Assignment 1

Thomas Seeberg Christiansen

October 2, 2024

1 Part 1

1.1 a) In your opinion, what were the most important turning points in the history of deep learning?

The most important turning points for deep learning, would in my opinion, be the computational increase with the usage of GPUs, the usage of backpropagation, and in the recent years the use of transformers.
The development of faster hardware meant that training could be done at a much faster rate and larger datasets could be utilized. AlexNet was one of the cornerstones in the GPU boom by using two GTX 580's GPUs training a convolutional neural network.
An earlier turning point, in the 1980s, was the formation of backpropagation which would allow the networks to learn from their predictions errors and adjust the weigths accordingly.
Lastly, in recent years, the advancement of transformer-based architecture has given a new boost to deep learning, especially leading to the development of generative pre-trained transformers (GPTs), as seen in models like "ChatGPT".

1.2 b) Explain the ADAM optimizer.

The Adaptive Moment Estimation (ADAM) optimizer is used to update the weights in a network by combining the ideas of momentum and adaptive learning rates. It works by maintaining two moving averages: one for the gradient (momentum term) and another for the squared gradient (RMSprop term). These averages are bias-corrected, and weights are updated using both terms to adapt the learning rate for each parameter.

1.3 c) Assume data input is a single 30x40 pixel image. First layer is a convolutional layer with 5 filters, with kernel size 3x2, step size (1,1) and padding='valid'. What are the output dimensions?

The output dimensions is given by

$$\frac{W - K + 2P}{S} + 1$$

where W is the input volume size, K is the kernel size, S the stride, and P the amount of padding. So we have the following:

$$\text{Output Height} = \frac{W - K + 2P}{S} + 1 = \frac{30 - 3 + 2 \cdot 0}{1} + 1 = \frac{27}{1} + 1 = 28$$

$$\text{Output Width} = \frac{W - K + 2P}{S} + 1 = \frac{40 - 2 + 2 \cdot 0}{1} + 1 = \frac{38}{1} + 1 = 39$$

and since the convolutional layer has 5 filters, the output will have 5 channels. Hence, the output dimensions will be 28x39x5.

1.4 d) Assuming ReLU activations and offsets, and that the last layer is softmax, how many parameters does this network have:

The number of parameters is calculated by looking at the number of layers and number of neurons in each layer. We have 1 input layer, 3 hidden layers, and 1 output layer. Since its a fully connected network the 5 input neurons connect with each of the 1st hidden layer neuron, and so on. Hence, we get the following weights

$$\text{Weights} = 5 \times 5 + 5 \times 5 + 5 \times 5 + 5 \times 3 = 90$$

We, also, need to look at the biases present for each neuron after the input layer.

$$\text{Biases} = 5 + 5 + 5 + 3 = 18$$

In total we have the following number of parameters:

$$\text{Total} = 90 + 18 = 109$$

1.5 e) For a given minibatch, the targets are [1,4, 5, 8] and the network output is [0.1,4.4,0.2,10]. If the loss function is "torch.nn.HuberLoss(reduction='mean', delta=1.0)", what is the loss for this minibatch?

The Huber loss function is defined by the following piecewise

$$L_\delta(y - \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & |y - \hat{y}| \leq \delta \\ \delta(|y - \hat{y}| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases}$$

where y is the target, $\hat{y}$ is the predicted output, δ the threshold. With the targets $y = [1, 4, 5, 8]$ and predicted outputs $\hat{y} = [0.1, 4.4, 0.2, 10]$, we can compute the loss for each. First we determine each of the cases.

$$|y - \hat{y}| = |1 - 0.1| = 0.9$$
$$|y - \hat{y}| = |4 - 4.4| = 0.4$$
$$|y - \hat{y}| = |5 - 0.2| = 4.8$$
$$|y - \hat{y}| = |8 - 10| = 2.0$$

then the losses

$$L(1, 0.1) = \frac{1}{2}(1 - 0.1)^2 = 0.405$$
$$L(4, 4.4) = \frac{1}{2}(4 - 4.4)^2 = 0.08$$
$$L(5, 0.2) = 1 \cdot (|5 - 0.2| - \frac{1}{2} \cdot 1) = 4.3$$
$$L(8, 10) = 1 \cdot (|8 - 10| - \frac{1}{2} \cdot 1) = 1.5$$

In total we have the mean loss as

$$\frac{0.405 + 0.08 + 4.3 + 1.5}{4} = 1.57$$

This can also be done with torch by the following code:

[1]:
```python
import torch
# Targets and outputs
targets = torch.tensor([1, 4, 5, 8])
outputs = torch.tensor([0.1, 4.4, 0.2, 10])

# Define the Huber loss with delta = 1.0
huber_loss = torch.nn.HuberLoss(reduction='mean', delta=1.0)

# Calculate the loss
loss = huber_loss(outputs, targets)
print(loss.item())
```

```
1.571250081062317
```

2 Part 2

[2]:
```python
import os
import pandas as pd
from torchvision.io import read_image
import numpy as np
import torch
from torch import nn
from torch.utils.data import DataLoader
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt
from torchsummary import summary
import torch_directml
from PIL import Image
```

2.1 Dataset builder

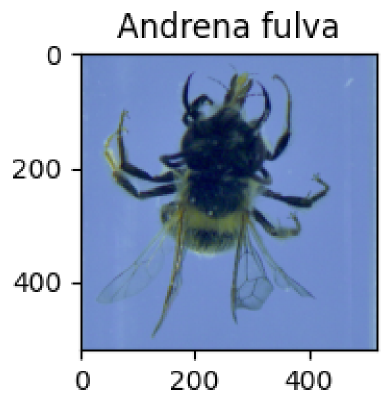
[3]:
```python
class InsectDataset(Dataset):
    def __init__(self, annotations_file, img_dir, root_dir, transform=None):
        """
        directory setup of the images and labels
        root_dir: Main data directory
        annotations_file: csv file
        img_dir: image directory
        """
        self.root_dir = root_dir
        annotations_path =  os.path.join(self.root_dir, annotations_file)
        self.img_labels = pd.read_csv(annotations_path)
        self.img_dir = img_dir
```

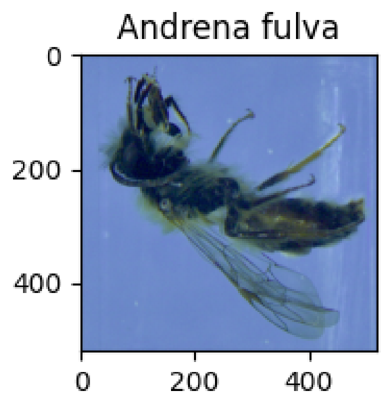
```python
        self.transform = transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        """
        Retrieve image via filename
        Open image
        Retrieve label
        transform if needed
        return image and label
        """
        img_path = os.path.join(self.root_dir,self.img_dir, self.img_labels.
 ↪iloc[idx, 2])
        # print(f"Loading image from: {img_path}")    # Debug print
        image = Image.open(img_path)

        label = self.img_labels.iloc[idx, 1]        # Retrieve label
        # print(f"----------Labelled = {label}\n")    # Debug print
        if self.transform:
            image = self.transform(image)

        return image, label
```

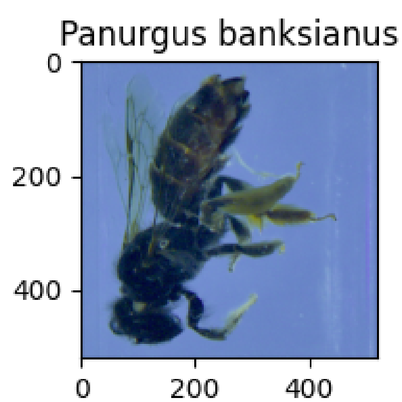
[4]:

```python
transform = transforms.Compose([
    transforms.Resize((520,520)),
    transforms.ToTensor()])

batch_size = 4

# Set up the dataset.
dataset = InsectDataset(annotations_file='insects.csv',␣
 ↪img_dir='Insects',root_dir='Data/', transform=transform)

# Set up the dataset.
trainloader = torch.utils.data.DataLoader(dataset,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          num_workers=0)

# get some images
dataiter = iter(trainloader)
images, labels = next(dataiter)

for i in range(5): #Run through 5 batches
    images, labels = next(dataiter)
```

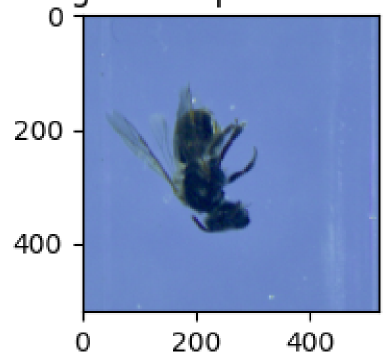
```
    for image, label in zip(images,labels): # Run through all samples in a batch
        plt.figure(figsize=(4, 2))
        plt.imshow(np.transpose(image.numpy(), (1, 2, 0)))
        plt.title(label)
```



