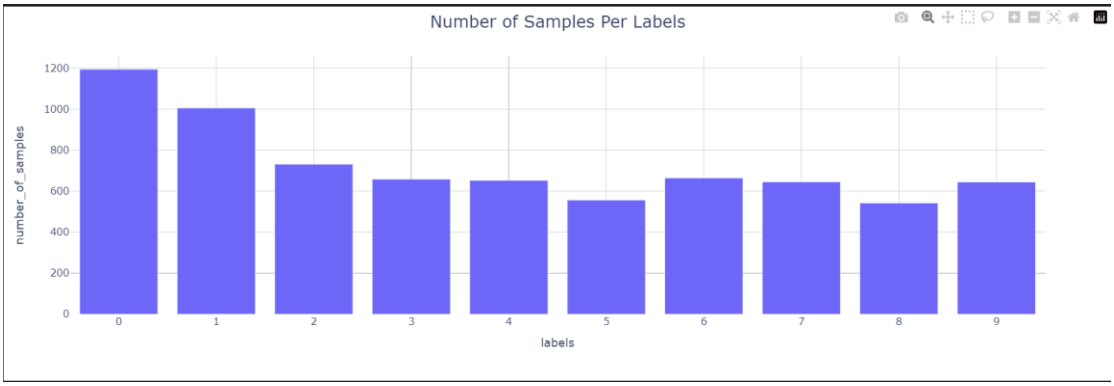
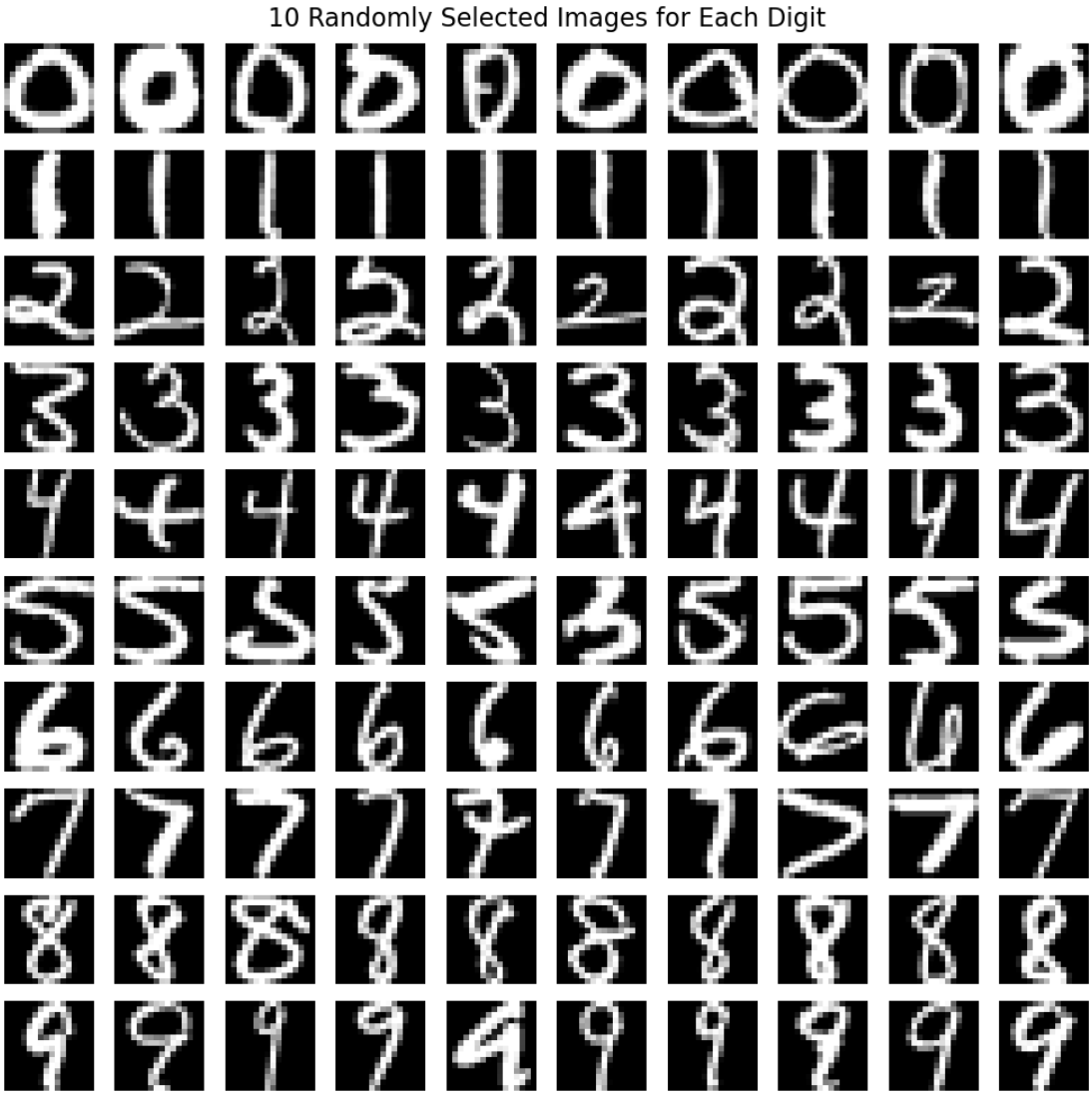
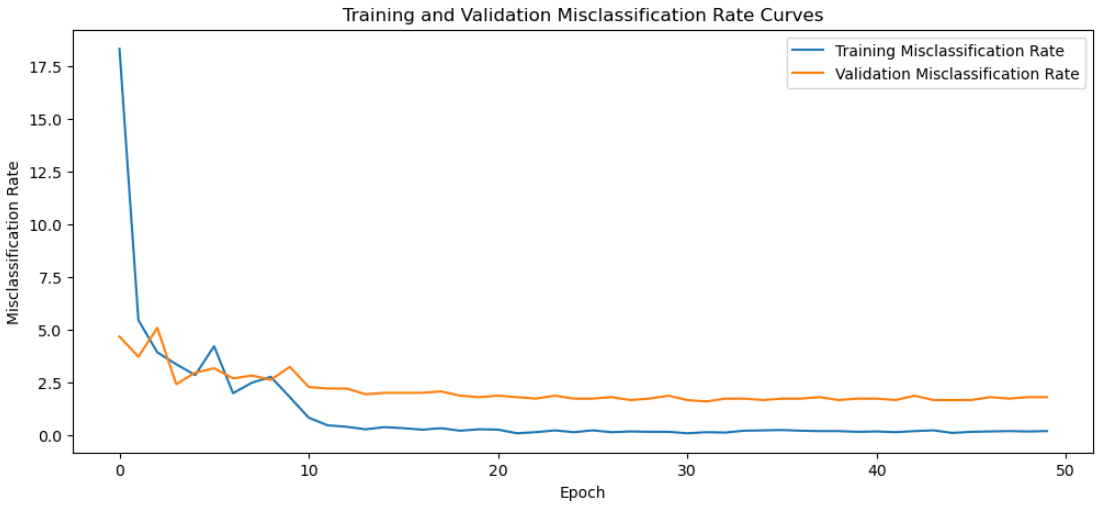
a. Number of images in each digit for the training dataset (Q3) (2 pts)



