Multi-Digit Number Recognition from Street View Imagery Using Deep Convolutional Neural Networks

E4040.2019Fall.XSQD.report

Manchun Sun ms5705, Xuchen Qiu xq2187, Ziyuan Xiong zz2246

Columbia University

Abstract

This project is an implementation of the given paper of recognizing the series of numbers in the street view images. The goal and the objective are to recognize the numbers and the length with as high accuracy as possible (over 90%). The significance is that we could apply our model to recognize the numbers in street view images automatically and mark them on maps. The major challenge is to deal with large datasets and to design a neural network with proper architecture and train it to achieve high accuracy. We used Google Cloud with GPUs, tried Keras instead of TensorFlow and tuned the hyperparameters to overcome the challenges. The final result gives an accuracy of over 90% with extra dataset included in training, and our goal was successfully achieved.

1. Introduction

In this project, we tried to replicate the paper "Multi-digit number recognition from street view imagery using deep convolutional neural networks, 2014" written by Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. The paper focuses on recognizing the number and letter series in the Street View House Numbers (SVHN) dataset, the Internal Street View dataset and CAPTCHA puzzles dataset using convolution neural network architecture. The major goal and objective is to design and train a neural network to recognize the series of numbers in the dataset with high accuracy.

One of the biggest challenges of this project is the dataset contains a great number of images which made preprocessing the data and training the neural network very costly. Hence we used Google Cloud with GPU to do all the process on the cloud to save time and efforts. Another big challenge is that we have to train an architecture with high accuracy since the result in the paper is so outstanding. Thus we applied extensive hyperparameter tuning using grid search and Bayesian optimization to achieve this goal.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The original paper used the probabilistic model to calculate the joint probability of the digit sequence by assuming independence. Next, it preprocessed the SVHN dataset and load the images. Then it used convolutional neural network to make predictions on the SVHN dataset. After that, it adjusted the model for two more challenging datasets, the Internal Street View dataset and CAPTCHA puzzles dataset and used the adjusted model to make predictions on them as well.

The flowchart of original paper is shown below.

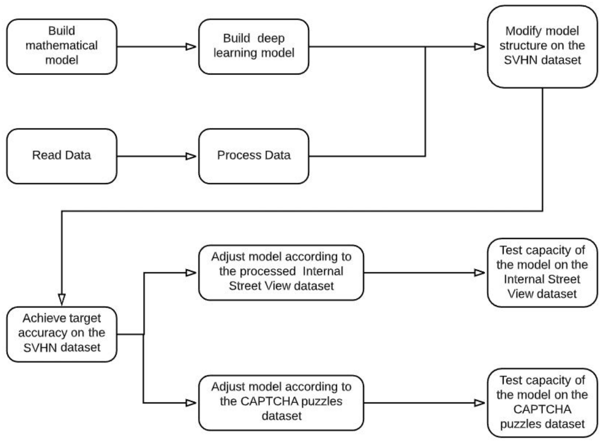

Build mathematical model
Build deep learning model
Modify model structure on the SVHN dataset
Read Data
Process Data
Achieve target accuracy on the SVHN dataset
Adjust model according to the processed Internal Street View dataset
Test capacity of the model on the Internal Street View dataset
Adjust model according to the CAPTCHA puzzles dataset
Test capacity of the model on the CAPTCHA puzzles dataset

Figure: Flowchart of Original Paper

2.2 Key Results of the Original Paper

The best model for predicting the SVHN dataset has an accuracy of 96.03%, and an accuracy of 98% with 95.64% coverage using confidence thresholding. It achieves a character-level accuracy of 97.84%. The best architecture consists of eight convolutional hidden layers. For the Internal Street View dataset, the obtained accuracy is 91% and 99% with 83% coverage using confidence thresholding. The architecture is similar to the SVHN one by replacing eight convolutional hidden layers to five. For the CAPTCHA puzzles dataset, the model is able to achieve 99.8% accuracy on transcribing the hardest reCAPTCHA puzzle with nine convolutional layers instead of 11.

3. Methodology

This section explains the objective and methodology of our implementation.