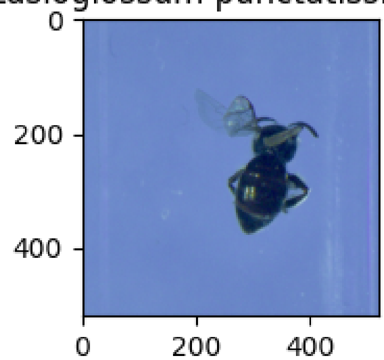




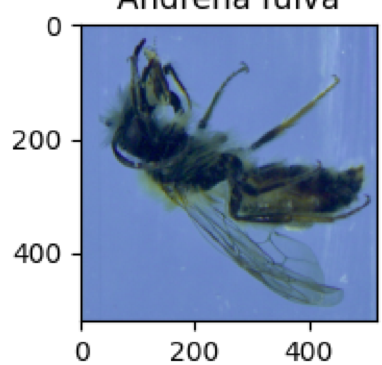
Lasioglossum punctatissimum

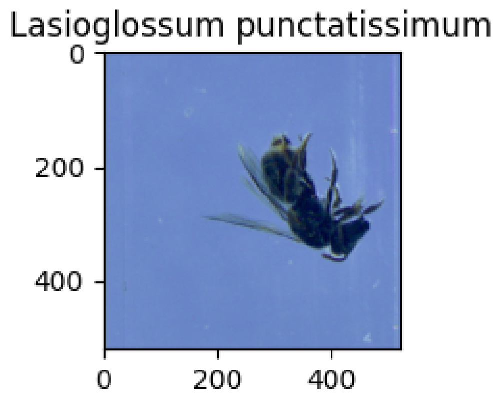


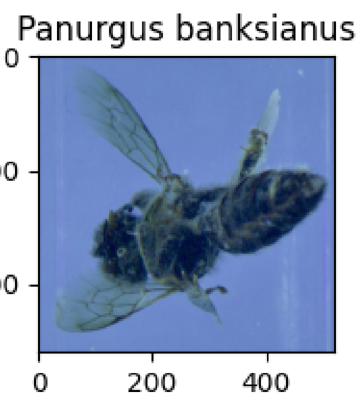
Lasioglossum punctatissimum

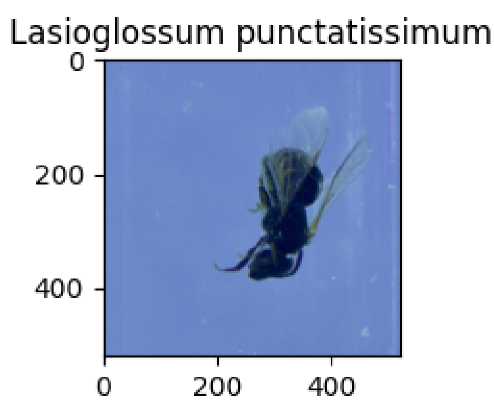


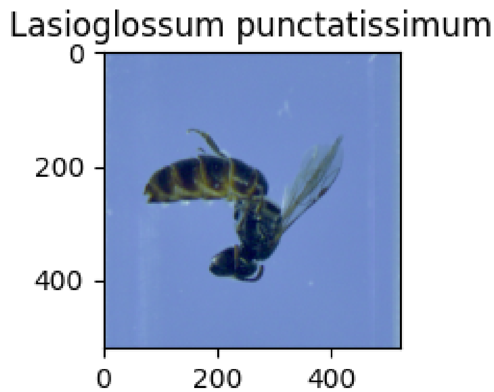
Andrena fulva

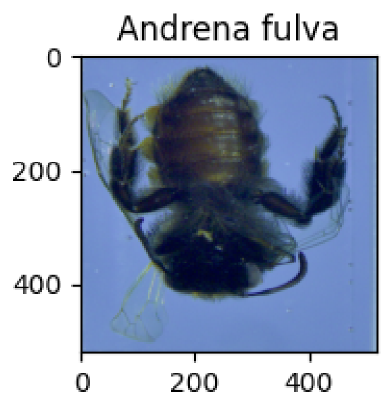


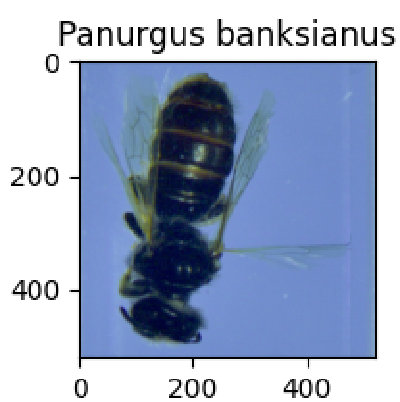

Lasioglossum punctatissimum
0
200
400
0
200
400

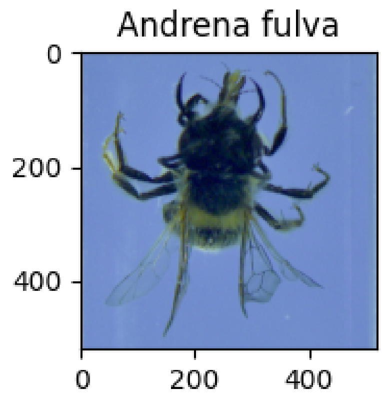

Panurgus banksianus
0
200
400
0
200
400

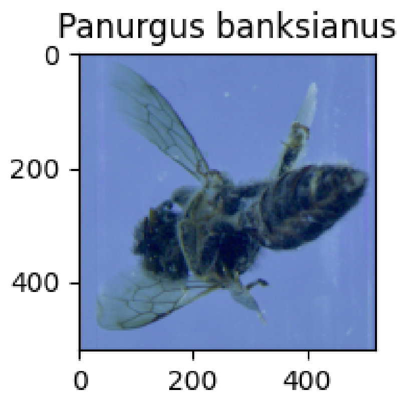

Lasioglossum punctatissimum
0
200
400
0
200
400

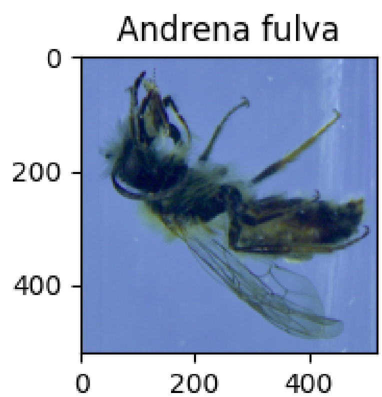

Lasioglossum punctatissimum

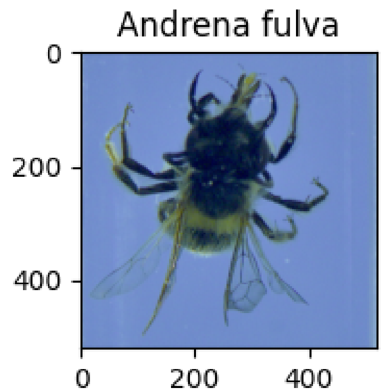

Andrena fulva

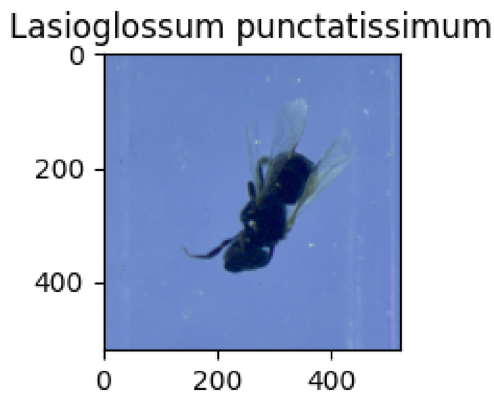

Panurgus banksianus

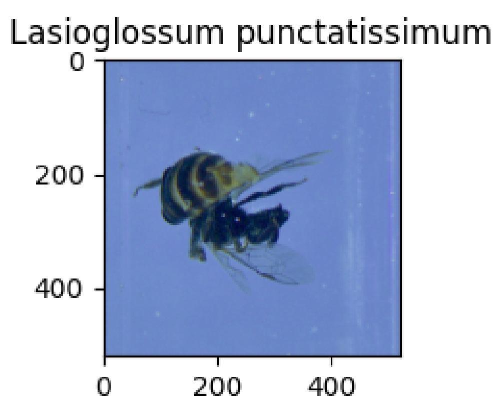

Andrena fulva
0
200
400
0
200
400

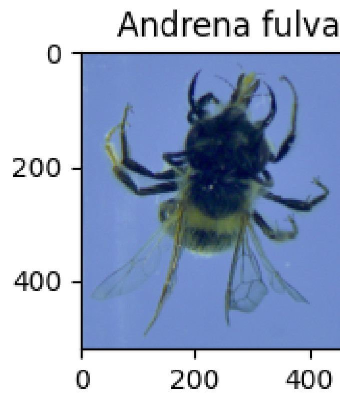

Panurgus banksianus
0
200
400
0
200
400

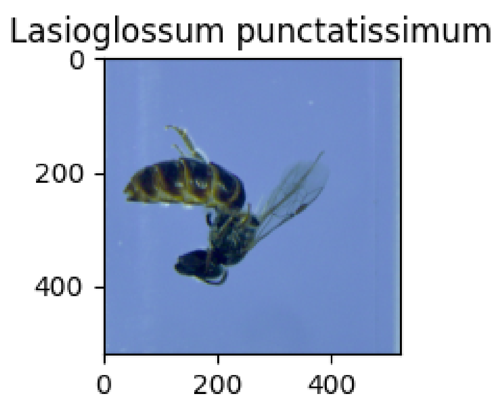

Andrena fulva
0
200
400
0
200
400


Andrena fulva
0
200
400
0
200
400


Lasioglossum punctatissimum
0
200
400
0
200
400


Lasioglossum punctatissimum
0
200
400
0
200
400





3 Part 3

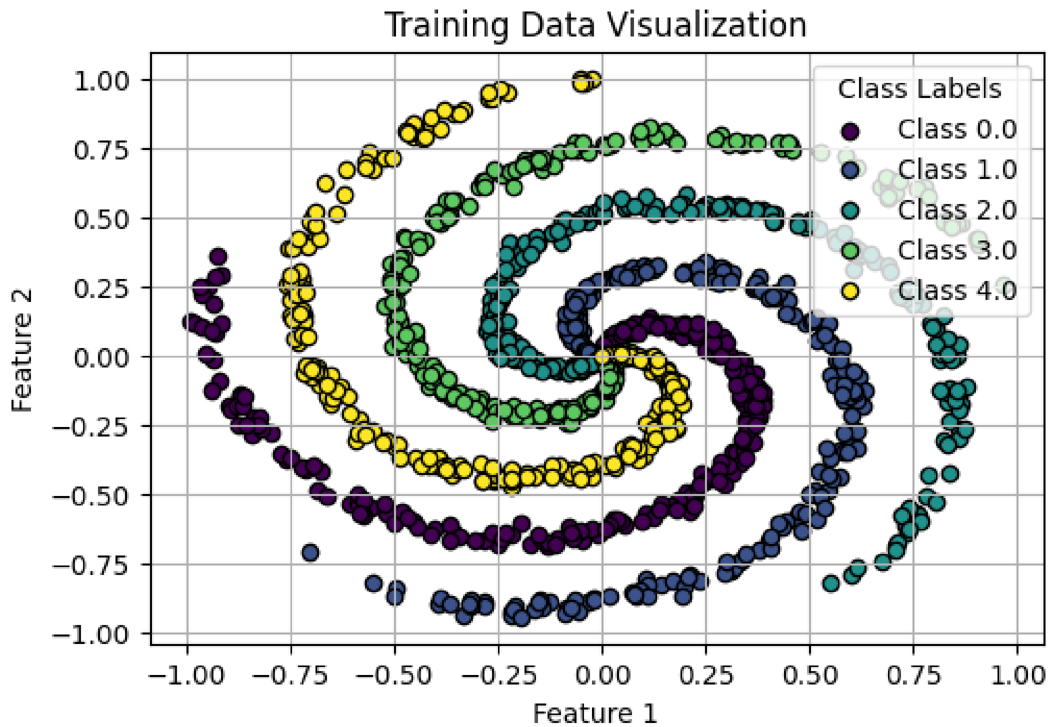
3.1 a)

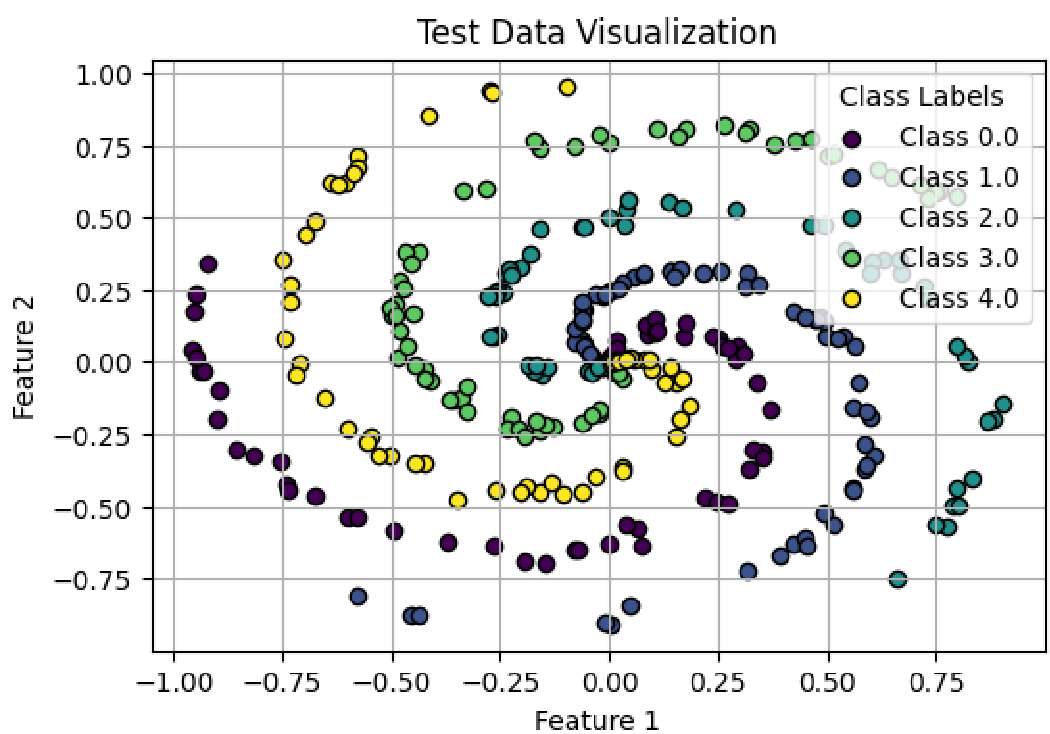
The data given is already described and I, therefore, just go straight ahead and extract the features and labels from both the trainData and testData ".txt" files. This is easily done using pandas.

[5]:
```
import pandas as pd

# Load the data from .txt files
train_data = pd.read_csv('Data/Part3/trainData.txt', sep='\s+', header=None)
test_data = pd.read_csv('Data/Part3/testData.txt', sep='\s+', header=None)

# The first column is the label, and the second and third columns are the␣
↪features
train_labels = train_data.iloc[:, 0]  # Labels (first column)
train_features = train_data.iloc[:, 1:]  # Features (second and third columns)
```

```
test_labels = test_data.iloc[:, 0]  # Labels (first column)
test_features = test_data.iloc[:, 1:]  # Features (second and third columns)

# Print the first few rows to verify the data
print(train_data.head())

# Define a function to visualize the data
def plot_data(features, labels, title):
    plt.figure(figsize=(6, 4))
    unique_labels = np.unique(labels)  # Get unique class labels
    colors = plt.cm.viridis(np.linspace(0, 1, len(unique_labels)))  # Generate␣
 ↪colors

    # Plot each class with its own color and label
    for label, color in zip(unique_labels, colors):
        idx = labels.to_numpy() == label  # Get indices of the current label
        plt.scatter(features.iloc[idx, 0], features.iloc[idx, 1], label=f'Class␣
 ↪{label}', color=color, edgecolor='k')

    plt.title(title)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend(title="Class Labels",loc="upper right")
    plt.grid(True)
    plt.show()

# Plot the training data
plot_data(train_features, train_labels, "Training Data Visualization")

# Plot the test data
plot_data(test_features, test_labels, "Test Data Visualization")
```

```
     0         1         2
0  2.0  0.243584  0.539536
1  0.0  0.029800  0.074531
2  4.0 -0.437585 -0.383632
3  2.0 -0.224602  0.407026
4  3.0  0.284853  0.800316
```


Training Data Visualization
Class Labels
Class 0.0
Class 1.0
Class 2.0
Class 3.0
Class 4.0
Feature 2
Feature 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


Test Data Visualization
Class Labels
Class 0.0
Class 1.0
Class 2.0
Class 3.0
Class 4.0
Feature 2
Feature 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75

The plots then shows a spiraling arms like a galaxy. We Can easily see the datapoints being classified with their corresponding labels.

3.2 b)

- **Describe your network**
  The network need to take the two inputs and expand them for more parameters before shrieking to the 5 classes. Therefore, a number of linear layers is needed starting with a input layer taking 2 input features and outputting 256, then a hidden from 256 to 128 and 128 to 64, and then a output layer going from 64 to the 5 classes. Each layer is being followed by a ReLU activation function.
- **Describe your training strategy**
  The network is trained by the Adam optimizer and Cross Entropy loss function with appropriate values for learning rate and batch size parameters. This strategy was build using our prior code from week 3 and 4 where we have seen how certain optimizers and loss functions behave and that the Adam optimizer and Cross Entropy loss function often is the prefered choice due to being efficent.
  Training is performed in batches of 32 with a learning rate of 0.001 running for 100 epochs or until 10 consecutive epochs with no improvement in the loss value.

