574 Project 3 Report

Jingyi Zhao (50245749)
Zheng Kai (50247576)
Can Yao (50243637)

Abstract

The goal of this project is to use machine learning to solve a classification problem by recognizing a 28 * 28 grayscale handwritten digital image and identify it as a digit among 0,1,2,3, …, 9. First we extract feature values and labels from the MNIST data, and then we use the softmax regression, single hidden layer neural network and convolutional neural network package to train on the MNIST digit images and tune hyperparameters. In these process, we use Mini-batch stochastic gradient descent to compute weight. Then we use the testing data in MNIST and USPS data to test our model, and the accuracy of USPS data for logistic regression is 31.7%, for SNN is 51.4%, and CNN is 62.6%. From the result, we strongly recommend the CNN as a training model. The no free lunch theorem is true at every case.

Introduction

In this project, we train three different classification models on MNIST datasets. For MNIST dataset, we divide the original 60,000 training datas into 50,000 training datas as training set, and another 10,000 datas as validation set. When we finished training the model on the training set, first we test the model on MNIST test set, then we use the 2,000 samples in the test set from USPS to test the accuracy of the classification model. So from the result we can know the best accurate method.

We evaluate our solution on a separate validation dataset using classiciation error rate:

$$E = \frac{N_{\text{wrong}}}{N_{\text{V}}}$$

where N_{wrong} is the number of misclassification and N_{V} is the size of the validation dataset. Under the 1-of-K coding scheme, the input will be classified as

$$C = \arg\max_i y_i.$$

In the task 1, we choose the softmax regression to train on the MNIST digit images. The softmax function is defined as following:

$$P(y = j \mid z^{(i)}) = \phi_{softmax}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^{k} e^{z_k^{(i)}}},$$

where we define the net input z as

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \sum_{l=0}^{m} w_l x_l = \mathbf{w}^T \mathbf{x}.$$

And then in order to train our logistic model via gradient descent, we need to define a cost function as following:

$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=0}^{n} H(T_i, O_i)$$

which is the average of all cross-entropies over our n training samples. The cross-entropy function is defined as

$$H(T_i, O_i) = -\sum_m T_i \cdot log(O_i).$$

Finally, we use the gradient descent to iteratively update the weight matrix until we reach a specified number of epochs (passes over the training set) or reach the desired cost threshold, the formula is following:

$$\nabla \mathbf{w}_j J(\mathbf{W}) = -\frac{1}{n} \sum_{i=0}^{n} \left[\mathbf{x}^{(i)} \left(T_i - O_i \right) \right]$$

In the task 2, the SNN consists of three layers, feed layer, hidden layer and output layer. The output of first layer can be used as the next level of input, thereby adding a hidden layer, resulting in a single hidden layer neural network. Hidden layer connected to the input and output layer. It is the feature space, the number of hidden units is the dimension of the feature space, or how many features of this group of data. Because we have many neural units, so we can do classification by using the results do OR function and AND function. Obviously, the

data entered is known, and the variables are only those connection weights. The feed-forward propagation is as follows

$$z_j = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)}$$

$$y_k = \frac{\exp\left(a_k\right)}{\sum_j \exp\left(a_j\right)}$$

where z_j are the activation of the hidden layer and h (·) is the activation function for the hidden layer.

We use cross-entropy error function to represent loss during training.

$$E\left(\mathbf{x}\right) = -\sum_{k=1}^{K} t_k \ln y_k$$

The gradient of the error function would be:

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

Having the gradients, we will be able to use stochastic gradient descent to train the neural network.

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} E\left(\mathbf{x}\right)$$

where **w** is all parameters of the neural network.

Experiment Design

- General Procedure

In the project, all our model is based on the MNIST training data. The MNIST is split into a training set (50000), validation set(10000), and the test set(10000). Each 28 x 28 image was converted to a 1 x 784 feature vector. Plus the bias, each input has 784 features. The validation set is used to pick hyper parameter for each model, and the test set is used to evaluate the performance of each model. After the model is trained, we apply the model to a different population (USPS) with 20000 images to see the robustness of each model.

