michael bipes

pytorch custom datasets

data (data_path)

pizza_steak_sushi (image_path)

Github
- test (train_dir)
  - pizza, steak, sushi
- train (test_dir)
- ...

0: import pytorch and setup device agnostic code

```
import torch... from torch import nn
device = "cuda" if torch.cuda.is_available() else "cpu"
```

1: get data

```
import requests... import zipfile ... from pathlib import Path ...
data_path = Path("data/") ... image_path = data_path / "pizza_steak_sushi"
if image_path.is_dir():
else: image_path.mkdir(parents = True, exist_ole = True)
with open(data_path / "pizza_steak_sushi.zip", "wb") as f:
    request = request.get(" github raw link ")
    f.write(request.content)
with zipfile.ZipFile(data_path / "pizza_steak_sushi.zip", "r") as zip_ref:
    zip_ref.extractall(image_path)
train_dir = image_path / "train" ... test_dir = image_path / "test"
```

indentation indicators

3: transforming data

```
import torch... from torch.utils.data import DataLoader
from torchvision import datasets, transforms
data_transform = transform.Compose(transform.Resize(size=(64,64)), transform.RandomHorizontalFlip(p=0.5),
                 transform.ToTensor()])
```

(for model 1, insert train augmentation for test data)

4: option 1, loading image data using image folder

```
from torchvision import datasets
train_data = datasets.ImageFolder(root = train_dir, transform = data_transform, target_transform = None)
test_data = datasets.ImageFolder(root = test_dir, transform = data_transform)
class_names = train_data.classes
from torch.utils.data import DataLoader
train_dataloader = DataLoader(dataset = train_data, batch_size = 1, num_workers = 1, shuffle = True)
test " " test " " " False
```

5: option 2, loading image data with custom dataset

```
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
target_directory = train_dir
class_names = sorted [ entry.name for entry in list(os.scandir (image_path / "train")) ]
def find_classes (directory: str) -> Tuple [List [str], Dict [str, int]]:
    classes = sorted (entry.name for entry in os.scandir (directory) if entry.is_dir())
    if not classes: raise FileNotFoundError (f"Couldn't find any classes in {directory}.")
    class_to_idx = { cls_name: i for i, cls_name in enumerate (classes) }
    return classes, class_to_idx
from torch.utils.data import Dataset
class ImageFolderCustom (Dataset):
    def __init__ (self, targ_dir: str, transform=None) -> None:
        self.paths: list (pathlib.Path (targ_dir).glob ("*/*.jpg"))
        self.transform = transform
        self.classes, self.class_to_idx = find_classes (targ_dir)
    def load_image ( self, index: int) -> Image.Image:
        image_path = self.paths [index] ... return Image.open (image_path)
    def __len__ (self) -> int:
        return len ( self.paths)
    def __getitem__ (self, index: int) -> Tuple [ torch.Tensor, int ]:
        img = self.load_image (index)
        class_name = self.paths [index].parent.name
        class_idx = self.class_to_idx (class_name)
    if self.transform: return self.transform (img), class_idx
    else return img, class_idx
train_transforms = data_transform
test_transforms = data_transform ( — random flip, don't augment test data, view pil)
train_data_custom = ImageFolderCustom ( targ_dir = train_dir, transform = train_transforms)
test " " " test " test "
from torch.utils.data import DataLoader
train_dataloader_custom = DataLoader ( dataset = train_data_custom, batch_size = 1, num_workers = 0, shuffle = True)
test " " test " " False
```

returns sample of data, data and label (X, y)

pytorch custom datasets continued...

