LPRNet: License Plate Recognition via Deep Neural Networks

Sergey Zherzdev
ex-Intel*
IOTG Computer Vision Group
sergeyzherzdev@gmail.com

Alexey Gruzdev
Intel
IOTG Computer Vision Group
alexey.gruzdev@intel.com

Abstract

This paper proposes LPRNet - end-to-end method for Automatic License Plate Recognition without preliminary character segmentation. Our approach is inspired by recent breakthroughs in Deep Neural Networks, and works in real-time with recognition accuracy up to **95%** *for Chinese license plates:* **3 ms/plate** *on nVIDIA® GeForce™GTX 1080 and* **1.3 ms/plate** *on Intel® Core™i7-6700K CPU.*

LPRNet consists of the lightweight Convolutional Neural Network, so it can be trained in end-to-end way. To the best of our knowledge, LPRNet is the first real-time License Plate Recognition system that does not use RNNs. As a result, the LPRNet algorithm may be used to create embedded solutions for LPR that feature high level accuracy even on challenging Chinese license plates.

1. Introduction

Automatic License Plate Recognition is a challenging and important task which is used in traffic management, digital security surveillance, vehicle recognition, parking management of big cities. This task is a complex problem due to many factors which include but are not limited to: blurry images, poor lighting conditions, variability of license plates numbers (including special characters *e.g.* logograms for China, Japan), physical impact (deformations), weather conditions (see some examples in Fig. 1).

The robust Automatic License Plate Recognition system needs to cope with a variety of environments while maintaining a high level of accuracy, in other words this system should work well in natural conditions.

This paper tackles the License Plate Recognition problem and introduces the LPRNet algorithm, which is designed to work without pre-segmentation and consequent recognition of characters. In the present paper, we do not consider License Plate Detection problem, however, for our particular case it can be done through LBP-cascade.

[0]This work was done when Sergey was an Intel employee

LPRNet is based on Deep Convolutional Neural Network. Recent studies proved effectiveness and superiority of Convolutional Neural Networks in many Computer Vision tasks such as image classification, object detection and semantic segmentation. However, running most of them on embedded devices still remains a challenging problem.

LPRNet is a very efficient neural network, which takes only **0.34 GFLops** to make a single forward pass. Also, our model is real-time on Intel Core i7-6700K SkyLake CPU with high accuracy on challenging Chinese License plates and can be trained end-to-end. Moreover, LPRNet can be partially ported on FPGA, which can free up CPU power for other parts of the pipeline. Our main contributions can be summarized as follows:

- LPRNet is a real-time framework for high-quality license plate recognition supporting template and character independent variable-length license plates, performing LPR without character pre-segmentation, trainable end-to-end from scratch for different national license plates.
- LPRNet is the first real-time approach that does not use Recurrent Neural Networks and is lightweight enough to run on variety of platforms, including embedded devices.
- Application of LPRNet to real traffic surveillance video shows that our approach is robust enough to handle difficult cases, such as perspective and camera-dependent distortions, hard lighting conditions, change of viewpoint, etc.

The rest of the paper is organized as follows. Section 2 describes the related work. In sec. 3 we review the details of the LPRNet model. Sec. 4 reports the results on Chinese License Plates and includes an ablation study of our algorithm. We summarize and conclude our work in sec. 5.

2. Related Work

In the earlier works on general LP recognition, such as [1] the pipeline consist of character segmentation and char-

Figure 1. Example of LPRNet recognitions

acter classification stages:

- Character segmentation typically uses different hand-crafted algorithms, combining projections, connectivity and contour based image components. It takes a binary image or intermediate representation as input so character segmentation quality is highly affected by the input image noise, low resolution, blur or deformations.
- Character classification typically utilizes one of the optical character recognition (OCR) methods - adopted for LP character set.

Since classification follows the character segmentation, end-to-end recognition quality depends heavily on the applied segmentation method. In order to solve the problem of character segmentation there were proposed end-to-end Convolutional Neural Networks (CNNs) based solutions taking the whole LP image as input and producing the output character sequence.

The segmentation-free model in [2] is based on variable length sequence decoding driven by connectionist temporal classification (CTC) loss [3, 4]. It uses hand-crafted features LBP built on a binarized image as CNN input to produce character classes probabilities. Applied to all input image positions via the sliding window approach it makes the input sequence for the bi-directional Long-Short Term Memory (LSTM) [5] based decoder. Since the decoder output and target character sequence lengths are different, CTC loss is used for the pre-segmentation free end-to-end training.

The model in [6] mostly follows the approach described in [2] except that the sliding window method was replaced by CNN output spatial splitting to the RNN input sequence ("sliding window" over feature map instead of input).

