End-to-End CNN California License Plate Recognition

Haoran Cheng, ECE Department, UCLA, and Guanli Qiao, ECE Department UCLA
Jiabao Zhai, ECE Department, UCLA

***Abstract*— This paper focuses on providing a method to implement car plate recognition on various platforms. This technique can be applied in many circumstances such as the lifting rod in a parking lot. Our system work by passing an image file of a license plate into the CNN network and outputting 7 characters, these 7 characters will be determined by the respective dense layer with the highest possibility predicted by the End-to-End CNN. If the platform allows, the prediction accuracy for each prediction can reach up to 90%.**

I. Introduction

This paper is going to introduce how to implement license plate recognition on a hardware platform with an End-to-End CNN model trained in the cloud. The implementation will contain two parts. One is the End-to-End CNN model train on the cloud. This part will need a large number of license plates as a data set. The training will be done by python using the CNN training library in TensorFlow. Another part is to implement the CNN model into a platform that can process the recognition by digitizing the license plate image. The parameter for each module should be obtained from the pre-trained model on the cloud. The platform will use these parameters to construct a CNN model that has the same capability as the one in the cloud. This step only involves mathematical functions to the image matrix.

A. History

In today's technical background, license plate recognition systems can be realized in a variety of ways and these technologies are very mature. For example, using Sobal edge detection and projection to locate the character on the license plate[1]. And then pass it through the neural network one by one. The advantage of this technique is that it doesn't need to consider the format of the plate. Instead of using image projection, the End-to-End CNN model involves less image digital processing and has a low threshold of use, but there are requirements for hardware.

B. Global Constraints

There are many factors that will affect the accuracy of recognition. The most basic factor is the number of samples. In this system, there are 36 possible outputs in total. In general, to train a CNN model, there should be at least 70 samples for each possible output. Otherwise, the accuracy of recognition can not be improved even with the best CNN architecture.

The revolution of the data set is also a major issue. To simplify our training, all the images must crop into a set revolution. It is 31*104 pixels for our project.

Hardware's capability plays a sufficient role when the CNN model is implemented. By getting the same parameter from the CNN training, the model on the hardware platform should perform the same as the model in training. During this process, the calculation on the hardware should keep the same accuracy as the training. But in general, when the network is implemented on a platform, the accuracy is decreased especially when the model contains a large number of parameters. The hardware might not get the exact same number as the model on the cloud. The model on the cloud is based on a better hardware background. There are many strategies that can help to resin the performance. For example, instead of using

float, using double to define the variable on the hardware is a good way to reduce the loss of performance.

II. Motivation

The idea of End-to-End CNN in projects can be widely used not only to recognize the license plate. It can be applied to the auto-drive systems which need to track multiple targets.

The license plate recognition project can be regarded as a demo, means to explore the potential of the End-to-End CNN model.

III. Approach

The project will first train an End-to-End CNN network on a cloud platform like Google Colab, the network should require up to 95% accuracy on the image that doesn't contain in training data set. Then use the parameter of each layer to construct the network on the hardware with mathematical functions to the image matrix.

A. Team organization

The project can be divided into two parts. One is to train a network on the cloud, one is the implementation. It is like Fig.1

Guanli and Jiabao focus on the training and Haoran implements the model and is responsible to do the research, simulation, and testing of necessary techniques.

B. Plan

For the first two weeks, we plan to watch some videos about the application of CNN on license plate recognition. We found that most of the mature systems use digital image processing to locate and crop out the character from a license plate image. Then by feeding the cropped character image into the CNN model, it will gain a prediction of what this character is. We will have a similar flow with that but the image goes into the CNN directly and outputs the prediction like Fig.2. In week 1, we also start to import the image from the local computer to Google Colab and test the BMP file reader on the hardware.

By the end of week 2, we decided to use an End-to-End CNN for our project like Fig.3. It is built by the reference of Captcha recognition[2].

