CSCE 636 NEURAL NETWORK PROJECT REPORT

July 14, 2019

Prateek Shroff
Texas A&M University
Department of Computer Science and Engineering

Contents

1 Topic

The project aims to build an end-to-end deep learning based license plate detection and its number recognition system from images. The model is combined two stage (Detection + Recognition) network trained end-to-end on a GPU enabled server using Chinese City Parking Dataset (CCPD).

2 Dataset

The neural model is trained using the CCPD. The dataset contains around 300K images with different variations in form of tilt, rotation and taken in challenging environment conditions like snow, haze etc. Since, the image size varies in the dataset, the images are resized to 480×480. Further, image preprocessing is done before feeding images to the model. The train-test split is 150k and 33k images respectively.

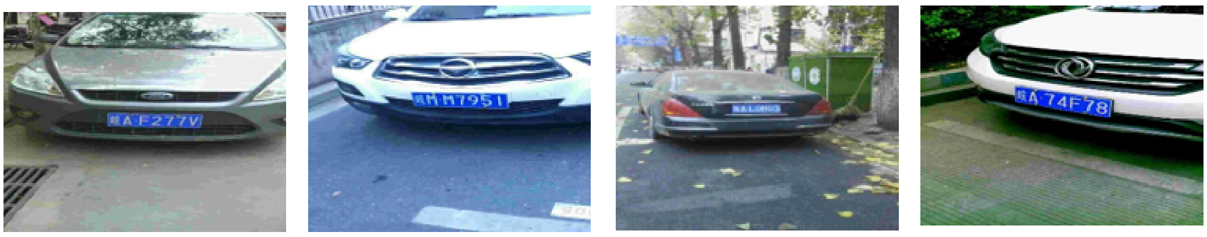


Figure 1: Sample Images present in CCPD.

3 Architecture and Methodology

3.1 Architecture

The architecture of the model can be segregated into two independent modules: ***Detection*** and ***Recognition***. The architecture is inspired by an existing paper by Xu [1] and have taken code reference from the GitHub here. To increase its versatility, I have made two major changes in the overall architecture of the model. First, in the recognition module, I have changed linear layers to bidirectional LSTMs. Second, I have replaced the cross-entropy loss with the more specific sequential loss Connectionist

Temporal Classification (CTC) [2]. In the following subsections, I have described in detailed different modules, CTC loss and highlighted the changes applied over the baseline model.