- Validation Process & Parameter Tuning

Validation data is used to minimize overfitting. This set does not adjust the weight of the neural network. If the accuracy based on the training data is increased after the adjustment of the training data, but the accuracy of the verification data is not increased or decreased, it means overfitting occur and early stopping should be introduced. We have successfully picked the best parameters for each model (see section parameter tuning part).

- Softmax Logistic regression (LR design):

The logistic classification is shown below:

$$a[i] = \text{Softmax}(W*x[i] + b)$$

$$y[i] = \exp(a[i])/\text{sum}(a[k])$$

where x[i] is ith training sample and y[i] is the predicted output vector corresponding of sample.We use cross entropy loss function to represent a loss to optimize with; Mini-batch gradient descent is used to optimize this loss. Learning rate and batch size were tuned to optimize the performance.

- Single hidden layer neural network (SNN design):

The SNN layer composes of 3 layers: The first layer(Feed layer) has 784 neurons; The hidden layer has 100 neurons; and the output layer has 10 neurons representing the classification of the output. The first connection (784 x 100) and second connection (100 x 10) weights were initialized randomly. The final output is processed by a sigmoid function and a simple 2-step back-propagation process is implemented. Learning rate , minibatch size, and hidden layer size was then tuned to optimize the performance.

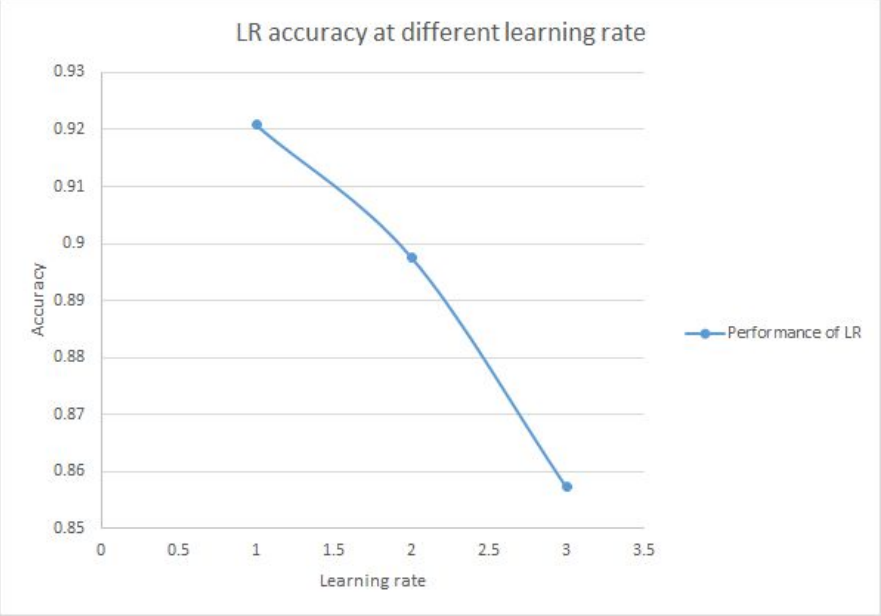
- Convolutional neural network (CNN design)

The model was consulted from tensorflow and slides from TA. The major part is referred from Tensorflow official website. We used 5 layers deep neural network to ensure the loss function converges.

Parameter Tuning:

During gradient descent, all training was run at an large enough epoch to ensure the loss converges. Thus, we are not reporting the number of epoch as parameter here.

- LR
    - The larger the learning rate the worse the performance. Thus we pick 0.01 as learning rate.



    - The model is sensitive to the min-batch size during gradient descent, as shown below, thus we choose the batch size to be 10.

Batch size	Accuracy
10	92%
100	89%
1000	77%

    - The number of epochs chosen in gradient descend are shown and explained in data analysis section.

- SNN
    - The model is sensitive to the learning rate during gradient descent, as shown below, the difference between 0.5 and 1 is small, we prefer to choose smaller learning rate because the larger learning rate may cause over-fitting, thus we choose the size to be 0.5.