In contrast [7] uses the CNN-based model for the whole LP image to produce the global LP embedding which is decoded to a 11-character-length sequence via 11 fully connected model heads. Each of the heads is trained to classify the i-th target string character (which is assumed to be padded to the predefined fixed length), so the whole recognition can be done in a single feed-forward pass. It also utilizes the Spatial Transformer Network (STN) [8] to reduce the effect of input image deformations.

The algorithm in [9] makes an attempt to solve both license plate detection and license plate recognition problems by single Deep Neural Network.

Recent work [10] tries to exploit synthetic data generation approach based on Generative Adversarial Networks [11] for data generation procedure to obtain large representative license plates dataset.

In our approach, we avoided using hand-crafted features over a binarized image - instead we used raw RGB pixels as CNN input. The LSTM-based sequence decoder working on outputs of a sliding window CNN was replaced with a fully convolutional model which output is interpreted as character probabilities sequence for CTC loss training and greedy or prefix search string inference. For better performance the pre-decoder intermediate feature map was augmented by the global context embedding as described in [12]. Also the backbone CNN model was reduced significantly using the low computation cost basic building block inspired by SqueezeNet Fire Blocks [13] and Inception Blocks of [14, 15, 16]. Batch Normalization [17] and Dropout [18] techniques were used for regularization.

LP image input size affects both the computational cost and the recognition quality [19], as a result there is a trade-off between using high [6] or moderate [7, 2] resolution.

3. LPRNet

3.1. Design architecture

In this section we describe our LPRNet network architecture design in detail.

In recent studies tend to use parts of the powerful classification networks such as VGG, ResNet or GoogLeNet as 'backbone' for their tasks by applying transfer learning techniques. However, this is not the best option for building fast and lightweight networks, so in our case we redesigned

main 'backbone' network applying recently discovered architecture tricks.

The basic building block of our CNN model backbone (Table 2) was inspired by SqueezeNet Fire Blocks [13] and Inception Blocks of [14, 15, 16]. We also followed the research best practices and used Batch Normalization [17] and ReLU activation after each convolution operation.

In a nutshell our design consists of:

- location network with Spatial Transformer Layer [8] (optional)
- light-weight convolutional neural network (backbone)
- per-position character classification head
- character probabilities for further sequence decoding
- post-filtering procedure

First, the input image is preprocessed by the Spatial Transformer Layer, as proposed in [8]. This step is optional but allows to explore how one can transform the input image to have better characteristics for recognition. The original LocNet (see the Table 1) architecture was used to estimate optimal transformation parameters.

Layer Type	Parameters	
Input	94x24 pixels RGB image	
AvgPooling	#32 3x3 stride 2	—
Convolution	#32 5x5 stride 3	#32 5x5 stride 5
Concatenation	channel-wise	
Dropout	0.5 ratio	
FC	#32 with TanH activation	
FC	#6 with scaled TanH activation	

Table 1. **LocNet architecture**

Layer Type	Parameters/Dimensions
Input	$C_{in} \times H \times W$ feature map
Convolution	# $C_{out}/4$ 1x1 stride 1
Convolution	# $C_{out}/4$ 3x1 strideh=1, padh=1
Convolution	# $C_{out}/4$ 1x3 stridew=1, padw=1
Convolution	# C_{out} 1x1 stride 1
Output	$C_{out} \times H \times W$ feature map

Table 2. **Small Basic Block**

The backbone network architecture is described in Table 3. The backbone takes a raw RGB image as input and calculates spatially distributed rich features. Wide convolution (with 1×13 kernel) utilizes the local character context instead of using LSTM-based RNN. The backbone subnetwork output can be interpreted as a sequence of character probabilities whose length corresponds to the input image pixel width. Since the decoder output and the target character sequence lengths are of different length, we apply the method of CTC loss [20] - for segmentation-free end-to-end training. CTC loss is a well-known approach for situations where input and output sequences are misaligned and have variable lengths. Moreover, CTC provides an efficient way to go from probabilities at each time step to the probability of an output sequence. More detailed explanation about CTC loss can be found in .

Layer Type	Parameters
Input	94x24 pixels RGB image
Convolution	#64 3x3 stride 1
MaxPooling	#64 3x3 stride 1
Small basic block	#128 3x3 stride 1
MaxPooling	#64 3x3 stride (2, 1)
Small basic block	#256 3x3 stride 1
Small basic block	#256 3x3 stride 1
MaxPooling	#64 3x3 stride (2, 1)
Dropout	0.5 ratio
Convolution	#256 4x1 stride 1
Dropout	0.5 ratio
Convolution	# class_number 1x13 stride 1

Table 3. **Back-bone Network Architecture**

To further improve performance, the pre-decoder intermediate feature map was augmented with the global context embedding as in [12]. It is computed via a fully-connected layer over backbone output, tiled to the desired size and concatenated with backbone output. In order to adjust the depth of feature map to the character class number additional 1×1 convolution is applied.