3.1.1 Detection

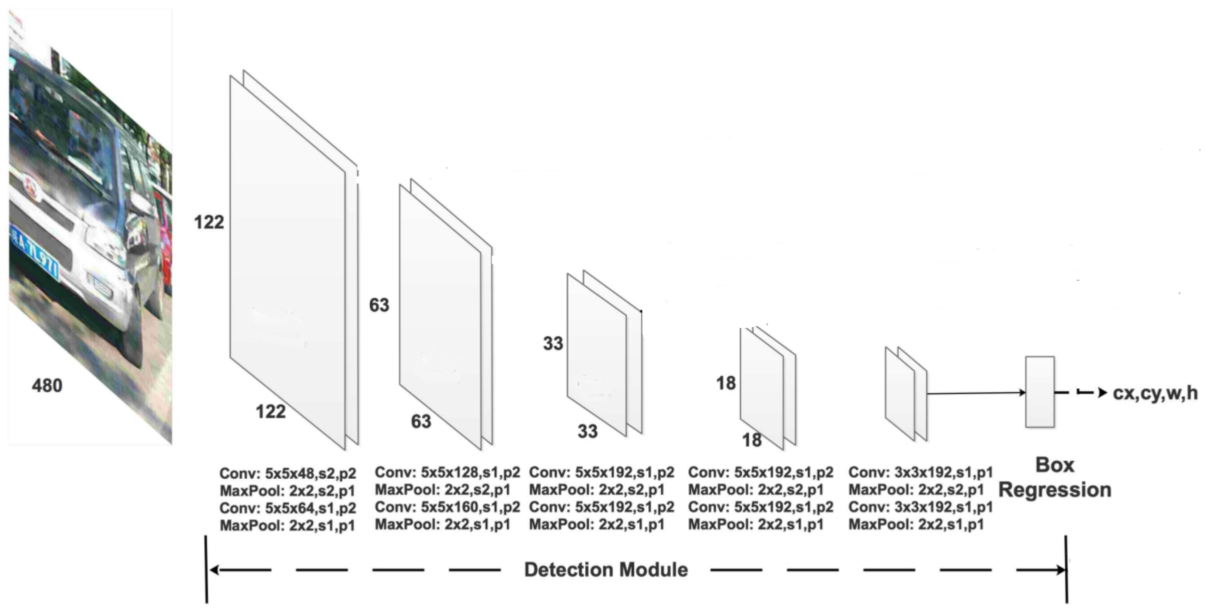


Figure 2: Detection Module

Given the front or the back view of a car, we need to detect the license plate in the whole image. Specifically, we need to have the coordinates of the bounding box surrounding the licence plate of the car. In order to fully specify the location of box in an image, the detection module outputs four variables (cx,cy,w,h). Let, *(bx, by)* represents the x-coordinate and y-coordinate of top-left most corner of bounding box respectively. Also, *(bh,bw)* represents the height and weight of bounding box. These values are scaled to height *(H)* and width *(W)* of input image. So, the bounding box (cx,cy,w,h) satisfies:

$$cx = \frac{bx}{W} \quad cy = \frac{by}{H} \quad w = \frac{bw}{W} \quad h = \frac{bh}{H}, \quad 0 < cx, cy, w, h < 1 \tag{1}$$

Essentially, the detection module is modelled as a regression problem. It consists of 10 convolutional neural networks stacked sequentially where each layer of convolution is followed by a batch normalization, ReLU activation function, max pooling, and a drop out layer as shown in figure 2. The detailed dimensions of each layer is given in table 2. These layers extract features from the given image. As the number of layers increases, the number of channels increases, and the size of the feature maps decreases progressively. The aim of using 10 layers is to gradually downsize the input dimension (480) to lower dimension. This helps to propagate the rich information present in high resolution of the image without loss of much information.

Finally, at the end of 10^{th} convolutional block, the output feature map is flatten and fed to three linear layers which are collectively called "Box Regression" in the figure 2to give center coordinates, height, and width of the bounding box. This is the detection module of baseline and there has been **no change** done in detection module in my updated model.

3.1.2 Recognition

The second module is the *Recognition* module. The aim of this module is to recognize and extract the licence number from the licence plate detected by detection module. The recognition module underwent significant changes compared to the baseline module. 3 shows the part of module which uses the output of detection module to prepare/extract features as input to subsequent layers in recognition module.

As stated in previous sections, the detection module outputs the center, the height, and the width of of bounding box for licence plate denoted by (cx, cy, w, h). Since, the area of license plate is relatively small compare to whole image, extracting features from the lower convolutional layers provide fine details of the object. This helps in overall improvement in the semantic characteristics of the license plate. This is concept is demonstrated in works like [3]. For the feature map of size m×n with p channels, the recognition module extracts feature maps in the bounding box region of size (m*h)×(n*w) with p channels. Concretely, referring to figure 3, the *dotted lines* denotes using the values (cx,cy,w,h) to find the bounding box in lower layers 1^{st}, 3^{rd}, and 5^{rd}. So, The sizes of extracted feature maps are (122*h) × (122*w) × 64, (63*h) × (63*w) × 160 and , (33*h) × (33*w) × 192, where 122 is the dimension of 1^{st} layer, 63 dimension of 3^{rd} layer, and 33 dimension of 5^{rd} layer. The *solid lines* denotes the extracted features from lower layers passed to ROI pooling Layer. Please note here, the spatial dimension of each of the extracted layers is different.

