CS273 Project: Street View House Numbers

Xiaoxi Wang (35725203) Cheng-Liang Hsieh (92968380) Zongchang Lyu (28287165)

1. Dataset Overview

SVHN is a real-world image data set for developing machine learning and object recognition algorithms which incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images).

The data format we choose is format 2. We print out the shape of training data X which is a 4-D matrix (32, 32, 3, 73257), that means all digits have been resized to a fixed resolution of 32-by-32 pixels. 3 is the height means they are color picture, and 73257 is the number of training data. y is the corresponding label.

Figure 1: Dataset in format

2. Data analysis

2.1 Class distribution

Before any data process, we want to find the class distribution of each data set, in which 1 is the most. We print out the number of each data set, Training set: 73257; Test set: 26032; Extra set: 531131.

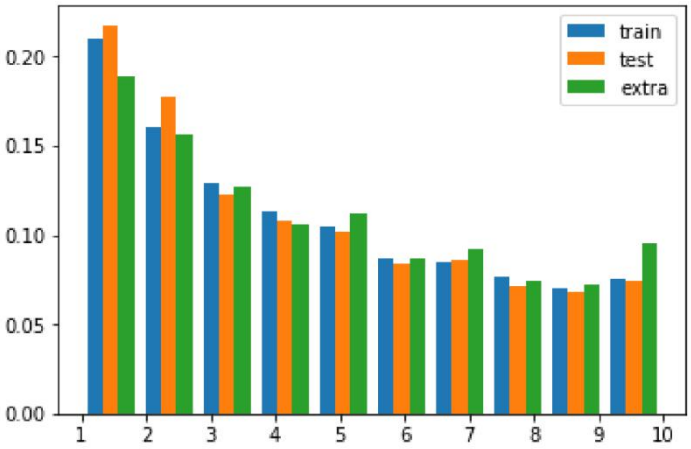


Figure 2: Class distribution

2.2 Data preprocess:

The purpose of data preprocessing is to ensure the quality of data so that it can serve better for subsequent analysis and modeling. We did 3 works in this step:

1. First, we shuffle the training set.
2. Then, we split original training set into training and validation set: 7:3.
3. Finally, gray image and RBG image z-score normalize.

After split the training data, we print out the length of new data set (Training set: 51280, Test set: 21977) and visualize class distribution, which is similar with original dataset.

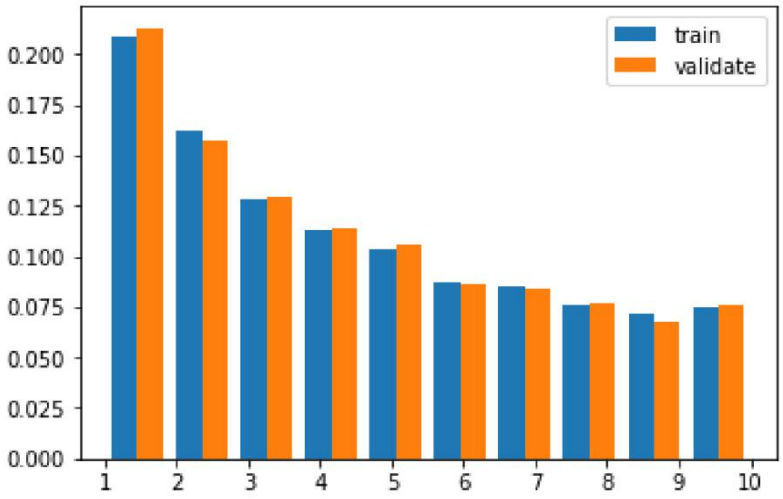
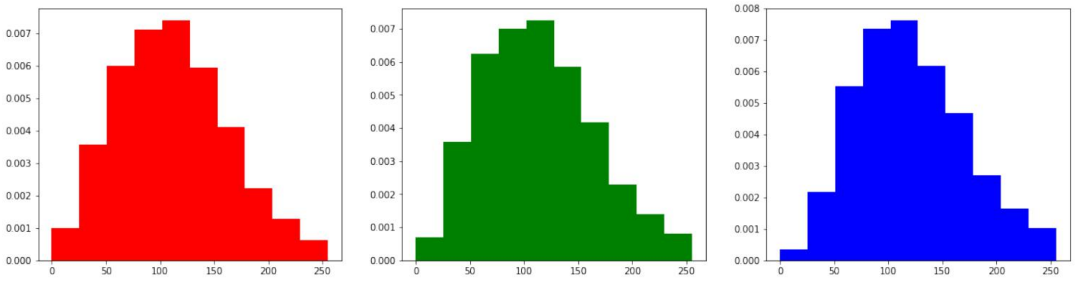