[6]:
```
class NeuralNet(nn.Module):
    def __init__(self, input_size, number_classes):
        super().__init__()

        # Linear Layers
        self.linear1 = nn.Linear(input_size, 256)
        self.linear2 = nn.Linear(256, 128)
        self.linear3 = nn.Linear(128, 64)
        self.linear4 = nn.Linear(64, number_classes)

        # Activation Function
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.linear1(x)
        x = self.relu(x)
        x = self.linear2(x)
        x = self.relu(x)
        x = self.linear3(x)
        x = self.relu(x)
        x = self.linear4(x)
        return x

# Hyperparameters
input_size = 2
number_classes = 5
batch_size = 32
lr = 0.001
```

```python
# Device
device = "cpu"

# Load the training/test data/labels to tensor's
training_data = torch.tensor(train_features.values, dtype=torch.float32)
training_labels = torch.tensor(train_labels.values, dtype=torch.long)
testing_data = torch.tensor(test_features.values, dtype=torch.float32)
testing_labels = torch.tensor(test_labels.values, dtype=torch.long)

# Combine to a dataset
train_dataset = torch.utils.data.TensorDataset(training_data, training_labels)
test_dataset = torch.utils.data.TensorDataset(testing_data, testing_labels)

# Load DataLoader
train_dataloader = DataLoader(train_dataset, batch_size=batch_size,␣
↪shuffle=True,pin_memory=True)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size,␣
↪shuffle=True,pin_memory=True)

# Neural Network
model = NeuralNet(input_size, number_classes).to(device)

# Optimizer and Loss
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)

class trainNN:

    def __init__(self, model, train_dataloader, test_dataloader, loss_fn,␣
↪optimizer, num_epochs, patience):
        """
        Inputs
        """
        self.model = model
        self.train_dataloader = train_dataloader
        self.test_dataloader = test_dataloader
        self.loss_fn = loss_fn
        self.optimizer = optimizer
        self.num_epochs = num_epochs
        self.patience = patience
        self.reset()

    def reset(self):
        """
        reset function. This includes weigths and parameters.
        """
```

```python
        self.train_losses = []
        self.test_losses = []
        self.train_accuracies = []
        self.test_accuracies = []
        self.number_epochs = 1

        def reset_weights(m):
            if hasattr(m, 'reset_parameters'):
                m.reset_parameters()
        self.model.apply(reset_weights)

        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=lr)

    def _train(self, dataloader, model, loss_fn, optimizer):
        """
        Training function
        """
        size = len(dataloader.dataset)
        model.train()
        total_loss, correct = 0, 0  # Track the total loss for the epoch

        for batch, (X, y) in enumerate(dataloader):
            X, y = X.to(device), y.to(device)

            # Compute prediction error
            pred = model(X)
            loss = loss_fn(pred, y)
            total_loss += loss.item()  # Accumulate loss

            # Backpropagation
            model.zero_grad()
            loss.backward()
            optimizer.step()

            # Calculate accuracy
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

            # Print loss for the first batch and then every 5th batch
            if batch == 0 or (batch + 1) % 5 == 0 or (batch + 1) == len(dataloader):
                current = (batch + 1) * len(X) if (batch + 1) < len(dataloader) else size
                print(f"loss: {loss.item():>7f}  [{current:>5d}/{size:>5d}]")
        avg_loss = total_loss / len(dataloader)  # Calculate average loss for the epoch
        avg_accuracy = correct / size
```

```
        self.train_losses.append(avg_loss)  # Store the average loss for the␣
↪epoch
        self.train_accuracies.append(avg_accuracy) # Store the average accuracy␣
↪for the epoch

        print(f"Train loss: {avg_loss:>7f}, Accuracy: {(100*avg_accuracy):>0.
↪1f}%")

    def _test(self, dataloader, model, loss_fn):
        """
        Test function
        """
        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
        correct /= size
        self.test_losses.append(test_loss)  # Store the test loss for the epoch
        self.test_accuracies.append(correct)

        print(f"Test Error: \n Avg loss: {test_loss:>8f}, Accuracy:␣
↪{(100*correct):>0.1f}% \n")
        return round(test_loss,3)

    def _plot_results(self):
        """
        Plotting function
        """
        plt.subplot(2,1,1)
        plt.plot(range(1, self.number_epochs+1), self.train_losses,␣
↪label="Train Loss")
        plt.plot(range(1, self.number_epochs+1), self.test_losses, label="Test␣
↪Loss")
        plt.xlabel("Epochs")
        plt.ylabel("Loss")
        plt.title("Training and Testing Loss over Epochs")
        plt.legend()
        plt.show()
```

```
        plt.subplot(2,1,2)
        plt.plot(range(1, self.number_epochs+1), self.train_accuracies,␣
↪label="Train Accuracy")
        plt.plot(range(1, self.number_epochs+1), self.test_accuracies,␣
↪label="Test Accuracy")
        plt.xlabel("Epochs")
        plt.ylabel("Accuracy")
        plt.title("Training and Testing Accuracy over Epochs")
        plt.legend()
        plt.show()

    def TrainModel(self):
        """
        Execute training function
        """
        # Early stopping parameters
        best_test_loss = float('inf')
        epochs_without_improvement = 0

        for t in range(self.num_epochs):
            print(f"Epoch {t+1}\n-------------------------------")
            # Print number of batches only for the first epoch
            if t == 0:
                num_batches = len(self.train_dataloader)
                print(f"Number of batches: {num_batches}")
            self._train(train_dataloader, model, loss_fn, optimizer)
            current_test_loss = self._test(test_dataloader, model, loss_fn)

            # Early stopping logic
            if current_test_loss < best_test_loss:
                best_test_loss = current_test_loss
                epochs_without_improvement = 0
            else:
                epochs_without_improvement += 1

            if epochs_without_improvement >= self.patience:
                print(f"Early stopping triggered after {self.number_epochs+1}␣
↪epochs.")
                break
            print(f"Epochs without improvement: {epochs_without_improvement+1}")
            self.number_epochs += 1
        print("Training completed.")
        self._plot_results()
```

