Yue Wang
5293 Image Analysis
Individual Project on SVHN Classification with CNN

i. Explanation of how CNN works

CNN, convolutional neural network, is a deep learning method that performs well in image recognition and classification. It is built up by input lay, hidden layers, and and output layer. The hidden layers can be further break down by convolutional layers, pooling layers and fully connected layers.

In the convolutional layer, we can define kernel of a particular size, which is going to slide across the image and extract features from the graph, and the number of filters. The result can be passed into the pooling layer that would use a threshold to determine if we find the particular feature or not. This layer would also help reduce dimensionality. Then, in the fully connected layer, we connect the the filter with the extracted feature to compute a score, and the highest score will be labeled the same as the shape corresponding to the filter.

ii. Explanation of each and every parameter/hyper parameter of CNN with its function

a. Learning rate: It controls the speed of the model adjusting itself in response to estimated errors.
b. Epochs: The number of time to run the algorithm to the dataset.
c. Batch Size: the number samples that the model processes
d. Stride: The number of pixel steps for filter to through images
e. Padding: If we use 0 paddings, we add 0s around the image to ensure the edges of images are covered.
f. Kernel Size: Kernel size is the size of filters, the commonly used filters are $1\times1, 3\times3, 5\times5$ which are little blocks that scans the image.
g. Optimizer: Algorithms that can help the model reduce losses.
h. Activation functions: A specific function that allows the model to study nonlinear features.
i. Number of filters: the number of filters is like the number of neurons that will result into a feature map.

iii. Detailed explanation of you code logic

The overall process includes the following steps:

1. Load data;
2. Prepare training and testing data set: for the training and testing dataset, extract the images and their corresponding labels, and make them to be four sets of data: train_images, train_target, test_images, test_target;
3. Process two sets of data into the form required by the model

The training and testing image data sets are in dimensions (32, 32, 3, 73257) and (32,32,3,26032) respectively. The images are in RGB.

I changed them into gray scale and convert the dimensions into (73257, 32, 32, 1) and (26032, 32, 32, 1) so that the first dimension matches with the first dimension of labels data sets.
Then I process the 10 labels into one-hot form.

4. Build the Model
   In the modeling step is to build the architecture by adding layers.
5. Train the Model and Validate it
   Fit data to the model, choose validation data, and choose number of generations
6. Evaluate
   Observe the accuracy rate
   See the prediction
   Losses