In our plan, we should be able to start our training and finished the construction of our CNN frame by week 3. From week 3 to week 7, we will focus on the improvement of the CNN model. Since precision is lost when CNN moves from cloud to hardware, we have to improve our accuracy as high as possible. As the project moves forward, we successfully implement the model on the hardware in week 8. For the rest of the time, we are focusing on the optimization of our system.

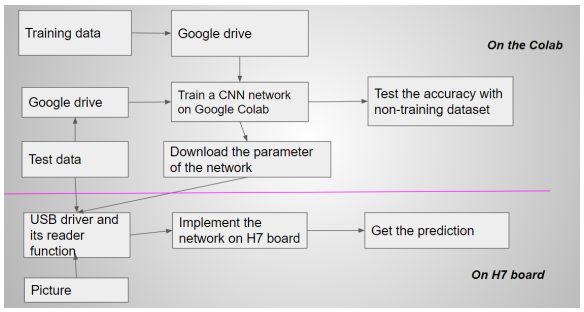


Fig.1: The block diagram of the project

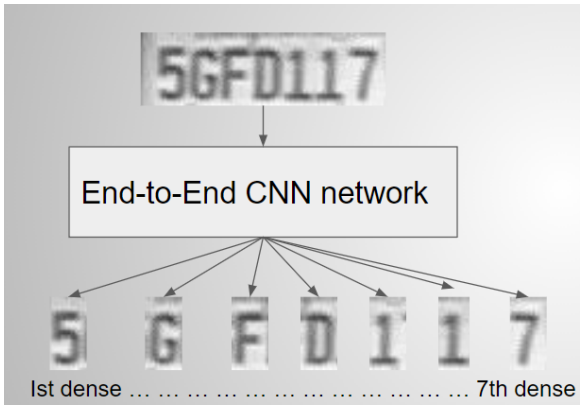


Fig.2: example for our input and output.The End-to-End CNN will do 7 times prediction on these 7 positions on the plate and have a possibility of what is it for each dense layer. We label 0-9 as number 0-9 and character 'A'-'Z' as number 10-35.

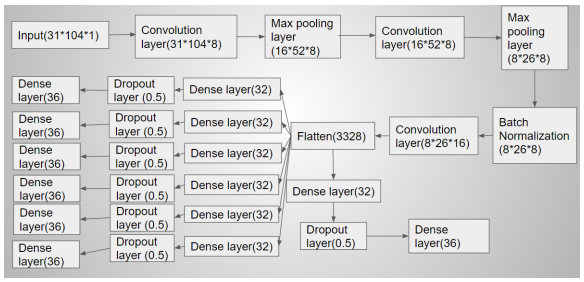


Fig.3: Structure of the End-to-End CNN

C.Standard

Since our CNN training is based on the data set that is already converted in grayscale base on the conversion formula:

$$G.S. = 0.2989\ R + 0.5870\ G + 0.1140\ B$$

The BMP file that we use to train only contains one channel, respected to BMP file contain 3 channels in general. So we only use one channel for training. The data set is shown in Fig.4.

Also, we assume that the input should be converted in grayscale. This is the reason why we randomly take out 20 of the license plate from our data set to test our demo. We are not going to use the 20 images to train our network.

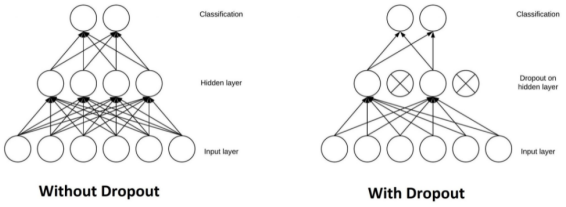
Fig.4: An example of our data.

D.Theory

The improvement of our CNN network is a big part of our project. We have several strategies.

Adding a dropout layer between the full connection layer and result dense layer is the most efficient way for our project. The dropout layer can optimize the training by disabling some of the neurons of the full connection layer. The principle of the dropout layer shows in Fig.5.

