notebook-6

October 14, 2024

1 Assignment 6: Neural Networks

2 Problem 1: The dataset

Loading the MNIST dataset from `torchvision.datasets`:

[263]:
```
import csv

import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor

train_data = datasets.MNIST(
        root='data',
        train=True,
        download=True,
        transform=ToTensor()
)

test_data = datasets.MNIST(
        root='data',
        train=False,
        download=True,
        transform=ToTensor()
)
```

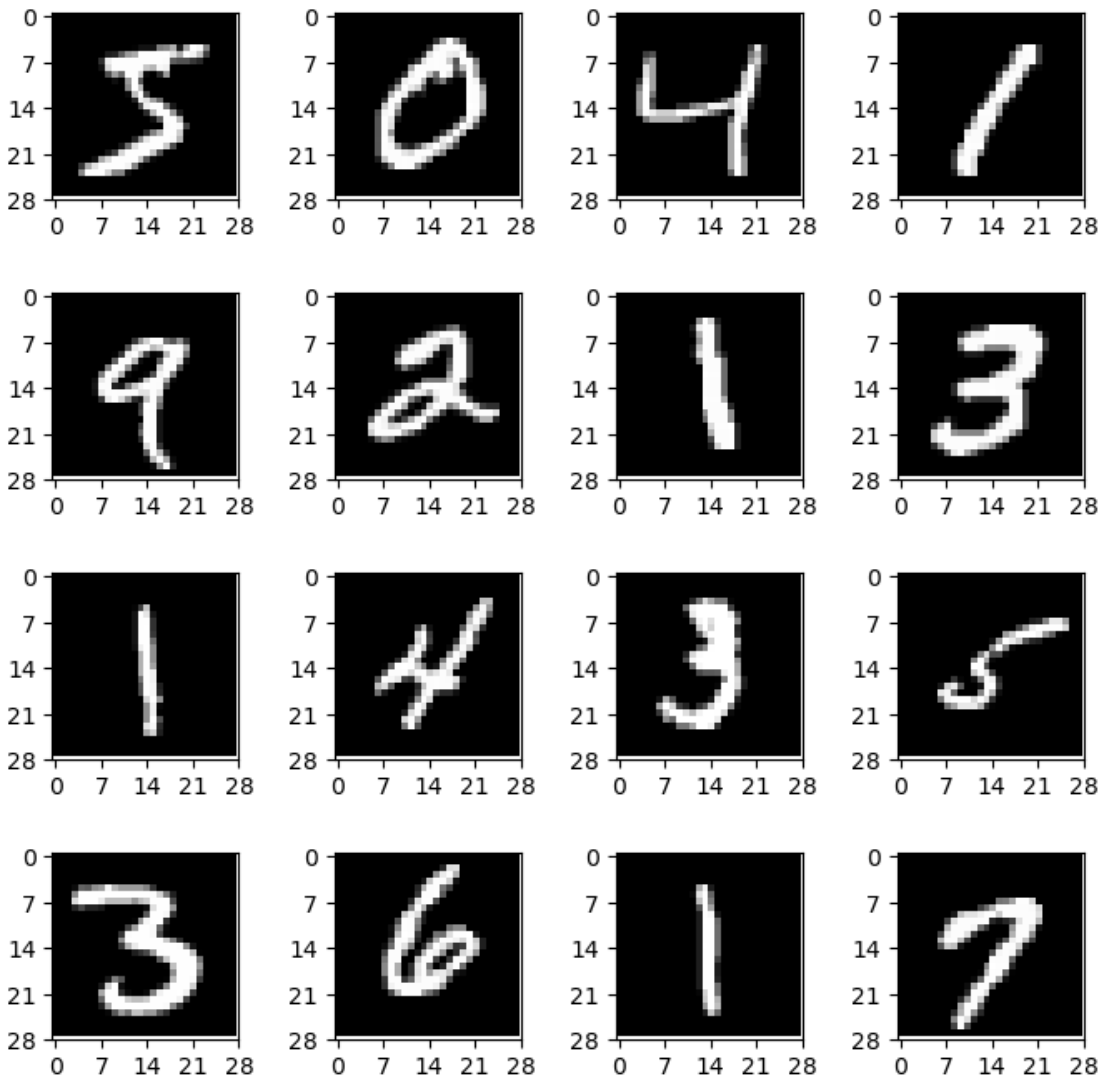
Wrapping the dataset in a `DataLoader`:

[264]:
```
train_loader = DataLoader(
        train_data,
        batch_size=64,
)

test_loader = DataLoader(
        test_data,
        batch_size=64,
```

```
)
```

Plotting the first images from the training set:

[265]:
```
import matplotlib.pyplot as plt

images, _ = next(iter(train_loader))

ppr = 4  # plots per row

fig, ax = plt.subplots(
        ppr, ppr,
        figsize=(ppr * 2, ppr * 2)
)
plt.subplots_adjust(wspace=0.5, hspace=0.5)
for i in range(ppr ** 2):
        ax[i // ppr, i % ppr].imshow(
                images[i].squeeze(),
                cmap='gray'
        )
        ax[i // ppr, i % ppr].set_xticks(
                range(0, 29, 7))
        ax[i // ppr, i % ppr].set_yticks(
                range(0, 29, 7))

plt.savefig('figures/mnist.png')
```



3 Problem 2: Single hidden layer

3.1 Designing the model

Creating a neural network model with a single hidden layer with ReLU activation.

Using SGD as optimizer and CrossEntropy as loss function.

Also, defining the device to be used:

[266]:
```
device = (
        "cuda"
        if torch.cuda.is_available()
        else "mps"
```

```
        if torch.backends.mps.is_available()
        else "cpu"
)
print(f"Using device: {device}")
```

```
Using device: mps
```

[267]:
```
class FNN(nn.Module):
        def __init__(self):
                super(FNN, self).__init__()
                self.flatten = nn.Flatten()
                self.linear_stack = nn.Sequential(
                        nn.Linear(28 * 28, 512),
                        nn.ReLU(),
                        nn.Linear(512, 10),
                )

        def forward(self, x):
                x = self.flatten(x)
                x = self.linear_stack(x)
                return x


model = FNN().to(device)
```

[268]:
```
loss_fn = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(
        model.parameters(),
        lr=1e-3
)
```

3.2 Defining the training, testing, and evaluation functions

Defining the training of the model:

[269]:
```
def train(dataloader, model, loss_fn, optimizer,
          logging=False):
        losses = []
        size = len(dataloader.dataset)
        model.train()
        for batch, (X, y) in enumerate(dataloader):
                X, y = X.to(device), y.to(device)

                pred = model(X)
                loss = loss_fn(pred, y)

                optimizer.zero_grad()
```

```
                loss.backward()
                optimizer.step()

                if batch % 100 == 0:
                        losses.append(loss.item())
                        if logging:
                                current = batch * len(X)
                                print(
                                        f"loss: {loss.item():>7f}  [{current:
↪>5d}/{size:>5d}]"
                                )
        return losses
```

Defining the testing of the model performance against the `test_data`:

[270]:
```
def test(dataloader, model, loss_fn,
         logging=True):
        size = len(dataloader.dataset)
        num_batches = len(dataloader)
        model.eval()
        test_loss, correct = 0, 0
        with (torch.no_grad()):
                for X, y in dataloader:
                        X, y = X.to(device), y.to(device)
                        pred = model(X)
                        test_loss += loss_fn(pred, y).item()
                        correct += (
                                        pred.argmax(1) == y
                        ).type(torch.float).sum().item()
        test_loss /= num_batches
        accuracy = round(correct / size * 100, 4)
        if logging:
                print(f"\n accuracy: {accuracy:>0.1f}%")
                print(f" test loss: {test_loss:>8f}\n")
        return accuracy, test_loss
```

Defining the export of the accuracy data to a csv file:

[271]:
```
def save_accuracy_data(
                filename, accuracy_list
):
        with open(filename, mode='w') as file:
                writer = csv.writer(file)
                writer.writerow(["epoch", "accuracy"])
                for epoch, accuracy in enumerate(
                                accuracy_list):
                        writer.writerow([epoch + 1, accuracy])
```

Defining the plotting of the training and test loss:

[272]:
```
def plot_loss(
                train_loss, test_loss, epochs, title, filename
):
        epoch_length = len(train_loss) // len(test_loss)
        max_lines = 10
        epoch_step = max(1, len(epochs) // max_lines)
        plt.plot(train_loss, label="train")
        plt.plot(
                range(
                        epoch_length,
                        len(train_loss) + epoch_length,
                        epoch_length
                ),
                test_loss, label="test", marker='o'
        )
        selected_epochs = epochs[::epoch_step]
        for epoch in selected_epochs:
                plt.axvline(
                        epoch * epoch_length,
                        color='gray', linestyle='--', alpha=0.5
                )
        plt.xticks(
                [0] + list(range(
                        epoch_length,
                        len(train_loss) + 1,
                        epoch_length * epoch_step
                )),
                [0] + list(selected_epochs)
        )

        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.legend()
        plt.title(title)
        plt.savefig(f"figures/{filename}")
```

3.3 Training and evaluating the model

Training the model for 10 epochs and evaluating its performance:

[273]:
```
epochs = range(1, 11)

train_loss = []
test_loss = []
accuracy_list = []
for t in epochs:
        print(f"training epoch {t}...")
```

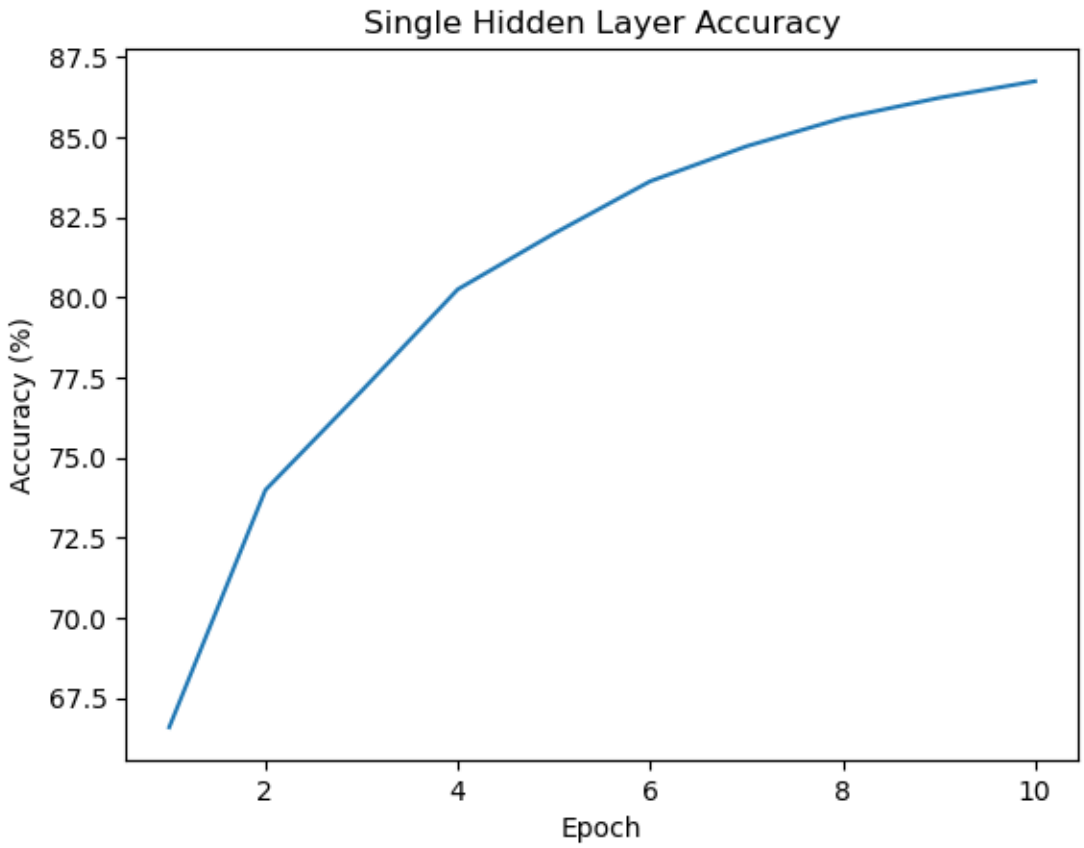
```
        epoch_train_loss = train(
                train_loader, model, loss_fn, optimizer,
        )
        accuracy, epoch_test_loss = test(
                test_loader, model, loss_fn
        )

        train_loss.extend(epoch_train_loss)
        test_loss.append(epoch_test_loss)
        accuracy_list.append(accuracy)

        print(f"--------------------------------")

torch.save(
        model.state_dict(),
        "models/single_layer.pth"
)
print("training complete, model saved")
```

```
training epoch 1…

 accuracy: 66.6%
 test loss: 2.064460

--------------------------------
training epoch 2…

 accuracy: 74.0%
 test loss: 1.763783

--------------------------------
training epoch 3…

 accuracy: 77.1%
 test loss: 1.426791

--------------------------------
training epoch 4…

 accuracy: 80.2%
 test loss: 1.144369

--------------------------------
training epoch 5…

 accuracy: 82.0%
 test loss: 0.946353
```

```
--------------------------------
training epoch 6…

 accuracy: 83.6%
 test loss: 0.812521

--------------------------------
training epoch 7…

 accuracy: 84.7%
 test loss: 0.719491

--------------------------------
training epoch 8…

 accuracy: 85.6%
 test loss: 0.652010

--------------------------------
training epoch 9…

 accuracy: 86.2%
 test loss: 0.601070

--------------------------------
training epoch 10…

 accuracy: 86.7%
 test loss: 0.561327

--------------------------------
training complete, model saved
```

