04-pytorch-custom-datasets-video

October 26, 2023

1 04. PyTorch Custom Datasets Video Notebook

We've used some datasets with PyTorch before.

But how do you get your own data into PyTorch?

One of the ways to do so is via: custom datasets.

1.1 Domain libraries

Depending on what you're working on, vision, text, audio, recommendation, you'll want to look into each of the PyTorch domain libraries for existing data loading functions and customizable data loading functions.

Resources: * Book version of the course materials for 04: https://www.learnpytorch.io/04_pytorch_custom_datasets/ * Ground truth version of notebook 04: https://github.com/mrdbourke/pytorch-deep-learning/blob/main/04_pytorch_custom_datasets.ipynb

1.2 0. Importing PyTorch and setting up device-agnostic code

[1]:
```
import torch
from torch import nn

# Note: PyTorch 1.10.0+ is required for this course
torch.__version__
```

[1]:
```
'1.11.0+cu113'
```

[2]:
```
# Setup device-agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

[2]:
```
'cuda'
```

[3]:
```
!nvidia-smi
```

```
Thu Apr 28 02:21:26 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
```

```
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   44C    P0    28W / 250W |      2MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

1.3 1. Get data

Our dataset is a subset of the Food101 dataset.

Food101 starts 101 different classes of food and 1000 images per class (750 training, 250 testing).

Our dataset starts with 3 classes of food and only 10% of the images (~75 training, 25 testing).

Why do this?

When starting out ML projects, it's important to try things on a small scale and then increase the scale when necessary.

The whole point is to speed up how fast you can experiment.

[4]:
```python
import requests
import zipfile
from pathlib import Path

# Setup path to a data folder
data_path = Path("data/")
image_path = data_path / "pizza_steak_sushi"

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

# Download pizza, steak and suhsi data
with open(data_path / "pizza_steak_sushi.zip", "wb") as f:
```

```
    request = requests.get("https://github.com/mrdbourke/pytorch-deep-learning/
↪raw/main/data/pizza_steak_sushi.zip")
    print("Downloading pizza, steak, suhsi data...")
    f.write(request.content)

# Unzip pizza, steak, sushi data
with zipfile.ZipFile(data_path / "pizza_steak_sushi.zip", "r") as zip_ref:
  print("Unzipping pizza, steak and sushi data...")
  zip_ref.extractall(image_path)
```

```
data/pizza_steak_sushi does not exist, creating one…
Downloading pizza, steak, suhsi data…
Unzipping pizza, steak and sushi data…
```

1.4 2. Becoming one with the data (data preparation and data exploration)

[5]:
```
import os
def walk_through_dir(dir_path):
  """Walks through dir_path returning its contents."""
  for dirpath, dirnames, filenames in os.walk(dir_path):
    print(f"There are {len(dirnames)} directories and {len(filenames)} images␣
↪in '{dirpath}'.")
```

[6]:
```
walk_through_dir(image_path)
```

```
There are 2 directories and 0 images in 'data/pizza_steak_sushi'.
There are 3 directories and 0 images in 'data/pizza_steak_sushi/test'.
There are 0 directories and 31 images in 'data/pizza_steak_sushi/test/sushi'.
There are 0 directories and 19 images in 'data/pizza_steak_sushi/test/steak'.
There are 0 directories and 25 images in 'data/pizza_steak_sushi/test/pizza'.
There are 3 directories and 0 images in 'data/pizza_steak_sushi/train'.
There are 0 directories and 72 images in 'data/pizza_steak_sushi/train/sushi'.
There are 0 directories and 75 images in 'data/pizza_steak_sushi/train/steak'.
There are 0 directories and 78 images in 'data/pizza_steak_sushi/train/pizza'.
```

[7]:
```
# Setup train and testing paths
train_dir = image_path / "train"
test_dir = image_path / "test"

train_dir, test_dir
```

[7]:
```
(PosixPath('data/pizza_steak_sushi/train'),
 PosixPath('data/pizza_steak_sushi/test'))
```

1.4.1 2.1 Visualizing and image

Let's write some code to: 1. Get all of the image paths 2. Pick a random image path using Python's random.choice() 3. Get the image class name using `pathlib.Path.parent.stem` 4. Since we're working with images, let's open the image with Python's PIL 5. We'll then show the image and print metadata

[8]:

```
image_path
```

[8]:

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

[9]:

```
# /content/data/pizza_steak_sushi
```

[10]:

```
import random
from PIL import Image

# Set seed
# random.seed(42)

# 1. Get all image paths
image_path_list = list(image_path.glob("*/*/*.jpg"))

# 2. Pick a random image path
random_image_path = random.choice(image_path_list)

# 3. Get image class from path name (the image class is the name of the␣
 ↪directory where the image is stored)
image_class = random_image_path.parent.stem

# 4. Open image
img = Image.open(random_image_path)

# 5. Print metadata
print(f"Random image path: {random_image_path}")
print(f"Image class: {image_class}")
print(f"Image height: {img.height}")
print(f"Image width: {img.width}")
img
```

```
Random image path: data/pizza_steak_sushi/train/steak/1615395.jpg
Image class: steak
Image height: 384
Image width: 512
```

[10]:

```
[11]: import numpy as np
      import matplotlib.pyplot as plt

      # Turn the image into an array
      img_as_array = np.asarray(img)

      # Plot the image with matplotlib
      plt.figure(figsize=(10, 7))
      plt.imshow(img_as_array)
      plt.title(f"Image class: {image_class} | Image shape: {img_as_array.shape} ->␣
        ↪[height, width, color_channels] (HWC)")
      plt.axis(False);
```

Image class: steak | Image shape: (384, 512, 3) -> [height, width, color_channels] (HWC)

[12]:

```
img_as_array
```

[12]:

```
array([[[ 17,  12,  16],
        [ 16,  11,  15],
        [ 14,   9,  13],
        …,
        [ 29,  14,   9],
        [ 26,  12,   9],
        [ 24,  10,   9]],

       [[ 16,  11,  17],
        [ 15,  10,  16],
        [ 14,   9,  15],
        …,
        [ 26,  10,  10],
        [ 24,  10,   9],
        [ 23,   9,   9]],

       [[ 15,  10,  17],
        [ 15,  10,  17],
        [ 14,   9,  16],
```

```
 …,
 [ 25,   9,  12],
 [ 22,   7,  12],
 [ 21,   6,  11]],

…,

[[134, 125,  66],
 [136, 126,  67],
 [141, 126,  69],
 …,
 [ 39,  62,  76],
 [ 37,  63,  78],
 [ 34,  62,  76]],

[[135, 126,  67],
 [135, 124,  68],
 [137, 122,  67],
 …,
 [ 37,  60,  74],
 [ 33,  61,  75],
 [ 30,  58,  72]],

[[141, 132,  75],
 [138, 127,  71],
 [138, 123,  68],
 …,
 [ 37,  60,  74],
 [ 31,  59,  73],
 [ 28,  56,  70]]], dtype=uint8)
```

1.5 3. Transforming data

Before we can use our image data with PyTorch: 1. Turn your target data into tensors (in our case, numerical representation of our images). 2. Turn it into a `torch.utils.data.Dataset` and subsequently a `torch.utils.data.DataLoader`, we'll call these `Dataset` and `DataLoader`.

[13]:
```python
import torch
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
```

1.5.1 3.1 Transforming data with `torchvision.transforms`

Transforms help you get your images ready to be used with a model/perform data augmentation - https://pytorch.org/vision/stable/transforms.html

[14]:

```
# Write a transform for image
data_transform = transforms.Compose([
  # Resize our images to 64x64
  transforms.Resize(size=(64, 64)),
  # Flip the images randomly on the horizontal
  transforms.RandomHorizontalFlip(p=0.5),
  # Turn the image into a torch.Tensor
  transforms.ToTensor()
])
```

[15]:

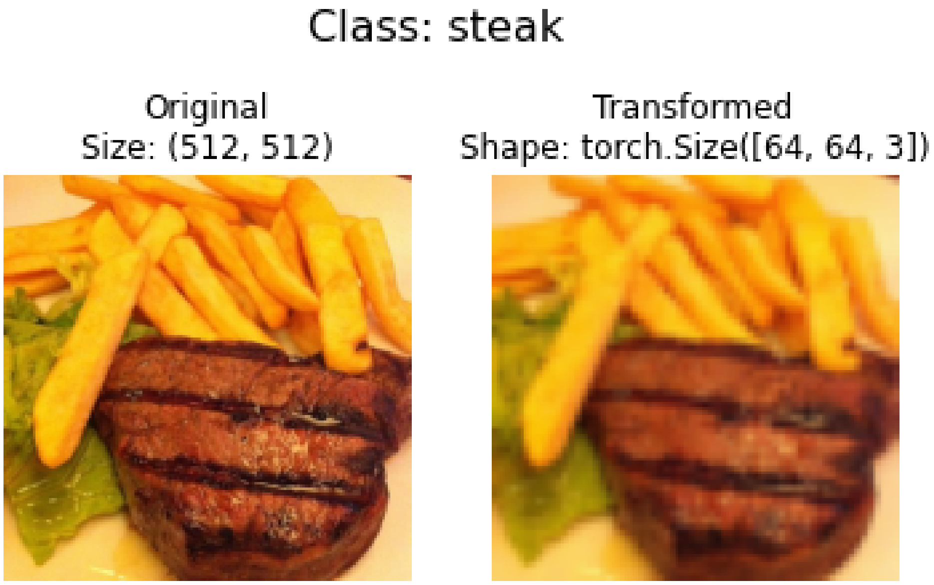
```
data_transform(img).shape
```

[15]:

```
torch.Size([3, 64, 64])
```

[16]:

```
def plot_transformed_images(image_paths: list, transform, n=3, seed=None):
  """
  Selects random images from a path of images and loads/transforms
  them then plots the original vs the transformed version.
  """
  if seed:
    random.seed(seed)
  random_image_paths = random.sample(image_paths, k=n)
  for image_path in random_image_paths:
    with Image.open(image_path) as f:
      fig, ax = plt.subplots(nrows=1, ncols=2)
      ax[0].imshow(f)
      ax[0].set_title(f"Original\nSize: {f.size}")
      ax[0].axis(False)

      # Transform and plot target image
      transformed_image = transform(f).permute(1, 2, 0) # note we will need to␣
 ↪change shape for matplotlib (C, H, W) -> (H, W, C)
      ax[1].imshow(transformed_image)
      ax[1].set_title(f"Transformed\nShape: {transformed_image.shape}")
      ax[1].axis("off")

      fig.suptitle(f"Class: {image_path.parent.stem}", fontsize=16)

plot_transformed_images(image_paths=image_path_list,
                        transform=data_transform,
                        n=3,
                        seed=None)
```


