Street View House Numbers Recognition

***Abstract*—We used three different machine learning algorithms, including Logistic Regression, Neural Network and Convolutional Neural Network to recognize digits from Street View House Numbers data set[1]. We implemented the algorithms with Tensorflow[2], and compared their performance, as well as pros and cons. Wel also implemented SimpleNet[3] for comparison with our models.**

I. Introduction

Character recognition can be considered as a solved task, especially after the concept of CNN is formalized by LeCun et.al.[1]. Character recognition in the natural scene is a much harder problem due to the high flexibility of input. A typical example is the SVHN (Street View House Numbers) data set. This data set consists of 73257 training images and 26032 test images obtained from Google Street View images(Figure 1), each of which is 32×32 with RGB channels. Based on benchmark.ai[3], the state-of-the-art solution from Cubuk et.al.[2] achieved 1.02% error rate.

In this project we studied the performance of three different algorithms on SVHN dataset, namely, Logistic Regression, Feedforward Neural Network, and Convolutional Neural Network, compared their performance, and discussed their pros and cons. For comparison purpose we also implemented SimpleNet [3].

We conclude that SVHN dataset is not linearly separable, thus linear model such as Logistic Regression will not generate an acceptable performance. Non-linear models such as Feedforward Neural Network is able to achieve a decent result, while a deep Convolutional Neural Network can extract image features more efficiently and generate much better performance.

II. Algorithms

To simplify coding, we implemented an interface inspired by Keras. With this interface, a neural network can be created by simply stacking the layers, without considering the tensor connections between layers.

A. Logistic Regression

Logistic Regression is one of the traditional machine learning algorithms and is the most commonly implemented by machine learning and statistical software packages.

The preprocessed input images are 32×32 with 1 channel, thus the data shape is $32 \times 32 \times 1$. In order to use in logistic regression, an input image is flattened into one single vector with $D = 1024$ dimensions. The output consists of $K = 10$ classes. We apply logistic regression to the input to compute the loss:

$$\mathcal{L}(\mathbf{w}, b) = \theta(\mathbf{w}^\intercal \mathbf{x} + b)$$

, where $\mathbf{w} \in \mathbb{R}^{D \times K}, b \in \mathbb{R}^K$, and $\sigma(z)_i = e^{\sigma_i} / \sum_{k=1}^{K} e^{\theta_k}$ is the softmax function.

L2 regularization is used to avoid overfitting. Thus the loss function can be written as:

$$\mathcal{L}(\mathbf{w}, b) = \theta(\mathbf{w}^\intercal \mathbf{x} + b) + \lambda \|\mathbf{w}\|$$

In this project we choose to implement Logistic Regression with Tensorflow. since logistic regression is equivalent to the `Dense` layer in neural network, we can reuse the neural network code. A `Flatten` layer reshapes the input images to vector then a `Dense` layer will apply the Logistic Regression.

Fig. 1: 32×32 cropped samples from SVHN data set.

B. Neural Network

Neural network is usually referred as "multiple layer perceptron" (or MLP) as it uses one or more "hidden layers". Neural network can simulate non-linear models of any complexity.

For this project, we used a simple network with only one hidden layer with 1024 hidden units. Similar to Logistic Regression, the input images are reshaped to vectors with $D = 1024$. Figure 2 shows the architecture of the neural network we used.

