Optimiser 1

```
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_ma
from helper_train import train_model
from helper_plotting import plot_training_loss, plot_accuracy, show_examples, plot_c
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/cifa
100%  170498071/170498071 [00:10<00:00, 17861031.45it/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 CNN1RMS(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 +
                            # , 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 +
                            # , 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_rms = CNN1RMS(num_classes=10)


model1_rms = model1_rms.to(DEVICE)


print(summary(model1_rms, (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
            Linear-7                 [-1, 100]          51,300
              ReLU-8                 [-1, 100]               0
            Linear-9                  [-1, 10]           1,010
================================================================
Total params: 54,838
Trainable params: 54,838
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.11
Params size (MB): 0.21
Estimated Total Size (MB): 0.32
----------------------------------------------------------------
None
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py:454: UserWarning
  self.padding, self.dilation, self.groups)
```

```
optimizer_rms = torch.optim.RMSprop(model1_rms.parameters())
scheduler_rms = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer_rms,
                                                           factor=0.1,
```

```
                                                mode='max',
                                                verbose=True)
```

```
minibatch_loss_list_rms, train_acc_list_rms, valid_acc_list_rms = train_model(
    model=model1_rms,
    num_epochs=NUM_EPOCHS,
    train_loader=train_loader,
    valid_loader=valid_loader,
    test_loader=test_loader,
    optimizer=optimizer_rms,
    device=DEVICE,
    scheduler=None,
    scheduler_on='valid_acc',
    logging_interval=100)
```

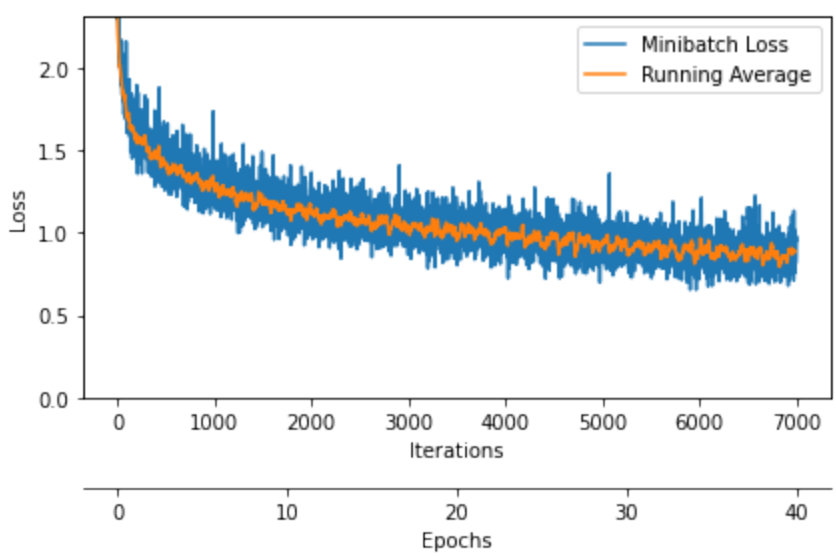
```
Epoch: 037/040 | Batch 0000/0175 | Loss: 0.8526
Epoch: 037/040 | Batch 0100/0175 | Loss: 0.8403
Epoch: 037/040 | Train: 71.17% | Validation: 56.78%
Time elapsed: 47.88 min
Epoch: 038/040 | Batch 0000/0175 | Loss: 0.7887
Epoch: 038/040 | Batch 0100/0175 | Loss: 1.2266
Epoch: 038/040 | Train: 71.57% | Validation: 56.60%
Time elapsed: 49.18 min
Epoch: 039/040 | Batch 0000/0175 | Loss: 0.7900
Epoch: 039/040 | Batch 0100/0175 | Loss: 0.9850
Epoch: 039/040 | Train: 71.37% | Validation: 57.90%
Time elapsed: 50.47 min
Epoch: 040/040 | Batch 0000/0175 | Loss: 0.7807
Epoch: 040/040 | Batch 0100/0175 | Loss: 0.8716
Epoch: 040/040 | Train: 69.35% | Validation: 55.66%
Time elapsed: 51.81 min
Total Training Time: 51.81 min
Test accuracy 55.17%
```