Class: steak
Original
Size: (512, 512)
Transformed
Shape: torch.Size([64, 64, 3])


Class: sushi
Original
Size: (512, 512)
Transformed
Shape: torch.Size([64, 64, 3])

1.6 4. Option 1: Loading image data using `ImageFolder`

We can load image classification data using `torchvision.datasets.ImageFolder` - https://pytorch.org/vision/stable/generated/torchvision.datasets.ImageFolder.html#torchvision.datasets.ImageFo

[17]:
```
# Use ImageFolder to create dataset(s)
from torchvision import datasets
train_data = datasets.ImageFolder(root=train_dir,
                                  transform=data_transform, # a transform for␣
 ↪the data
                                  target_transform=None) # a transform for the␣
 ↪label/target

test_data = datasets.ImageFolder(root=test_dir,
                                 transform=data_transform)

train_data, test_data
```

[17]:
```
(Dataset ImageFolder
     Number of datapoints: 225
     Root location: data/pizza_steak_sushi/train
     StandardTransform
 Transform: Compose(
                Resize(size=(64, 64), interpolation=bilinear, max_size=None,
antialias=None)
                RandomHorizontalFlip(p=0.5)
                ToTensor()
```

```
             ), Dataset ImageFolder
      Number of datapoints: 75
      Root location: data/pizza_steak_sushi/test
      StandardTransform
 Transform: Compose(
                Resize(size=(64, 64), interpolation=bilinear, max_size=None,
antialias=None)
                RandomHorizontalFlip(p=0.5)
                ToTensor()
             ))
```

[18]:
```
train_dir, test_dir
```

[18]:
```
(PosixPath('data/pizza_steak_sushi/train'),
 PosixPath('data/pizza_steak_sushi/test'))
```

[19]:
```
# Get class names as list
class_names = train_data.classes
class_names
```

[19]:
```
['pizza', 'steak', 'sushi']
```

[20]:
```
# Get class names as dict
class_dict = train_data.class_to_idx
class_dict
```

[20]:
```
{'pizza': 0, 'steak': 1, 'sushi': 2}
```

[21]:
```
# Check the lengths of our dataset
len(train_data), len(test_data)
```

[21]:
```
(225, 75)
```

[22]:
```
train_data.samples[0]
```

[22]:
```
('data/pizza_steak_sushi/train/pizza/1008844.jpg', 0)
```

[23]:
```
# Index on the train_data Dataset to get a single image and label
img, label = train_data[0][0], train_data[0][1]
print(f"Image tensor:\n {img}")
print(f"Image shape: {img.shape}")
print(f"Image datatype: {img.dtype}")
print(f"Image label: {label}")
print(f"Label datatype: {type(label)}")
```

```
Image tensor:
 tensor([[[0.1137, 0.1020, 0.0980,  …, 0.1255, 0.1216, 0.1176],
         [0.1059, 0.0980, 0.0980,  …, 0.1294, 0.1294, 0.1294],
```

```
          [0.1020, 0.0980, 0.0941,  …, 0.1333, 0.1333, 0.1333],
          …,
          [0.1098, 0.1098, 0.1255,  …, 0.1686, 0.1647, 0.1686],
          [0.0863, 0.0941, 0.1098,  …, 0.1686, 0.1647, 0.1686],
          [0.0863, 0.0863, 0.0980,  …, 0.1686, 0.1647, 0.1647]],

         [[0.0745, 0.0706, 0.0745,  …, 0.0588, 0.0588, 0.0588],
          [0.0706, 0.0706, 0.0745,  …, 0.0627, 0.0627, 0.0627],
          [0.0706, 0.0745, 0.0745,  …, 0.0706, 0.0706, 0.0706],
          …,
          [0.1255, 0.1333, 0.1373,  …, 0.2510, 0.2392, 0.2392],
          [0.1098, 0.1176, 0.1255,  …, 0.2510, 0.2392, 0.2314],
          [0.1020, 0.1059, 0.1137,  …, 0.2431, 0.2353, 0.2275]],

         [[0.0941, 0.0902, 0.0902,  …, 0.0196, 0.0196, 0.0196],
          [0.0902, 0.0863, 0.0902,  …, 0.0196, 0.0157, 0.0196],
          [0.0902, 0.0902, 0.0902,  …, 0.0157, 0.0157, 0.0196],
          …,
          [0.1294, 0.1333, 0.1490,  …, 0.1961, 0.1882, 0.1804],
          [0.1098, 0.1137, 0.1255,  …, 0.1922, 0.1843, 0.1804],
          [0.1059, 0.1020, 0.1059,  …, 0.1843, 0.1804, 0.1765]]])
Image shape: torch.Size([3, 64, 64])
Image datatype: torch.float32
Image label: 0
Label datatype: <class 'int'>
```

[24]:
```
# Rearrange the order dimensions
img_permute = img.permute(1, 2, 0)

# Print out different shapes
print(f"Original shape: {img.shape} -> [color_channels, height, width]")
print(f"Image permute: {img_permute.shape} -> [height, width, color_channels]")

# Plot the image
plt.figure(figsize=(10, 7))
plt.imshow(img_permute)
plt.axis("off")
plt.title(class_names[label], fontsize=14)
```

```
Original shape: torch.Size([3, 64, 64]) -> [color_channels, height, width]
Image permute: torch.Size([64, 64, 3]) -> [height, width, color_channels]
```

[24]:
```
Text(0.5, 1.0, 'pizza')
```

1.6.1 4.1 Turn loaded images into `DataLoader`'s

A `DataLoader` is going to help us turn our `Dataset`'s into iterables and we can customise the `batch_size` so our model can see `batch_size` images at a time.

[25]:
```
import os
os.cpu_count()
```

[25]:
```
2
```

[26]:
```
# Turn train and test datasets into DataLoader's
from torch.utils.data import DataLoader
BATCH_SIZE=1
train_dataloader = DataLoader(dataset=train_data,
```

```
                              batch_size=BATCH_SIZE,
                              num_workers=1,
                              shuffle=True)

test_dataloader = DataLoader(dataset=test_data,
                             batch_size=BATCH_SIZE,
                             num_workers=1,
                             shuffle=False)

train_dataloader, test_dataloader
```

[26]:

```
(<torch.utils.data.dataloader.DataLoader at 0x7f2a1bf8b490>,
 <torch.utils.data.dataloader.DataLoader at 0x7f2a1bf8bb50>)
```

[27]:

```
len(train_dataloader), len(test_dataloader)
```

[27]:

```
(225, 75)
```

[28]:

```
img, label = next(iter(train_dataloader))

# Batch size will now be 1, you can change the batch size if you like
print(f"Image shape: {img.shape} -> [batch_size, color_channels, height,␣
 ↪width]")
print(f"Label shape: {label.shape}")
```

```
Image shape: torch.Size([1, 3, 64, 64]) -> [batch_size, color_channels, height,
width]
Label shape: torch.Size([1])
```

1.7 5 Option 2: Loading Image Data with a Custom `Dataset`

1. Want to be able to load images from file
2. Want to be able to get class names from the Dataset
3. Want to be able to get classes as dictionary from the Dataset

Pros: * Can create a `Dataset` out of almost anything * Not limited to PyTorch pre-built `Dataset` functions

Cons: * Even though you could create `Dataset` out of almost anything, it doesn't mean it will work… * Using a custom `Dataset` often results in us writing more code, which could be prone to errors or performance issues

All custom datasets in PyTorch, often subclass - https://pytorch.org/docs/stable/data.html#torch.utils.data.Datas

[29]:

```
import os
import pathlib
import torch

from PIL import Image
```

```
from torch.utils.data import Dataset
from torchvision import transforms
from typing import Tuple, Dict, List
```

[30]:

```
# Instance of torchvision.datasets.ImageFolder()
train_data.classes, train_data.class_to_idx
```

[30]:

```
(['pizza', 'steak', 'sushi'], {'pizza': 0, 'steak': 1, 'sushi': 2})
```

1.7.1 5.1 Creating a helper function to get class names

We want a function to: 1. Get the class names using `os.scandir()` to traverse a target directory (ideally the directory is in standard image classification format). 2. Raise an error if the class names aren't found (if this happens, there might be something wrong with the directory structure). 3. Turn the class names into a dict and a list and return them.

[31]:

```
# Setup path for target directory
target_directory = train_dir
print(f"Target dir: {target_directory}")

# Get the class names from the target directory
class_names_found = sorted([entry.name for entry in list(os.
 ↪scandir(target_directory))])
class_names_found
```

```
Target dir: data/pizza_steak_sushi/train
```

[31]:

```
['pizza', 'steak', 'sushi']
```

[32]:

```
list(os.scandir(target_directory))
```

[32]:

```
[<DirEntry 'sushi'>, <DirEntry 'steak'>, <DirEntry 'pizza'>]
```

[33]:

```
def find_classes(directory: str) -> Tuple[List[str], Dict[str, int]]:
    """Finds the class folder names in a target directory."""
    # 1. Get the class names by scanning the target directory
    classes = sorted(entry.name for entry in os.scandir(directory) if entry.
 ↪is_dir())

    # 2. Raise an error if class names could not be found
    if not classes:
        raise FileNotFoundError(f"Couldn't find any classes in {directory}...␣
 ↪please check file structure.")

    # 3. Create a dictionary of index labels (computers prefer numbers rather␣
 ↪than strings as labels)
    class_to_idx = {class_name: i for i, class_name in enumerate(classes)}
```

```
    return classes, class_to_idx
```

```
[34]: find_classes(target_directory)
```

```
[34]: (['pizza', 'steak', 'sushi'], {'pizza': 0, 'steak': 1, 'sushi': 2})
```

1.7.2 5.2 Create a custom `Dataset` to replicate `ImageFolder`

To create our own custom dataset, we want to:

1. Subclass `torch.utils.data.Dataset`
2. Init our subclass with a target directory (the directory we'd like to get data from) as well as a transform if we'd like to transform our data.
3. Create several attributes:

- paths - paths of our images
- transform - the transform we'd like to use
- classes - a list of the target classes
- class_to_idx - a dict of the target classes mapped to integer labels

4. Create a function to `load_images()`, this function will open an image
5. Overwrite the `__len()__` method to return the length of our dataset
6. Overwrite the `__getitem()__` method to return a given sample when passed an index

```
[35]: # 0. Write a custom dataset class
from torch.utils.data import Dataset

# 1. Subclass torch.utils.data.Dataset
class ImageFolderCustom(Dataset):
  # 2. Initialize our custom dataset
  def __init__(self,
               targ_dir: str,
               transform=None):
    # 3. Create class attributes
    # Get all of the image paths
    self.paths = list(pathlib.Path(targ_dir).glob("*/*.jpg"))
    # Setup transform
    self.transform = transform
    # Create classes and class_to_idx attributes
    self.classes, self.class_to_idx = find_classes(targ_dir)

  # 4. Create a function to load images
  def load_image(self, index: int) -> Image.Image:
    "Opens an image via a path and returns it."
    image_path = self.paths[index]
    return Image.open(image_path)

  # 5. Overwrite __len__()
  def __len__(self) -> int:
```

```
    "Returns the total number of samples."
    return len(self.paths)

  # 6. Overwrite __getitem__() method to return a particular sample
  def __getitem__(self, index: int) -> Tuple[torch.Tensor, int]:
    "Returns one sample of data, data and label (X, y)."
    img = self.load_image(index)
    class_name = self.paths[index].parent.name # expects path in format:␣
↪data_folder/class_name/image.jpg
    class_idx = self.class_to_idx[class_name]

    # Transform if necessary
    if self.transform:
      return self.transform(img), class_idx # return data, label (X, y)
    else:
      return img, class_idx # return untransformed image and label
```

[36]:

```
img, label = train_data[0]
```

[37]:

```
img, label
```

[37]:

```
(tensor([[[0.1176, 0.1216, 0.1255,  …, 0.0980, 0.1020, 0.1137],
          [0.1294, 0.1294, 0.1294,  …, 0.0980, 0.0980, 0.1059],
          [0.1333, 0.1333, 0.1333,  …, 0.0941, 0.0980, 0.1020],
          …,
          [0.1686, 0.1647, 0.1686,  …, 0.1255, 0.1098, 0.1098],
          [0.1686, 0.1647, 0.1686,  …, 0.1098, 0.0941, 0.0863],
          [0.1647, 0.1647, 0.1686,  …, 0.0980, 0.0863, 0.0863]],

         [[0.0588, 0.0588, 0.0588,  …, 0.0745, 0.0706, 0.0745],
          [0.0627, 0.0627, 0.0627,  …, 0.0745, 0.0706, 0.0706],
          [0.0706, 0.0706, 0.0706,  …, 0.0745, 0.0745, 0.0706],
          …,
          [0.2392, 0.2392, 0.2510,  …, 0.1373, 0.1333, 0.1255],
          [0.2314, 0.2392, 0.2510,  …, 0.1255, 0.1176, 0.1098],
          [0.2275, 0.2353, 0.2431,  …, 0.1137, 0.1059, 0.1020]],

         [[0.0196, 0.0196, 0.0196,  …, 0.0902, 0.0902, 0.0941],
          [0.0196, 0.0157, 0.0196,  …, 0.0902, 0.0863, 0.0902],
          [0.0196, 0.0157, 0.0157,  …, 0.0902, 0.0902, 0.0902],
          …,
          [0.1804, 0.1882, 0.1961,  …, 0.1490, 0.1333, 0.1294],
          [0.1804, 0.1843, 0.1922,  …, 0.1255, 0.1137, 0.1098],
          [0.1765, 0.1804, 0.1843,  …, 0.1059, 0.1020, 0.1059]]]), 0)
```

[38]:

```
# Create a transform
from torchvision import transforms
```

```
train_transforms = transforms.Compose([
                                      transforms.Resize(size=(64, 64)),
                                      transforms.RandomHorizontalFlip(p=0.5),
                                      transforms.ToTensor()
])

test_transforms = transforms.Compose([
                                      transforms.Resize(size=(64, 64)),
                                      transforms.ToTensor()
])
```

[39]:

```
# Test out ImageFolderCustom
train_data_custom = ImageFolderCustom(targ_dir=train_dir,
                                      transform=train_transforms)

test_data_custom = ImageFolderCustom(targ_dir=test_dir,
                                     transform=test_transforms)
```

[40]:

```
train_data_custom, test_data_custom
```

[40]:

```
(<__main__.ImageFolderCustom at 0x7f2a1c032590>,
 <__main__.ImageFolderCustom at 0x7f2a1c032990>)
```

[41]:

```
len(train_data), len(train_data_custom)
```

[41]:

```
(225, 225)
```

[42]:

```
len(test_data), len(test_data_custom)
```

[42]:

```
(75, 75)
```

[43]:

```
train_data_custom.classes
```

[43]:

```
['pizza', 'steak', 'sushi']
```

[44]:

```
train_data_custom.class_to_idx
```

[44]:

```
{'pizza': 0, 'steak': 1, 'sushi': 2}
```

[45]:

```
# Check for equality between original ImageFolder Dataset and ImageFolderCustom␣
 ↪Dataset
print(train_data_custom.classes==train_data.classes)
print(test_data_custom.classes==test_data.classes)
```

```
True
True
```

1.7.3 5.3 Create a function to display random images

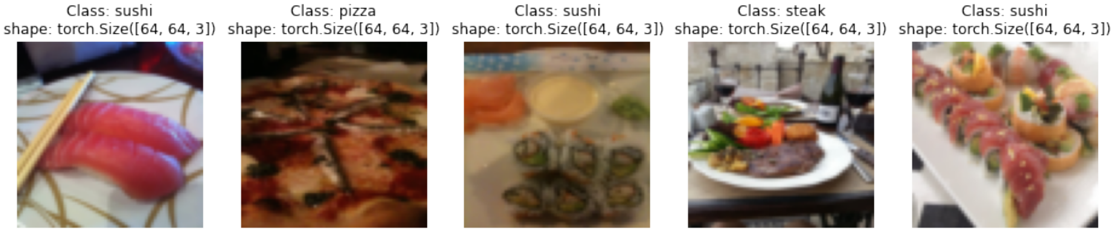
1. Take in a `Dataset` and a number of other parameters such as class names and how many images to visualize.
2. To prevent the display getting out of hand, let's cap the number of images to see at 10.
3. Set the random seed for reproducibility
4. Get a list of random sample indexes from the target dataset.
5. Setup a matplotlib plot.
6. Loop through the random sample indexes and plot them with matploltib.
7. Make sure the dimensions of our images line up with matplotlib (HWC)

[46]:

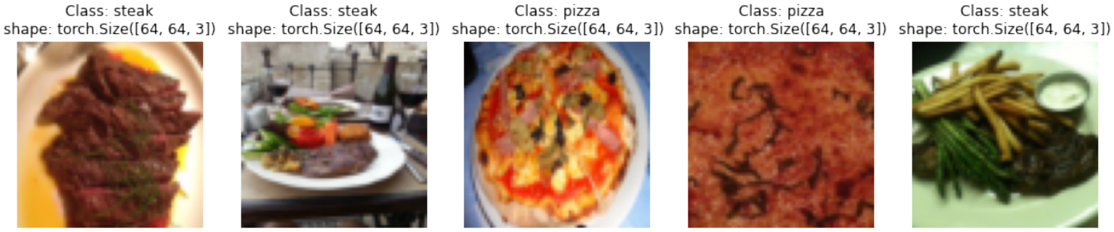
```
# 1. Create a function to take in a dataset
def display_random_images(dataset: torch.utils.data.Dataset,
                          classes: List[str] = None,
                          n: int = 10,
                          display_shape: bool = True,
                          seed: int = None):
  # 2. Adjust display if n is too high
  if n > 10:
    n = 10
    display_shape = False
    print(f"For display, purposes, n shouldn't be larger than 10, setting to 10␣
↪and removing shape display.")

  # 3. Set the seed
  if seed:
    random.seed(seed)

  # 4. Get random sample indexes
  random_samples_idx = random.sample(range(len(dataset)), k=n)

  # 5. Setup plot
  plt.figure(figsize=(16, 8))

  # 6. Loop through random indexes and plot them with matplotlib
  for i, targ_sample in enumerate(random_samples_idx):
    targ_image, targ_label = dataset[targ_sample][0], dataset[targ_sample][1]

    # 7. Adjust tensor dimensions for plotting
    targ_image_adjust = targ_image.permute(1, 2, 0) # [color_channels, height,␣
↪width] -> [height, width, color_channels]

    # Plot adjusted samples
    plt.subplot(1, n, i+1)
    plt.imshow(targ_image_adjust)
    plt.axis("off")
    if classes:
      title = f"Class: {classes[targ_label]}"
```

```
      if display_shape:
        title = title + f"\nshape: {targ_image_adjust.shape}"
    plt.title(title)
```

[47]:
```
# Display random images from the ImageFolder created Dataset
display_random_images(train_data,
                      n=5,
                      classes=class_names,
                      seed=None)
```



[48]:
```
# Display random images from the ImageFolderCustom Dataset
display_random_images(train_data_custom,
                      n=5,
                      classes=class_names,
                      seed=None)
```



1.7.4 5.4 Turn custom loaded images into DataLoader's

[49]:
```
from torch.utils.data import DataLoader
BATCH_SIZE = 1
NUM_WORKERS = os.cpu_count()
train_dataloader_custom = DataLoader(dataset=train_data_custom,
                                     batch_size=BATCH_SIZE,
                                     num_workers=NUM_WORKERS,
                                     shuffle=True)

test_dataloader_custom = DataLoader(dataset=test_data_custom,
```

```
                                         batch_size=BATCH_SIZE,
                                         num_workers=NUM_WORKERS,
                                         shuffle=False)

train_dataloader_custom, test_dataloader_custom
```

[49]:
```
(<torch.utils.data.dataloader.DataLoader at 0x7f2a1b61aa10>,
 <torch.utils.data.dataloader.DataLoader at 0x7f2a1b61aed0>)
```

[50]:
```
# Get image and label from custom datloader
img_custom, label_custom = next(iter(train_dataloader_custom))

# Print out the shapes
img_custom.shape, label_custom.shape
```

[50]:
```
(torch.Size([1, 3, 64, 64]), torch.Size([1]))
```

1.8 6. Other forms of transforms (data augmentation)

Data augmentation is the process of artificially adding diversity to your training data.