Figure 3: Class distribution after split

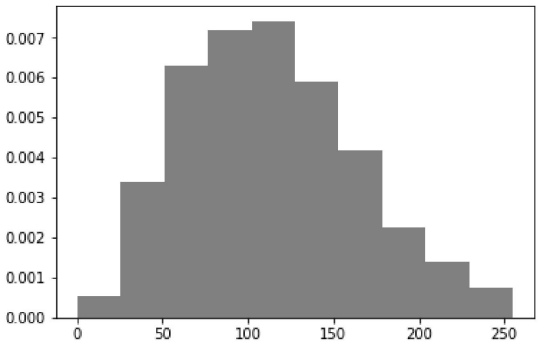
Color images are often built of several stacked color channels, each of them representing value levels of the given channel. In this case, what we get are RGB images which are composed of three independent channels for red, green and blue primary color components.



Mean: 111.48 std:50.52 Mean:113.04 std: 51.52 Mean: 120.49 std: 50.20

Figure 4: data distribution by channel

From the data distribution we get above, we can see the three independent channels are similar of each other, so we implement gray image. We prefer gray image, because it is a one-layer image from 0-255 whereas the RGB have three different layer of image. What is more, the inherent complexity of gray level images is lower than that of color images.



Mean: 113.4298 std: 50.2126

Figure 5: Gray image

Image after gray processing are looks like below:

Figure 6: Samples of image after gray processing

3. Convolution Neuron Network

To set up CNN model, we decided to use convolution layer + Relu( ) activation function + pooling layer twice. For the first time, we use nn.conv2d(input_dim, 6, 5), we set the parameter out_channel = 6, so the height of the image become 6, and the filter is 5*5. We do not give padding value, so the image become 28*28*6, after pooling layer nn.Maxpool2d(2) which the maximum value from every 2*2 filters, then images become 14*14*6. For the second time, we use nn.conv2d(6, 16, 5) and nn.Maxpool2d(2), after that image become 5*5*16. For the full connection layer, we use nn.linear( ) and set parameter ncls = 10 (because we have number range from 0-9, 10 classes). After we trained the model, we begin to talk about influence of some parameters and how to choose them.

3.1 Batch size

To choose batch size is a trade-off. If it is too small, it will no longer strictly decent and be easily overfitting, while if it is too big, computation will be slowly and the convergence becomes harder. We choose batch size from bsize = [64, 128, 256] and evaluate their performance. By plotting the accuracy and loss function in respect of different batch size, we find out batch-size = 128 is optimal.

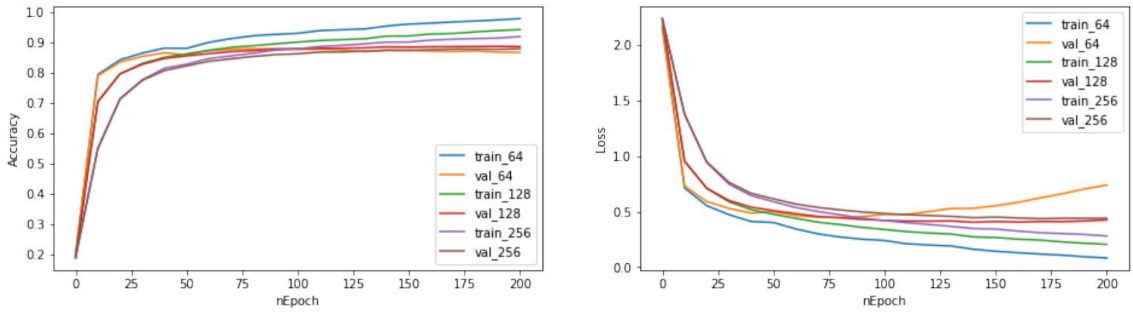


Figure 7: Accuracy and loss function of different batch size

3.2 Learning rate

Learning rate is important for model performance, and how to choose it is also a trade-off. If the learning rate is too small, the loss will drop very slowly, while if learning rate is too big, it would cause the network to converge to local minimal and the loss would start to increase. We choose learning rate from lrates = [0.001, 0.0005, 0.0001] and evaluate their performance. By plotting the accuracy and loss function in respect of different learning rate, we choose lrate = 0.001 and exponential lrate scheduler alpha = 0.95.

