# Multi-digit Recognition Based on Convolutional Neural Network with Adversarial Training

E4040.2019Fall.DCNN.report
Jiayang Zhou jz3121, Bahador Bahmani bb2969, Wenrui Zhang wz2492
*Columbia University*

## Abstract

*Digit recognition is essential for machines to understand humans' society because digits exist broadly such as in handwriting, account number and street number, et al. The challenge is that the digits vary in shape, font, and position and there is much interference including shadow and irrelevant symbols surrounding the digit. In this project, we constructed a multilayer convolutional neural network and enhance the robustness using adversarial attack. The accuracy can achieve 89% for digit series recognition using the SVHN dataset. The feature of the model is examined using the "train" data in SVHN by using different ratio of training set and validation set and early stopping.*

## 1. Introduction

Digit series recognition is a fundamental task for artificial intelligence to obtain information in handwriting digits, account number and street number, et al. from the surrounding environment. The shape, position and angle of digits vary from person to person, from place to place, which deems digit recognition to be a challenge [1]. The research started from 1985 using multilayer perceptron. Then neural network and support vector machine were combined to enhance the performance. Later in 2003, convolution neural network was introduced and achieved 0.4% error rate [2].

In this project, the multi-digit recognition system is built using convolution neural network with max pooling to implement the method explained in [1]. The project aims to achieve an accuracy of over 80% for a series of numbers. The dataset pipeline are constructed to preprocess the numbers, for instance, segmenting a digits group. To ensure the network is robust to the noise, adversarial training is implemented. We enrich our model and data-set with the adversarial attack to enhance the confidence level of predictions and check the robustness of our CNN architecture for the SVHN application.

## 2. Summary of the Original Paper

## 2.1 Methodology of the Original Paper

Before introducing the method implemented in this paper, the method in the sample article [1] will be shown below.

Structure of the method in the article

Objective function: $\max P(\boldsymbol{S}|X), P(\boldsymbol{S}|X) = P(L=n|X)\prod_{i=1}^{n} P(\boldsymbol{S_i} = s_i |X)$

$X = X - mean(X)$

Train: $X \rightarrow convolutional\ layers \rightarrow 1\ locally\ connected\ layer \rightarrow 2\ dense\ layers$

Predict: s = $(l, s_1, \dots, s_l) = argmax_{L,S_1,\dots,S_L} \log \mathrm{P}(S|X)$

Fig 1.1: Structure of the network in [1]

In Fig1.1, S represents the sequence of random variables S1, …, Sn. X represents the input image and s is the prediction based on the training results. The maximum length that can be recognized is constraint by P(L=n|X). In the training process, convolution network is utilized to extract features from the input image. Then the features are fed into the softmax layer. Backpropagation and stochastic gradient descent is also included in this process. The best architecture for SVHN dataset in this paper is claimed to include eight convolutional hidden layers, one locally connected hidden layer, and two densely connected hidden layers. The number of hidden layers is modified for different dataset in the article.

## 2.2 Key Results of the Original Paper

For single digit in SVHN dataset, an accuracy of 97.84% is achieved. To obtain a result similar to the recognition by humans, 98% accuracy is set as a threshold to filter out the transcriptions below the confidence. The valid transcriptions take up to 95.64% of the total transcripts, which is practical to be added to the map. In addition, they achieve 99.8% accuracy in the reCAPTCHA puzzle, which is a reverse Turing test to distinguish human and machine using the distorted texts.

## 3. Methodology

In this section, we will talk about the data we used and the way we trained the dataset, including the model structure we used and the training process. We will discuss the most important aspects of our strategy to tackle the problem. We first explain what kind of data we use, and how we pre-process the data to be used in our model. Secondly, we will describe the architecture of Neural Network utilized in this study. Finally, we will introduce traversal attack method as a new contribution to the model developed in [1].

## 3.1. Objectives and Technical Challenges

In this project, the first objective is to construct a multi-layer deep convolutional neural network using Tensorflow 1.13 framework and to achieve an accuracy of over 80% for a series of digits. In addition, the second objective is to add adversarial attack using Tensorflow 2.0 framework. During data preprocessing, the font and position of the digits are various. There are also shadow and non-digit components in the images, which might be identified as numbers.

