Dropout 2 -> 20, 40, 80

```
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 [00:10<00:00, 16996222.21it/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 CNN2Dropout(torch.nn.Module):
  def __init__(self, num_classes, drop_probas=[]):
    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.Dropout2d(p=drop_probas[0]),
            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.Dropout2d(p=drop_probas[1]),
            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.Dropout2d(p=drop_probas[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_dropout = CNN2Dropout(num_classes=10, drop_probas=[0.2, 0.4, 0.8])


model2_dropout = model2_dropout.to(DEVICE)


print(summary(model2_dropout, (3, 16, 16)))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 16, 16]             448
         Dropout2d-2          [-1, 16, 16, 16]               0
              ReLU-3          [-1, 16, 16, 16]               0
         MaxPool2d-4            [-1, 16, 8, 8]               0
            Conv2d-5            [-1, 32, 8, 8]           2,080
         Dropout2d-6            [-1, 32, 8, 8]               0
              ReLU-7            [-1, 32, 8, 8]               0
         MaxPool2d-8            [-1, 32, 4, 4]               0
            Conv2d-9            [-1, 64, 4, 4]           8,256
        Dropout2d-10            [-1, 64, 4, 4]               0
             ReLU-11            [-1, 64, 4, 4]               0
        MaxPool2d-12            [-1, 64, 2, 2]               0
           Linear-13                 [-1, 100]          25,700
             ReLU-14                 [-1, 100]               0
           Linear-15                  [-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): 0.18
Params size (MB): 0.14
Estimated Total Size (MB): 0.33
----------------------------------------------------------------
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_dropout = torch.optim.SGD(model2_dropout.parameters(), lr=0.1)
scheduler2_dropout = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer2_dropout,
                                                                factor=0.1,
                                                                mode='max',
                                                                verbose=True)



minibatch_loss_list_dropout2, train_acc_list_dropout2, valid_acc_list_dropout2 = train_model(
    model=model2_dropout,
    num_epochs=NUM_EPOCHS,
    train_loader=train_loader,
    valid_loader=valid_loader,
    test_loader=test_loader,
    optimizer=optimizer2_dropout,
```

```
    device=DEVICE,
    scheduler=None,
    scheduler_on='valid_acc',
    logging_interval=100)
```