In the case of image data, this may mean applying various image transformations to the training images.

This practice hopefully results in a model that's more generalizable to unseen data.

Let's take a look at one particular type of data augmentation used to train PyTorch vision models to state of the art levels…

Blog post: https://pytorch.org/blog/how-to-train-state-of-the-art-models-using-torchvision-latest-primitives/#break-down-of-key-accuracy-improvements

[51]:
```
# Let's look at trivailaugment - https://pytorch.org/vision/stable/
 ↪auto_examples/plot_transforms.html#trivialaugmentwide
from torchvision import transforms

train_transform = transforms.Compose([
                                      transforms.Resize(size=(224, 224)),
                                      transforms.
 ↪TrivialAugmentWide(num_magnitude_bins=31),
                                      transforms.ToTensor()
])

test_transform = transforms.Compose([
                                      transforms.Resize(size=(224, 224)),
                                      transforms.ToTensor()
])
```

[52]:
```
image_path
```

[52]:

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

[53]:

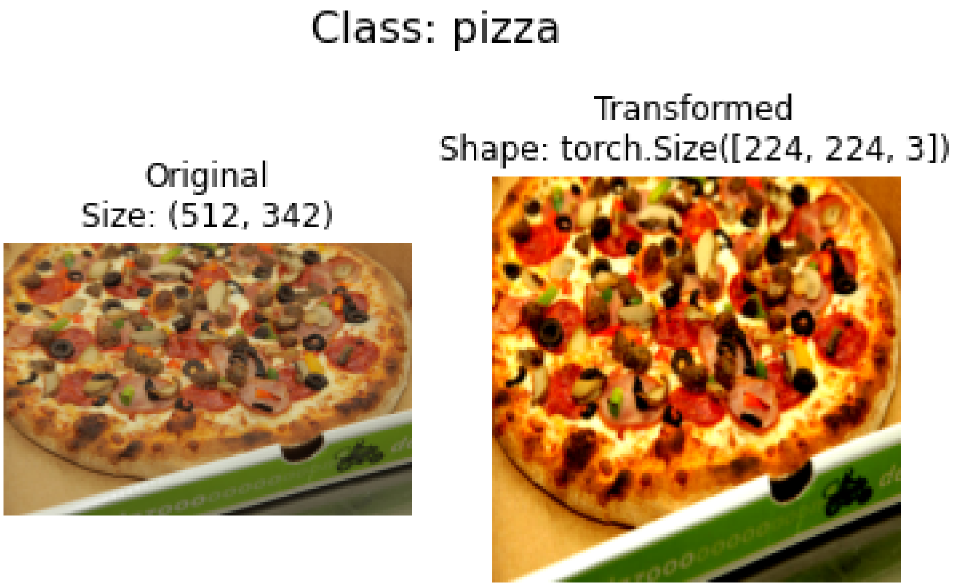
```
# Get all image paths
image_path_list = list(image_path.glob("*/*/*.jpg"))
image_path_list[:10]
```

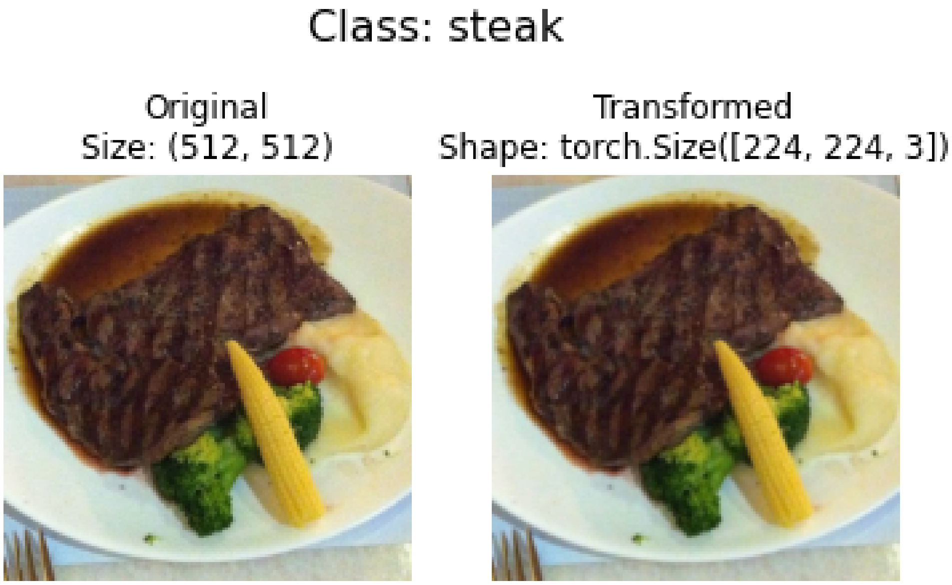
[53]:

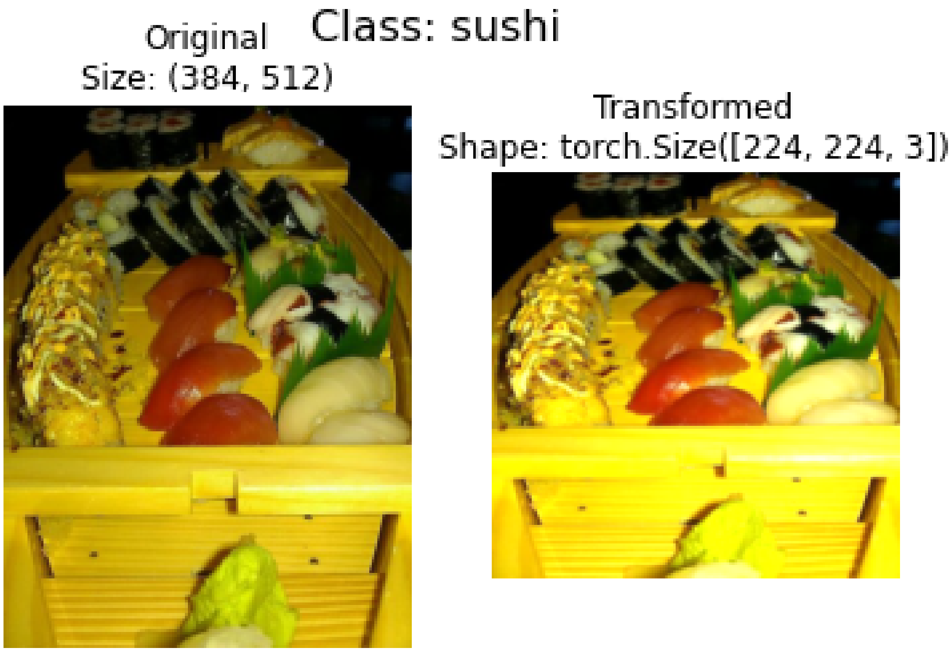
```
[PosixPath('data/pizza_steak_sushi/test/sushi/2521706.jpg'),
 PosixPath('data/pizza_steak_sushi/test/sushi/499605.jpg'),
 PosixPath('data/pizza_steak_sushi/test/sushi/988559.jpg'),
 PosixPath('data/pizza_steak_sushi/test/sushi/1987407.jpg'),
 PosixPath('data/pizza_steak_sushi/test/sushi/1172255.jpg'),
 PosixPath('data/pizza_steak_sushi/test/sushi/479711.jpg'),
 PosixPath('data/pizza_steak_sushi/test/sushi/1742201.jpg'),
 PosixPath('data/pizza_steak_sushi/test/sushi/1230335.jpg'),
 PosixPath('data/pizza_steak_sushi/test/sushi/3837522.jpg'),
 PosixPath('data/pizza_steak_sushi/test/sushi/207578.jpg')]
```

[54]:

```
# Plot random transformed images
plot_transformed_images(
    image_paths=image_path_list,
    transform=train_transform,
    n=3,
    seed=None
)
```







1.9 7. Model 0: TinyVGG without data augmentation

Let's replicate TinyVGG architecture from the CNN Explainer website: https://poloclub.github.io/cnn-explainer/

1.9.1 7.1 Creating transforms and loading data for Model 0

[55]:

```
# Create simple transform
simple_transform = transforms.Compose([
                                       transforms.Resize(size=(64, 64)),
                                       transforms.ToTensor()
])
```

[56]:

```
# 1. Load and transform data
from torchvision import datasets
train_data_simple = datasets.ImageFolder(root=train_dir,
                                         transform=simple_transform)
test_data_simple = datasets.ImageFolder(root=test_dir,
                                        transform=simple_transform)

# 2. Turn the datasets into DataLoaders
import os
from torch.utils.data import DataLoader

# Setup batch size and number of works
BATCH_SIZE = 32
NUM_WORKERS = os.cpu_count()

# Create DataLoader's
train_dataloader_simple = DataLoader(dataset=train_data_simple,
                                     batch_size=BATCH_SIZE,
                                     shuffle=True,
                                     num_workers=NUM_WORKERS)
test_dataloader_simple = DataLoader(dataset=test_data_simple,
                                    batch_size=BATCH_SIZE,
                                    shuffle=False,
                                    num_workers=NUM_WORKERS)
```

1.9.2 7.2 Create TinyVGG model class

[57]:

```
class TinyVGG(nn.Module):
  """
  Model architecture copying TinyVGG from CNN Explainer: https://poloclub.
 ↪github.io/cnn-explainer/
  """
  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,
                  kernel_size=3,
                  stride=1,
                  padding=0),
        nn.ReLU(),
        nn.Conv2d(in_channels=hidden_units,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=0),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2,
                     stride=2) # default stride value is same as kernel_size
    )
    self.conv_block_2 = nn.Sequential(
        nn.Conv2d(in_channels=hidden_units,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=0),
        nn.ReLU(),
        nn.Conv2d(in_channels=hidden_units,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=0),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2,
                     stride=2) # default stride value is same as kernel_size
    )
    self.classifier = nn.Sequential(
        nn.Flatten(),
        nn.Linear(in_features=hidden_units*13*13,
                  out_features=output_shape)
    )

  def forward(self, x):
    x = self.conv_block_1(x)
    # print(x.shape)
    x = self.conv_block_2(x)
    # print(x.shape)
    x = self.classifier(x)
    # print(x.shape)
    return x
    # return self.classifier(self.conv_block_2(self.conv_block_1(x))) # benefits from operator fusion: https://horace.io/brrr_intro.html
```

```
[58]: torch.manual_seed(42)
      model_0 = TinyVGG(input_shape=3, # number of color channels in our image data
                        hidden_units=10,
                        output_shape=len(class_names)).to(device)

      model_0
```

```
[58]: TinyVGG(
        (conv_block_1): Sequential(
          (0): Conv2d(3, 10, kernel_size=(3, 3), stride=(1, 1))
          (1): ReLU()
          (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1))
          (3): ReLU()
          (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
      ceil_mode=False)
        )
        (conv_block_2): Sequential(
          (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1))
          (1): ReLU()
          (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1))
          (3): ReLU()
          (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
      ceil_mode=False)
        )
        (classifier): Sequential(
          (0): Flatten(start_dim=1, end_dim=-1)
          (1): Linear(in_features=1690, out_features=3, bias=True)
        )
      )
```

