In [1]:

```
import torch
# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

```
/home/jannie/.conda/envs/DL/lib/python3.8/site-packages/tqdm/auto.py:22: T
qdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Out[1]:

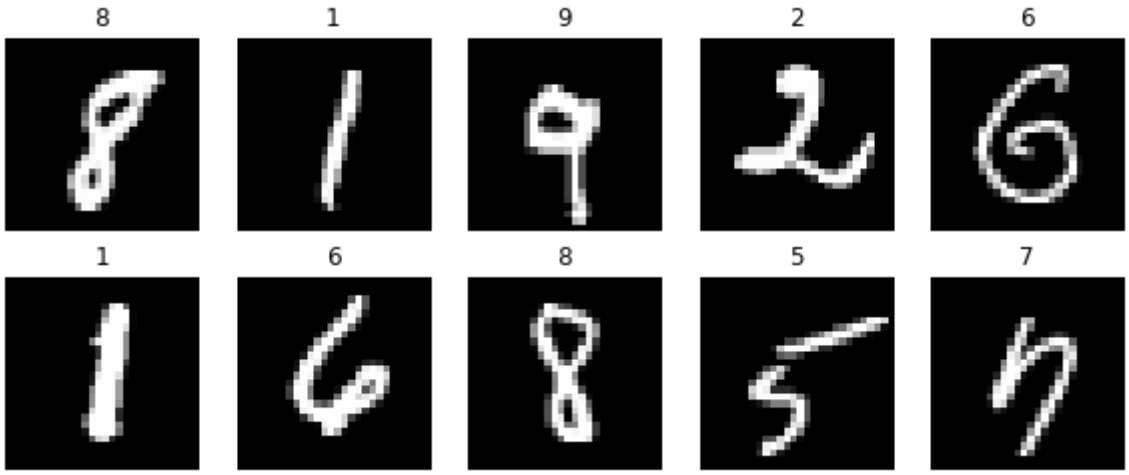
```
device(type='cuda')
```

In [2]:

```
from torchvision import datasets, transforms
from torchvision.transforms import ToTensor
from torch import nn
import torch.nn.functional as F

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

val_data, test_data = torch.utils.data.random_split(test_data, [int(0.9 * len(test_data
)), int(0.1 * len(test_data))])
```

In [3]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

figure = plt.figure(figsize=(10, 4))
cols, rows = 5, 2
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(train_data), size=(1,)).item()
    img, label = train_data[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(label)
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```



In [4]:

```
from torch.utils.data import DataLoader
loaders = {
    'train' : torch.utils.data.DataLoader(train_data,
                                          batch_size=200,
                                          shuffle=True,
                                          num_workers=1),

    'test'  : torch.utils.data.DataLoader(test_data,
                                          batch_size=100,
                                          shuffle=False,
                                          num_workers=1),

    'validate' : torch.utils.data.DataLoader(val_data,
                                          batch_size=100,
                                          shuffle=False,
                                          num_workers=1)
}
```

In [5]:

```
class vanilla_RNN(nn.Module):
    def __init__(self):

        super().__init__()

        self.rnn = nn.RNN(input_size = 28,
                          hidden_size = 128,
                          num_layers = 1,
                          bidirectional = False,
                          batch_first = True)
        self.fc = nn.Linear(128, 10)

    def forward(self, x):

        output, hidden = self.rnn(x)
        out = self.fc(output[:,-1,:])

        return out
```

In [6]:

```
class vanilla_LSTM(nn.Module):

    def __init__(self):
        super(vanilla_LSTM, self).__init__()
        self.hidden_size = 128
        self.num_layers = 1
        self.lstm = nn.LSTM(input_size = 28, hidden_size = 128, num_layers = 1, batch_f
irst=True)
        self.fc = nn.Linear(128, 10)

    def forward(self, x):
        # Set initial hidden and cell states
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
        # Passing in the input and hidden state into the model and  obtaining outputs
        out, hidden = self.lstm(x, (h0, c0))  # out: tensor of shape (batch_size, seq_l
ength, hidden_size)

        #Reshaping the outputs such that it can be fit into the fully connected layer
        out = self.fc(out[:, -1, :])
        return out
```

In [7]:

```
class vanilla_GRU(nn.Module):

    def __init__(self): #class constructor with params for hidden layer and input size
        super(vanilla_GRU, self).__init__() #calls the parent constructor

        #configuring the RNN
        self.gru = nn.GRU(input_size = 28,
                          hidden_size = 128,
                          num_layers = 1,
                          batch_first = True)

        #we want to use the output of the Hidden Layer for the next time step
        self.fc = nn.Linear(128,10) #as size of the output is 10

    def forward(self,x): #defines the forward pass and also the structure of the networ
k thus helping backprop

        h_0 = torch.zeros(1, x.size(0), 128).to(device)
        out, h_n = self.gru(x, (h_0))

        #we want the output at the last time step alone
        out = self.fc(out[:,-1,:]) #obtain the output of the last hidden state
        return out
```

