# 07 PyTorch Experiment Tracking

Machine learning is very experimental.

In order to figure out which experiments are worth pursuing, that's where **experiment tracking** comes in, it helps you to figure out what doesn't work so you can figure out what **does** work.

In this notebook, we're going to see an example of programmatically tracking experiments.

Resources:

- Book version of notebook: https://www.learnpytorch.io/07_pytorch_experiment_tracking/
- Ask a question: https://github.com/mrdbourke/pytorch-deep-learning/discussions
- Extra-curriculum: https://madewithml.com/courses/mlops/experiment-tracking/

```
import torch
import torchvision

print(torch.__version__)
print(torchvision.__version__)
```

```
1.13.0.dev20220627+cu113
0.14.0.dev20220627+cu113
```

```
# For this notebook to run with updated APIs, we need torch 1.12+ and
torchvision 0.13+
try:
    import torch
    import torchvision
    assert int(torch.__version__.split(".")[1]) >= 12, "torch version
should be 1.12+"
    assert int(torchvision.__version__.split(".")[1]) >= 13,
"torchvision version should be 0.13+"
    print(f"torch version: {torch.__version__}")
    print(f"torchvision version: {torchvision.__version__}")
except:
    print(f"[INFO] torch/torchvision versions not as required,
installing nightly versions.")
    !pip3 install -U --pre torch torchvision torchaudio --extra-index-
url https://download.pytorch.org/whl/nightly/cu113
    import torch
    import torchvision
    print(f"torch version: {torch.__version__}")
    print(f"torchvision version: {torchvision.__version__}")
```

```
torch version: 1.13.0.dev20220627+cu113
torchvision version: 0.14.0.dev20220627+cu113
```

```
# Continue with regular imports
import matplotlib.pyplot as plt
import torch
import torchvision

from torch import nn
from torchvision import transforms

# Try to get torchinfo, install it if it doesn't work
try:
    from torchinfo import summary
except:
    print("[INFO] Couldn't find torchinfo... installing it.")
    !pip install -q torchinfo
    from torchinfo import summary

# Try to import the going_modular directory, download it from GitHub
if it doesn't work
try:
    from going_modular.going_modular import data_setup, engine
except:
    # Get the going_modular scripts
    print("[INFO] Couldn't find going_modular scripts... downloading
them from GitHub.")
    !git clone https://github.com/mrdbourke/pytorch-deep-learning
    !mv pytorch-deep-learning/going_modular .
    !rm -rf pytorch-deep-learning
    from going_modular.going_modular import data_setup, engine

# Setup device agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

```
{"type":"string"}
```

```
# Set seeds
def set_seeds(seed: int=42):
    """Sets random sets for torch operations.

    Args:
        seed (int, optional): Random seed to set. Defaults to 42.
    """
    # Set the seed for general torch operations
    torch.manual_seed(seed)
    # Set the seed for CUDA torch operations (ones that happen on the
GPU)
    torch.cuda.manual_seed(seed)

set_seeds()
```

# 1. Get data

Want to get pizza, steak, sushi images.

So we can run experiments building FoodVision Mini and see which model performs best.

```
import os
import zipfile

from pathlib import Path

import requests

# example source:
https://github.com/mrdbourke/pytorch-deep-learning/raw/main/data/pizza
_steak_sushi.zip

def download_data(source: str,
                  destination: str,
                  remove_source: bool = True) -> Path:
  """Downloads a zipped dataset from source and unzips to
destination."""
  # Setup path to data folder
  data_path = Path("data/")
  image_path = data_path /destination

  # If the image folder doesn't exist, create it
  if image_path.is_dir():
    print(f"[INFO] {image_path} directory already exists, skipping
download.")
  else:
    print(f"[INFO] Did not find {image_path} directory, creating
one...")
    image_path.mkdir(parents=True, exist_ok=True)

    # Download the target data
    target_file = Path(source).name
    with open(data_path / target_file, "wb") as f:
      request = requests.get(source)
      print(f"[INFO] Downloading {target_file} from {source}...")
      f.write(request.content)

    # Unzip target file
    with zipfile.ZipFile(data_path / target_file, "r") as zip_ref:
      print(f"[INFO] Unzipping {target_file} data...")
      zip_ref.extractall(image_path)

    # Remove .zip file if needed
    if remove_source:
      os.remove(data_path / target_file)
```

```
  return image_path

image_path =
download_data(source="https://github.com/mrdbourke/pytorch-deep-
learning/raw/main/data/pizza_steak_sushi.zip",
                           destination="pizza_steak_sushi")
image_path
```

```
[INFO] data/pizza_steak_sushi directory already exists, skipping
download.
```

```
PosixPath('data/pizza_steak_sushi')
```

# 2. Create Datasets and DataLoaders

## 2.1 Create DataLoaders with manual transforms

The goal with transforms is to ensure your custom data is formatted in a reproducible way as well as a way that will suit pretrained models.

```
# Setup directories
train_dir = image_path / "train"
test_dir = image_path /"test"

train_dir, test_dir
```

```
(PosixPath('data/pizza_steak_sushi/train'),
 PosixPath('data/pizza_steak_sushi/test'))
```

```
# Setup ImageNet normalization levels
# See here: https://pytorch.org/vision/0.12/models.html
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

# Create transform pipeline manually
from torchvision import transforms
manual_transforms = transforms.Compose([
                                        transforms.Resize((224, 224)),
                                        transforms.ToTensor(),
                                        normalize
])
print(f"Manually created transforms: {manual_transforms}")

# Create DataLoaders
from going_modular.going_modular import data_setup
train_dataloader, test_dataloader, class_names =
data_setup.create_dataloaders(train_dir=train_dir,

test_dir=test_dir,
```

```
transform=manual_transforms,

batch_size=32)
train_dataloader, test_dataloader, class_names

Manually created transforms: Compose(
    Resize(size=(224, 224), interpolation=bilinear, max_size=None,
antialias=None)
    ToTensor()
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
)

(<torch.utils.data.dataloader.DataLoader at 0x7fe2dadb1590>,
 <torch.utils.data.dataloader.DataLoader at 0x7fe2dadb16d0>,
 ['pizza', 'steak', 'sushi'])
```

## 2.2 Create DataLoaders using automatically created transforms

The same principle applies for automatic transforms: we want our custom data in the same format as a pretrained model was trained on.