1.9.3 7.3 Try a forward pass on a single image (to test the model)

```
[59]: # Get a single image batch
      image_batch, label_batch = next(iter(train_dataloader_simple))
      image_batch.shape, label_batch.shape
```

```
[59]: (torch.Size([32, 3, 64, 64]), torch.Size([32]))
```

```
[60]: # Try a forward pass
      model_0(image_batch.to(device))
```

```
[60]: tensor([[ 2.0789e-02, -1.9351e-03,  9.5318e-03],
              [ 1.8427e-02,  2.4670e-03,  6.6757e-03],
              [ 1.7699e-02,  1.0262e-03,  9.4657e-03],
              [ 2.4441e-02, -3.3526e-03,  9.6011e-03],
              [ 1.9930e-02,  6.6316e-04,  1.0779e-02],
              [ 2.1281e-02,  2.0434e-03,  5.0046e-03],
```

```
        [ 2.0999e-02,  1.2870e-04,  1.2473e-02],
        [ 2.1577e-02, -1.9507e-03,  9.6941e-03],
        [ 2.4504e-02, -4.7745e-03,  8.5280e-03],
        [ 2.0252e-02, -4.7293e-04,  1.0908e-02],
        [ 2.2215e-02, -4.1837e-04,  9.8123e-03],
        [ 2.2313e-02, -2.1622e-03,  9.4455e-03],
        [ 2.1841e-02, -3.7132e-03,  8.3783e-03],
        [ 2.2863e-02, -1.7723e-03,  1.0287e-02],
        [ 2.1647e-02, -4.4139e-03,  9.5022e-03],
        [ 2.2096e-02, -4.1426e-03,  9.3853e-03],
        [ 2.1209e-02, -4.4219e-03,  1.1475e-02],
        [ 2.1711e-02, -2.7656e-03,  8.5006e-03],
        [ 1.9951e-02,  2.8268e-05,  8.4380e-03],
        [ 1.8298e-02,  1.6306e-03,  8.5499e-03],
        [ 2.0768e-02,  1.7942e-03,  7.9412e-03],
        [ 1.9834e-02, -3.9071e-03,  9.8740e-03],
        [ 2.0893e-02,  1.3043e-04,  8.4190e-03],
        [ 2.3202e-02, -3.4329e-03,  9.4937e-03],
        [ 2.0501e-02, -2.5545e-03,  8.4874e-03],
        [ 1.8145e-02,  2.0844e-03,  8.2223e-03],
        [ 2.0304e-02, -1.7637e-03,  7.8751e-03],
        [ 1.7263e-02, -3.3585e-04,  1.2474e-02],
        [ 1.8347e-02, -1.5215e-03,  9.4640e-03],
        [ 1.9868e-02, -2.3248e-03,  9.0062e-03],
        [ 2.2065e-02, -4.6434e-03,  1.2666e-02],
        [ 1.8059e-02,  2.6858e-03,  5.7899e-03]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
```

1.9.4 7.4 Use `torchinfo` to get an idea of the shapes going through our model

[61]:
```
# Install torchinfo, import if it's available
try:
  import torchinfo
except:
  !pip install torchinfo
  import torchinfo

from torchinfo import summary
summary(model_0, input_size=[1, 3, 64, 64])
```

```
Collecting torchinfo
  Downloading torchinfo-1.6.5-py3-none-any.whl (21 kB)
Installing collected packages: torchinfo
Successfully installed torchinfo-1.6.5
```

[61]:
```
================================================================================
==========
```

```
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
TinyVGG                                   --                        --
 Sequential: 1-1                         [1, 10, 30, 30]           --
     Conv2d: 2-1                        [1, 10, 62, 62]           280
     ReLU: 2-2                          [1, 10, 62, 62]           --
     Conv2d: 2-3                        [1, 10, 60, 60]           910
     ReLU: 2-4                          [1, 10, 60, 60]           --
     MaxPool2d: 2-5                     [1, 10, 30, 30]           --
 Sequential: 1-2                         [1, 10, 13, 13]           --
     Conv2d: 2-6                        [1, 10, 28, 28]           910
     ReLU: 2-7                          [1, 10, 28, 28]           --
     Conv2d: 2-8                        [1, 10, 26, 26]           910
     ReLU: 2-9                          [1, 10, 26, 26]           --
     MaxPool2d: 2-10                    [1, 10, 13, 13]           --
 Sequential: 1-3                         [1, 3]                    --
     Flatten: 2-11                      [1, 1690]                 --
     Linear: 2-12                       [1, 3]                    5,073
================================================================================
==========
Total params: 8,083
Trainable params: 8,083
Non-trainable params: 0
Total mult-adds (M): 5.69
================================================================================
==========
Input size (MB): 0.05
Forward/backward pass size (MB): 0.71
Params size (MB): 0.03
Estimated Total Size (MB): 0.79
================================================================================
==========
```

1.9.5 7.5 Create train and test loops functions

- `train_step()` - takes in a model and dataloader and trains the model on the dataloader.
- `test_step()` - takes in a model and dataloader and evaluates the model on the dataloader.

[62]:
```python
# Create train_step()
def train_step(model: torch.nn.Module,
               dataloader: torch.utils.data.DataLoader,
               loss_fn: torch.nn.Module,
               optimizer:torch.optim.Optimizer,
               device=device):
  # Put the model in train mode
  model.train()
```

```
  # Setup train loss and train accuracy values
  train_loss, train_acc = 0, 0

  # Loop through data loader data batches
  for batch, (X, y) in enumerate(dataloader):
    # Send data to the target device
    X, y = X.to(device), y.to(device)

    # 1. Forward pass
    y_pred = model(X) # output model logits

    # 2. Calculate the loss
    loss = loss_fn(y_pred, y)
    train_loss += loss.item()

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backward
    loss.backward()

    # 5. Optimizer step
    optimizer.step()

    # Calculate accuracy metric
    y_pred_class = torch.argmax(torch.softmax(y_pred, dim=1), dim=1)
    train_acc += (y_pred_class==y).sum().item()/len(y_pred)

  # Adjust metrics to get average loss and accuracy per batch
  train_loss = train_loss / len(dataloader)
  train_acc = train_acc / len(dataloader)
  return train_loss, train_acc
```

[63]:

```
# Create a test step
def test_step(model: torch.nn.Module,
              dataloader: torch.utils.data.DataLoader,
              loss_fn: torch.nn.Module,
              device=device):
  # Put model in eval mode
  model.eval()

  # Setup test loss and test accuracy values
  test_loss, test_acc = 0,  0

  # Turn on inference mode
  with torch.inference_mode():
```

```
    # Loop through DataLoader batches
    for batch, (X, y) in enumerate(dataloader):
      # Send data to the target device
      X, y = X.to(device), y.to(device)

      # 1. Forward pass
      test_pred_logits = model(X)

      # 2. Calculate the loss
      loss = loss_fn(test_pred_logits, y)
      test_loss += loss.item()

      # Calculate the accuracy
      test_pred_labels = test_pred_logits.argmax(dim=1)
      test_acc += ((test_pred_labels == y).sum().item()/len(test_pred_labels))

  # Adjust metrics to get average loss and accuracy per batch
  test_loss = test_loss / len(dataloader)
  test_acc = test_acc / len(dataloader)
  return test_loss, test_acc
```

1.9.6 7.6 Creating a `train()` function to combine `train_step()` and `test_step()`

[64]:

```
from tqdm.auto import tqdm

# 1. Create a train function that takes in various model parameters + optimizer␣
 ↪+ dataloaders + loss function
def train(model: torch.nn.Module,
          train_dataloader,
          test_dataloader,
          optimizer,
          loss_fn: torch.nn.Module = nn.CrossEntropyLoss(),
          epochs: int = 5,
          device=device):

  # 2. Create empty results dictionary
  results = {"train_loss": [],
             "train_acc": [],
             "test_loss": [],
             "test_acc": []}

  # 3. 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)

    # 4. Print out what's happening
    print(f"Epoch: {epoch} | Train loss: {train_loss:.4f} | Train acc:␣
↪{train_acc:.4f} | Test loss: {test_loss:.4f} | Test acc: {test_acc:.4f}")

    # 5. 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)

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

1.9.7 7.7 Train and evaluate model 0

[65]:

```
# Set random seeds
torch.manual_seed(42)
torch.cuda.manual_seed(42)

# Set number of epochs
NUM_EPOCHS = 5

# Recreate an instance of TinyVGG
model_0 = TinyVGG(input_shape=3, # number of color channels of our target images
                  hidden_units=10,
                  output_shape=len(train_data.classes)).to(device)

# Setup loss function and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model_0.parameters(),
                             lr=0.001)

# Start the timer
from timeit import default_timer as timer
start_time = timer()

# Train model_0
model_0_results = train(model=model_0,
                        train_dataloader=train_dataloader_simple,
                        test_dataloader=test_dataloader_simple,
                        optimizer=optimizer,
```

```
                        loss_fn=loss_fn,
                        epochs=NUM_EPOCHS)

# End the timer and print out how long it took
end_time = timer()
print(f"Total training time: {end_time-start_time:.3f} seconds")
```

```
  0%|          | 0/5 [00:00<?, ?it/s]