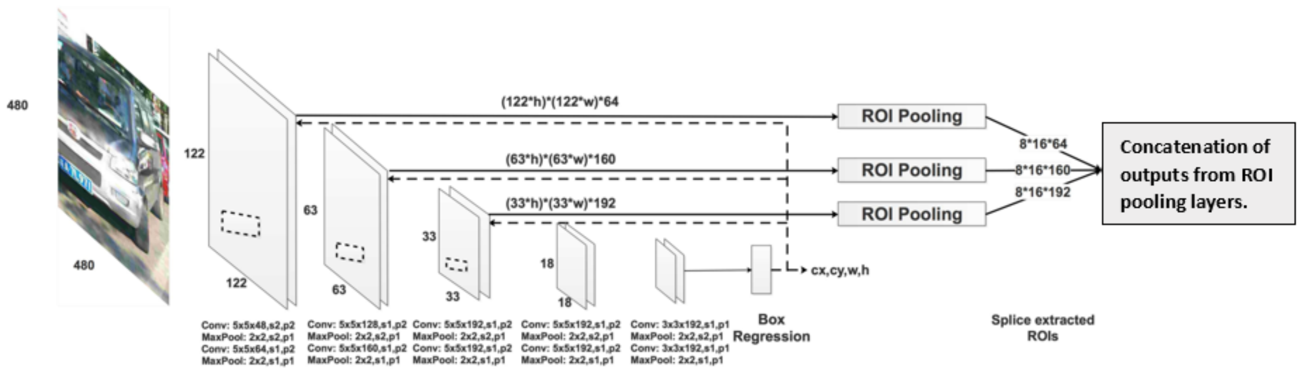
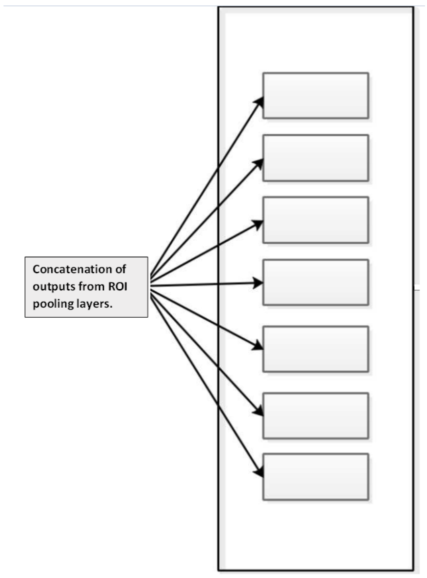


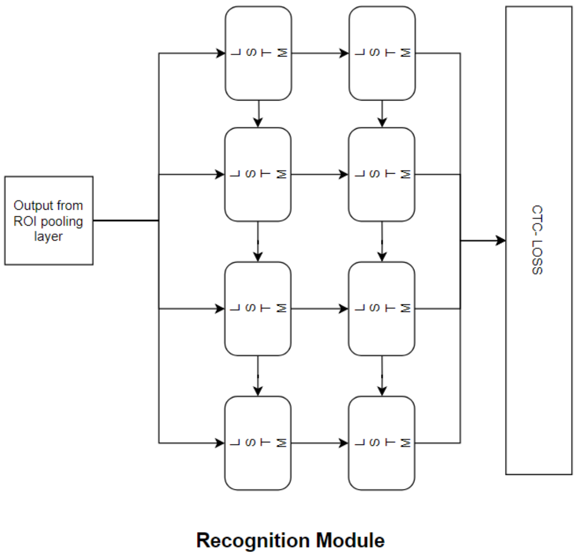
Figure 3: Recognition Module

ROI stands for Region Of Interest. Designed in [4], it was used to convert features maps of different object having different spatial size to fixed spatial size. Here, we get feature maps from three different layers hence those are three different spatial dimension (122,63,33). So, ROI is exploited here to convert each feature map into fixed size of (16*8) while there respective channel dimension remain same. So, three instance of ROI pooling layer is required to convert (122*h) × (122*w) × 64 → (8 × 16 × 64) for first layer, (63*h) × (63*w) × 160 → (8 × 16 × 160) for third layer, and (33*h) × (33*w) × 192 → (8 × 16 × 192) for fifth layer. Finally, the fixed size feature maps are concatenated along the channel dimension to output a feature map of (8 × 16 × 416).

Please refer to figure 4, to see the difference in recognition module from the baseline model. Figure 4a shows the fixed size feature map of size (8 × 16 × 416) is passed to 7 classifier to get 7 different output representing 7 different letters of license plate. On other hand, this whole module has been changed to represent the license literals as sequence modelling problem seen in 4b. In the new design, the feature maps from ROI pooling layer is passed to bi-directional LSTM. The feature map is reshaped to get 128 (16*8) sequence length vector with the channel dimension 416 as input embedding size to LSTM. Here, 128 also represents the time step for LSTM. The output of first LSTM layer is fed to linear layer and then another bidirectional LSTM is stacked on top of it. The output of the final LSTM unit (from final BiLSTM layer) is passed to a linear layer. For the detailed dimensions of each layer please refer to table 2. Finally, the dimension of output of recognition module is (N × 128 × 69) where N is batch size. Here, 69 dimension is chosen representing the number of different literals 'a time step'/LSTM can take. The different literals are total number of Chinese characters, total English alphabets, $(0-9)$ digits, and one dash ('-'). So, to put it all together, the output of recognition module is a 128-length sequence where each element is a vector of 69 representing the *score or probability* for each of 69 characters.



(a) Baseline Recognition Module



(b) Changed Recognition Module

Figure 4: The figure on left side shows the baseline model which uses a simple 7 classifers and cross entropy loss. While figure on right side shows new changes made. The removal of linear classifiers with additional of LSTM layers along with CTC loss makes it detect variable numbers in license plate.

The output of this stacked bidirectional LSTM is passed to CTC loss which helps to form number sequence of license plate. Please refer to section 3.2 to see how CTC loss is formulate and used to train model and refer to 3.4 to see methodology to extract the license number during inference.

3.2 Training Loss

The baseline model uses two different losses: Smooth L1 Loss for detection module and cross entropy for the recognition module. With new changes in recognition module, cross entropy loss is no longer valid. Therefore, a new loss called **Connectionist Temporal Classification (CTC loss)** is used.

We feed output of the recognition module (N × 128 × 69) which are score/probability of 69 literals at each 128 time step, and the ground truth actual sequence to this loss.

The Conditional probability is mathematically represented as

$$\underbrace{p(Y|X)}_{\text{The CTC Conditional Probability}} = \underbrace{\sum_{A \in A_X}}_{\textbf{marginalize}\text{ over set of valid alignment}} \underbrace{\prod_{t=1}^{T} p_t(a_t|X)}_{\text{computing the probability for a single alignment set by step}} \tag{2}$$

To better understand, lets say the recognition module outputs 2 time-steps with 3 possible literal (a,b,-). The score map (output of recognition module) can be seen as

