Problem 1

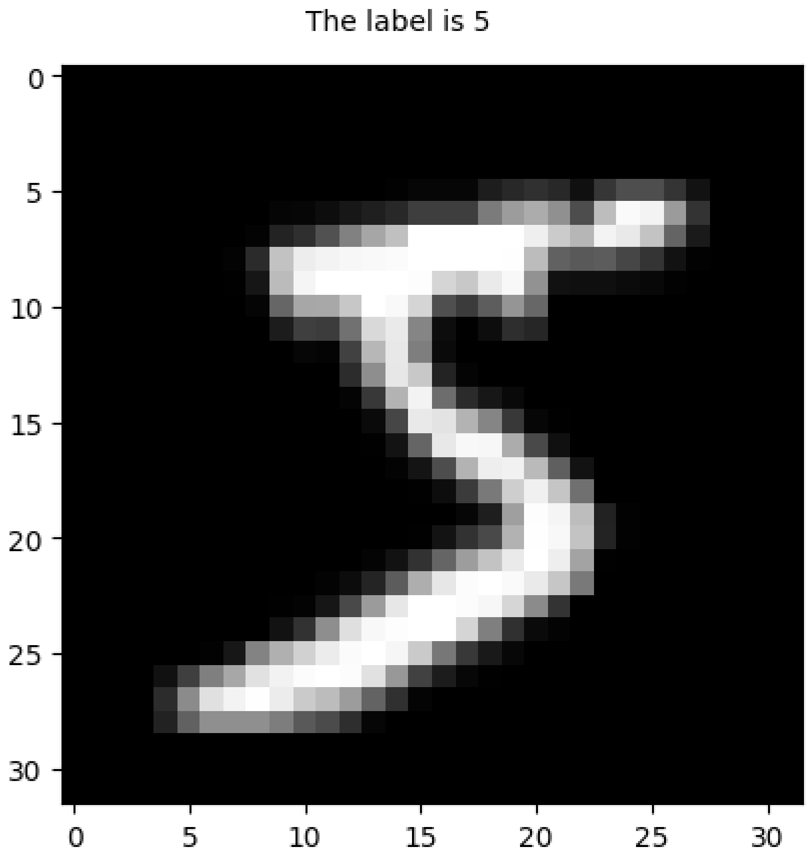
In [ ]:
```
import numpy as np
from datetime import datetime

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader

from torchvision import datasets, transforms

%matplotlib inline
import matplotlib.pyplot as plt
```

In [ ]:
```
# define transforms
transforms = transforms.Compose([transforms.Resize((32, 32)), transforms.

# download and create datasets
train_dataset = datasets.MNIST(
    root="mnist_data", train=True, transform=transforms, download=True
)

valid_dataset = datasets.MNIST(root="mnist_data", train=False, transform=
```

1.1.1

In [ ]:
```
plt.imshow(train_dataset[0][0].squeeze(), cmap="gray")
plt.text(10, -2, "The label is " + str(train_dataset[0][1]))
```

Out[ ]:
```
Text(10, -2, 'The label is 5')
```



In [ ]:
```
# hyper parameters
RANDOM_SEED = 42
LEARNING_RATE = 0.001
BATCH_SIZE = 32
N_EPOCHS = 15

IMG_SIZE = 32
N_CLASSES = 10
```

1.1.2

In [ ]:
```
# define the data loaders
train_loader = DataLoader(dataset=train_dataset, batch_size=BATCH_SIZE, s
valid_loader = DataLoader(dataset=valid_dataset, batch_size=BATCH_SIZE, s
```

1.1.3

In [ ]:
```
def train(train_loader, model, criterion, optimizer):
    """
    Train one epoch.
    """

    model.train()
    running_loss = 0

    for X, y_true in train_loader:

        optimizer.zero_grad()
```

```
        # Forward pass
        y_hat, _ = model(X)
        loss = criterion(y_hat, y_true)
        running_loss += loss.item() * X.size(0)

        # Backward pass
        loss.backward()
        optimizer.step()

    epoch_loss = running_loss / len(train_loader.dataset)
    return model, optimizer, epoch_loss
```