iv. Starting model and how you choose that

Below is the summary of the initial model:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 30, 30, 16)        160
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 15, 15, 16)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 13, 13, 32)        4640
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 6, 6, 32)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                11530
=================================================================
Total params: 16,330
Trainable params: 16,330
Non-trainable params: 0
```

Figure 1: table summary of the starting model

For the initial model, I chose to start with basic model that only consists of three typical components, convolutional layer, pooling layers, dense layers. I added two convolutional layers, each with a subsampling layer, starting from a small number of filters to larger number of filters, which goes deeper to look for features. The dense layer performs classification and classifies features into 10 groups, one for each digit.

The resulting testing accuracy 83.01%, and loss is 62.89%. The later models will aim to improve accuracy and reduce losses base on this model.

v. Model evaluation

I will use two ways to evaluate the model: the loss function and the accuracy rate.

vi. How you reached final model from staring model with intermediate states (with visualization)

Starting Model Performance:

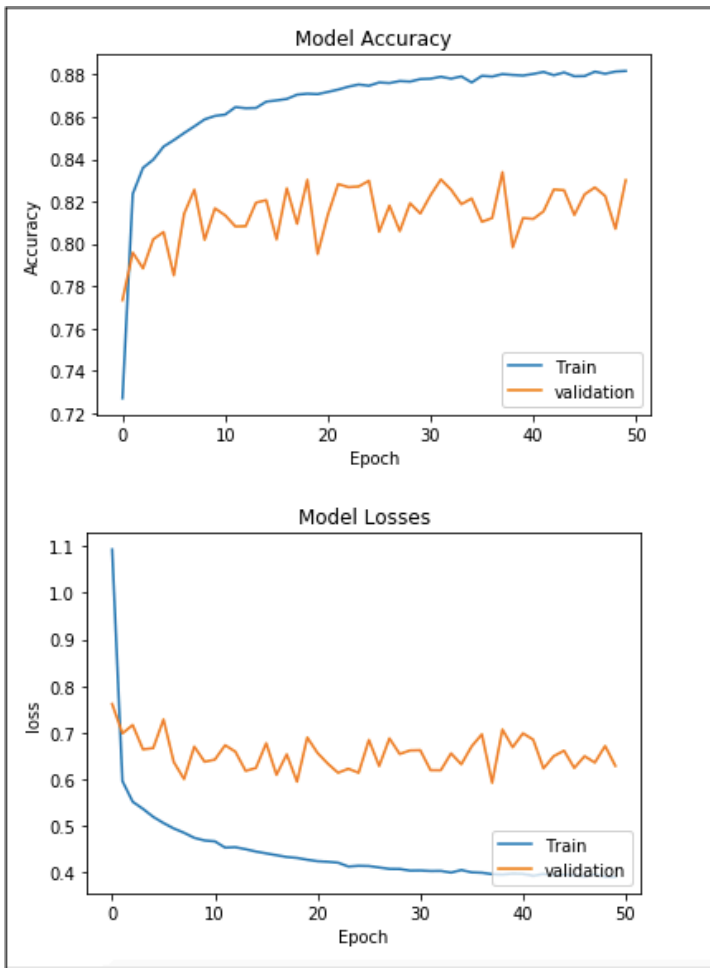

Figure 2: Performance of the staring model

Based on the starting model, I first tried modifying the architectures and compare the effects of each modification:

- Change number of max pooling layer and its position: Reducing the number of max pooling layer will cause overfitting, therefore, the model loses prediction

ability when it is applied to a new dataset. As the accuracy for training data climbing, the accuracy for the validating dataset shows a declining trend.

