ML

Assignment - 3

Ojasva Singh
2020318

Attempting **Sections A and C**

Section A

1.

a.

Ans 1) Input data $\Rightarrow [1.2, 0.8, 2.0]$

Targets $\Rightarrow [3.0, 2.5, 4.0]$

Learning rate $= 0.01$

(Assumption) Initial weights, $w_1 = 0.4$ & $w_2 = 0.5$

biases, $b_1 = 0.2$ & $b_2 = 0.1$

Function $\rightarrow$ RELU

We need to train the network using mean squared loss.

Forward Pass

$x = 1.2$

$h = \text{Relu}(w_1 x_1 + b_1)$

$= \max(0, w_1 x_1 + b)$

$= \max(0, 0.4(1.2) + 0.2)$

$= \max(0, 0.68)$

$h_1 = 0.68$

$\hat{y} = 0.68$

$= 0.68(w_2) + b_2$

$= 0.68(0.5) + 0.1$

$\boxed{\hat{y}_1 = 0.44}$

$x = 0.8$

$h = \max(0, w_1 x_2 + b_1)$

$= \max(0, 0.4(0.8) + 0.2)$

$= \max(0, 0.52)$

$h_2 = 0.52$

$\hat{y} = 0.52$

$= 0.52(0.5) + 0.1$

$= 0.26 + 0.1$

$\boxed{\hat{y}_2 = 0.36}$

$x = 2.0$

$h = \max(0, 0.4(2) + 0.2)$

$h = \max(0, 1)$

$h_3 = 1$

$\hat{y} = 1$

$= 1(0.5) + 0.1$

$\boxed{\hat{y}_3 = 0.6}$

$h_i = [h_1, h_2, h_3] = [0.68, 0.52, 1]$

$y_i = [0.44, 0.36, 0.6]$

Calculating loss

$$MSE = \frac{1}{3}\left[(3-0.44)^2 + (2.5-0.36)^2 + (4-0.6)^2\right]$$

$$= \frac{1}{3}\left((2.56)^2 + (2.14)^2 + (3.4)^2\right]$$

$$= 1/3\,(6.55 + 11.56 + 4.57) = \frac{22.68}{3}$$

$\boxed{\text{Loss} = 7.56}$

Back propogation

Firstly we'll do it for w_2 & b_2

using chain rule,

$$\Rightarrow \frac{\partial MSE}{\partial w_2} = \frac{\partial MSE}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_2}$$

① ②

① $\frac{\partial MSE}{\partial \hat{y}_i} = \frac{\partial (MSE)}{\partial \hat{y}_i}$ $\left[MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \right]$

$$\frac{\partial MSE}{\partial \hat{y}_i} = \frac{2}{n} (\hat{y}_i - y_i)$$

② $\frac{\partial \hat{y}_i}{\partial w_2} = \frac{\partial (\hat{y}_i)}{\partial w_2}$ $[y_i = w_2 h_i + b_2]$

$$\frac{\partial \hat{y}_i}{\partial w_2} = h_i$$

$$\therefore \boxed{\frac{\partial MSE}{\partial w_2} = \frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \cdot h_i}$$

$$\Rightarrow \frac{\partial MSE}{\partial b_2} = \frac{\partial MSE}{\partial y_i} \cdot \frac{\partial y_i}{\partial b_2} \quad \text{(using chain rule)}$$

$$= \frac{2}{n} (\hat{y}_i - y_i) \cdot \underbrace{\frac{d(w_2 h_i + b_2)}{d b_2}}_{=1}$$

$$\therefore \boxed{\frac{\partial MSE}{\partial b_2} = \frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)}$$

Now calculating for w_1 and b_1.

$$\Rightarrow \frac{\partial MSE}{\partial w_1} = \underset{①}{\frac{\partial MSE}{\partial \hat{y}}} \cdot \underset{②}{\frac{\partial \hat{y}}{\partial h}} \cdot \underset{③}{\frac{\partial h}{\partial w_1}} \quad \text{(chain rule)}$$