1.1.4

In [ ]:
```
def validate(valid_loader, model, criterion):
    """
    Function for the validation step of the training loop.
    Returns the model and the loss on the test set.
    """

    model.eval()
    running_loss = 0

    for X, y_true in valid_loader:

        # Forward pass and record loss
        y_hat, _ = model(X)
        loss = criterion(y_hat, y_true)
        running_loss += loss.item() * X.size(0)

    epoch_loss = running_loss / len(valid_loader.dataset)

    return model, epoch_loss
```

In [ ]:
```
def training_loop(
    model, criterion, optimizer, train_loader, valid_loader, epochs, prin
):
    """
    Function defining the entire training loop
    """

    # set objects for storing metrics
    best_loss = 1e10
    train_losses = []
    valid_losses = []
    train_accs = []
    valid_accs = []

    # Train model
    for epoch in range(0, epochs):

        # training
        model, optimizer, train_loss = train(train_loader, model, criteri
        train_losses.append(train_loss)

        # validation
        with torch.no_grad():
            model, valid_loss = validate(valid_loader, model, criterion)
```

```
            valid_losses.append(valid_loss)

        if epoch % print_every == (print_every - 1):

            train_acc = get_accuracy(
                model,
                train_loader,
            )
            train_accs.append(train_acc)
            valid_acc = get_accuracy(model, valid_loader)
            valid_accs.append(valid_acc)

            print(
                f"{datetime.now().time().replace(microsecond=0)} "
                f"Epoch: {epoch}\t"
                f"Train loss: {train_loss:.4f}\t"
                f"Valid loss: {valid_loss:.4f}\t"
                f"Train accuracy: {100 * train_acc:.2f}\t"
                f"Valid accuracy: {100 * valid_acc:.2f}"
            )

    performance = {
        "train_losses": train_losses,
        "valid_losses": valid_losses,
        "train_acc": train_accs,
        "valid_acc": valid_accs,
    }

    return model, optimizer, performance
```

1.1.5

In [ ]:

```
def get_accuracy(model, data_loader):
    """
    Function for computing the accuracy of the predictions over the entir
    """

    correct_pred = 0
    n = 0

    with torch.no_grad():
        model.eval()
        for X, y_true in data_loader:

            _, y_prob = model(X)
            _, predicted_labels = torch.max(y_prob, 1)

            n += y_true.size(0)
            correct_pred += (predicted_labels == y_true).sum()

    return correct_pred.float() / n


def plot_performance(performance):
    """
    Function for plotting training and validation losses
    """
```

```
    # temporarily change the style of the plots to seaborn
    # plt.style.use("seaborn")

    fig, ax = plt.subplots(1, 2, figsize=(16, 4.5))
    for key, value in performance.items():
        if "loss" in key:
            ax[0].plot(value, label=key)
        else:
            ax[1].plot(value, label=key)
    ax[0].set(title="Loss  over epochs", xlabel="Epoch", ylabel="Loss")
    ax[1].set(title="accuracy over epochs", xlabel="Epoch", ylabel="Loss"
    ax[0].legend()
    ax[1].legend()
    plt.show()

    # change the plot style to default
    plt.style.use("default")
```

1.2.1

In [ ]:
```
class LeNet5(nn.Module):

    def __init__(self, n_classes):
        super(LeNet5, self).__init__()

        self.feature_extractor = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, strid
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2),
            nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stri
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2),
            nn.Conv2d(in_channels=16, out_channels=120, kernel_size=5, st
            nn.Tanh(),
        )
        self.classifier = nn.Sequential(
            nn.Linear(in_features=120, out_features=84),
            nn.Tanh(),
            nn.Linear(in_features=84, out_features=n_classes),
        )

    def forward(self, x):
        x = self.feature_extractor(x)
        x = torch.flatten(x, 1)
        logits = self.classifier(x)
        probs = F.softmax(logits, dim=1)
        return logits, probs
```