3.1. Objectives and Technical Challenges

Objective:

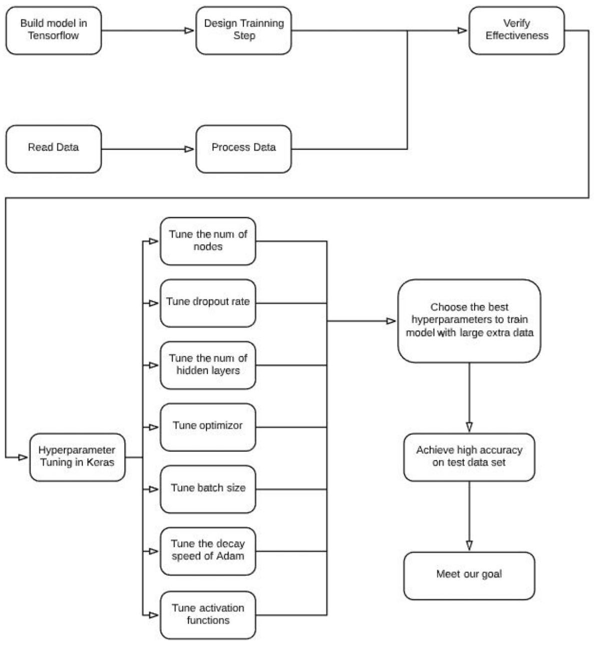
1. The first objective is to load the images and their true labels.
2. Second, we need the crop and resize the images to make it easier or our model to classify them.
3. Third, we need to build a neural network architectural to do the image classification task.
4. Fourth, we need to train the model to make it proficient in recognizing the images and classify it correctly and achieve a high accuracy on the test set.

Technical Challenges:

1. First, the dataset consists of a large number of images, which made the dataset very big and hard to handle.
2. Second, the true labels are stored in a MATLAB file so we need to load the file into Python readable format.
3. Third, we need to crop the images using their bounding box label and make reasonable adjustments to make sure the key information in the image is captured.
4. Fourth, we need to build a neural network architecture with the reasonable and appropriate structure (number of convolutional layers, etc.).
5. Fifth, it is challenging to train the network to achieve a high accuracy. Extensive hyperparameter tuning is needed.
6. Sixth, most of our models are very large and deep. Long time training on the cloud is needed.

3.2 Problem Formulation and Design

The flow chart of our design to solve this Multi-Digit Number Recognition problem is shown below.



Build model in Tensorflow
Design Trainning Step
Verify Effectiveness
Read Data
Process Data
Tune the num of nodes
Tune dropout rate
Tune the num of hidden layers
Tune optimizor
Tune batch size
Tune the decay speed of Adam
Tune activation functions
Hyperparameter Tuning in Keras
Choose the best hyperparameters to train model with large extra data
Achieve high accuracy on test data set
Meet our goal

Figure: Flow Chart of Solving the Multi-Digit Number Recognition Problem

3.2.1 Mathematical Model Methodology

Our approach is to use the same mathematical model as the paper to calculate the probability of a specific sequence.

Define S as a collection of N random variables and $S_1, \ldots, S_N$ as the elements of the sequence and an L as the length of the sequence. An important assumption is that the identities of the separate digits are independent from each other. Hence the probability of a specific sequence is given by

$$P(\mathrm{S} = \mathrm{s}|X) = P(L = n \mid X)\Pi_{i=1}^{n} P(S_i = s_i \mid X).$$

3.2.2 Data Processing Methodology

Image cropping and data augmentation was implemented to increase the effectiveness and diversity of the data.

3.2.3 Model Methodology - TensorFlow

In the beginning, we built our network using TensorFlow structure. Since we have written training algorithm in our assignment to train LeNet model in TensorFlow, we modified it to train our network. The training function includes sub functions such as the "train" function, which is used to train and validate our previously built neural network, the "evaluate" function, which is used to calculate the accuracy of the model, the "train step" function, which is used to set the type of optimizer used in training (Adam, RMSprop etc.) and so on.

3.2.4 Model Methodology - Keras

Since the model in written in TensorFlow is very time costly, including around 9 to 10 hours of training. We thus consider using Keras instead giving the impending deadline. We also compared the performances between the model in Keras and model in TensorFlow and found that they were similar. Building the architecture using Keras is much easier since Keras has more built-in functions to simplify the process.

3.2.5 Parameter Tuning Methodology

We slightly modified the parameters of the network to test its performance. We further fine tune the hyperparameters to make our model perform better, including tuning learning rate, batch size, dropout rate, number of epoch etc. We also tried various types of optimizer to see the performance, for example, RMSprop, Adam, Stochastic Gradient Descent, etc. We major used two methodologies in tuning the hyperparameters, including Bayesian optimization and manual tuning. After determining the hyperparameters, we combined the training data with extra data to test the performance of our model on a larger dataset. The paper did not introduce the detailed hyperparameter tuning algorithm and the exact hyperparameters used. Hence we use our own approach to decide what values could be potential good hyperparameters, etc.