① $$\frac{\partial MSE}{\partial \hat{y}} = \frac{2}{n}(\hat{y}_i - y_i)$$

② $$\frac{\partial \hat{y}}{\partial h} = \frac{\partial(w_2 h + b_2)}{\partial h}$$

$$= w_2$$

③ $$\frac{\partial h}{\partial w_1} = \frac{\partial(w_1 x_1 + b_1)}{\partial w_1}$$

$$= x_1$$

$$\boxed{\therefore \frac{\partial MSE}{\partial w_1} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i) \cdot w_2 \cdot x_1}$$

$$\Rightarrow \frac{\partial MSE}{\partial b_1} = \frac{\partial MSE}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h} \cdot \underbrace{\frac{\partial h}{\partial b_1}}$$

$$= \frac{\partial(w_1 x_1 + b_1)}{\partial b_1} = 1$$

$$\boxed{\therefore = \frac{2}{n}\Sigma(\hat{y}_i - y_i) \cdot w_2}$$

Updating the weights from the equations obtained.

$x_i = [1.2, 0.8, 2.0]$

$h_i = [0.68, 0.52, 1]$

$\hat{y}_i = [0.44, 0.36, 0.6]$

$y_i = [3.0, 2.5, 4.0]$

$$\frac{\partial MSE}{\partial w_2} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\cdot h_i$$

$$= \frac{2}{3}\left[(0.44-3.0)\cdot 0.68 + (0.36-2.5)\cdot 0.52 + (0.6-4.0)\cdot 1\right]$$

$$\boxed{\frac{\partial MSE}{\partial w_2} = -4.16}$$

$$\frac{\partial MSE}{\partial b_2} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i) = \frac{2}{3}\left((0.44-3) + (0.36-2.5) + (0.6-4.0)\right)$$

$$\boxed{\frac{\partial MSE}{\partial b_2} = -5.4}$$

$$\frac{\partial MSE}{\partial w_1} = \frac{2}{n}\sum(\hat{y}_i - y_i)\cdot w_2 \cdot x_i = \frac{2}{3}\left[(0.44-3)\cdot 0.5\cdot 1.2 + (0.36-2.5)\, 0.5\cdot 0.8 + (0.6-4.0)\cdot 0.5\cdot 2\right]$$

$$\boxed{\frac{\partial MSE}{\partial w_1} = -3.87}$$

$$\frac{\partial MSE}{\partial b_1} = \frac{2}{n}\sum(\hat{y}_i - y_i)\cdot w_2 = \frac{2}{3}\left[(0.44-0.3)\cdot 0.5 + (0.36-2.5)\, 0.5 + (0.6-4)\cdot 0.5\right]$$

$$\boxed{\frac{\partial MSE}{\partial b_1} = -2.76}$$

New

$$w_1' = w_1 - \eta\frac{\partial MSE}{\partial w_1} = 0.4 - 0.01(-3.87) = 0.4387$$

$$b_1' = b_1 - \eta\frac{\partial MSE}{\partial b_1} = 0.2 - 0.01(-2.76) = 0.2276$$

$$w_2' = w_2 - \eta\frac{\partial MSE}{\partial w_2} = 0.5 - 0.01(-4.16) = 0.5416$$

$$b_2' = b_2 - \eta\frac{\partial MSE}{\partial b_2} = 0.1 - 0.01(-5.4) = 0.154$$

b.

b) Nonlinear SVM trained w/ a Gaussian Kernel

For an unseen test sample the classification rule will be,

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + b\right)$$ [Returns +1 or -1]

$f(x) > 0$ $f(x) < 0$

n: no. of training samples.