1.2.2

In [ ]:
```
class MLP(nn.Module):

    def __init__(self, layers):
        super(MLP, self).__init__()

        self.all_layers = nn.ModuleList()
```

```
        for i in range(1, len(layers)):
            self.all_layers.append(
                nn.Linear(in_features=layers[i - 1], out_features=layers[
            )
            if i != len(layers) - 1:
                self.all_layers.append(nn.Tanh())
        self.all_layers = nn.Sequential(*self.all_layers)

    def forward(self, x):
        x = x.view(x.shape[0], -1)
        logits = self.all_layers(x)
        probs = F.softmax(logits, dim=1)
        return logits, probs
```

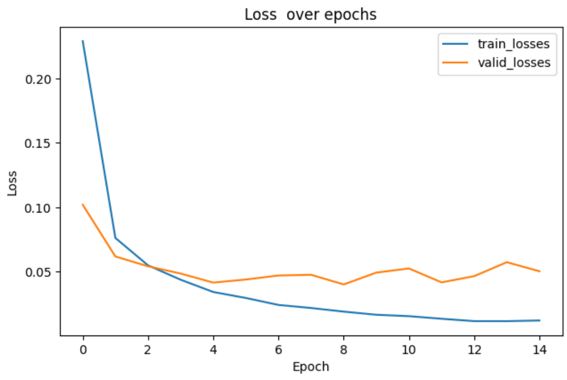
1.3.1

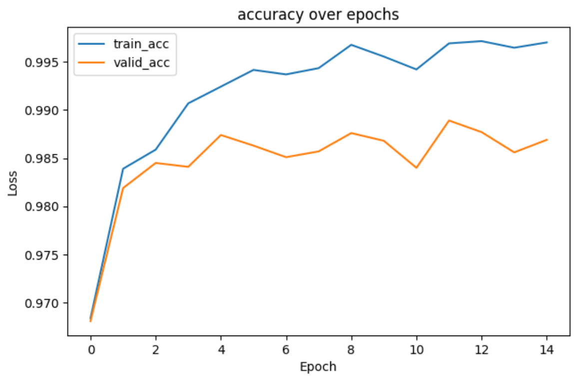
In [ ]:
```
torch.manual_seed(RANDOM_SEED)

model = LeNet5(N_CLASSES)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
criterion = nn.CrossEntropyLoss()
```