```
# Setup dirs
train_dir = image_path / "train"
test_dir = image_path / "test"

# Setup pretrained weights (plenty of these weights available in
torchvision.models v0.13+)
import torchvision
weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT #
"DEFAULT" = best available

# Get transforms from weights (these are the transforms used to train
a particular or obtain a particular set of weights)
automatic_transforms = weights.transforms()
print(f"Automatically created transforms: {automatic_transforms}")

# Create DataLoaders
train_dataloader, test_dataloader, class_names =
data_setup.create_dataloaders(train_dir=train_dir,

test_dir=test_dir,

transform=automatic_transforms,

batch_size=32)
train_dataloader, test_dataloader, class_names

Automatically created transforms: ImageClassification(
    crop_size=[224]
```

```
    resize_size=[256]
    mean=[0.485, 0.456, 0.406]
    std=[0.229, 0.224, 0.225]
    interpolation=InterpolationMode.BICUBIC
)
```

```
(<torch.utils.data.dataloader.DataLoader at 0x7fe2dae4ae90>,
 <torch.utils.data.dataloader.DataLoader at 0x7fe2dae07890>,
 ['pizza', 'steak', 'sushi'])
```

## 3. Getting a pretrained model, freeze the base layers and change the classifier head

```
# Note: This is how a pretrained model would be created prior to
torchvision v0.13
# model =
torchvision.models.efficientnet_b0(pretrained=True).to(device) # OLD

# Download the pretrained weights for EfficientNet_B0
weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT #
"DEFAULT" = best available weights

# Setup the model with the pretrained weights and send it to the
target device
model = torchvision.models.efficientnet_b0(weights=weights).to(device)
# model
```

```
Downloading:
"https://download.pytorch.org/models/efficientnet_b0_rwightman-
3dd342df.pth" to
/root/.cache/torch/hub/checkpoints/efficientnet_b0_rwightman-
3dd342df.pth
```


```
{"model_id":"98cad9ec7c6b48dc91f3dfa151019292","version_major":2,"vers
ion_minor":0}
```


```
# Freeze all base layers by setting their requires_grad attribute to
False
for param in model.features.parameters():
  # print(param)
  param.requires_grad = False

# Adjust the classifier head
set_seeds()
model.classifier = nn.Sequential(
    nn.Dropout(p=0.2, inplace=True),
    nn.Linear(in_features=1280,
out_features=len(class_names))).to(device)
```

```
from torchinfo import summary

summary(model,
        input_size=(32, 3, 224, 224),
        verbose=0,
        col_names=["input_size", "output_size", "num_params",
"trainable"],
        col_width=20,
        row_settings=["var_names"])
```

```
==========================================================================
==========================================================================
Layer (type (var_name))                                      Input
Shape          Output Shape         Param #              Trainable
==========================================================================
==========================================================================
EfficientNet (EfficientNet)                                  [32, 3,
224, 224]    [32, 3]              --                   Partial
├─Sequential (features)                                      [32, 3,
224, 224]    [32, 1280, 7, 7]     --                   False
│    └─Conv2dNormActivation (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
|    └─Conv2dNormActivation (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
==========================================================================
==========================================================================
```

# 4. Train a single model and track results

```
# Define loss function optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

To track experiments, we're going to use TensorBoard:
https://www.tensorflow.org/tensorboard/

And to interact with TensorBoard, we can use PyTorch's SummaryWriter -
https://pytorch.org/docs/stable/tensorboard.html

- Also see here:
  https://pytorch.org/docs/stable/tensorboard.html#torch.utils.tensorboard.writer.SummaryWriter

```
# Setup a SummaryWriter
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
writer
```

```
<torch.utils.tensorboard.writer.SummaryWriter at 0x7fe2d94bb5d0>
```

```
from tqdm.auto import tqdm
from typing import Dict, List, Tuple

from going_modular.going_modular.engine import train_step, test_step

def train(model: torch.nn.Module,
          train_dataloader: torch.utils.data.DataLoader,
          test_dataloader: torch.utils.data.DataLoader,
          optimizer: torch.optim.Optimizer,
          loss_fn: torch.nn.Module,
          epochs: int,
          device: torch.device) -> Dict[str, List]:
    """Trains and tests a PyTorch model.

    Passes a target PyTorch models through train_step() and
test_step()
    functions for a number of epochs, training and testing the model
    in the same epoch loop.

    Calculates, prints and stores evaluation metrics throughout.

    Args:
```

```
    model: A PyTorch model to be trained and tested.
    train_dataloader: A DataLoader instance for the model to be
trained on.
    test_dataloader: A DataLoader instance for the model to be tested
on.
    optimizer: A PyTorch optimizer to help minimize the loss function.
    loss_fn: A PyTorch loss function to calculate loss on both
datasets.
    epochs: An integer indicating how many epochs to train for.
    device: A target device to compute on (e.g. "cuda" or "cpu").

    Returns:
    A dictionary of training and testing loss as well as training and
    testing accuracy metrics. Each metric has a value in a list for
    each epoch.
    In the form: {train_loss: [...],
              train_acc: [...],
              test_loss: [...],
              test_acc: [...]}
    For example if training for epochs=2:
             {train_loss: [2.0616, 1.0537],
              train_acc: [0.3945, 0.3945],
              test_loss: [1.2641, 1.5706],
              test_acc: [0.3400, 0.2973]}
    """
    # Create empty results dictionary
    results = {"train_loss": [],
               "train_acc": [],
               "test_loss": [],
               "test_acc": []
    }

    # Loop through training and testing steps for a number of epochs
    for epoch in tqdm(range(epochs)):
        train_loss, train_acc = train_step(model=model,
                                           dataloader=train_dataloader,
                                           loss_fn=loss_fn,
                                           optimizer=optimizer,
                                           device=device)
        test_loss, test_acc = test_step(model=model,
          dataloader=test_dataloader,
          loss_fn=loss_fn,
          device=device)

        # Print out what's happening
        print(
          f"Epoch: {epoch+1} | "
          f"train_loss: {train_loss:.4f} | "
          f"train_acc: {train_acc:.4f} | "
          f"test_loss: {test_loss:.4f} | "
```

```
          f"test_acc: {test_acc:.4f}"
        )

        # Update results dictionary
        results["train_loss"].append(train_loss)
        results["train_acc"].append(train_acc)
        results["test_loss"].append(test_loss)
        results["test_acc"].append(test_acc)

        ### New: Experiment tracking ###
        # See SummaryWriter documentation
        writer.add_scalars(main_tag="Loss",
                           tag_scalar_dict={"train_loss": train_loss,
                                            "test_loss": test_loss},
                           global_step=epoch)

        writer.add_scalars(main_tag="Accuracy",
                           tag_scalar_dict={"train_acc": train_acc,
                                            "test_acc": test_acc},
                           global_step=epoch)

        writer.add_graph(model=model,
                         input_to_model=torch.randn(32, 3, 224,
224).to(device))

    # Close the writer
    writer.close()
    ### End new ###

    # Return the filled results at the end of the epochs
    return results

# Train model
# Note: not using engine.train(), since we updated the train()
function above
set_seeds()
results = train(model=model,
                train_dataloader=train_dataloader,
                test_dataloader=test_dataloader,
                optimizer=optimizer,
                loss_fn=loss_fn,
                epochs=5,
                device=device)
```

{"model_id":"3799609be39c4acab9659d16719049d3","version_major":2,"vers
ion_minor":0}

```
Epoch: 1 | train_loss: 1.0929 | train_acc: 0.4023 | test_loss: 0.9125
| test_acc: 0.5502
Epoch: 2 | train_loss: 0.8966 | train_acc: 0.6562 | test_loss: 0.7839
| test_acc: 0.8561
```

```
Epoch: 3 | train_loss: 0.8045 | train_acc: 0.7422 | test_loss: 0.6716
| test_acc: 0.8864
Epoch: 4 | train_loss: 0.6787 | train_acc: 0.7305 | test_loss: 0.6697
| test_acc: 0.8258
Epoch: 5 | train_loss: 0.7066 | train_acc: 0.7188 | test_loss: 0.6737
| test_acc: 0.7737

results

{'test_acc': [0.5501893939393939,
  0.8560606060606061,
  0.8863636363636364,
  0.8257575757575758,
  0.7736742424242425],
 'test_loss': [0.9124558766682943,
  0.7839208642641703,
  0.671596626440684,
  0.6696672836939493,
  0.6737416386604309],
 'train_acc': [0.40234375, 0.65625, 0.7421875, 0.73046875, 0.71875],
 'train_loss': [1.0929433777928352,
  0.8965702056884766,
  0.8045311793684959,
  0.6786761954426765,
  0.706612229347229]}
```

