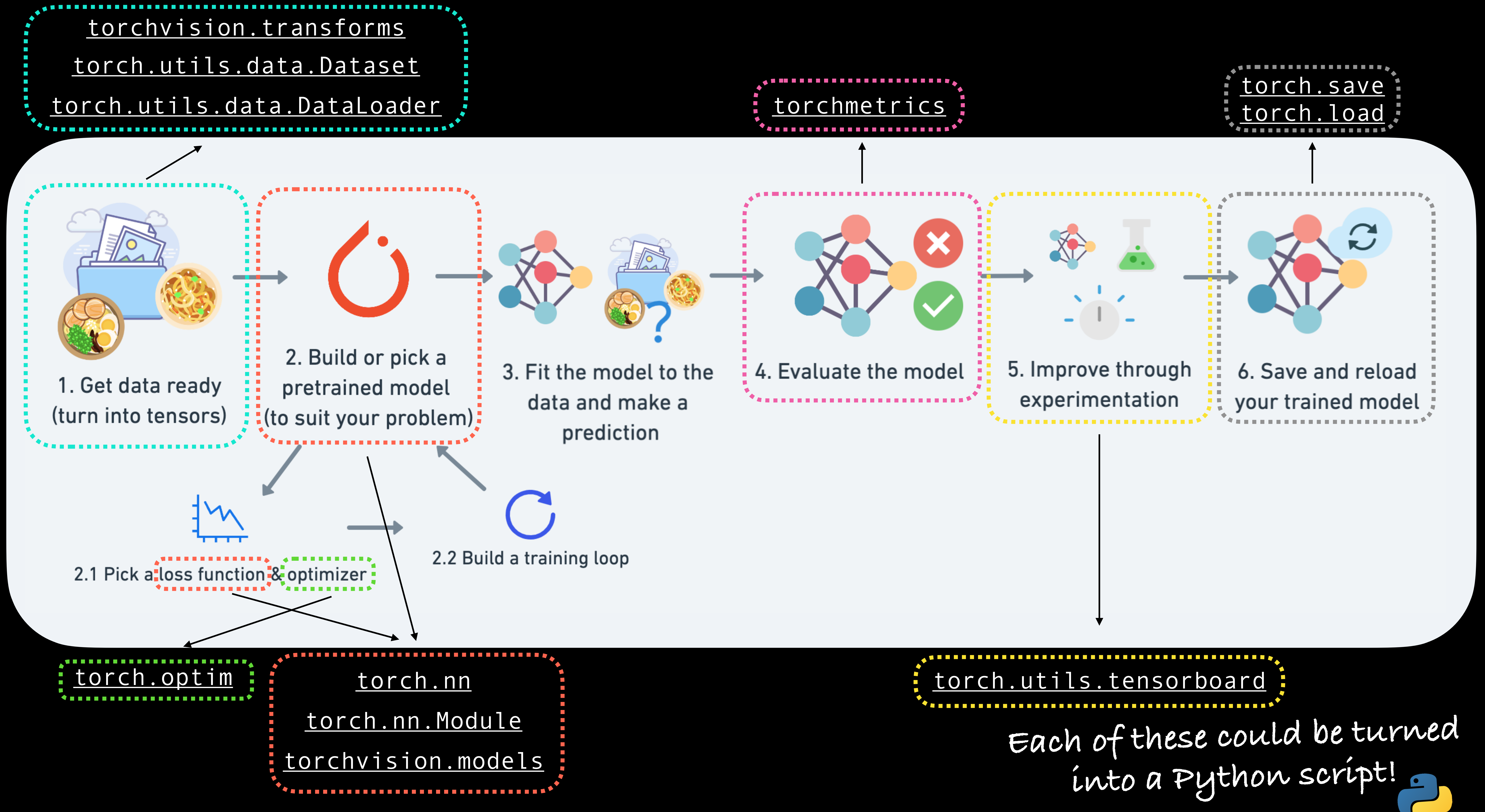



Notebook 05 Part 1: Cell mode




Notebook 05 Part 2: Script mode

(turns useful code into Python scripts)




**See more:** https://pytorch.org/tutorials/beginner/ptcheat.html

# 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