In [ ]:
```
model, optimizer, performance_1 = training_loop(
    model, criterion, optimizer, train_loader, valid_loader, N_EPOCHS
)
```

```
21:50:18 Epoch: 0       Train loss: 0.2290      Valid loss: 0.1020      Tr
ain accuracy: 96.84     Valid accuracy: 96.81
21:50:47 Epoch: 1       Train loss: 0.0762      Valid loss: 0.0619      Tr
ain accuracy: 98.39     Valid accuracy: 98.19
21:51:16 Epoch: 2       Train loss: 0.0550      Valid loss: 0.0542      Tr
ain accuracy: 98.59     Valid accuracy: 98.45
21:51:45 Epoch: 3       Train loss: 0.0438      Valid loss: 0.0486      Tr
ain accuracy: 99.07     Valid accuracy: 98.41
21:52:16 Epoch: 4       Train loss: 0.0343      Valid loss: 0.0416      Tr
ain accuracy: 99.24     Valid accuracy: 98.74
21:52:46 Epoch: 5       Train loss: 0.0297      Valid loss: 0.0440      Tr
ain accuracy: 99.42     Valid accuracy: 98.63
21:53:15 Epoch: 6       Train loss: 0.0243      Valid loss: 0.0471      Tr
ain accuracy: 99.37     Valid accuracy: 98.51
21:53:45 Epoch: 7       Train loss: 0.0219      Valid loss: 0.0477      Tr
ain accuracy: 99.43     Valid accuracy: 98.57
21:54:16 Epoch: 8       Train loss: 0.0191      Valid loss: 0.0402      Tr
ain accuracy: 99.68     Valid accuracy: 98.76
21:54:45 Epoch: 9       Train loss: 0.0166      Valid loss: 0.0494      Tr
ain accuracy: 99.55     Valid accuracy: 98.68
21:55:14 Epoch: 10      Train loss: 0.0155      Valid loss: 0.0526      Tr
ain accuracy: 99.42     Valid accuracy: 98.40
21:55:44 Epoch: 11      Train loss: 0.0135      Valid loss: 0.0418      Tr
ain accuracy: 99.69     Valid accuracy: 98.89
21:56:13 Epoch: 12      Train loss: 0.0117      Valid loss: 0.0466      Tr
ain accuracy: 99.71     Valid accuracy: 98.77
21:56:42 Epoch: 13      Train loss: 0.0117      Valid loss: 0.0574      Tr
ain accuracy: 99.65     Valid accuracy: 98.56
21:57:11 Epoch: 14      Train loss: 0.0122      Valid loss: 0.0504      Tr
ain accuracy: 99.70     Valid accuracy: 98.69
```

In [ ]:
```
plot_performance(performance_1)
```





1.3.2

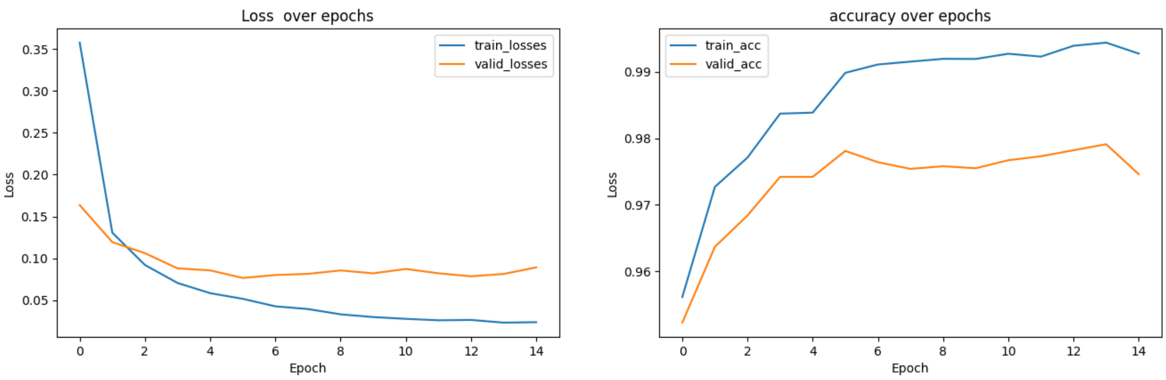
In [ ]:
```
torch.manual_seed(RANDOM_SEED)
layers = [1024, 256, 64, 16, N_CLASSES]
model = MLP(layers)
print(model)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
criterion = nn.CrossEntropyLoss()
```

```
MLP(
  (all_layers): Sequential(
    (0): Linear(in_features=1024, out_features=256, bias=True)
    (1): Tanh()
    (2): Linear(in_features=256, out_features=64, bias=True)
    (3): Tanh()
    (4): Linear(in_features=64, out_features=16, bias=True)
    (5): Tanh()
    (6): Linear(in_features=16, out_features=10, bias=True)
  )
)
```