## 5. View our model's results with TensorBoard

There are a few ways to view TensorBoard results, see them here: https://www.learnpytorch.io/07_pytorch_experiment_tracking/#5-view-our-models-results-in-tensorboard

```
# Let's view our experiments from within the notebook
%load_ext tensorboard
%tensorboard --logdir runs

<IPython.core.display.Javascript object>
```

## 6. Create a function to prepare a `SummaryWriter()` instance

By default our `SummaryWriter()` class saves to `log_dir`.

How about if we wanted to save different experiments to different folders?

In essence, one experiment = one folder.

For example, we'd like to track:

- Experiment date/timestamp
- Experiment name

- Model name
- Extra - is there anything else that should be tracked?

Let's create a function to create a `SummaryWriter()` instance to take all of these things into account.

So ideally we end up tracking experiments to a directory:

`runs/YYYY-MM-DD/experiment_name/model_name/extra`

```
from torch.utils.tensorboard import SummaryWriter
def create_writer(experiment_name: str,
                  model_name: str,
                  extra: str = None):
  """Creates a torch.utils.tensorboard.writer.SummaryWriter() instance
tracking to a specific directory."""
  from datetime import datetime
  import os

  # Get timestamp of current date in reverse order
  timestamp = datetime.now().strftime("%Y-%m-%d")

  if extra:
    # Create log directory path
    log_dir = os.path.join("runs", timestamp, experiment_name,
model_name, extra)
  else:
    log_dir = os.path.join("runs", timestamp, experiment_name,
model_name)
  print(f"[INFO] Created SummaryWriter saving to {log_dir}")
  return SummaryWriter(log_dir=log_dir)

example_writer = create_writer(experiment_name="data_10_percent",
                               model_name="effnetb0",
                               extra="5_epochs")
example_writer
```

```
[INFO] Created SummaryWriter saving to
runs/2022-06-28/data_10_percent/effnetb0/5_epochs

<torch.utils.tensorboard.writer.SummaryWriter at 0x7fe1d40ff0d0>
```

## 6.1 Update the `train()` function to include a `writer` parameter

```
from tqdm.auto import tqdm
from typing import Dict, List, Tuple

from going_modular.going_modular.engine import train_step, test_step

def train(model: torch.nn.Module,
          train_dataloader: torch.utils.data.DataLoader,
```

```
          test_dataloader: torch.utils.data.DataLoader,
          optimizer: torch.optim.Optimizer,
          loss_fn: torch.nn.Module,
          epochs: int,
          device: torch.device,
          writer: torch.utils.tensorboard.writer.SummaryWriter) ->
Dict[str, List]:
    """Trains and tests a PyTorch model.

    Passes a target PyTorch models through train_step() and
test_step()
    functions for a number of epochs, training and testing the model
    in the same epoch loop.

    Calculates, prints and stores evaluation metrics throughout.

    Args:
    model: A PyTorch model to be trained and tested.
    train_dataloader: A DataLoader instance for the model to be
trained on.
    test_dataloader: A DataLoader instance for the model to be tested
on.
    optimizer: A PyTorch optimizer to help minimize the loss function.
    loss_fn: A PyTorch loss function to calculate loss on both
datasets.
    epochs: An integer indicating how many epochs to train for.
    device: A target device to compute on (e.g. "cuda" or "cpu").

    Returns:
    A dictionary of training and testing loss as well as training and
    testing accuracy metrics. Each metric has a value in a list for
    each epoch.
    In the form: {train_loss: [...],
              train_acc: [...],
              test_loss: [...],
              test_acc: [...]}
    For example if training for epochs=2:
             {train_loss: [2.0616, 1.0537],
              train_acc: [0.3945, 0.3945],
              test_loss: [1.2641, 1.5706],
              test_acc: [0.3400, 0.2973]}
    """
    # Create empty results dictionary
    results = {"train_loss": [],
               "train_acc": [],
               "test_loss": [],
               "test_acc": []
    }

    # Loop through training and testing steps for a number of epochs
```

```
    for epoch in tqdm(range(epochs)):
        train_loss, train_acc = train_step(model=model,
                                           dataloader=train_dataloader,
                                           loss_fn=loss_fn,
                                           optimizer=optimizer,
                                           device=device)
        test_loss, test_acc = test_step(model=model,
          dataloader=test_dataloader,
          loss_fn=loss_fn,
          device=device)

        # Print out what's happening
        print(
          f"Epoch: {epoch+1} | "
          f"train_loss: {train_loss:.4f} | "
          f"train_acc: {train_acc:.4f} | "
          f"test_loss: {test_loss:.4f} | "
          f"test_acc: {test_acc:.4f}"
        )

        # Update results dictionary
        results["train_loss"].append(train_loss)
        results["train_acc"].append(train_acc)
        results["test_loss"].append(test_loss)
        results["test_acc"].append(test_acc)

        ### New: Experiment tracking ###
        if writer:
          # See SummaryWriter documentation
          writer.add_scalars(main_tag="Loss",
                             tag_scalar_dict={"train_loss":
train_loss,
                                              "test_loss": test_loss},
                             global_step=epoch)

          writer.add_scalars(main_tag="Accuracy",
                             tag_scalar_dict={"train_acc": train_acc,
                                              "test_acc": test_acc},
                             global_step=epoch)

          writer.add_graph(model=model,
                           input_to_model=torch.randn(32, 3, 224,
224).to(device))

          # Close the writer
          writer.close()

        else:
          pass
    ### End new ###
```

```
    # Return the filled results at the end of the epochs
    return results
```

# 7. Setting up a series of modelling experiments

- Challenge: Setup 2x modelling experiments with effnetb0, pizza, steak, sushi data and train one model for 5 epochs and another model for 10 epochs

## 7.1 What kind of experiments should you run?

The number of machine learning experiments you can run, is like the number of different models you can build... almost limitless.

However, you can't test everything...

So what should you test?

- Change the number of epochs
- Change the number of hidden layers/units
- Change the amount of data (right now we're using 10% of the Food101 dataset for pizza, steak, sushi)
- Change the learning rate
- Try different kinds of data augmentation
- Choose a different model architecture

This is why transfer learning is so powerful, because, it's a working model that you can apply to your own problem.

## 7.2 What experiments are we going to run?

We're going to turn three dials:

1. Model size - EffnetB0 vs EffnetB2 (in terms of number of parameters)
2. Dataset size - 10% of pizza, steak, sushi images vs 20% (generally more data = better results)
3. Training time - 5 epochs vs 10 epochs (generally longer training time = better results, up to a point)

To begin, we're still keeping things relatively small so that our experiments run quickly.