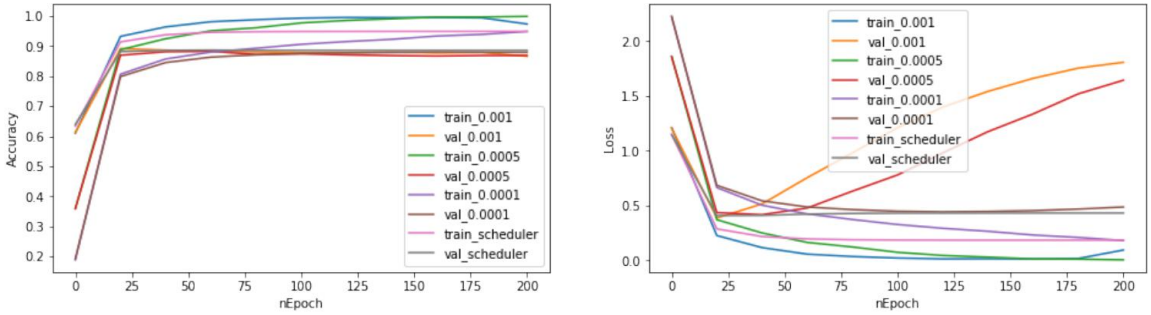


Figure 8: Accuracy and loss function of different learning rate

3.3 Normalization

In order to eliminate problem of comparability between different indicator, data need to perform z-score normalization. After trained the model, we get performance below, and find out normalization in this case did not provide much improvement. We analyze it is because the gray pixel value itself is an approximate normal distribution as shown in Figure.5.

```
Epoch 0: accTr: 0.7917, lossTr: 0.7056, accVa: 0.7894, lossVa: 0.7228
Epoch 20: accTr: 0.9574, lossTr: 0.1532, accVa: 0.8907, lossVa: 0.4301
Epoch 40: accTr: 0.9818, lossTr: 0.0759, accVa: 0.8871, lossVa: 0.5799
Epoch 60: accTr: 0.9889, lossTr: 0.0525, accVa: 0.8837, lossVa: 0.6834
Epoch 80: accTr: 0.9912, lossTr: 0.0447, accVa: 0.8824, lossVa: 0.7317
Epoch 100: accTr: 0.9920, lossTr: 0.0419, accVa: 0.8819, lossVa: 0.7514
Epoch 120: accTr: 0.9922, lossTr: 0.0410, accVa: 0.8821, lossVa: 0.7589
Epoch 140: accTr: 0.9923, lossTr: 0.0406, accVa: 0.8820, lossVa: 0.7616
Epoch 160: accTr: 0.9924, lossTr: 0.0405, accVa: 0.8821, lossVa: 0.7627
Epoch 180: accTr: 0.9924, lossTr: 0.0405, accVa: 0.8821, lossVa: 0.7630
```

Figure 9: Performance after normalization

3.4 Color information

We also want to see the performance of colorful image. We find out the validation accuracy of color image is 0.8820. Color information doesn't bring much

performance improvement. As it has been mentioned in Sec 2.2, that may because RGB channels' distributions are similar and conform to an approximate normal distribution, too.

