In [1]:
```
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.conv1 = nn.Conv2d(1, 32, 3, 1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1, padding=1)

        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)

        self.fc1 = nn.Linear(3136, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)

        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)

        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)

        output = F.log_softmax(x, dim=1)
        return output
```

In [3]:
```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        output = model(data)
        loss = F.nll_loss(output, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
```

```
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break
```

In [4]:
```
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    i=0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max l
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

    images, labels = next(iter(test_loader))
    output1 = model(images)
    pred1 = output1.argmax(dim=1, keepdim=True)
```

In [5]:
```
def show_(model, test_loader):
    model.eval()
    images, labels = next(iter(test_loader))

    with torch.no_grad():
        output1 = model(images)

    pred1 = output1.argmax(dim=1, keepdim=True)

    for i in range(9):
        print(f"GT #{i+1}: {labels[i].item()} | Prediction #{i+1}: {pred1[i].item()

        plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].cpu().numpy().squeeze(), cmap="gray")
        plt.title(f"GT: {labels[i].item()} | Pred: {pred1[i].item()}", fontsize=10)
        plt.axis("off")

    plt.show()
```

In [6]:
```
def main():
    # Training settings
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
    parser.add_argument('--batch-size', type=int, default=4, metavar='N',
                        help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=1, metavar='N',
```

```
                        help='number of epochs to train (default: 14)')
    parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                        help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                        help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--no-mps', action='store_true', default=False,
                        help='disables macOS GPU training')
    parser.add_argument('--dry-run', action='store_true', default=False,
                        help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                        help='how many batches to wait before logging training stat
    parser.add_argument('--save-model', action='store_true', default=True,
                        help='For Saving the current Model')

    args = parser.parse_args(['--batch-size', '64', '--test-batch-size', '1000', '-
    use_cuda = not args.no_cuda and torch.cuda.is_available()
    use_mps = not args.no_mps and torch.backends.mps.is_available()

    torch.manual_seed(args.seed)

    if use_cuda:
        device = torch.device("cuda")
    elif use_mps:
        device = torch.device("mps")
    else:
        device = torch.device("cpu")

    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])

    dataset1 = datasets.MNIST('../data', train=True, download=True,
                       transform=transform)
    dataset2 = datasets.MNIST('../data', train=False,
                       transform=transform)

    train_loader = torch.utils.data.DataLoader(dataset1,**train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    model = Net().to(device)
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
```

```
    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        show_(model,  test_loader)
        scheduler.step()

    if args.save_model:
        torch.save(model.state_dict(), "mnist_cnn.pt")
```