**Our goal:** a model that is well performing but still small enough to run on a mobile device or web browser, so FoodVision Mini can come to life.

If you had infinite compute + time, you should basically always choose the biggest model and biggest dataset you can. See: http://www.incompleteideas.net/IncIdeas/BitterLesson.html

## 7.3 Download different datasets

We want two datasets:

1. Pizza, steak, sushi 10% - https://github.com/mrdbourke/pytorch-deep-learning/raw/main/data/pizza_steak_sushi.zip
2. Pizza, steak, sushi 20% - https://github.com/mrdbourke/pytorch-deep-learning/raw/main/data/pizza_steak_sushi_20_percent.zip

They were created with: https://github.com/mrdbourke/pytorch-deep-learning/blob/main/extras/04_custom_data_creation.ipynb

```
# Download 10 percent and 20 percent datasets
data_10_percent_path =
download_data(source="https://github.com/mrdbourke/pytorch-deep-
learning/raw/main/data/pizza_steak_sushi.zip",
                                    destination="pizza_steak_sushi")

data_20_percent_path =
download_data(source="https://github.com/mrdbourke/pytorch-deep-
learning/raw/main/data/pizza_steak_sushi_20_percent.zip",

destination="pizza_steak_sushi_20_percent")
```

```
[INFO] data/pizza_steak_sushi directory already exists, skipping
download.
[INFO] Did not find data/pizza_steak_sushi_20_percent directory,
creating one...
[INFO] Downloading pizza_steak_sushi_20_percent.zip from
https://github.com/mrdbourke/pytorch-deep-learning/raw/main/data/pizza
_steak_sushi_20_percent.zip...
[INFO] Unzipping pizza_steak_sushi_20_percent.zip data...
```

## 7.4 Transform Datasets and Create DataLoaders

We'll to transform our data in a few ways:

1. Resize the images to (224, 224)
2. Make sure image tensor values are between [0, 1]
3. Normalize the images so they have the same data distribution as ImageNet

```
# Setup training directory paths
train_dir_10_percent = data_10_percent_path / "train"
train_dir_20_percent = data_20_percent_path / "train"

# Setup the test directory
test_dir = data_10_percent_path / "test"

train_dir_10_percent, train_dir_20_percent, test_dir
```

```
(PosixPath('data/pizza_steak_sushi/train'),
 PosixPath('data/pizza_steak_sushi_20_percent/train'),
 PosixPath('data/pizza_steak_sushi/test'))

from torchvision import transforms

# Setup ImageNet normalization levels
# See here: https://pytorch.org/vision/0.12/models.html
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

# Compose transforms into a pipleine
simple_transform = transforms.Compose([
                                       transforms.Resize((224, 224)),
                                       transforms.ToTensor(),
                                       normalize
])

BATCH_SIZE = 32

# Create 10% training and test DataLoaders
train_dataloader_10_percent, test_dataloader, class_names =
data_setup.create_dataloaders(train_dir=train_dir_10_percent,

test_dir=test_dir,

transform=simple_transform,

batch_size=BATCH_SIZE)

# Create 20% training and test DataLoaders
train_dataloader_20_percent, test_dataloader, class_names =
data_setup.create_dataloaders(train_dir=train_dir_20_percent,

test_dir=test_dir,

transform=simple_transform,

batch_size=BATCH_SIZE)

print(f"Number of batches of size {BATCH_SIZE} in 10% train data:
{len(train_dataloader_10_percent)}")
print(f"Number of batches of size {BATCH_SIZE} in 20% train data:
{len(train_dataloader_20_percent)}")
print(f"Number of batches of size {BATCH_SIZE} in 10% test data:
{len(test_dataloader)}")
print(f"Class names: {class_names}")

Number of batches of size 32 in 10% train data: 8
Number of batches of size 32 in 20% train data: 15
```

```
Number of batches of size 32 in 10% test data: 3
Class names: ['pizza', 'steak', 'sushi']
```

## 7.5 Create feature extractor models

We want two functions:

1. Creates a `torchvision.models.efficientnet_b0()` feature extractor with a frozen backbone/base layers and a custom classifier head (EffNetB0).
2. Creates a `torchvision.models.efficientnet_b2()` feature extractor with a frozen backbone/base layers and a custom classifier head (EffNetB2).

```
import torchvision

# Create an EffNetB2
effnetb2_weights = torchvision.models.EfficientNet_B2_Weights.DEFAULT
# "DEFAULT" = best available
effnetb2 =
torchvision.models.efficientnet_b2(weights=effnetb2_weights)

# effnetb2
```

```
Downloading:
"https://download.pytorch.org/models/efficientnet_b2_rwightman-
bcdf34b7.pth" to
/root/.cache/torch/hub/checkpoints/efficientnet_b2_rwightman-
bcdf34b7.pth
```


```
{"model_id":"321952803de84cecaa56a0b250abefa0","version_major":2,"vers
ion_minor":0}
```


```
summary(model=effnetb2,
        input_size=(32, 3, 224, 224),
        verbose=0,
        col_names=["input_size", "output_size", "num_params",
"trainable"],
        col_width=20,
        row_settings=["var_names"])
```

```
==========================================================================
==========================================================================
Layer (type (var_name))                                      Input
Shape          Output Shape         Param #              Trainable
==========================================================================
==========================================================================
EfficientNet (EfficientNet)                                  [32, 3,
224, 224]    [32, 1000]           --                   True
├─Sequential (features)                                      [32, 3,
224, 224]    [32, 1408, 7, 7]     --                   True
│    └─Conv2dNormActivation (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
|    |    └─MBConv (1)                                      [32, 16,
112, 112]   [32, 16, 112, 112]   612                  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
|    |    └─MBConv (2)                                      [32, 24,
56, 56]     [32, 24, 56, 56]     10,710               True
|    └─Sequential (3)                                       [32, 24,
56, 56]     [32, 48, 28, 28]     --                   True
|    |    └─MBConv (0)                                      [32, 24,
56, 56]     [32, 48, 28, 28]     16,518               True
|    |    └─MBConv (1)                                      [32, 48,
28, 28]     [32, 48, 28, 28]     43,308               True
|    |    └─MBConv (2)                                      [32, 48,
28, 28]     [32, 48, 28, 28]     43,308               True
|    └─Sequential (4)                                       [32, 48,
28, 28]     [32, 88, 14, 14]     --                   True
|    |    └─MBConv (0)                                      [32, 48,
28, 28]     [32, 88, 14, 14]     50,300               True
|    |    └─MBConv (1)                                      [32, 88,
14, 14]     [32, 88, 14, 14]     123,750              True
|    |    └─MBConv (2)                                      [32, 88,
14, 14]     [32, 88, 14, 14]     123,750              True
|    |    └─MBConv (3)                                      [32, 88,
14, 14]     [32, 88, 14, 14]     123,750              True
|    └─Sequential (5)                                       [32, 88,
14, 14]     [32, 120, 14, 14]    --                   True
|    |    └─MBConv (0)                                      [32, 88,
14, 14]     [32, 120, 14, 14]    149,158              True
|    |    └─MBConv (1)                                      [32, 120,
14, 14]    [32, 120, 14, 14]    237,870              True
|    |    └─MBConv (2)                                      [32, 120,
14, 14]    [32, 120, 14, 14]    237,870              True
|    |    └─MBConv (3)                                      [32, 120,
14, 14]    [32, 120, 14, 14]    237,870              True
```

```
|    └─Sequential (6)                                          [32, 120,
14, 14]    [32, 208, 7, 7]      --                  True
|    |    └─MBConv (0)                                         [32, 120,
14, 14]    [32, 208, 7, 7]      301,406             True
|    |    └─MBConv (1)                                         [32, 208,
7, 7]      [32, 208, 7, 7]      686,868             True
|    |    └─MBConv (2)                                         [32, 208,
7, 7]      [32, 208, 7, 7]      686,868             True
|    |    └─MBConv (3)                                         [32, 208,
7, 7]      [32, 208, 7, 7]      686,868             True
|    |    └─MBConv (4)                                         [32, 208,
7, 7]      [32, 208, 7, 7]      686,868             True
|    └─Sequential (7)                                          [32, 208,
7, 7]      [32, 352, 7, 7]      --                  True
|    |    └─MBConv (0)                                         [32, 208,
7, 7]      [32, 352, 7, 7]      846,900             True
|    |    └─MBConv (1)                                         [32, 352,
7, 7]      [32, 352, 7, 7]      1,888,920           True
|    └─Conv2dNormActivation (8)                                [32, 352,
7, 7]      [32, 1408, 7, 7]     --                  True
|    |    └─Conv2d (0)                                         [32, 352,
7, 7]      [32, 1408, 7, 7]     495,616             True
|    |    └─BatchNorm2d (1)                                    [32,
1408, 7, 7]     [32, 1408, 7, 7]     2,816               True
|    |    └─SiLU (2)                                           [32,
1408, 7, 7]     [32, 1408, 7, 7]     --                  --
├─AdaptiveAvgPool2d (avgpool)                                  [32,
1408, 7, 7]     [32, 1408, 1, 1]     --                  --
├─Sequential (classifier)                                      [32,
1408]           [32, 1000]           --                  True
|    └─Dropout (0)                                            [32,
1408]           [32, 1408]           --                  --
|    └─Linear (1)                                            [32,
1408]           [32, 1000]           1,409,000           True
==========================================================================
==========================================================================
Total params: 9,109,994
Trainable params: 9,109,994
Non-trainable params: 0
Total mult-adds (G): 21.09
==========================================================================
==========================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 5017.79
Params size (MB): 36.44
Estimated Total Size (MB): 5073.49
==========================================================================
==========================================================================
```

```
import torchvision
from torch import nn

OUT_FEATURES = len(class_names)

# Create an EffNetB0 feature extractor
def create_effnetb0():
  # Get the weights and setup a model
  weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT
  model =
torchvision.models.efficientnet_b0(weights=weights).to(device)

  # Freeze the base model layers
  for param in model.features.parameters():
    param.requires_grad = False

  # Change the classifier head
  set_seeds()
  model.classifier = nn.Sequential(
      nn.Dropout(p=0.2, inplace=True),
      nn.Linear(in_features=1280, out_features=OUT_FEATURES)
  ).to(device)

  # Give the model a name
  model.name = "effnetb0"
  print(f"[INFO] Created new {model.name} model...")
  return model

# Create an EffNetB2 feature extractor
def create_effnetb2():
  # Get the weights and setup a model
  weights = torchvision.models.EfficientNet_B2_Weights.DEFAULT
  model =
torchvision.models.efficientnet_b2(weights=weights).to(device)

  # Freeze the base model layers
  for param in model.features.parameters():
    param.requires_grad = False

  # Change the classifier head
  set_seeds()
  model.classifier = nn.Sequential(
      nn.Dropout(p=0.3, inplace=True),
      nn.Linear(in_features=1408, out_features=OUT_FEATURES)
  ).to(device)

  # Give the model a name
  model.name = "effnetb2"
  print(f"[INFO] Created new {model.name} model...")
  return model
```

```
effnetb2.classifier

Sequential(
  (0): Dropout(p=0.3, inplace=True)
  (1): Linear(in_features=1408, out_features=1000, bias=True)
)

created_model_test_effnetb2 = create_effnetb2()
created_model_test_effnetb0 = create_effnetb0()

[INFO] Created new effnetb2 model...
[INFO] Created new effnetb0 model...

# Check out EffNetB2 feature extractor
summary(model=created_model_test_effnetb2,
        input_size=(32, 3, 224, 224),
        verbose=0,
        col_names=["input_size", "output_size", "num_params",
"trainable"],
        col_width=20,
        row_settings=["var_names"])

==========================================================================
==========================================================================
Layer (type (var_name))                                      Input
Shape          Output Shape         Param #              Trainable
==========================================================================
==========================================================================
EfficientNet (EfficientNet)                                  [32, 3,
224, 224]    [32, 3]              --                   Partial
├─Sequential (features)                                      [32, 3,
224, 224]    [32, 1408, 7, 7]     --                   False
│    └─Conv2dNormActivation (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
│    │    └─MBConv (1)                                       [32, 16,
112, 112]   [32, 16, 112, 112]   (612)                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
|    |    └─MBConv (2)                                                  [32, 24,
56, 56]     [32, 24, 56, 56]     (10,710)             False
|    └─Sequential (3)                                                   [32, 24,
56, 56]     [32, 48, 28, 28]     --                   False
|    |    └─MBConv (0)                                                  [32, 24,
56, 56]     [32, 48, 28, 28]     (16,518)             False
|    |    └─MBConv (1)                                                  [32, 48,
28, 28]     [32, 48, 28, 28]     (43,308)             False
|    |    └─MBConv (2)                                                  [32, 48,
28, 28]     [32, 48, 28, 28]     (43,308)             False
|    └─Sequential (4)                                                   [32, 48,
28, 28]     [32, 88, 14, 14]     --                   False
|    |    └─MBConv (0)                                                  [32, 48,
28, 28]     [32, 88, 14, 14]     (50,300)             False
|    |    └─MBConv (1)                                                  [32, 88,
14, 14]     [32, 88, 14, 14]     (123,750)            False
|    |    └─MBConv (2)                                                  [32, 88,
14, 14]     [32, 88, 14, 14]     (123,750)            False
|    |    └─MBConv (3)                                                  [32, 88,
14, 14]     [32, 88, 14, 14]     (123,750)            False
|    └─Sequential (5)                                                   [32, 88,
14, 14]     [32, 120, 14, 14]    --                   False
|    |    └─MBConv (0)                                                  [32, 88,
14, 14]     [32, 120, 14, 14]    (149,158)            False
|    |    └─MBConv (1)                                                  [32, 120,
14, 14]    [32, 120, 14, 14]    (237,870)            False
|    |    └─MBConv (2)                                                  [32, 120,
14, 14]    [32, 120, 14, 14]    (237,870)            False
|    |    └─MBConv (3)                                                  [32, 120,
14, 14]    [32, 120, 14, 14]    (237,870)            False
|    └─Sequential (6)                                                   [32, 120,
14, 14]    [32, 208, 7, 7]      --                   False
|    |    └─MBConv (0)                                                  [32, 120,
14, 14]    [32, 208, 7, 7]      (301,406)            False
|    |    └─MBConv (1)                                                  [32, 208,
7, 7]      [32, 208, 7, 7]      (686,868)            False
|    |    └─MBConv (2)                                                  [32, 208,
7, 7]      [32, 208, 7, 7]      (686,868)            False
|    |    └─MBConv (3)                                                  [32, 208,
7, 7]      [32, 208, 7, 7]      (686,868)            False
|    |    └─MBConv (4)                                                  [32, 208,
7, 7]      [32, 208, 7, 7]      (686,868)            False
|    └─Sequential (7)                                                   [32, 208,
7, 7]      [32, 352, 7, 7]      --                   False
|    |    └─MBConv (0)                                                  [32, 208,
7, 7]      [32, 352, 7, 7]      (846,900)            False
|    |    └─MBConv (1)                                                  [32, 352,
```

```
7, 7]      [32, 352, 7, 7]      (1,888,920)          False
│    └─Conv2dNormActivation (8)                              [32, 352,
7, 7]      [32, 1408, 7, 7]     --                   False
│    │    └─Conv2d (0)                                       [32, 352,
7, 7]      [32, 1408, 7, 7]     (495,616)            False
│    │    └─BatchNorm2d (1)                                  [32,
1408, 7, 7]     [32, 1408, 7, 7]     (2,816)              False
│    │    └─SiLU (2)                                         [32,
1408, 7, 7]     [32, 1408, 7, 7]     --                   --
├─AdaptiveAvgPool2d (avgpool)                               [32,
1408, 7, 7]     [32, 1408, 1, 1]     --                   --
├─Sequential (classifier)                                   [32,
1408]           [32, 3]              --                   True
│    └─Dropout (0)                                          [32,
1408]           [32, 1408]           --                   --
│    └─Linear (1)                                           [32,
1408]           [32, 3]              4,227                True
==========================================================================
==========================================================================
Total params: 7,705,221
Trainable params: 4,227
Non-trainable params: 7,700,994
Total mult-adds (G): 21.04
==========================================================================
==========================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 5017.53
Params size (MB): 30.82
Estimated Total Size (MB): 5067.62
==========================================================================
==========================================================================

# Check out EffNetB0 feature extractor model
summary(model=created_model_test_effnetb0,
        input_size=(32, 3, 224, 224),
        verbose=0,
        col_names=["input_size", "output_size", "num_params",
"trainable"],
        col_width=20,
        row_settings=["var_names"])

==========================================================================
==========================================================================
Layer (type (var_name))                                   Input
Shape          Output Shape         Param #              Trainable
==========================================================================
==========================================================================
EfficientNet (EfficientNet)                               [32, 3,
224, 224]    [32, 3]              --                   Partial
├─Sequential (features)                                   [32, 3,
```

```
224, 224]    [32, 1280, 7, 7]     --                    False
|    └─Conv2dNormActivation (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
│    └─Conv2dNormActivation (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
==========================================================================
==========================================================================
```