4. Implementation

In this part, we introduce the details of the structure of our network, the results we obtained in hyperparameter tuning, and our implementation of the neural network and our results.

4.1. Deep Learning Network

4.1.1 Data Processing

We used the Google Street View House Number (SVHN) Dataset as the original paper, which has three parts train.tar.gz, test.tar.gz , extra.tar.gz.
(website: http://ufldl.stanford.edu/housenumbers/)

In the beginning, we only used the "train" dataset of SVHN, which consists of 33402 images. However, the validation accuracy is not so satisfactory (approximately 70%), so we further add the "extra" dataset to train our model, which includes 202,353 images. With the "extra" dataset added to training, our model performed much better than before, and our best model achieved an accuracy of 92.44% on test set.

1. Read and Crop Data: we unzipped the file and cropped the original data according to their bounding information, which is stored in digitStruct.mat file inside. We stored the cropped images in extra_cropped, test cropped and train cropped folders. (crop.py)

Figure: Test Data after Cropping

Figure: Test Data After Data Augmentation

2. Store Data and Data Augmentation: we read the label data to array of shape (n, 6) and image data to big arrays of shape (n, 64, 64, 3). We stored these arrays in .npz files. Then, we did data augmentation by adding small gaussian blur, randomly cropping, changing the color of the pixels, scaling, zooming, shearing and so on. Below is a sample of our data augmentation process.
(read_labels.ipython, read_image_to_array.ipynb, data_augmentation.ipynb)

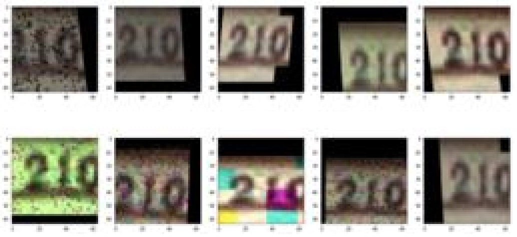


Figure: Data Augmentation Process

4.1.2 Network Built in Tensorflow

Without directly using the Tensorflow function, the layer functions that we have learnt and build in Assignment 2 (layers.py) were used to provide formaler and safer deep learning model.
In order to ensure the effectiveness of our model, we first design our deep learning model in the same structure and parameters as the one was used in the original paper. After verifying the capacity of our model, we adjusted the model structure like using different numbers of hidden layers, changing activation functions and modifying the number of nodes in layers in order to achieve a better performance.
The parameters in the training step was first set up to normal values without dropout and learning rate decay. The Gradient Descent and Adam optimizers were used in the first tensorflow model and achieved an accuracy of 55% and 69% respectively. The hyperparameter tuning was later performed in Keras since the model training in Keras is much more efficient and time-saving.