b. Plots of 10 randomly selected images (Q4) (4 pts)



c. Plot of numbers of epochs versus misclassification rates (Q5b) (4 pts)



d. Reproductivity of the results (4 pts)
Yes, but I use 50 epochs when submit prediction. In this case(epoch=30), the result might be a slightly different.

e. Number of parameters (2 pts)
108490

```
[38]
    myparameter = sum(p.numel() for p in model.parameters() if p.requires_grad)
    myparameter

...  108490
```

f. The difficulty during training (8 pts)
Try to use GPU to train might be the hardest part, but I still figure it out.

g. Explained model architecture (6 pts)
I use 3 layers
First layer is linear input to 256, batch normalization, and relu.
Second layer is linear 256 to 128, batch normalization, and relu.
Third layer is linear 128 to 64, batch normalization, and relu.
Dropout of 0.5, and then is final output layer.
I tried different combination and other layer, but this one has the best result.

```python
class SimpleNN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, 256)  # Increase units to 256 for better feature extraction
        self.bn1 = nn.BatchNorm1d(256)  # Batch normalization after first layer
        self.relu = nn.ReLU()  # ReLU activation

        self.fc2 = nn.Linear(256, 128)  # Second fully connected layer with 128 units
        self.bn2 = nn.BatchNorm1d(128)  # Batch normalization after second layer

        self.fc3 = nn.Linear(128, 64)  # Third layer with 64 units
        self.bn3 = nn.BatchNorm1d(64)  # Batch normalization

        self.dropout = nn.Dropout(0.5)  # Dropout to reduce overfitting

        self.final_fc = nn.Linear(64, output_dim)  # Final output layer

    def forward(self, x):
        x = self.fc1(x)
        x = self.bn1(x)  # Apply batch normalization
        x = self.relu(x)

        x = self.fc2(x)
        x = self.bn2(x)  # Batch normalization
        x = self.relu(x)

        x = self.fc3(x)
        x = self.bn3(x)  # Batch normalization
        x = self.relu(x)

        x = self.dropout(x)  # Apply dropout

        x = self.final_fc(x)  # Output layer (no softmax because CrossEntropyLoss will handle it)
        return x
```