Introduction

The aim of this project was building an algorithm that can identify the hand-written numbers. In this project, our team used the Neural Network (NN). The NN showed the high performance with sufficient techniques. The NN's ability to create various boundaries within multi-layer resulted high performance, while other techniques (kernel SVM, KNN) are only able to create linear or overfitting boundaries. In this report, our NN algorithm and other data preprocessing techniques are described.

Redistribution of data

In order to remedy the effect of imbalanced training set, we designed an algorithm that equally balance each data according to their labels. For each label, we pick EpL(Elements per Label) amount of data corresponding to that label stochastically. This also alleviates the effect of under-fitting by providing the neural network more amount of training data when chosen large enough EpL.

algorithm balanceData (T, L, EpL)
-Input-
T: Training data
L: Label for training data
EpL: Elements per Label
-Output-
B: Balanced set of data with size EpL * number of labels. There are EpL number of data that has same label.

1. Initialize B with size EpL * number of labels.
2. for each label in L as iL
3. repeat EpL times
4. select a random element R from T that has label iL
5. append the index of R from T into the array B.

Dropout

Dropout is a method to relieve overfitting and to improve generalization. In the feed-forwarding part of the neural network, the layer unit is stochastically dropped out to 0.
The dropout reduces computational cost by reducing the number of fitting parameters in one iteration.
Also, to reduce the computational time, our team used matrix multiplication rather than for-loop at every column or row. Moreover, matrix size were pre-allocated to prevent repetitive declaration of matrix variables. These two techniques are suited for the Matlab programming.
algorithm dropout(M, R)
-Input-
M: Neural Network model
R: Dropout rate
-Output-
M: Neural Network model with each layer stochastically dropped out

```
1. for each layer in M.layer except the final layer
2.    for each unit in layer
3.        Make the unit value 0 with rate R
```

Neural Network

For algorithm to train SVHN dataset, we used neural network. The neural network consists of feedforwarding, back propagation, and weight update. For the hyper-parameters we used for this neural network,
Learning rate: 0.005
Number of hidden layers: 5
Number of units in hidden layers: [128 256 384 256 128]
Batch size: 1
Elements per label: 1200
Training-Test period: 200 iterations / test
Stop Early policy: 50

Also, to reduce the computational time, our team used matrix multiplication rather than for-loop at every column or row. Moreover, matrix size were pre-allocated to prevent repetitive declaration of matrix variables. These two techniques are suited for the Matlab programming. We also implemented stopping the training early. For every Training-Test period iteration, we check for accuracy in the validation set. We save the model if it scored the max accuracy. When the max accuracy doesn't change for the next Stop Early Policy tests, stop the training immediately and outputs the saved model. This allows reducing of the computational time as well as preventing the over-fitting problem.
In order to improve performance of the neural network, we implemented decaying learning rate which decreases the learning rate over course of time.

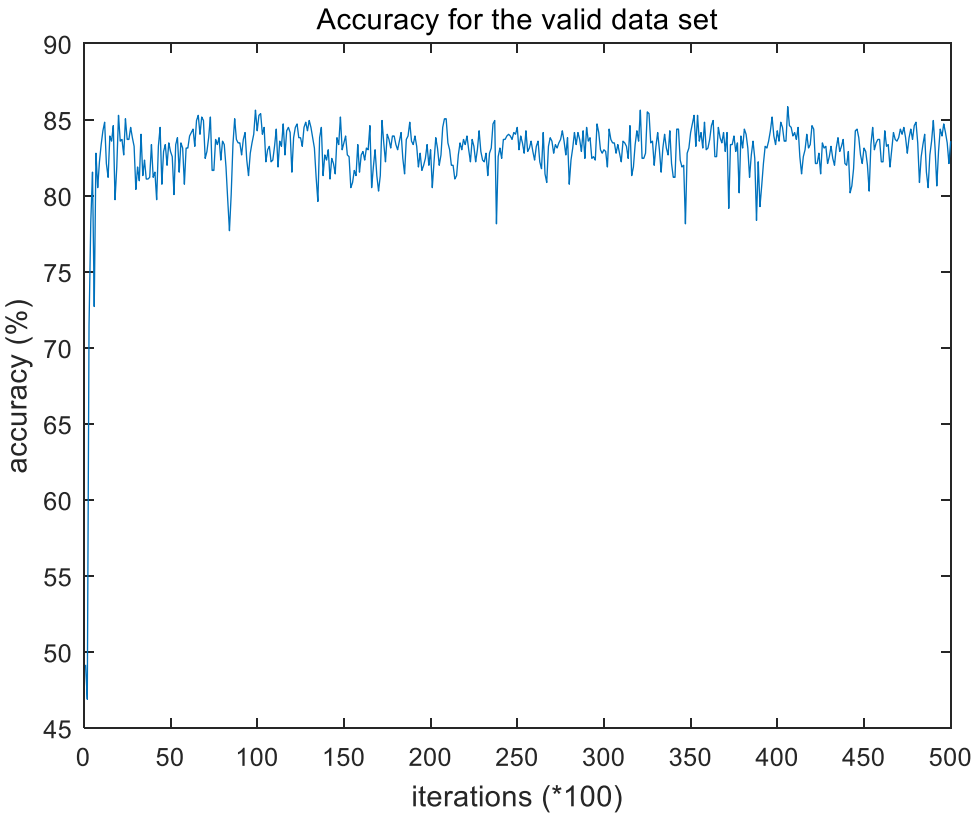
```
algorithm trainNeuralNetwork(T, L, V, Vl)
  -Input-
    T: Training data
    L: Label for training data
    V: Validation set
    Vl: Validation set label
  -Output-
    M: Trained Neural Network model

// Initialize the model
1. Initialize M.layer, M.weight M.bias, M.epoch, M.iterationNum
2.    repeat i = 1:size of T
3.        M = Feedforward(M, T(i))
4.        updateVal = Backpropagate(M, L(i))
5.        M = WeightUpdate(M, updateVal - i)
6.        if stopEarly(M, V, Vl) is True
7.            break
8.        M = M.maxModel
```

```
algorithm stopEarly(M, V, Vl)
  -Input-
    M: Neural Network model
    V: Validation set
    Vl: Validation set label
  -Output-
    True/False: Whether to stop the training or not
1. Val_acc = Validation(V,Vl)
2. if Val_acc > M.max_acc
3.    M.maxModel = M
4.    M.iterationNum = 0
5.    return False
6. else
7.    if M.iterationNum >= Stop Early Policy
8.       return True
9.    M.iterationNum = M.iterationNum + 1

algorithm Feedforward(M, inputData)
  -Input-
    M: Neural Network model
    inputData: sample of data
  -Output-
    M: Neural Network model with updated layer values

1. for each layer from the second layer in M.layer
2.    update the layer unit value by M.layer(layer) <- M.layer(layer -1) * M.weight(layer)
+ M.bias(layer)
// For this project, we used tanh function for activation function.
3.    activate M.layer with activation function

algorithm Backpropagate(M, labelData)
  -Input-
    M: Neural Network model feedforwarded
    labelData: label for feedforwarded
  -Output-
    updateVal: layer gradient

1. Initialize updateVal with size of M.weight
2. dLdOut = M.layer(Last layer) - labelData
3. for index_layer 1 through (number of layers -1)
4.      updateVal = dLdOut * derivative of activation function
5.      for weight_iteration (number of layers - index_layer + 2) through (number of layers)
6.          updateVal = M.weight(weight_iteration)' * updateVal
7.          updateVal = updateVal .* (1 - M.weight(weight_iteration-1).^2)
8.      updateVal = updateVal * M.layer(number of layers - index_layer)'

algorithm WeightUpdate(M, updateVal, R)
```