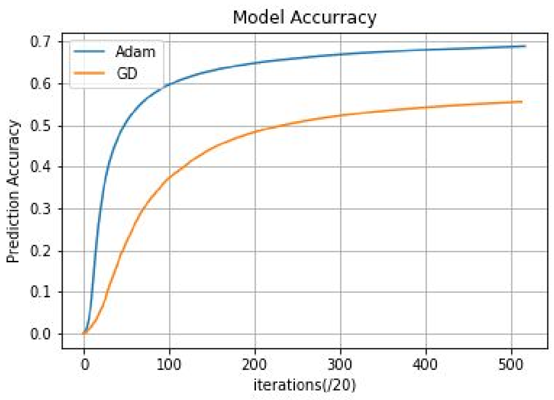


Figure: Accuracy Performances of GD and Adam

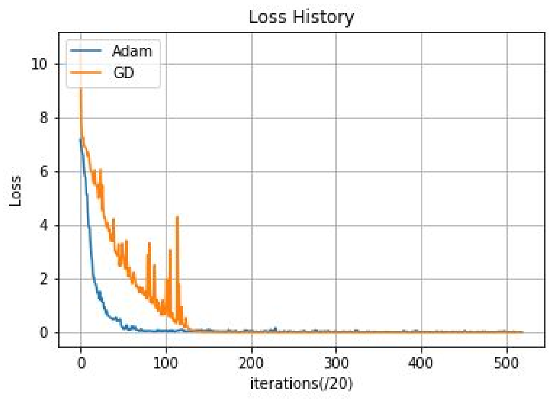
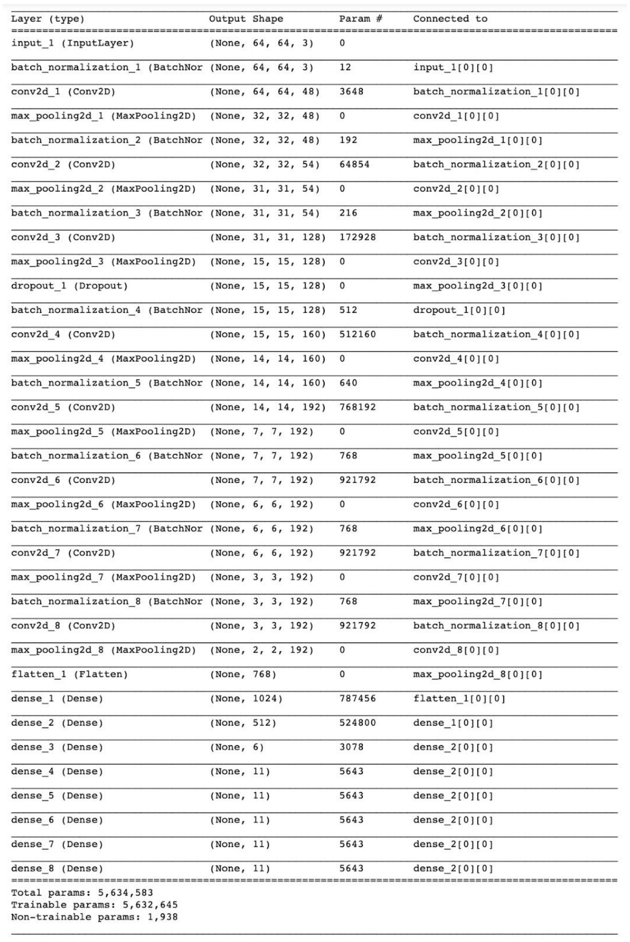


Figure: Loss Performances of GD and Adam

Model Training(train_with_predict.py)

Model Architecture (My_cnn.py):
The detailed structure is described in the following figure.



```
Layer (type)                     Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)             (None, 64, 64, 3)    0
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 64, 64, 3)    12          input_1[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)                (None, 64, 64, 48)   3648        batch_normalization_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)   (None, 32, 32, 48)   0           conv2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 32, 32, 48)   192         max_pooling2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)                (None, 32, 32, 54)   64854       batch_normalization_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)   (None, 31, 31, 54)   0           conv2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 31, 31, 54)   216         max_pooling2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)                (None, 31, 31, 128)  172928      batch_normalization_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)   (None, 15, 15, 128)  0           conv2d_3[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 15, 15, 128)  0           max_pooling2d_3[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 15, 15, 128)  512         dropout_1[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)                (None, 15, 15, 160)  512160      batch_normalization_4[0][0]
__________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)   (None, 14, 14, 160)  0           conv2d_4[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 14, 14, 160)  640         max_pooling2d_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)                (None, 14, 14, 192)  768192      batch_normalization_5[0][0]
__________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)   (None, 7, 7, 192)    0           conv2d_5[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 7, 7, 192)    768         max_pooling2d_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)                (None, 7, 7, 192)    921792      batch_normalization_6[0][0]
__________________________________________________________________________________________________
max_pooling2d_6 (MaxPooling2D)   (None, 6, 6, 192)    0           conv2d_6[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 6, 6, 192)    768         max_pooling2d_6[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)                (None, 6, 6, 192)    921792      batch_normalization_7[0][0]
__________________________________________________________________________________________________
max_pooling2d_7 (MaxPooling2D)   (None, 3, 3, 192)    0           conv2d_7[0][0]
__________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 3, 3, 192)    768         max_pooling2d_7[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)                (None, 3, 3, 192)    921792      batch_normalization_8[0][0]
__________________________________________________________________________________________________
max_pooling2d_8 (MaxPooling2D)   (None, 2, 2, 192)    0           conv2d_8[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 768)          0           max_pooling2d_8[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                  (None, 1024)         787456      flatten_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                  (None, 512)          524800      dense_1[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                  (None, 6)            3078        dense_2[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                  (None, 11)           5643        dense_2[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                  (None, 11)           5643        dense_2[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                  (None, 11)           5643        dense_2[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                  (None, 11)           5643        dense_2[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                  (None, 11)           5643        dense_2[0][0]
==================================================================================================
Total params: 5,634,583
Trainable params: 5,632,645
Non-trainable params: 1,938
__________________________________________________________________________________________________
```