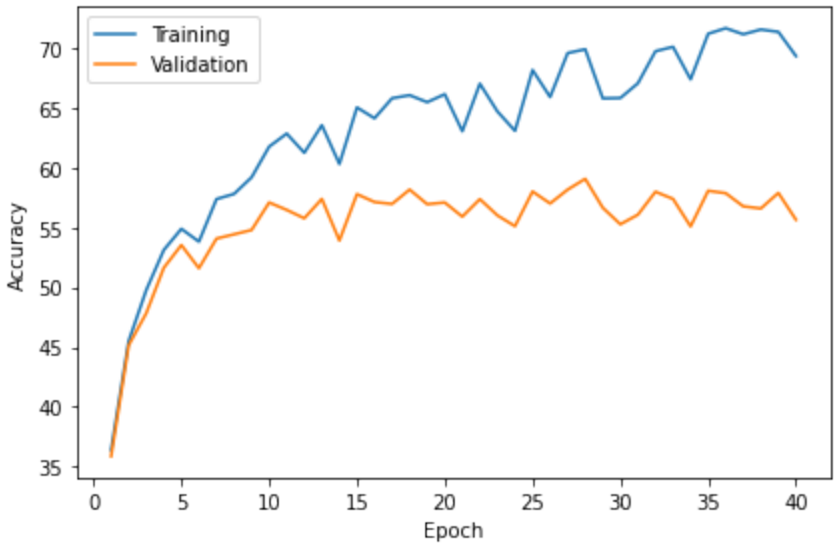
```
plot_training_loss(minibatch_loss_list=minibatch_loss_list_rms,
                   num_epochs=NUM_EPOCHS,
                   iter_per_epoch=len(train_loader),
                   results_dir=None,
                   averaging_iterations=20)

plt.show()

plot_accuracy(train_acc_list=train_acc_list_rms,
              valid_acc_list=valid_acc_list_rms,
              results_dir=None)

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






```
class CNN2RMS(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 +
                            # , 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 +
                            # , padding=2),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2), #output 8 / 2 => output 4

            torch.nn.Conv2d(32, 64, kernel_size=2, padding="same"), # output 7 - 2 +
                            # , padding=2),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2) #output 8 / 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_rms = CNN2RMS(num_classes=10)


    model2_rms = model2_rms.to(DEVICE)


    print(summary(model2_rms, (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
```

```
    optimizer_rms2 = torch.optim.RMSprop(model2_rms.parameters())
    scheduler_rms2 = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer_rms2,
                                                                factor=0.1,
```

```
                                                mode='max',
                                                verbose=True)
```