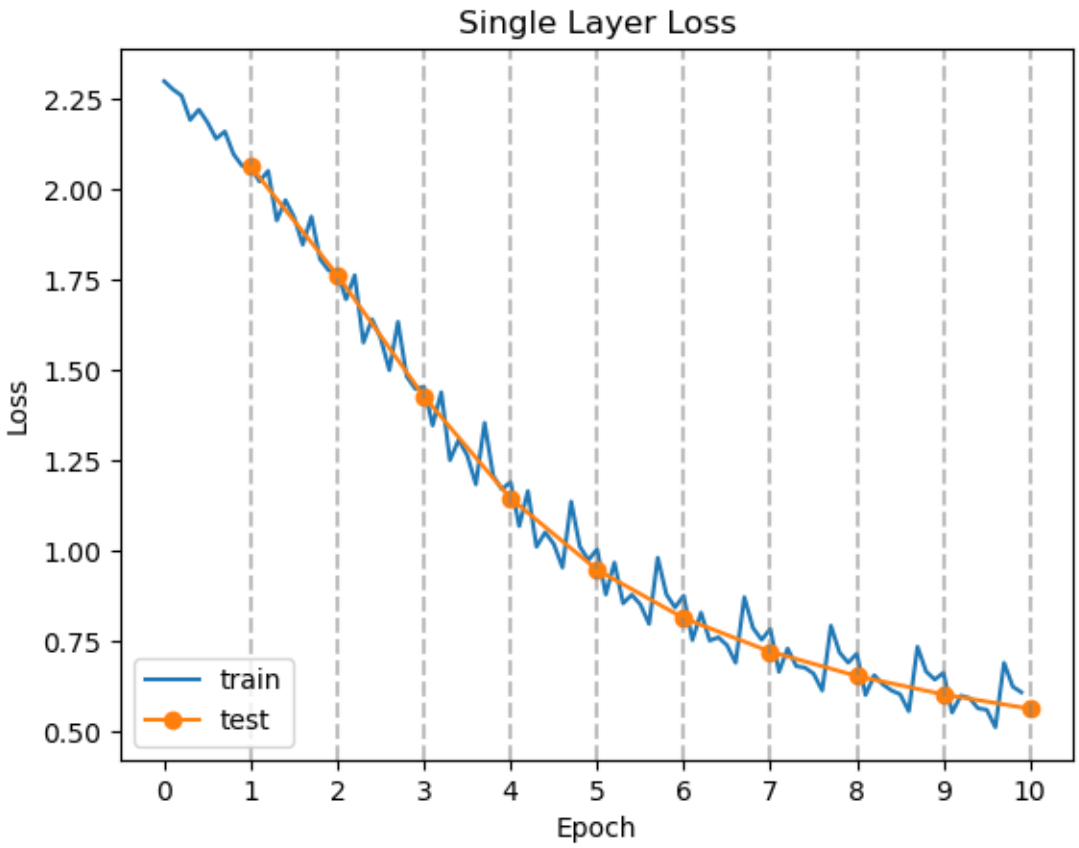
Saving and plotting the accuracy data:

[274]:
```
save_accuracy_data(
        "models/single_layer_accuracy.csv",
        accuracy_list
)

plt.plot(epochs, accuracy_list)
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.title("Single Hidden Layer Accuracy")
plt.savefig(
        "figures/single_layer_accuracy.png")
```



Plotting the training and test loss:

[275]:
```
plot_loss(
        train_loss, test_loss, epochs,
        "Single Layer Loss",
        "single_layer_loss.png"
)
```



4 Problem 3: Two hidden layers

4.1 Designing the model

Creating a neural network model with two hidden layers with ReLU activation.

Reusing the CrossEntropy as loss function and using SGD with L2 regularization as optimizer.

Reusing the device definition:

[248]:
```
class FNN2(nn.Module):
        def __init__(self):
                super(FNN2, self).__init__()
                self.flatten = nn.Flatten()
                self.linear_stack = nn.Sequential(
                        nn.Linear(28 * 28, 500),
                        nn.ReLU(),
                        nn.Linear(500, 300),
                        nn.ReLU(),
                        nn.Linear(300, 10),
                )
```

```
        def forward(self, x):
                x = self.flatten(x)
                x = self.linear_stack(x)
                return x


model = FNN2().to(device)
```

[249]:
```
optimizer = torch.optim.SGD(
        model.parameters(),
        lr=0.1,
        weight_decay=1e-4
)
```

4.2 Training and evaluating the model

Train the model for 40 epochs and evaluate its performance:

[250]:
```
epochs = range(1, 41)

train_loss = []
test_loss = []
accuracy_list = []
for t in epochs:
        print(f"training epoch {t}...")

        epoch_train_loss = train(
                train_loader, model, loss_fn, optimizer,
        )
        accuracy, epoch_test_loss = test(
                test_loader, model, loss_fn
        )

        train_loss.extend(epoch_train_loss)
        test_loss.append(epoch_test_loss)
        accuracy_list.append(accuracy)

        print(f"-------------------------------")

torch.save(
        model.state_dict(),
        "models/two_layer.pth"
)
print("training complete, model saved")
```

```
training epoch 1…
loss: 2.306354  [    0/60000]
```

```
loss: 0.639907  [ 6400/60000]
loss: 0.376103  [12800/60000]
loss: 0.393578  [19200/60000]
loss: 0.247066  [25600/60000]
loss: 0.306391  [32000/60000]
loss: 0.198794  [38400/60000]
loss: 0.291336  [44800/60000]
loss: 0.268996  [51200/60000]
loss: 0.289349  [57600/60000]

 accuracy: 93.5%
 test loss: 0.213384

------------------------------
training epoch 2…
loss: 0.124922  [    0/60000]
loss: 0.176776  [ 6400/60000]
loss: 0.127721  [12800/60000]
loss: 0.283724  [19200/60000]
loss: 0.113152  [25600/60000]
loss: 0.205738  [32000/60000]
loss: 0.092275  [38400/60000]
loss: 0.220331  [44800/60000]
loss: 0.160659  [51200/60000]
loss: 0.215368  [57600/60000]

 accuracy: 95.6%
 test loss: 0.142748

------------------------------
training epoch 3…
loss: 0.080809  [    0/60000]
loss: 0.121869  [ 6400/60000]
loss: 0.092733  [12800/60000]
loss: 0.197430  [19200/60000]
loss: 0.065760  [25600/60000]
loss: 0.149600  [32000/60000]
loss: 0.072447  [38400/60000]
loss: 0.179037  [44800/60000]
loss: 0.124387  [51200/60000]
loss: 0.170140  [57600/60000]

 accuracy: 96.6%
 test loss: 0.111089

------------------------------
training epoch 4…
loss: 0.054775  [    0/60000]
```

```
loss: 0.102790  [ 6400/60000]
loss: 0.078299  [12800/60000]
loss: 0.145227  [19200/60000]
loss: 0.045690  [25600/60000]
loss: 0.113134  [32000/60000]
loss: 0.062570  [38400/60000]
loss: 0.140803  [44800/60000]
loss: 0.101603  [51200/60000]
loss: 0.133380  [57600/60000]

 accuracy: 97.0%
 test loss: 0.093897

------------------------------
training epoch 5…
loss: 0.035720  [    0/60000]
loss: 0.087336  [ 6400/60000]
loss: 0.059451  [12800/60000]
loss: 0.120312  [19200/60000]
loss: 0.033163  [25600/60000]
loss: 0.087179  [32000/60000]
loss: 0.054606  [38400/60000]
loss: 0.108818  [44800/60000]
loss: 0.087525  [51200/60000]
loss: 0.100318  [57600/60000]

 accuracy: 97.3%
 test loss: 0.084124

------------------------------
training epoch 6…
loss: 0.028255  [    0/60000]
loss: 0.073175  [ 6400/60000]
loss: 0.045623  [12800/60000]
loss: 0.099716  [19200/60000]
loss: 0.021359  [25600/60000]
loss: 0.066162  [32000/60000]
loss: 0.046891  [38400/60000]
loss: 0.085273  [44800/60000]
loss: 0.083144  [51200/60000]
loss: 0.077994  [57600/60000]

 accuracy: 97.4%
 test loss: 0.079029

------------------------------
training epoch 7…
loss: 0.025306  [    0/60000]
```

```
loss: 0.057727  [ 6400/60000]
loss: 0.037718  [12800/60000]
loss: 0.067371  [19200/60000]
loss: 0.016086  [25600/60000]
loss: 0.050749  [32000/60000]
loss: 0.040136  [38400/60000]
loss: 0.068121  [44800/60000]
loss: 0.076883  [51200/60000]
loss: 0.056774  [57600/60000]

 accuracy: 97.6%
 test loss: 0.074893

-------------------------------
training epoch 8…
loss: 0.023121  [    0/60000]
loss: 0.047457  [ 6400/60000]
loss: 0.030275  [12800/60000]
loss: 0.039446  [19200/60000]
loss: 0.012059  [25600/60000]
loss: 0.035669  [32000/60000]
loss: 0.032065  [38400/60000]
loss: 0.052159  [44800/60000]
loss: 0.070707  [51200/60000]
loss: 0.042229  [57600/60000]

 accuracy: 97.7%
 test loss: 0.071835

-------------------------------
training epoch 9…
loss: 0.020973  [    0/60000]
loss: 0.034979  [ 6400/60000]
loss: 0.024846  [12800/60000]
loss: 0.031590  [19200/60000]
loss: 0.010755  [25600/60000]
loss: 0.026014  [32000/60000]
loss: 0.024519  [38400/60000]
loss: 0.038010  [44800/60000]
loss: 0.062108  [51200/60000]
loss: 0.032637  [57600/60000]

 accuracy: 97.8%
 test loss: 0.069923

-------------------------------
training epoch 10…
loss: 0.019926  [    0/60000]
```

```
loss: 0.028000  [ 6400/60000]
loss: 0.020781  [12800/60000]
loss: 0.026761  [19200/60000]
loss: 0.010283  [25600/60000]
loss: 0.020168  [32000/60000]
loss: 0.016260  [38400/60000]
loss: 0.027560  [44800/60000]
loss: 0.055147  [51200/60000]
loss: 0.025072  [57600/60000]

 accuracy: 97.7%
 test loss: 0.070033

------------------------------
training epoch 11…
loss: 0.020494  [    0/60000]
loss: 0.022352  [ 6400/60000]
loss: 0.016055  [12800/60000]
loss: 0.023432  [19200/60000]
loss: 0.009174  [25600/60000]
loss: 0.014574  [32000/60000]
loss: 0.011752  [38400/60000]
loss: 0.018883  [44800/60000]
loss: 0.044497  [51200/60000]
loss: 0.019663  [57600/60000]

 accuracy: 97.8%
 test loss: 0.069702

------------------------------
training epoch 12…
loss: 0.017560  [    0/60000]
loss: 0.017512  [ 6400/60000]
loss: 0.012768  [12800/60000]
loss: 0.019130  [19200/60000]
loss: 0.008413  [25600/60000]
loss: 0.011352  [32000/60000]
loss: 0.008595  [38400/60000]
loss: 0.014671  [44800/60000]
loss: 0.038991  [51200/60000]
loss: 0.016308  [57600/60000]

 accuracy: 97.8%
 test loss: 0.069244

------------------------------
training epoch 13…
loss: 0.015213  [    0/60000]
```

```
loss: 0.013910  [ 6400/60000]
loss: 0.011942  [12800/60000]
loss: 0.013639  [19200/60000]
loss: 0.007267  [25600/60000]
loss: 0.008855  [32000/60000]
loss: 0.006584  [38400/60000]
loss: 0.011283  [44800/60000]
loss: 0.027501  [51200/60000]
loss: 0.013249  [57600/60000]

 accuracy: 97.9%
 test loss: 0.068065

--------------------------------
training epoch 14…
loss: 0.013912  [    0/60000]
loss: 0.012580  [ 6400/60000]
loss: 0.012087  [12800/60000]
loss: 0.011297  [19200/60000]
loss: 0.006457  [25600/60000]
loss: 0.006670  [32000/60000]
loss: 0.005493  [38400/60000]
loss: 0.009650  [44800/60000]
loss: 0.020879  [51200/60000]
loss: 0.011520  [57600/60000]

 accuracy: 97.9%
 test loss: 0.067845

--------------------------------
training epoch 15…
loss: 0.014021  [    0/60000]
loss: 0.010329  [ 6400/60000]
loss: 0.011283  [12800/60000]
loss: 0.009269  [19200/60000]
loss: 0.005887  [25600/60000]
loss: 0.005685  [32000/60000]
loss: 0.004851  [38400/60000]
loss: 0.007993  [44800/60000]
loss: 0.014697  [51200/60000]
loss: 0.010123  [57600/60000]

 accuracy: 97.9%
 test loss: 0.067993

--------------------------------
training epoch 16…
loss: 0.013906  [    0/60000]
```

```
loss: 0.009898  [ 6400/60000]
loss: 0.010435  [12800/60000]
loss: 0.008116  [19200/60000]
loss: 0.004950  [25600/60000]
loss: 0.004973  [32000/60000]
loss: 0.003892  [38400/60000]
loss: 0.006992  [44800/60000]
loss: 0.011988  [51200/60000]
loss: 0.008258  [57600/60000]

 accuracy: 97.9%
 test loss: 0.067592

------------------------------
training epoch 17…
loss: 0.012608  [    0/60000]
loss: 0.008520  [ 6400/60000]
loss: 0.009716  [12800/60000]
loss: 0.006985  [19200/60000]
loss: 0.004676  [25600/60000]
loss: 0.004444  [32000/60000]
loss: 0.003331  [38400/60000]
loss: 0.006199  [44800/60000]
loss: 0.011053  [51200/60000]
loss: 0.007409  [57600/60000]

 accuracy: 97.9%
 test loss: 0.067690

------------------------------
training epoch 18…
loss: 0.011077  [    0/60000]
loss: 0.007372  [ 6400/60000]
loss: 0.009193  [12800/60000]
loss: 0.005969  [19200/60000]
loss: 0.004382  [25600/60000]
loss: 0.004125  [32000/60000]
loss: 0.002928  [38400/60000]
loss: 0.005716  [44800/60000]
loss: 0.009354  [51200/60000]
loss: 0.006711  [57600/60000]

 accuracy: 97.9%
 test loss: 0.067855

------------------------------
training epoch 19…
loss: 0.009815  [    0/60000]
```

```
loss: 0.006786  [ 6400/60000]
loss: 0.008641  [12800/60000]
loss: 0.005299  [19200/60000]
loss: 0.004193  [25600/60000]
loss: 0.003712  [32000/60000]
loss: 0.002477  [38400/60000]
loss: 0.005268  [44800/60000]
loss: 0.008240  [51200/60000]
loss: 0.005989  [57600/60000]

 accuracy: 98.0%
 test loss: 0.066742

------------------------------
training epoch 20…
loss: 0.008864  [    0/60000]
loss: 0.006424  [ 6400/60000]
loss: 0.008399  [12800/60000]
loss: 0.004535  [19200/60000]
loss: 0.003845  [25600/60000]
loss: 0.003414  [32000/60000]
loss: 0.002217  [38400/60000]
loss: 0.004922  [44800/60000]
loss: 0.007256  [51200/60000]
loss: 0.005305  [57600/60000]

 accuracy: 98.0%
 test loss: 0.066518

------------------------------
training epoch 21…
loss: 0.007382  [    0/60000]
loss: 0.006084  [ 6400/60000]
loss: 0.007735  [12800/60000]
loss: 0.004180  [19200/60000]
loss: 0.003740  [25600/60000]
loss: 0.003110  [32000/60000]
loss: 0.001996  [38400/60000]
loss: 0.004678  [44800/60000]
loss: 0.006610  [51200/60000]
loss: 0.004916  [57600/60000]

 accuracy: 98.0%
 test loss: 0.066020

------------------------------
training epoch 22…
loss: 0.006563  [    0/60000]
```

```
loss: 0.005469  [ 6400/60000]
loss: 0.007392  [12800/60000]
loss: 0.003913  [19200/60000]
loss: 0.003339  [25600/60000]
loss: 0.002976  [32000/60000]
loss: 0.001849  [38400/60000]
loss: 0.004486  [44800/60000]
loss: 0.005931  [51200/60000]
loss: 0.004587  [57600/60000]

 accuracy: 98.1%
 test loss: 0.064838

------------------------------
training epoch 23…
loss: 0.006124  [    0/60000]
loss: 0.005541  [ 6400/60000]
loss: 0.006917  [12800/60000]
loss: 0.003555  [19200/60000]
loss: 0.003128  [25600/60000]
loss: 0.002899  [32000/60000]
loss: 0.001691  [38400/60000]
loss: 0.004304  [44800/60000]
loss: 0.005407  [51200/60000]
loss: 0.004323  [57600/60000]

 accuracy: 98.1%
 test loss: 0.064149

------------------------------
training epoch 24…
loss: 0.005654  [    0/60000]
loss: 0.005100  [ 6400/60000]
loss: 0.006581  [12800/60000]
loss: 0.003339  [19200/60000]
loss: 0.002949  [25600/60000]
loss: 0.002728  [32000/60000]
loss: 0.001612  [38400/60000]
loss: 0.004142  [44800/60000]
loss: 0.004932  [51200/60000]
loss: 0.004141  [57600/60000]

 accuracy: 98.1%
 test loss: 0.063627

------------------------------
training epoch 25…
loss: 0.005268  [    0/60000]
```

```
loss: 0.004988  [ 6400/60000]
loss: 0.006433  [12800/60000]
loss: 0.003255  [19200/60000]
loss: 0.002829  [25600/60000]
loss: 0.002687  [32000/60000]
loss: 0.001544  [38400/60000]
loss: 0.004016  [44800/60000]
loss: 0.004654  [51200/60000]
loss: 0.003910  [57600/60000]

 accuracy: 98.2%
 test loss: 0.063239

------------------------------
training epoch 26…
loss: 0.004938  [    0/60000]
loss: 0.004806  [ 6400/60000]
loss: 0.006150  [12800/60000]
loss: 0.003115  [19200/60000]
loss: 0.002786  [25600/60000]
loss: 0.002591  [32000/60000]
loss: 0.001485  [38400/60000]
loss: 0.003951  [44800/60000]
loss: 0.004356  [51200/60000]
loss: 0.003761  [57600/60000]

 accuracy: 98.2%
 test loss: 0.062936

------------------------------
training epoch 27…
loss: 0.004448  [    0/60000]
loss: 0.004514  [ 6400/60000]
loss: 0.005900  [12800/60000]
loss: 0.002995  [19200/60000]
loss: 0.002624  [25600/60000]
loss: 0.002543  [32000/60000]
loss: 0.001424  [38400/60000]
loss: 0.003861  [44800/60000]
loss: 0.004279  [51200/60000]
loss: 0.003665  [57600/60000]

 accuracy: 98.2%
 test loss: 0.062646

------------------------------
training epoch 28…
loss: 0.004368  [    0/60000]
```

```
loss: 0.004423  [ 6400/60000]
loss: 0.005772  [12800/60000]
loss: 0.002903  [19200/60000]
loss: 0.002609  [25600/60000]
loss: 0.002465  [32000/60000]
loss: 0.001392  [38400/60000]
loss: 0.003773  [44800/60000]
loss: 0.003996  [51200/60000]
loss: 0.003551  [57600/60000]

 accuracy: 98.2%
 test loss: 0.062430

------------------------------
training epoch 29…
loss: 0.004047  [    0/60000]
loss: 0.004239  [ 6400/60000]
loss: 0.005507  [12800/60000]
loss: 0.002931  [19200/60000]
loss: 0.002498  [25600/60000]
loss: 0.002472  [32000/60000]
loss: 0.001363  [38400/60000]
loss: 0.003699  [44800/60000]
loss: 0.003883  [51200/60000]
loss: 0.003444  [57600/60000]

 accuracy: 98.2%
 test loss: 0.062208

------------------------------
training epoch 30…
loss: 0.004020  [    0/60000]
loss: 0.004055  [ 6400/60000]
loss: 0.005384  [12800/60000]
loss: 0.002778  [19200/60000]
loss: 0.002439  [25600/60000]
loss: 0.002451  [32000/60000]
loss: 0.001339  [38400/60000]
loss: 0.003662  [44800/60000]
loss: 0.003669  [51200/60000]
loss: 0.003378  [57600/60000]

 accuracy: 98.2%
 test loss: 0.062123

------------------------------
training epoch 31…
loss: 0.003790  [    0/60000]
```

```
loss: 0.003961  [ 6400/60000]
loss: 0.005179  [12800/60000]
loss: 0.002810  [19200/60000]
loss: 0.002358  [25600/60000]
loss: 0.002444  [32000/60000]
loss: 0.001290  [38400/60000]
loss: 0.003612  [44800/60000]
loss: 0.003677  [51200/60000]
loss: 0.003274  [57600/60000]

 accuracy: 98.2%
 test loss: 0.061778

------------------------------
training epoch 32…
loss: 0.003727  [    0/60000]
loss: 0.003869  [ 6400/60000]
loss: 0.005105  [12800/60000]
loss: 0.002740  [19200/60000]
loss: 0.002293  [25600/60000]
loss: 0.002421  [32000/60000]
loss: 0.001292  [38400/60000]
loss: 0.003517  [44800/60000]
loss: 0.003611  [51200/60000]
loss: 0.003245  [57600/60000]

 accuracy: 98.2%
 test loss: 0.061663

------------------------------
training epoch 33…
loss: 0.003551  [    0/60000]
loss: 0.003762  [ 6400/60000]
loss: 0.004998  [12800/60000]
loss: 0.002709  [19200/60000]
loss: 0.002258  [25600/60000]
loss: 0.002420  [32000/60000]
loss: 0.001271  [38400/60000]
loss: 0.003507  [44800/60000]
loss: 0.003464  [51200/60000]
loss: 0.003234  [57600/60000]

 accuracy: 98.2%
 test loss: 0.061411

------------------------------
training epoch 34…
loss: 0.003439  [    0/60000]
```

