Baseline

```
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt
import PIL


from torchsummary import summary


# From local helper files
from helper_evaluation import set_all_seeds, set_deterministic, compute_confusion_matrix
from helper_train import train_model
from helper_plotting import plot_training_loss, plot_accuracy, show_examples, plot_confusion_matrix
from helper_dataset import get_dataloaders_cifar10, UnNormalize


RANDOM_SEED = 123
BATCH_SIZE = 256
NUM_EPOCHS = 40
DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')


train_transforms = torchvision.transforms.Compose([
    torchvision.transforms.Resize((16, 16)),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                                    ])


test_transforms = torchvision.transforms.Compose([
    torchvision.transforms.Resize((16, 16)),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])


train_loader, valid_loader, test_loader = get_dataloaders_cifar10(
    batch_size=BATCH_SIZE,
    validation_fraction=0.1,
    train_transforms=train_transforms,
    test_transforms=test_transforms,
    num_workers=2)

# Checking the dataset
for images, labels in train_loader:
    print('Image batch dimensions:', images.shape)
    print('Image label dimensions:', labels.shape)
    print('Class labels of 10 examples:', labels[:10])
    break
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to data/cifar-10-python.tar.gz
100%  170498071/170498071 [00:02<00:00, 72251698.13it/s]
Extracting data/cifar-10-python.tar.gz to data
Image batch dimensions: torch.Size([256, 3, 16, 16])
Image label dimensions: torch.Size([256])
```

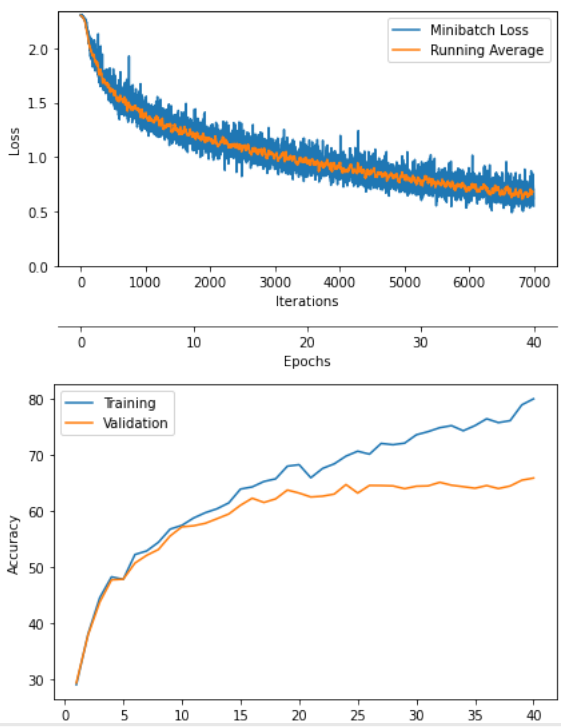
```
class CNN1(torch.nn.Module):
  def __init__(self, num_classes):
    super().__init__()
    self.features = torch.nn.Sequential(
            # Conv 1
            torch.nn.Conv2d(3, 16, kernel_size=3, padding="same"), # output 16 - 3 + 1 => 16
                        # , stride=4, padding=2),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2), # 16 / 2 => output 8

            # Conv 2
            torch.nn.Conv2d(16, 32, kernel_size=3, padding="same"), # output 7 - 2 + 1 => 8
                        # , padding=2),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2) #output 8 / 2 => output 4
```

```
        )

        self.classifier = torch.nn.Sequential(
             torch.nn.Linear(32*4*4, 100),
              torch.nn.ReLU(inplace=True),
              torch.nn.Linear(100, num_classes),
        )

      def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        # print(x.size())
        logits = self.classifier(x)
        return logits


    model1 = CNN1(num_classes=10)


    model1 = model1.to(DEVICE)


    print(summary(model1, (3, 16, 16)))
```

```
    ----------------------------------------------------------------
            Layer (type)               Output Shape         Param #
    ================================================================
                Conv2d-1          [-1, 16, 16, 16]             448
                  ReLU-2          [-1, 16, 16, 16]               0
             MaxPool2d-3            [-1, 16, 8, 8]               0
                Conv2d-4            [-1, 32, 8, 8]           4,640
                  ReLU-5            [-1, 32, 8, 8]               0
             MaxPool2d-6            [-1, 32, 4, 4]               0
                Linear-7                 [-1, 100]          51,300
                  ReLU-8                 [-1, 100]               0
                Linear-9                  [-1, 10]           1,010
    ================================================================
    Total params: 57,398
    Trainable params: 57,398
    Non-trainable params: 0
    ----------------------------------------------------------------
    Input size (MB): 0.00
    Forward/backward pass size (MB): 0.11
    Params size (MB): 0.22
    Estimated Total Size (MB): 0.33
    ----------------------------------------------------------------
    None
```

```
    optimizer1 = torch.optim.SGD(model1.parameters(), lr=0.1)
    scheduler1 = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
                                                            factor=0.1,
                                                            mode='max',
                                                            verbose=True)



    minibatch_loss_list1, train_acc_list1, valid_acc_list1 = train_model(
        model=model1,
        num_epochs=NUM_EPOCHS,
        train_loader=train_loader,
        valid_loader=valid_loader,
        test_loader=test_loader,
        optimizer=optimizer1,
        device=DEVICE,
        scheduler=None,
        scheduler_on='valid_acc',
        logging_interval=100)


    plot_training_loss(minibatch_loss_list=minibatch_loss_list1,
                       num_epochs=NUM_EPOCHS,
                       iter_per_epoch=len(train_loader),
                       results_dir=None,
                       averaging_iterations=20)

    plt.show()

    plot_accuracy(train_acc_list=train_acc_list1,
                  valid_acc_list=valid_acc_list1,
                  results_dir=None)
```

```
# plt.ylim([80, 100])
plt.show()
```

