Proceedings of the 2018 IEEE
International Conference on Cyborg and Bionic Systems
Shenzhen, China, October 25-27, 2018

FPGA-based Implementation of Hand Gesture Recognition Using Convolutional Neural Network

Tongtong Zhang[1], Weiguo Zhou[1], Xin Jiang[1], Yunhui Liu[1,2], $Fellow, IEEE$

***Abstract*— Convolutional Neural Network (CNN) is a deep learning algorithm which is widely used in image processing and pattern recognition due to its robustness to feature invariance. However, it is also computation-intensive that results in a bad real-time performance. Field Programmable Gate Arrays (FPGA) has good performance in energy-efficiency, flexibility of parallel processing and pipelined operations. Thus it is expected to be used for accelerating deep learning algorithm. In this research, a FPGA based system is developed to realize the real-time Hand Gesture Recognition. We train a designed CNN model with caffe framework and obtain the model's parameters on PC. Bilinear interpolation algorithm is used to adjust the size of the image captured by camera. Then we use FPGA to implement the inference process of Hand Gesture Recognition with obtained parameters by designing an accelerator using Xilinx SDx tools.**

I. INTRODUCTION

Convolutional Neural Network(CNN) [1] is one of the advanced algorithms of image processing due to its superiority in accuracy and capability of automatic feature extraction. Compared with traditional image processing algorithms in which it needs to design a feature extractor in advance, a CNN can automatically determine the essential features for a specific purpose. It is also computation-intensive and energy-consuming.

To improve the calculation speed and reduce the ennergy consumption, a lot of approaches have been attempted. For example, the utilization of Graphics Processing Units (GPU) [2] [3] demonstrates good performance in acceleration of the training and inference. It obtains a obviously speed acceleration than CPU. However its energy dissipation property prohibits its application in mobile embedded systems. The another attemption is the utilization of ASIC chip [4] [5]. Although this technology has the advantage in performance and energy efficiency, its high fabrication cost and limitation in flexibility make the technology unattractive. Given these constraints, FPGA is becoming to a prior choice due to its low power consumption, reconfigurability as well as its reasonable price. Some researchers have done some works before [6] [7] [8] [9] [10], but their works mainly concentrate on the implementation of acceleration programs.

This research is concentrated on implementing a CNN architecture in embedded hardware. We not only realize the transplantation of the network, but also a real time image capturing and displaying. Firstly, training of the network is conducted on PC and it provides well tuned weights and parameters for inference usage on FPGA later. Then the inference of CNN is executed on FPGA with the pretrained parameters. The paper is organised as follows. In section II, some preliminary knowledge about Convolutional Neural Network is introduced. In section III, we introduce how this research is carried out. Firstly, image preprocessing is employed to make the selected dataset satisfy network input. Then we come up with a simple model that can be implemented in hardware and train it on PC. Lastly, introduce how to transplant the network into FPGA platform. The experiment results is shown in section IV and the conclusion remarks and discussion on the future work is given in section V.

*This work was supported by Shenzhen Peacock Plan Team grant (KQTD20140630150243062), Shenzhen Key Laboratory grant(ZDSYS20140508161825065) and Shenzhen Fundamental Research grant (JCYJ20170811155308088).

[1]School of Mechanical Engineering and Automation, Harbin Institute of Technology(Shenzhen), Shenzhen, China.

[2]Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong, Hong Kong.

II. PRELIMINARY KNOWLEDGE

Convolutional Neural Network is a well-known deep learning algorithm and the related solutions represent the state-of-the-art of image classification and visual recognition. Its structure is characterized by a chain of multiple layers and it applies consecutive kernels on the input image for the purpose of extracting feature maps from input image. In the output, it use a fully connected layer to calculate the possibilities of input image [11] [6]. Generally, a CNN is consisted of input layer, hidden layer, and output layer and the hidden layer is made up of convolutional layer, sub-sampling layer and fully connected layer.