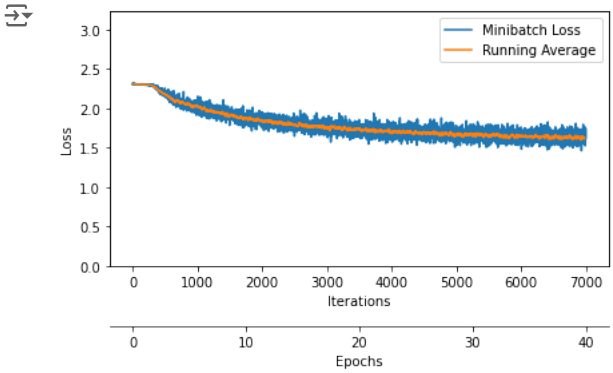
```
Epoch: 025/040 | Batch 0100/0175 | Loss: 1.7826
Epoch: 025/040 | Train: 42.79% | Validation: 42.90%
Time elapsed: 32.42 min
Epoch: 026/040 | Batch 0000/0175 | Loss: 1.8124
Epoch: 026/040 | Batch 0100/0175 | Loss: 1.6125
Epoch: 026/040 | Train: 43.49% | Validation: 43.46%
Time elapsed: 33.74 min
Epoch: 027/040 | Batch 0000/0175 | Loss: 1.6881
Epoch: 027/040 | Batch 0100/0175 | Loss: 1.6497
Epoch: 027/040 | Train: 43.63% | Validation: 43.68%
Time elapsed: 35.05 min
Epoch: 028/040 | Batch 0000/0175 | Loss: 1.6946
Epoch: 028/040 | Batch 0100/0175 | Loss: 1.7066
Epoch: 028/040 | Train: 44.52% | Validation: 44.10%
Time elapsed: 36.38 min
Epoch: 029/040 | Batch 0000/0175 | Loss: 1.6888
Epoch: 029/040 | Batch 0100/0175 | Loss: 1.7111
Epoch: 029/040 | Train: 44.09% | Validation: 44.14%
Time elapsed: 37.68 min
Epoch: 030/040 | Batch 0000/0175 | Loss: 1.6651
Epoch: 030/040 | Batch 0100/0175 | Loss: 1.6679
Epoch: 030/040 | Train: 44.30% | Validation: 44.10%
Time elapsed: 38.99 min
Epoch: 031/040 | Batch 0000/0175 | Loss: 1.5694
Epoch: 031/040 | Batch 0100/0175 | Loss: 1.6693
Epoch: 031/040 | Train: 44.55% | Validation: 44.78%
Time elapsed: 40.28 min
Epoch: 032/040 | Batch 0000/0175 | Loss: 1.5961
Epoch: 032/040 | Batch 0100/0175 | Loss: 1.6088
Epoch: 032/040 | Train: 44.87% | Validation: 44.66%
Time elapsed: 41.58 min
Epoch: 033/040 | Batch 0000/0175 | Loss: 1.5836
Epoch: 033/040 | Batch 0100/0175 | Loss: 1.6808
Epoch: 033/040 | Train: 45.49% | Validation: 45.18%
Time elapsed: 42.86 min
Epoch: 034/040 | Batch 0000/0175 | Loss: 1.7051
Epoch: 034/040 | Batch 0100/0175 | Loss: 1.7858
Epoch: 034/040 | Train: 45.16% | Validation: 45.54%
Time elapsed: 44.17 min
Epoch: 035/040 | Batch 0000/0175 | Loss: 1.6559
Epoch: 035/040 | Batch 0100/0175 | Loss: 1.6319
Epoch: 035/040 | Train: 45.35% | Validation: 45.30%
Time elapsed: 45.46 min
Epoch: 036/040 | Batch 0000/0175 | Loss: 1.5705
Epoch: 036/040 | Batch 0100/0175 | Loss: 1.5781
Epoch: 036/040 | Train: 45.98% | Validation: 45.56%
Time elapsed: 46.76 min
Epoch: 037/040 | Batch 0000/0175 | Loss: 1.6377
Epoch: 037/040 | Batch 0100/0175 | Loss: 1.6536
Epoch: 037/040 | Train: 45.22% | Validation: 45.02%
Time elapsed: 48.09 min
Epoch: 038/040 | Batch 0000/0175 | Loss: 1.6114
Epoch: 038/040 | Batch 0100/0175 | Loss: 1.5813
Epoch: 038/040 | Train: 46.50% | Validation: 45.60%
Time elapsed: 49.38 min
Epoch: 039/040 | Batch 0000/0175 | Loss: 1.7643
Epoch: 039/040 | Batch 0100/0175 | Loss: 1.6195
```

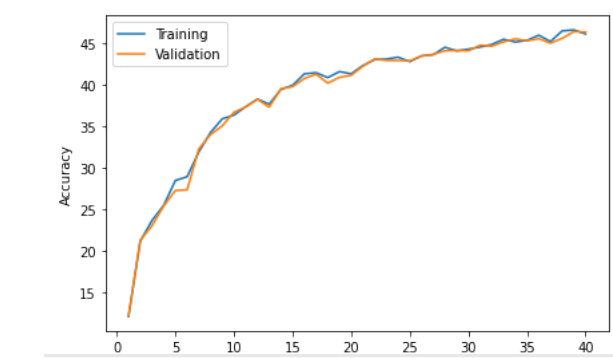
```
plot_training_loss(minibatch_loss_list=minibatch_loss_list_dropout2,
                   num_epochs=NUM_EPOCHS,
                   iter_per_epoch=len(train_loader),
                   results_dir=None,
                   averaging_iterations=20)

plt.show()

plot_accuracy(train_acc_list=train_acc_list_dropout2,
              valid_acc_list=valid_acc_list_dropout2,
              results_dir=None)

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






```
class CNN1Dropout(torch.nn.Module):
  def __init__(self, num_classes, drop_probas=[]):
    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.Dropout2d(p=drop_probas[0]),
            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.Dropout2d(p=drop_probas[1]),
            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.Dropout2d(p=drop_probas[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(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_dropout = CNN1Dropout(num_classes=10, drop_probas=[0.2, 0.4, 0.8])


model1_dropout = model1_dropout.to(DEVICE)
```

```
print(summary(model1_dropout, (3, 16, 16)))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 16, 16]             448
         Dropout2d-2          [-1, 16, 16, 16]               0
              ReLU-3          [-1, 16, 16, 16]               0
         MaxPool2d-4            [-1, 16, 8, 8]               0
            Conv2d-5            [-1, 32, 8, 8]           2,080
         Dropout2d-6            [-1, 32, 8, 8]               0
              ReLU-7            [-1, 32, 8, 8]               0
         MaxPool2d-8            [-1, 32, 4, 4]               0
            Linear-9                 [-1, 100]          51,300
             ReLU-10                 [-1, 100]               0
           Linear-11                  [-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): 0.15
Params size (MB): 0.21
Estimated Total Size (MB): 0.37
----------------------------------------------------------------
None
```