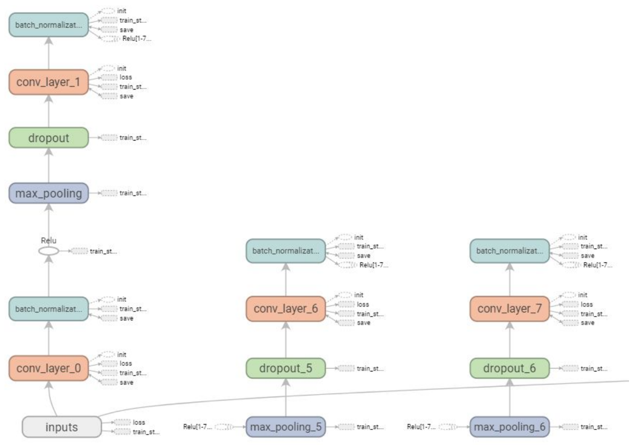


Figure: Neural Network Architecture

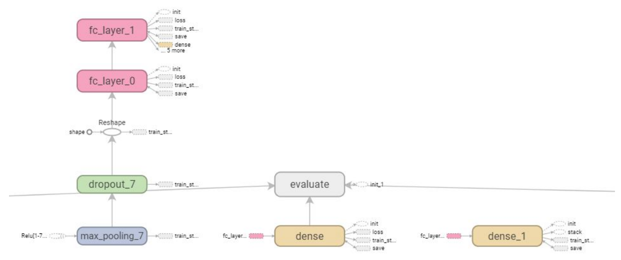


Figure: Neural Network Architecture (continued)

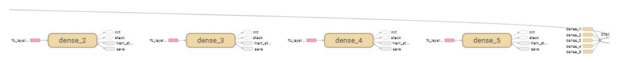


Figure: Neural Network Architecture (continued)

4.1.3 Hyperparameters Tuning in Keras

Since running our tensorflow model took several hours, to find the best model, we utilized Keras package to run models with different hyperparameters and find the best model quickly. We tuned our hyperparameters both with Manually Design and Bayesian Optimization, which is an approach that uses Bayes Theorem to direct the search in order to find the minimum or maximum of an objective function.

In this part, we used control variate method. For example, we used the same dropout rates, epoch sizes, batch sizes and hidden layer numbers with different optimizer to test the effect of different optimizers.

Name	Optimizer	Layers	Dropout Rate	Learning Rate	Decay
Model1_1	SGD	8	0	0.001	no
Model1_2	RMSProp	8	0	0.001	yes
Model1_3	Nadam	8	0	0.001	yes
Model1_4	Adam	8	0	0.001	yes
Model2_1	Adam	8	0.3	0.001	yes
Model2_2	Adam	8	0.5	0.001	yes
Model3_1	Adam	2	0	0.001	yes
Model3_2	Adam	4	0	0.001	yes
Model3_4	Adam	6	0	0.001	yes
Model3_5	Adam	8	0	0.001	yes
Model4_1	Adam	8	0	0.005	yes
Model4_2	Adam	8	0	0.01	yes

Figure: Manually Hyperparameter Design

4.1.3.1 Optimizer

Firstly, we tuned our model with four optimizers responsively - Stochastics Gradient Descent Optimizer, RMSProp Optimizer, Adam Optimizer and Nadam Optimizer. On test set, they achieved 1.14%, 81.93%, 85.61% and 83.39% accuracy respectively.