For the decoding procedure at the inference stage we considered 2 options: greedy search and beam search. While greedy search takes the maximum of class probabilities in each position, beam search maximizes the total probability of the output sequence [3, 4].

For post-filtering we use a task-oriented language model implemented as a set of the target country LP templates. Note that post-filtering is applied together with Beam Search. The post-filtering procedure gets top-N most probable sequences found by beam search and returns the first one that matches the set of predefined templates which depends on country LP regulations.

3.2. Training details

All training experiments were done with the help of TensorFlow [21].

We train our model with '**Adam**' optimizer using batch size of **32**, initial learning rate **0.001** and gradient noise scale of **0.001**. We drop the learning rate once after every **100k** iterations by a factor of **10** and train our network for **250k** iterations in total.

In our experiments we use data augmentation by random affine transformations, *e.g*. rotation, scaling and shift.

It is worth mentioning, that application of LocNet from the beginning of training leads to degradation of results, because LocNet cannot get reasonable gradients from a recognizer which is typically too weak for the first few iterations. So, in our experiments, we turn LocNet on only after **5k** iterations.

All other hyper-parameters were chosen by cross-validation over the target dataset.

4. Results of the Experiments

The LPRNet baseline network, from which we started our experiments with different architectures, was inspired by [2]. It's mainly based on Inception blocks followed by a bidirectional LSTM (biLSTM) decoder and trained with CTC loss. We first performed some experiments aimed at replacing biLSTM with biGRU cells, but we did not observe any clear benefits of using biGRU over biLSTM.

Then, we focused on eliminating of the complicated biLSTM decoder, because most modern embedded devices still do not have sufficient compute and memory to efficiently execute biLSTM. Importantly, our LSTM is applied to a spatial sequence rather than to a temporal one. Thus all LSTM inputs are known upfront both at the training stage as well as at the inference stage. Therefore we believe that RNN can be replaced by spatial convolutions without a significant drop in accuracy. The RNN-less model with some backbone modifications is referenced as LPRNet basic and it was described in details in sec. 3.

To improve runtime performance we also modified LPRNet basic by using 2×2 strides for all pooling layers. This modification (the LPRNet reduced model) reduces the size of intermediate feature maps and total inference computational cost significantly (see GFLOPs column of the Table 4).

4.1. Chinese License Plates dataset

We tested our approach on a private dataset containing various images of Chinese license plates collected from different security and surveillance cameras. This dataset was first run through the LPB-based detector to get bounding boxes for each license plate. Then, all license plates were manually labeled. In total, the dataset contains **11696** cropped license plate images, which are split as **9:1** into training and validation subsets respectively.

Automatically cropped license plate images were used for training to make the network more robust to detection artifacts because in some cases plates are cropped with some background around edges, while in other cases they are cropped too close to edges with no background at all or event with some parts of the license plate missing.

Table 4 shows recognition accuracies achieved by different models.

Method	Recognition Accuracy, %	GFLOPs
LPRNet baseline	94.1	0.71
LPRNet basic	95.0	0.34
LPRNet reduced	94.0	0.163

Table 4. **Results on Chinese License Plates.**

4.2. Ablation study

It is of vital importance to conduct the ablation study to identify correlation between various enhancements and respective accuracy/performance improvements. This helps other researchers adopt ideas from the paper and reuse most promising architecture approaches. Table 5 shows a summary of architecture approaches and their impact on accuracy.

Approach	LPRNet							
Global context				✓	✓	✓	✓	✓
Data augm.	✓	✓	✓		✓	✓	✓	✓
STN-alignment		✓	✓			✓	✓	✓
Beam Search			✓				✓	✓
Post-filtering			✓					✓
Accuracy, %	53.4	58.6	59.0	62.95	91.6	94.4	94.4	95.0

Table 5. **Effects of various tricks on LPRNet quality.**

As one can see, the largest accuracy gain (**36%**) was achieved using the global context. The data augmentation techniques also help to improve accuracy significantly (by **28.6%**). Without using data augmentation and the global context we could not train the model from scratch.

The STN-based alignment subnetwork provides noticeable improvement of 2.8-5.2%. Beam Search with post-filtering further improves recognition accuracy by 0.4-0.6%.

4.3. Performance analysis

The LPRNet reduced model was ported to various hardware platforms including CPU, GPU and FPGA. The results are presented in the Table 6.

Target platform	1 LP processing time
GPU + cuDNN	3 ms
CPU (using Caffe [22])	11-15 ms
CPU + FPGA (using DLA [23])	4 ms[1]
CPU (using IE from Intel OpenVINO [24])	1.3 ms

Table 6. **Performance analysis.**

[1]The LPRNet reduced model was used

Here GPU is nVIDIA® GeForce™1080, CPU is Intel® Core™i7-6700K SkyLake, FPGA is Intel® Arria™10 and IE is for Inference Engine from Intel® OpenVINO.