[7]:
```
train = trainNN(model, train_dataloader, test_dataloader, loss_fn, optimizer,␣
↪100, 10)
train.reset()
```

```
train.TrainModel()
```

```
Epoch 1
-------------------------------
Number of batches: 38
loss: 1.599170  [   32/ 1200]
loss: 1.609358  [  160/ 1200]
loss: 1.588189  [  320/ 1200]
loss: 1.576432  [  480/ 1200]
loss: 1.554649  [  640/ 1200]
loss: 1.562531  [  800/ 1200]
loss: 1.537484  [  960/ 1200]
loss: 1.516447  [ 1120/ 1200]
loss: 1.431424  [ 1200/ 1200]
Train loss: 1.545708, Accuracy: 27.3%
Test Error:
 Avg loss: 1.454326, Accuracy: 28.4%

Epochs without improvement: 1
Epoch 2
-------------------------------
loss: 1.401246  [   32/ 1200]
loss: 1.475572  [  160/ 1200]
loss: 1.464338  [  320/ 1200]
loss: 1.466062  [  480/ 1200]
loss: 1.473617  [  640/ 1200]
loss: 1.322773  [  800/ 1200]
loss: 1.429705  [  960/ 1200]
loss: 1.185124  [ 1120/ 1200]
loss: 1.306701  [ 1200/ 1200]
Train loss: 1.378204, Accuracy: 30.8%
Test Error:
 Avg loss: 1.280153, Accuracy: 31.4%

Epochs without improvement: 1
Epoch 3
-------------------------------
loss: 1.132774  [   32/ 1200]
loss: 1.412665  [  160/ 1200]
loss: 1.342972  [  320/ 1200]
loss: 1.151224  [  480/ 1200]
loss: 1.162513  [  640/ 1200]
loss: 1.209087  [  800/ 1200]
loss: 1.126050  [  960/ 1200]
loss: 1.199793  [ 1120/ 1200]
loss: 1.087925  [ 1200/ 1200]
Train loss: 1.215863, Accuracy: 37.5%
Test Error:
```

```
 Avg loss: 1.079637, Accuracy: 43.5%

Epochs without improvement: 1
Epoch 4
-------------------------------
loss: 1.158961  [   32/ 1200]
loss: 1.036284  [  160/ 1200]
loss: 1.092382  [  320/ 1200]
loss: 0.871192  [  480/ 1200]
loss: 0.982630  [  640/ 1200]
loss: 0.970542  [  800/ 1200]
loss: 0.924467  [  960/ 1200]
loss: 0.821984  [ 1120/ 1200]
loss: 0.981934  [ 1200/ 1200]
Train loss: 0.991536, Accuracy: 57.2%
Test Error:
 Avg loss: 0.815865, Accuracy: 71.6%

Epochs without improvement: 1
Epoch 5
-------------------------------
loss: 0.833152  [   32/ 1200]
loss: 0.804250  [  160/ 1200]
loss: 0.912747  [  320/ 1200]
loss: 0.753561  [  480/ 1200]
loss: 0.603806  [  640/ 1200]
loss: 0.844943  [  800/ 1200]
loss: 0.774638  [  960/ 1200]
loss: 0.546869  [ 1120/ 1200]
loss: 0.516882  [ 1200/ 1200]
Train loss: 0.722239, Accuracy: 74.3%
Test Error:
 Avg loss: 0.569290, Accuracy: 80.9%

Epochs without improvement: 1
Epoch 6
-------------------------------
loss: 0.620693  [   32/ 1200]
loss: 0.696194  [  160/ 1200]
loss: 0.423361  [  320/ 1200]
loss: 0.653524  [  480/ 1200]
loss: 0.529997  [  640/ 1200]
loss: 0.323716  [  800/ 1200]
loss: 0.497326  [  960/ 1200]
loss: 0.380399  [ 1120/ 1200]
loss: 0.527917  [ 1200/ 1200]
Train loss: 0.512587, Accuracy: 83.0%
Test Error:
```

```
 Avg loss: 0.432872, Accuracy: 86.3%

Epochs without improvement: 1
Epoch 7
-------------------------------
loss: 0.497101  [   32/ 1200]
loss: 0.503120  [  160/ 1200]
loss: 0.292819  [  320/ 1200]
loss: 0.447846  [  480/ 1200]
loss: 0.360764  [  640/ 1200]
loss: 0.348666  [  800/ 1200]
loss: 0.294143  [  960/ 1200]
loss: 0.418371  [ 1120/ 1200]
loss: 0.164051  [ 1200/ 1200]
Train loss: 0.382560, Accuracy: 85.6%
Test Error:
 Avg loss: 0.309664, Accuracy: 88.3%

Epochs without improvement: 1
Epoch 8
-------------------------------
loss: 0.416358  [   32/ 1200]
loss: 0.251955  [  160/ 1200]
loss: 0.439511  [  320/ 1200]
loss: 0.321455  [  480/ 1200]
loss: 0.308661  [  640/ 1200]
loss: 0.397789  [  800/ 1200]
loss: 0.266449  [  960/ 1200]
loss: 0.252878  [ 1120/ 1200]
loss: 0.485748  [ 1200/ 1200]
Train loss: 0.300852, Accuracy: 89.5%
Test Error:
 Avg loss: 0.270774, Accuracy: 90.0%

Epochs without improvement: 1
Epoch 9
-------------------------------
loss: 0.330293  [   32/ 1200]
loss: 0.153322  [  160/ 1200]
loss: 0.235454  [  320/ 1200]
loss: 0.291082  [  480/ 1200]
loss: 0.378107  [  640/ 1200]
loss: 0.250685  [  800/ 1200]
loss: 0.272888  [  960/ 1200]
loss: 0.201700  [ 1120/ 1200]
loss: 0.125779  [ 1200/ 1200]
Train loss: 0.238073, Accuracy: 92.1%
Test Error:
```

```
 Avg loss: 0.245069, Accuracy: 92.3%

Epochs without improvement: 1
Epoch 10
-------------------------------
loss: 0.250822  [   32/ 1200]
loss: 0.280645  [  160/ 1200]
loss: 0.157765  [  320/ 1200]
loss: 0.177841  [  480/ 1200]
loss: 0.266559  [  640/ 1200]
loss: 0.121479  [  800/ 1200]
loss: 0.206895  [  960/ 1200]
loss: 0.108644  [ 1120/ 1200]
loss: 0.159602  [ 1200/ 1200]
Train loss: 0.199387, Accuracy: 92.8%
Test Error:
 Avg loss: 0.182653, Accuracy: 92.6%

Epochs without improvement: 1
Epoch 11
-------------------------------
loss: 0.133803  [   32/ 1200]
loss: 0.259332  [  160/ 1200]
loss: 0.237416  [  320/ 1200]
loss: 0.218765  [  480/ 1200]
loss: 0.228523  [  640/ 1200]
loss: 0.128342  [  800/ 1200]
loss: 0.151450  [  960/ 1200]
loss: 0.285597  [ 1120/ 1200]
loss: 0.217764  [ 1200/ 1200]
Train loss: 0.170004, Accuracy: 94.6%
Test Error:
 Avg loss: 0.190967, Accuracy: 93.6%

Epochs without improvement: 2
Epoch 12
-------------------------------
loss: 0.229562  [   32/ 1200]
loss: 0.135510  [  160/ 1200]
loss: 0.138341  [  320/ 1200]
loss: 0.187783  [  480/ 1200]
loss: 0.160174  [  640/ 1200]
loss: 0.122275  [  800/ 1200]
loss: 0.148793  [  960/ 1200]
loss: 0.102670  [ 1120/ 1200]
loss: 0.093559  [ 1200/ 1200]
Train loss: 0.146386, Accuracy: 95.8%
Test Error:
```

```
 Avg loss: 0.154451, Accuracy: 94.0%

Epochs without improvement: 1
Epoch 13
-------------------------------
loss: 0.179343  [   32/ 1200]
loss: 0.225782  [  160/ 1200]
loss: 0.061572  [  320/ 1200]
loss: 0.141602  [  480/ 1200]
loss: 0.177903  [  640/ 1200]
loss: 0.194055  [  800/ 1200]
loss: 0.081279  [  960/ 1200]
loss: 0.047758  [ 1120/ 1200]
loss: 0.028653  [ 1200/ 1200]
Train loss: 0.126447, Accuracy: 96.3%
Test Error:
 Avg loss: 0.149339, Accuracy: 94.0%

Epochs without improvement: 1
Epoch 14
-------------------------------
loss: 0.079591  [   32/ 1200]
loss: 0.114927  [  160/ 1200]
loss: 0.056526  [  320/ 1200]
loss: 0.071502  [  480/ 1200]
loss: 0.110482  [  640/ 1200]
loss: 0.163468  [  800/ 1200]
loss: 0.214211  [  960/ 1200]
loss: 0.104735  [ 1120/ 1200]
loss: 0.057704  [ 1200/ 1200]
Train loss: 0.115713, Accuracy: 97.2%
Test Error:
 Avg loss: 0.123994, Accuracy: 95.3%

Epochs without improvement: 1
Epoch 15
-------------------------------
loss: 0.202325  [   32/ 1200]
loss: 0.225172  [  160/ 1200]
loss: 0.120143  [  320/ 1200]
loss: 0.071568  [  480/ 1200]
loss: 0.064517  [  640/ 1200]
loss: 0.087598  [  800/ 1200]
loss: 0.063949  [  960/ 1200]
loss: 0.056353  [ 1120/ 1200]
loss: 0.028945  [ 1200/ 1200]
Train loss: 0.104861, Accuracy: 96.8%
Test Error:
```

```
 Avg loss: 0.137908, Accuracy: 95.3%

Epochs without improvement: 2
Epoch 16
-------------------------------
loss: 0.056666  [   32/ 1200]
loss: 0.088152  [  160/ 1200]
loss: 0.062125  [  320/ 1200]
loss: 0.043679  [  480/ 1200]
loss: 0.079378  [  640/ 1200]
loss: 0.118479  [  800/ 1200]
loss: 0.081684  [  960/ 1200]
loss: 0.112442  [ 1120/ 1200]
loss: 0.090807  [ 1200/ 1200]
Train loss: 0.102095, Accuracy: 97.5%
Test Error:
 Avg loss: 0.105673, Accuracy: 96.3%

Epochs without improvement: 1
Epoch 17
-------------------------------
loss: 0.107765  [   32/ 1200]
loss: 0.082401  [  160/ 1200]
loss: 0.065923  [  320/ 1200]
loss: 0.028473  [  480/ 1200]
loss: 0.093823  [  640/ 1200]
loss: 0.168564  [  800/ 1200]
loss: 0.106895  [  960/ 1200]
loss: 0.030676  [ 1120/ 1200]
loss: 0.121434  [ 1200/ 1200]
Train loss: 0.089874, Accuracy: 97.4%
Test Error:
 Avg loss: 0.098062, Accuracy: 96.7%

Epochs without improvement: 1
Epoch 18
-------------------------------
loss: 0.039159  [   32/ 1200]
loss: 0.122581  [  160/ 1200]
loss: 0.014194  [  320/ 1200]
loss: 0.091638  [  480/ 1200]
loss: 0.056365  [  640/ 1200]
loss: 0.082471  [  800/ 1200]
loss: 0.113612  [  960/ 1200]
loss: 0.149626  [ 1120/ 1200]
loss: 0.091038  [ 1200/ 1200]
Train loss: 0.079423, Accuracy: 97.9%
Test Error:
```

```
 Avg loss: 0.091568, Accuracy: 96.7%

Epochs without improvement: 1
Epoch 19
-------------------------------
loss: 0.049797  [   32/ 1200]
loss: 0.029178  [  160/ 1200]
loss: 0.113015  [  320/ 1200]
loss: 0.081564  [  480/ 1200]
loss: 0.088980  [  640/ 1200]
loss: 0.019864  [  800/ 1200]
loss: 0.147647  [  960/ 1200]
loss: 0.124973  [ 1120/ 1200]
loss: 0.086383  [ 1200/ 1200]
Train loss: 0.075531, Accuracy: 97.6%
Test Error:
 Avg loss: 0.091782, Accuracy: 97.0%

Epochs without improvement: 2
Epoch 20
-------------------------------
loss: 0.072399  [   32/ 1200]
loss: 0.105177  [  160/ 1200]
loss: 0.057250  [  320/ 1200]
loss: 0.034445  [  480/ 1200]
loss: 0.130604  [  640/ 1200]
loss: 0.016941  [  800/ 1200]
loss: 0.149771  [  960/ 1200]
loss: 0.009663  [ 1120/ 1200]
loss: 0.155512  [ 1200/ 1200]
Train loss: 0.071543, Accuracy: 98.2%
Test Error:
 Avg loss: 0.088307, Accuracy: 97.7%

Epochs without improvement: 1
Epoch 21
-------------------------------
loss: 0.023015  [   32/ 1200]
loss: 0.081578  [  160/ 1200]
loss: 0.101746  [  320/ 1200]
loss: 0.068070  [  480/ 1200]
loss: 0.162807  [  640/ 1200]
loss: 0.151228  [  800/ 1200]
loss: 0.233461  [  960/ 1200]
loss: 0.072230  [ 1120/ 1200]
loss: 0.139825  [ 1200/ 1200]
Train loss: 0.072059, Accuracy: 97.6%
Test Error:
```

```
 Avg loss: 0.097182, Accuracy: 96.3%

Epochs without improvement: 2
Epoch 22
-------------------------------
loss: 0.139019  [   32/ 1200]
loss: 0.080993  [  160/ 1200]
loss: 0.053904  [  320/ 1200]
loss: 0.010127  [  480/ 1200]
loss: 0.078419  [  640/ 1200]
loss: 0.053003  [  800/ 1200]
loss: 0.107636  [  960/ 1200]
loss: 0.099190  [ 1120/ 1200]
loss: 0.006945  [ 1200/ 1200]
Train loss: 0.066505, Accuracy: 98.1%
Test Error:
 Avg loss: 0.100748, Accuracy: 96.3%

Epochs without improvement: 3
Epoch 23
-------------------------------
loss: 0.024083  [   32/ 1200]
loss: 0.131390  [  160/ 1200]