- Convolutional layers are used to identify and extract features which are collected into a so-called feature maps within the image. In order to extract these features from the image, convolutional layers apply a series of K kernels on the image and the output of each kernel is a feature map. Generally, a convolution layer is consisted of convolution operation, sub-sampling and activation function layer [7].

$$O_{i,j,k} = \sum_{h=0}^{H_k} \sum_{m=0}^{W_k} \sum_{c=0}^{C_k} (\omega_{h,m,c} * x_{i+h,j+m,c}) + b_k \quad (1)$$

The convolutional operation can be seen as the result of the product of the correspongding position of the kernel and the local area of the image. Equation (1) shows how the convolution operation is implemented [1]. The kernel is made up of a $H_k * W_k * C_k$ matrix of weights, where H_k represents the height of the kernel,

978-1-5386-7355-3/18/$31.00 ©2018 IEEE

W_k represents its width, C_k represent its channels and b_k is the bias value of the k_{th} kernel, $x_{i,j}$ represents the pixel value of point (i, j) in pixel coordinates. We can treat the kernel as a small window that slides on the image from topleft to downright. Every time the window slides with a step, it executes convolutional operation between the kernel parameters and the corresponding pixels to produce a pixel value as the output feature map.

- The function of the sub-sampling layer is to further abstract the features. Usually he takes the mean or maximum value of the feature map as a output feature. After sub-sampling, we can obtain a generalization features with much less dimensions. At the same time, this can improve the system's robustness and it will not be easy overfitting.
- Activation function layer is used to add non-linear properties to the convolutional layers output feature map in order to make up the expression ability of linear function as linear function can not modify complex non-lnlinear issuses well [8]. Sigmoid function, tanh function and ReLU function are generally used activation function. They can be seen in Fig. 2. It obviously that the gradient of sigmoid and tanh is very smooth in the saturation region. It is close to 0, so it is easy to cause the problem of vanishing gradient and slow down the convergence speed. In contrast, ReLU function has only one threshold of zero. ReLU's gradient is a constant in most cases and helps solve network convergence problems. Thus ReLU function is widely used.
- Then a fully-connected layer, which is also known as Multi-Layer Perception (MLP) [12] [13], receives the input features. The features are then converted into a vector from the last convolutional layer feature map.

$$o_j = \sum_{i=0}^{I} (\omega_{i,j} * x_j) + b_j \tag{2}$$

Equation (2) shows how MLP works. $\omega_{i,j}$ is the weight parameter between i^{th} neuron in the vector and j^{th} neuron in output vector.Each neuron in the two vectors is connected to each other. And the number of j is equal to the number of classification classes. Generally, there is a normalization operation [1] followed to the last layer to convert the output in the range [0,1]. It ensures that the sum of the output is 1. In this way, the final output can be regarded as the probability of the input image belonging to a specific class. The image is classified as the one which has the highest probability.

III. IMPLEMENTATION

In this research, the whole work is divided into two stages: the first one is model training stages on PC and the other one is network inference stage with pretrained weights and bias parameters on FPGA platform.

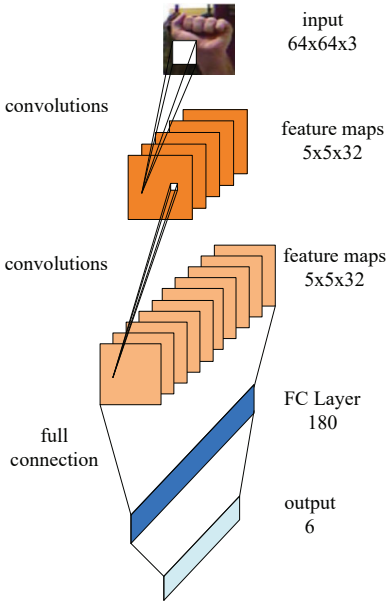
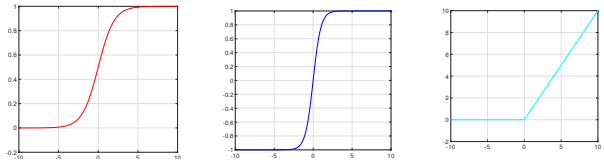


Fig. 1. Structure of CNN Network



(a) Sigmoid (b) Tanh (c) ReLU

Fig. 2. Activation Function

A. Hand Gesture Dataset

In this research, we choose the dataset that contains 6 different hand gestures as we want to denote six different meanings: start, left, right, up, down and stop. Two critical problems exist in the dataset if they are considered to be used in training a CNN model. The first one is that these images have different size, so they cannot be used directly as the input due to the number of neurons in the fully-connected layer is a specific value. To handle this problem, all the images are resized into 64x64. The another issuse regarding the dataset is that its number is limited. It will lead to overfitting problem during the model training process. In order to solve this problem. Data augmentation [14] strategies are used such as adding random noise, image rotation, reflection and contrast conversion. After resizing and data augmentation operation, we confer the gestures with different labels from 1 to 6. Then the images in the data set will be used as the input training data of CNN model. Some samples in the dataset are shown in Fig. 3.