C. Convolutional Neural Network

Convolutional Neural Network (or CNN) is a class of deep neural networks most applied to image recognition tasks. A typical CNN includes the following types of layers:

[1] http://ufldl.stanford.edu/housenumbers/
[2] https://www.tensorflow.org/
[3] https://benchmarks.ai/svhn

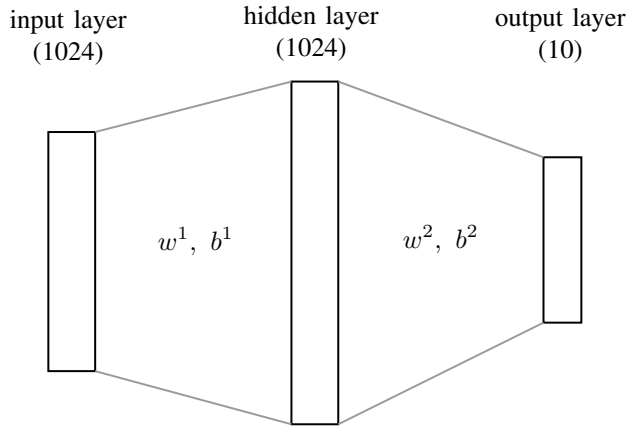

Fig. 2: Neural Network Architecture.

- Convolutiona layer: This type of layers apply convolution operation to the input. Convolution operation produces less parameters than simple fully connected layer.
- Pooling layer: Pooling layers reduce the dimension of the input by combining adjacent outputs. The most common pooling layer is max pooling.
- Fully Connected layer: A fully connected layer is the same layer as in multiple layer perceptron network. This type of layers are usually used after the input data is flattened to produce output for prediction.

For CNN we used an architecture consists of one convolutional layer and one subsampling layer. For comparison, we also implemented Simplet[3] in Tensorflow. Table I shows the architecture of the CNN and SimpleNet.

CNN	SimpleNet
conv2d(32/3x3), relu, bn	conv2d(64/3x3), bn, relu
	[conv2d(32/3x3, bn, relu] x 3
max_pooling(2x2)	max_pooling(2x2)
	[conv2d(32/3x3), bn, relu] x 2
	conv2d(64/3x3), bn, relu
	max_pooling(2x2)
	[conv2d(64/3x3), bn, relu] x 2
	max_pooling(2x2)
	conv2d(128/3x3), bn, relu
	conv2d(256/1x1), bn, relu
	conv2d(64/1x1), bn, relu
	max_pooling(2x2)
	conv2d(64/3x3), bn, relu
	max_pooling(2x2)
flatten	flatten
dense(1024)	
dropout(0.5)	
dense(10)	dense(10)

TABLE I: Convolutional Neural Network Architecture. "bn" is batch normalization layer. CNN uses input moment as batch normalization parameter, and SimpleNet uses batch normalization with decay=0.95. The number in "[layers] x 3" denotes repetition of the layers.

III. Experiments and Analysis

Data Preprocessing. Before applying algorithms, we preprocessed the data set by first converting images from RGB to grayscale, then executing an auto contrast, in order to keep the contrast of images at the same level. The preprocessed images are shown in Figure 3.

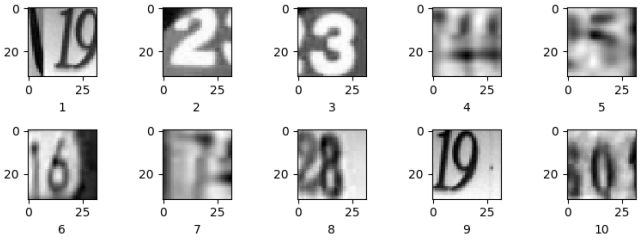

Fig. 3: Preprocessed images of digits 0 - 9.

Dynamic Learning Rate. During the experiment, we realized that choosing an appropriate learning rate is critical for the performance of neural network. However a very small learning rate will increase the training time significantly. Thus we decide to use dynamic learning rate which decreases exponentially. Concretely, the learning rate starts with 0.001, and decreases to $1/10$ every 100 epochs. This dynamic learning rate played an important role in the training, as illustrated later in Figure 8 and 10.

We trained each model with the train data for 400 epochs, and plotted the training/test error every 10 epochs. Figure 4, 6, and 8 show the loss and error for each model respectively. Table II shows the loss and error rate at epoch 400.