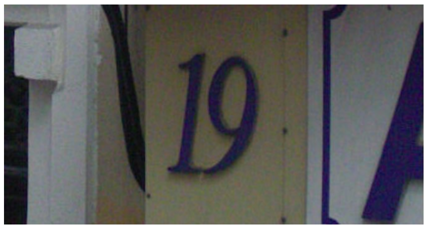
## 3.2. Data-set and pre-processing

The data we used is Street View House Number (SVHN) dataset. SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting [2].

The dataset has two formats, one is the full-number format, the other is the cropped-digit format. We chose the first one to do the training since it's closer to the real-world problem, i.e. recognizing multi-digit numbers as a whole.

In the full-number-format SVHN dataset, there are three zipped files: training data, test data and extra data. As the names may show, training data is used to train models and test data is used to examine models. In the extra data, there are some images that are easier to learn, i.e. they can also be used as training data.

Each zipped file contains images of house numbers and a .mat file. There are two fields in this .mat file. One is the name field, giving the names of images, and the other is the bbox field, describing the labels and the positions of each digit in images.



name: 1.png
bbox: {'label': [1.0, 9.0], 'top': [77.0, 81.0], 'left': [246.0, 323.0], 'height': [219.0, 219.0], 'width': [81.0, 96.0]}

Fig 3.1: An example of the images in SVHN dataset together with the name and bbox fields

Fig3.1 is an example of name and bbox fields together with the corresponding sample.

One thing to notify is that the names of images are stored in ASCII format. In Fig3.1, it is the result after the transformation of ASCII to string to get a better view of the data.

There are some challenges for recognizing this street view multi-digit dataset. One is that the digits can be in various fonts and positions to make the data difficult to learn, another problem is that it is highly possible that the light and shadow or other non-digit factors can be recognized as digits since digits are simple to form. For example, the line between light and shadow can be recognized as digit '1'.

To extract the data, we first get the information in the .mat file mentioned above, i.e. name and bbox fields. For the name field, as mentioned, we did the transformation from ASCII to string. We used this name to find the image we want. For bbox field, as shown in Fig3.1, there are there are 4 lists describing the position of each digit in the image: top, left, height and width. What we want is a cropped version of the original images. Since the origin of an image is at the left top, to extract the digits as a whole, we first generated the 'bottom' and 'right' for each digit by adding up 'top' and 'height', 'left' and 'width' respectively. Then, we used the minimum in 'left' and 'top' lists as the 'left' and 'top' for the whole digits, and the maximum of 'right' and 'bottom' lists as the 'right' and 'bottom' for the whole digits. To be specific, for the image in Fig3.1, after the modification above, the features describing the position of the digits are {'top': 77.0, 'left': 246.0, bottom: 296.0, 'width': 177.0}

In this case, we have the description of where should the cropped images be. Then, we can use the description mentioned above to do the cropping. After the cropping, we resized images into 64×64×3. Fig3.2 shows the result of Fig3.1 after cropping and resizing. Notice that there is another feature -- label in bbox. We will discuss the processing of it later.

## 3.3. Deep Convolutional Neural Network model

We used a CNN with 7 hidden layers (5 convolutional layers and 2 fully connected layers) to classify multi-digit images as Fig3.2, i.e. 64×64×3. Same as in the original paper, we used 6 softmax layers to do the classification (1 for the total number of digits and the other 5 for the value of each digit).

The convolutional layers are for feature extraction of the images. We set the kernel size to be gradually decreasing because the size of the features is decreasing accordingly. Plus, we want to have some wide-range features first, and then get deep into it. The number of features we set is gradually increasing, which is common in CNN, so that we can extract useful features easily.

In a single convolutional layer, we have batch normalization, a conv layer (the name is still convolutional layer, to distinguish from the convolutional layer in a large picture, we call it conv layer here) and a pooling layer.

Batch normalization can help the model learn the data better. The conv layer is for the model to extract simple features of the dataset and the pooling layer is to deal with more complex situations of the images, such as rotation of digits.

At the last convolutional layer, we added a dropout layer to prevent overfitting and try to improve the generalization accuracy.