## 7.6 Create experiments and set up training code

```python
# Create epoch list
num_epochs = [5, 10]

# Create models list (need to create a new model for each experiment)
models = ["effnetb0", "effnetb2"]

# Create a DataLoaders dictionary
train_dataloaders = {"data_10_percent": train_dataloader_10_percent,
                     "data_20_percent": train_dataloader_20_percent}
```

```
%%time
from going_modular.going_modular.utils import save_model

# Set seeds
set_seeds(seed=42)

# Keep track of experiment numbers
experiment_number = 0

# Loop through each DataLoader
for dataloader_name, train_dataloader in train_dataloaders.items():
  # Loop through the epochs
  for epochs in num_epochs:
    # Loop through each model name and create a new model instance
    for model_name in models:

      # Print out info
      experiment_number += 1
      print(f"[INFO] Experiment number: {experiment_number}")
      print(f"[INFO] Model: {model_name}")
      print(f"[INFO] DataLoader: {dataloader_name}")
      print(f"[INFO] Number of epochs: {epochs}")

      # Select and create the model
      if model_name == "effnetb0":
        model = create_effnetb0()
      else:
        model = create_effnetb2()

      # Create a new loss and optimizer for every model
      loss_fn = nn.CrossEntropyLoss()
      optimizer = torch.optim.Adam(params=model.parameters(),
lr=0.001)

      # Train target model with target dataloader and track
experiments
      # Note: using train() rather than engine.train()
      train(model=model,
            train_dataloader=train_dataloader,
            test_dataloader=test_dataloader,
            optimizer=optimizer,
            loss_fn=loss_fn,
            epochs=epochs,
            device=device,
            writer=create_writer(experiment_name=dataloader_name,
                                 model_name=model_name,
                                 extra=f"{epochs}_epochs"))

      # Save the model to file so we can import it later if need be
      save_filepath =
```

```
f"07_{model_name}_{dataloader_name}_{epochs}_epochs.pth"
      save_model(model=model,
                 target_dir="models",
                 model_name=save_filepath)
      print("-"*50 + "\n")
```

```
[INFO] Experiment number: 1
[INFO] Model: effnetb0
[INFO] DataLoader: data_10_percent
[INFO] Number of epochs: 5
[INFO] Created new effnetb0 model...
[INFO] Created SummaryWriter saving to
runs/2022-06-28/data_10_percent/effnetb0/5_epochs
```


```
{"model_id":"44f62855a2fe49d98c4f219e379ff8b8","version_major":2,"vers
ion_minor":0}
```


```
Epoch: 1 | train_loss: 1.0433 | train_acc: 0.4805 | test_loss: 0.9283
| test_acc: 0.4782
Epoch: 2 | train_loss: 0.9400 | train_acc: 0.5469 | test_loss: 0.8346
| test_acc: 0.5492
Epoch: 3 | train_loss: 0.8243 | train_acc: 0.6953 | test_loss: 0.7293
| test_acc: 0.8248
Epoch: 4 | train_loss: 0.7029 | train_acc: 0.7773 | test_loss: 0.6177
| test_acc: 0.8759
Epoch: 5 | train_loss: 0.6150 | train_acc: 0.8867 | test_loss: 0.5830
| test_acc: 0.8864
[INFO] Saving model to:
models/07_effnetb0_data_10_percent_5_epochs.pth
--------------------------------------------------

[INFO] Experiment number: 2
[INFO] Model: effnetb2
[INFO] DataLoader: data_10_percent
[INFO] Number of epochs: 5
[INFO] Created new effnetb2 model...
[INFO] Created SummaryWriter saving to
runs/2022-06-28/data_10_percent/effnetb2/5_epochs
```


```
{"model_id":"1fa379b53b1b4bdab7864f77d09ec4a1","version_major":2,"vers
ion_minor":0}
```


```
Epoch: 1 | train_loss: 1.0861 | train_acc: 0.4023 | test_loss: 0.9619
| test_acc: 0.6903
Epoch: 2 | train_loss: 0.8960 | train_acc: 0.6250 | test_loss: 0.9031
| test_acc: 0.6818
Epoch: 3 | train_loss: 0.8458 | train_acc: 0.6641 | test_loss: 0.8036
| test_acc: 0.8049
Epoch: 4 | train_loss: 0.7027 | train_acc: 0.8516 | test_loss: 0.6922
| test_acc: 0.9176
```

```
Epoch: 5 | train_loss: 0.7005 | train_acc: 0.7422 | test_loss: 0.6509
| test_acc: 0.8968
[INFO] Saving model to:
models/07_effnetb2_data_10_percent_5_epochs.pth
--------------------------------------------------

[INFO] Experiment number: 3
[INFO] Model: effnetb0
[INFO] DataLoader: data_10_percent
[INFO] Number of epochs: 10
[INFO] Created new effnetb0 model...
[INFO] Created SummaryWriter saving to
runs/2022-06-28/data_10_percent/effnetb0/10_epochs

{"model_id":"6cd298c9251543bb9546efffbb441841","version_major":2,"vers
ion_minor":0}

Epoch: 1 | train_loss: 1.0433 | train_acc: 0.4805 | test_loss: 0.9283
| test_acc: 0.4782
Epoch: 2 | train_loss: 0.9400 | train_acc: 0.5469 | test_loss: 0.8346
| test_acc: 0.5492
Epoch: 3 | train_loss: 0.8243 | train_acc: 0.6953 | test_loss: 0.7293
| test_acc: 0.8248
Epoch: 4 | train_loss: 0.7029 | train_acc: 0.7773 | test_loss: 0.6177
| test_acc: 0.8759
Epoch: 5 | train_loss: 0.6150 | train_acc: 0.8867 | test_loss: 0.5830
| test_acc: 0.8864
Epoch: 6 | train_loss: 0.5431 | train_acc: 0.8750 | test_loss: 0.5943
| test_acc: 0.8561
Epoch: 7 | train_loss: 0.6385 | train_acc: 0.7109 | test_loss: 0.5933
| test_acc: 0.8561
Epoch: 8 | train_loss: 0.5075 | train_acc: 0.8008 | test_loss: 0.5382
| test_acc: 0.8968
Epoch: 9 | train_loss: 0.4688 | train_acc: 0.9180 | test_loss: 0.5318
| test_acc: 0.8759
Epoch: 10 | train_loss: 0.5795 | train_acc: 0.7344 | test_loss: 0.4907
| test_acc: 0.8759
[INFO] Saving model to:
models/07_effnetb0_data_10_percent_10_epochs.pth
--------------------------------------------------

[INFO] Experiment number: 4
[INFO] Model: effnetb2
[INFO] DataLoader: data_10_percent
[INFO] Number of epochs: 10
[INFO] Created new effnetb2 model...
[INFO] Created SummaryWriter saving to
runs/2022-06-28/data_10_percent/effnetb2/10_epochs
```

