Parking System License Plate Detection Based on Convolution Neural Networks GPU Optimization

Ziad Elkhatib, Adel Ben Mnaouer, Omar Mashaal, Nor Azman Ismail, Mohd Azman Bin Abas and Fuad Abdulgaleel

December 15, 2021

Parking System License Plate Detection based on Convolution Neural Networks GPU Optimization

Ziad El Khatib, Adel Ben Mnaouer, Omar Mashaal, Nor Azman Ismail, Mohd Azman Bin Abas, Fuad Abdulgaleel

Canadian University Dubai and University Teknologi Malaysia
ziad.elkhatib@cud.ac.ae; adel@cud.ac.ae; omar.mashaal@cud.ac.ae, azman.abas@utm.my, azman@utm.my, mdfarid@utm.my

***Abstract*—This paper presents a parking system license plate detection based on convolution neural networks GPU optimization. When number of strides increased 2 by 2 it reduced memory allocation and reduced training time by half, however accuracy is also reduced from 80% to 75.79%. Accuracy is traded-off to avoid running into GPU memory allocation issues.**

***Keywords*— Convolutional neural networks; License plate detection; GPU optimization.**

I. INTRODUCTION

Parking system license plate detection based on convolution neural networks (CNN) uses convolutional layers that are either completely interconnected or max pooled [4], [6], [7], [10], [12].

The convolutional layer performs a convolutional operation on the input before passing the result to the next layer. The network can be much deeper due to this convolutional operation. With this, convolutional neural networks can be effective in image and video recognition, however it requires graphics processing unit GPU optimization to avoid running into memory issues.

Parking systems license plate detection based on convolution neural networks requires graphics processing unit GPU optimization when training convolutional neural network algorithm. Otherwise it would generate several errors including out of memory error. A plate detection based on convolution neural networks GPU optimization is presented. When number of strides increased 2 by 2 it reduced memory allocation and reduced training time by half, however accuracy is also reduced from 80% to 75.79%. Accuracy is traded-off to avoid running into memory allocation issues. Investigation in the use of neural network algorithms for plate detection is performed in [1], [2], [3], [4], [5] and [6]. However, they neural network processing is not performed on graphics processing unit GPU framework. Other published work [7], [8], [9], [10], [11], [12] and [13] do not propose graphics processing unit GPU optimization for convolutional neural network processing.

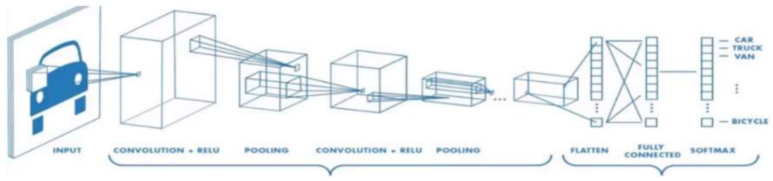

Fig. 1. Convolutional neural networks algorithm.

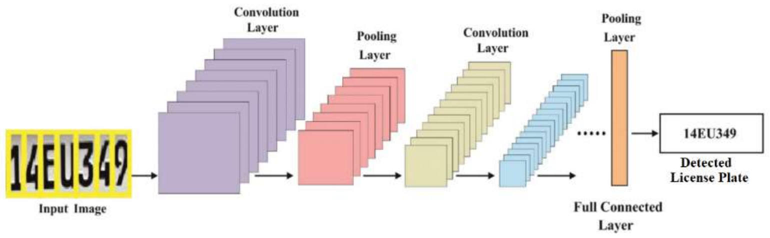

Fig. 2. License plate detection convolution neural networks.

Parking system license plate detection based on convolution GPU optimization is shown in Figure 1 and Figure 2. It uses Nvidia Jetson Nano device connected to machine vision sensors including AI camera 800M CSI interface IMX219-160 module and an IP camera. AI camera connected to the camera port on the Jetson Nano device and IP camera attached via USB. The 128-core Maxwell architecture-based GPU and Quad-core ARM CPU on the Jetson device does the real-time video analytics.

The Nvidia Jetson Nano is one of the System on Modules (SoM) developed by Nvidia Corporation, with graphics processing unit GPU accelerated processing in mind. The SoM consists of 128-core NVIDIA Maxwell™ architecture-based GPU, controlled by a CPU with Quad-core ARM A57 architecture, along with 4GB of DDR4 RAM. The Jetson nano can be used as a general purpose Linux-powered computer, which has advanced uses in neural networks, thanks to its GPU accelerated processor [4], [6], [7], [10], [12].