loss: 0.190066  [  320/ 1200]
loss: 0.009271  [  480/ 1200]
loss: 0.123551  [  640/ 1200]
loss: 0.032633  [  800/ 1200]
loss: 0.093070  [  960/ 1200]
loss: 0.015757  [ 1120/ 1200]
loss: 0.037138  [ 1200/ 1200]
Train loss: 0.068039, Accuracy: 97.9%
Test Error:
 Avg loss: 0.081815, Accuracy: 97.3%

Epochs without improvement: 1
Epoch 24
-------------------------------
loss: 0.101273  [   32/ 1200]
loss: 0.067157  [  160/ 1200]
loss: 0.035169  [  320/ 1200]
loss: 0.113065  [  480/ 1200]
loss: 0.051849  [  640/ 1200]
loss: 0.014376  [  800/ 1200]
loss: 0.115634  [  960/ 1200]
loss: 0.088494  [ 1120/ 1200]
loss: 0.002840  [ 1200/ 1200]
Train loss: 0.057073, Accuracy: 98.7%
Test Error:
```

```
 Avg loss: 0.067940, Accuracy: 98.0%

Epochs without improvement: 1
Epoch 25
-------------------------------
loss: 0.053267  [   32/ 1200]
loss: 0.031379  [  160/ 1200]
loss: 0.005828  [  320/ 1200]
loss: 0.031130  [  480/ 1200]
loss: 0.107042  [  640/ 1200]
loss: 0.119486  [  800/ 1200]
loss: 0.041261  [  960/ 1200]
loss: 0.071992  [ 1120/ 1200]
loss: 0.045246  [ 1200/ 1200]
Train loss: 0.057506, Accuracy: 98.3%
Test Error:
 Avg loss: 0.096163, Accuracy: 97.3%

Epochs without improvement: 2
Epoch 26
-------------------------------
loss: 0.051404  [   32/ 1200]
loss: 0.009384  [  160/ 1200]
loss: 0.043250  [  320/ 1200]
loss: 0.030697  [  480/ 1200]
loss: 0.013901  [  640/ 1200]
loss: 0.015600  [  800/ 1200]
loss: 0.042340  [  960/ 1200]
loss: 0.017196  [ 1120/ 1200]
loss: 0.091048  [ 1200/ 1200]
Train loss: 0.053126, Accuracy: 98.7%
Test Error:
 Avg loss: 0.078607, Accuracy: 96.3%

Epochs without improvement: 3
Epoch 27
-------------------------------
loss: 0.055575  [   32/ 1200]
loss: 0.009792  [  160/ 1200]
loss: 0.103288  [  320/ 1200]
loss: 0.022260  [  480/ 1200]
loss: 0.047346  [  640/ 1200]
loss: 0.038951  [  800/ 1200]
loss: 0.035775  [  960/ 1200]
loss: 0.012291  [ 1120/ 1200]
loss: 0.070904  [ 1200/ 1200]
Train loss: 0.052021, Accuracy: 98.6%
Test Error:
```

```
 Avg loss: 0.069037, Accuracy: 97.0%

Epochs without improvement: 4
Epoch 28
-------------------------------
loss: 0.008827  [   32/ 1200]
loss: 0.114846  [  160/ 1200]
loss: 0.038336  [  320/ 1200]
loss: 0.030818  [  480/ 1200]
loss: 0.046810  [  640/ 1200]
loss: 0.013422  [  800/ 1200]
loss: 0.222977  [  960/ 1200]
loss: 0.020030  [ 1120/ 1200]
loss: 0.052963  [ 1200/ 1200]
Train loss: 0.051662, Accuracy: 98.6%
Test Error:
 Avg loss: 0.085965, Accuracy: 96.3%

Epochs without improvement: 5
Epoch 29
-------------------------------
loss: 0.038811  [   32/ 1200]
loss: 0.085596  [  160/ 1200]
loss: 0.039483  [  320/ 1200]
loss: 0.022291  [  480/ 1200]
loss: 0.062497  [  640/ 1200]
loss: 0.003894  [  800/ 1200]
loss: 0.060444  [  960/ 1200]
loss: 0.104052  [ 1120/ 1200]
loss: 0.079492  [ 1200/ 1200]
Train loss: 0.049001, Accuracy: 98.6%
Test Error:
 Avg loss: 0.067182, Accuracy: 97.3%

Epochs without improvement: 1
Epoch 30
-------------------------------
loss: 0.061634  [   32/ 1200]
loss: 0.173028  [  160/ 1200]
loss: 0.069791  [  320/ 1200]
loss: 0.098111  [  480/ 1200]
loss: 0.022697  [  640/ 1200]
loss: 0.137307  [  800/ 1200]
loss: 0.005880  [  960/ 1200]
loss: 0.033507  [ 1120/ 1200]
loss: 0.001455  [ 1200/ 1200]
Train loss: 0.061606, Accuracy: 98.3%
Test Error:
```

```
 Avg loss: 0.066528, Accuracy: 97.3%

Epochs without improvement: 2
Epoch 31
-------------------------------
loss: 0.036351  [   32/ 1200]
loss: 0.010064  [  160/ 1200]
loss: 0.005700  [  320/ 1200]
loss: 0.005753  [  480/ 1200]
loss: 0.022160  [  640/ 1200]
loss: 0.017437  [  800/ 1200]
loss: 0.002459  [  960/ 1200]
loss: 0.114794  [ 1120/ 1200]
loss: 0.072738  [ 1200/ 1200]
Train loss: 0.044681, Accuracy: 98.9%
Test Error:
 Avg loss: 0.051309, Accuracy: 99.0%

Epochs without improvement: 1
Epoch 32
-------------------------------
loss: 0.047798  [   32/ 1200]
loss: 0.052361  [  160/ 1200]
loss: 0.003693  [  320/ 1200]
loss: 0.003118  [  480/ 1200]
loss: 0.022115  [  640/ 1200]
loss: 0.065776  [  800/ 1200]
loss: 0.021642  [  960/ 1200]
loss: 0.044279  [ 1120/ 1200]
loss: 0.070532  [ 1200/ 1200]
Train loss: 0.043801, Accuracy: 98.5%
Test Error:
 Avg loss: 0.073748, Accuracy: 97.3%

Epochs without improvement: 2
Epoch 33
-------------------------------
loss: 0.065711  [   32/ 1200]
loss: 0.052027  [  160/ 1200]
loss: 0.030550  [  320/ 1200]
loss: 0.027347  [  480/ 1200]
loss: 0.038229  [  640/ 1200]
loss: 0.043996  [  800/ 1200]
loss: 0.051736  [  960/ 1200]
loss: 0.008764  [ 1120/ 1200]
loss: 0.096156  [ 1200/ 1200]
Train loss: 0.041457, Accuracy: 98.7%
Test Error:
```

```
 Avg loss: 0.054704, Accuracy: 99.0%

Epochs without improvement: 3
Epoch 34
-------------------------------
loss: 0.048287  [   32/ 1200]
loss: 0.014121  [  160/ 1200]
loss: 0.044754  [  320/ 1200]
loss: 0.004969  [  480/ 1200]
loss: 0.038507  [  640/ 1200]
loss: 0.041395  [  800/ 1200]
loss: 0.051291  [  960/ 1200]
loss: 0.063141  [ 1120/ 1200]
loss: 0.006266  [ 1200/ 1200]
Train loss: 0.035637, Accuracy: 99.0%
Test Error:
 Avg loss: 0.049560, Accuracy: 98.0%

Epochs without improvement: 1
Epoch 35
-------------------------------
loss: 0.023135  [   32/ 1200]
loss: 0.002997  [  160/ 1200]
loss: 0.033895  [  320/ 1200]
loss: 0.082411  [  480/ 1200]
loss: 0.017891  [  640/ 1200]
loss: 0.019680  [  800/ 1200]
loss: 0.066042  [  960/ 1200]
loss: 0.018910  [ 1120/ 1200]
loss: 0.123980  [ 1200/ 1200]
Train loss: 0.041058, Accuracy: 98.7%
Test Error:
 Avg loss: 0.051129, Accuracy: 98.0%

Epochs without improvement: 2
Epoch 36
-------------------------------
loss: 0.014678  [   32/ 1200]
loss: 0.025199  [  160/ 1200]
loss: 0.004426  [  320/ 1200]
loss: 0.048345  [  480/ 1200]
loss: 0.002353  [  640/ 1200]
loss: 0.013440  [  800/ 1200]
loss: 0.027784  [  960/ 1200]
loss: 0.053939  [ 1120/ 1200]
loss: 0.031647  [ 1200/ 1200]
Train loss: 0.038643, Accuracy: 98.8%
Test Error:
```

```
 Avg loss: 0.050805, Accuracy: 98.3%

Epochs without improvement: 3
Epoch 37
-------------------------------
loss: 0.038129  [   32/ 1200]
loss: 0.029302  [  160/ 1200]
loss: 0.060778  [  320/ 1200]
loss: 0.077699  [  480/ 1200]
loss: 0.002442  [  640/ 1200]
loss: 0.002358  [  800/ 1200]
loss: 0.076937  [  960/ 1200]
loss: 0.015768  [ 1120/ 1200]
loss: 0.006858  [ 1200/ 1200]
Train loss: 0.036364, Accuracy: 99.0%
Test Error:
 Avg loss: 0.055978, Accuracy: 97.7%

Epochs without improvement: 4
Epoch 38
-------------------------------
loss: 0.004998  [   32/ 1200]
loss: 0.062285  [  160/ 1200]
loss: 0.030809  [  320/ 1200]
loss: 0.020507  [  480/ 1200]
loss: 0.068113  [  640/ 1200]
loss: 0.031009  [  800/ 1200]
loss: 0.019288  [  960/ 1200]
loss: 0.015271  [ 1120/ 1200]
loss: 0.005229  [ 1200/ 1200]
Train loss: 0.036716, Accuracy: 98.8%
Test Error:
 Avg loss: 0.053202, Accuracy: 97.7%

Epochs without improvement: 5
Epoch 39
-------------------------------
loss: 0.034129  [   32/ 1200]
loss: 0.036915  [  160/ 1200]
loss: 0.031089  [  320/ 1200]
loss: 0.011022  [  480/ 1200]
loss: 0.003164  [  640/ 1200]
loss: 0.003916  [  800/ 1200]
loss: 0.040093  [  960/ 1200]
loss: 0.073843  [ 1120/ 1200]
loss: 0.112792  [ 1200/ 1200]
Train loss: 0.037937, Accuracy: 98.8%
Test Error:
```

```
 Avg loss: 0.047056, Accuracy: 98.0%

Epochs without improvement: 1
Epoch 40
-------------------------------
loss: 0.028605  [   32/ 1200]
loss: 0.016160  [  160/ 1200]
loss: 0.040019  [  320/ 1200]
loss: 0.043176  [  480/ 1200]
loss: 0.053096  [  640/ 1200]
loss: 0.050797  [  800/ 1200]
loss: 0.053807  [  960/ 1200]
loss: 0.035509  [ 1120/ 1200]
loss: 0.001269  [ 1200/ 1200]
Train loss: 0.035334, Accuracy: 99.0%
Test Error:
 Avg loss: 0.047450, Accuracy: 98.0%

Epochs without improvement: 2
Epoch 41
-------------------------------
loss: 0.041416  [   32/ 1200]
loss: 0.063508  [  160/ 1200]
loss: 0.015427  [  320/ 1200]
loss: 0.092017  [  480/ 1200]
loss: 0.034533  [  640/ 1200]
loss: 0.108607  [  800/ 1200]
loss: 0.133861  [  960/ 1200]
loss: 0.010837  [ 1120/ 1200]
loss: 0.058195  [ 1200/ 1200]
Train loss: 0.040061, Accuracy: 98.8%
Test Error:
 Avg loss: 0.057794, Accuracy: 97.3%

Epochs without improvement: 3
Epoch 42
-------------------------------
loss: 0.006742  [   32/ 1200]
loss: 0.001024  [  160/ 1200]
loss: 0.043006  [  320/ 1200]
loss: 0.074317  [  480/ 1200]
loss: 0.002652  [  640/ 1200]
loss: 0.017105  [  800/ 1200]
loss: 0.003180  [  960/ 1200]
loss: 0.046401  [ 1120/ 1200]
loss: 0.044988  [ 1200/ 1200]
Train loss: 0.033172, Accuracy: 99.1%
Test Error:
```

```
 Avg loss: 0.066433, Accuracy: 98.0%

Epochs without improvement: 4
Epoch 43
-------------------------------
loss: 0.004457  [   32/ 1200]
loss: 0.006240  [  160/ 1200]
loss: 0.009464  [  320/ 1200]
loss: 0.013539  [  480/ 1200]
loss: 0.040041  [  640/ 1200]
loss: 0.027142  [  800/ 1200]
loss: 0.018608  [  960/ 1200]
loss: 0.075177  [ 1120/ 1200]
loss: 0.029163  [ 1200/ 1200]
Train loss: 0.034226, Accuracy: 99.0%
Test Error:
 Avg loss: 0.054702, Accuracy: 98.0%

Epochs without improvement: 5
Epoch 44
-------------------------------
loss: 0.041187  [   32/ 1200]
loss: 0.001366  [  160/ 1200]
loss: 0.088880  [  320/ 1200]
loss: 0.018574  [  480/ 1200]
loss: 0.040348  [  640/ 1200]
loss: 0.000864  [  800/ 1200]
loss: 0.088168  [  960/ 1200]
loss: 0.029446  [ 1120/ 1200]
loss: 0.062437  [ 1200/ 1200]
Train loss: 0.034840, Accuracy: 98.5%
Test Error:
 Avg loss: 0.055733, Accuracy: 98.0%

Epochs without improvement: 6
Epoch 45
-------------------------------
loss: 0.002692  [   32/ 1200]
loss: 0.047794  [  160/ 1200]
loss: 0.019752  [  320/ 1200]
loss: 0.007327  [  480/ 1200]
loss: 0.014030  [  640/ 1200]
loss: 0.010177  [  800/ 1200]
loss: 0.087246  [  960/ 1200]
loss: 0.038918  [ 1120/ 1200]
loss: 0.006341  [ 1200/ 1200]
Train loss: 0.034701, Accuracy: 98.8%
Test Error:
```