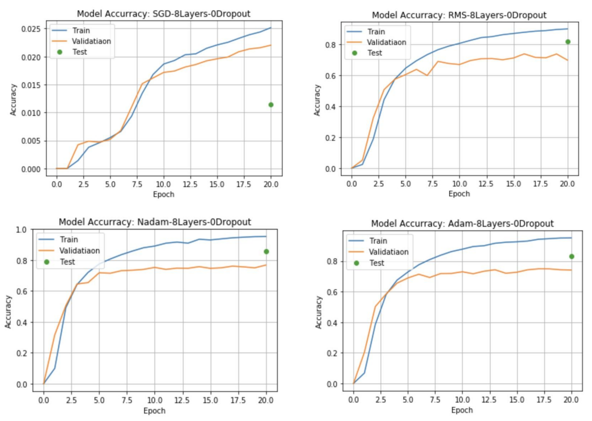


Figure: Train and Test Accuracy for Different Optimizers

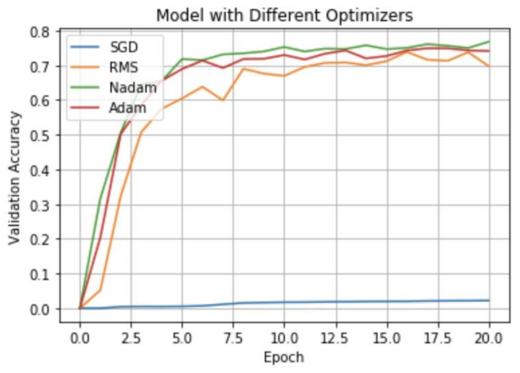
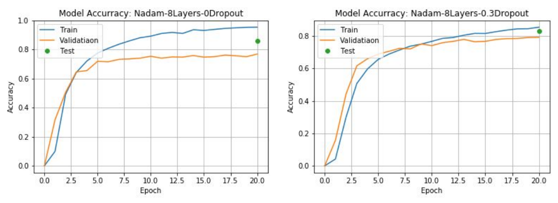


Figure: Validation Accuracy for Different Optimizers

It could be seen that Nadam Optimizer and Adam Optimizer had the best validation accuracy. SGD Optimizer had the worst validation accuracy. However, it was still keeping increase, which indicates it has a lower convergence rate. If we add the epoch sizes for SGD, it will keep improving. Taken into account both accuracy and time consuming, we used Adam Optimizer in the following process.
(Optimizer_Manual.ipython)

4.1.3.2 Dropout Rate

In this part, we implemented three dropout rates - 0, 0.3, 0.5 on 8 hidden layer CNN model with adam Optimizer. On test set, they reached 83.39%, 82.98%, 72.60% accuracies respectively as shown below.



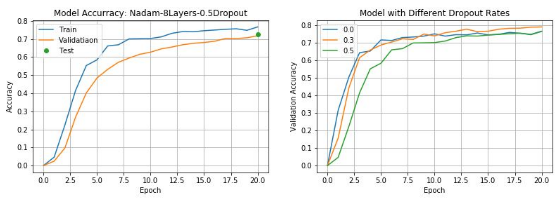


Figure: Validation Accuracy for Different Dropout rate

It could be concluded that when using 0.3 dropout rate, our model run fastest without sacrificing accuracy.

4.1.3.3 Hidden Layer

For hidden layers, we trained five models with 2, 4, 6, 8 and 10 layers respectively. We found model with 2 and 4 hidden layers both have extraordinary low accuracy on test sets (below 10%). When we trained our model with 6 hidden layers, the accuracy increased dramatically to 65%. For model with 8 and 10 hidden layers, both of them have high performance, which is almost 80%. However model with 8 layers ran more quickly.

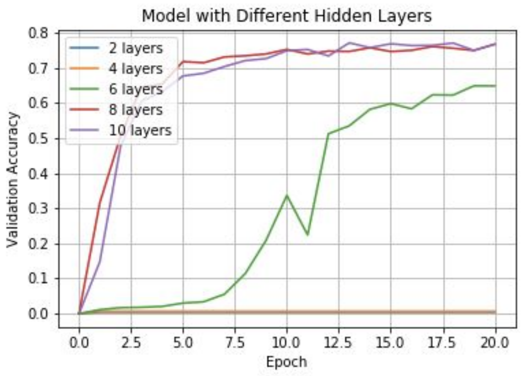


Figure: Validation Accuracy for Different Hidden Layers

4.1.3.4 Learning Rate

In this part, we implemented three learning rates - 0.001, 0.002, 0.005 on 8 hidden layer CNN model with adam Optimizer. On test set, they reached accuracy of 83.39%, 9.50%, 83.11% respectively. Models with higher learning rates are running more quickly.