```
minibatch_loss_list_rms2, train_acc_list_rms2, valid_acc_list_rms2 = train_model(
    model=model2_rms,
    num_epochs=NUM_EPOCHS,
    train_loader=train_loader,
    valid_loader=valid_loader,
    test_loader=test_loader,
    optimizer=optimizer_rms2,
    device=DEVICE,
    scheduler=None,
    scheduler_on='valid_acc',
    logging_interval=100)
```

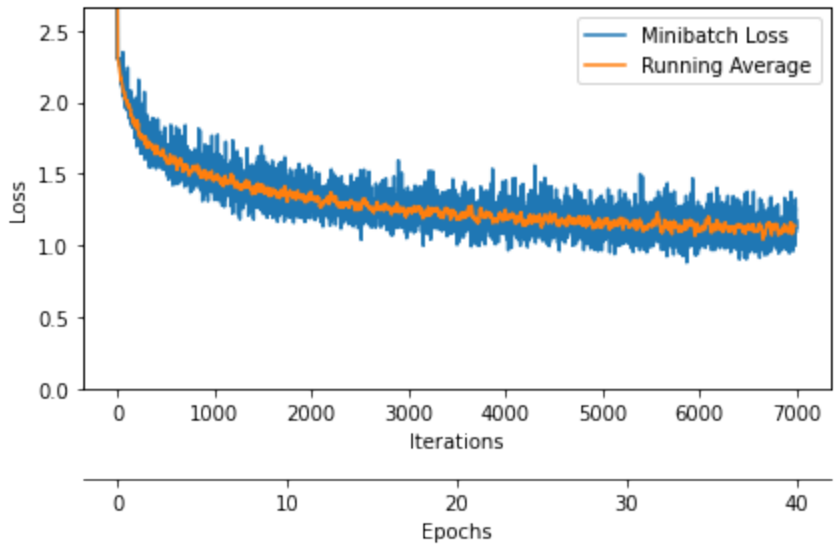
```
Epoch: 036/040 | Batch 0100/0175 | Loss: 1.0481
Epoch: 036/040 | Train: 59.92% | Validation: 54.12%
Time elapsed: 45.01 min
Epoch: 037/040 | Batch 0000/0175 | Loss: 1.0873
Epoch: 037/040 | Batch 0100/0175 | Loss: 1.0833
Epoch: 037/040 | Train: 53.34% | Validation: 48.46%
Time elapsed: 46.26 min
Epoch: 038/040 | Batch 0000/0175 | Loss: 1.3300
Epoch: 038/040 | Batch 0100/0175 | Loss: 1.1263
Epoch: 038/040 | Train: 61.58% | Validation: 54.84%
Time elapsed: 47.51 min
Epoch: 039/040 | Batch 0000/0175 | Loss: 1.1002
Epoch: 039/040 | Batch 0100/0175 | Loss: 1.0842
Epoch: 039/040 | Train: 63.25% | Validation: 56.04%
Time elapsed: 48.76 min
Epoch: 040/040 | Batch 0000/0175 | Loss: 1.0204
Epoch: 040/040 | Batch 0100/0175 | Loss: 1.1030
Epoch: 040/040 | Train: 63.51% | Validation: 55.92%
```

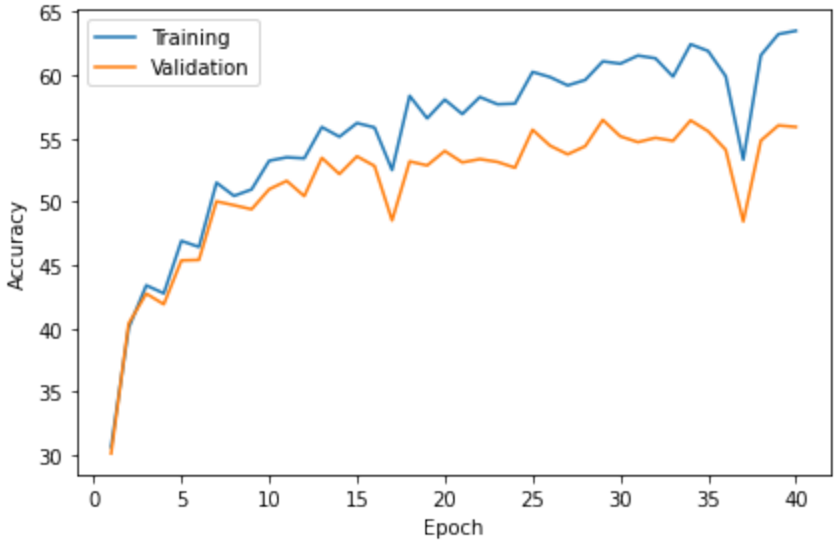
```
plot_training_loss(minibatch_loss_list=minibatch_loss_list_rms2,
                   num_epochs=NUM_EPOCHS,
                   iter_per_epoch=len(train_loader),
                   results_dir=None,
                   averaging_iterations=20)

plt.show()

plot_accuracy(train_acc_list=train_acc_list_rms2,
              valid_acc_list=valid_acc_list_rms2,
              results_dir=None)

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






```
import pandas as pd

results = pd.DataFrame({"Number of Parameters": [54838, 37494], "Accuracy": [70, 63]}

results
```

	Number of Parameters	Accuracy
CNN1	54838	70
CNN2	37494	63