B. Convolutional Neural Network

Generally, the depth of convolutional neural network determines the accuracy of network in large extent. However, considering the FPGA resource condition, the depth of the network should be limited. If the network is too deep, the

Fig. 3. Samples of Hand Gesture

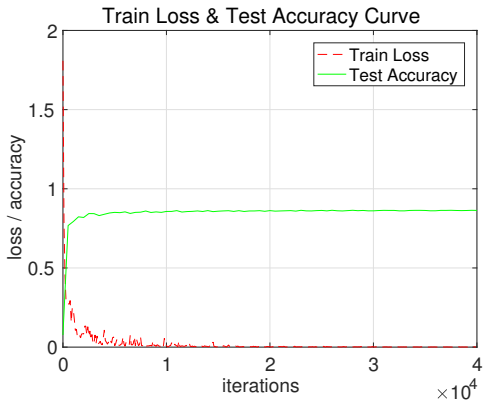


Fig. 4. CNN model training results

platform resource will be out of usage. Thus, in this research, we adopt a simple convolutional neural network with two convolutional layers and two fully-connected layers. We choose ReLU function as activation function due to its superiority mentioned before, and it can be easily realized with less resource consumption compared with other functions. The structure of this model is shown in Fig. 1. The input of it is image with resolution 64x64 in three channels, the kernel size of of two convolutional layer are all 5x5x32, the first fully connected layer has 180 neurons and the last layer has 6 neurons which is equal to the number of classes.

C. Network Training

Firstly, we construct the model with caffe framework and then train the model on PC. The Fig. 4 shows the general information about the training. It reveals that the training loss is considerably high at the very beginning and then it converges rapidly to a relatively low level. At the same time, the inference accuracy on test dataset achieves a stable level at approximately 86.3 percentage. After training, we extract weights and bias parameters of the model for inference usage on FPGA platform.

D. Hardware Implementation

Convolutional Neural Network algorithm is a computation-intensive processing and its computation is mainly concentrated on the convolutional layer. So

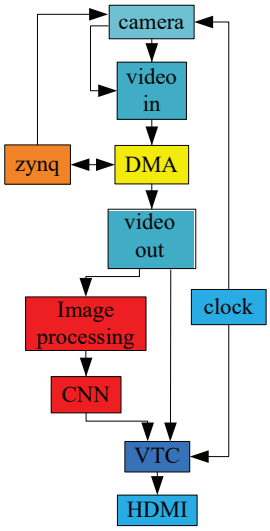


Fig. 5. Hardware Architecture Schematic Diagram

realizing the convolutions calculation in hardware is a feasible choice for accelerating the algorithm. Fig. 5 illustrates the general process of the experiment.

If we ignore the red blocks in the schematic diagram, then it will be an image capturing and displaying task. This task can be seen as two stages: the first stage is image acquisition in which real-time images are captured and saved in DDR by using DMA. The next stage is to realize the display. These functions can be realized in real-time to the video stream.

The red blocks in Fig. 5 play the role of image scaling and classification. The image processing part realizes image loading from DDR and adjust the image to the size required by CNN input. The CNN component executes inference process with the resized images and pretrained weights and bias parameters. It will identify the class that the image belongs to. This will produce a class label for the image and the label will be displayed in a monitor through HDMI interface.