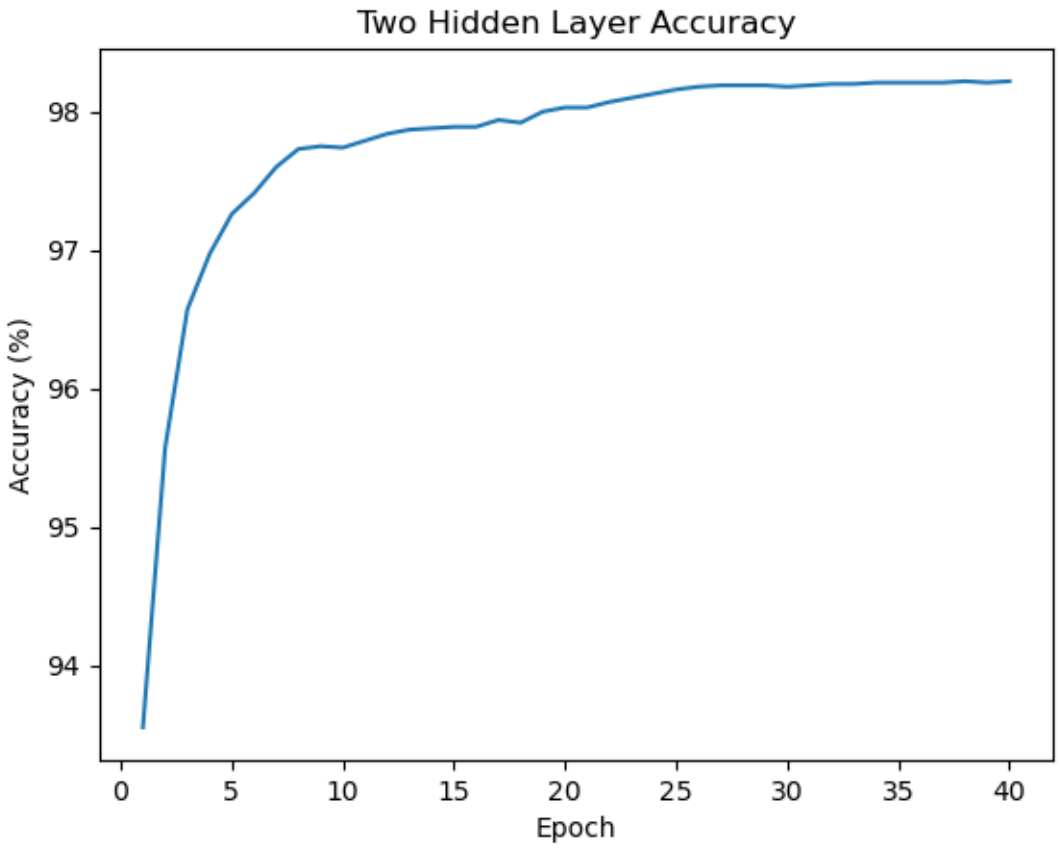
```
loss: 0.003699  [ 6400/60000]
loss: 0.004899  [12800/60000]
loss: 0.002748  [19200/60000]
loss: 0.002208  [25600/60000]
loss: 0.002471  [32000/60000]
loss: 0.001247  [38400/60000]
loss: 0.003405  [44800/60000]
loss: 0.003459  [51200/60000]
loss: 0.003145  [57600/60000]

 accuracy: 98.2%
 test loss: 0.061361

------------------------------
training epoch 35…
loss: 0.003344  [    0/60000]
loss: 0.003653  [ 6400/60000]
loss: 0.004764  [12800/60000]
loss: 0.002711  [19200/60000]
loss: 0.002208  [25600/60000]
loss: 0.002448  [32000/60000]
loss: 0.001240  [38400/60000]
loss: 0.003436  [44800/60000]
loss: 0.003339  [51200/60000]
loss: 0.003133  [57600/60000]

 accuracy: 98.2%
 test loss: 0.061101

------------------------------
training epoch 36…
loss: 0.003267  [    0/60000]
loss: 0.003483  [ 6400/60000]
loss: 0.004739  [12800/60000]
loss: 0.002667  [19200/60000]
loss: 0.002142  [25600/60000]
loss: 0.002486  [32000/60000]
loss: 0.001236  [38400/60000]
loss: 0.003356  [44800/60000]
loss: 0.003391  [51200/60000]
loss: 0.003109  [57600/60000]

 accuracy: 98.2%
 test loss: 0.060975

------------------------------
training epoch 37…
loss: 0.003136  [    0/60000]
```

```
loss: 0.003465  [ 6400/60000]
loss: 0.004664  [12800/60000]
loss: 0.002683  [19200/60000]
loss: 0.002116  [25600/60000]
loss: 0.002464  [32000/60000]
loss: 0.001218  [38400/60000]
loss: 0.003336  [44800/60000]
loss: 0.003242  [51200/60000]
loss: 0.003124  [57600/60000]

 accuracy: 98.2%
 test loss: 0.060811

------------------------------
training epoch 38…
loss: 0.003154  [    0/60000]
loss: 0.003453  [ 6400/60000]
loss: 0.004500  [12800/60000]
loss: 0.002634  [19200/60000]
loss: 0.002097  [25600/60000]
loss: 0.002496  [32000/60000]
loss: 0.001216  [38400/60000]
loss: 0.003329  [44800/60000]
loss: 0.003270  [51200/60000]
loss: 0.003113  [57600/60000]

 accuracy: 98.2%
 test loss: 0.060566

------------------------------
training epoch 39…
loss: 0.003088  [    0/60000]
loss: 0.003361  [ 6400/60000]
loss: 0.004461  [12800/60000]
loss: 0.002653  [19200/60000]
loss: 0.002026  [25600/60000]
loss: 0.002516  [32000/60000]
loss: 0.001212  [38400/60000]
loss: 0.003282  [44800/60000]
loss: 0.003179  [51200/60000]
loss: 0.003070  [57600/60000]

 accuracy: 98.2%
 test loss: 0.060467

------------------------------
training epoch 40…
loss: 0.003024  [    0/60000]
```

```
loss: 0.003369  [ 6400/60000]
loss: 0.004407  [12800/60000]
loss: 0.002614  [19200/60000]
loss: 0.001981  [25600/60000]
loss: 0.002479  [32000/60000]
loss: 0.001200  [38400/60000]
loss: 0.003276  [44800/60000]
loss: 0.003159  [51200/60000]
loss: 0.003059  [57600/60000]

 accuracy: 98.2%
 test loss: 0.060361

--------------------------------
training complete, model saved
```