Dropout layer:



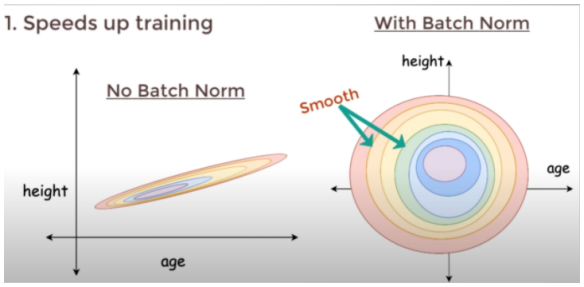
https://www.baeldung.com/cs/ml-relu-dropout-layers

Fig.5: Principle of dropout layer.

When the model is training, the dropout layer enforces the rest of the neuron to make the decision with its respective weight since another half of the neuron is disabled. This method can increase the accuracy of the whole model because when we use the model to make a prediction, all the neurons will be used for that prediction.

The second method we use is called batch normalization, Fig.5. This technique extends the range of the layer feature so that the training doesn't get overshot when the model adjusts its weight to adapt to the data.

Batch normalization:

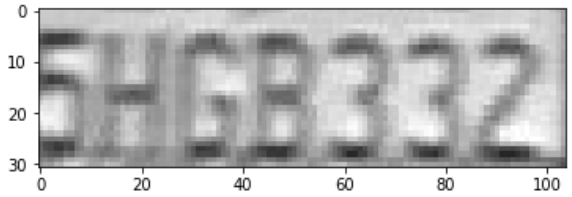


https://www.youtube.com/watch?v=DtEq44FTPM4

Fig.5: After the batch normalization, the "selectable" range is larger for the training.

Then we do a data augmentation to extend our data set since the number of the English character's samples is less than 70. We rotate the range of -8 degrees to 8 degrees from the original data randomly to get the new data. We also zoom out the original data in a ratio randomly down to 0.8 to get another data set. By doing the data augmentation, like Fig.6, we extend our data set from 721 samples to 2163 samples. This strategy led to a promotion of recognition accuracy from 70% to 93% for the English character.

Data augmentation:



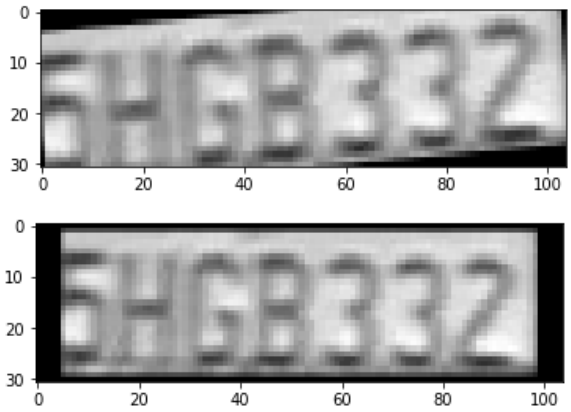


Fig.6: From top to the bottom are the original image, rotated image, and the zoom-out image respectively.

E. Software/Hardware

This project is developed for the H7 board with its development software STM32CubeIDE. It is based on the C language. The frame of the CNN network is based on a mini-project provided by Professor D. M. Briggs and Jiawei Zhang, Dezhan TU, the teaching assistant of UCLA ECE113DB on 2022 Fall Quarter.

In our project, the image of the license plate will be put into the USB driver and connected to the H7 board like Fig.7.

Fig.7: The hardware for this projection.

This project required the optimization skill for C language to do edge computing because of the large number of parameters. The memory of the hardware is limited.

F. Produce

1. Load the data to the training platform

To train a CNN network, we need to load the data to the model with each image matched to a correct label with the license plate character. The license plate contains 7 characters and we used a TXT file to store the label. Each raw represent a label for a license plate like Fig.8. The image is named in a form of 'plate (X)', where X is the Xth license plate(like Fig.4). Once we load the data on the cloud, we are ready to construct the End-to-End CNN network.