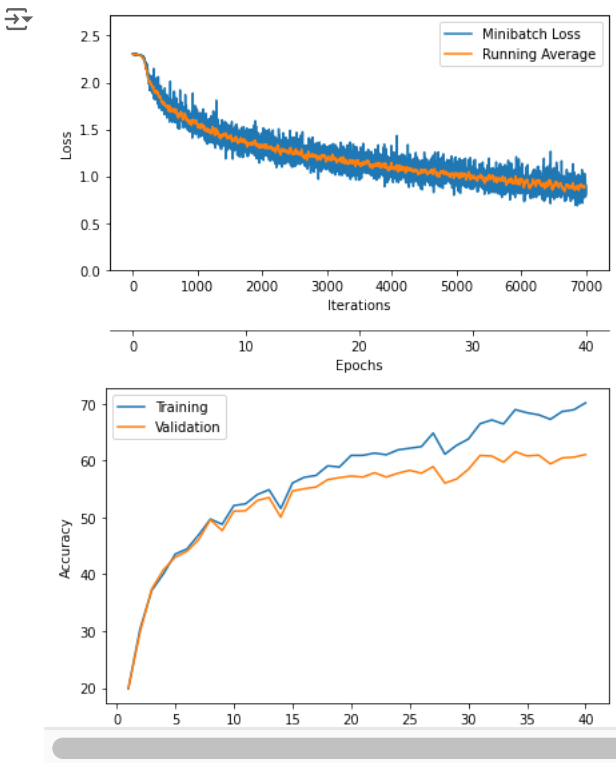



```
class CNN2(torch.nn.Module):
  def __init__(self, num_classes):
    super().__init__()
    self.features = torch.nn.Sequential(
            # Conv 1
            torch.nn.Conv2d(3, 16, kernel_size=3, padding="same"), # output 16 - 3 + 1 => 16
                            # , stride=4, padding=2),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2), # 16 / 2 => output 8

            # Conv 2
            torch.nn.Conv2d(16, 32, kernel_size=2, padding="same"), # output 7 - 2 + 1 => 8
                            # , padding=2),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2), #output 8 / 2 => output 4

            # Conv 3
            torch.nn.Conv2d(32, 64, kernel_size=2, padding="same"), # output 7 - 2 + 1 => 4
                            # , padding=2),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2) #output 4 / 2 => output 2

    )

    self.classifier = torch.nn.Sequential(
         torch.nn.Linear(64*2*2, 100),
          torch.nn.ReLU(inplace=True),
          torch.nn.Linear(100, num_classes),
    )

  def forward(self, x):
    x = self.features(x)
    x = torch.flatten(x, 1)
```

```
        # print(x.size())
        logits = self.classifier(x)
        return logits


    model2 = CNN2(num_classes=10)


    model2 = model2.to(DEVICE)


    print(summary(model2, (3, 16, 16)))
```

```
    ----------------------------------------------------------------
            Layer (type)               Output Shape         Param #
    ================================================================
                Conv2d-1          [-1, 16, 16, 16]             448
                  ReLU-2          [-1, 16, 16, 16]               0
             MaxPool2d-3            [-1, 16, 8, 8]               0
                Conv2d-4            [-1, 32, 8, 8]           2,080
                  ReLU-5            [-1, 32, 8, 8]               0
             MaxPool2d-6            [-1, 32, 4, 4]               0
                Conv2d-7            [-1, 64, 4, 4]           8,256
                  ReLU-8            [-1, 64, 4, 4]               0
             MaxPool2d-9            [-1, 64, 2, 2]               0
               Linear-10                 [-1, 100]          25,700
                 ReLU-11                 [-1, 100]               0
               Linear-12                  [-1, 10]           1,010
    ================================================================
    Total params: 37,494
    Trainable params: 37,494
    Non-trainable params: 0
    ----------------------------------------------------------------
    Input size (MB): 0.00
    Forward/backward pass size (MB): 0.12
    Params size (MB): 0.14
    Estimated Total Size (MB): 0.27
    ----------------------------------------------------------------
    None
    /usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py:454: UserWarning: Using padding='same' with even kernel leng
      self.padding, self.dilation, self.groups)
```

```
    optimizer2 = torch.optim.SGD(model2.parameters(), lr=0.1)
    scheduler2 = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer2,
                                                            factor=0.1,
                                                            mode='max',
                                                            verbose=True)



    minibatch_loss_list2, train_acc_list2, valid_acc_list2 = train_model(
        model=model2,
        num_epochs=NUM_EPOCHS,
        train_loader=train_loader,
        valid_loader=valid_loader,
        test_loader=test_loader,
        optimizer=optimizer2,
        device=DEVICE,
        scheduler=None,
        scheduler_on='valid_acc',
        logging_interval=100)


    plot_training_loss(minibatch_loss_list=minibatch_loss_list2,
                       num_epochs=NUM_EPOCHS,
                       iter_per_epoch=len(train_loader),
                       results_dir=None,
                       averaging_iterations=20)

    plt.show()

    plot_accuracy(train_acc_list=train_acc_list2,
                  valid_acc_list=valid_acc_list2,
                  results_dir=None)

    # plt.ylim([80, 100])
    plt.show()
```




```
import pandas as pd

results = pd.DataFrame({"Number of Parameters": [57398, 37494], "Accuracy": [80, 70]}, index = ["CNN1", "CNN2"])

results
```

	Number of Parameters	Accuracy
CNN1	57398	80
CNN2	37494	70

The simpler CNN is better for this dataset as the more complex model will have very very less pixels to work with

Start coding or generate with AI.