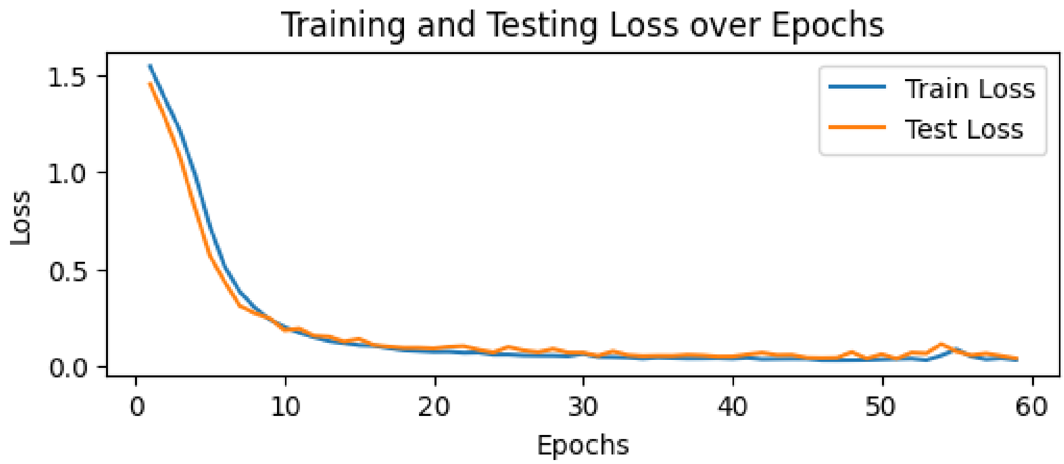
```
 Avg loss: 0.040671, Accuracy: 98.7%

Epochs without improvement: 1
Epoch 46
-------------------------------
loss: 0.059902  [   32/ 1200]
loss: 0.003473  [  160/ 1200]
loss: 0.004726  [  320/ 1200]
loss: 0.001625  [  480/ 1200]
loss: 0.005309  [  640/ 1200]
loss: 0.046668  [  800/ 1200]
loss: 0.003211  [  960/ 1200]
loss: 0.002072  [ 1120/ 1200]
loss: 0.001626  [ 1200/ 1200]
Train loss: 0.027766, Accuracy: 99.1%
Test Error:
 Avg loss: 0.039231, Accuracy: 98.7%

Epochs without improvement: 1
Epoch 47
-------------------------------
loss: 0.093964  [   32/ 1200]
loss: 0.040940  [  160/ 1200]
loss: 0.026537  [  320/ 1200]
loss: 0.001144  [  480/ 1200]
loss: 0.012998  [  640/ 1200]
loss: 0.021639  [  800/ 1200]
loss: 0.073528  [  960/ 1200]
loss: 0.004855  [ 1120/ 1200]
loss: 0.001211  [ 1200/ 1200]
Train loss: 0.027566, Accuracy: 99.2%
Test Error:
 Avg loss: 0.040453, Accuracy: 98.7%

Epochs without improvement: 2
Epoch 48
-------------------------------
loss: 0.061475  [   32/ 1200]
loss: 0.030804  [  160/ 1200]
loss: 0.000652  [  320/ 1200]
loss: 0.050947  [  480/ 1200]
loss: 0.011705  [  640/ 1200]
loss: 0.039699  [  800/ 1200]
loss: 0.004203  [  960/ 1200]
loss: 0.001038  [ 1120/ 1200]
loss: 0.008619  [ 1200/ 1200]
Train loss: 0.028091, Accuracy: 99.0%
Test Error:
```