Note that for SimpleNet we trained two versions, without dropout and with dropout respectively. This will be disscussed in detail in Section III-D.

model	train loss	test loss	train err	test err
Logistic	2.1448	2.2447	71.06%	74.18%
NN	0.5280	0.6798	7.86%	13.51%
CNN	0.2314	0.5987	3.02%	13.78%
SimpeNet(w/o dropout)	0.0055	0.2982	0.00%	5.50%
SimpeNet(w/ dropout)	0.0546	0.1982	0.13%	3.38%

TABLE II: Loss and error rate at epoch 400.

A. Logistic Regression

Figure 4 shows the loss and error rate of Logistic Regression. At epoch 400 the test error rate is 74.18%. The optimization converges after 200 epoch, and from the error plot we can see that there is no significant raise in test error, indicating that overfitting is not happening.

Figure 5 shows the confusion matrix of Logistic Regression. Most digits are predicted as "1", "2", and "5", while complicated digits such as "3", "4", "7", and "9" have almost no prediction.

Therefore, we conclude that the linear model is overly simple that underfit this dataset.

B. Feedforward Neural Network

Figure 6 shows the loss and error rate of Feedforward Neural Network. This model is able to fit the non-linear nature of the data set and produced a decent performance with 13.51% error rate.

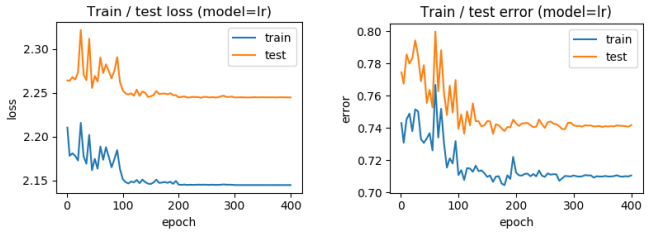

Fig. 4: Loss(left) and error rate(right) in training and test phase, for Logistic Regression model.

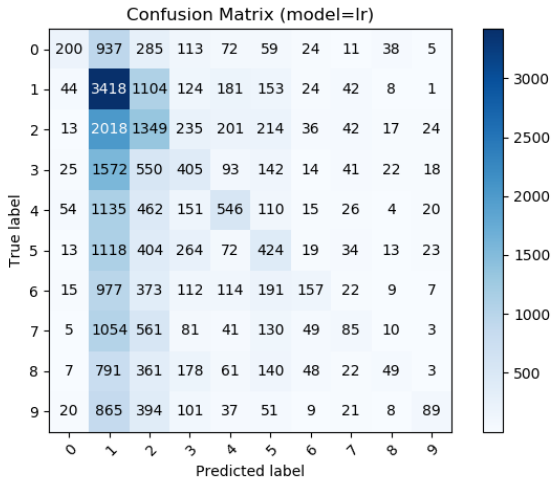

Fig. 5: Confusion matrix for Logistic Regression.

However, neural network need much more free parameters. In our model, the hidden layer has 1024 hidden units which requires 1024×1024 free parameters, and output layer requires 1024×10 parameters, in total approximately 1M parameters are required.

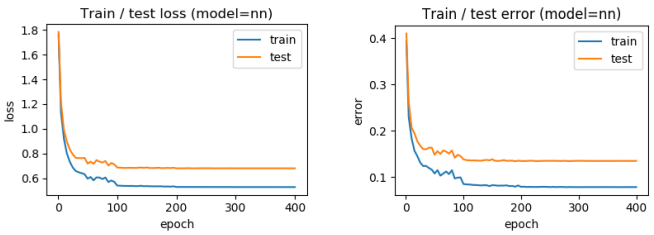

Fig. 6: Loss(left) and error rate(right) in training and test phase, for Feedforward Neural Network model.