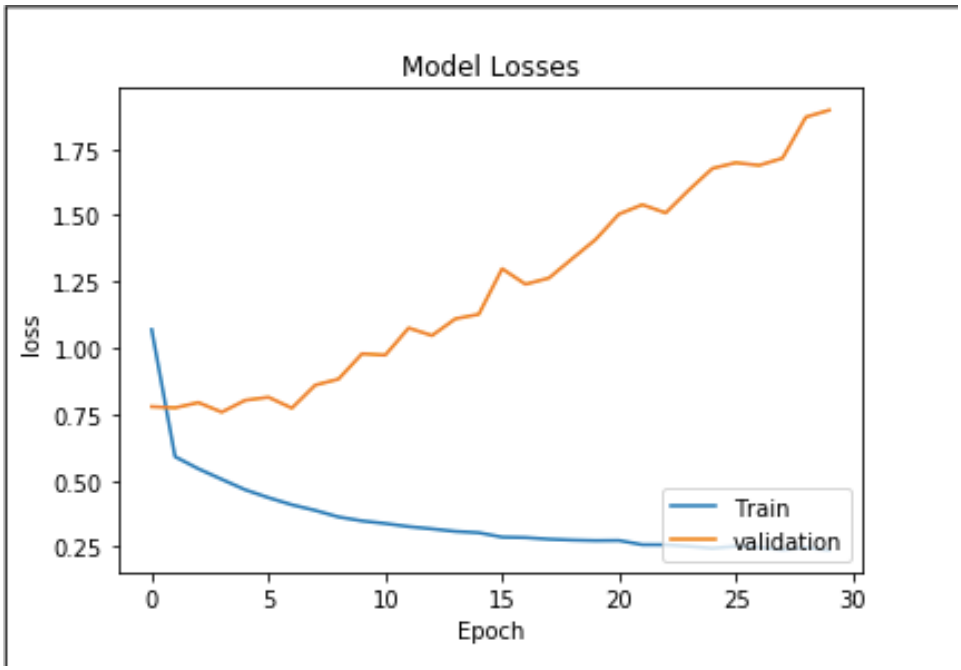

Figure 3: Loss graph after removing the max pooling layer

- Change the number of filters: I doubled the number of filters of the initial model. Increasing the number of filters will not help with the generalization ability. It helps to increase the accuracy for the training dataset. However, the accuracy for the testing set is not improved.

- Add more convolutional layers: The model performs well on the training dataset, but it doesn't have as good prediction ability. It has the same problem as the previous one.

- Add more fully connected layers with higher output dimension/Add dimension the dimension of fully connect layer: The model losses for validating dataset keeps increasing as the loss for training dataset decrease. However, the accuracy for the training is increased.

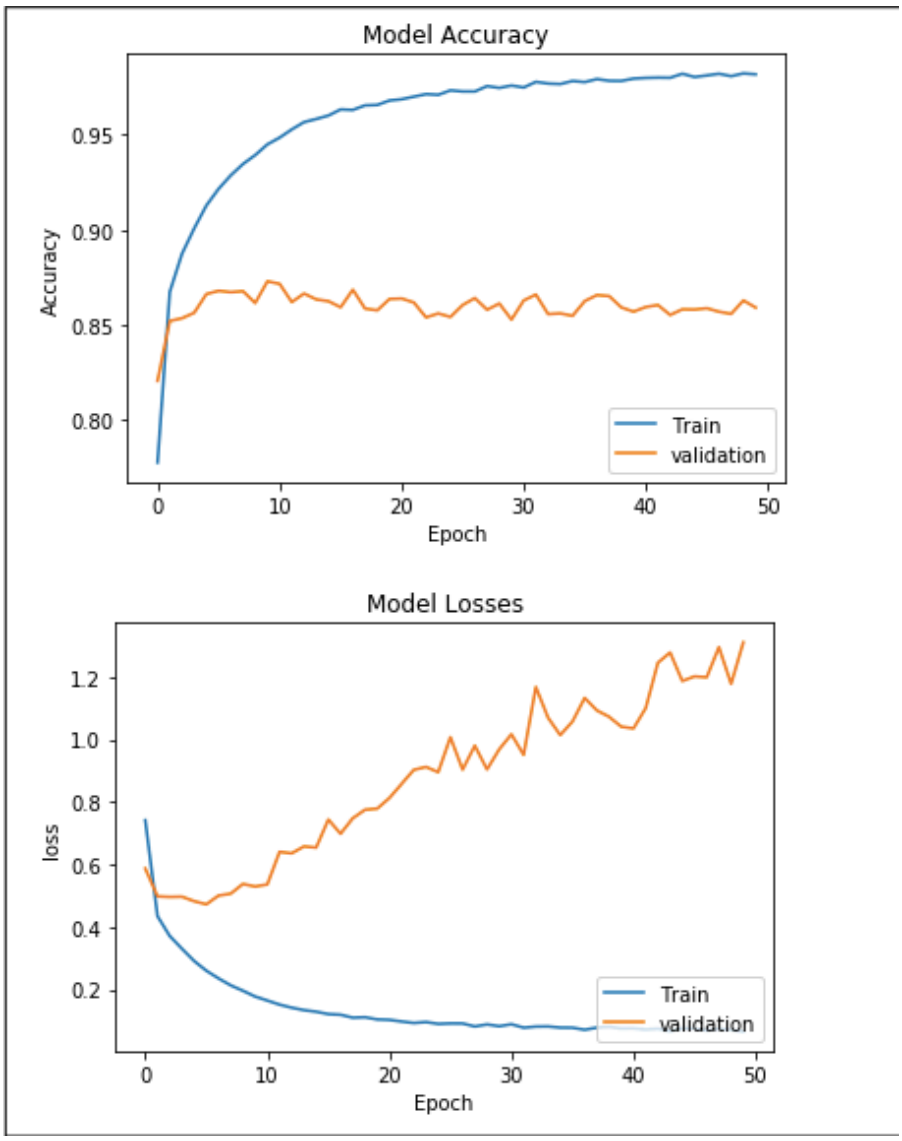

Figure 4: Three Fully Connected layers with dimensions 128, 64, 10

- Add Dropout: Shows the ability to reduce overfitting. Narrows the gap between accuracy between training and validating dataset.

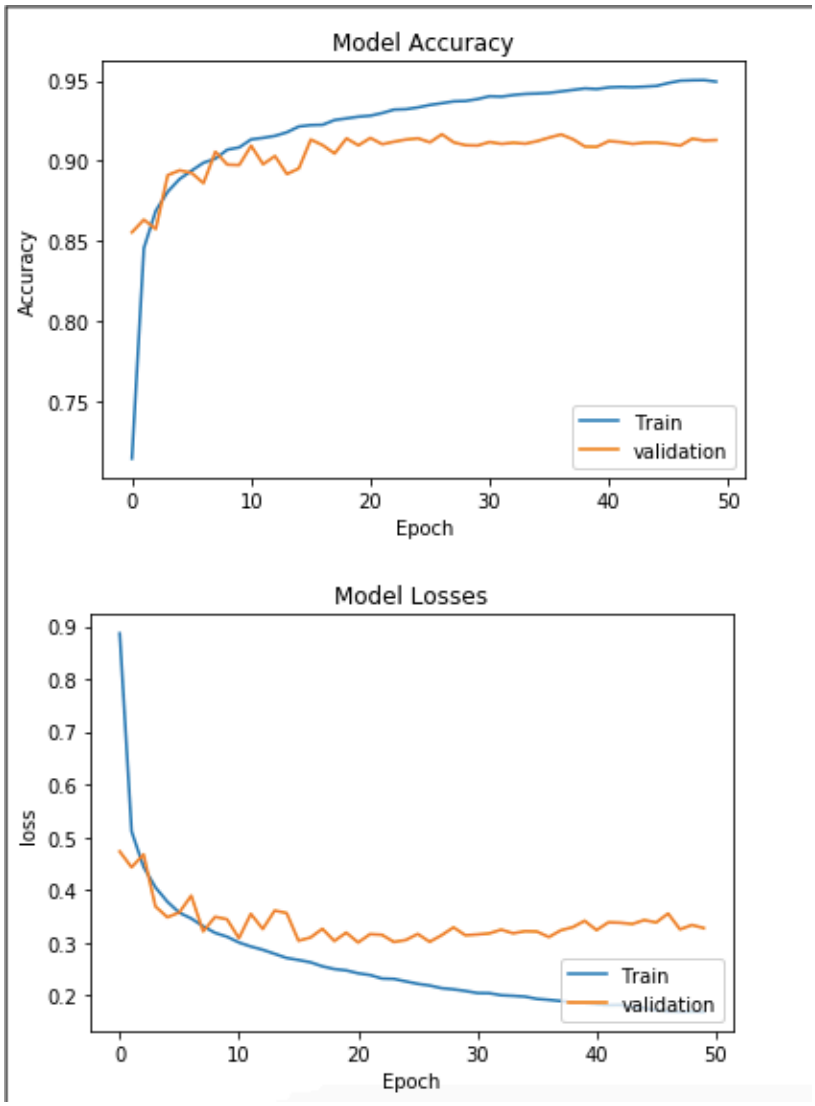

Figure 5: Performance after adding the dropout layer

- Tune learning rate parameter: I tried learning rate to be 0.01 and 0.001. Adding the learning rate of 0.01 doesn't improve the model performance much. The accuracy increased by 0.2%, but figure 6 shows that the prediction accuracy is very unstable. Learning rate of 0.001 improve the training accuracy by 2% and testing accuracy rate by 0.3%. And the prediction accuracy rates are stable for across all epochs (Figure 7).

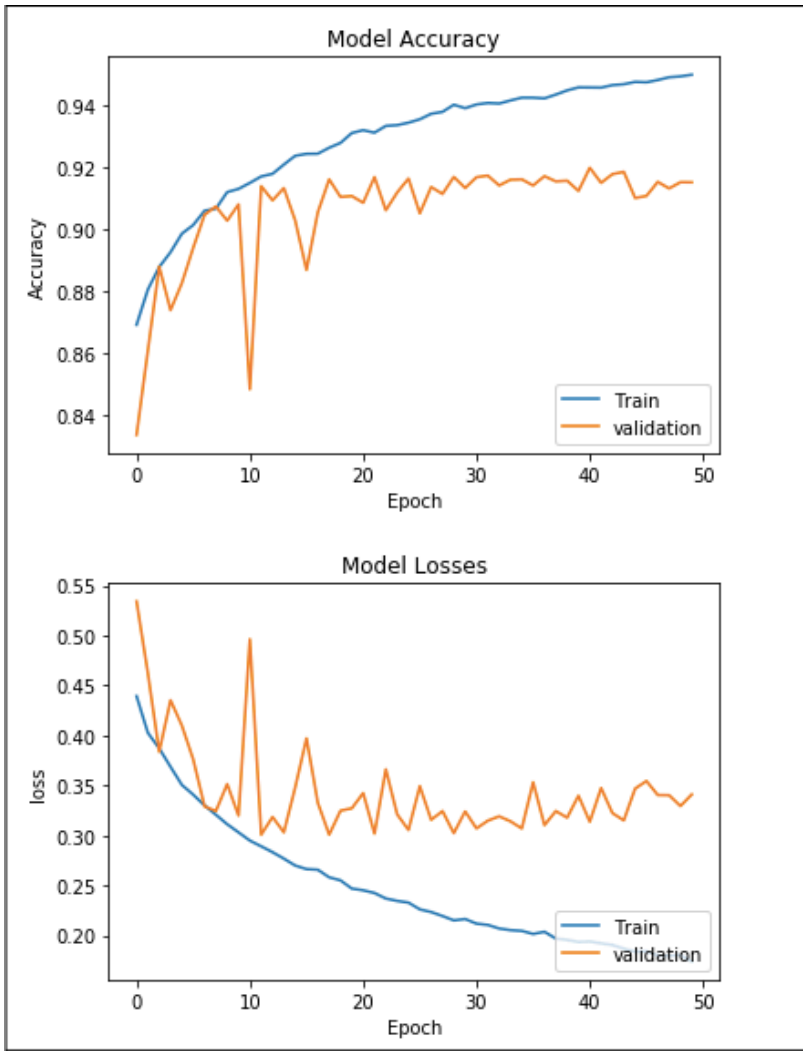

Figure 6:Final model with Learning rate = 0.01, epochs=50