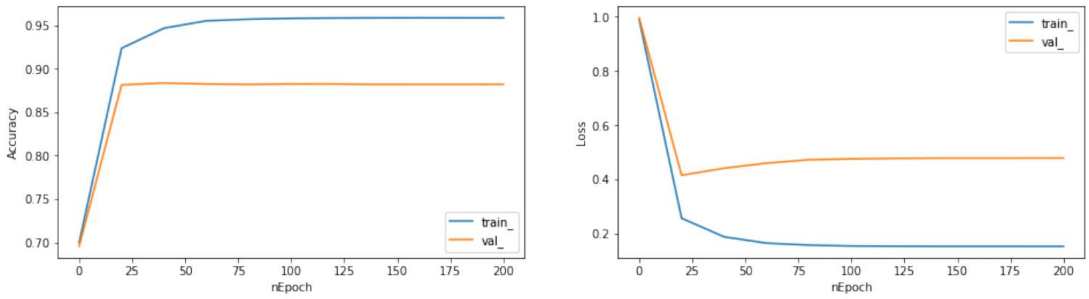


Figure 10: performance of color image

3.5 Dropout

Overfitting is a major problem in neuron network, and we can implement dropout to mitigate overfitting. Dropout means let the activation value of a certain neuron stop work with a probability P. This would make the model more general, because it does not rely too much on some local feature. We add nn.dropout( ) when we setup CNN model in our code. After dropout, the validation accuracy is 0.8831.

```
Epoch 0: accTr: 0.4042, lossTr: 1.7752, accVa: 0.4036, lossVa: 1.7818
Epoch 20: accTr: 0.9259, lossTr: 0.2542, accVa: 0.8838, lossVa: 0.3981
Epoch 40: accTr: 0.9461, lossTr: 0.1914, accVa: 0.8855, lossVa: 0.4272
Epoch 60: accTr: 0.9550, lossTr: 0.1655, accVa: 0.8842, lossVa: 0.4408
Epoch 80: accTr: 0.9576, lossTr: 0.1583, accVa: 0.8834, lossVa: 0.4504
Epoch 100: accTr: 0.9584, lossTr: 0.1556, accVa: 0.8834, lossVa: 0.4538
Epoch 120: accTr: 0.9586, lossTr: 0.1547, accVa: 0.8832, lossVa: 0.4552
Epoch 140: accTr: 0.9588, lossTr: 0.1544, accVa: 0.8831, lossVa: 0.4555
Epoch 160: accTr: 0.9588, lossTr: 0.1542, accVa: 0.8831, lossVa: 0.4557
Epoch 180: accTr: 0.9588, lossTr: 0.1542, accVa: 0.8831, lossVa: 0.4557
Epoch 200: accTr: 0.9588, lossTr: 0.1542, accVa: 0.8831, lossVa: 0.4557
```

Figure 10: performance after dropout

3.6 Better architecture

We decided to use ResNet18 (residual network) for better performance. The residual network become easier to optimize by adding shortcut connection. Every ResNet have three main part: input, output and convolution part.