```
optimizer1_dropout = torch.optim.SGD(model1_dropout.parameters(), lr=0.1)
scheduler1_dropout = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer1_dropout,
                                                                factor=0.1,
                                                                mode='max',
                                                                verbose=True)
```

```
minibatch_loss_list_dropout1, train_acc_list_dropout1, valid_acc_list_dropout1 = train_model(
    model=model1_dropout,
    num_epochs=NUM_EPOCHS,
    train_loader=train_loader,
    valid_loader=valid_loader,
    test_loader=test_loader,
    optimizer=optimizer1_dropout,
    device=DEVICE,
    scheduler=None,
    scheduler_on='valid_acc',
    logging_interval=100)
```

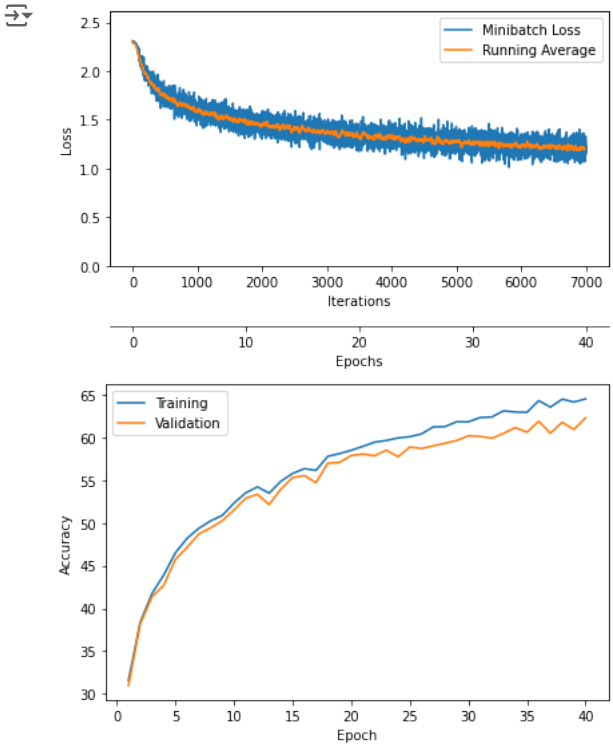
```
Epoch: 036/040 | Batch 0000/0175 | Loss: 1.2477
Epoch: 036/040 | Batch 0100/0175 | Loss: 1.2451
Epoch: 036/040 | Train: 64.36% | Validation: 61.94%
Time elapsed: 53.74 min
Epoch: 037/040 | Batch 0000/0175 | Loss: 1.2403
Epoch: 037/040 | Batch 0100/0175 | Loss: 1.1724
Epoch: 037/040 | Train: 63.60% | Validation: 60.54%
Time elapsed: 55.25 min
Epoch: 038/040 | Batch 0000/0175 | Loss: 1.1645
Epoch: 038/040 | Batch 0100/0175 | Loss: 1.2893
Epoch: 038/040 | Train: 64.53% | Validation: 61.82%
Time elapsed: 56.75 min
Epoch: 039/040 | Batch 0000/0175 | Loss: 1.1437
Epoch: 039/040 | Batch 0100/0175 | Loss: 1.1176
Epoch: 039/040 | Train: 64.20% | Validation: 60.98%
Time elapsed: 58.23 min
Epoch: 040/040 | Batch 0000/0175 | Loss: 1.1646
Epoch: 040/040 | Batch 0100/0175 | Loss: 1.2772
Epoch: 040/040 | Train: 64.57% | Validation: 62.34%
Time elapsed: 59.73 min
Total Training Time: 59.73 min
Test accuracy 60.19%
```

```
plot_training_loss(minibatch_loss_list=minibatch_loss_list_dropout1,
                   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_dropout1,
              valid_acc_list=valid_acc_list_dropout1,
              results_dir=None)

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




```
import pandas as pd

results = pd.DataFrame({"Number of Parameters": [54838, 37494], "Accuracy": [46, 64]}, index = ["CNN1", "CNN2"])

results
```

	Number of Parameters	Accuracy
CNN1	54838	46
CNN2	37494	64

Very surprisingly adding a 80% dropout on the last layer improved the efficiency many fold as it made the model simpler with very low overfitting. This could be the model I would consider running for longer time.

```
Start coding or generate with AI.
```