α_i: Lagrange's multiplier

y_i: labels

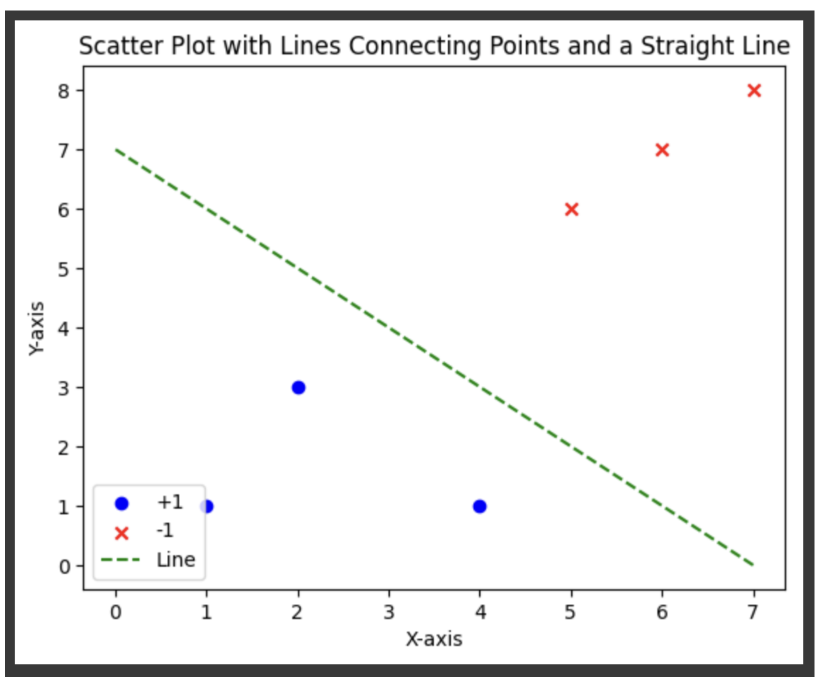
$K(x, x_i)$: kernel f^n.

$$= \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right)$$

b: bias.

c.

i. Yes the points are linearly separable, you can see from the graph



ii. Equation of the decision boundary:

(b) There are 2 classes (+) and (−)

The decision boundary will lie b/w the support vectors.

As we can see from the graph,

(2,3) blue points will behave as the support vectors

(5,6), Red point will " " " " ".

→ Decision boundary will lie b/w these points.

Mid point of these 2 points.

(2,3) & (5,6)

$$= \left(\frac{2+5}{2}, \frac{3+6}{2}\right)$$

$$= (3.5, 4.5)$$ (Point on the decision boundary)

$$\text{slope} = -\left(\frac{6-3}{5-2}\right)^{-1}$$

$$= \frac{\cancel{3}}{\cancel{3}} \, 1$$

$$= -1$$

$$(y - y_1) = m(x - x_1)$$

$$y - 4.5 = -1(x - 3.5)$$

$$y - 8 = x$$

$\boxed{y = x + 8}$ eq^n of the decision boundary.

iii.

(c) Since we calculated the eqn in the previous part using (2,3) & (5,6) as the support vectors. These 2 points will represent the support vectors from their respective classes.

iv.

(d). Margin of the decision boundary.
From theory we know that the distance will be the same b/w the decision boundary & the support vector.

$\therefore$ Margin $=$ 2D.

$$D = \frac{|ax_1 + by_1 + c|}{\sqrt{a^2+b^2}}$$

$(x_1, y_1) = (2,3)$ eqn: $y = x+8, -x-y+8=0$

$$= \frac{|-1(2) - 1(3) + 8|}{\sqrt{(-1)^2 + (-1)^2}}$$

$$= \frac{-5+8}{\sqrt{2}} = 3/\sqrt{2} = 2.121$$

$\boxed{\text{Margin} = 4.242}$

v. If we remove any one of the support vectors from this equation then the decision boundary will change. This will happen because the current decision boundary is dependent on the 2,3 and 5,6 support vectors which are only one from each class. If there had been more than one support vector for both the classes then the decision boundary wouldn't have changed. Removing any one support vector will enforce another point of it's class to act as the support vector which will change the equation of the decision boundary.

Section C

1. Data Preparation

a. Downloaded the train_32x23.mat from the link provided and read the data into the variable using a library.

Downloaded and read the .mat file into a variable

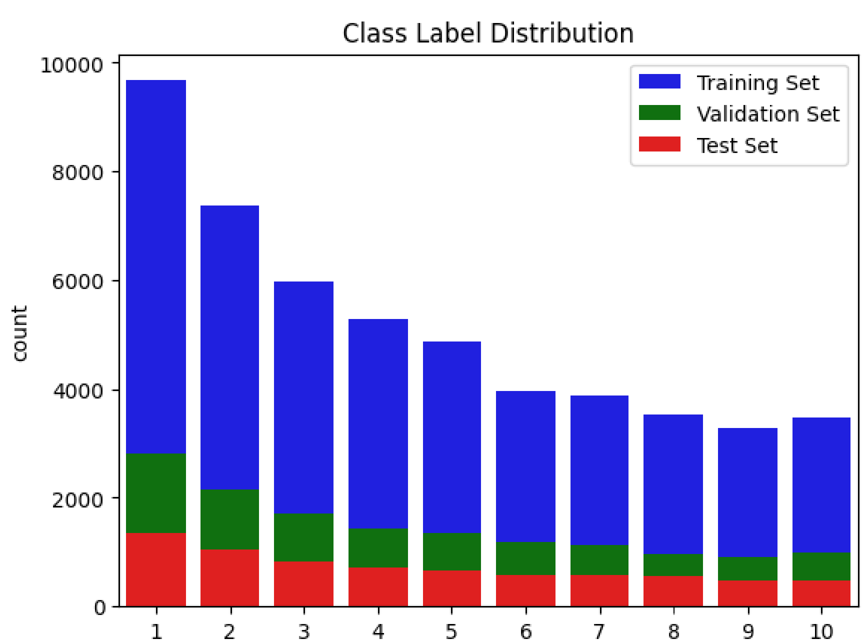
```python
import scipy.io
data = scipy.io.loadmat('/Volumes/A/IIIT/SEM 7/ML/Assignment 3/train_32x32.mat')
```

b. Splitted the dataset into 20% validation and 10% testing

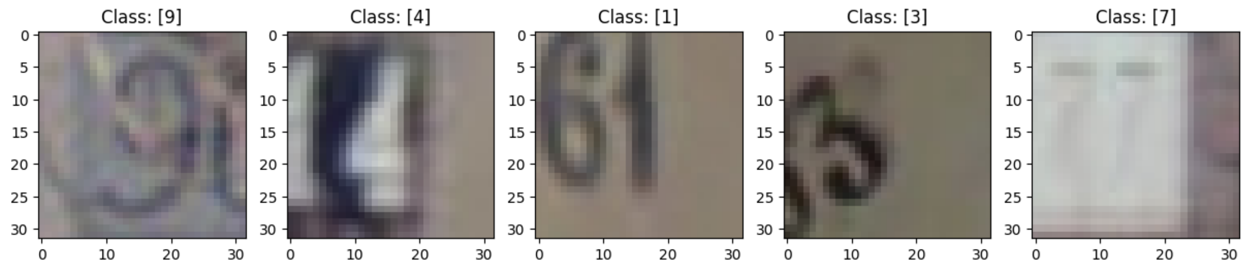
```python
# Split the data into training (70%) and a temporary dataset (30%)
x_train, x_temp, y_train, y_temp = train_test_split(x, y, test_size=0.3, random_state=42)

#Split temporary dataset into validation (20%) and testing (10%)
x_validation, x_test, y_validation, y_test = train_test_split(x_temp, y_temp, test_size=1/3, random_state=42)
```

c. Visualizing the distribution of classes in the three splitted sets



d. Visualization of 5 unique models from the training dataset



2. Model Training and Activation Functions

a. Neural Network with 2 hidden layers (excluding input and final layers)

A neural network with 2 hidden layers

```python
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Create an instance of MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=100)

# Train the model on the training data
mlp.fit(x_train, y_train)
# Test the model on the testing data
y_pred = mlp.predict(x_test)

# Print the accuracy score
print("Accuracy:", accuracy_score(y_test, y_pred))
```

[10]

```
/Users/ojasvasingh/Library/Python/3.9/lib/python/site-packages/sklearn/neural_network
  y = column_or_1d(y, warn=True)
Accuracy: 0.1865956865956866
```

b. GridSearch to find optimal hyperparameters(batch-size and hidden layer size)

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
Best hyperparameters: {'batch_size': 64, 'hidden_layer_sizes': (256, 128)}
```

c. Model training using various activation functions

```python
# Train the model and record loss at epoch for m1
validation_loss1 = []
training_loss1 = []
m1 = MLPClassifier(hidden_layer_sizes=(256,128), activation='logistic', batch_size=64,max_iter=15)

for epoch in range(m1.max_iter):
    m1.partial_fit(x_train, y_train, classes=np.unique(y_train))

    # Calculate training loss using training_scores_
    training_loss = m1.loss_
    training_loss1.append(training_loss)
    # Calculate validation loss using validation_scores_
    validation_loss = 1.0 - m1.score(x_validation, y_validation)
    validation_loss1.append(validation_loss)
```

```python
# Train the model and record loss at each epoch for m2
validation_loss2 = []
training_loss2 = []
m2 = MLPClassifier(hidden_layer_sizes=(256,128), activation='relu', batch_size=64,max_iter=15)
for epoch in range(m2.max_iter):
    m2.partial_fit(x_train, y_train, classes=np.unique(y_train))

    # Calculate training loss using training_scores_
    training_loss = m2.loss_
    training_loss2.append(training_loss)
    # Calculate validation loss using validation_scores_
    validation_loss = 1.0 - m2.score(x_validation, y_validation)
    validation_loss2.append(validation_loss)
```

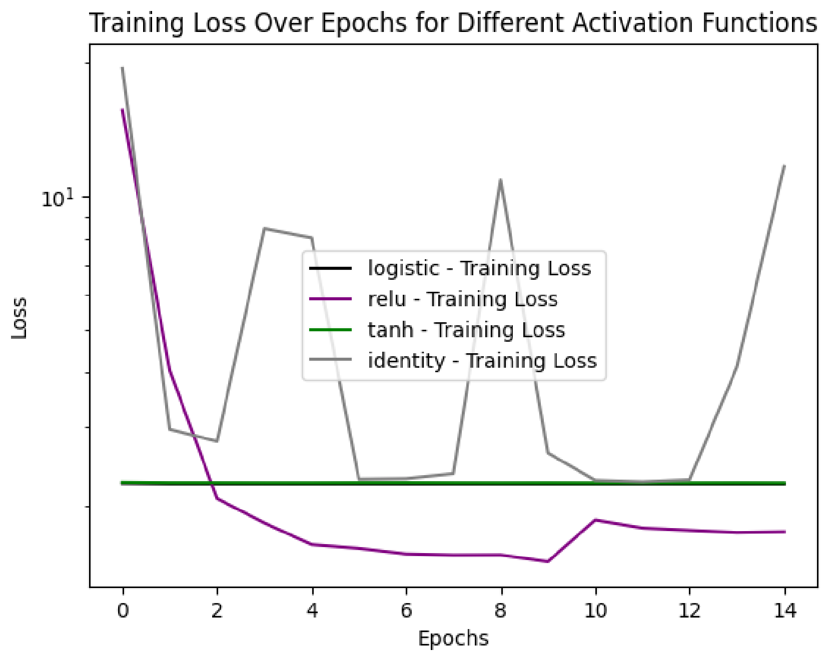
```python
# Train the model and record loss at each epoch for m3
validation_loss3 = []
training_loss3 = []
m3 = MLPClassifier(hidden_layer_sizes=(256,128), activation='tanh', batch_size=64,max_iter=15)
for epoch in range(m3.max_iter):
    m3.partial_fit(x_train, y_train, classes=np.unique(y_train))

    # Calculate training loss using training_scores_
    training_loss = m3.loss_
    training_loss3.append(training_lo
    # Calculate validation loss using
(variable) x_validation: Any
    validation_loss = 1.0 - m3.score(x_validation, y_validation)
    validation_loss3.append(validation_loss)
```

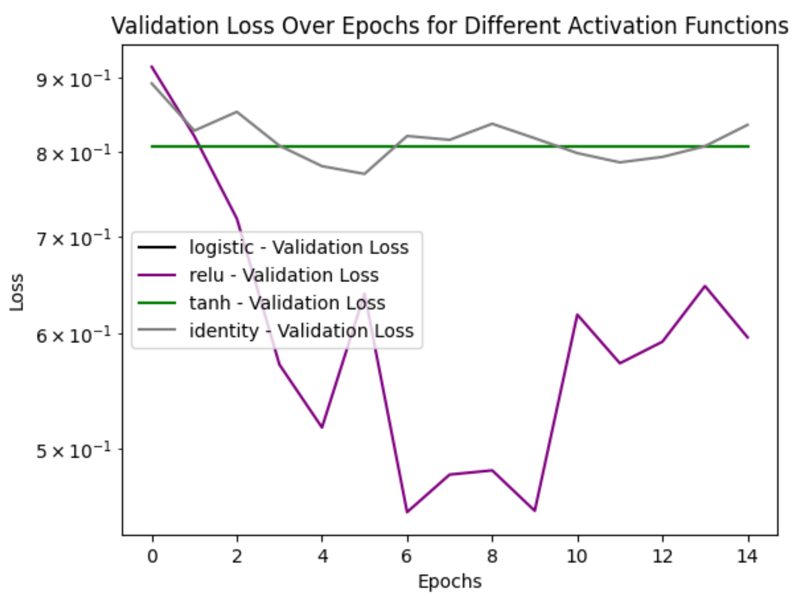
```python
# Train the model and record loss at each epoch for m4
validation_loss4 = []
training_loss4 = []
m4 = MLPClassifier(hidden_layer_sizes=(256,128), activation='identity', batch_size=64,max_iter=15)
for epoch in range(m4.max_iter):
    m4.partial_fit(x_train, y_train, classes=np.unique(y_train))

    # Calculate training loss using training_scores_
    training_loss = m4.loss_
    training_loss4.append(training_loss)
    # Calculate validation loss using validation_scores_
    validation_loss = 1.0 - m4.score(x_validation, y_validation)
    validation_loss4.append(validation_loss)
```

d. Training Loss vs Epochs



Validation Loss vs Epochs



e. **From both the curves** we see that,

For logistic and tanh function the curve is a straight line which shows that the models are not able to capture the complexity of the dataset because of their distribution restrictions. Logistic function which is the sigmoid function has values between 0 and 1, tanh has values between -1 and 1 which is part of the reason that the models are not able to capture the true patterns effectively.

For identity function the curve doesn't converge and shows that the model is not learning effectively, it is mostly used for linear regression tasks, other reasons could be different hyperparameters for the model.

For the Relu function the curve learns over epochs, the initial high values show sensitivity to the initialization and over epochs it converges and lowers the loss. It is suitable for neural networks.

f. Best accuracy achieved on the test using the best hyper parameters.

Training model with RELU activation function and other best hyper parameters

```
model_final = MLPClassifier(hidden_layer_sizes=(256, 128), activation='relu', batch_size=64, max_iter=300)
model_final.fit(x_train, y_train)
y_pred = model_final.predict(x_test)
```

```
/Users/ojasvasingh/Library/Python/3.9/lib/python/site-packages/sklearn/neural_network/_multilayer_perceptron.py:1102
  y = column_or_1d(y, warn=True)
```

+ Code + Markdown

Best accuracy achieved on the test set

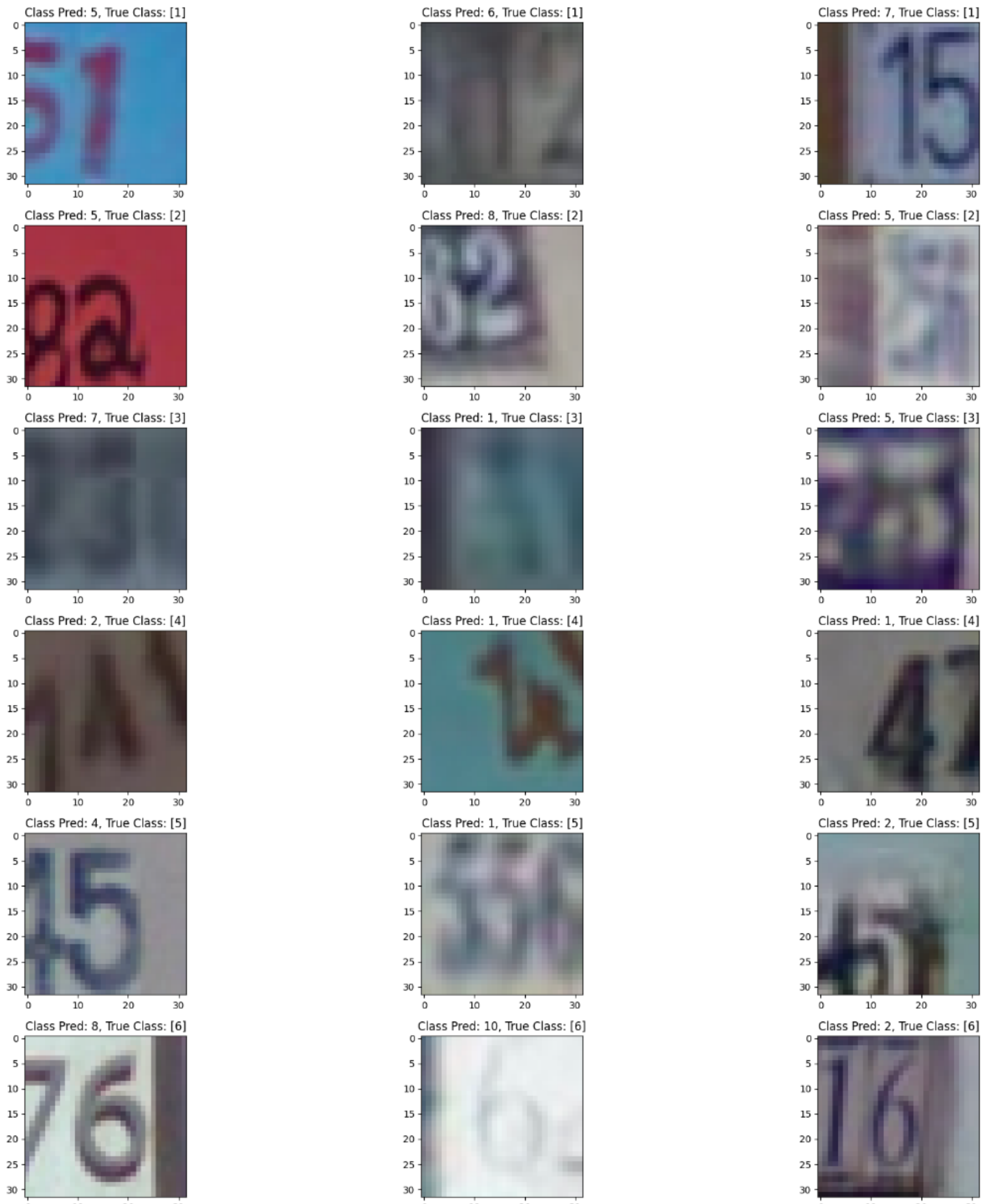
```
# Evaluate the accuracy of the model on the test set
test_accuracy = model_final.score(x_test, y_test)
val_accuracy = model_final.score(x_validation, y_validation)
# Print the best accuracy achieved on the test set
print("Best accuracy on validation set:", val_accuracy)
print("Best accuracy on test set:", test_accuracy)
```

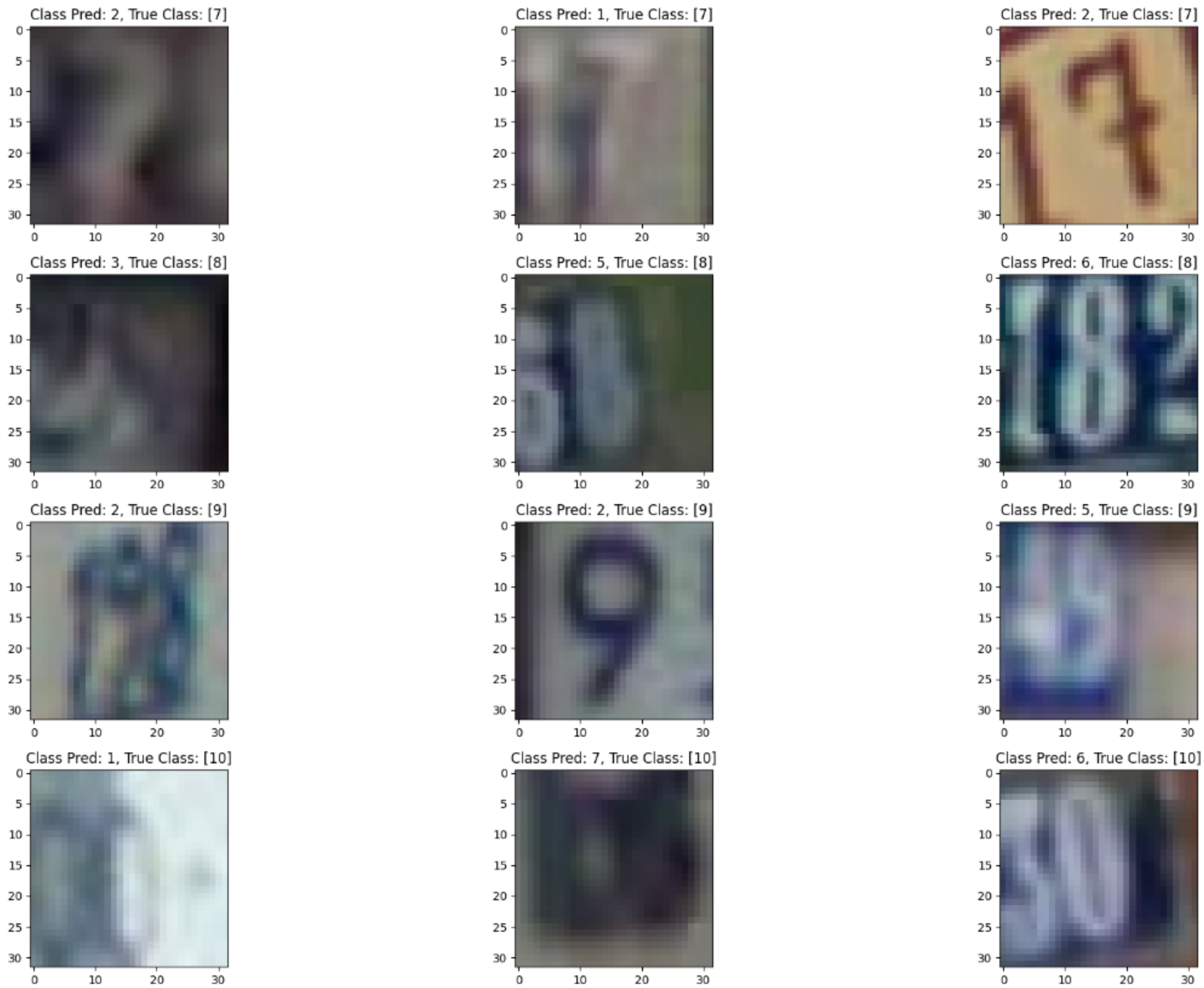
```
Best accuracy on validation set: 0.5655200655200655
Best accuracy on test set: 0.5540540540540541
```

Yes, my model is able to reach a decent accuracy. After using Grid Search I was able to reach the optimal number of neurons and batch size for the network. After looking at the training loss and validation loss curve vs the epochs the best activation function was Relu. For the max_iter part, I tried and tested over some different number of iterations and got the best results at 300 iterations.

3. Visualization of Incorrect Predictions

a. Visualization of 3 misclassified images along with their predicted classes





b. Reasons for model misclassifications:

From the images plotted above for the predicted and actual classes of images we can conclude that

1. Some images have more than one number visible and the other number is being predicted.
2. Some images are blurred and the model is not able to comprehend the actual class and therefore predicts some other class.
3. The Shapes of some numbers resemble each other and the model cannot differentiate when the image is not very clear.
4. Some numbers are half visible in the images, leading to misinterpretation.