Now we are going to talk about the modification of the labels. Since we have 6 softmax layers, we modified the structure of labels to a 6-column matrix with the first column represents the number of digits in an image and the rest represents the value of digits. If the digit is 0, the label will be 10. This is because 0 is used to represent None. For example, if the multi-digit in an image is 62, then the 6-column label should be [2, 6, 2, 0, 0, 0]. We then applied one-hot encoding and seperate the 6 columns to fit the model's input shape.

In the training part, we used early stopping, trying to have a better generalization accuracy.

Fig 3.2: The image after cropping and resizing

## 3.4. Adversarial Attack

Initially, we follow the model in [1] and there is no study on the confidence analysis of their model for images with worse quality which are not avoidable on some applications, i.e. someone may take photos with not a good mobile phone or maybe because of speed or space limitation we prefer to save data with much lower quality. To address this point we were interested to attack our model by adversarial data and then re-train the model with these new data [4]. Moreover, recently, it is shown in [5] that non-robust features can be found robustly via information provided by adversarial examples.

Adversarial examples are made by adding some level of noise to the actual image. In this work, we follow the method described by [4] known as Fast Gradient Sign Method. In this approach, the noise kernel function is defined by the model's gradient function. Let's say we have input-target pair $(x, y)$ and the cost function for the NN model is $J(\theta; x, y)$ where $\theta$ is the NN weights. We first train the model to find $\theta$ for the ground truth input-output pairs. Then, we create adversarial examples by the following equation:

$$\bar{x} = x + \varepsilon\, sign(\nabla_x J(\theta; x, y))$$

where $\bar{x}$ is the adversarial example corresponded to $x$ with the same label $y$. Figure (3.3) shows how this equation produces adverisial example.

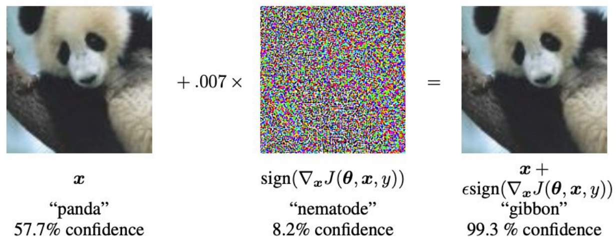


Fig 3.3: An adverisla image constructed with a small noise added to the ground truth image.

Using this method, we introduced an extra hyper parameter that controls the amount of added noise to the original image. After creating enough adversarial data for both training and testing, we will train our base model again and check how the accuracy and generalization could be affected.

## 4. Implementation

Following the objectives mentioned in section 3.1, we implemented a 5 convolutional layers with 2 fully connected layers using Tensorflow 1.13 and 6 softmax layers and then added adversarial attack onto the network using Tensorflow 2.0. Some methods used during training such as early stopping and dropout are implemented using Keras. The techniques utilized in convolutional layer such as max pooling and batch normalization are also based on Keras.

## 4.1. Deep Convolutional Neural Network

The architecture of the CNN we used is as Fig4.1. As described, we have 7 layers (5 convolutional layers and 2 fully connected layers) together with 6 softmax layers.

In the training, we used the data in the following manner: training data, extra data, training data. The reason is that the amount of images in the training data is much smaller than that in extra data but the training data is harder to learn. To try to eliminate the time used when training with the extra data, we first used the training set. Then, we used extra data to make the model have a higher generalization accuracy. When we have used the extra data, it was like transfer learning if we train the model using the training set. In this case, the model can learn these more complex images better.

In the training process, we set the patience of early stopping to be 5, the batch size to be 128 and the epoch number to be 20. Plus, the ratio of training data and validation data is 8 to 2. The optimizer we used was Adam since it usually can have a better performance over other optimizers.

## 4.2. Adversarial Attack

We create adversarial examples, i.e. *attack* step, after training the model with normal data-set. Then, we have access to the gradient of weights which is required by the approach introduced earlier. After creating adversarial examples, we evaluate the performance of our model against these new inputs. At this stage we expect to get a lower accuracy as the model will be confused to classify them. Next, we split these data into adversarial training and test data sets and re-train the same model by feeding the adversarial training data-set, i.e. this step called *defend*. It is noteworthy to indicate that in this framework generator (attack step) and discriminator (defend step) models use the same CNN architectures described in Fig 4.1. The main idea of how to implement the adversarial attack in the context of TensorFlow V2 comes from [6]. The pseudocode is provided in the table below.

```
Algorithm 1 Adversarial data generation
  Input:  Initial image x, Target class y
  Output:  adversarial image $x_{adv}$
  Parameters: Perturbation coefficient $\epsilon$
  while i< batch size of $x_{adv}$
      $x_{adv}[i]$ <-  $x[i] + \epsilon \cdot \mathrm{sign}(\nabla_x J(\theta; x, y))$
  end while
  return $x_{adv}$
```