Epoch: 0 | Train loss: 1.1063 | Train acc: 0.3047 | Test loss: 1.0983 | Test
acc: 0.3116
Epoch: 1 | Train loss: 1.0995 | Train acc: 0.3320 | Test loss: 1.0698 | Test
acc: 0.5417
Epoch: 2 | Train loss: 1.0862 | Train acc: 0.4922 | Test loss: 1.0799 | Test
acc: 0.5227
Epoch: 3 | Train loss: 1.0826 | Train acc: 0.4102 | Test loss: 1.0598 | Test
acc: 0.5729
Epoch: 4 | Train loss: 1.0631 | Train acc: 0.4141 | Test loss: 1.0612 | Test
acc: 0.5540
Total training time: 10.171 seconds
```

[66]:
```
model_0_results
```

[66]:
```
{'test_acc': [0.3115530303030303,
  0.5416666666666666,
  0.5227272727272728,
  0.5729166666666666,
  0.5539772727272728],
 'test_loss': [1.098314603169759,
  1.069807728131612,
  1.0799124638239543,
  1.0598418315251668,
  1.0611690680185955],
 'train_acc': [0.3046875, 0.33203125, 0.4921875, 0.41015625, 0.4140625],
 'train_loss': [1.1063424944877625,
  1.0995023548603058,
  1.0862251669168472,
  1.0826159715652466,
  1.0630627423524857]}
```

1.9.8 7.8 Plot the loss curves of Model 0

A **loss curve** is a way of tracking your model's progress over time.

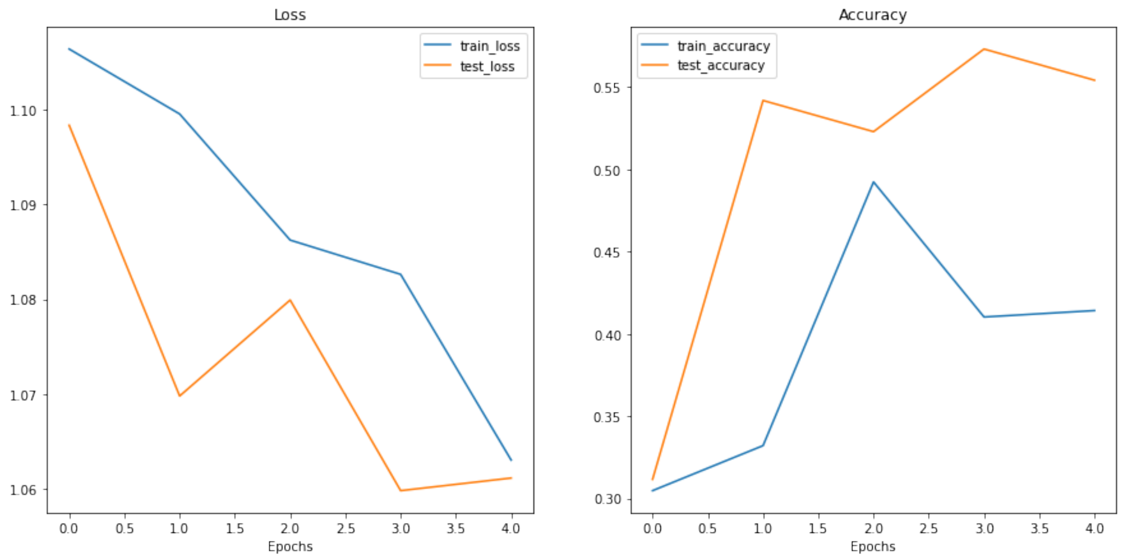
A good guide for different loss curves can be seen here: https://developers.google.com/machine-learning/testing-debugging/metrics/interpretic

[67]:
```
# Get the model_0_results keys
model_0_results.keys()
```

[67]:
```
dict_keys(['train_loss', 'train_acc', 'test_loss', 'test_acc'])
```

[68]:
```
def plot_loss_curves(results: Dict[str, List[float]]):
  """Plots training curves of a results dictionary."""
  # Get the loss values of the results dictionary(training and test)
  loss = results["train_loss"]
  test_loss = results["test_loss"]

  # Get the accuracy values of the results dictionary (training and test)
  accuracy = results["train_acc"]
  test_accuracy = results["test_acc"]

  # Figure out how mnay epochs there were
  epochs = range(len(results["train_loss"]))

  # Setup a plot
  plt.figure(figsize=(15, 7))

  # Plot the loss
  plt.subplot(1, 2, 1)
  plt.plot(epochs, loss, label="train_loss")
  plt.plot(epochs, test_loss, label="test_loss")
  plt.title("Loss")
  plt.xlabel("Epochs")
  plt.legend()

  # Plot the accuracy
  plt.subplot(1, 2, 2)
  plt.plot(epochs, accuracy, label="train_accuracy")
  plt.plot(epochs, test_accuracy, label="test_accuracy")
  plt.title("Accuracy")
  plt.xlabel("Epochs")
  plt.legend();
```

[69]:
```
plot_loss_curves(model_0_results)
```



1.10 8. What should an ideal loss curve look like?

https://developers.google.com/machine-learning/testing-debugging/metrics/interpretic

A loss curve is one of the most helpful ways to troubleshoot a model.

1.11 9. Model 1: TinyVGG with Data Augmentation

Now let's try another modelling experiment this time using the same model as before with some data augmentation.

1.11.1 9.1 Create transform with data augmentation

[70]:
```
# Create training transform with TriviailAugment
from torchvision import transforms
train_transform_trivial = transforms.Compose([
                                               transforms.Resize(size=(64, 64)),
                                               transforms.
 ↪TrivialAugmentWide(num_magnitude_bins=31),
                                               transforms.ToTensor()
])

test_transform_simple = transforms.Compose([
                                            transforms.Resize(size=(64, 64)),
                                            transforms.ToTensor()
])
```

1.11.2 9.2 Create train and test Dataset's and DataLoader's with data augmentation

[71]:
```
# Turn image folders into Datasets
from torchvision import datasets
train_data_augmented = datasets.ImageFolder(root=train_dir,
                                            transform=train_transform_trivial)
test_data_simple = datasets.ImageFolder(root=test_dir,
                                        transform=test_transform_simple)
```

[72]:
```
# Turn our Datasets into DataLoaders
import os
from torch.utils.data import DataLoader
BATCH_SIZE = 32
NUM_WORKERS = os.cpu_count()

torch.manual_seed(42)
train_dataloader_augmented = DataLoader(dataset=train_data_augmented,
                                        batch_size=BATCH_SIZE,
                                        shuffle=True,
                                        num_workers=NUM_WORKERS)

test_dataloader_simple = DataLoader(dataset=test_data_simple,
                                    batch_size=BATCH_SIZE,
                                    shuffle=False,
                                    num_workers=NUM_WORKERS)
```

1.11.3 9.3 Construct and train model 1

This time we'll be using the same model architecture except this time we've augmented the training data.

[73]:
```
# Create model_1 and send it to the target device
torch.manual_seed(42)
model_1 = TinyVGG(input_shape=3,
                  hidden_units=10,
                  output_shape=len(train_data_augmented.classes)).to(device)

model_1
```

[73]:
```
TinyVGG(
  (conv_block_1): Sequential(
    (0): Conv2d(3, 10, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
```

```
  (conv_block_2): Sequential(
    (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=1690, out_features=3, bias=True)
  )
)
```

Wonderful! Now we've a model and dataloaders, let's create a loss function and an optimizer and call upon our `train()` function to train and evaluate our model.

[74]:
```python
# Set random seeds
torch.manual_seed(42)
torch.cuda.manual_seed(42)

# Set the number of epochs
NUM_EPOCHS = 5

# Setup loss function
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model_1.parameters(),
                             lr=0.001)

# Start the timer
from timeit import default_timer as timer
start_time = timer()

# Train model 1
model_1_results = train(model=model_1,
                        train_dataloader=train_dataloader_augmented,
                        test_dataloader=test_dataloader_simple,
                        optimizer=optimizer,
                        loss_fn=loss_fn,
                        epochs=NUM_EPOCHS,
                        device=device)

# End the timer and print out how long it took
end_time = timer()
print(f"Total training time for model_1: {end_time-start_time:.3f} seconds")
```

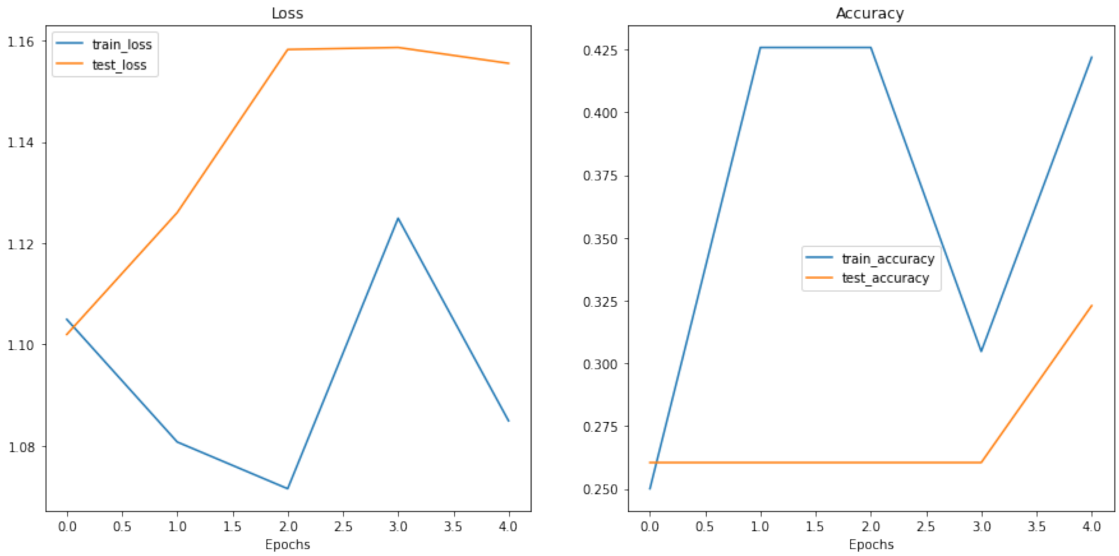
```
  0%|          | 0/5 [00:00<?, ?it/s]
```