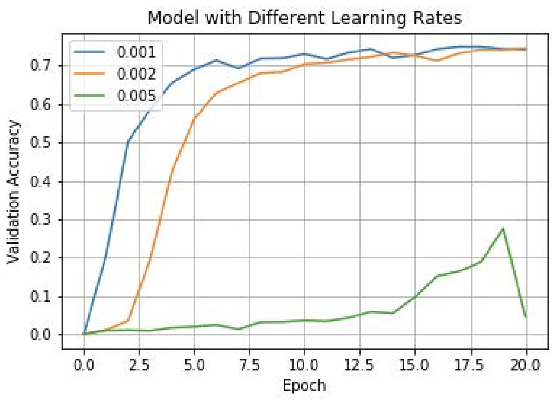


Figure: Validation Accuracy for Different Learning Rates

Since our learning rates are dynamic, they have rival influence on our final results. However, when the learning rate reached 0.005, the accuracy is pretty low, which means it can cause undesirable divergent behavior in loss function when learning rate is too large.

4.1.3.5 Bayesian Optimization

Bayesian Optimization provides a principled technique based on Bayes Theorem to direct a search of a global optimization problem that is efficient and effective. It works by building a probabilistic model of the objective function, that is then searched efficiently with an acquisition function before candidate samples are chosen for evaluation on the real objective function. Bayesian Optimization is often used in applied machine learning/deep learning to tune the hyperparameters of a given well-performing model on a validation dataset.

We have used scikit-optimize package to implement Bayesian Optimization hyperparameter tuning. We chose to tune learning_rate, number of dense layers ,number of input nodes, number of dense nodes, kind of activation function,batch_size,adam_decay etc. However, due to the large amount of time required to search for the hyperparameters, we did not manage to obtain the final results before the deadline. But our code successfully compiled and we've attached it to GitHub.

4.2. Software Design

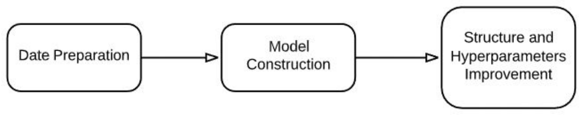


Figure: Top Level Flow Chart

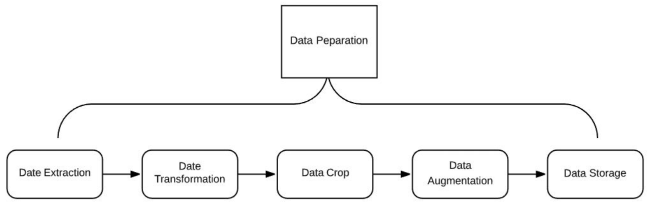


Figure: Detailed Flowchart for Data Preparation

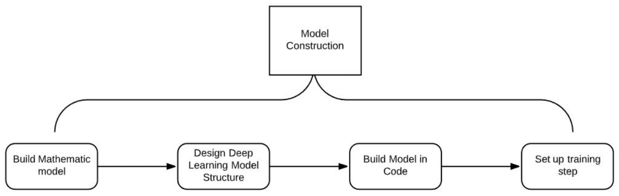


Figure: Detailed Flowchart for Model Construction

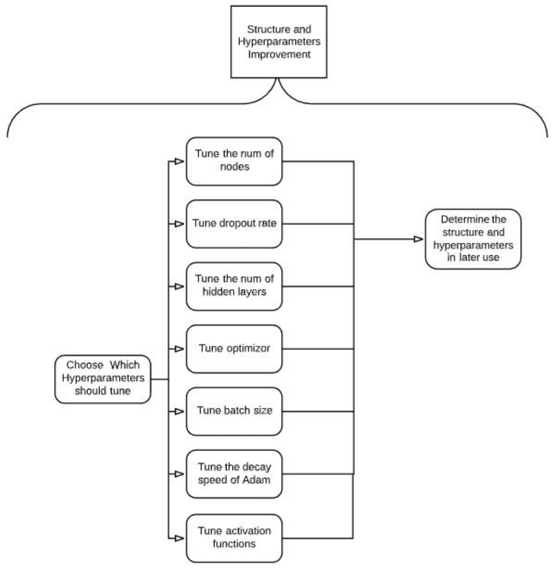

Structure and Hyperparameters Improvement
Tune the num of nodes
Tune dropout rate
Tune the num of hidden layers
Tune optimizor
Tune batch size
Tune the decay speed of Adam
Tune activation functions
Choose Which Hyperparameters should tune
Determine the structure and hyperparameters in later use

Figure: Detailed flowchart for Structure and Hyperparameters Improvement

5. Results

5.1. Project Results

The loss and accuracy history of our best model is shown below. From the picture, we can see that out best prediction accuracy has achieved 92.4%, which is an acceptable high accuracy.