As mentioned before, the convolutional layer is the most computation-intensive process among all the layers in CNN structure. Thus, realizing it in hardware is the key for performance improvement. But it is trivial to implement the multiplication operation between kernel and local area of the image with HDL(Hardware Description Language). Besides this, memory assignment is another challenge. Considering these issues, a synthesis tool called SDSoC [15] is employed in this research. It can convert software code into hardware blocks. To realize the convolutional operation, the pseudocode of algorithm can be seen as follow.

pragama HLS PIPELINE and # pragama HLS UNROLL are the indirective that help to synthesis hardware circuits [16] [17]. The PIPELINE indirective indicates pipeline operation. It will not reduce the first but the successive modules' delay, which results in improvement in system throughout capacity. In this work, all kernels are put through pipeline operation. Fig. 7 shows its process.

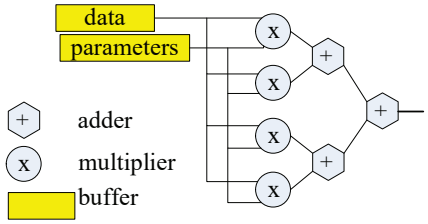


Fig. 6. Matrix Multiplication Acceleration in Hardware

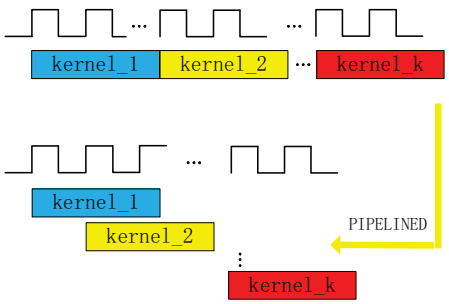


Fig. 7. Kernels PIPELINE operation

Algorithm 1 Convolutional Layer Pseudocode

```
for(k=0;k<K;k++) #kernels
{
#pragma HLS PIPELINE
  for(c=0;c<C;c++) #channels
    {
    for(i=0;i<W;i++)#image width
    {
        for(j=0;j<H;j++)#image height
        {
            for(x=0;x<WK;x++) #kernel width
            {
#pragma HLS UNROLL
                for(y=0;y<HK;y++)
                {
rst[k][i][j] += kernel[k][c][x][y] * inputFM[c][i+x][j+y]
} } } } } }
```

Matrix expansion also plays an important role in system performance improvement in terms of computing speed acceleration. The expansion realizes parallel processing between kernel parameters and pixels in local areas of images. Firstly, we use buffer to cache image data and parameters. We should split the buffer to accelerate the algorithm in hardware. As the image and kernel are all two dimensional matrix, so we split them in second dimension. The accelerated convolution operation principle in hardware is shown in Fig. 6. Data and network parameters are read from buffer and these values are multiplied with each other simultaneously first and then are summed to their productions. It thus contributes to complete the multiplication and addition operation within one system clock.

The implementation of ReLU function in hardware is very

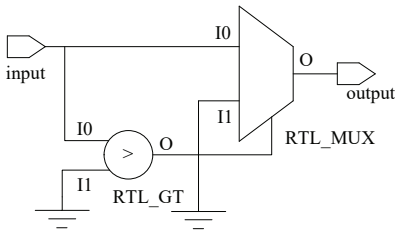


Fig. 8. Hardware Implementation of ReLU Function

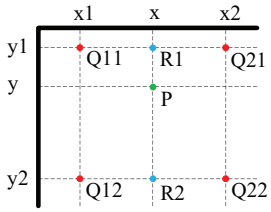


Fig. 9. Bilinear Interpolation Algorithm

easy, it only consumes an operator and a multiplexer. The operator compares whether the input is over than zero or not and its result determines the output of multiplexer. The hardware implementation result is shown in Fig. 8.

E. Bilinear Interpolation

In order to satisfy the input requirement of the Convolutional Neural Network, the images captured by the camera must be resized to be the same as that of the training images. The captured images have the resolution of 1080p while the resolution of the training data is 64x64.

Bilinear interpolation algorithm [18] is one of well adopted approaches for image resizing. Mathematically, bilinear interpolation can be seen as a linear interpolation of a function with two variables. The core idea of it is to perform a linear interpolation in two directions. Fig. 9 shows how bilinear interpolation algorithm works.