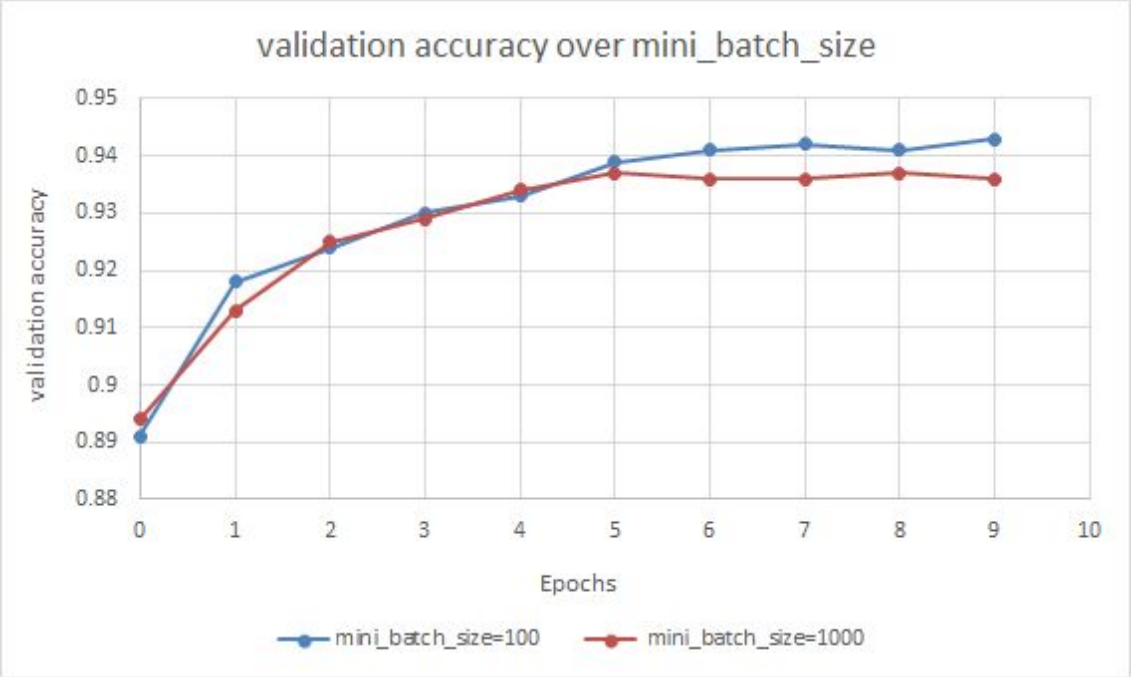
Learning rate	Accuracy
0.01	55.7%
0.1	92.4%
0.5	94.1%
1	94.3%

    - The model is sensitive to the number of hidden units during gradient descent, as shown below, thus we choose the size to be 100

Hidden units	Accuracy
10	88.5%
100	94.1%
500	94.0%
1000	9.9%

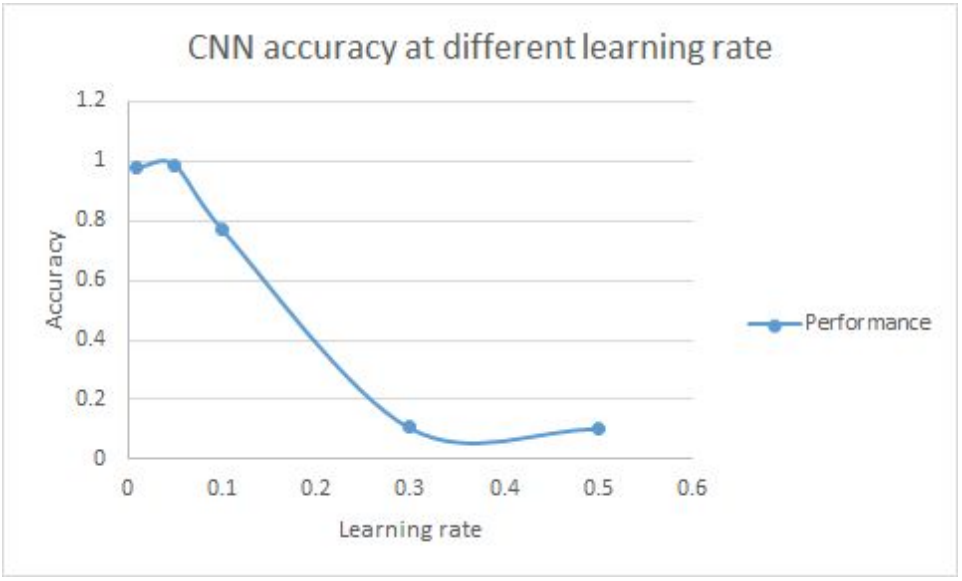
    - The number of epochs chosen in gradient descend are shown and explained in data analysis section.