In [8]:

```
class bidirectional_RNN(nn.Module):
    def __init__(self):

        super().__init__()

        self.rnn = nn.RNN(input_size = 28,
                          hidden_size = 128,
                          num_layers = 2,
                          bidirectional = True,
                          batch_first = True)
        self.fc = nn.Linear(128*2, 10)

    def forward(self, x):

        output, hidden = self.rnn(x)
        hidden = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim = 1)
        out = self.fc(hidden)

        return out
```

In [9]:

```
class bidirectional_LSTM(nn.Module):

    def __init__(self):
        super(bidirectional_LSTM, self).__init__()
        self.input_dim = 28
        self.hidden_dim = 128
        self.batch_size = 100
        self.num_layers = 2

        #Define the initial linear hidden layer
        self.init_linear = nn.Linear(self.input_dim, self.input_dim)

        # Define the LSTM layer
        self.lstm = nn.LSTM(self.input_dim, self.hidden_dim, self.num_layers, batch_fir
st=True, bidirectional=True)

        # Define the output layer
        self.linear = nn.Linear(self.hidden_dim * 2, 10)

    def init_hidden(self):
        # This is what we'll initialise our hidden state as
        return (torch.zeros(self.num_layers, self.batch_size, self.hidden_dim),
                torch.zeros(self.num_layers, self.batch_size, self.hidden_dim))

    def forward(self, input):
        #Forward pass through initial hidden layer
        linear_input = self.init_linear(input)
        # Forward pass through LSTM layer
        # shape of lstm_out: [batch_size, input_size ,hidden_dim]
        # shape of self.hidden: (a, b), where a and b both
        # have shape (batch_size, num_layers, hidden_dim).
        out, self.hidden = self.lstm(linear_input)

        y_pred = self.linear(out[:, -1, :])
        return y_pred
```

In [10]:

```
sequence_length = 28
input_size = 28
hidden_size = 128
num_layers = 2
num_classes = 10
batch_size = 100
num_epochs = 20
learning_rate = 0.001
```

In [11]:

```
from torch import optim
loss_func = nn.CrossEntropyLoss()
```

In [12]:

```
def train(no_epochs, model, loaders):
    train_loss = list()
    val_loss = list()
    pred_accuracy = list()
    best_val_loss = 1
    for epoch in range(no_epochs):
        total_train_loss = 0
        total_val_loss = 0

        model.train()
        # training
        for itr, (images, labels) in enumerate(loaders['train']):

            images = images.reshape(-1, sequence_length, input_size).to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(images)
            loss = loss_func(outputs, labels)
            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_train_loss += loss.item()

        total_train_loss = total_train_loss / (itr + 1)
        train_loss.append(total_train_loss)

        # validation
        model.eval()
        total = 0
        for itr, (images, labels) in enumerate(loaders['validate']):
            images = images.reshape(-1, sequence_length, input_size).to(device)
            labels = labels.to(device)
            outputs = model(images)

            loss = loss_func(outputs, labels)
            total_val_loss += loss.item()

            pred = torch.nn.functional.softmax(outputs, dim=1)
            for i, p in enumerate(pred):
                if labels[i] == torch.max(p.data, 0)[1]:
                    total = total + 1

        accuracy = total / len(val_data)
        pred_accuracy.append(accuracy)

        total_val_loss = total_val_loss / (itr + 1)
        val_loss.append(total_val_loss)

        print('\nEpoch: {}/{}, Train Loss: {:.8f}, Val Loss: {:.8f}, Val Accuracy: {:.8
f}'.format(epoch + 1, no_epochs, total_train_loss, total_val_loss, accuracy))

        if total_val_loss < best_val_loss:
            best_val_loss = total_val_loss
            print("Saving the model state dictionary for Epoch: {} with Validation los
s: {:.8f}".format(epoch + 1, total_val_loss))
            model_state = model.state_dict()
```

```
    fig=plt.figure(figsize=(10, 4))
    plt.subplot(1,2,1)
    plt.plot(np.arange(1, no_epochs+1), train_loss, label="Train loss")
    plt.plot(np.arange(1, no_epochs+1), val_loss, label="Validation loss")
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title("Loss Plot")
    plt.legend(loc='upper right')

    plt.subplot(1,2,2)
    plt.plot(np.arange(1, no_epochs+1), pred_accuracy, label="Prediction accuracy")
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title("Accuracy plot")
    plt.legend(loc='lower right')
    return model_state
```

In [13]:

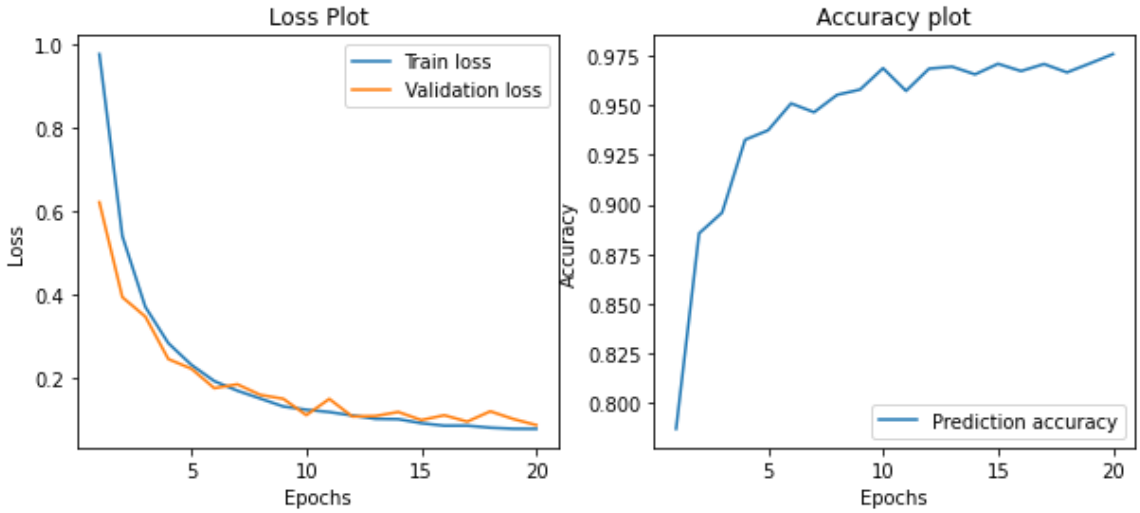
```
def evaluate(model, loaders):

    model.eval()
    total_test_loss = 0
    total = 0

    with torch.no_grad():

        for itr, (images, labels) in enumerate(loaders['test']):
            images = images.reshape(-1, sequence_length, input_size).to(device)
            labels = labels.to(device)
            outputs = model(images)

            loss = loss_func(outputs, labels)
            total_test_loss += loss.item()

            pred = torch.nn.functional.softmax(outputs, dim=1)
            for i, p in enumerate(pred):
                if labels[i] == torch.max(p.data, 0)[1]:
                    total = total + 1

        accuracy = total / len(test_data)
        total_test_loss = total_test_loss / (itr + 1)

    return loss, accuracy
```

In [14]:

```
model = vanilla_RNN().to(device)
optimizer = optim.Adam(model.parameters(), lr = 0.001)
model_state = train(num_epochs, model, loaders)
torch.save(model_state, 'vanilla_RNN_model.pt')
model.load_state_dict(torch.load('vanilla_RNN_model.pt'))

test_loss, test_acc = evaluate(model, loaders)

print('Vanilla RNN model results')
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Epoch: 1/20, Train Loss: 0.97517774, Val Loss: 0.62048852, Val Accuracy:
0.78677778
Saving the model state dictionary for Epoch: 1 with Validation loss: 0.620
48852

Epoch: 2/20, Train Loss: 0.53943257, Val Loss: 0.39307294, Val Accuracy:
0.88544444
Saving the model state dictionary for Epoch: 2 with Validation loss: 0.393
07294

Epoch: 3/20, Train Loss: 0.36983069, Val Loss: 0.34663341, Val Accuracy:
0.89588889
Saving the model state dictionary for Epoch: 3 with Validation loss: 0.346
63341

Epoch: 4/20, Train Loss: 0.28300545, Val Loss: 0.24486435, Val Accuracy:
0.93266667
Saving the model state dictionary for Epoch: 4 with Validation loss: 0.244
86435

Epoch: 5/20, Train Loss: 0.23115147, Val Loss: 0.22190341, Val Accuracy:
0.93744444
Saving the model state dictionary for Epoch: 5 with Validation loss: 0.221
90341

Epoch: 6/20, Train Loss: 0.19208552, Val Loss: 0.17521200, Val Accuracy:
0.95100000
Saving the model state dictionary for Epoch: 6 with Validation loss: 0.175
21200

Epoch: 7/20, Train Loss: 0.16924073, Val Loss: 0.18449313, Val Accuracy:
0.94655556

Epoch: 8/20, Train Loss: 0.15035551, Val Loss: 0.15909125, Val Accuracy:
0.95533333
Saving the model state dictionary for Epoch: 8 with Validation loss: 0.159
09125

Epoch: 9/20, Train Loss: 0.13140099, Val Loss: 0.14979135, Val Accuracy:
0.95800000
Saving the model state dictionary for Epoch: 9 with Validation loss: 0.149
79135

Epoch: 10/20, Train Loss: 0.12359765, Val Loss: 0.11071366, Val Accuracy:
0.96877778
Saving the model state dictionary for Epoch: 10 with Validation loss: 0.11
071366

Epoch: 11/20, Train Loss: 0.11804831, Val Loss: 0.14940575, Val Accuracy:
0.95733333

Epoch: 12/20, Train Loss: 0.10976868, Val Loss: 0.10739222, Val Accuracy:
0.96855556
Saving the model state dictionary for Epoch: 12 with Validation loss: 0.10
739222

Epoch: 13/20, Train Loss: 0.10170116, Val Loss: 0.10854469, Val Accuracy:
0.96955556

Epoch: 14/20, Train Loss: 0.10097688, Val Loss: 0.11820927, Val Accuracy:
0.96566667
```

```
Epoch: 15/20, Train Loss: 0.09142614, Val Loss: 0.09886264, Val Accuracy:
0.97100000
Saving the model state dictionary for Epoch: 15 with Validation loss: 0.09
886264

Epoch: 16/20, Train Loss: 0.08539582, Val Loss: 0.11037436, Val Accuracy:
0.96733333

Epoch: 17/20, Train Loss: 0.08553846, Val Loss: 0.09514633, Val Accuracy:
0.97088889
Saving the model state dictionary for Epoch: 17 with Validation loss: 0.09
514633

Epoch: 18/20, Train Loss: 0.08058733, Val Loss: 0.11978015, Val Accuracy:
0.96666667

Epoch: 19/20, Train Loss: 0.07806271, Val Loss: 0.10143347, Val Accuracy:
0.97122222

Epoch: 20/20, Train Loss: 0.07783107, Val Loss: 0.08683704, Val Accuracy:
0.97588889
Saving the model state dictionary for Epoch: 20 with Validation loss: 0.08
683704
Vanilla RNN model results
Test Loss: 0.088 | Test Acc: 97.20%
```