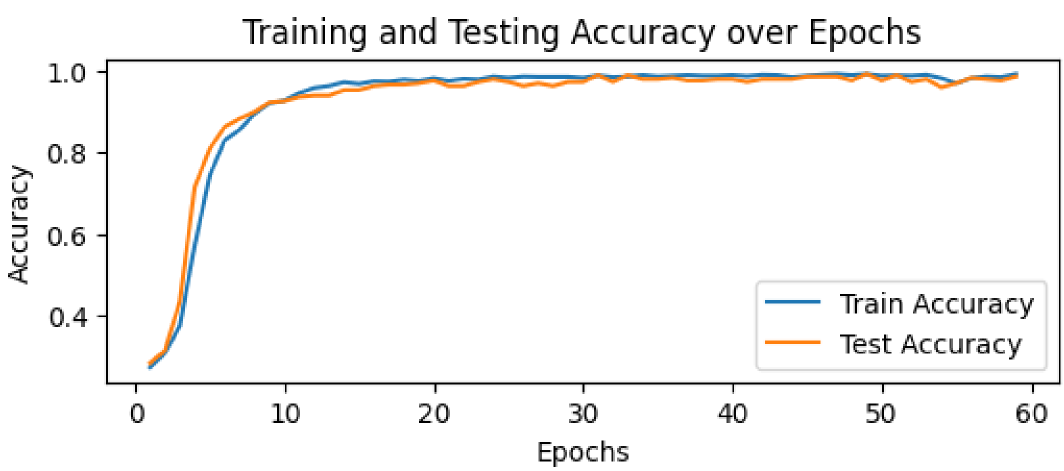
```
 Avg loss: 0.069906, Accuracy: 97.7%

Epochs without improvement: 3
Epoch 49
-------------------------------
loss: 0.003700  [   32/ 1200]
loss: 0.025943  [  160/ 1200]
loss: 0.019803  [  320/ 1200]
loss: 0.036297  [  480/ 1200]
loss: 0.015468  [  640/ 1200]
loss: 0.052720  [  800/ 1200]
loss: 0.003377  [  960/ 1200]
loss: 0.034636  [ 1120/ 1200]
loss: 0.000544  [ 1200/ 1200]
Train loss: 0.028965, Accuracy: 99.2%
Test Error:
 Avg loss: 0.036420, Accuracy: 99.3%

Epochs without improvement: 1
Epoch 50
-------------------------------
loss: 0.001091  [   32/ 1200]
loss: 0.002261  [  160/ 1200]
loss: 0.085223  [  320/ 1200]
loss: 0.010762  [  480/ 1200]
loss: 0.074212  [  640/ 1200]
loss: 0.039288  [  800/ 1200]
loss: 0.168483  [  960/ 1200]
loss: 0.043193  [ 1120/ 1200]
loss: 0.000609  [ 1200/ 1200]
Train loss: 0.031607, Accuracy: 98.8%
Test Error:
 Avg loss: 0.059301, Accuracy: 97.7%

Epochs without improvement: 2
Epoch 51
-------------------------------
loss: 0.030108  [   32/ 1200]
loss: 0.002364  [  160/ 1200]
loss: 0.019303  [  320/ 1200]
loss: 0.045999  [  480/ 1200]
loss: 0.121890  [  640/ 1200]
loss: 0.067903  [  800/ 1200]
loss: 0.008926  [  960/ 1200]
loss: 0.080965  [ 1120/ 1200]
loss: 0.039409  [ 1200/ 1200]
Train loss: 0.033938, Accuracy: 98.9%
Test Error:
```

```
 Avg loss: 0.035819, Accuracy: 99.0%

Epochs without improvement: 3
Epoch 52
-------------------------------
loss: 0.002428  [   32/ 1200]
loss: 0.001785  [  160/ 1200]
loss: 0.127533  [  320/ 1200]
loss: 0.152849  [  480/ 1200]
loss: 0.002284  [  640/ 1200]
loss: 0.009786  [  800/ 1200]
loss: 0.009633  [  960/ 1200]
loss: 0.031358  [ 1120/ 1200]
loss: 0.002026  [ 1200/ 1200]
Train loss: 0.035895, Accuracy: 98.8%
Test Error:
 Avg loss: 0.066964, Accuracy: 97.3%

Epochs without improvement: 4
Epoch 53
-------------------------------
loss: 0.164266  [   32/ 1200]
loss: 0.002963  [  160/ 1200]
loss: 0.005166  [  320/ 1200]
loss: 0.036988  [  480/ 1200]
loss: 0.004026  [  640/ 1200]
loss: 0.003018  [  800/ 1200]
loss: 0.124420  [  960/ 1200]
loss: 0.014657  [ 1120/ 1200]
loss: 0.017155  [ 1200/ 1200]
Train loss: 0.028152, Accuracy: 99.1%
Test Error:
 Avg loss: 0.063342, Accuracy: 98.0%

Epochs without improvement: 5
Epoch 54
-------------------------------
loss: 0.051972  [   32/ 1200]
loss: 0.001398  [  160/ 1200]
loss: 0.001850  [  320/ 1200]
loss: 0.044670  [  480/ 1200]
loss: 0.022197  [  640/ 1200]
loss: 0.116224  [  800/ 1200]
loss: 0.059265  [  960/ 1200]
loss: 0.451811  [ 1120/ 1200]
loss: 0.058172  [ 1200/ 1200]
Train loss: 0.052134, Accuracy: 98.2%
Test Error:
```

```
 Avg loss: 0.111386, Accuracy: 96.0%

Epochs without improvement: 6
Epoch 55
-------------------------------
loss: 0.179380  [   32/ 1200]
loss: 0.137194  [  160/ 1200]
loss: 0.011483  [  320/ 1200]
loss: 0.325902  [  480/ 1200]
loss: 0.228773  [  640/ 1200]
loss: 0.031239  [  800/ 1200]
loss: 0.053021  [  960/ 1200]
loss: 0.017213  [ 1120/ 1200]
loss: 0.092831  [ 1200/ 1200]
Train loss: 0.084044, Accuracy: 97.0%
Test Error:
 Avg loss: 0.072980, Accuracy: 97.0%

Epochs without improvement: 7
Epoch 56
-------------------------------
loss: 0.030765  [   32/ 1200]
loss: 0.003873  [  160/ 1200]
loss: 0.039260  [  320/ 1200]
loss: 0.072871  [  480/ 1200]
loss: 0.003018  [  640/ 1200]
loss: 0.052378  [  800/ 1200]
loss: 0.020337  [  960/ 1200]
loss: 0.051596  [ 1120/ 1200]
loss: 0.004437  [ 1200/ 1200]
Train loss: 0.047565, Accuracy: 98.2%
Test Error:
 Avg loss: 0.053940, Accuracy: 98.3%

Epochs without improvement: 8
Epoch 57
-------------------------------
loss: 0.072837  [   32/ 1200]
loss: 0.025629  [  160/ 1200]
loss: 0.065177  [  320/ 1200]
loss: 0.028957  [  480/ 1200]
loss: 0.023616  [  640/ 1200]
loss: 0.078935  [  800/ 1200]
loss: 0.006522  [  960/ 1200]
loss: 0.043495  [ 1120/ 1200]
loss: 0.001280  [ 1200/ 1200]
Train loss: 0.032025, Accuracy: 98.7%
Test Error:
```

