pytorch experiment tracking

0: import pytorch and setup device agnostic code

```
import torch ... from torch import nn ... import torchvision ... from torchvision import transforms
import matplotlib.pyplot as plt
from torchinfo import summary  ] if it does not work, → !pip install -q torchinfo
from going_modular.going_modular import data_setup, engine  ] if does not work, ↓
  #! git clone https://github.com/mrdbourke/pytorch-deep-learning
  #! mv pytorch-deep-learning/going_modular .
  #! rm -rf pytorch-deep-learning
device = "cuda" if torch.cuda.is_available() else "cpu"
```

1: download_data function

```
import os ... import zipfile ... from pathlib import Path
def download_data (source: str, destination: str, remove_source: bool = True) -> Path:
    data_path = Path("data/") ... image_path = data_path / destination
    if image_path.is_dir():
    else:
        image_path.mkdir (parents=True, exist_ok=True)
        target_file = Path(source).name
        with open(data_path / target_file, "wb") as f:
            request = request.get(source) ... f.write(request.content)
        with zipfile.ZipFile (data_path / target_file, "r") as zip_ref:
            zip_ref.extractall (image_path)
        if remove_source:
            os.remove (data_path / target_file)
    return image_path

def set_seeds (seed: int=42):
    torch.manual_seed(seed) ... torch.cuda.manual_seed(seed)
```

6. create a helper function to build SummaryWriter() instances

```
from torch.utils.tensorboard import SummaryWriter

def create_writer(experiment_name: str, model_name: str, extra: str=None) -> torch.utils.tensorboard.writer.SummaryWriter():
    from datetime import datetime ... import os
    timestamp = datetime.now().strftime("%Y-%m-%d")
    if extra:
        log_dir = os.path.join("runs", timestamp, experiment_name, model_name, extra)
    else:
        log_dir = os.path.join("runs", timestamp, experiment_name, model_name)
    return SummaryWriter(log_dir=log_dir)
```

create_writer("data_10_percent", "effnetb2", "5_epochs") = SummaryWriter(log_dir="runs/2022-06-04/data_10_percent/effnetb2/5_epochs/")

#add writer parameter to train()

... engine.py +

```
if writer:
    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.close
else:
    pass
```

7: Set up a series of modelling experiments

```
data_10_percent_path = download_data(source="github raw link", destination="pizza_steak_sushi")
data_20_percent_path = download_data(source="github rawlink", destination="pizza_steak_sushi_20_percent")
train_dir_10_percent = data_10_percent_path / "train"
"  20  "  20  "  "
test_dir = data_10_percent_path / "test"
from torchvision import transforms
normalize = transforms.Normalize(mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225])
simple_transform = transforms.Compose([transforms.Resize((224,224)), transforms.ToTensor(), normalize])
Batch_size = 32
```

Pytorch experiment tracking continued...

```
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)
"            20

        20

out_features = len(class_names)
def create_effnetb0():
    weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT
    model = torchvision.models.efficientnet_b0(weights=weights).to(device)
    for param in model.features.parameters():    param.requires_grad = False
    set_seeds()
    model.classifier = nn.Sequential(nn.Dropout(p=0.2), nn.Linear(in_features=1280, out_features=
                                     outfeatures)).to(device)
    model.name = "effnetb0"
    return model
def create_effnetb2():
                              B2
                           b2



                                 0.3                    1480



    return model
effnetb0 = create_effnetb0()     [ summary(model=effnetb2, input_size=(32,3,224,224), col_names=
effnetb2 = create_effnetb2()     [ ["input_size", "output_size", "num_params", "trainable"], col_width=20,
                                                               row_settings=["var_names"])
num_epochs = [5,10]
models = ["effnetb0", "effnetb2"]
train_dataloaders = {"data_10_percent": train_dataloader_10_percent, "   20   ":   20   }
%%time
from going_modular.going_modular.utils import save_model
set_seeds(seed=42)
experiment_number = 0
```

```
for dataloader_name, train_dataloader in train_dataloaders.items():
    for epochs in num_epochs:
        for model_name in models:

            experiment_number += 1

            if model_name == "effnetb0": model = create_effnetb0()
            else: model = create_effnetb2()

            loss_fn = nn.CrossEntropyLoss()

            optimizer = torch.optim.Adam(params=model.parameters(), lr=0.001)

            train(model, train_dataloader, test_dataloader, optimizer, loss_fn, epochs, device,
                  create_writer(experiment_name=dataloader_name, model_name=model_name, extra=f"{epochs}
                                                                                              -epochs"))
            save_filepath = f"07_{model_name}_{dataloader_name}_{epochs}_epochs.pth"

            save_model(model=model, target_dir="models", model_name=save_filepath)
```

```
%load_ext tensorboard
% tensorboard --logdir runs
```
} to view tensorboard in google collab

```
! tensorboard dev upload --logdir runs \
    --name "07. Pytorch Experiment Tracking: FoodVision Mini model results" \
    --description "Comparing results of different model size, training data amount and training time."
```
to upload results to TensorBoard.dev

```
% load in best model and make predictions with it
```