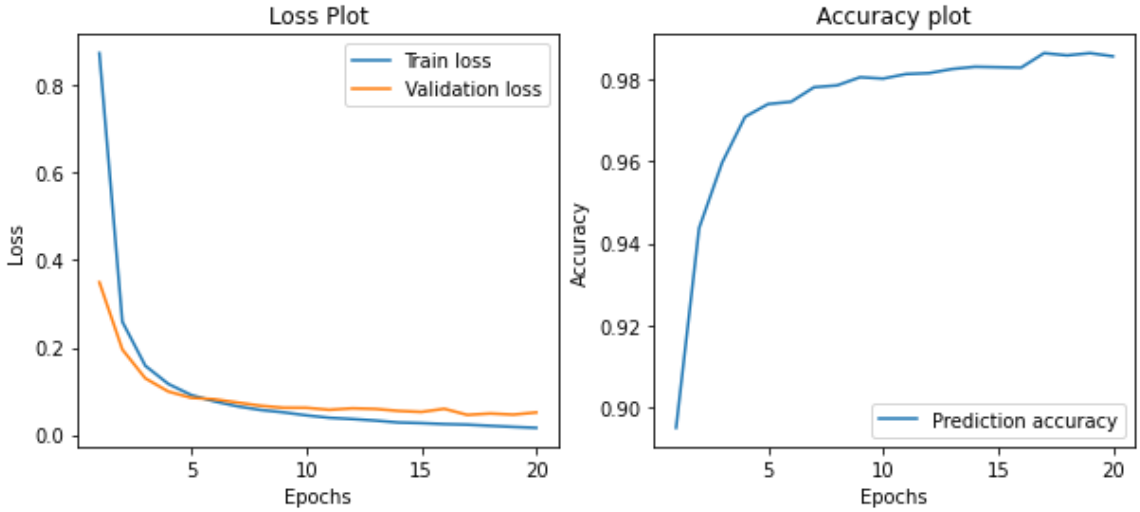


In [15]:

```
model = vanilla_GRU().to(device)
optimizer = optim.Adam(model.parameters(), lr = 0.001)
model_state = train(num_epochs, model, loaders)
torch.save(model_state, 'vanilla_GRU_model.pt')
model.load_state_dict(torch.load('vanilla_GRU_model.pt'))

test_loss, test_acc = evaluate(model, loaders)

print('Vanilla GRU model results')
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Epoch: 1/20, Train Loss: 0.87444531, Val Loss: 0.34955904, Val Accuracy:
0.89488889
Saving the model state dictionary for Epoch: 1 with Validation loss: 0.349
55904

Epoch: 2/20, Train Loss: 0.25915544, Val Loss: 0.19548014, Val Accuracy:
0.94366667
Saving the model state dictionary for Epoch: 2 with Validation loss: 0.195
48014

Epoch: 3/20, Train Loss: 0.15820616, Val Loss: 0.12942067, Val Accuracy:
0.95966667
Saving the model state dictionary for Epoch: 3 with Validation loss: 0.129
42067

Epoch: 4/20, Train Loss: 0.11661621, Val Loss: 0.09923165, Val Accuracy:
0.97088889
Saving the model state dictionary for Epoch: 4 with Validation loss: 0.099
23165

Epoch: 5/20, Train Loss: 0.09102242, Val Loss: 0.08471331, Val Accuracy:
0.97400000
Saving the model state dictionary for Epoch: 5 with Validation loss: 0.084
71331

Epoch: 6/20, Train Loss: 0.07666632, Val Loss: 0.08154206, Val Accuracy:
0.97455556
Saving the model state dictionary for Epoch: 6 with Validation loss: 0.081
54206

Epoch: 7/20, Train Loss: 0.06546477, Val Loss: 0.07406869, Val Accuracy:
0.97811111
Saving the model state dictionary for Epoch: 7 with Validation loss: 0.074
06869

Epoch: 8/20, Train Loss: 0.05705995, Val Loss: 0.06674988, Val Accuracy:
0.97855556
Saving the model state dictionary for Epoch: 8 with Validation loss: 0.066
74988

Epoch: 9/20, Train Loss: 0.05155889, Val Loss: 0.06206405, Val Accuracy:
0.98055556
Saving the model state dictionary for Epoch: 9 with Validation loss: 0.062
06405

Epoch: 10/20, Train Loss: 0.04461008, Val Loss: 0.06194239, Val Accuracy:
0.98022222
Saving the model state dictionary for Epoch: 10 with Validation loss: 0.06
194239

Epoch: 11/20, Train Loss: 0.03903708, Val Loss: 0.05731810, Val Accuracy:
0.98133333
Saving the model state dictionary for Epoch: 11 with Validation loss: 0.05
731810

Epoch: 12/20, Train Loss: 0.03620972, Val Loss: 0.06070051, Val Accuracy:
0.98155556

Epoch: 13/20, Train Loss: 0.03272778, Val Loss: 0.05929801, Val Accuracy:
0.98255556
```

```
Epoch: 14/20, Train Loss: 0.02832720, Val Loss: 0.05482672, Val Accuracy:
0.98311111
Saving the model state dictionary for Epoch: 14 with Validation loss: 0.05
482672

Epoch: 15/20, Train Loss: 0.02688796, Val Loss: 0.05243583, Val Accuracy:
0.98300000
Saving the model state dictionary for Epoch: 15 with Validation loss: 0.05
243583

Epoch: 16/20, Train Loss: 0.02463290, Val Loss: 0.05966445, Val Accuracy:
0.98288889

Epoch: 17/20, Train Loss: 0.02337500, Val Loss: 0.04567163, Val Accuracy:
0.98644444
Saving the model state dictionary for Epoch: 17 with Validation loss: 0.04
567163

Epoch: 18/20, Train Loss: 0.02067773, Val Loss: 0.04884062, Val Accuracy:
0.98588889

Epoch: 19/20, Train Loss: 0.01816293, Val Loss: 0.04629968, Val Accuracy:
0.98644444

Epoch: 20/20, Train Loss: 0.01602605, Val Loss: 0.05129423, Val Accuracy:
0.98566667
Vanilla GRU model results
Test Loss: 0.024 | Test Acc: 98.20%
```