    - The size of mini-batch does not have much impact on the SNN model, except that the larger the stride the slower the performance is. we pick the mini-batch size as 100.



Batch size	Accuracy
100	94.3%
1000	93.6%

- CNN:
    - Increasing learning rate enable a drastically drop in performance. Thus we pick learning rate as 0.05



- The size of mini-batch does not have much impact on the CNN model, except that the larger the stride the slower the performance is. we pick the mini-batch size as 10.

Batch size	Accuracy
10	98.3%
1000	98.7%
2000	98.0%

- The number of epochs chosen for gradient descent are explained in the data analysis part

Data analysis

- **Softmax Regression model (LR)**

The accuracy of training set and validation set (1-2) is plotted during the course of gradient descent (1-1). Although the loss does seems to converge, both the accuracy converges at a very high value (~92%). Thus we decide it is okey to drop out at epoch 30.

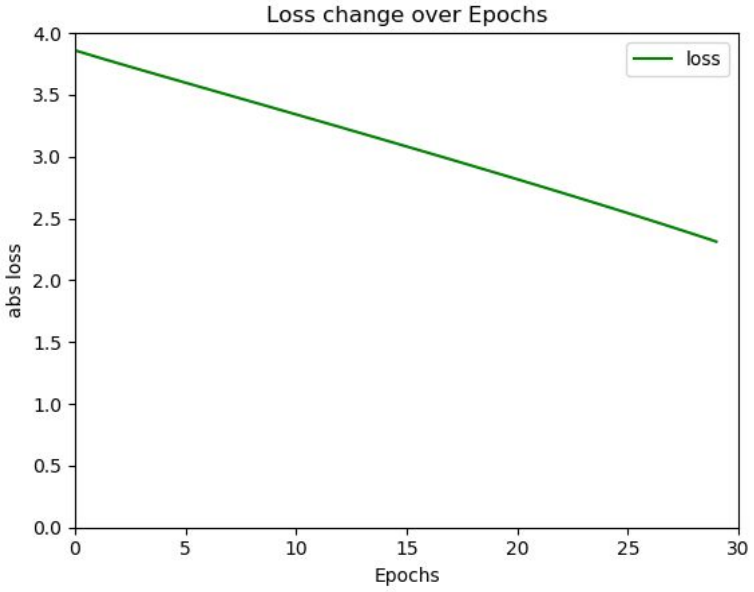


Figure 1-1: Entropy loss over epochs

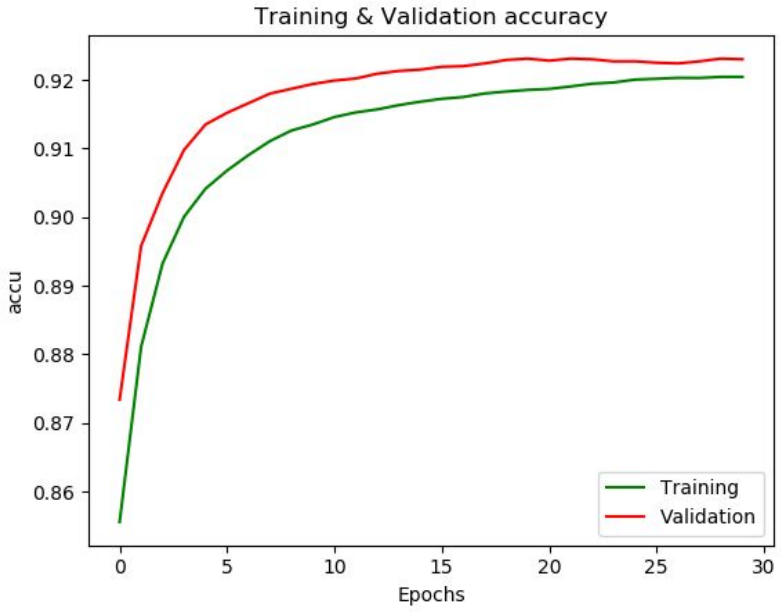


Figure 1-2: Validation & Trainign Accuracy on LR model

The model's performance on MNIST test set is 92%. It's performance on USPS data is shown below in (1-3), with only 36.7 % accuracy.