C. Convolutional Neural Network

Figure 8 shows the loss and error rate of Convolutional Neural Network. The performance (error rate ??) is similar to Feedforward Neural Network, however it requires much less free parameters. In our CNN model, convolution layer has $3 \times 3 \times 32 = 288$ parameters, the first dense layer has $16 \times 16 \times 128 = 32768$ parameters, and the output layer has $128 * 10 = 1280$ parameters, in total approximately 33k parameters are required. Compared to the 1M parameters in feedforward

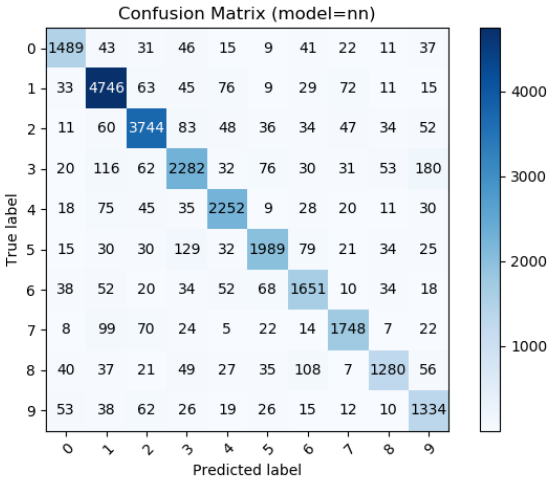

Fig. 7: Confusion matrix for Feedforward Neural Network.

neural network, this is a great improvement and can reduce the model complexity significantly.

During the training process we can also see that dynamic learning rate is important. The first converge occurred around 50 epochs, from where the loss became unstable, which indicates our initial learning rate 0.001 became too large. At epoch 100, the learning rate decreased to 0.0001, and the optimization continued. This shows that our dynamic learning rate approach worked well in this training, enabled our CNN model to achieve a better performance.

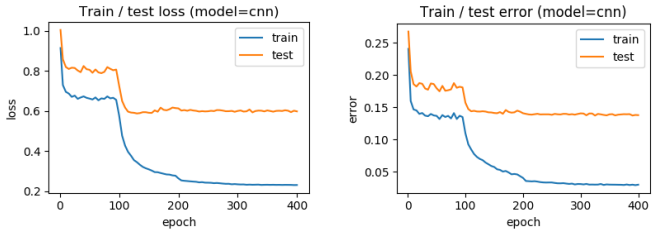

Fig. 8: Loss(left) and error rate(right) in training and test phase, for Convolutional Neural Network model.

D. SimpleNet

Among the four models in this project, SimpleNet(Figure 10) achieved the best performance, with test error rate 5.50% at epoch 400. This can be considered that it is a deep convolutional neural network that consists of enough free parameters to "remember" more patterns in the input images.

Our dynamic learning rate strategy also has a significantly positive effect that it enabled further optimization at epoch 100 and 200.

We also observed that while training error stays stable after 200 epoch, test error starts to rise. This indicates that overfitting happened after epoch 200. To reduce overfitting, we then added a dropout layer with keep probability 0.8 after the ReLU function of each convolution layer. Figure 11 shows that after applying dropout, the train error (0.13%) is higher than

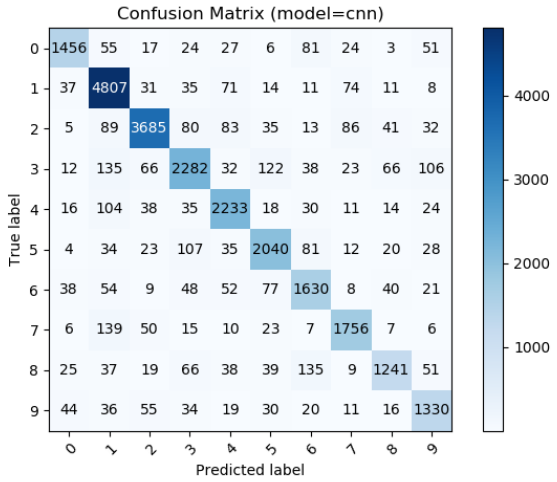

Fig. 9: Confusion matrix for Convolutional Neural Network.