```
Epoch: 0 | Train loss: 1.1049 | Train acc: 0.2500 | Test loss: 1.1019 | Test
acc: 0.2604
Epoch: 1 | Train loss: 1.0807 | Train acc: 0.4258 | Test loss: 1.1260 | Test
acc: 0.2604
Epoch: 2 | Train loss: 1.0715 | Train acc: 0.4258 | Test loss: 1.1582 | Test
acc: 0.2604
Epoch: 3 | Train loss: 1.1249 | Train acc: 0.3047 | Test loss: 1.1586 | Test
acc: 0.2604
Epoch: 4 | Train loss: 1.0849 | Train acc: 0.4219 | Test loss: 1.1555 | Test
acc: 0.3229
Total training time for model_1: 10.624 seconds
```

1.11.4 9.4 Plot the loss curves of model 1

A loss curve helps you evaluate your models performance overtime.

[75]:
```
plot_loss_curves(model_1_results)
```



1.12 10. Compare model results

After evaluating our modelling experiments on their own, it's important to compare them to each other.

There's a few different ways to do this: 1. Hard coding (what we're doing) 2. PyTorch + Tensorboard - https://pytorch.org/docs/stable/tensorboard.html 3. Weights & Biases - https://wandb.ai/site/experiment-tracking 4. MLFlow - https://mlflow.org/

[76]:
```
import pandas as pd
model_0_df = pd.DataFrame(model_0_results)
model_1_df = pd.DataFrame(model_1_results)
```

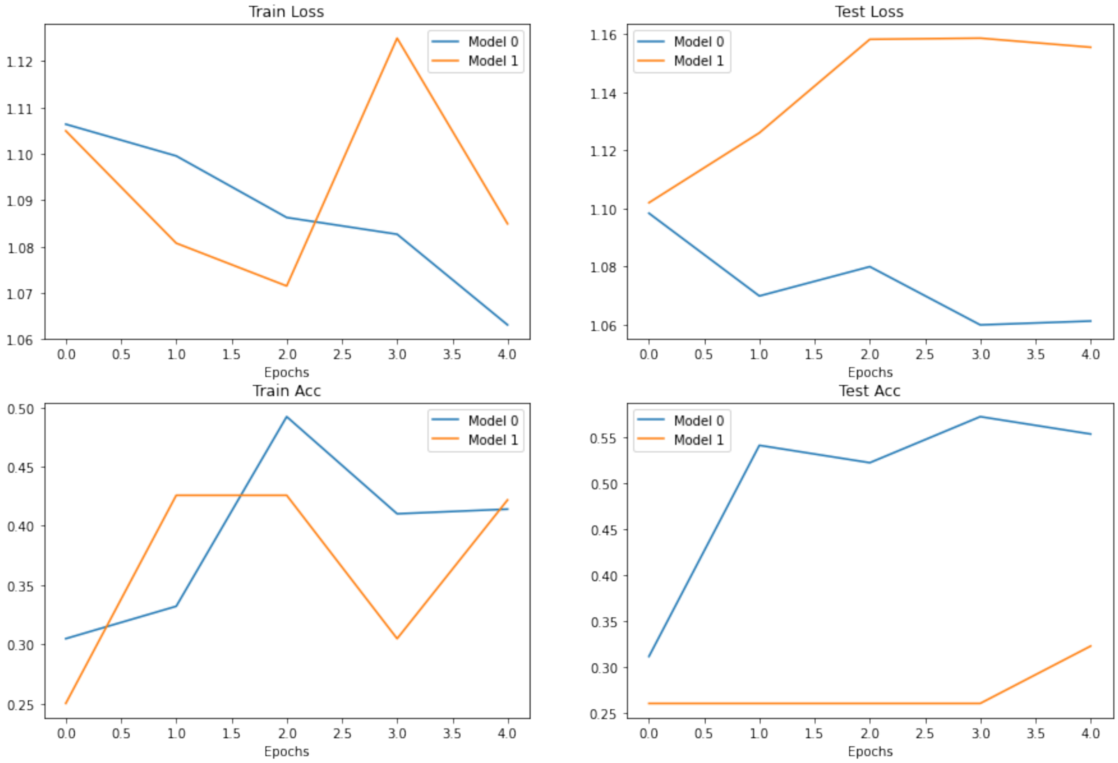
```
model_0_df
```

[76]:

```
   train_loss  train_acc  test_loss  test_acc
0    1.106342   0.304688   1.098315  0.311553
1    1.099502   0.332031   1.069808  0.541667
2    1.086225   0.492188   1.079912  0.522727
3    1.082616   0.410156   1.059842  0.572917
4    1.063063   0.414062   1.061169  0.553977
```

[83]:

```
# Setup a plot
plt.figure(figsize=(15, 10))

# Get number of epochs
epochs = range(len(model_0_df))

# Plot train loss
plt.subplot(2, 2, 1)
plt.plot(epochs, model_0_df["train_loss"], label="Model 0")
plt.plot(epochs, model_1_df["train_loss"], label="Model 1")
plt.title("Train Loss")
plt.xlabel("Epochs")
plt.legend()

# Plot test loss
plt.subplot(2, 2, 2)
plt.plot(epochs, model_0_df["test_loss"], label="Model 0")
plt.plot(epochs, model_1_df["test_loss"], label="Model 1")
plt.title("Test Loss")
plt.xlabel("Epochs")
plt.legend()

# Plot train accuracy
plt.subplot(2, 2, 3)
plt.plot(epochs, model_0_df["train_acc"], label="Model 0")
plt.plot(epochs, model_1_df["train_acc"], label="Model 1")
plt.title("Train Acc")
plt.xlabel("Epochs")
plt.legend()

# Plot test accuracy
plt.subplot(2, 2, 4)
plt.plot(epochs, model_0_df["test_acc"], label="Model 0")
plt.plot(epochs, model_1_df["test_acc"], label="Model 1")
plt.title("Test Acc")
plt.xlabel("Epochs")
plt.legend();
```



1.13 11. Making a prediction on a custom image

Although we've trained a model on custom data… how do you make a prediction on a sample/image that's not in either training or testing dataset.

[84]:
```python
# Download custom image
import requests

# Setup custom image path
custom_image_path = data_path / "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…
```

1.13.1 11.1 Loading in a custom image with PyTorch

We have to make sure our custom image is in the same format as the data our model was trained on.

- In tensor form with datatype (torch.float32)
- Of shape 64x64x3
- On the right device

We can read an image into PyTorch using - https://pytorch.org/vision/stable/generated/torchvision.io.read_image

[86]:
```
custom_image_path
```

[86]:
```
PosixPath('data/04-pizza-dad.jpeg')
```

[90]:
```
import torchvision

# Read in custom image
custom_image_uint8 = torchvision.io.read_image(str(custom_image_path))
print(f"Custom image tensor:\n {custom_image_uint8}")
print(f"Custom image shape: {custom_image_uint8.shape}")
print(f"Custom image datatype: {custom_image_uint8.dtype}")
```

```
Custom image tensor:
 tensor([[[154, 173, 181,  …,  21,  18,  14],
         [146, 165, 181,  …,  21,  18,  15],
         [124, 146, 172,  …,  18,  17,  15],
         …,
         [ 72,  59,  45,  …, 152, 150, 148],
         [ 64,  55,  41,  …, 150, 147, 144],
         [ 64,  60,  46,  …, 149, 146, 143]],

        [[171, 190, 193,  …,  22,  19,  15],
         [163, 182, 193,  …,  22,  19,  16],
         [141, 163, 184,  …,  19,  18,  16],
         …,
         [ 55,  42,  28,  …, 107, 104, 103],
         [ 47,  38,  24,  …, 108, 104, 102],
         [ 47,  43,  29,  …, 107, 104, 101]],

        [[119, 138, 147,  …,  17,  14,  10],
         [111, 130, 145,  …,  17,  14,  11],
         [ 87, 111, 136,  …,  14,  13,  11],
         …,
         [ 35,  22,   8,  …,  52,  52,  48],
         [ 27,  18,   4,  …,  50,  49,  44],
         [ 27,  23,   9,  …,  49,  46,  43]]], dtype=torch.uint8)
Custom image shape: torch.Size([3, 4032, 3024])
Custom image datatype: torch.uint8
```

[89]:

```
plt.imshow(custom_image_uint8.permute(1, 2, 0));
```

[89]:

```
<matplotlib.image.AxesImage at 0x7f29f09385d0>
```

1.13.2 11.2 Making a prediction on a custom image with a trained PyTorch model

[91]:

```
# Try to make a prediction on an image in uint8 format
model_1.eval()
with torch.inference_mode():
  model_1(custom_image_uint8.to(device))
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-91-0abf52e4b774> in <module>()
      2 model_1.eval()
      3 with torch.inference_mode():
----> 4   model_1(custom_image_uint8.to(device))

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
```

```
   1112         full_backward_hooks, non_full_backward_hooks = [], []

<ipython-input-57-16bcea8c1446> in forward(self, x)
     47
     48   def forward(self, x):
---> 49     x = self.conv_block_1(x)
     50     # print(x.shape)
     51     x = self.conv_block_2(x)

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
   1112         full_backward_hooks, non_full_backward_hooks = [], []

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/container.py in␣
 ↪forward(self, input)
    139     def forward(self, input):
    140         for module in self:
--> 141             input = module(input)
    142         return input
    143

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
   1112         full_backward_hooks, non_full_backward_hooks = [], []

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py in forward(self␣
 ↪input)
    445
    446     def forward(self, input: Tensor) -> Tensor:
--> 447         return self._conv_forward(input, self.weight, self.bias)
    448
    449 class Conv3d(_ConvNd):

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py in␣
 ↪_conv_forward(self, input, weight, bias)
    442                             _pair(0), self.dilation, self.groups)
    443         return F.conv2d(input, weight, bias, self.stride,
```

```
--> 444                         self.padding, self.dilation, self.groups)

    445
    446     def forward(self, input: Tensor) -> Tensor:

RuntimeError: Input type (torch.cuda.ByteTensor) and weight type (torch.cuda.
 ↪FloatTensor) should be the same
```

[96]:

```
# Load in the custom image and convert to torch.float32
custom_image = torchvision.io.read_image(str(custom_image_path)).type(torch.
 ↪float32) / 255.
custom_image
```

[96]:

```
tensor([[[0.6039, 0.6784, 0.7098,  …, 0.0824, 0.0706, 0.0549],
         [0.5725, 0.6471, 0.7098,  …, 0.0824, 0.0706, 0.0588],
         [0.4863, 0.5725, 0.6745,  …, 0.0706, 0.0667, 0.0588],
         …,
         [0.2824, 0.2314, 0.1765,  …, 0.5961, 0.5882, 0.5804],
         [0.2510, 0.2157, 0.1608,  …, 0.5882, 0.5765, 0.5647],
         [0.2510, 0.2353, 0.1804,  …, 0.5843, 0.5725, 0.5608]],