Table 4.1: Pseudocode of adversarial data generation

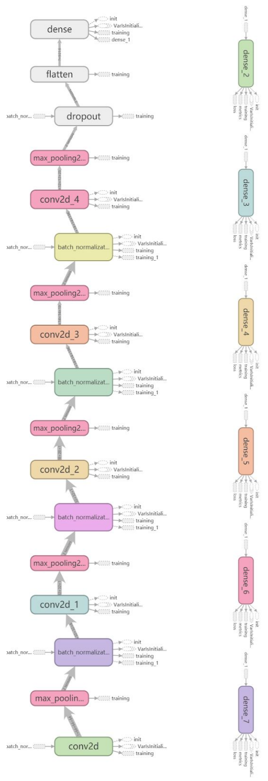


Fig 4.1: CNN structure

## 4.3. Software Design

As is shown in Fig 4.3, the data go through the preprocessing, convolutional layers and softmax layers. In our initial implementation, adversarial attack is not included therefore marked with dash line and the real samples go directly into the network. To examine the robustness of the network structure, adversarial attack is integrated into the preprocessing step. Note that the adversarial training process is adjusted if the output is incorrect.

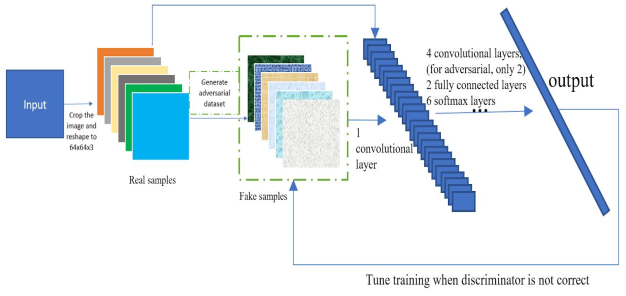

Input
Crop the image and reshape to 64x64x3
Real samples
Generate adversarial dataset
Fake samples
1 convolutional layer
4 convolutional layers (for adversarial, only 2)
2 fully connected layers
6 softmax layers
output
Tune training when discriminator is not correct

Fig 4.3: Dataflow of the multi-digit recognition system

## 5. Results

## 5.1. Project Results

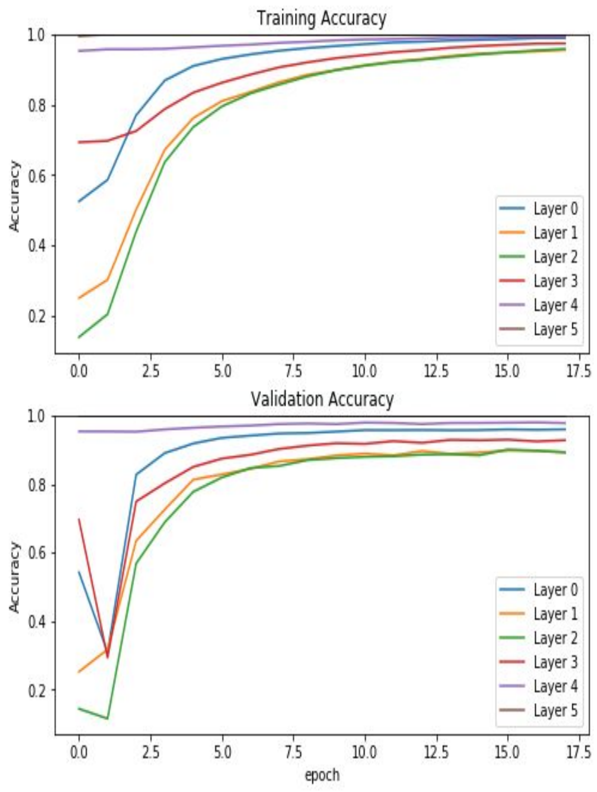
For each softmax, the test accuracy are as the following table.