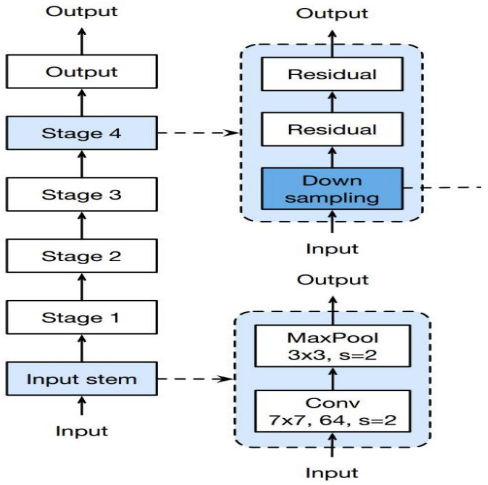


Figure 11: Architecture of ResNet18

ResNet18 build on imagenet dateset pretraining and we used **transfer learning**, which give us better inner parameters. We have also made some changes to suit our model. We changed the last layer (by adding model_ft.fc = nn.Linear(num_ftrs, 10), because we only have 10 output classes), and we also changed the first layer (by adding num_ftrs = model_ft.fc.in_features, to make it suitable for our input 32*32).

```
Epoch 0: accTr: 0.9245, lossTr: 0.2558, accVa: 0.9108, lossVa: 0.2990
Epoch 5: accTr: 0.9757, lossTr: 0.0934, accVa: 0.9329, lossVa: 0.2441
Epoch 10: accTr: 0.9938, lossTr: 0.0225, accVa: 0.9377, lossVa: 0.2886
Epoch 15: accTr: 0.9978, lossTr: 0.0081, accVa: 0.9387, lossVa: 0.3570
Epoch 20: accTr: 0.9980, lossTr: 0.0069, accVa: 0.9382, lossVa: 0.4137
Epoch 25: accTr: 0.9998, lossTr: 0.0008, accVa: 0.9419, lossVa: 0.4401
Epoch 30: accTr: 0.9999, lossTr: 0.0001, accVa: 0.9443, lossVa: 0.5497
Epoch 35: accTr: 1.0000, lossTr: 0.0000, accVa: 0.9438, lossVa: 0.6606
Epoch 40: accTr: 1.0000, lossTr: 0.0000, accVa: 0.9442, lossVa: 0.7420
```

Figure 12: Performance of ResNet18

3.7 Semi-supervised Learning

Semi-supervised learning is inductive, so the generated model can be used widely. We use extra 32*32.mat as unlabeled data for semi-supervised learning. We combine Semi-supervised learning with ResNet18, and the validation accuracy is 0.9415. The reason why performance did not get much improvement is that train 32*32.mat are more complex than extra 32*32.mat dataset. A simple dataset may have less improvement for a model trained by a complex dataset.

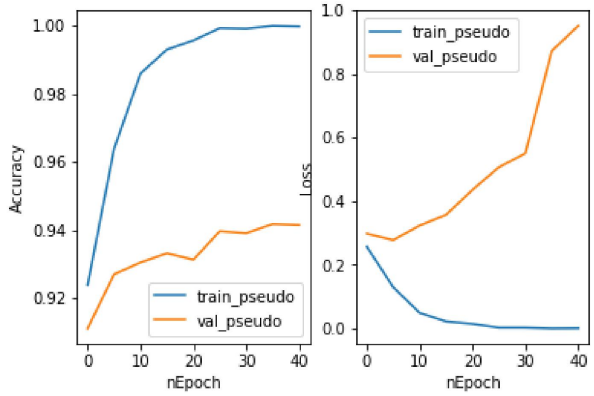


Figure 13: Performance of Semi-supervised learning

4. Model Evaluation And Comparison

	Simple CNN	CNN with dropout	ResNet18	Semi-Supervised
Accuracy	0.8820	0.8725	0.9442	0.9405

5.Result Visualization

For ResNet18 model, we visualize the predict result and find out number 8 is the hardest one for our model to predict. We also show some mispredict image.

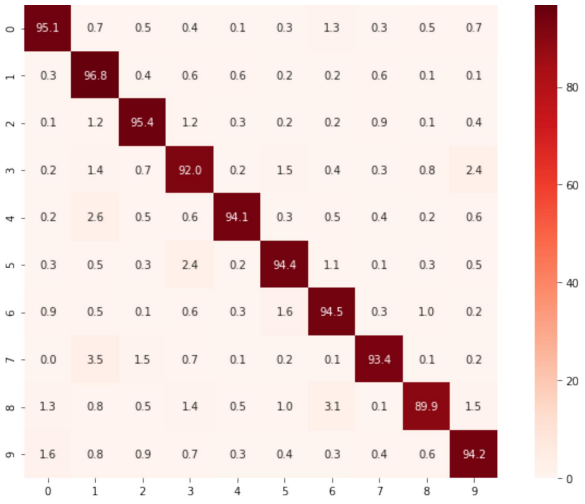


Figure 14: Visualization of ResNet18 result

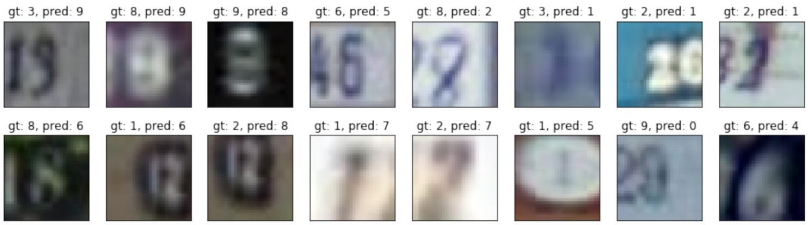


Figure 15: Sample of mispredict image