We need to map the output image pixels into original input image. Supposed that $P0(x0, y0)$ is pixel point in output image ,then the mapped point $P(x, y)$ can be found according to equation (3), where SW and SH indicate scale factor in X direction and Y direction,respectively. $Q_{ij}(i = 1, 2; j = 1, 2)$ are neighbour pixel points of mapped point in original image and they can be found according to equation (4).

$$\begin{cases} x = \dfrac{x_0}{SW} \\ y = \dfrac{y_0}{SH} \end{cases} \tag{3}$$

$$Qij = \left(\left[\frac{x_0}{SW}\right] + i - 1, \left[\frac{y_0}{SH}\right] + j - 1\right) \tag{4}$$

Once all the points are found in input image, we can use bilinear interpolation algorithm to calculate the pixel value in output image. Firstly, X direction interpolation is calculated to obtain the pixels value in point $R1, R2$ with the equation (5).

(a) original image (b) sub-sampling image

Fig. 10. Bilinear Interpolation Result

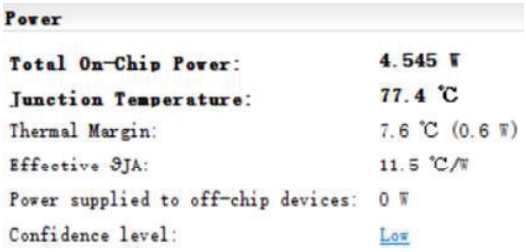


Fig. 11. FPGA Power consumption

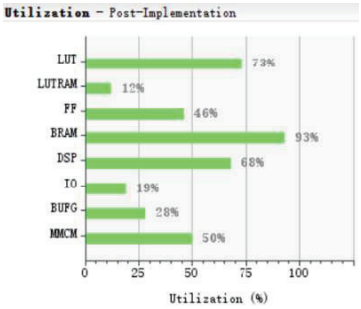


Fig. 12. FPGA Area Utilization

$$pixel\_Ri = u * pixel\_Q2i + (1-u) * pixel\_Q1i \quad (5)$$

$$pixel\_P = v * pixel\_R2 + (1-v) * pixel\_R1 \quad (6)$$

where

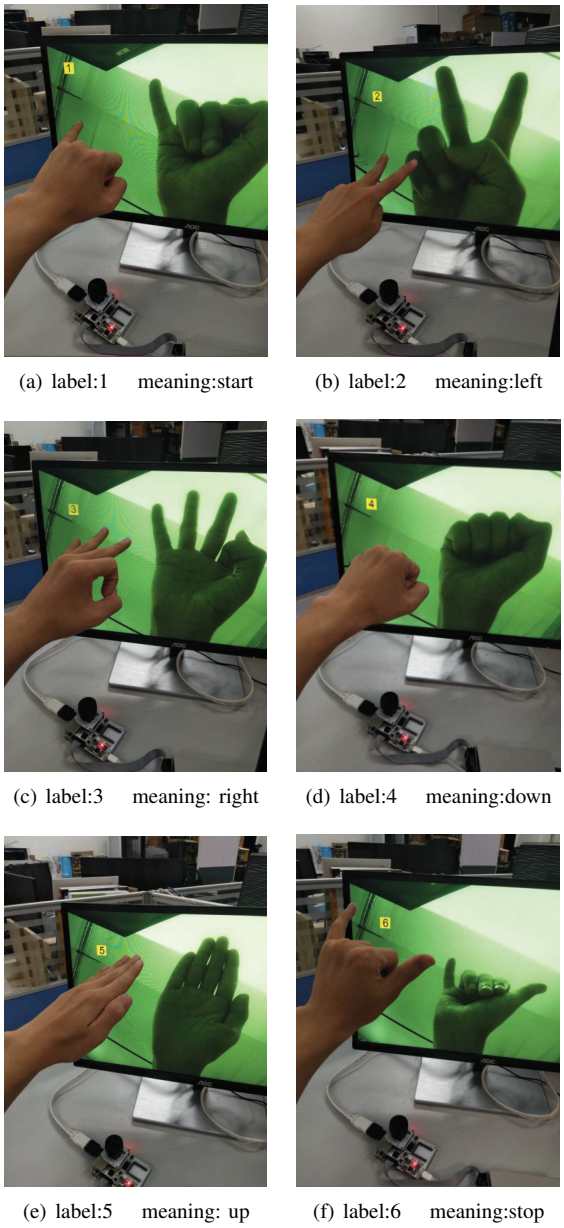
$$u = x - \left[\frac{x_0}{SW}\right] \qquad v = y - \left[\frac{y_0}{SH}\right]$$

Then, the same operation is executed in y direction and it calculates the pixel value with the calculated pixels value in R1 and R2 with the equation (6).