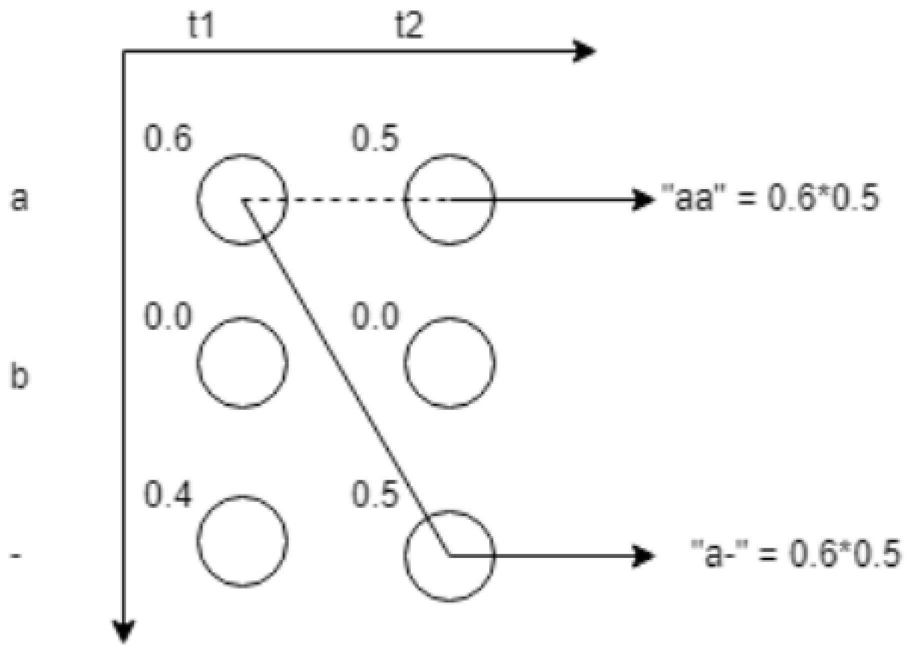


Figure 5: An example to represent how the score/probability of an alignment is computed.

First, we need to understand how the CTC finds the actual sequence. Referring to figure 5, the correct alignment is extracted by taking highest score at each time-step and taking the corresponding literal. And then the repetitive literals are collapsed into a single literal. Example: alignment "aa" will be collapsed to "a" or for other alignments "ccaaaattt" will be collapsed to "cat". But it suffers from a problem; for words like "deep" it will collapse "e" to have "dep". CTC loss overcomes this my introducing a dash ("-"). For instance

- "a-" or "aa" or "-a" → "a"
- "de-ep" or "de—eee–p" → "deep"

So, all the consecutive literals are collapsed till it hits the dash and then the dashes are removed to get the sequence. This is how it extracts the sequence during inference.

To calculate CTC loss, we first find the probability of a single alignment. An alignment consist of taking the probability of a valid sequence which is multiplying the scores of literal at each time-step.

Referring to figure 5, let's say the correct label is "a". So, the possible alignments are "aa", "-a", "a-". The probability of single alignment "aa" is computed by multiplying the the score of "a" in first time step "t1" and the score of "a" in second time step "t2" represented by dotted arrow in figure 5. Similarly for "a-" and "-a" alignments. Hence, the total CTC conditional probability is sum of all the "possible" alignments shown in equation 1. The loss is just **negative likelihood** of this Conditional probability.

Implementation Details
Initially, I tried to use the implementation provided by Baidu link. But my network didn't converge at all. So, I upgraded the PyTorch version and ported the whole code (previously in PyTorch v0.3) to newer version v1.0. This allowed me to use the native CTC loss implementation with CuDnn support. This loss converged well with the model.

3.3 Training Methodology

Preprocessing the input is first resized to 480x480, converted to float32 and normalized by dividing with 255.0

Shape of INPUT Tensors Input shape for train: (N, 3, 480,480) Input shape for test/val: (1,3,480,480)

Shape of Output Tensors Output shape: [(N,4) , (N,128,69)]
Where, (N,4): denotes the bounding box location of N images in batch size (detection module output) (N,128,69): denotes the 128 time steps with 69 literal scores for each time step. (recognition module output)

Shape of Output Tensor for Each Layer The shape of tensor as it passes through various hidden layer in detection module is shown in table 1. The shape of tensor as it passes through ROI pooling layer and BiLSTM layer in recognition module is shown in table 2 (N represents batch_size)