5. Conclusions and Future Work

In this work, we have shown that for License Plate Recognition one can utilize pretty small convolutional neural networks. LPRNet model was introduced, which can be used for challenging data, achieving up **to 95%** recognition accuracy. Architecture details, its motivation and the ablation study was conducted.

We showed that LPRNet can perform inference in real-time on a variety of hardware architectures including CPU, GPU and FPGA. We have no doubt that LPRNet could attain real-time performance even on more specialized embedded low-power devices.

The LPRNet can likely be compressed using modern pruning and quantization techniques, which would potentially help to reduce the computational complexity even further.

As a future direction of research, LPRNet work can be extended by merging CNN-based detection part into our algorithm, so that both detection and recognition tasks will be evaluated as a single network in order to outperform the LBP-based cascaded detector quality.

6. Acknowledgements

We would like to thank Intel® IOTG Computer Vision (ICV) Optimization team for porting the model to the Intel Inference Engine of OpenVINO, as well as Intel® IOTG Computer Vision (ICV) OVX FPGA team for porting the model to the DLA. We also would like to thank Intel® PSG DLA and Intel® Computer Vision SDK teams for their tools and support.

References

[1] C. N. E. Anagnostopoulos, I. E. Anagnostopoulos, I. D. Psoroulas, V. Loumos, and E. Kayafas, "License Plate Recognition From Still Images and Video Sequences: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 3, pp. 377–391, Sep. 2008. 1

[2] H. Li and C. Shen, "Reading Car License Plates Using Deep Convolutional Neural Networks and LSTMs," *arXiv:1601.05610 [cs]*, Jan. 2016, arXiv: 1601.05610. 2, 4

[3] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, 2012th ed. Heidelberg ; New York: Springer, Feb. 2012. 2, 3

[4] A. Graves, S. Fernndez, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 369–376. 2, 3

[5] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. 2

[6] T. K. Cheang, Y. S. Chong, and Y. H. Tay, "Segmentation-free Vehicle License Plate Recognition using ConvNet-RNN," *arXiv:1701.06439 [cs]*, Jan. 2017, arXiv: 1701.06439. 2

[7] V. Jain, Z. Sasindran, A. Rajagopal, S. Biswas, H. S. Bharadwaj, and K. R. Ramakrishnan, "Deep Automatic License Plate Recognition System," in *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing*, ser. ICVGIP '16. New York, NY, USA: ACM, 2016, pp. 6:1–6:8. 2

[8] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial Transformer Networks," *arXiv:1506.02025 [cs]*, Jun. 2015, arXiv: 1506.02025. 2, 3

[9] H. Li, P. Wang, and C. Shen, "Towards End-to-End Car License Plates Detection and Recognition with Deep Neural Networks," *ArXiv e-prints*, Sep. 2017. 2

[10] X. Wang, Z. Man, M. You, and C. Shen, "Adversarial Generation of Training Examples: Applications to Moving Vehicle License Plate Recognition," *ArXiv e-prints*, Jul. 2017. 2

[11] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *ArXiv e-prints*, Jun. 2014. 2

[12] W. Liu, A. Rabinovich, and A. C. Berg, "ParseNet: Looking Wider to See Better," *arXiv:1506.04579 [cs]*, Jun. 2015, arXiv: 1506.04579. 2, 3

[13] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5mb model size," *arXiv:1602.07360 [cs]*, Feb. 2016, arXiv: 1602.07360. 2, 3

[14] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *arXiv:1602.07261 [cs]*, Feb. 2016, arXiv: 1602.07261. 2, 3

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv:1409.4842 [cs]*, Sep. 2014, arXiv: 1409.4842. 2, 3

[16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *arXiv:1512.00567 [cs]*, Dec. 2015, arXiv: 1512.00567. 2, 3

[17] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv:1502.03167 [cs]*, Feb. 2015, arXiv: 1502.03167. 2, 3

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. 2

[19] S. Agarwal, D. Tran, L. Torresani, and H. Farid, "Deciphering Severely Degraded License Plates," San Francisco, CA, 2017. 2

[20] A. Hannun, "Sequence modeling with ctc," *Distill*, 2017, https://distill.pub/2017/ctc. 3

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *arXiv:1603.04467 [cs]*, Mar. 2016, arXiv: 1603.04467. 3

[22] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014. 4

[23] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An OpenCL(TM) Deep Learning Accelerator on Arria 10," *arXiv:1701.03534 [cs]*, Jan. 2017, arXiv: 1701.03534. 4

[24] "Intel OpenVINO Toolkit | Intel Software." [Online]. Available: https://software.intel.com/en-us/articles/OpenVINO-InferEngine 4