{"model_id":"c2c40c14fe8b4a579ba9352af0764e63","version_major":2,"version_minor":0}

```
Epoch: 1 | train_loss: 1.0861 | train_acc: 0.4023 | test_loss: 0.9619 | test_acc: 0.6903
Epoch: 2 | train_loss: 0.8960 | train_acc: 0.6250 | test_loss: 0.9031 | test_acc: 0.6818
Epoch: 3 | train_loss: 0.8458 | train_acc: 0.6641 | test_loss: 0.8036 | test_acc: 0.8049
Epoch: 4 | train_loss: 0.7027 | train_acc: 0.8516 | test_loss: 0.6922 | test_acc: 0.9176
Epoch: 5 | train_loss: 0.7005 | train_acc: 0.7422 | test_loss: 0.6509 | test_acc: 0.8968
Epoch: 6 | train_loss: 0.5988 | train_acc: 0.8984 | test_loss: 0.6597 | test_acc: 0.8769
Epoch: 7 | train_loss: 0.6206 | train_acc: 0.8047 | test_loss: 0.6037 | test_acc: 0.9384
Epoch: 8 | train_loss: 0.5341 | train_acc: 0.8164 | test_loss: 0.6002 | test_acc: 0.8570
Epoch: 9 | train_loss: 0.5031 | train_acc: 0.9258 | test_loss: 0.5606 | test_acc: 0.8570
Epoch: 10 | train_loss: 0.5309 | train_acc: 0.8008 | test_loss: 0.5462 | test_acc: 0.8977
[INFO] Saving model to: models/07_effnetb2_data_10_percent_10_epochs.pth
--------------------------------------------------

[INFO] Experiment number: 5
[INFO] Model: effnetb0
[INFO] DataLoader: data_20_percent
[INFO] Number of epochs: 5
[INFO] Created new effnetb0 model...
[INFO] Created SummaryWriter saving to runs/2022-06-28/data_20_percent/effnetb0/5_epochs
```