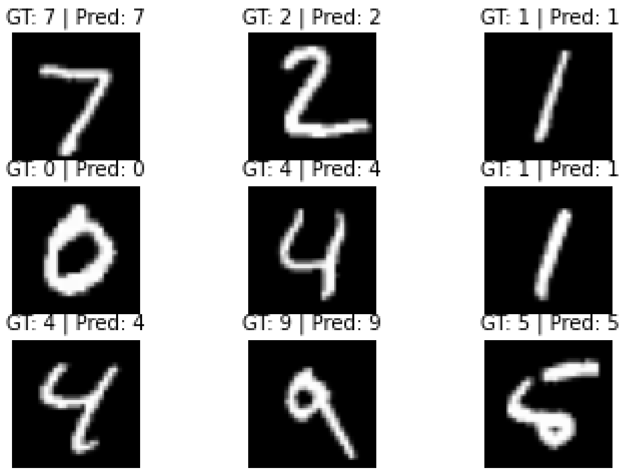
In [24]:
```
if __name__ == '__main__':
    main()
```

```
Train Epoch: 1 [0/60000 (0%)]	Loss: 2.308479
Train Epoch: 1 [640/60000 (1%)]	Loss: 1.590655
Train Epoch: 1 [1280/60000 (2%)]	Loss: 0.790749
Train Epoch: 1 [1920/60000 (3%)]	Loss: 0.657724
Train Epoch: 1 [2560/60000 (4%)]	Loss: 0.350074
Train Epoch: 1 [3200/60000 (5%)]	Loss: 0.369764
Train Epoch: 1 [3840/60000 (6%)]	Loss: 0.348133
Train Epoch: 1 [4480/60000 (7%)]	Loss: 0.391021
Train Epoch: 1 [5120/60000 (9%)]	Loss: 0.366617
Train Epoch: 1 [5760/60000 (10%)]	Loss: 0.277427
Train Epoch: 1 [6400/60000 (11%)]	Loss: 0.152517
Train Epoch: 1 [7040/60000 (12%)]	Loss: 0.243094
Train Epoch: 1 [7680/60000 (13%)]	Loss: 0.349941
Train Epoch: 1 [8320/60000 (14%)]	Loss: 0.133925
Train Epoch: 1 [8960/60000 (15%)]	Loss: 0.200861
Train Epoch: 1 [9600/60000 (16%)]	Loss: 0.119459
Train Epoch: 1 [10240/60000 (17%)]	Loss: 0.524526
Train Epoch: 1 [10880/60000 (18%)]	Loss: 0.160262
Train Epoch: 1 [11520/60000 (19%)]	Loss: 0.468837
Train Epoch: 1 [12160/60000 (20%)]	Loss: 0.116855
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.174491
Train Epoch: 1 [13440/60000 (22%)]	Loss: 0.066993
Train Epoch: 1 [14080/60000 (23%)]	Loss: 0.153936
Train Epoch: 1 [14720/60000 (25%)]	Loss: 0.301342
Train Epoch: 1 [15360/60000 (26%)]	Loss: 0.088914
Train Epoch: 1 [16000/60000 (27%)]	Loss: 0.349144
Train Epoch: 1 [16640/60000 (28%)]	Loss: 0.321146
Train Epoch: 1 [17280/60000 (29%)]	Loss: 0.091606
Train Epoch: 1 [17920/60000 (30%)]	Loss: 0.160549
Train Epoch: 1 [18560/60000 (31%)]	Loss: 0.129304
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.212323
Train Epoch: 1 [19840/60000 (33%)]	Loss: 0.224101
Train Epoch: 1 [20480/60000 (34%)]	Loss: 0.029677
Train Epoch: 1 [21120/60000 (35%)]	Loss: 0.134393
Train Epoch: 1 [21760/60000 (36%)]	Loss: 0.007470
Train Epoch: 1 [22400/60000 (37%)]	Loss: 0.109912
Train Epoch: 1 [23040/60000 (38%)]	Loss: 0.131253
Train Epoch: 1 [23680/60000 (39%)]	Loss: 0.148079
Train Epoch: 1 [24320/60000 (41%)]	Loss: 0.021673
Train Epoch: 1 [24960/60000 (42%)]	Loss: 0.093328
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.098482
Train Epoch: 1 [26240/60000 (44%)]	Loss: 0.107773
Train Epoch: 1 [26880/60000 (45%)]	Loss: 0.279972
Train Epoch: 1 [27520/60000 (46%)]	Loss: 0.157392
Train Epoch: 1 [28160/60000 (47%)]	Loss: 0.204619
Train Epoch: 1 [28800/60000 (48%)]	Loss: 0.071182
Train Epoch: 1 [29440/60000 (49%)]	Loss: 0.078604
Train Epoch: 1 [30080/60000 (50%)]	Loss: 0.074912
Train Epoch: 1 [30720/60000 (51%)]	Loss: 0.090031
Train Epoch: 1 [31360/60000 (52%)]	Loss: 0.076186
Train Epoch: 1 [32000/60000 (53%)]	Loss: 0.133295
Train Epoch: 1 [32640/60000 (54%)]	Loss: 0.080817
Train Epoch: 1 [33280/60000 (55%)]	Loss: 0.092732
Train Epoch: 1 [33920/60000 (57%)]	Loss: 0.025839
Train Epoch: 1 [34560/60000 (58%)]	Loss: 0.063290
Train Epoch: 1 [35200/60000 (59%)]	Loss: 0.173058
```

```
Train Epoch: 1 [35840/60000 (60%)]	Loss: 0.107424
Train Epoch: 1 [36480/60000 (61%)]	Loss: 0.046419
Train Epoch: 1 [37120/60000 (62%)]	Loss: 0.048043
Train Epoch: 1 [37760/60000 (63%)]	Loss: 0.070353
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.156137
Train Epoch: 1 [39040/60000 (65%)]	Loss: 0.026961
Train Epoch: 1 [39680/60000 (66%)]	Loss: 0.125925
Train Epoch: 1 [40320/60000 (67%)]	Loss: 0.066434
Train Epoch: 1 [40960/60000 (68%)]	Loss: 0.091253
Train Epoch: 1 [41600/60000 (69%)]	Loss: 0.097733
Train Epoch: 1 [42240/60000 (70%)]	Loss: 0.053124
Train Epoch: 1 [42880/60000 (71%)]	Loss: 0.150244
Train Epoch: 1 [43520/60000 (72%)]	Loss: 0.305708
Train Epoch: 1 [44160/60000 (74%)]	Loss: 0.021459
Train Epoch: 1 [44800/60000 (75%)]	Loss: 0.212554
Train Epoch: 1 [45440/60000 (76%)]	Loss: 0.207091
Train Epoch: 1 [46080/60000 (77%)]	Loss: 0.254228
Train Epoch: 1 [46720/60000 (78%)]	Loss: 0.084887
Train Epoch: 1 [47360/60000 (79%)]	Loss: 0.163142
Train Epoch: 1 [48000/60000 (80%)]	Loss: 0.164793
Train Epoch: 1 [48640/60000 (81%)]	Loss: 0.084105
Train Epoch: 1 [49280/60000 (82%)]	Loss: 0.051359
Train Epoch: 1 [49920/60000 (83%)]	Loss: 0.094097
Train Epoch: 1 [50560/60000 (84%)]	Loss: 0.097264
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.169708
Train Epoch: 1 [51840/60000 (86%)]	Loss: 0.047835
Train Epoch: 1 [52480/60000 (87%)]	Loss: 0.016487
Train Epoch: 1 [53120/60000 (88%)]	Loss: 0.088631
Train Epoch: 1 [53760/60000 (90%)]	Loss: 0.067330
Train Epoch: 1 [54400/60000 (91%)]	Loss: 0.027201
Train Epoch: 1 [55040/60000 (92%)]	Loss: 0.048748
Train Epoch: 1 [55680/60000 (93%)]	Loss: 0.226113
Train Epoch: 1 [56320/60000 (94%)]	Loss: 0.054036
Train Epoch: 1 [56960/60000 (95%)]	Loss: 0.030440
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.219541
Train Epoch: 1 [58240/60000 (97%)]	Loss: 0.102245
Train Epoch: 1 [58880/60000 (98%)]	Loss: 0.005240
Train Epoch: 1 [59520/60000 (99%)]	Loss: 0.002186

Test set: Average loss: 0.0412, Accuracy: 9863/10000 (99%)

GT #1: 7 | Prediction #1: 7
GT #2: 2 | Prediction #2: 2
GT #3: 1 | Prediction #3: 1
GT #4: 0 | Prediction #4: 0
GT #5: 4 | Prediction #5: 4
GT #6: 1 | Prediction #6: 1
GT #7: 4 | Prediction #7: 4
GT #8: 9 | Prediction #8: 9
GT #9: 5 | Prediction #9: 5
```