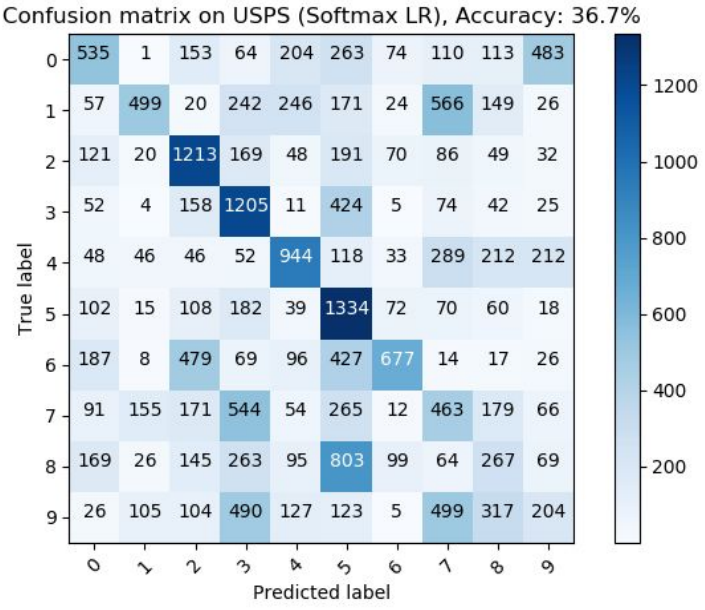


Figure 1-3: Confusion matrix of LR model on USPS data

- **Single hidden Layer Neural Network (SNN):**

The accuracy of training set and validation set (2-2) is plotted during the course of gradient descent (2-1). The Loss converges very fast during the beginning several epochs.

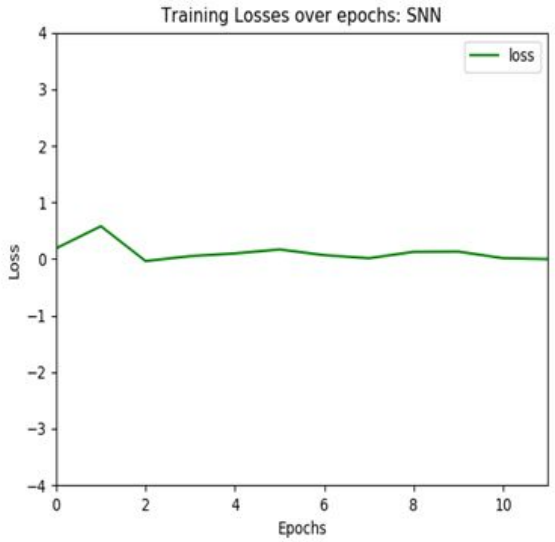


Figure 2-1: training losses over epochs for SNN

The training set, validation set accuracy also converges. at a very high value (97.5%).

Figure 2-2: Validation & Trainign Accuracy on SNN model

SNN model's performance on USPS is shown below in Figure 2-3, with an accuracy of 51.4%.