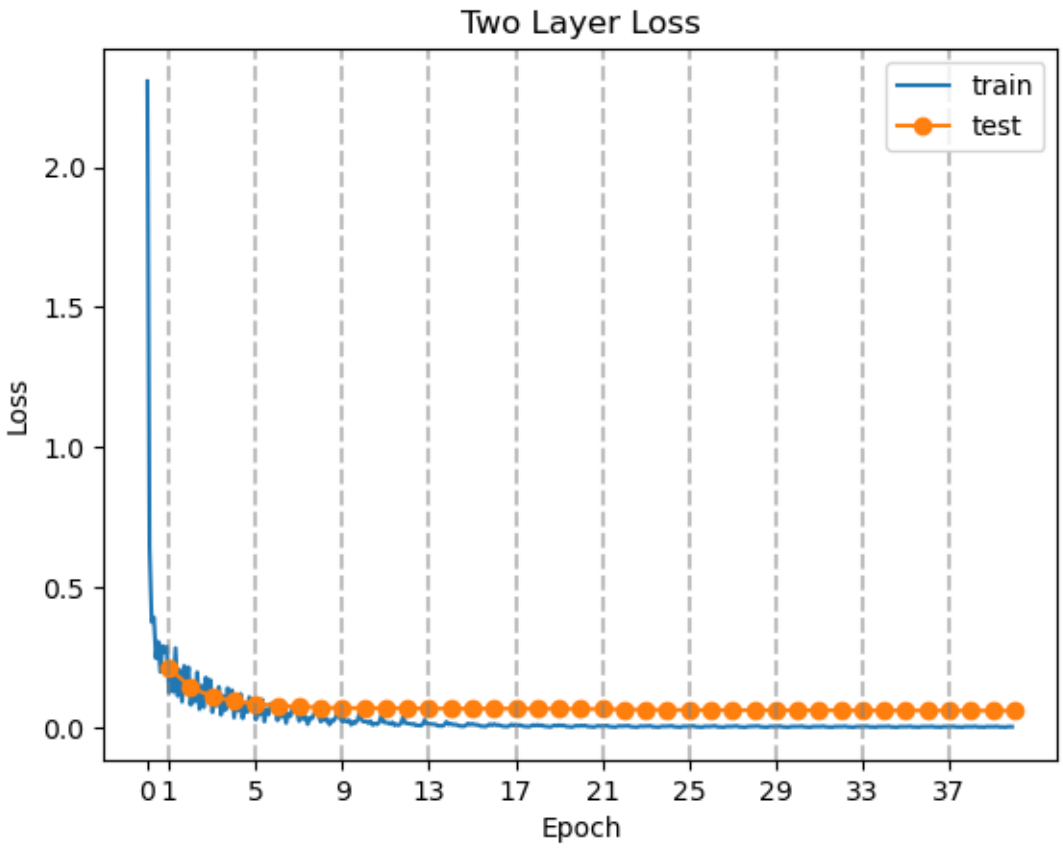
Saving and plotting the accuracy data:

[251]:
```
save_accuracy_data(
        "models/two_layer_accuracy.csv",
        accuracy_list
)
plt.plot(epochs, accuracy_list)
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.title("Two Hidden Layer Accuracy")
plt.savefig(
        "figures/two_layer_accuracy.png")
```



Plotting the training and test loss:

[252]:
```
plot_loss(
        train_loss, test_loss, epochs,
        "Two Layer Loss",
        "two_layer_loss.png"
)
```



5 Problem 4: Convolutional Neural Network

5.1 Designing the model

Creating a convolutional neural network model with two convolutional layers and two fully connected layers.

Reusing the CrossEntropy as loss function and reusing the SGD with L2 regularization as optimizer.

Reusing the device definition:

[253]:
```
class CNN(nn.Module):
        def __init__(self):
                super(CNN, self).__init__()
                self.conv_block = nn.Sequential(
                        nn.Conv2d(
                                1, 32,
                                kernel_size=3, stride=1, padding=1
                        ),
                        nn.BatchNorm2d(32),
                        nn.ReLU(inplace=True),
```

```
                        nn.Conv2d(
                                32, 64,
                                kernel_size=3, stride=1, padding=1
                        ),
                        nn.BatchNorm2d(64),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(kernel_size=2, stride=2),
                        nn.Conv2d(
                                64, 128,
                                kernel_size=3, stride=1, padding=1
                        ),
                        nn.BatchNorm2d(128),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(kernel_size=2, stride=2)
                )

                self.linear_block = nn.Sequential(
                        nn.Dropout(p=0.5),
                        nn.Linear(128 * 7 * 7, 128),
                        nn.BatchNorm1d(128),
                        nn.ReLU(inplace=True),
                        nn.Dropout(0.5),
                        nn.Linear(128, 64),
                        nn.BatchNorm1d(64),
                        nn.ReLU(inplace=True),
                        nn.Dropout(0.5),
                        nn.Linear(64, 10)
                )

        def forward(self, x):
                x = self.conv_block(x)
                x = x.view(x.size(0), -1)
                x = self.linear_block(x)
                return x


model = CNN().to(device)
```

[254]:
```
optimizer = torch.optim.SGD(
        model.parameters(),
        lr=0.01,
        weight_decay=1e-4
)
```

5.2 Training and evaluating the model

Train the model for 40 epochs and evaluate its performance:

[255]:
```
epochs = range(1, 41)

train_loss = []
test_loss = []
accuracy_list = []
for t in epochs:
        print(f"training epoch {t}...")

        epoch_train_loss = train(
                train_loader, model, loss_fn, optimizer,
        )
        accuracy, epoch_test_loss = test(
                test_loader, model, loss_fn
        )

        train_loss.extend(epoch_train_loss)
        test_loss.append(epoch_test_loss)
        accuracy_list.append(accuracy)

        print(f"-------------------------------")

torch.save(
        model.state_dict(),
        "models/cnn.pth"
)
print("training complete, model saved")
```

```
training epoch 1…

 accuracy: 97.4%
 test loss: 0.142887

-------------------------------
training epoch 2…

 accuracy: 98.4%
 test loss: 0.068661

-------------------------------
training epoch 3…

 accuracy: 98.6%
 test loss: 0.051451

-------------------------------
training epoch 4…

 accuracy: 98.8%
```

```
 test loss: 0.039866

------------------------------
training epoch 5…

 accuracy: 99.0%
 test loss: 0.034570

------------------------------
training epoch 6…

 accuracy: 99.0%
 test loss: 0.033022

------------------------------
training epoch 7…

 accuracy: 99.1%
 test loss: 0.028491

------------------------------
training epoch 8…

 accuracy: 99.1%
 test loss: 0.027911

------------------------------
training epoch 9…

 accuracy: 99.2%
 test loss: 0.026241

------------------------------
training epoch 10…

 accuracy: 99.1%
 test loss: 0.027570

------------------------------
training epoch 11…

 accuracy: 99.2%
 test loss: 0.025265

------------------------------
training epoch 12…

 accuracy: 99.2%
```

```
 test loss: 0.023863

------------------------------
training epoch 13…

 accuracy: 99.3%
 test loss: 0.022900

------------------------------
training epoch 14…

 accuracy: 99.3%
 test loss: 0.021457

------------------------------
training epoch 15…

 accuracy: 99.3%
 test loss: 0.021744

------------------------------
training epoch 16…

 accuracy: 99.3%
 test loss: 0.022095

------------------------------
training epoch 17…

 accuracy: 99.4%
 test loss: 0.020799

------------------------------
training epoch 18…

 accuracy: 99.3%
 test loss: 0.021223

------------------------------
training epoch 19…

 accuracy: 99.4%
 test loss: 0.019494

------------------------------
training epoch 20…

 accuracy: 99.4%
```