Task 1: CIFAR 10

In [8]:
```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [11]:
```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data\cifar-10-python.tar.gz
100%|██████████████████████████████████████████████████| 170498071/170498071 [03:26<00:00, 824253.40it/s]
Extracting ./data\cifar-10-python.tar.gz to ./data
```

Start of the model

In [31]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

# parameters: Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, d
# charactersitics: 60000 32x32
# 3 channels instead of 1

        self.conv1 = nn.Conv2d(3, 32, 3, 1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1, padding=1)

        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
# 32 -> 16 -> 8
# 64x8x8 = 4096
        self.fc1 = nn.Linear(4096, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
```

```
        x = F.relu(x)
        x = F.max_pool2d(x, 2)

        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)

        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)

        output = F.log_softmax(x, dim=1)
        return output
```

In [32]:
```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        output = model(data)
        loss = F.nll_loss(output, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break
```

In [33]:
```
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    i=0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max l
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

    images, labels = next(iter(test_loader))
```

```
    output1 = model(images)
    pred1 = output1.argmax(dim=1, keepdim=True)
```

In [69]:
```
def show_(model, test_loader):
    model.eval()
    images, labels = next(iter(test_loader))

    with torch.no_grad():
        output1 = model(images)

    pred1 = output1.argmax(dim=1, keepdim=True)

    for i in range(9):
        print(f"GT #{i+1}: {labels[i].item()} | Prediction #{i+1}: {pred1[i].item()

# .reshape(32,32,3). images.reshape()
# .T
# np.transpose (1, 2, 0)

        plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].cpu().numpy().T.squeeze().reshape(32,32,3), cmap="gray
        plt.imshow(np.transpose(images[i], (1, 2, 0)))
        plt.title(f"GT: {labels[i].item()} | Pred: {pred1[i].item()}", fontsize=10)
        plt.axis("off")

    plt.show()
```