| Soft max | 0 | 1 | 2 | 3 | 4 | 5 | Total |
|---|---|---|---|---|---|---|---|
| Acc | 0.9797 | 0.9492 | 0.9450 | 0.9795 | 0.9974 | 0.9998 | 0.8974 |

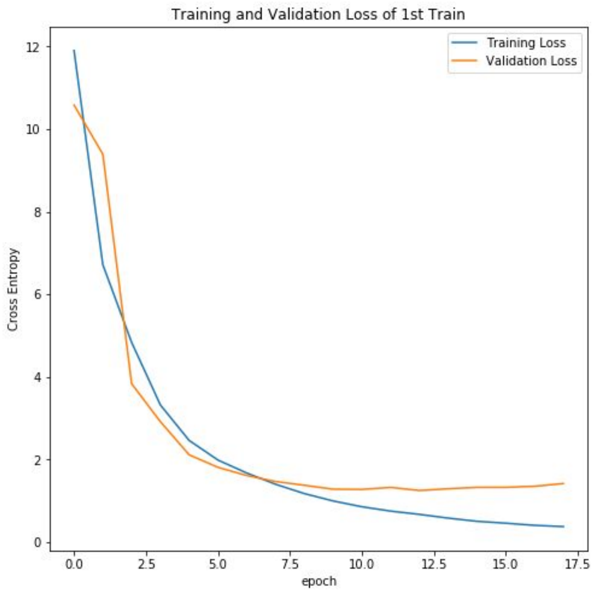
Table 5.1: The accuracy of the main training process

The total accuracy is the accuracy that the model predicts all of the digits successfully, including the number of digits.

As shown in the table, the accuracy for the last two digits are very high. This is because the amount of images containing more than 4 digits is rather small. Hence, it is easier to predict.



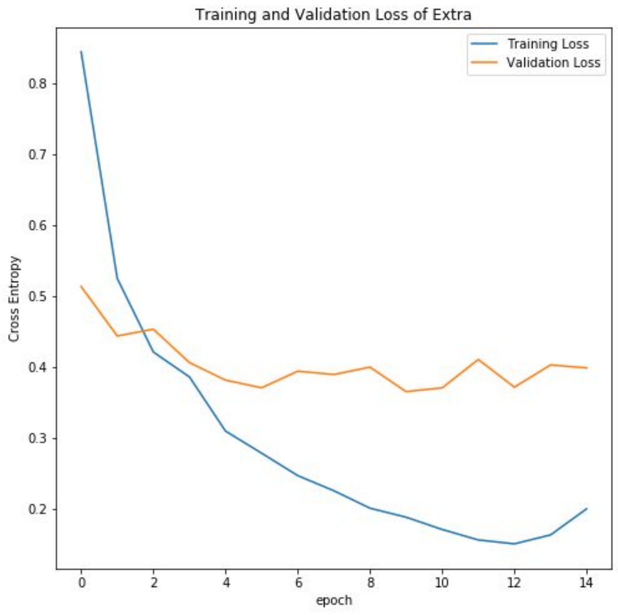
(a)



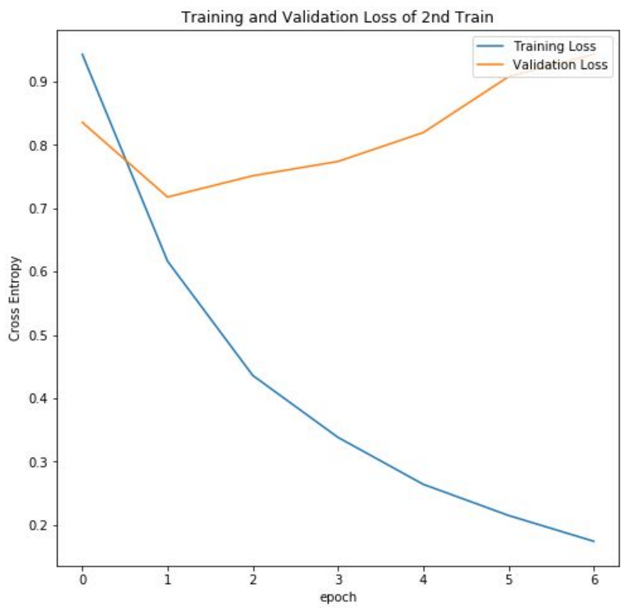
(b)

Fig 5.1: (a) The training/validation accuracy (b) The training/validation loss in total (the summation of the loss in every layer)

Fig5.1 shows the accuracy and loss of the first training part (use training data the first time) and Fig5.2 shows the loss of the second and third part.