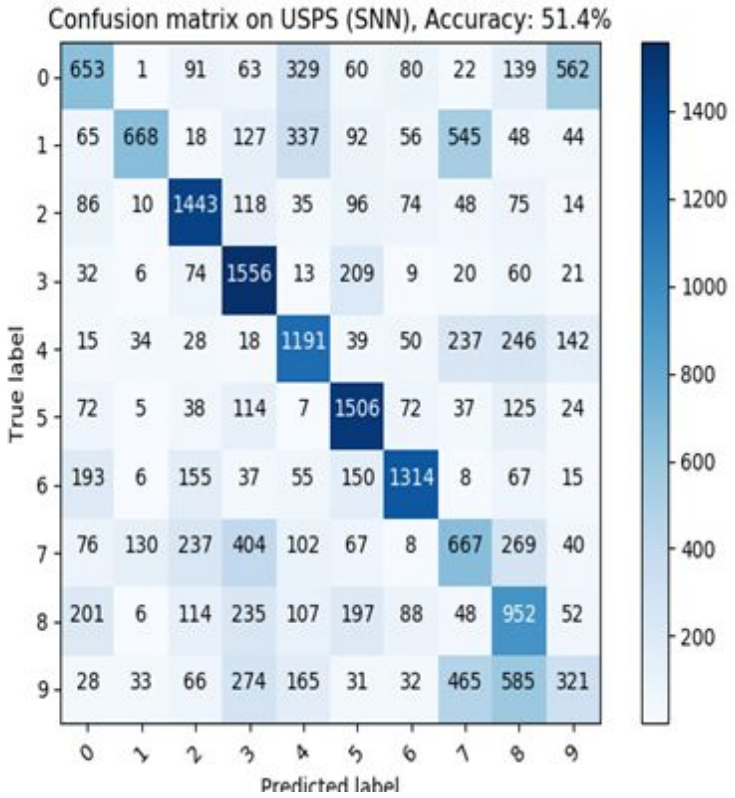


Figure 2-3: Confusion matrix of SNN model on USPS data

- **Convolutional Neural Network (CNN)**

According to Figure 3-1, the model converges > 6000 iterations. Thus we pick 10000 epoch as a safety threshold for gradient descent.

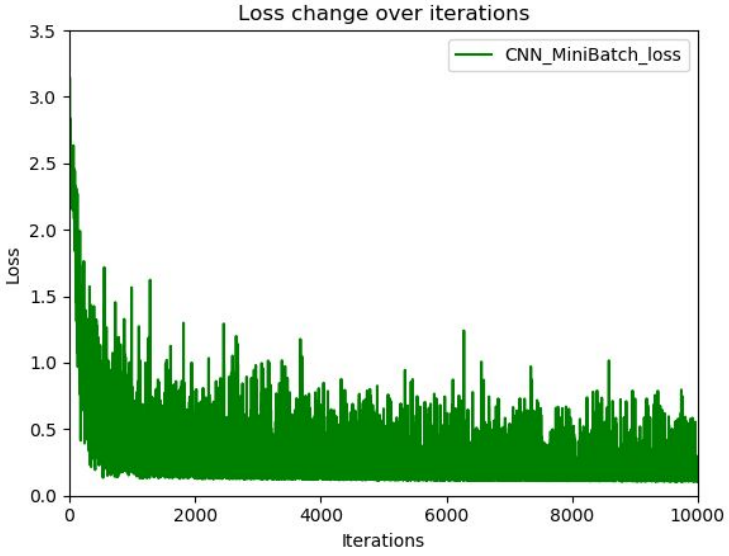


Figure 3-1: loss over epochs for CNN

The performance of CNN on USPS is 62.6%. Which is the highest performance amongst all models.