In [76]:
```
def main():
    # Training settings
    parser = argparse.ArgumentParser(description='TorchVision CIFAR10 Example')
    parser.add_argument('--batch-size', type=int, default=4, metavar='N',
                        help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=1, metavar='N',
                        help='number of epochs to train (default: 14)')
    parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                        help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                        help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--no-mps', action='store_true', default=False,
                        help='disables macOS GPU training')
    parser.add_argument('--dry-run', action='store_true', default=False,
                        help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                        help='how many batches to wait before logging training stat
    parser.add_argument('--save-model', action='store_true', default=True,
                        help='For Saving the current Model')

    args = parser.parse_args(['--batch-size', '64', '--test-batch-size', '1000', '-
    use_cuda = not args.no_cuda and torch.cuda.is_available()
```

```
    use_mps = not args.no_mps and torch.backends.mps.is_available()

    torch.manual_seed(args.seed)

    # if use_cuda:
    #     device = torch.device("cuda")
    # elif use_mps:
    #     device = torch.device("mps")
    device = torch.device("cpu")

    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    transform=transforms.Compose([
        transforms.ToTensor(),
        #transforms.Normalize((0.8764,), (0.301,))
        ])
# restore the normalisation to default values

    dataset1 = torchvision.datasets.CIFAR10(root='./data', train=True, download=Tru
                       transform=transform)
    dataset2 = torchvision.datasets.CIFAR10(root='./data', train=False, download=Tr
                       transform=transform)

    train_loader = torch.utils.data.DataLoader(dataset1,**train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    model = Net().to(device)
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        show_(model,  test_loader)
        scheduler.step()

    if args.save_model:
        torch.save(model.state_dict(), "CIFAR10_cnn.pt")
```