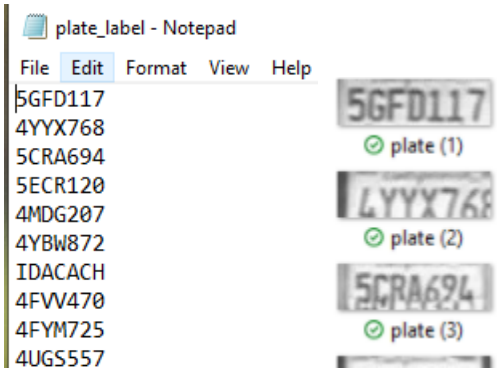


Fig.8: the label for the car plate.

2. Construct an End-to-End CNN

We used Google Colab with Tensorflow to develop the End-to-End CNN. Our model contains 3 convolution layers, 3 max-pooling layers, 1 batch normalization layer, 7 full-connection layers, 7 dropout layers, and 7 result dense layers(like Fig.3). The first convolution layer has 8 channels, the second one has 8 channels and the third one has 16 channels. The batch normalization layer is added between the third convolution layer and the third max-pooling layer. Then, there are 7 full-connection layers being connected with the flatten layer in parallel. Each full-connection layer has 32 neurons. For the dropout layer, we choose 0.5 as our dropout rate which means half of the neurons on the previous layer will not participate in prediction. Finally, it is the result dense layer, the output from this channel is a possibility. It has 36 channels. The output from the Xth channel means it has that possibility to be X. For example, if the 5th channel is 0.7, it means that the model thinks there is a 70% chance that it will be a 5. We use 10 to represent

‘A’, 35 to represent ‘Z’, etc. On the result dense layer, we added a regularizer to stabilize our training. The rate of the regularizer is 0.07. It helps to prevent overshoot when the model is training. We will explain it later.

After all the above step is done, we set up a validation split with a rate of 0.1. That means there will be 1/10 of our samples will be kicked out.Instead of using these data to feed the model, we use them to test our model and return an accuracy every time the model is trained.

3. Check the training result

We plot the Loss vs Epoch so that we can see if our model is overshooting. When we did, we notice that there is a peak before the loss of train and validation converge. It looks like Fig.9.

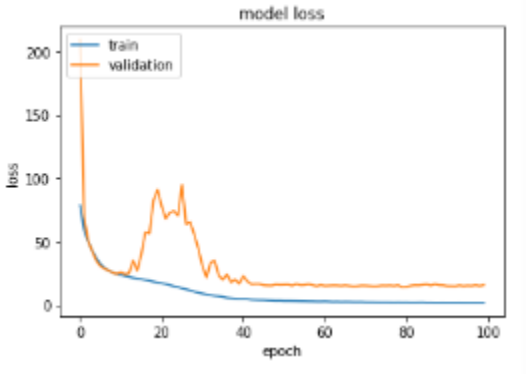


Fig.9: The high peak implies our model is unstable.

We use a regularizer to stable it. It can add punishment to the weights when the model is fitting the data.The result shows in Fig.10.

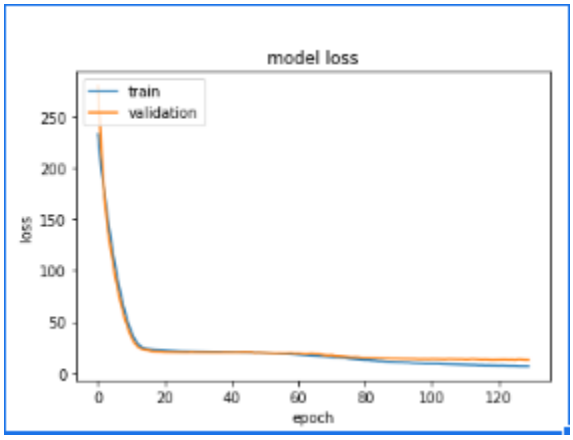


Fig.10: If the curve of the train surpasses above the curve of validation, that means the model is overfitting the data.

In order to obtain the best performance, we can adjust the number of channels for each layer. We enumerate every possible combination of structure and write down the accuracy, like Fig.11. In general, the number of channels should be the power of 2.

Dense\Structure	8-8-8-64
1	0.6081
2	0.1892
3	0.1486
4	0.1892
5	0.527
6	0.4054
7	0.3649

Fig.11: The accuracy of each dense layer for the current structure. 8-8-8-64 means all the convolution layers have 8 channels and the full-connection layer has 64 neurons.

The architecture of the model can be changed to fit the current project. The reason why we use the 8-8-16-32 structure(like Fig.3) is that we need to implement the model on the H7. The requirement for the implementation is to reduce the number of parameters. A complex model of course will have better performance but the number of parameters can be too large to implement. So it is a trade-off between the performance and the implementation. The 8-8-16-32 structure is the best solution we can find currently. Once we find the best architecture, we can move forward to implement it.

4. Implement the model,

First of all, we need to download the parameters from the model on the cloud. All the convolution layer, batch normalization layer, full-connection layer, and result dense layer has their own parameter. They will be used when implement.

To implement the model, we need to know what do these layers actually do on a mathematical level. The Relu Function, loss function, convolution layer, max-pooling

layer,full-connection layer, and result dense can refer to the article ‘Convolutional neural networks: an overview and application in radiology’[3]. For the batch normalization, that network will output 4 categories of parameters; mean, variance, gamma, beta, we use that parameter by plugging in the formula of Fig.12.We use a USB driver to stored the parameter and connected it to the H7.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

https://www.youtube.com/watch?v=DtEq44FTPM4

Fig.12: In this formula, x is the output from the last layer, and y will be the input for the next layer.

To correct our implementation, we print out every feature of each layer on the cloud’s model and feed in the same input to our H7’s model(Fig.13 and Fig.14). By comparing these two, we can ensure that we implement them correctly.

```
/////conv3_after_batch////////////////////////////////////////////////////////////
-0.159436821937561 -0.729459464550018 -0.191784173250198 -0.505000233650208 -0.430823475122452
0.319702982902527 -0.586118161678314 -0.592487335205078 -0.205240994691849 -0.159436821937561 -
0.963706791400909 -0.201601892709732 -0.188286066055298 -0.471723437309265 -0.319702982902527 -
0.430823475122452 -0.653694450855255 -0.495556056499481 -0.473238646984100 4.370310306549072 -0
0.159436821937561 -0.729459464550018 -0.191784173250198 -0.505000233650208 -0.430823475122452 -
0.319702982902527 -0.586118161678314 -0.592487335205078 -0.205240994691849
/////pool3////////////////////////////////////////////////////////////////////
-0.159436821937561 -0.729459464550018 -0.191784173250198 1.821881294250488 -0.430823475122452 1
0.319702982902527 -0.586118161678314 -0.592487335205078 -0.205240994691849 -0.159436821937561 0
4.370310306549072 -0.201601892709732 -0.188286066055298 0.084958940744400 -0.319702982902527 2.
```

Fig.13: feature from H7 with the same feed-in(for example).

```
[[[[-1.54415086e-01 -7.29193568e-01 -1.87904269e-01 -5.04629672e-01
    -4.30215120e-01  1.19710135e+00 -4.95342791e-01 -4.73021597e-01
    -9.63464022e-01 -1.98342264e-01 -1.85499847e-01 -4.71426010e-01
    -3.18698794e-01 -5.85847318e-01 -5.92292428e-01 -2.02854410e-01]
```

Fig.14: feature from the cloud with the same feed-in(for example).

Once the implementation is done we can test the model on the hardware to see if it can work.