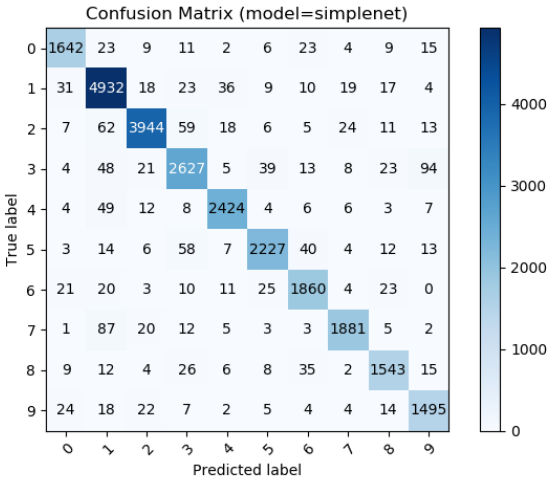

Fig. 12: Confusion matrix for SimpleNet(with dropout).

the model without dropout (approximately 0.00%), however it generalizes better, with test error (3.38%) lower than the model without dropout (5.50%). This shows that dropout can successfully prevent overfitting and eventually increase the prediction accuracy.

The confusion matrix for SimpleNet in Figure 12 shows the SimpleNet can classify the data set correctly.

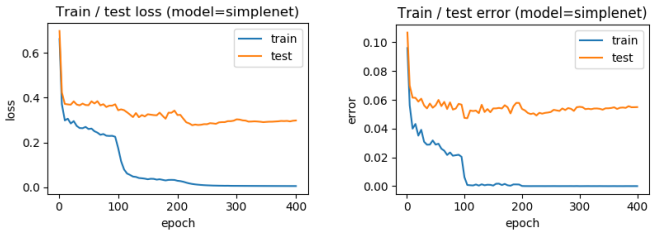

Fig. 10: Loss(left) and error rate(right) in training and test phase, for SimpleNet(without dropout).

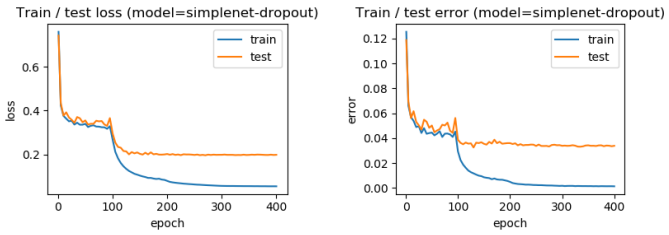

Fig. 11: Loss(left) and error rate(right) in training and test phase, for SimpleNet(with dropout).

IV. Conclusion

In this project we implemented three models: Logistic Regression, Feedforward Neural Network, and Convolutional Neural Network, and tested their performance on SVHN dataset. As a comparison, we also implemented SimpleNet, without dropout and with dropout.

What we have learned from this project:

- Logistic Regression does not perform well on complex data set due to its simplicity;
- Feedforward Neural Network can simulate any non-linear model thus is able to classify non-linear dataset; However the fully connected hidden layers require too many free parameters which causes slow training and huge storage requirement;
- Convolutional Neural Network uses convolution operation to extract features automatically from the input images, thus is able to achieve same or better performance than Feedforward Neural Network with less computational power. It also requires less parameters.
- Deep Convolutional Neural Network can achieve astonishing good performance, since it has more layers and parameters to learn more features from input images. However it is prone to overfitting, thus techniques to prevent overfitting such as batch normalization and dropout are usually necessary.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, november 1998.

[2] Ekin D. Cubuk et.al. *AutoAugment: Learning Augmentation Policies from Data*, arXiv:1805.09501

[3] Seyyed H. Hasanpour et.al. *Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures*, arXiv:1608.06037