{"model_id":"4ac449dde12d4269aee5ddc1ae103fd3","version_major":2,"version_minor":0}

```
Epoch: 1 | train_loss: 0.9645 | train_acc: 0.5938 | test_loss: 0.6673 | test_acc: 0.8864
Epoch: 2 | train_loss: 0.6941 | train_acc: 0.7833 | test_loss: 0.5834 | test_acc: 0.9072
Epoch: 3 | train_loss: 0.5817 | train_acc: 0.8313 | test_loss: 0.5024 | test_acc: 0.9280
Epoch: 4 | train_loss: 0.4582 | train_acc: 0.8917 | test_loss: 0.4186 | test_acc: 0.9072
Epoch: 5 | train_loss: 0.4519 | train_acc: 0.8604 | test_loss: 0.3827 | test_acc: 0.9280
[INFO] Saving model to: models/07_effnetb0_data_20_percent_5_epochs.pth
```

```
--------------------------------------------------

[INFO] Experiment number: 6
[INFO] Model: effnetb2
[INFO] DataLoader: data_20_percent
[INFO] Number of epochs: 5
[INFO] Created new effnetb2 model...
[INFO] Created SummaryWriter saving to
runs/2022-06-28/data_20_percent/effnetb2/5_epochs

{"model_id":"ac3e6e1aead3481886228b86cf25df01","version_major":2,"vers
ion_minor":0}

Epoch: 1 | train_loss: 0.9883 | train_acc: 0.5271 | test_loss: 0.7823
| test_acc: 0.8049
Epoch: 2 | train_loss: 0.7125 | train_acc: 0.8250 | test_loss: 0.6505
| test_acc: 0.8864
Epoch: 3 | train_loss: 0.5772 | train_acc: 0.8917 | test_loss: 0.5523
| test_acc: 0.9384
Epoch: 4 | train_loss: 0.5209 | train_acc: 0.8583 | test_loss: 0.5109
| test_acc: 0.9280
Epoch: 5 | train_loss: 0.4635 | train_acc: 0.8667 | test_loss: 0.4445
| test_acc: 0.9384
[INFO] Saving model to:
models/07_effnetb2_data_20_percent_5_epochs.pth
--------------------------------------------------

[INFO] Experiment number: 7
[INFO] Model: effnetb0
[INFO] DataLoader: data_20_percent
[INFO] Number of epochs: 10
[INFO] Created new effnetb0 model...
[INFO] Created SummaryWriter saving to
runs/2022-06-28/data_20_percent/effnetb0/10_epochs

{"model_id":"91695204baa94fb3bec484d4dcca8228","version_major":2,"vers
ion_minor":0}

Epoch: 1 | train_loss: 0.9645 | train_acc: 0.5938 | test_loss: 0.6673
| test_acc: 0.8864
Epoch: 2 | train_loss: 0.6941 | train_acc: 0.7833 | test_loss: 0.5834
| test_acc: 0.9072
Epoch: 3 | train_loss: 0.5817 | train_acc: 0.8313 | test_loss: 0.5024
| test_acc: 0.9280
Epoch: 4 | train_loss: 0.4582 | train_acc: 0.8917 | test_loss: 0.4186
| test_acc: 0.9072
Epoch: 5 | train_loss: 0.4519 | train_acc: 0.8604 | test_loss: 0.3827
| test_acc: 0.9280
Epoch: 6 | train_loss: 0.4218 | train_acc: 0.8917 | test_loss: 0.3734
| test_acc: 0.9384
```

```
Epoch: 7 | train_loss: 0.4005 | train_acc: 0.8646 | test_loss: 0.3395
| test_acc: 0.9176
Epoch: 8 | train_loss: 0.3453 | train_acc: 0.8896 | test_loss: 0.3497
| test_acc: 0.9384
Epoch: 9 | train_loss: 0.3740 | train_acc: 0.8938 | test_loss: 0.3457
| test_acc: 0.9384
Epoch: 10 | train_loss: 0.3839 | train_acc: 0.8938 | test_loss: 0.2856
| test_acc: 0.9176
[INFO] Saving model to:
models/07_effnetb0_data_20_percent_10_epochs.pth
---------------------------------------------------

[INFO] Experiment number: 8
[INFO] Model: effnetb2
[INFO] DataLoader: data_20_percent
[INFO] Number of epochs: 10
[INFO] Created new effnetb2 model...
[INFO] Created SummaryWriter saving to
runs/2022-06-28/data_20_percent/effnetb2/10_epochs

{"model_id":"847bcefa15744539b3f41d4658ba6cd0","version_major":2,"vers
ion_minor":0}

Epoch: 1 | train_loss: 0.9883 | train_acc: 0.5271 | test_loss: 0.7823
| test_acc: 0.8049
Epoch: 2 | train_loss: 0.7125 | train_acc: 0.8250 | test_loss: 0.6505
| test_acc: 0.8864
Epoch: 3 | train_loss: 0.5772 | train_acc: 0.8917 | test_loss: 0.5523
| test_acc: 0.9384
Epoch: 4 | train_loss: 0.5209 | train_acc: 0.8583 | test_loss: 0.5109
| test_acc: 0.9280
Epoch: 5 | train_loss: 0.4635 | train_acc: 0.8667 | test_loss: 0.4445
| test_acc: 0.9384
Epoch: 6 | train_loss: 0.3801 | train_acc: 0.9146 | test_loss: 0.4582
| test_acc: 0.9280
Epoch: 7 | train_loss: 0.3597 | train_acc: 0.8938 | test_loss: 0.4195
| test_acc: 0.9280
Epoch: 8 | train_loss: 0.3280 | train_acc: 0.9021 | test_loss: 0.4135
| test_acc: 0.9280
Epoch: 9 | train_loss: 0.3467 | train_acc: 0.8896 | test_loss: 0.4325
| test_acc: 0.8674
Epoch: 10 | train_loss: 0.3743 | train_acc: 0.8583 | test_loss: 0.3768
| test_acc: 0.9489
[INFO] Saving model to:
models/07_effnetb2_data_20_percent_10_epochs.pth
---------------------------------------------------

CPU times: user 2min 51s, sys: 27.2 s, total: 3min 18s
Wall time: 5min 29s
```

# 8. View experiments in TensorBoard

We've *experiment, experiment, experiment!*

Now let's *visualize, visualize, visualize*!

```
# Let's view our experiments within TensorBoard from within the
notebook
%load_ext tensorboard
%tensorboard --logdir runs
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 375), started 0:23:43 ago. (Use
'!kill 375' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

The best performing model was:

- Model: EffNetB2
- Dataset: Pizza, steak, sushi 20%
- Epochs: 10

And the overall trend of all the results was that more data, bigger model and longer training time generally led to better results.

```
# # Upload the results to TensorBoard.dev (uncomment to try it out)
!tensorboard dev upload --logdir runs \
    --name "07. PyTorch Experiment Tracking: FoodVision Mini model
result (video)" \
    --description "Comparing results of different model size, training
data amount and training time."
    --one_shot
```

You can view the experiments publically at TensorBoard.dev:
https://tensorboard.dev/experiment/4yXqVFNyQQymt4yfvvs6sA/

# 9. Load in the best model and mkae predictions with it

This is our best model filepath:
`models/07_effnetb2_data_20_percent_10_epochs.pth`

```
# Setup best model filepath
best_model_path = "models/07_effnetb2_data_20_percent_10_epochs.pth"

# Instantiate a new instance of EffNetB2 (to load in the saved
state_dict())
```

```
best_model = create_effnetb2()

# Load the saved best model state_dict()
best_model.load_state_dict(torch.load(best_model_path))
```

```
[INFO] Created new effnetb2 model...
```

```
<All keys matched successfully>
```

Our goal: create a FoodVision Mini model that performs well enough and is able to run on a mobile device/web browser.

```
# Check the model file size
from pathlib import Path

# Get the model size in bytes then convert it to megabytes
effnetb2_model_size = Path(best_model_path).stat().st_size //
(1024*1024)
print(f"EfficientNetB2 feature extractor model size:
{effnetb2_model_size} MB")
```

```
EfficientNetB2 feature extractor model size: 29 MB
```

```
# Import function to make prediction on images and plot them
from going_modular.going_modular.predictions import
pred_and_plot_image

# Get a random list of 3 image path names from the test dataset
import random
num_images_to_plot = 3
test_image_path_list = list(Path(data_20_percent_path /
"test").glob("*/*.jpg"))
test_image_path_sample = random.sample(test_image_path_list,
                                       k=num_images_to_plot)

for image_path in test_image_path_sample:
  pred_and_plot_image(model=best_model,
                      image_path=image_path,
                      class_names=class_names,
                      image_size=(224, 224))
```

Pred: pizza | Prob: 0.961

Pred: pizza | Prob: 0.927


## 9.1 Predict on a custom image with the best model

```
# Download custom image
import requests
from pathlib import Path

# Setup custom image path
custom_image_path = Path("data/04-pizza-dad.jpeg")

# Download the image if it doesn't already exist
if not custom_image_path.is_file():
    with open(custom_image_path, "wb") as f:
        # When downloading from GitHub, need to use the "raw" file
link
        request =
requests.get("https://raw.githubusercontent.com/mrdbourke/pytorch-
deep-learning/main/images/04-pizza-dad.jpeg")
        print(f"Downloading {custom_image_path}...")
        f.write(request.content)
else:
    print(f"{custom_image_path} already exists, skipping download.")
```

```
Downloading data/04-pizza-dad.jpeg...
```

```
# Predict on our own custom image
pred_and_plot_image(model=model,
                    image_path=custom_image_path,
                    class_names=class_names)
```

Pred: pizza | Prob: 0.982