Since the Jetson Nano is designed with special hardware. The hardware-accelerated parallel computing using the GPU, a special framework needs to be installed. Nvidia calls this special framework that enables parallel computing on the GPU CUDA (Compute Unified Device Architecture). The framework supports highly known machine learning frameworks like Tensorflow, Keras, and PyTorch, the CPU can invoke the CUDA functions on the GPU through CUDA framework and thus enables the parallel computing possibility. The flow diagram in Figure 3 below indicates the typical program flow when executing a GPU-accelerated [4], [6], [7], [10], [12].

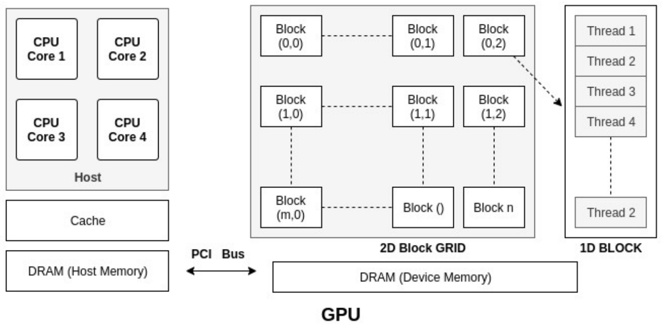

Fig. 3. Parallel computing on the GPU CUDA Compute Unified Device Architecture.

II. Convolutional nerual networks Detection

A convolution neural network license plate detection is a feedforward neural network as shown in Figure 4. The architecture of convolution neural network is takes a 2-dimensional structure as an input image [4], [6], [7], [10], [12].

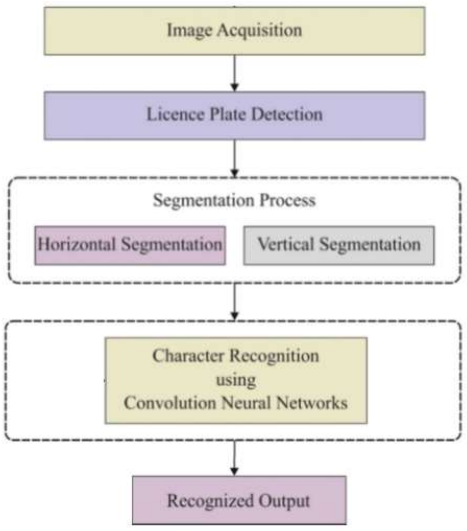

Fig. 4. Flow Chart of the CNN Based algorithm.

The convolutional neural network parameter layers are composed of small filters [4], [6], [7], [10], [12].

We slide or convolve each filter across the input during forward pass and compute product of the filter and the input [4], [6], [7], [10], [12].

If we have a 2D array of pixels' image x and we have a feature detector or filter w after applying a mathematical operation, we get an output which is called a feature map s(t) given by the following convolution function [4], [6], [7], [10], [12].

$$s[t] = (x * w)[t] = \Sigma x[a]w[a+t]$$

The feature detector filter in the convolution operation identifies the edges in the image.

We can do the same convolution operation as a sum over array elements where I is a 2-dimensional array and K is the filter kernel convolution function [4], [6], [7], [10], [12].

$$s[i,j] = \Sigma\Sigma I[m,n]K[i-m,k-n]$$

We can re-write equation to a cross correlation function [4], [6], [7], [10], [12].

$$s[i,j] = \Sigma\Sigma I[i-m,k-n]K[m,n]$$

We let filter slide over the input and it gives 2-dimensional activation map at each position [4], [6], [7], [10], [12].

The feature map dimensions are given by the following equation where the input image size is W and the feature detector field size is F and S is the stride [4], [6], [7], [10], [12].

$$Size\ of\ Feature\ Map\ = 1 + (\mathrm{W} - \mathrm{F} + 2\mathrm{P})/\mathrm{S}$$

We can apply linear transformation if we have a feature detector filter and a bias unit with the following equation [4], [6] and [7].

$$output\ = \mathrm{input} * \mathrm{weight} + \mathrm{bias}$$

After each convolution operation we apply ReLU activation function. ReLU function which replaces negative values with zero.

Next we apply max pooling function to have invariance to translation which helps detect features that are common in an input image [4], [6], [7], [10], [12].

Pool layer will down sample the previous layers feature map. It reduces the size and computation in the network [4], [6], [7], [10], [12].

Then we flatten the max pooled output that are input to the fully connected neural network. Fully connected layer compute class scores [4], [6], [7], [10], [12].

Fully connected layers have an activation function or a softmax activation in order to predict classes [4], [6], [7], [10], [12]. To compute the output is rearranged as a 1-D array as shown in Figure 5 and the following convolutional neural network algorithm.

```
_cnn = Sequential()
_cnn.add(InputLayer((28, 28)))
_cnn.add(Reshape((28, 28, 1)))
_cnn.add(Conv2D(filters=32, kernel_size=(2,2), strides=(1,1)))
_cnn.add(AveragePooling2D(pool_size=(2,2), strides=(1,1)))
_cnn.add(Conv2D(filters=64, kernel_size=(2,2), strides=(1,1)))
_cnn.add(AveragePooling2D(pool_size=(2,2), strides=(1,1)))
_cnn.add(Conv2D(filters=128, kernel_size=(2,2), strides=(1,1))
_cnn.add(AveragePooling2D(pool_size=(2,2), strides=(1,1)))
_cnn.add(Flatten())
_cnn.add(Dense(32, 'relu'))
_cnn.add(Dense(10, 'linear'))
_cnn.add(Softmax())
```

In a neural network weights are how neural networks learn. Weights are adjusted to compute signal strength. The weights are multiplied with the inputs(x) and add a bias term. This is given by the compact equation for forward propagation [4], [6], [7], [10], [12].

$$z = x * w + b$$

Using ReLU as the activation function we now predict the output and compare predicted output with the actual output value as shown in Figure 5. Since this is a classification problem we use cross entropy function [4], [6], [7], [10], [12].

We use cross entropy function since we are performing classification. Cross Entropy is a non-negative cost function with a range between 0 and 1 [4], [6], [7], [10], [12].

$$c = -\frac{1}{n}\Sigma[ylogy' + (1-y)\log(1-y')]$$

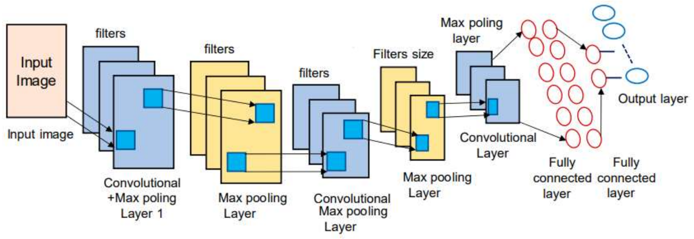

Fig. 5. License plate detection based on convolution neural networks.

III. GPU OPTIMIZATION

Epoch is defined when the complete dataset is used once for learning. We repeat the forward propagation for multiple epochs until we converge [4], [6], [7], [10], [12].

Learning rate adjust the weights. Lower value of the learning rate, the slower it will convergence. Learning rates are randomly initialized [4], [6], [7], [10], [12].

```
Epoch 1/10
1875/1875 [==============================] - ETA:
0s - loss: 1.2370 - accuracy: 0.5743INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 388s
207ms/step - -loss: 1.2370 - -accuracy: 0.5743 - -val_loss:
1792.4512 - -val_accuracy: 0.1000
Epoch 2/10
1875/1875 [==============================] - ETA:
0s - loss: 0.9263 - accuracy: 0.6467INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 388s
207ms/step - -loss: 0.9263 - -accuracy: 0.6467 - -val_loss:
318.3828 - -val_accuracy: 0.1000
Epoch 3/10
1875/1875 [==============================] - ETA:
0s - loss: 0.8388 - accuracy: 0.6824INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 384s
205ms/step - -loss: 0.8388 - -accuracy: 0.6824 - -val_loss:
716.1693 - -val_accuracy: 0.1000
Epoch 6/10
1875/1875 [==============================] - ETA:
0s - loss: 0.6437 - accuracy: 0.7614INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 386s
206ms/step - -loss: 0.6437 - -accuracy: 0.7614 - -val_loss:
24.1597 - -val_accuracy: 0.2179
Epoch 7/10
1875/1875 [==============================] - ETA:
0s - loss: 0.5930 - accuracy: 0.7844INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 388s
207ms/step - -loss: 0.5930 - -accuracy: 0.7844 - -val_loss:
230.5167 - -val_accuracy: 0.1000
Epoch 8/10
1875/1875 [==============================] - ETA:
0s - -loss: 0.5661 - -accuracy: 0.7948INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 389s
207ms/step - -loss: 0.5661 - -accuracy: 0.7948 - -val_loss:
86.5537 - -val_accuracy: 0.1000
Epoch 9/10
1875/1875 [==============================] - ETA:
0s - -loss: 0.5431 - -accuracy: 0.8018INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 397s
212ms/step - loss: 0.5431 - -accuracy: 0.8018 - -val_loss:
308.9158 - -val_accuracy: 0.1000
Epoch 10/10
1875/1875 [==============================] - ETA:
0s - loss: 0.5194 - -accuracy: 0.8078INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 408s
218ms/step - -loss: 0.5194 - -accuracy: 0.8078 - -val_loss:
19.9043 - -val_accuracy: 0.2378
```