        [[0.6706, 0.7451, 0.7569,  …, 0.0863, 0.0745, 0.0588],
         [0.6392, 0.7137, 0.7569,  …, 0.0863, 0.0745, 0.0627],
         [0.5529, 0.6392, 0.7216,  …, 0.0745, 0.0706, 0.0627],
         …,
         [0.2157, 0.1647, 0.1098,  …, 0.4196, 0.4078, 0.4039],
         [0.1843, 0.1490, 0.0941,  …, 0.4235, 0.4078, 0.4000],
         [0.1843, 0.1686, 0.1137,  …, 0.4196, 0.4078, 0.3961]],

        [[0.4667, 0.5412, 0.5765,  …, 0.0667, 0.0549, 0.0392],
         [0.4353, 0.5098, 0.5686,  …, 0.0667, 0.0549, 0.0431],
         [0.3412, 0.4353, 0.5333,  …, 0.0549, 0.0510, 0.0431],
         …,
         [0.1373, 0.0863, 0.0314,  …, 0.2039, 0.2039, 0.1882],
         [0.1059, 0.0706, 0.0157,  …, 0.1961, 0.1922, 0.1725],
         [0.1059, 0.0902, 0.0353,  …, 0.1922, 0.1804, 0.1686]]])
```

[93]:

```
model_1.eval()
with torch.inference_mode():
  model_1(custom_image.to(device))
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-93-8ea0521c5284> in <module>()
      1 model_1.eval()
      2 with torch.inference_mode():
----> 3   model_1(custom_image.to(device))
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
   1112         full_backward_hooks, non_full_backward_hooks = [], []

<ipython-input-57-16bcea8c1446> in forward(self, x)
     51     x = self.conv_block_2(x)
     52     # print(x.shape)
---> 53     x = self.classifier(x)
     54     # print(x.shape)
     55     return x

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
   1112         full_backward_hooks, non_full_backward_hooks = [], []

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/container.py in␣
 ↪forward(self, input)
    139     def forward(self, input):
    140         for module in self:
--> 141             input = module(input)
    142         return input
    143 

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
   1112         full_backward_hooks, non_full_backward_hooks = [], []

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/linear.py in␣
 ↪forward(self, input)
    101 
    102     def forward(self, input: Tensor) -> Tensor:
--> 103         return F.linear(input, self.weight, self.bias)
```

```
    104
    105     def extra_repr(self) -> str:

RuntimeError: mat1 and mat2 shapes cannot be multiplied (10x756765 and 1690x3)
```

[105]:

```python
# Create transform pipeline to resize image
from torchvision import transforms
custom_image_transform = transforms.Compose([
                                             transforms.Resize(size=(64, 64))
])

# Transfrom target image
custom_image_transformed = custom_image_transform(custom_image)

# Print out the shapes
print(f"Original shape: {custom_image.shape}")
print(f"Transformed shape: {custom_image_transformed.shape}")
```

```
Original shape: torch.Size([3, 4032, 3024])
Transformed shape: torch.Size([3, 64, 64])
```

[106]:

```python
plt.imshow(custom_image_transformed.permute(1, 2, 0))
```

[106]:

```
<matplotlib.image.AxesImage at 0x7f29f0d467d0>
```

```
[107]: # This will error: image not on right device
model_1.eval()
with torch.inference_mode():
  custom_image_pred = model_1(custom_image_transformed)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-107-1fdc4ee1b1a3> in <module>()
      1 model_1.eval()
      2 with torch.inference_mode():
----> 3   custom_image_pred = model_1(custom_image_transformed)

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
   1112         full_backward_hooks, non_full_backward_hooks = [], []

<ipython-input-57-16bcea8c1446> in forward(self, x)
     47
     48   def forward(self, x):
---> 49     x = self.conv_block_1(x)
     50     # print(x.shape)
     51     x = self.conv_block_2(x)

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
   1112         full_backward_hooks, non_full_backward_hooks = [], []

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/container.py in␣
 ↪forward(self, input)
    139     def forward(self, input):
    140         for module in self:
--> 141             input = module(input)
    142         return input
    143

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
```

```
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
   1112         full_backward_hooks, non_full_backward_hooks = [], []

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py in forward(self ␣
 ↪input)
    445
    446     def forward(self, input: Tensor) -> Tensor:
--> 447         return self._conv_forward(input, self.weight, self.bias)
    448
    449 class Conv3d(_ConvNd):

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py in␣
 ↪_conv_forward(self, input, weight, bias)
    442                             _pair(0), self.dilation, self.groups)
    443         return F.conv2d(input, weight, bias, self.stride,
--> 444                         self.padding, self.dilation, self.groups)

    445
    446     def forward(self, input: Tensor) -> Tensor:

RuntimeError: Expected all tensors to be on the same device, but found at least␣
 ↪two devices, cpu and cuda:0! (when checking argument for argument weight in␣
 ↪method wrapper___slow_conv2d_forward)
```

[108]:

```
# This will error: no batch size
model_1.eval()
with torch.inference_mode():
  custom_image_pred = model_1(custom_image_transformed.to(device))
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-108-f83d7bb52387> in <module>()
      1 model_1.eval()
      2 with torch.inference_mode():
----> 3   custom_image_pred = model_1(custom_image_transformed.to(device))

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
```

```
   1112         full_backward_hooks, non_full_backward_hooks = [], []

<ipython-input-57-16bcea8c1446> in forward(self, x)
     51     x = self.conv_block_2(x)
     52     # print(x.shape)
---> 53     x = self.classifier(x)
     54     # print(x.shape)
     55     return x

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
   1112         full_backward_hooks, non_full_backward_hooks = [], []

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/container.py in␣
 ↪forward(self, input)
    139     def forward(self, input):
    140         for module in self:
--> 141             input = module(input)
    142         return input
    143 

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *input, **kwargs)
   1108         if not (self._backward_hooks or self._forward_hooks or self.
 ↪_forward_pre_hooks or _global_backward_hooks
   1109                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110             return forward_call(*input, **kwargs)
   1111         # Do not call functions when jit is used
   1112         full_backward_hooks, non_full_backward_hooks = [], []

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/linear.py in␣
 ↪forward(self, input)
    101 
    102     def forward(self, input: Tensor) -> Tensor:
--> 103         return F.linear(input, self.weight, self.bias)
    104 
    105     def extra_repr(self) -> str:

RuntimeError: mat1 and mat2 shapes cannot be multiplied (10x169 and 1690x3)
```

[110]:

```
custom_image_transformed.shape, custom_image_transformed.unsqueeze(0).shape
```

[110]:
```
(torch.Size([3, 64, 64]), torch.Size([1, 3, 64, 64]))
```

[112]:
```
# This should this work? (added a batch size...)
model_1.eval()
with torch.inference_mode():
  custom_image_pred = model_1(custom_image_transformed.unsqueeze(0).to(device))
custom_image_pred
```

[112]:
```
tensor([[ 0.0701,  0.0501, -0.2080]], device='cuda:0')
```

Note, to make a prediction on a custom image we had to: * Load the image and turn it into a tensor * Make sure the image was the same datatype as the model (torch.float32) * Make sure the image was the same shape as the data the model was trained on (3, 64, 64) with a batch size... (1, 3, 64, 64) * Make sure the image was on the same device as our model

[115]:
```
# Convert logits -> prediction probabilities
custom_image_pred_probs = torch.softmax(custom_image_pred, dim=1)
custom_image_pred_probs
```

[115]:
```
tensor([[0.3653, 0.3581, 0.2766]], device='cuda:0')
```

[121]:
```
# Convert prediction probabilities -> prediction labels
custom_image_pred_label = torch.argmax(custom_image_pred_probs, dim=1).cpu()
custom_image_pred_label
```

[121]:
```
tensor([0])
```

[123]:
```
class_names[custom_image_pred_label]
```

[123]:
```
'pizza'
```

1.13.3 11.3 Putting custom image prediction together: building a function

Ideal outcome:

A function where we pass an image path to and have our model predict on that image and plot the image + prediction.

[126]:
```
def pred_and_plot_image(model: torch.nn.Module,
                        image_path: str,
                        class_names: List[str] = None,
                        transform=None,
                        device=device):
  """Makes a prediction on a target image with a trained model and plots the␣
↪image and prediction."""
  # Load in the image
  target_image = torchvision.io.read_image(str(image_path)).type(torch.float32)
```

```
  # Divide the image pixel values by 255 to get them between [0, 1]
  target_image = target_image / 255.

  # Transform if necessary
  if transform:
    target_image = transform(target_image)

  # Make sure the model is on the target device
  model.to(device)

  # Turn on eval/inference mode and make a prediction
  model.eval()
  with torch.inference_mode():
    # Add an extra dimension to the image (this is the batch dimension, e.g.␣
↪our model will predict on batches of 1x image)
    target_image = target_image.unsqueeze(0)

    # Make a prediction on the image with an extra dimension
    target_image_pred = model(target_image.to(device)) # make sure the target␣
↪image is on the right device

  # Convert logits -> prediction probabilities
  target_image_pred_probs = torch.softmax(target_image_pred, dim=1)

  # Convert predction probabilities -> prediction labels
  target_image_pred_label = torch.argmax(target_image_pred_probs, dim=1)

  # Plot the image alongside the prediction and prediction probability
  plt.imshow(target_image.squeeze().permute(1, 2, 0)) # remove batch dimension␣
↪and rearrange shape to be HWC
  if class_names:
    title = f"Pred: {class_names[target_image_pred_label.cpu()]} | Prob:␣
↪{target_image_pred_probs.max().cpu():.3f}"
  else:
    title = f"Pred: {target_image_pred_label} | Prob: {target_image_pred_probs.
↪max().cpu():.3f}"
  plt.title(title)
  plt.axis(False)
```

[131]:

```
# Pred on our custom image
pred_and_plot_image(model=m,odel_1
                    image_path=custom_image_path,
                    class_names=class_names,
                    transform=custom_image_transform,
                    device=device)
```

1.14 Exercises

For all exercises and extra-curriculum, see here: https://www.learnpytorch.io/04_pytorch_custom_datasets/#exe