5. Testing

We use a USB driver to stored the image and connected it to the H7. Then change the name of the BMP file handler to the name of the goal image. The example for the output result is shown in Fig.15.

```
The third dense//////////////////////////////////
prediction of number 0 is 0.000771425955463
prediction of number 1 is 0.000763474323321
prediction of number 2 is 0.000685995270032
prediction of number 3 is 0.000985467340797
prediction of number 4 is 0.000939464196563
prediction of number 5 is 0.000662823324092
prediction of number 6 is 0.000627943023574
prediction of number 7 is 0.000489839818329
prediction of number 8 is 0.000752907071728
prediction of number 9 is 0.000774314510636
prediction of number 10 is 0.002007392467931
prediction of number 11 is 0.001283761695959
prediction of number 12 is 0.001769162481651
prediction of number 13 is 0.001411681063473
prediction of number 14 is 0.005007790867239
prediction of number 15 is 0.863825321197510
prediction of number 16 is 0.002133933128789
prediction of number 17 is 0.001097376109101
prediction of number 18 is 0.009650384075940
prediction of number 19 is 0.005878288764507
prediction of number 20 is 0.004943945910782
prediction of number 21 is 0.015441976487637
prediction of number 22 is 0.001510921400040
prediction of number 23 is 0.001202451530844
prediction of number 24 is 0.001821197336540
prediction of number 25 is 0.001837892574258
prediction of number 26 is 0.001143190194853
prediction of number 27 is 0.002114646602422
prediction of number 28 is 0.008603001944721
prediction of number 29 is 0.012127544730902
prediction of number 30 is 0.001688710646704
prediction of number 31 is 0.002498421119526
prediction of number 32 is 0.035434130579233
prediction of number 33 is 0.001080296700820
prediction of number 34 is 0.002668233821169
prediction of number 35 is 0.004364558961242
```

Fig.15: The prediction for the third character of the image. We take the one with the highest possible, which is 15, with respect to the English character ‘F’.

The model will continue to run until all 7 dense is predicted.

IV. Results

The project is tested on 20 images of license plates that are satisfied with the standard quality.

A. Test result

The chart below shows the correct rate for each character on the license plate.

	Correct rate
1st character	0.85

2nd character	0.5
3rd character	0.45
4th character	0.5
5th character	0.65
6th character	0.70
7th character	0.75

B. Discussion of the result

The model on the plate performs the same as we expected. Because of the trade-off between the performance and parameter, we expected the model has an accuracy of around 60% on average. When we use the same samples to test the model on the cloud, it gives the same prediction. That means our model is working as we expected.

The model is sensitive to the first character because the first character contains fewer possible results. The correct rate for the 3rd, 4th and 5th character is lower because it has 26 possible results when most of the license plate samples are in the California format.

C. Potential of this project

If the hardware allows, we can further implement the model with a 32-64-64-128 structure, which has a 95% correct rate on average for each character. It contains 300000+parameters.

Through the implementation, we gained a new appreciation for the gap between hardware and cloud when I implement the CNN on the H7. we print out the feature of each layer from H7 and cloud when we feed in the same input. we find there are slight differences between H7 and the cloud. These numbers are equal when they were rounded in 2nd decimal point. In general, the slight difference should not matter. But since there are huge numbers of parameters being involved with this system, the difference is sufficient. Especially the dense layer did a summation of 26656 numbers which is going to amplify this difference.(Fig.16 and Fig.17)

Fig.16: The feature on the hardware

Fig.17: The feature on the cloud

The result comes closer to the feature on the Colab since I change the data type from float to double on my last layer, which will retain higher accuracy. But we can implement with a double for the whole system because double need more space. So it is a trade-off between accuracy and implementation.

V.Referance

[1]C. Lee, ECE Department, UCLA, and E. Xie, “Automatic Plate Number Recognition for Parking System”, ECE Department UCLA, 2018

[2]Manvi Goel, Nandini Sidana, Siddharthsamber, “CAPTCHA Recognition using Convolutional Neural Network”,2020

[3]Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629 (2018). https://doi.org/10.1007/s13244-018-0639-9