(a)



(b)

Fig 5.2: (a) The training and validation accuracy of the second training process (b) The training and validation accuracy of the third training process

As for the run time, for training data, it takes about 1 minute for 1 epoch, and for extra data, it takes about 5.5 minutes for 1 epoch. Hence, the total training time is about 2 hours.

We also tested the performance of the model in different settings. Because training with extra data is time-consuming, the following results are all based only on train data.

| Ratio | 0 | 1 | 2 | 3 | 4 | 5 | Total |
|---|---|---|---|---|---|---|---|
| 0.25 | 0.9600 | 0.8781 | 0.8802 | 0.9480 | 0.9942 | 0.9998 | 0.7654 |
| 0.20 | 0.9474 | 0.8756 | 0.8674 | 0.9382 | 0.9927 | 0.9998 | 0.7511 |
| 0.15 | 0.9571 | 0.8856 | 0.8704 | 0.9473 | 0.9940 | 0.9998 | 0.7648 |
| 0.10 | 0.9574 | 0.8867 | 0.8821 | 0.9526 | 0.9933 | 0.9998 | 0.7762 |

Table 5.2: The accuracy when using different ratio of training set and validation set (with epoch 20 and early stopping patience 5)

| Patience | 0 | 1 | 2 | 3 | 4 | 5 | Total | Stop Ep |
|---|---|---|---|---|---|---|---|---|
| 5 | 0.9474 | 0.8756 | 0.8674 | 0.9382 | 0.9927 | 0.9998 | 0.7511 | 15 |
| 10 | 0.9522 | 0.8834 | 0.8731 | 0.9482 | 0.9934 | 0.9984 | 0.7676 | 20 |
| 15 | 0.9575 | 0.8849 | 0.8682 | 0.9447 | 0.9926 | 0.9998 | 0.7636 | 26 |
| 20 | 0.9547 | 0.8809 | 0.8685 | 0.9502 | 0.9945 | 0.9994 | 0.7684 | 28 |

Table 5.3: The accuracy when using different early stopping patience (with ratio 0.2 and epoch 50)

In Table5.3, Stop Ep means the epoch when the early stopping happens.

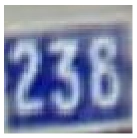
From Table5.2 and Table5.3, we can see that the ratio and patience in this range don't change the performance a lot.

We also tried to preprocess the data first (delete mean and normalize), the result is shown in Table5.4. The accuracy doesn't change a lot.

| Softmax | 0 | 1 | 2 | 3 | 4 | 5 | Total |
|---|---|---|---|---|---|---|---|
| Acc | 0.9574 | 0.8872 | 0.8810 | 0.9500 | 0.9932 | 0.9998 | 0.7768 |

Table 5.1: The accuracy of training with preprocessed data

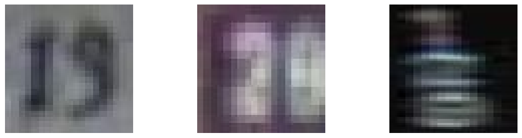
Finally, we take a look at the predictions of the model. Fig5.3 shows an example of successfully predicted image. In this figure, '3' at the beginning of the prediction represents the number of digits.



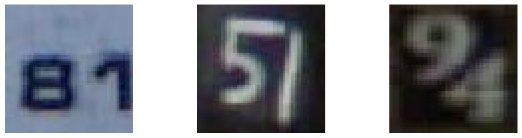
Prediction: [3, 2, 3, 8, 0, 0]
Ground Truth: [2, 3, 8]

Fig 5.3: An example of successfully predicted image

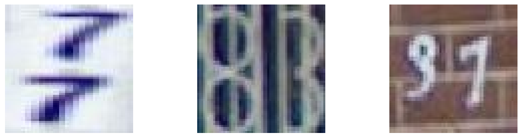
In Fig5.4, there are some images that are not successfully predicted.