In [ ]:
```
model, optimizer, performance_2 = training_loop(
    model, criterion, optimizer, train_loader, valid_loader, N_EPOCHS
)
```

```
21:57:18 Epoch: 0      Train loss: 0.3575     Valid loss: 0.1636      Tr
ain accuracy: 95.61    Valid accuracy: 95.23
21:57:24 Epoch: 1      Train loss: 0.1307     Valid loss: 0.1195      Tr
ain accuracy: 97.27    Valid accuracy: 96.37
21:57:31 Epoch: 2      Train loss: 0.0922     Valid loss: 0.1063      Tr
ain accuracy: 97.71    Valid accuracy: 96.84
21:57:37 Epoch: 3      Train loss: 0.0707     Valid loss: 0.0881      Tr
ain accuracy: 98.37    Valid accuracy: 97.42
21:57:44 Epoch: 4      Train loss: 0.0586     Valid loss: 0.0858      Tr
ain accuracy: 98.39    Valid accuracy: 97.42
21:57:50 Epoch: 5      Train loss: 0.0518     Valid loss: 0.0767      Tr
ain accuracy: 98.98    Valid accuracy: 97.81
21:57:56 Epoch: 6      Train loss: 0.0428     Valid loss: 0.0802      Tr
ain accuracy: 99.11    Valid accuracy: 97.64
21:58:03 Epoch: 7      Train loss: 0.0396     Valid loss: 0.0816      Tr
ain accuracy: 99.15    Valid accuracy: 97.54
21:58:09 Epoch: 8      Train loss: 0.0333     Valid loss: 0.0857      Tr
ain accuracy: 99.19    Valid accuracy: 97.58
21:58:15 Epoch: 9      Train loss: 0.0300     Valid loss: 0.0822      Tr
ain accuracy: 99.19    Valid accuracy: 97.55
21:58:22 Epoch: 10     Train loss: 0.0279     Valid loss: 0.0874      Tr
ain accuracy: 99.27    Valid accuracy: 97.67
21:58:28 Epoch: 11     Train loss: 0.0262     Valid loss: 0.0823      Tr
ain accuracy: 99.23    Valid accuracy: 97.73
21:58:35 Epoch: 12     Train loss: 0.0266     Valid loss: 0.0787      Tr
ain accuracy: 99.39    Valid accuracy: 97.82
21:58:41 Epoch: 13     Train loss: 0.0234     Valid loss: 0.0815      Tr
ain accuracy: 99.44    Valid accuracy: 97.91
21:58:47 Epoch: 14     Train loss: 0.0239     Valid loss: 0.0893      Tr
ain accuracy: 99.27    Valid accuracy: 97.46
```

In [ ]:
```
plot_performance(performance_2)
```



1.4 Comparison of these two models.

Convolutional Layers: parameters = (input_channel * filter_size + 1) * output_channel

Fully Connected Layers: parameters = input_channel * output_channel + output_channel

1.What is the number of trainable parameters of LeNet? 5 points

- Convolutional Layers 1: (5*5*1+1) * 6 = 156
- Convolutional Layers 2: (5*5*6+1) * 16 = 2416

- Convolutional Layers 3: (5*5*16+1) * 120 = 48120
- Fully Connected Layers 1: 120*84+84 = 10164
- Fully Connected Layers 2: 84*10+10 = 850
- Total: 156 + 2416 + 48120 + 10164 + 850 = 61706

  2.What is the number of trainable parameters of MLP? 5 points

- Fully Connected Layers 1: 1024*256+256 = 262400
- Fully Connected Layers 2: 256*64+64 = 16448
- Fully Connected Layers 3: 64*16+16 = 1040
- Last Fully Connected Layers: 16*10+10 = 170
- Total: 262400 + 16448 + 1040 + 170 = 280058

  3.Which model has better performance in terms of prediction accuracy on the test data? Give a reason why this model works better than the other. 10 points

LeNet is better.

When using MLP to process images, it is often necessary to flatten the image from its original form into a one-dimensional vector. This flattening process destroys the spatial relationship between pixels.

LeNet is able to understand the spatial relationship between pixels in an image

Statement of Collaboration (5 points)

I do it by myself.