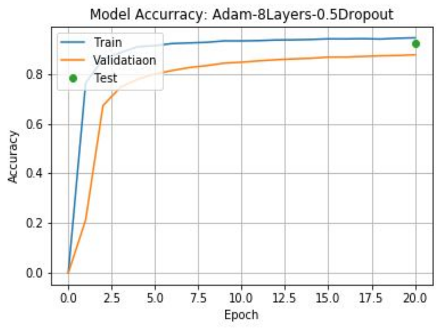


Figure: Accuracy of Best Model

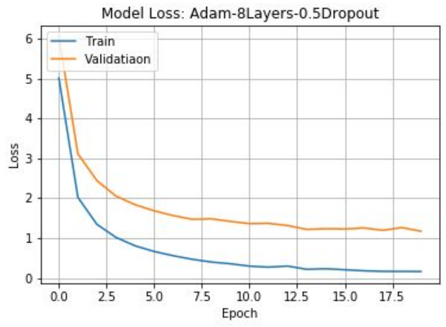


Figure: Loss of Best Model

5.2. Comparison of Results

Training Time: The paper states that the total training time for their best architecture is around 10 hours, which is relatively the same as our network implemented in TensorFlow. However, the network implemented in Keras is much more efficient, which only cost around 15 minutes to train the "train" dataset, and around 4 hours when training on the "extra" dataset.

Accuracy: The best accuracy of the model stated in the paper on the SVHN dataset is 96.03%, while the highest accuracy of our models is 92.44%.

5.3. Discussion of Insights Gained

Our results is a bit worse than the results in the given paper. Our best model used the same architecture as the paper's, including the same number of convolutional hidden layers, locally connected hidden layer, densely connected hidden layers, number of units, max pooling window size, stride size, convolution kernels size etc. We indeed used the whole dataset ("training" and "extra" dataset for training and validating the model, and "test" set for testing the model). However, we did not know what are the hyperparameters used in the paper. Hence we did necessary hyperparameter tuning to further make our results better. However, it is likely that we did not obtain all the optimal hyperparameters. Also, we did not consider to reject the case when the number of numbers in the image is greater than 5, which decreased our accuracy. Hence it is reasonable and acceptable that we did not have a performance as high as the papers'.

6. Conclusion

In sum, this project focuses on the implementation of neural network model on real-world image recognition, which is exactly what the paper has done. The dataset consists of street view images which consist of a series of numbers and it is very practical in real-world applications in geology. We have successfully load and preprocess the images in our own way. Also, we have successfully built our network in both TensorFlow and Keras. After that, we conducted cross-validation, hyperparameter tuning etc. to verify our results. Finally, we managed to obtain over 90% accuracy, which is our primary goal and objective. One highlight of our project is that we have used both TensorFlow and Keras to build our model and compare their performance, which is not mentioned and included in the original paper. The most significant lesson is to be aware of the significant time cost in neural network training. Hence powerful resources like Google Cloud machines needed to be prepared. However, it was a pity that we did not have enough time to adjust and apply our model on the Internal Street View data and the CAPTCHA puzzles dataset. Hence we could further modify our model and test its performance on more challenging datasets like these as well.

6. Acknowledgement

We would like to express our deepest gratitude towards Professor Zoran Kostic and all the TAs for the E4040 course. Professor Kostic has helped us lay a solid foundation for the deep learning theory, and TAs have offered us many practical and useful suggestions to our project during their office hours. Without them, we would not have been able to complete this project.

7. References

Include all references - papers, code, links, books.

[1]
https://github.com/cu-zk-courses-org/e4040-2019fall-project-XSQD-zx2246-xq2187-ms5705

[2] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks, 2014.

8. Appendix

8.1 Individual student contributions in fractions - table

	CUID1	CUID2	CUID3
Last Name	Sun ms5705	Qiu xq2187	Xiong zz2246
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Keras, TensorFlow Model Building	Keras, TensorFlow Model Building	Keras, TensorFlow Model Building
What I did 2	Data Augmentation	Data Preprocessing	Data Visualization
What I did 3	Report and GitHub Summarization	Report Summarization and GitHub Repo	Report Summarization and Flowcharts