As a result, we can get the output pixel value according to equation (3)-(6) and the bilinear interpolation results can be seen in Fig. 10.

IV. EXPERIMENT RESULTS

We implement the image processing and convolutional neural network algorithm by using SDSoC tool. The final result of the hand gesture classification result is displayed on screen. It contains two components: one is the captured image and the other is a label that indicates the class that the image belongs to. The overall power consumption and its total resource utilization of the implemented design are shown in Fig. 12 and Fig. 11 , respectively. The overview of the experiment results are shown in Fig. 13.



(a) label:1 meaning:start (b) label:2 meaning:left

(c) label:3 meaning: right (d) label:4 meaning:down

(e) label:5 meaning: up (f) label:6 meaning:stop

Fig. 13. Predicted Results

V. CONCLUSION

This research presents a real time hand gesture recognition system based on FPGA. It is characterized by its high robustness of rotation invariance and translation invariance.

As for the future research topic, a target recognition algorithm will be adopted to localize hand location and then send the selected area to the recognition algorithm to classify.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] S.-H. Han and K.-Y. Lee, "Implemetation of image classification cnn using multi thread gpu," in *SoC Design Conference (ISOCC), 2017 International*. IEEE, 2017, pp. 296–297.

[3] S. I. Baykal, D. Bulut, and O. K. Sahingoz, "Comparing deep learning performance on bigdata by using cpus and gpus," in *2018 Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT)*. IEEE, 2018, pp. 1–6.

[4] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Toward accelerating deep learning at scale using specialized hardware in the datacenter," in *Hot Chips 27 Symposium (HCS), 2015 IEEE*. IEEE, 2015, pp. 1–38.

[5] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014, pp. 609–622.

[6] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *Field-Programmable Custom Computing Machines (FCCM), 2016 IEEE 24th Annual International Symposium on*. IEEE, 2016, pp. 40–47.

[7] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[8] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2018.

[9] D. Patel, R. Parmar, A. Desai, and S. Sheth, "Gesture recognition using fpga and ov7670 camera," in *Inventive Systems and Control (ICISC), 2017 International Conference on*. IEEE, 2017, pp. 1–4.

[10] L. Suriano, A. Rodriguez, K. Desnos, M. Pelcat, and E. de la Torre, "Analysis of a heterogeneous multi-core, multi-hw-accelerator-based system designed using preesm and sdsoc," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2017 12th International Symposium on*. IEEE, 2017, pp. 1–7.

[11] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[12] F. Huang, "The application of multi-layer perception networks in the parameters optimization of stamping forming process," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 4. IEEE, 2008, pp. 882–886.

[13] D. Lee and H. Yeo, "A study on the rear-end collision warning system by considering neural network," in *Intelligent Vehicles Symposium (IV), 2015 IEEE*. IEEE, 2015, pp. 24–30.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[15] K. Srijongkon, R. Duangsoithong, N. Jindapetch, M. Ikura, and S. Chumpol, "Sdsoc based development of vehicle counting system using adaptive background method," in *Micro and Nanoelectronics (RSM), 2017 IEEE Regional Symposium on*. IEEE, 2017, pp. 235–238.

[16] Y. Zhou and J. Jiang, "An fpga-based accelerator implementation for deep convolutional neural networks," in *Computer Science and Network Technology (ICCSNT), 2015 4th International Conference on*, vol. 1. IEEE, 2015, pp. 829–832.

[17] J.-W. Chang and S.-J. Kang, "Optimizing fpga-based convolutional neural networks accelerator for image super-resolution," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 2018, pp. 343–348.

[18] D. Suresha and H. Prakash, "Single picture super resolution of natural images using n-neighbor adaptive bilinear interpolation and absolute asymmetry based wavelet hard thresholding," in *Applied and Theoretical Computing and Communication Technology (iCATccT), 2016 2nd International Conference on*. IEEE, 2016, pp. 387–393.