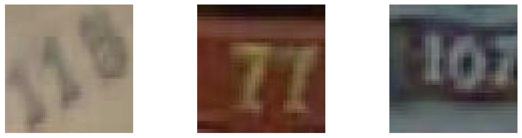
Pred: [2, 1, 9, 0, 0, 0] [2, 2, 9, 0, 0, 0] [3, 2, 10, 3, 0, 0]
Truth: [2, 1, 3] [2, 2, 8] [3, 1, 2, 9]
(a)



[2, 6, 7, 0, 0, 0] [2, 5, 7, 0, 0, 0] [1, 2, 0, 0, 0, 0]
[2, 8, 1] [2, 5, 1] [2, 9, 4]



[1, 4, 0, 0, 0, 0] [2, 8, 8, 0, 0, 0] [2, 3, 4, 0, 0, 0]
[2, 7, 7] [2, 8, 3] [2, 3, 7]
(b)



[3, 1, 1, 8, 0, 0] [2, 7, 7, 0, 0, 0] [4, 1, 8, 10, 0, 0]
[2, 1, 5] [2, 4, 7] [3, 7, 1, 10]
(c)

Fig 5.4: Some examples that are wrongly predicted

In Fig5.4, (a) are some images that are also hard to predict for humans. These images are either in low quality or very small (making it unclear after cropping and resizing). (b) are some images that are easy for humans to predict. From these images, we can know that the CNN model may not be good at predicting vertical digits (94) and it can wrongly see the lines caused by the background as the digits (83 to 88, 37 to 34). Plus, what is interesting is that it often predict 1 to be 7 (81 to 67, 51 to 57), caused by other digits or the fonts. According to our other results, such false predictions can happen also between 6, 8 and 9, etc. One reason for this kind of false predictions may be that the training data has some unusual digits (fonts etc), making the model overfit to these unusual images. (c) are some interesting images that the label may be a bit confusing.

There may be some images that humans cannot easily predict but the CNN model can, but due to the limitation of time, we didn't really find such images.

## 5.2. Adversarial effect

In this section, we study the robustness of our CNN model by evaluating its performance on adversarial attacks. The generator and discriminator has the same architecture as mentioned before. The generator model is trained with regular data-set, excluding the extra data-set. Then we update the weights of the CNN generator model using 6000 randomly generated adversarial training data. The original source of these data are chosen from the training data-set. We also generate 2000 adversal data from our base test data. We study the effect of adversarial hyperparameter $\varepsilon = 0.05,\ 0.1,\ 0.2$ with three different levels of noise. For example, Fig5.5 shows advertised images generated by different values of $\varepsilon$. Obviously, higher value leads to more noise and consequently it makes it harder for the model to predict the right label. But, the other side is that it can help the model to learn some other hidden features which is not easy in a normal way.

We summarize the accuracy of the discriminator model against 2000 adversarial test-data which are not observed by neither generator and discriminator during the training process. Each row is corresponded to a level of $\varepsilon$ and when this parameter is zero it means we do not have any adversarial data-point or defending. Generally, increasing $\varepsilon$ leads to lower accuracy as expected since the images will be more noisy. It is noteworthy to indicate that even with noisy data the model is sufficiently robust to have relatively similar accuracy. Also this indicates that our model has a reliable generalization. This aspect is very important specifically for applications where the data points are sampled from different sources with different qualities. For example, in our application, these images are obtained mainly from people cameras or phones with considerably various qualities altered by device technology, weather, a person's skill, etc.

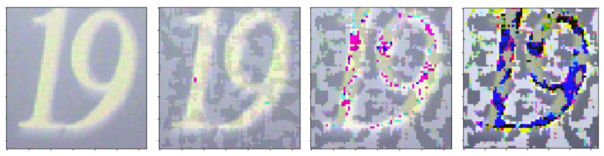


Fig 5.5: from left to right $\varepsilon = 0,\ 0.05,\ 0.1,\ 0.2$ .

| Ratio | 0 | 1 | 2 | 3 | 4 | 5 | Total |
|---|---|---|---|---|---|---|---|
| 0.0 | 0.9445 | 0.8515 | 0.8325 | 0.9255 | 0.989 | 1.0 | 0.8325 |
| 0.05 | 0.9388 | 0.850 | 0.8305 | 0.9218 | 0.9914 | 0.9998 | 0.8305 |
| 0.1 | 0.9404 | 0.8490 | 0.8357 | 0.9259 | 0.9924 | 0.9998 | 0.8357 |
| 0.2 | 0.9264 | 0.8402 | 0.8217 | 0.9165 | 0.9920 | 0.9998 | 0.8217 |

Table 5.4: the accuracy of discriminator against 2000 adversarial test data samples for the original model, $\varepsilon = 0$ , and three levels of $\varepsilon = 0.05,\ 0.1,\ 0.2$ .

