1. Define Hyper-parameters and Device Configuration

```
# Hyper parameters
sequence_length = 28
input_size = 28
hidden_size = 128
num_layers = 10
num_classes = 10
batch_size = 50
num_epochs = 3
learning_rate = 0.001
```

2. Build Model

```
class Classifier(nn.Module):
  def __init__(self, intput_size, hidden_size, num_layers, num_classes,
drop_percent=0.2, model="RNN"):
    super(Classifier, self).__init__()
    self.model = model

    self.hidden_size = hidden_size
    self.num_layers = num_layers
    self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=
True)
    self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_firs
t=True)
    self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=
True)
    self.dropout = nn.Dropout(drop_percent)

    self.fc = nn.Linear(hidden_size, num_classes)

  def forward(self, x):
    # set initial hidden states and cell states
    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(d
evice) # torch.size([2, 50, 128])
    c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(d
evice) # torch.size([2, 50, 128])


    if self.model == "RNN":
      out, _  = self.rnn(x,h0)
    elif self.model == "LSTM":
      out, _  = self.lstm(x, (h0, c0)) # output: tensor [batch_size, se
q_length, hidden_size]
      out = self.dropout(out)
    elif self.model == "GRU":
      out, _  = self.gru(x, h0)
    else:
```

```
            print("choose a model in ['RNN', 'LSTM', 'GRU']")
            raise


        #Decode the hidden state of the last time step
        out = self.dropout(out)
        out = self.fc(out[:,-1,:])

        return out
```

3. Set Loss & Optimizer

```
criterion = nn.BCEWithLogitsLoss().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

4. Train & Test

4.1 RNN

```
model = Classifier(input_size, hidden_size, num_layers, num_classes, mo
del="RNN").to(device)
```

Result

Epoch [3/3], Step[1200/1200], Loss:0.5960

Test Accuracy of RNN model on the 10000 test images: 88.65%

4.2 LSTM

```
model = Classifier(input_size, hidden_size, num_layers, num_classes, mo
del="LSTM").to(device)
```

Result

Epoch [3/3], Step[1200/1200], Loss:0.0806

Test Accuracy of LSTM model on the 10000 test images: 97.11%

4.3 GRU

```
model = Classifier(input_size, hidden_size, num_layers, num_classes, mo
del="GRU").to(device)
```

Result

```
Epoch [1/3], Step[400/1200], Loss:0.6697

Epoch [1/3], Step[800/1200], Loss:0.0822

Epoch [1/3], Step[1200/1200], Loss:0.1668
```

```
Epoch [2/3], Step[400/1200], Loss:0.0415
Epoch [2/3], Step[800/1200], Loss:0.1800
Epoch [2/3], Step[1200/1200], Loss:0.2348
Epoch [3/3], Step[400/1200], Loss:0.0500
Epoch [3/3], Step[800/1200], Loss:0.1116
Epoch [3/3], Step[1200/1200], Loss:0.0149
```

5. Model Save and Load

```
torch.save(model.state_dict(),"model_Choiinsung.pth")
test_model = Classifier(input_size, hidden_size, num_layers, num_classe
s, model="GRU").to(device)

test_model.load_state_dict(torch.load("model_Choiinsung.pth"))
```

6. Final Result

```
Test Accuracy of GRU model on the 10000 test images: 97.23%
```