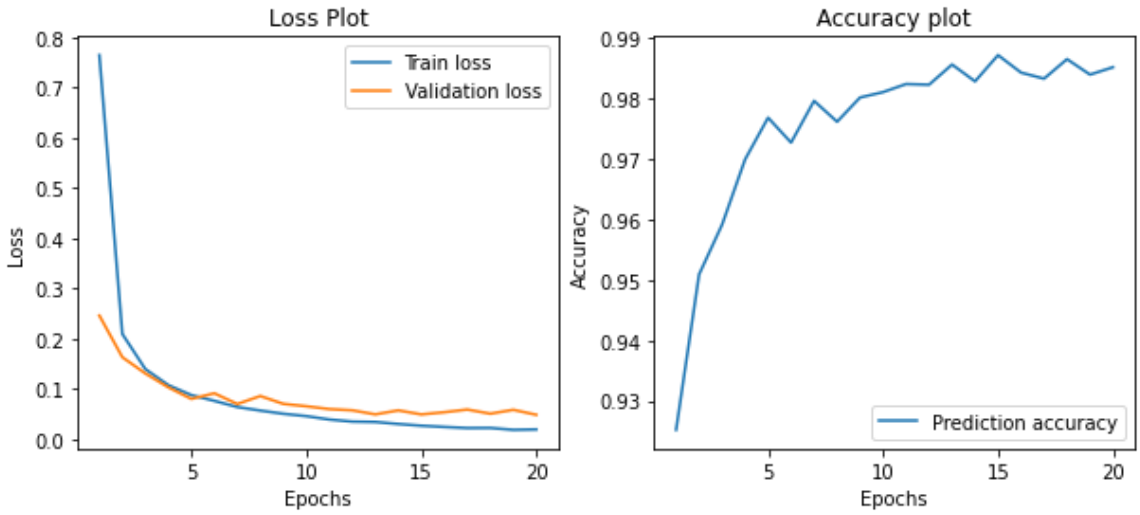


In [16]:

```
model = vanilla_LSTM().to(device)
optimizer = optim.Adam(model.parameters(), lr = 0.001)
model_state = train(num_epochs, model, loaders)
torch.save(model_state, 'vanilla_LSTM_model.pt')
model.load_state_dict(torch.load('vanilla_LSTM_model.pt'))

test_loss, test_acc = evaluate(model, loaders)

print('Vanilla LSTM model results')
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Epoch: 1/20, Train Loss: 0.76411998, Val Loss: 0.24644218, Val Accuracy:
0.92522222
Saving the model state dictionary for Epoch: 1 with Validation loss: 0.246
44218

Epoch: 2/20, Train Loss: 0.20979486, Val Loss: 0.16354853, Val Accuracy:
0.95088889
Saving the model state dictionary for Epoch: 2 with Validation loss: 0.163
54853

Epoch: 3/20, Train Loss: 0.14000265, Val Loss: 0.13134005, Val Accuracy:
0.95911111
Saving the model state dictionary for Epoch: 3 with Validation loss: 0.131
34005

Epoch: 4/20, Train Loss: 0.10803818, Val Loss: 0.10363672, Val Accuracy:
0.96988889
Saving the model state dictionary for Epoch: 4 with Validation loss: 0.103
63672

Epoch: 5/20, Train Loss: 0.08827737, Val Loss: 0.08067820, Val Accuracy:
0.97677778
Saving the model state dictionary for Epoch: 5 with Validation loss: 0.080
67820

Epoch: 6/20, Train Loss: 0.07685473, Val Loss: 0.09187520, Val Accuracy:
0.97266667

Epoch: 7/20, Train Loss: 0.06455282, Val Loss: 0.07051696, Val Accuracy:
0.97955556
Saving the model state dictionary for Epoch: 7 with Validation loss: 0.070
51696

Epoch: 8/20, Train Loss: 0.05749893, Val Loss: 0.08638609, Val Accuracy:
0.97611111

Epoch: 9/20, Train Loss: 0.05113921, Val Loss: 0.07084790, Val Accuracy:
0.98011111

Epoch: 10/20, Train Loss: 0.04643240, Val Loss: 0.06608514, Val Accuracy:
0.98100000
Saving the model state dictionary for Epoch: 10 with Validation loss: 0.06
608514

Epoch: 11/20, Train Loss: 0.03968920, Val Loss: 0.06045963, Val Accuracy:
0.98233333
Saving the model state dictionary for Epoch: 11 with Validation loss: 0.06
045963

Epoch: 12/20, Train Loss: 0.03559493, Val Loss: 0.05799250, Val Accuracy:
0.98222222
Saving the model state dictionary for Epoch: 12 with Validation loss: 0.05
799250

Epoch: 13/20, Train Loss: 0.03484806, Val Loss: 0.04995423, Val Accuracy:
0.98555556
Saving the model state dictionary for Epoch: 13 with Validation loss: 0.04
995423

Epoch: 14/20, Train Loss: 0.03085302, Val Loss: 0.05788246, Val Accuracy:
0.98277778
```

```
Epoch: 15/20, Train Loss: 0.02741418, Val Loss: 0.04985972, Val Accuracy:
0.98711111
Saving the model state dictionary for Epoch: 15 with Validation loss: 0.04
985972

Epoch: 16/20, Train Loss: 0.02497833, Val Loss: 0.05421448, Val Accuracy:
0.98422222

Epoch: 17/20, Train Loss: 0.02276444, Val Loss: 0.05955724, Val Accuracy:
0.98322222

Epoch: 18/20, Train Loss: 0.02296204, Val Loss: 0.05128547, Val Accuracy:
0.98644444

Epoch: 19/20, Train Loss: 0.01920049, Val Loss: 0.05899967, Val Accuracy:
0.98388889

Epoch: 20/20, Train Loss: 0.02002532, Val Loss: 0.04909056, Val Accuracy:
0.98511111
Saving the model state dictionary for Epoch: 20 with Validation loss: 0.04
909056
Vanilla LSTM model results
Test Loss: 0.004 | Test Acc: 98.80%
```