## 5.3. Comparison of Results

Compared with the result in the original paper, our total accuracy is about 10% lower for the recognition of a series of digits.

One reason why our accuracy is lower may be that we didn't train with many epochs. Since we used early stopping, patience may be too low so that we stopped the training at the time that the model was trying to get rid of the local minima (the error increases), i.e. we didn't wait for enough time for the model to get to a better point.

Another reason may be that the structure of models are different. In the original paper, they used CNN models with more layers and the got higher accuracy when they increased the number of layers.

## 6. Conclusion

In conclusion, in this project we constructed a multi-layer convolutional neural network to recognize a series of digits. The accuracy can reach 89% for a series of digits. In general, the objective is accomplished. The adversarial attack is added to increase the robustness of the network when trained with "train" dataset provided. In the future, adversarial "extra" dataset can be used to train the network to see how a larger amount of data will influence the adversarial training.

## Acknowledgement

This work is based on Google's research of Multi-digit number recognition from street view imagery using deep convolutional neural networks.

We also referred to the work of [7] for the extraction of bbox (especially the bbox_helper and get_bbox functions) and [8] for building our model structure. For the adversarial part we referred to [6]. Plus, we referred to the assignments to build our own functions to download the datasets.

The codes about this report are in [9].

## 7. References

[1] Goodfellow, J. Ian, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. "Multi-digit number recognition from street view imagery using deep convolutional neural networks." *arXiv preprint arXiv:1312.6082* (2013).
[2] D. Cireşan, U. Meier, L. Gambardella, and J. Schmidhuber. "Deep, Big, Simple Neural Nets for Handwritten Digit Recognition." Neural Computation 2010 22:12, 3207-3220
[3] http://ufldl.stanford.edu/housenumbers/
[4] Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572* (2014).
[5] Ilyas, Andrew, et al. "Adversarial examples are not bugs, they are features." *arXiv preprint arXiv:1905.02175* (2019).
[6] https://medium.com/analytics-vidhya/implementing-adversarial-attacks-and-defenses-in-keras-tensorflow-2-0-cab6120c5715
[7] https://github.com/bdiesel/tensorflow-svhn/blob/mast-er/digit_struct.py
[8] https://juejin.im/post/5c04e342f265da6165015391
[9] https://github.com/cu-zk-courses-org/e4040-2019fall -project-dcnn-jz3121-wz2492-bb2969

## 8. Appendix

### 8.1 Individual student contributions in fractions - table

| | jz3121 | bb2969 | wz2492 |
|---|---|---|---|
| Last Name | Zhou | Bahmani | Zhang |
| Fraction of (useful) total contribution | 1/3 | 1/3 | 1/3 |
| What I did 1 | Wrote codes about data download and | Implemented a modeler class in python3 | Did a literature review on the method of digit |

| | | | |
|---|---|---|---|
| | extraction (get_svhn.py, data_extractor.py) and built the CNN model (dense.py) | with the main functionality of adversarial training (generator-discriminator) based on the keras and TF2 as the backend | recognition including support vector machine, multi-layer perceptron |
| What I did 2 | Performed training and analysis of recognition (recognition_pipeline.ipynb, analysis.ipynb, recognition_analysis_nonextra.ipynb) | Performed adversarial image generation and adversarial training. Interpreted the results. code contributions are listed as “adversarial_main.ipynb”, “adversarial_tools.py”, and “adversarial_modeler.py”. | Verified the code of data download and extraction; test the functionality of CNN model including training and recognition |
| What I did 3 | Wrote report: Chapter 3.2, 3.3, 4.1, 5.1, 5.3 | Wrote report: Chapter 3, 3.4, 4.2, 5.2 | Wrote report: abstract, Chapter 1, 2, 3.1, 4, 4.3, 6 |