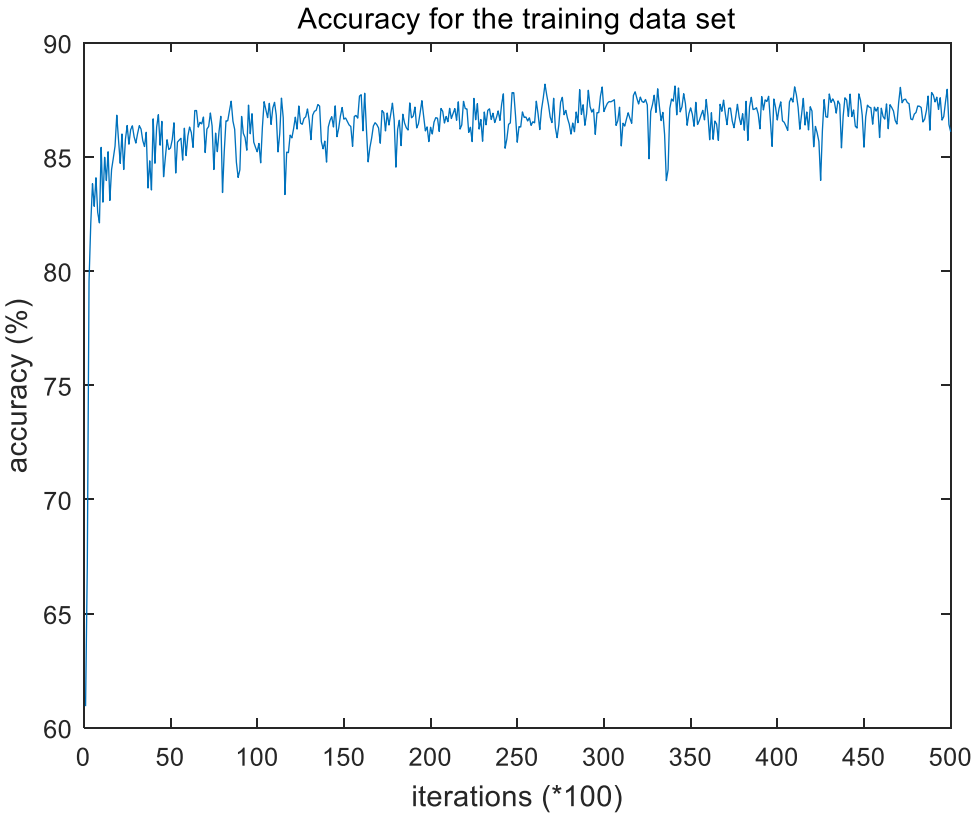
-Input-
M: Neural Network model
updateVal: layer gradient
R: Learning rate
-Output-
M: Updated Neural Network

1. for each layer from second layer in M.layer
2. M.weight(layer) = M.weight(layer) - updateVal(layer) * R

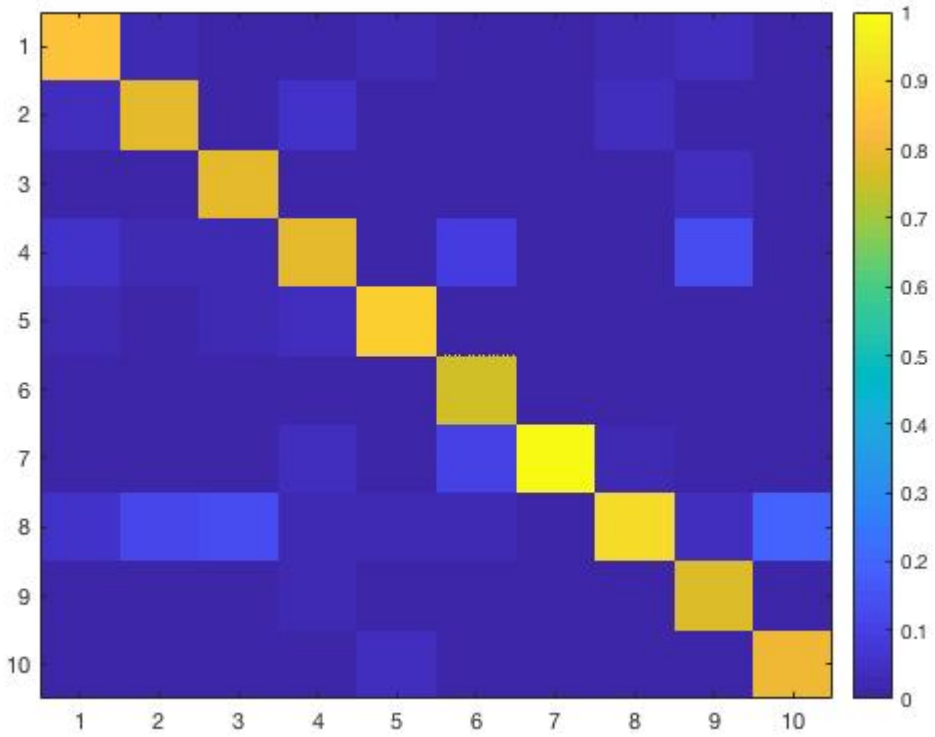
Results



<figure 1> Validation loss



<figure 2> Training data set



<figure 3> confusion matrix. Columns mean real tag in the valid data and rows mean classified tag.