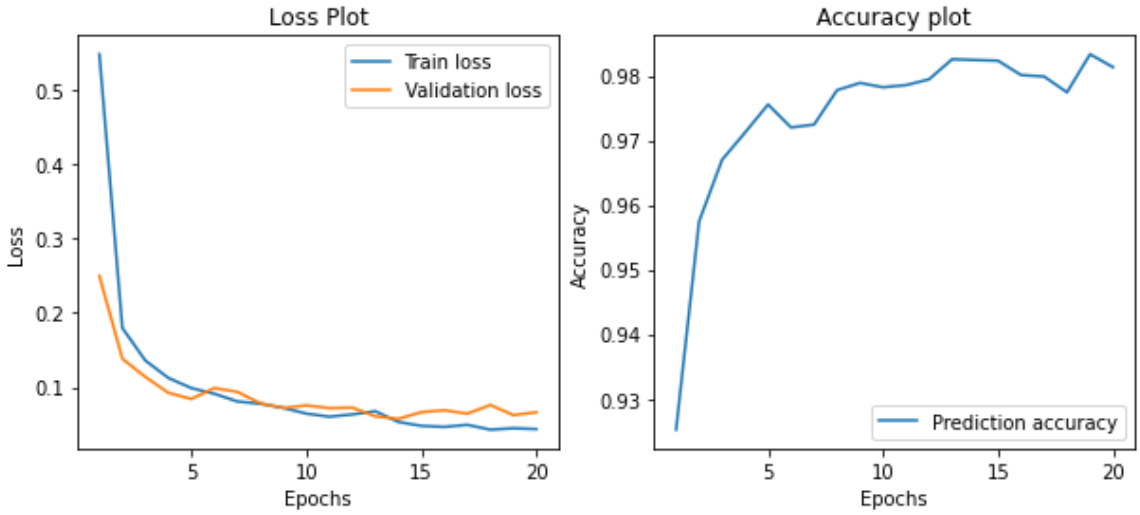


In [17]:

```
model = bidirectional_RNN().to(device)
optimizer = optim.Adam(model.parameters(), lr = 0.001)
model_state = train(num_epochs, model, loaders)
torch.save(model_state, 'bidirectional_RNN.pt')
model.load_state_dict(torch.load('bidirectional_RNN.pt'))

test_loss, test_acc = evaluate(model, loaders)

print('Bidirectional RNN model results')
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Epoch: 1/20, Train Loss: 0.54811556, Val Loss: 0.24979376, Val Accuracy:
0.92522222
Saving the model state dictionary for Epoch: 1 with Validation loss: 0.249
79376

Epoch: 2/20, Train Loss: 0.17975089, Val Loss: 0.13824866, Val Accuracy:
0.95744444
Saving the model state dictionary for Epoch: 2 with Validation loss: 0.138
24866

Epoch: 3/20, Train Loss: 0.13587633, Val Loss: 0.11383420, Val Accuracy:
0.96700000
Saving the model state dictionary for Epoch: 3 with Validation loss: 0.113
83420

Epoch: 4/20, Train Loss: 0.11248182, Val Loss: 0.09241115, Val Accuracy:
0.97122222
Saving the model state dictionary for Epoch: 4 with Validation loss: 0.092
41115

Epoch: 5/20, Train Loss: 0.09883686, Val Loss: 0.08417205, Val Accuracy:
0.97555556
Saving the model state dictionary for Epoch: 5 with Validation loss: 0.084
17205

Epoch: 6/20, Train Loss: 0.09117610, Val Loss: 0.09880965, Val Accuracy:
0.97200000

Epoch: 7/20, Train Loss: 0.08087183, Val Loss: 0.09368221, Val Accuracy:
0.97244444

Epoch: 8/20, Train Loss: 0.07765990, Val Loss: 0.07837296, Val Accuracy:
0.97777778
Saving the model state dictionary for Epoch: 8 with Validation loss: 0.078
37296

Epoch: 9/20, Train Loss: 0.07223659, Val Loss: 0.07208146, Val Accuracy:
0.97888889
Saving the model state dictionary for Epoch: 9 with Validation loss: 0.072
08146

Epoch: 10/20, Train Loss: 0.06445502, Val Loss: 0.07548660, Val Accuracy:
0.97822222

Epoch: 11/20, Train Loss: 0.06030202, Val Loss: 0.07184276, Val Accuracy:
0.97855556
Saving the model state dictionary for Epoch: 11 with Validation loss: 0.07
184276

Epoch: 12/20, Train Loss: 0.06333194, Val Loss: 0.07249329, Val Accuracy:
0.97944444

Epoch: 13/20, Train Loss: 0.06766919, Val Loss: 0.06077226, Val Accuracy:
0.98255556
Saving the model state dictionary for Epoch: 13 with Validation loss: 0.06
077226

Epoch: 14/20, Train Loss: 0.05323501, Val Loss: 0.05736636, Val Accuracy:
0.98244444
Saving the model state dictionary for Epoch: 14 with Validation loss: 0.05
736636
```

```
Epoch: 15/20, Train Loss: 0.04799999, Val Loss: 0.06635465, Val Accuracy:
0.98233333

Epoch: 16/20, Train Loss: 0.04655570, Val Loss: 0.06896956, Val Accuracy:
0.98011111

Epoch: 17/20, Train Loss: 0.04936698, Val Loss: 0.06437075, Val Accuracy:
0.97988889

Epoch: 18/20, Train Loss: 0.04285524, Val Loss: 0.07605352, Val Accuracy:
0.97744444

Epoch: 19/20, Train Loss: 0.04474034, Val Loss: 0.06240450, Val Accuracy:
0.98333333

Epoch: 20/20, Train Loss: 0.04365080, Val Loss: 0.06623910, Val Accuracy:
0.98133333
Bidirectional RNN model results
Test Loss: 0.084 | Test Acc: 97.70%
```