Blocks	Layers	Dimensions
Conv-Block1	Conv2d	(N,48,240,240)
	BatchNorm2d+ReLU	(N,48,240,240)
	MaxPool+Dropout	(N,48,121,121)
Conv-Block2	Conv2d	(N,64,121,121)
	BatchNorm2d+ReLU	(N,64,121,121)
	MaxPool+Dropout	(N,64,122,122)
Conv-Block3	Conv2d	(N,128,122,122)
	BatchNorm2d+ReLU	(N,128,122,122)
	MaxPool+Dropout	(N,128,62,62)
Conv-Block4	Conv2d	(N,160,62,62)
	BatchNorm2d+ReLU	(N,160,62,62)
	MaxPool+Dropout	(N,160,63,63)
Conv-Block5	Conv2d	(N,192,63,63)
	BatchNorm2d+ReLU	(N,192,63,63)
	MaxPool+Dropout	(N,192,32,32)
Conv-Block6	Conv2d	(N,192,32,32)
	BatchNorm2d+ReLU	(N,192,32,32)
	MaxPool+Dropout	(N,192,33,33)
Conv-Block7	Conv2d	(N,192,33,33)
	BatchNorm2d+ReLU	(N,192,33,33)
	MaxPool+Dropout	(N,192,17,17)
Conv-Block8	Conv2d	(N,192,17,17)
	BatchNorm2d+ReLU	(N,192,17,17)
	MaxPool+Dropout	(N,192,18,18)
Conv-Block9	Conv2d	(N,192,18,18)
	BatchNorm2d+ReLU	(N,192,18,18)
	MaxPool+Dropout	(N,192,10,10)
Conv-Block10	Conv2d	(N,192,10,10)
	BatchNorm2d+ReLU	(N,192,10,10)
	MaxPool+Dropout	(N,192,11,11)

Table 1: Layers of Detection Module with their Respective Dimensions.

Layers	**Output Shape**
ROI layer for feature map 1	(N,64,8,16)
ROI layer for feature map 3	(N, 160,8,16)
ROI layer for feature map 5	(N,192,8,16)
Concatenated output	(N,416,8,16)
Reshaped and input to first BiLSTM	(N,128,416)
Output of first BiLSTM layer(+linear)	(N,128,256)
Output of second BiLSTM layer(+ linear)	(N,128, 69)

Table 2: Layer Schematics for Recognition Module

HyperParameters Tuning

Table 3 shows the summary of different hyperparameters I tuned around in order to achieve results.

HyperParameters	**Played around for tuning**	**Final Value**
Batch Size	8,64,128	128
Epochs	Detection Only: 25,50 Detection + Recognition: 4	Detection Only: 25 Detection + Recognition: 4
DropOut	0.2,0.5	0.2
Optimizier	Adam, SGD	Adam
Learning rate	0.001, 0.005	0.001
Dimension of hidden units in BiLSTM	256,416,512	256

Table 3: Different Hyperparameters

3.4 Inference Methodology

Search Beam Algorithm During inference, the output of model is decoded to extract the best alignment. There are the various decoding algorithm present. The one I used was *Best path search beam* algorithm. During inference, the output of recognition module i.e., the score map of dimension 128 time step and 69 score at each time step which represents score of each possible literal. In Best Path algorithm, the score with highest value is picked at each time step and then corresponding literal is use at that time step. For example, below is an example of 120 sequence length.

- aaa–w————6——————xx——————8—————-11– → aw6x81

.

As explained in CTC loss section 3.2, the CTC loss introduces a dash to avoid collapsing the same consecutive literal. So, the consecutive same characters are collapsed and the dashes are removed to get model predicted sequence.

4 Performance and Results

Training Performance

The Log file is attached with the report (log1.out). It contains the training performance of only recognition loss as the detection module was not trained again separately. The pretrained weights from previous trained module were used to initialize the weights of detection module and were made fixed/untrainable.

Loss Diagram:

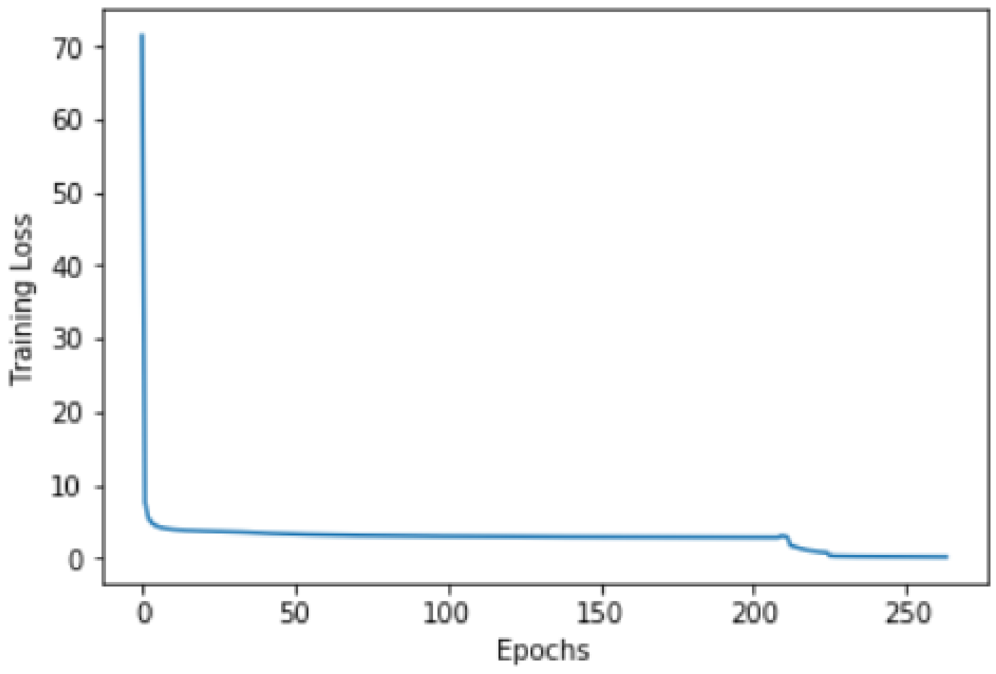


Figure 6: Changes in loss during training.

Testing Accuracy
Comparing to the paper, the testing accuracy is sightly low because of the difference in training epochs. While the paper trained for more than 70 epochs. Due to HPRC resource constraint, I could train for only 4 epochs. (Took around 42 hrs on 2 GPUs to train on 150K training images). But I had a slight improvement in performance compare to the baseline recognition module when trained for same number of the epochs (4- epochs).
Baseline Recognition Model Test Accuracy: 92%
Changed Recognition Model Test Accuracy: 93.8%
These accuracy represents character level character. That is total number of characters correctly recognized by the model.

5 Instructions to train/test model

5.1 Install Dependencies

- Python 3+
- Pytorch v1.0+
- Numpy, imutils, opencv2
- TensorboardX
- Files: rpnet.py, roi_pooling.py, load_data.py, utils.py, Detection_weights.pth

5.2 Annotated Code

The code base spans over multiple files and hence is attached with the submission.Also, it is available here.

5.3 Dataset

The dataset is huge containing around 300k images; hence it is not included as part of submission. Please refer to the official page here in order to download dataset. However, in order to test the model performance and evaluate via GUI, a small subset of images has been included at the GitHub page here. The test images provided were not part of training and also varying difficulty in terms of weather, snow, haze, blur etc.

5.4 Execution

The steps to train and test have been explained in detailed way at the README page of GitHub submission here.

6 Instructions on GUI usage

6.1 Install Dependencies

- Python 3+
- Pytorch v1.0+
- Numpy, imutils, opencv2
- Tkinter
- Files: weight_4.pth, model.py, utils.py, roi_pooling.py

6.2 Annotated Code

The code spans over multiple files and hence is attached and also available at the GUI folder of GitHub submission here.

6.3 Execution

Run the GUI.py file to start the GUI. (python GUI.py)

6.4 Youtube Link

Click here to watch the video demo for the GUI.

7 Conclusion

To sum up all, the change in recognition module from linear classifiers to BiLSTMs and corresponding required change in loss from cross entropy loss to CTC loss has made the model able to detect any number of literals in the license plate of the car. This has increased the overall versatility of the model and also made it one step closer to real-world end-to-end license plate detection and recognition system.

References

[1] Zhenbo Xu, Wei Yang, Ajin Meng, Nanxue Lu, and Huan Huang. Towards end-to-end license plate detection and recognition: A large dataset and baseline. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 255–271, 2018.

[2] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[3] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.

[4] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.