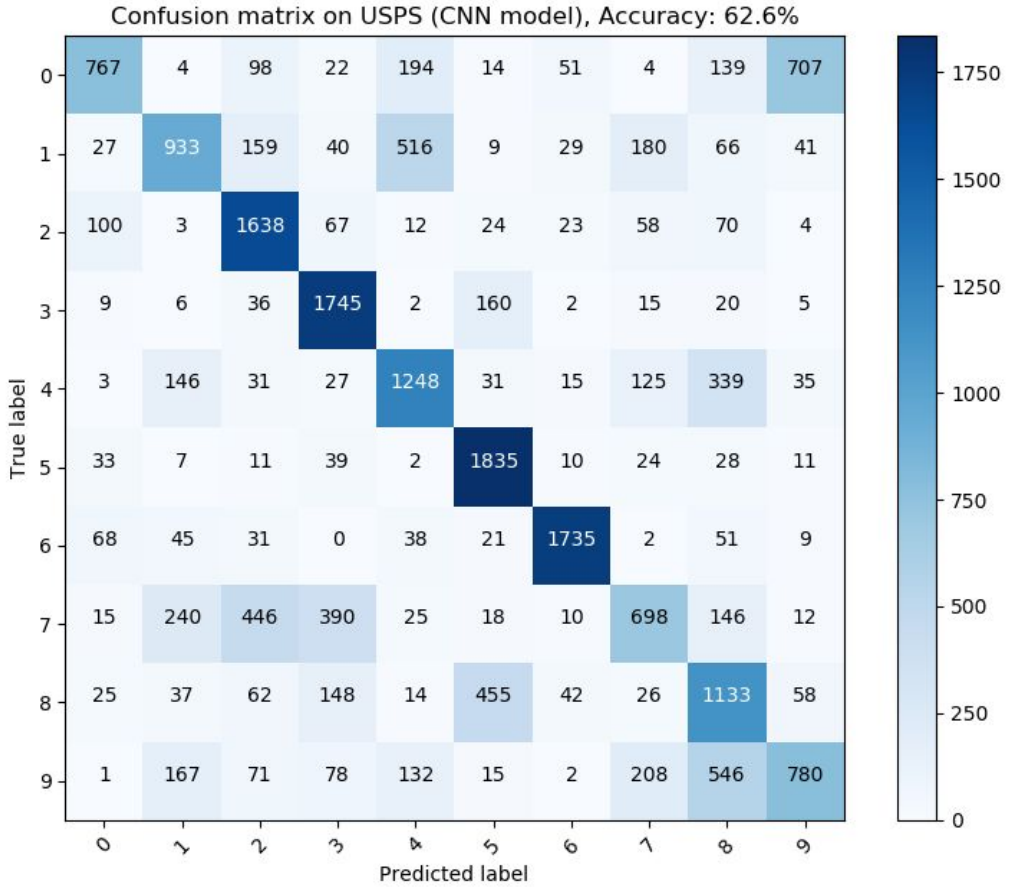


Figure 3-2: Confusion matrix for Confusion matrix of CNN model on USPS data

The overall performance for all models are summarized below in Figure 4.

	Training (MNIST)	Validation (MNIST)	Test (MNIST)	USPS (MNIST)
LR	92%	92%	92%	36%
SNN	93%	97%	97%	51%
CNN	98%	97%	98%	63%

Figure 4： Summary of performance over different models

The No free lunch theorem (NFL) theorem states that there is no model that works best for all problems. It implies the model should be tailored for different populations of data, probably changing the hyper parameters, or increasing epoch ranges. We have confirmed here that the NFL holds that even if a model is great, it will perform badly once used on a different pool of data.

Conclusion

The performance of the three models from low to high is softmax model, SNN model, CNN model. Therefore, we strongly recommend using the CNN model for digital identification.

Bonus part

We also implement the Bayesian logistic regression (BLR) and Variational logistic regression.

BLR's confusion matrix on USPS dataset is shown below in (5): its performance is 39%