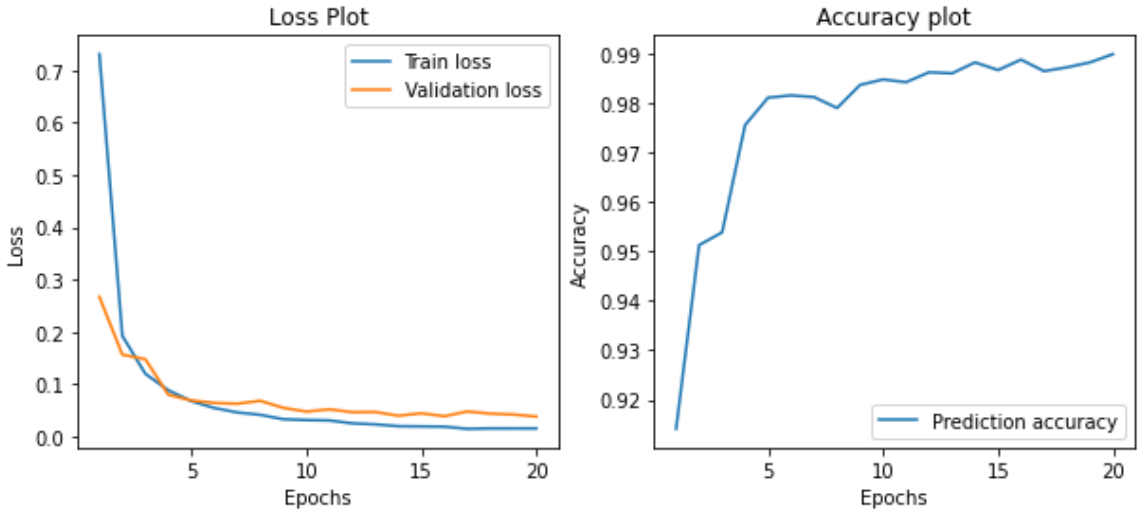


In [18]:

```
model = bidirectional_LSTM().to(device)
optimizer = optim.Adam(model.parameters(), lr = 0.001)
model_state = train(num_epochs, model, loaders)
torch.save(model_state, 'bidirectional_LSTM.pt')
model.load_state_dict(torch.load('bidirectional_LSTM.pt'))

test_loss, test_acc = evaluate(model, loaders)

print('Bidirectional LSTM model results')
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Epoch: 1/20, Train Loss: 0.73112448, Val Loss: 0.26739744, Val Accuracy:
0.91411111
Saving the model state dictionary for Epoch: 1 with Validation loss: 0.267
39744

Epoch: 2/20, Train Loss: 0.19277683, Val Loss: 0.15675487, Val Accuracy:
0.95122222
Saving the model state dictionary for Epoch: 2 with Validation loss: 0.156
75487

Epoch: 3/20, Train Loss: 0.12031965, Val Loss: 0.14809577, Val Accuracy:
0.95377778
Saving the model state dictionary for Epoch: 3 with Validation loss: 0.148
09577

Epoch: 4/20, Train Loss: 0.08871723, Val Loss: 0.08022628, Val Accuracy:
0.97544444
Saving the model state dictionary for Epoch: 4 with Validation loss: 0.080
22628

Epoch: 5/20, Train Loss: 0.06798497, Val Loss: 0.06915358, Val Accuracy:
0.98100000
Saving the model state dictionary for Epoch: 5 with Validation loss: 0.069
15358

Epoch: 6/20, Train Loss: 0.05488413, Val Loss: 0.06452350, Val Accuracy:
0.98144444
Saving the model state dictionary for Epoch: 6 with Validation loss: 0.064
52350

Epoch: 7/20, Train Loss: 0.04640022, Val Loss: 0.06309995, Val Accuracy:
0.98111111
Saving the model state dictionary for Epoch: 7 with Validation loss: 0.063
09995

Epoch: 8/20, Train Loss: 0.04183024, Val Loss: 0.06855873, Val Accuracy:
0.97888889

Epoch: 9/20, Train Loss: 0.03337030, Val Loss: 0.05523058, Val Accuracy:
0.98355556
Saving the model state dictionary for Epoch: 9 with Validation loss: 0.055
23058

Epoch: 10/20, Train Loss: 0.03191328, Val Loss: 0.04802801, Val Accuracy:
0.98466667
Saving the model state dictionary for Epoch: 10 with Validation loss: 0.04
802801