After exploring the effects to the model performance after changing different parameter and the structures, final model was built with the goal to increase the accuracy of training data while reducing overfitting. Based on the starting model, the final model uses five convolutional layers with the max pooling layer which help increase the accuracy of the training dataset. It added dropout to reduce overfitting. To further improve the performance, I set the learning rate to be 0.001. The resulting validation accuracy rate is 91.63%.

Below is the summary of the final model and evaluation of its performance:

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 30, 30, 16)        160
_________________________________________________________________
batch_normalization_3 (Batch (None, 30, 30, 16)        64
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 28, 28, 32)        4640
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 12, 12, 64)        18496
_________________________________________________________________
batch_normalization_4 (Batch (None, 12, 12, 64)        256
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 10, 10, 32)        18464
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 5, 5, 32)          0
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 3, 3, 16)          4624
_________________________________________________________________
batch_normalization_5 (Batch (None, 3, 3, 16)          64
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 1, 1, 16)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1, 1, 16)          0
_________________________________________________________________
flatten_10 (Flatten)         (None, 16)                0
_________________________________________________________________
dense_18 (Dense)             (None, 64)                1088
_________________________________________________________________
dense_19 (Dense)             (None, 10)                650
=================================================================
Total params: 48,506
Trainable params: 48,314
Non-trainable params: 192
```