When number of strides increased 2 by 2 it reduced memory allocation and reduced training time by half, however accuracy is also reduced from 80% to 75.79% as shown in convolution neural network summary below. Accuracy is traded-off to avoid running into GPU memory allocation issues.

```
Epoch 8/10
1874/1875 [============================>.] - ETA:
0s - loss: 0.6869 - -accuracy: 0.7544INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 31s
16ms/step - -loss: 0.6870 - -accuracy: 0.7544 - -val_loss:
0.6206 - -val_accuracy: 0.7806
Epoch 9/10
1872/1875 [============================>.] - ETA:
0s - loss: 0.6788 - -accuracy: 0.7559INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 31s
17ms/step - -loss: 0.6786 - -accuracy: 0.7559 - -val_loss:
0.6021 - val_accuracy: 0.7845
Epoch 10/10
1872/1875 [============================>.] - ETA:
0s - loss: 0.6758 - accuracy: 0.7579INFO:tensorflow:Assets
written to: cnn_model/assets
1875/1875 [==============================] - 31s
16ms/step - -loss: 0.6758 - -accuracy: 0.7579 - -val_loss:
0.6260 - val_accuracy: 0.7728
```

IV. Performance

TABLE I. ACCURACY PERFORMANCE.

MODEL	ACCURACY	STRIDE	TRAINING TIME	EPOCH
WITHOUT GPU OPTIMIZATION	80%	1 BY 1	30 MINUTES	10
WITH GPU OPTIMIZATION	75.79%	2 BY 2	REDUCED BY HALF	10

Table 1 shows the accuracy performance with and without GPU optimization.

Conclusion

A parking system license plate detection based on convolution neural networks GPU optimization is presented. When number of strides increased 2 by 2 it reduced memory allocation and reduced training time by half, however accuracy is also reduced from 80% to 75.79% as shown in convolution neural network summary Table 1. Accuracy is traded-off to avoid running into GPU memory allocation issues.

Acknowledgment

The authors are grateful to all who contributed to the research CUD-UTM research collaboration funding.

References

[1] S. Du, M. Ibrahim, M. Shehata, and W. Badawy, "Automatic license plate recognition (ALPR): A state-of-the-art review," IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 2, pp. 311 - 325, Feb 2013.

[2] Y. Ma, C. Gou, K. Wang, Y. Yao, and Z. Li, "Vehicle license plate recognition based on extremal regions and restricted boltzmann machines," IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 4, pp. 1096–1107, April 2016.

[3] A. Zacepins, A. V. Kozitsky, P. Ramesh, and M. Shreve, "Segmentation and annotation-free license plate recognition with deep localization and failure identification," IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 9, pp. 2351–2363, Sept 2017.

[4] M. S. Z. Masood, G. Shu, A. Dehghan, and E. G. Ortiz, "License plate detection and recognition using deeply learned convolutional neural networks," CoRR, vol. abs/1703.07330, 2017.

[5] R. Panahi and I. Gholampour, "Accurate detection and recognition of vehicle plate numbers for high-speed applications," IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 4, pp. 767–779, April 2017.

[6] S. Montazzolli and C. R. Jung, "Real-time brazilian license plate detection and recognition using deep convolutional neural networks," 30th SIBGRAPI Conference on Graphics, Patterns and Images, pp. 55–62, Oct 2017.

[7] M. A. Rafique, W. Pedrycz, and M. Jeon, "Vehicle license plate detection using region-based convolutional neural networks," Soft Computing, June 2017.

[8] D. Menotti, G. Chiachia, A. X. Falc˜ao, and V. J. O. Neto, "Vehicle license plate recognition with random convolutional networks," in 2014 27th SIBGRAPI Conference on Graphics, Patterns and Images, pp. 298–303, Aug 2014.

[9] P. Svoboda, M. Hradiˇs, L. Marˇs´ık, and P. Zemc´ık, "CNN for license plate motion deblurring," in 2016 IEEE International Conference on Image Processing (ICIP), pp. 3832–3836, Sept 2016.

[10] H. Li and C. Shen, "Reading car license plates using deep convolutional neural networks and LSTMs," CoRR, vol. abs/1601.05610 June 2016.

[11] F. Awan, N. Crespi, "A Comparative Analysis of Machine/Deep Learning Models for Parking Space Availability Prediction", Jan. 2020.

[12] G. Ning, Z. Zhang, C. Huang, X. Ren, H. Wang, C. Cai, and Z. He, "Spatially supervised recurrent convolutional neural networks for visual object tracking," in 2017 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–4, May 2017.

[13] M. Donoser, C. Arth, and H. Bischof, "Detecting, tracking and recognizing license plates," in Computer Vision – ACCV 2007, Y. Yagi, S. B. Kang, I. S. Kweon, and H. Zha, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, , pp. 447–456, Aug 2007.