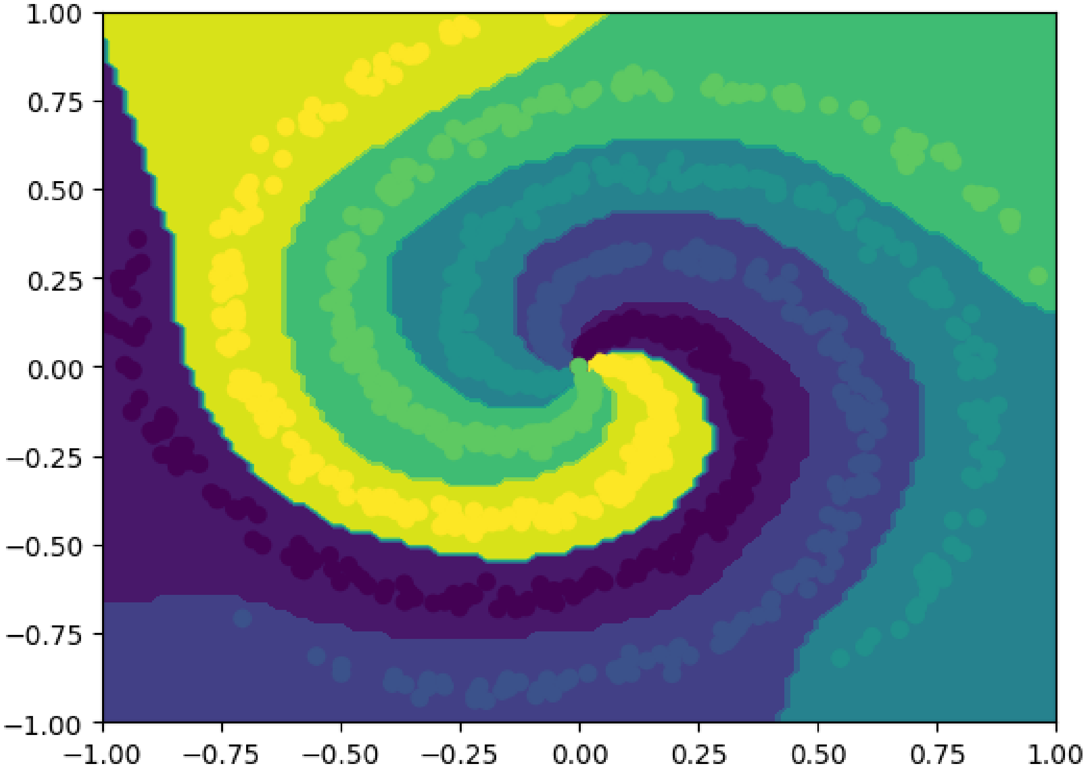
```
 Avg loss: 0.062920, Accuracy: 98.0%

Epochs without improvement: 9
Epoch 58
-------------------------------
loss: 0.062117  [   32/ 1200]
loss: 0.039744  [  160/ 1200]
loss: 0.002672  [  320/ 1200]
loss: 0.013355  [  480/ 1200]
loss: 0.064070  [  640/ 1200]
loss: 0.135024  [  800/ 1200]
loss: 0.020150  [  960/ 1200]
loss: 0.055089  [ 1120/ 1200]
loss: 0.007418  [ 1200/ 1200]
Train loss: 0.038077, Accuracy: 98.5%
Test Error:
 Avg loss: 0.050397, Accuracy: 97.7%

Epochs without improvement: 10
Epoch 59
-------------------------------
loss: 0.067249  [   32/ 1200]
loss: 0.059732  [  160/ 1200]
loss: 0.079318  [  320/ 1200]
loss: 0.000759  [  480/ 1200]
loss: 0.000671  [  640/ 1200]
loss: 0.001094  [  800/ 1200]
loss: 0.031435  [  960/ 1200]
loss: 0.008574  [ 1120/ 1200]
loss: 0.011886  [ 1200/ 1200]
Train loss: 0.029347, Accuracy: 99.2%
Test Error:
 Avg loss: 0.037859, Accuracy: 98.7%

Early stopping triggered after 60 epochs.
Training completed.
```





[8]:
```
res = 100
x,y=np.meshgrid(np.linspace(-1,1,res),np.linspace(-1,1,res))
xy=np.concatenate((x.reshape(-1,1),y.reshape(-1,1)),axis=1)
z=model(torch.tensor(xy).float()).detach().numpy()
z=np.argmax(z,1).reshape(res,res)
plt.contourf(x,y,z)
plt.scatter(training_data[:,0],training_data[:,1],c=training_labels)
```

[8]: <matplotlib.collections.PathCollection at 0x2ba79058fd0>



- **Describe your results and discuss the observed performance**
  The network is able to reach an accuracy of about 97-98% within the 100 epochs. The network often converges after 50-60 epochs with the stopping parameters i've set. However, the last few percentages are hard to reach due to how our data is centered and tightly clustered about (0,0).
- **Visualize network performance**
  The visualization shows the trouble with classifying the center data, nevertheless, we still see good seperation between the 5 classes.