Epoch: 11/20, Train Loss: 0.03113611, Val Loss: 0.05233173, Val Accuracy:
0.98411111

Epoch: 12/20, Train Loss: 0.02562720, Val Loss: 0.04685495, Val Accuracy:
0.98611111
Saving the model state dictionary for Epoch: 12 with Validation loss: 0.04
685495

Epoch: 13/20, Train Loss: 0.02362626, Val Loss: 0.04720469, Val Accuracy:
0.98588889

Epoch: 14/20, Train Loss: 0.02011457, Val Loss: 0.04009928, Val Accuracy:
0.98811111
```

```
Saving the model state dictionary for Epoch: 14 with Validation loss: 0.04
009928

Epoch: 15/20, Train Loss: 0.01967791, Val Loss: 0.04474581, Val Accuracy:
0.98655556

Epoch: 16/20, Train Loss: 0.01922041, Val Loss: 0.03915894, Val Accuracy:
0.98866667
Saving the model state dictionary for Epoch: 16 with Validation loss: 0.03
915894

Epoch: 17/20, Train Loss: 0.01504952, Val Loss: 0.04820582, Val Accuracy:
0.98633333

Epoch: 18/20, Train Loss: 0.01592723, Val Loss: 0.04417105, Val Accuracy:
0.98711111

Epoch: 19/20, Train Loss: 0.01581710, Val Loss: 0.04242246, Val Accuracy:
0.98811111

Epoch: 20/20, Train Loss: 0.01585666, Val Loss: 0.03856109, Val Accuracy:
0.98977778
Saving the model state dictionary for Epoch: 20 with Validation loss: 0.03
856109
Bidirectional LSTM model results
Test Loss: 0.015 | Test Acc: 98.50%
```



In [19]:

```
for itr, (images, labels) in enumerate(loaders['test']):
        images = images.reshape(-1, sequence_length, input_size).to(device)
        labels = labels.to(device)
        outputs = model(images)
        #pred = model(images)
        pred = torch.nn.functional.softmax(outputs, dim=1)
        pred_label = np.zeros((100,1))
        for i, p in enumerate(pred):
            pred_label[i] = int(torch.max(p.data, 0)[1])
```

In [20]:

```
fig=plt.figure(figsize=(20, 10))
for test_images, test_labels in loaders['test']:
    for i in range(1,8):
        sample_image = test_images[i]
        sample_label = pred_label[i]
        img = transforms.ToPILImage(mode='L')(sample_image)
        fig.add_subplot(1, 8, i)
        plt.title(str(sample_label))
        plt.imshow(img)
plt.show()
```