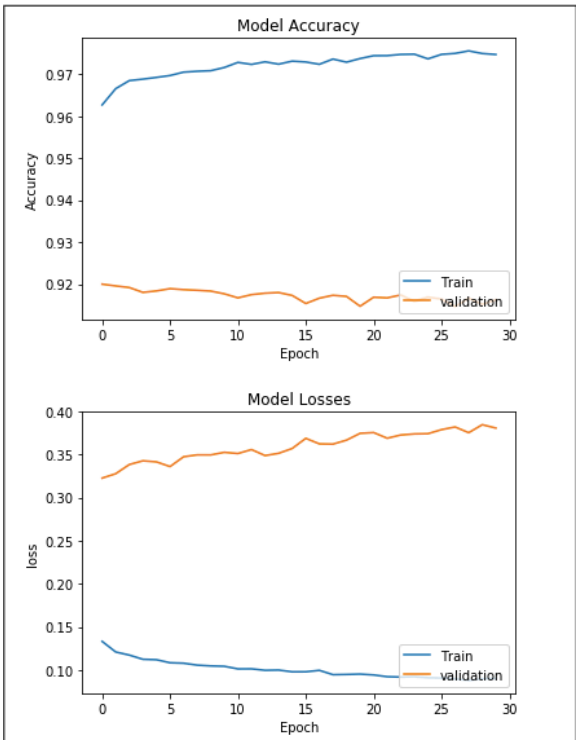

Figure 7:Final model and its performance with learning rate = 0.001, epochs = 30

vii. What are the difficulties faced while implementing the model?

First, there are many hyper parameters;
Second, there are various ways to change the arrangement of different layers;
Third, the training process is long;

Overall, it is a very tedious process. To implement the model, I had to configure different parameters manually and try different combinations each time after the structures of the models are modified. Although many CNN models have been built with parameters that yield the satisfactory results, there is not too much to borrow from because each dataset is different. At some time during the training process, it is hard to distinguish if I should change the structure of the model or continue to refine the parameters. Since training time for each model is quite long, it's impossible or costly to test out all possibilities. These are all the difficulties I met when I implemented the model.

viii. How can you improve the model further?

There are several methods that will potentially improve the model:

1. Expand the training sample set, for example, by data augmentation.
2. Further tune the hyper parameters.
   When training the model, I tried epochs of 30 and 50. Increasing the number of epoch will potentially improve the performance.

We can see that changing the learning rate definitely helps with the performance for the training data, and the accuracy for testing data is increased by 0.1%. We can definitely test on more learning rates to find the best one that fits our model.

3. The overfitting issue of the final model is not severe, but we can try to add weight decay or early stopping to see the results.
4. Go deeper or wider with the network.

Reference Links

1) https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7

2) https://medium.com/@dipti.rohan.pawar/improving-performance-of-convolutional-neural-network-2ecfe0207de7

3) https://thesai.org/Downloads/Volume10No6/Paper_38-Hyperparameter_Optimization_in_Convolutional_Neural_Network.pdf