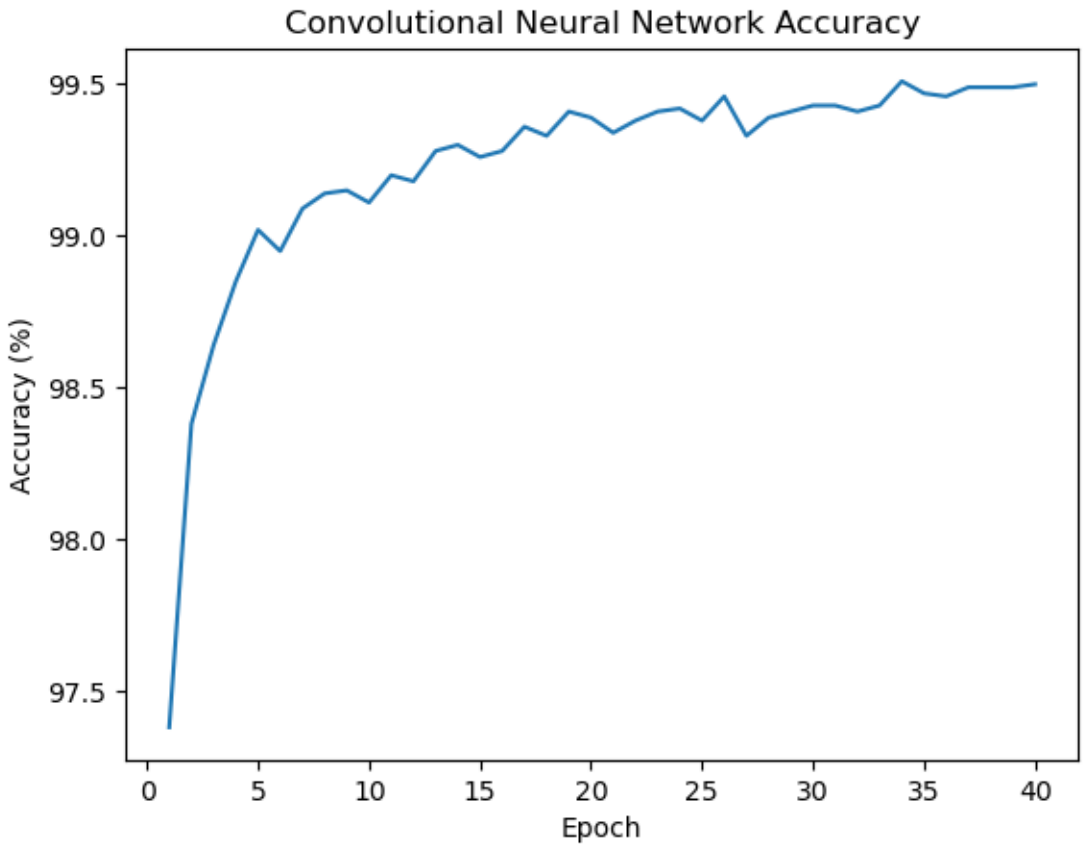
```
 test loss: 0.019845

------------------------------
training epoch 21…

 accuracy: 99.3%
 test loss: 0.019796

------------------------------
training epoch 22…

 accuracy: 99.4%
 test loss: 0.019206

------------------------------
training epoch 23…

 accuracy: 99.4%
 test loss: 0.018460

------------------------------
training epoch 24…

 accuracy: 99.4%
 test loss: 0.018835

------------------------------
training epoch 25…

 accuracy: 99.4%
 test loss: 0.019072

------------------------------
training epoch 26…

 accuracy: 99.5%
 test loss: 0.018342

------------------------------
training epoch 27…

 accuracy: 99.3%
 test loss: 0.020868

------------------------------
training epoch 28…

 accuracy: 99.4%
```

```
 test loss: 0.020156

------------------------------
training epoch 29…

 accuracy: 99.4%
 test loss: 0.018249

------------------------------
training epoch 30…

 accuracy: 99.4%
 test loss: 0.017419

------------------------------
training epoch 31…

 accuracy: 99.4%
 test loss: 0.017828

------------------------------
training epoch 32…

 accuracy: 99.4%
 test loss: 0.018062

------------------------------
training epoch 33…

 accuracy: 99.4%
 test loss: 0.018124

------------------------------
training epoch 34…

 accuracy: 99.5%
 test loss: 0.017239

------------------------------
training epoch 35…

 accuracy: 99.5%
 test loss: 0.016532

------------------------------
training epoch 36…

 accuracy: 99.5%
```

```
 test loss: 0.017117

--------------------------------
training epoch 37…

 accuracy: 99.5%
 test loss: 0.017294

--------------------------------
training epoch 38…

 accuracy: 99.5%
 test loss: 0.017370

--------------------------------
training epoch 39…

 accuracy: 99.5%
 test loss: 0.016511

--------------------------------
training epoch 40…

 accuracy: 99.5%
 test loss: 0.016606

--------------------------------
training complete, model saved
```

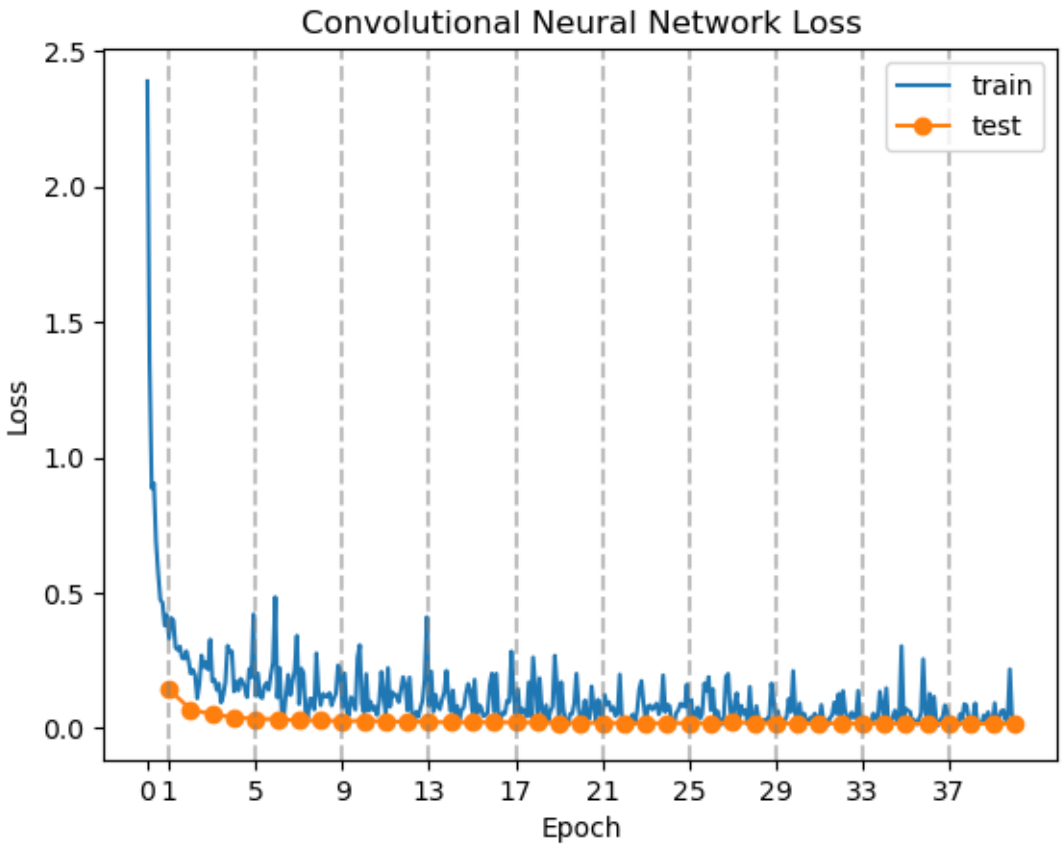
Saving and plotting the accuracy data:

[256]:
```
save_accuracy_data("models/cnn_accuracy.csv",
                   accuracy_list)
plt.plot(epochs, accuracy_list)
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.title("Convolutional Neural Network Accuracy")
plt.savefig("figures/cnn_accuracy.png")
```



Plotting the training and test loss:

[257]:
```
plot_loss(
        train_loss, test_loss, epochs,
        "Convolutional Neural Network Loss",
        "cnn_loss.png"
)
```


Convolutional Neural Network Loss
train
test
Loss
Epoch
2.5
2.0
1.5
1.0
0.5
0.0
0 1
5
9
13
17
21
25
29
33
37