7: model0

pytorch computer vision Fashion MNIST Model V2

```
try: import torchinfo
except: !pip install torchinfo ... import torchinfo
from torchinfo import summary
```

super cool!:

```
summary(model0, input_size=[1,3,64,64])
```

skip 8 and 9

```
def train_step(model: torch.nn.Module, dataloader = torch.utils.data.DataLoader, loss_fn: torch.nn.Module,
               optimizer: torch.optim.Optimizer):
    model.train()
    train_loss, train_acc = 0,0
    for batch, (X,y) in enumerate(dataloader):
        X,y = X.to(device), y.to(device)
        y_pred = model(X)
        loss = loss_fn(y_pred, y)
        train_loss += loss.item()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        y_pred_class = torch.argmax(torch.softmax(y_pred, dim=1), dim=1)
        train_acc += (y_pred_class == y).sum().item()/len(y_pred)
    train_loss = train_loss /len(dataloader)
    train_acc = train_acc /len(dataloader)
    return train_loss, train_acc
def test_step(model: ↑, dataloader: ↑, loss_fn: ↑)
    model.eval()
    test_loss, test_acc = 0,0
    with torch.inference_mode():
        for batch, (X,y) in enumerate(dataloader):
            X,y = X.to(device), y.to(device)
            test_pred_logits = model(X)
            loss = loss_fn(test_pred_logits, y)
            test_loss += loss.item()
```

```
            test_pred_labels = test_pred_logits.argmax(dim=1)
            test_acc += ((test_pred_labels == y).sum().item()/len(test_pred_labels))
    test_loss = test_loss / len(dataloader)
    test_acc = test_acc / len(dataloader)
    return test_loss, test_acc
from tqdm.auto import tqdm
def train(model: ↑, train_dataloader: ↑, test_dataloader: ↑, optimizer: ↑,
          loss_fn: torch.nn.Module = nn.CrossEntropyLoss(), epochs: int=5)
    results = {"train_loss": [], "train_acc": [], "test_loss": [], "test_acc": []}
    for epoch in tqdm(range(epochs)):
        train_loss, train_acc = train_step(model=↑, dataloader=↑, loss_fn=↑, optimizer=↑)
        test_loss, test_acc = test_step(model=↑, dataloader=↑, loss_fn=↑)
        # print and update results (see colab)
torch.manual_seed(42) ... torch.cuda.manual_seed(42)
NUM_EPOCHS = 5
model0 = TinyVGG(input_shape=3, hidden_units=15, output_shape=len(train_data.classes)).to(device)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model0.parameters(), lr=0.001)
from timeit import default_timer as timer
start_time = timer()
model0_results = train(model=model0, train_dataloader=train_dataloader, test_dataloader=test_dataloader,
                       optimizer=optimizer, loss_fn=loss_fn, epochs=NUM_EPOCHS)
end_time = timer()
# end_time - start_time

## make prediction on a random image
import requests
custom_image_path = data_path / "img_name"
if not custom_image_path.is_file():
    with open(custom_image_path, "wb") as f:
        request = requests.get("github raw link")
        f.write(request.content)
else: # custom image path already exists
```

pytorch custom datasets continued...

```
import torchvision
custom_image = torchvision.io.read_image(str(custom_image_path)).type(torch.float32)
custom_image = custom_image / 255
custom_image_transform = transforms.Compose([ transforms.Resize((64, 64)), ])
from matplotlib import pyplot as plt
def pred_and_plot_image(model: torch.nn.Module, image_path: str, class_names: List[str] = None, transform=None,
                        device: torch.device = device):
    target_image = torchvision.io.read_image(str(image_path)).type(torch.float32)
    target_image = target_image / 255
    if transform: target_image = transform(target_image)
    model.to(device)
    model.eval()
    with torch.inference_mode():
        target_img = target_image.unsqueeze(dim=0)
        target_img_pred = model(target_image.to(device))
    target_image_pred_probs = torch.softmax(target_image_pred, dim=1)
    target_image_pred_label = torch.argmax(target_image_pred_probs, dim=1)
    plt.imshow(target_image.squeeze().permute(1,2,0)
    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}|
    plt.title(title) ... plt.axis(False);
pred_and_plot_image(model=model0, image_path=custom_image_path, class_names=class_names,
                    transform=custom_image_transform, device=device)
```

pred: pizza | prob: 0.351