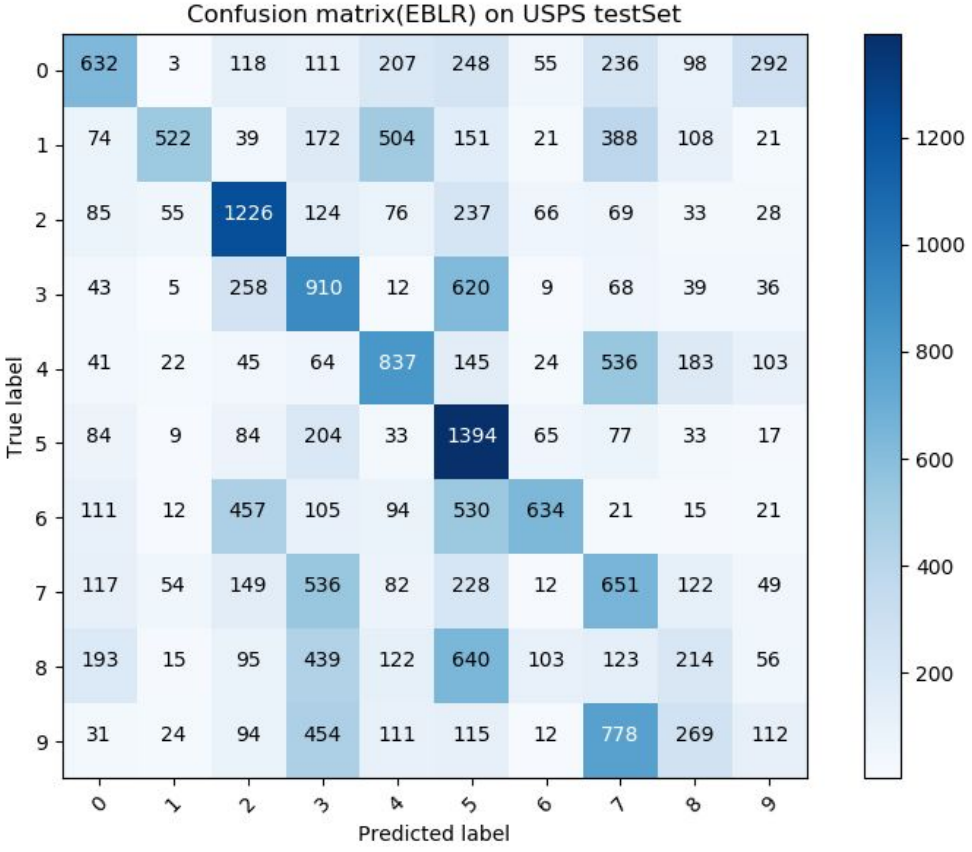


Figure 5: Empirical bayesian logistic regression confusion matrix on USPS data

VBLR's performance on USPS is shown in (6) with a performance of 0.37

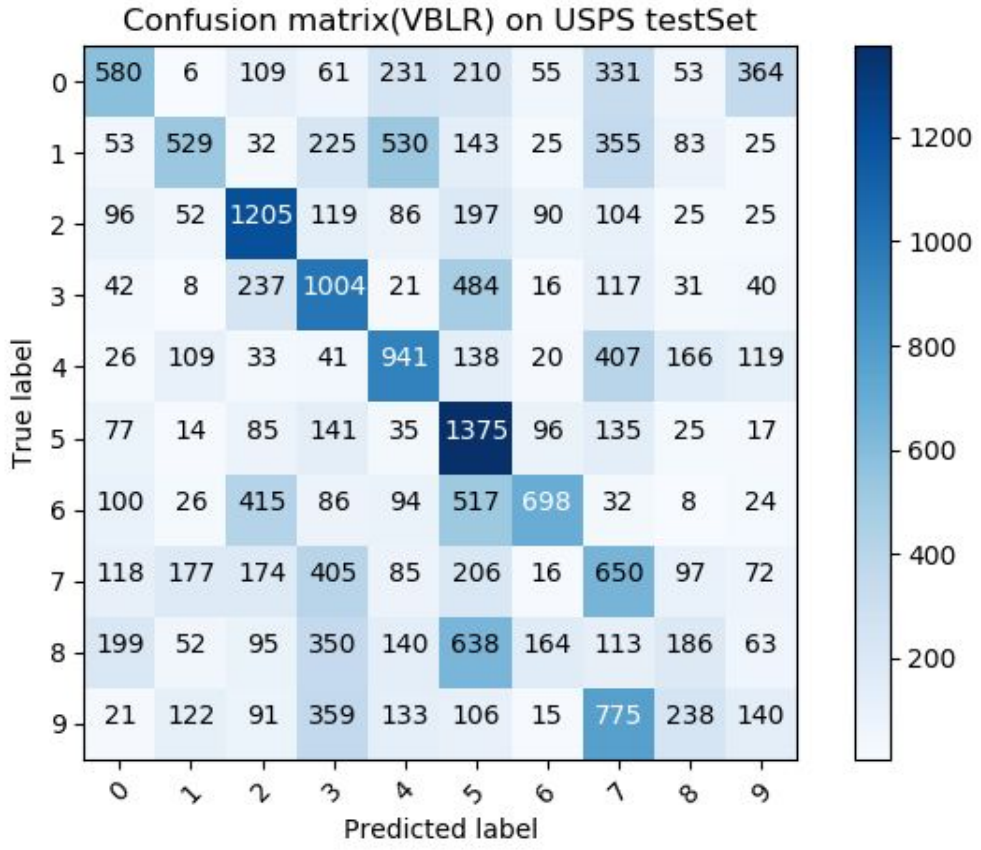


Figure 6: variational bayesian logistic regression confusion matrix on USPS data