In [77]:
```
if __name__ == '__main__':
    main()
```

```
Files already downloaded and verified
Files already downloaded and verified
Train Epoch: 1 [0/50000 (0%)]   Loss: 2.306554
Train Epoch: 1 [640/50000 (1%)] Loss: 2.297998
Train Epoch: 1 [1280/50000 (3%)]        Loss: 2.300219
Train Epoch: 1 [1920/50000 (4%)]        Loss: 2.332988
Train Epoch: 1 [2560/50000 (5%)]        Loss: 2.219170
Train Epoch: 1 [3200/50000 (6%)]        Loss: 2.196541
Train Epoch: 1 [3840/50000 (8%)]        Loss: 2.196548
Train Epoch: 1 [4480/50000 (9%)]        Loss: 2.242943
Train Epoch: 1 [5120/50000 (10%)]       Loss: 2.078732
Train Epoch: 1 [5760/50000 (12%)]       Loss: 2.149945
Train Epoch: 1 [6400/50000 (13%)]       Loss: 1.909985
Train Epoch: 1 [7040/50000 (14%)]       Loss: 1.918373
Train Epoch: 1 [7680/50000 (15%)]       Loss: 2.022701
Train Epoch: 1 [8320/50000 (17%)]       Loss: 2.103451
Train Epoch: 1 [8960/50000 (18%)]       Loss: 1.845189
Train Epoch: 1 [9600/50000 (19%)]       Loss: 2.072267
Train Epoch: 1 [10240/50000 (20%)]      Loss: 1.731654
Train Epoch: 1 [10880/50000 (22%)]      Loss: 1.860479
Train Epoch: 1 [11520/50000 (23%)]      Loss: 1.833301
Train Epoch: 1 [12160/50000 (24%)]      Loss: 1.952206
Train Epoch: 1 [12800/50000 (26%)]      Loss: 1.749430
Train Epoch: 1 [13440/50000 (27%)]      Loss: 1.766858
Train Epoch: 1 [14080/50000 (28%)]      Loss: 1.871141
Train Epoch: 1 [14720/50000 (29%)]      Loss: 1.853204
Train Epoch: 1 [15360/50000 (31%)]      Loss: 2.068903
Train Epoch: 1 [16000/50000 (32%)]      Loss: 1.862259
Train Epoch: 1 [16640/50000 (33%)]      Loss: 1.678312
Train Epoch: 1 [17280/50000 (35%)]      Loss: 1.726234
Train Epoch: 1 [17920/50000 (36%)]      Loss: 1.653036
Train Epoch: 1 [18560/50000 (37%)]      Loss: 1.602842
Train Epoch: 1 [19200/50000 (38%)]      Loss: 1.648876
Train Epoch: 1 [19840/50000 (40%)]      Loss: 1.975251
Train Epoch: 1 [20480/50000 (41%)]      Loss: 1.607041
Train Epoch: 1 [21120/50000 (42%)]      Loss: 1.593012
Train Epoch: 1 [21760/50000 (43%)]      Loss: 1.595536
Train Epoch: 1 [22400/50000 (45%)]      Loss: 1.707301
Train Epoch: 1 [23040/50000 (46%)]      Loss: 1.624081
Train Epoch: 1 [23680/50000 (47%)]      Loss: 1.653694
Train Epoch: 1 [24320/50000 (49%)]      Loss: 1.544035
Train Epoch: 1 [24960/50000 (50%)]      Loss: 1.609733
Train Epoch: 1 [25600/50000 (51%)]      Loss: 1.730791
Train Epoch: 1 [26240/50000 (52%)]      Loss: 1.587650
Train Epoch: 1 [26880/50000 (54%)]      Loss: 1.967132
Train Epoch: 1 [27520/50000 (55%)]      Loss: 1.552680
Train Epoch: 1 [28160/50000 (56%)]      Loss: 1.551827
Train Epoch: 1 [28800/50000 (58%)]      Loss: 1.446637
Train Epoch: 1 [29440/50000 (59%)]      Loss: 1.301654
Train Epoch: 1 [30080/50000 (60%)]      Loss: 1.615399
Train Epoch: 1 [30720/50000 (61%)]      Loss: 1.428114
Train Epoch: 1 [31360/50000 (63%)]      Loss: 1.465205
Train Epoch: 1 [32000/50000 (64%)]      Loss: 1.436392
Train Epoch: 1 [32640/50000 (65%)]      Loss: 1.484613
Train Epoch: 1 [33280/50000 (66%)]      Loss: 1.337771
Train Epoch: 1 [33920/50000 (68%)]      Loss: 1.979562
```

```
Train Epoch: 1 [34560/50000 (69%)]      Loss: 1.544573
Train Epoch: 1 [35200/50000 (70%)]      Loss: 1.629724
Train Epoch: 1 [35840/50000 (72%)]      Loss: 1.500942
Train Epoch: 1 [36480/50000 (73%)]      Loss: 1.497993
Train Epoch: 1 [37120/50000 (74%)]      Loss: 1.570508
Train Epoch: 1 [37760/50000 (75%)]      Loss: 1.331535
Train Epoch: 1 [38400/50000 (77%)]      Loss: 1.598611
Train Epoch: 1 [39040/50000 (78%)]      Loss: 1.433401
Train Epoch: 1 [39680/50000 (79%)]      Loss: 1.498942
Train Epoch: 1 [40320/50000 (81%)]      Loss: 1.884073
Train Epoch: 1 [40960/50000 (82%)]      Loss: 1.644880
Train Epoch: 1 [41600/50000 (83%)]      Loss: 1.497560
Train Epoch: 1 [42240/50000 (84%)]      Loss: 1.679169
Train Epoch: 1 [42880/50000 (86%)]      Loss: 1.502085
Train Epoch: 1 [43520/50000 (87%)]      Loss: 1.464834
Train Epoch: 1 [44160/50000 (88%)]      Loss: 1.534443
Train Epoch: 1 [44800/50000 (90%)]      Loss: 1.443520
Train Epoch: 1 [45440/50000 (91%)]      Loss: 1.608087
Train Epoch: 1 [46080/50000 (92%)]      Loss: 1.621673
Train Epoch: 1 [46720/50000 (93%)]      Loss: 1.587801
Train Epoch: 1 [47360/50000 (95%)]      Loss: 1.409627
Train Epoch: 1 [48000/50000 (96%)]      Loss: 1.323767
Train Epoch: 1 [48640/50000 (97%)]      Loss: 1.396524
Train Epoch: 1 [49280/50000 (98%)]      Loss: 1.206007
Train Epoch: 1 [49920/50000 (100%)]     Loss: 1.515292

Test set: Average loss: 1.3673, Accuracy: 5067/10000 (51%)

GT #1: 4 | Prediction #1: 4
GT #2: 2 | Prediction #2: 5
GT #3: 1 | Prediction #3: 1
GT #4: 8 | Prediction #4: 8
GT #5: 2 | Prediction #5: 2
GT #6: 8 | Prediction #6: 8
GT #7: 2 | Prediction #7: 2
GT #8: 8 | Prediction #8: 3
GT #9: 1 | Prediction #9: 1
```


GT: 4 | Pred: 4
GT: 2 | Pred: 5
GT: 1 | Pred: 1
GT: 8 | Pred: 8
GT: 2 | Pred: 2
GT: 8 | Pred: 8
GT: 2 | Pred: 2
GT: 8 | Pred: 3
GT: 1 | Pred: 1

In [ ]:

In [ ]: