A RISC-V Based Coprocessor Accelerator Technology Research for Convolution Neural Networks

Shuhua Zhang\textsuperscript{1,2}, Jie Tong\textsuperscript{1}, Jun Zhang\textsuperscript{1}, Yuqing Lei\textsuperscript{1}, Minghao Zhang\textsuperscript{1}, Dang Li\textsuperscript{1} and Lanruo Wang\textsuperscript{1}

\textsuperscript{1}China Electric Power Research Institute, No. 15 Xiaoying East Road, Qinghe, Beijing 100192, China
\textsuperscript{2}School of Electrical and Electronic Engineering, North China Electric Power University, Beijing 102206, China
Email: zhq_hd@163.com

Abstract. The advancements of neural networks have significantly improved the accuracy of a variety of intelligent applications, such as image processing, voice recognition and so on. However, one of the challenges is to accelerate the speed of inference with the networks designed to be deeper. In this paper, we first realize the algorithm involved in software. Then, combined with the expanded characteristic of RISC-V architecture, the acceleration of convolution operation taking up the largest proportion of computation in CNN is realized by coprocessor expansion mode on open-source Hummingbird E203 processor. Through tests, convolution coprocessor improves the performance of CNN while ensuring the accuracy in function. Finally, the Cifar-10 image classification, a common benchmark in machine learning, is used to verify the feasibility in functions and comparatively analysis the pure software-based and coprocessor-based implementations.

1. Introduction
In the past few years, the explosive growth of data promotes the development of deep learning. Convolutional neural networks (CNNs), an important sector of deep learning, have made a breakthrough in the field of computer vision, image and video analytics, creating new avenues for smart applications. Apart from the conventional mainstream cloud computing, shifting the power of deep learning to end devices or edge devices receives increasing attention from both industry and academia [1]. In the software perspectives, CNNs can reduce the amount of computation via sparse connections, weight sharing, pooling and other techniques. However, the amount of computation is still large due to the complex application scenarios as well as the inherit characteristics of neural network's structure. Most of the embedded device processors cannot meet with the requirements of such computation. Thus, it still remains a challenge to develop the applications of artificial intelligence in embedded system.

CNN models are often refined and designed much deeper and more complicated to improve accuracy. Traditionally, CNNs are often implemented on the commercial GPU or general CPU [2]. GPU has excellent ability of parallel computing but with prohibitive energy consumption and cost. The design of CPU is to execute the computer instructions and its serial characteristic cannot fully use parallelism of the internal structure of CNNs. Researchers begin to explore less conventional options (e.g. FPGAs [3] and ASICs [4]) to deal with the growing computational and energy efficiency requirements. With the feature of reconfigurability, FPGAs seem more suitable for deep learning acceleration, presenting much more flexibility and versatility than ASICs as well as more efficiency than CPUs and GPUs [5].
However, there are two challenges when designing an FPGA accelerator [6]. The first one is software because it is expensive to move an application into hardware and maintain it as code changes. The other one is deployment of the accelerator SoC hardware due to the difficulty of interconnecting many compute and interface cores at full bandwidth. RISC-V, an open instruction set architecture (ISA), is a promising candidate to solve the problems. It was fabricated by University of California at Berkeley, aimed at supplying with free and standard ISA suitable for hardware implementation [7]. Hummingbird processor is one of open source RISC-V architecture-based processors and SoC. Compared with the mainstream architectures such as X86 and ARM, the open scalability merit of RISC-V architecture makes it feasible to expand coprocessor accelerators to some specific applications. In view of this, this paper focuses on the implementation of scalable coprocessor of hummingbird processor and develops the accelerator for convolution operation. Given the presented context, the main contributions of this paper are:

- We propose a software-hardware coprocessor based on RISC-V architecture to accelerate the speed of convolution operation.
- We deploy the coprocessor in practical applications. We further implement Hummingbird E203 to support the CNNs.
- We conduct the cifar-10 image classification application. The CNNs related algorithm and convolution operation hardware acceleration coprocessor have been verified in function and compared in performance.

2. Related Work
A CNN accelerator leveraging coprocessor mechanism and partial reconguration is proposed on ARM1136 softcore microprocessor to realize accelerator dynamic reconfiguration online [8]. AlexNet model is implemented to validate the architecture using the MINIST database of handwritten numbers. The speed has been greatly improved. Furthermore, a large-scale parallel coprocessor is designed to accelerate CNNs, which is implemented on an off-the-shelf PCI-FPGA [9]. A model independent reconfigurable cooperative processing architecture is put forward on Xilinx virtex 7 FPGA to accelerate the implementation of CNNs, which exploits maximum data parallelism via parallel Multiply and Accumulate (MAC) units with caching techniques and interconnection networks [10]. Limited precision 32-bit Q-format fixed point quantization is also introduced for arithmetic representations and operations. An automation design tool is fabricated based on hardware architecture of Rainman and the parameterized FPGA module of Corerain technology, which simplifies the design process of CNN and applies for universal CNN models [11]. Researchers begin to explore RISC-V processor-based for acceleration for CNNs. A CNN processor based on RISC-V structure is raised by adopting the classic five-stage pipeline structure, implements instruction buffer memory and data buffer memory, and adds FLASH, SRAM, SDRAM and other peripheral devices [12]. To accelerate the execution of the convolution operation, vector store instruction, vector load instruction, vector addition instruction, and convolution operation instruction are employed. A domain-specific instruction set RV-CNN composed of 9 matrix instructions is adopted, which obtains higher code density than general ISAS by abstracting CNN into instructions [13].

3. Convolutional Neural Networks
Convolutional neural networks (CNNs), a common type of deep learning architecture, are inspired by the cognitive mechanism of biological vision and proposed by neuroscience [14]. LeCun [15] published a paper and established the modern structure of CNN, and later improved it. CNNs received extensive attention from both industry and academia since a team of researchers from the University of Toronto won the competition by a substantial margin in the ImageNet challenge in 2012 [16]. At present, the topological structures of CNNs have different forms, but their basic structures are similar. In this paper, we will illustrate the topological structure of CNNs. As illustrated in figure 1, A typical CNN consists of input layer, hidden layer and output layer. The hidden layer is mainly composed of convolution layer, pooling layer and full connection layer. The convolution layer and pooling layer are generally connected
alternately by local connection. The full connection layer is generally distributed in the last few layers of the network.

3.1. Input Layer
By using the method of gradient descent during training, the images will be classified into RGB three colors by the input layer of CNNs, which is conducive to the computation of CNNs.

3.2. Convolution Layer
Convolution layer is the core module of CNNs, the computation of which often takes up more than 90% of the network. Figure 2 shows the progress of characteristic diagram by convolution operation, where each input characteristic diagram corresponds to a convolution kernel, and the dotted boxes of different colors in input diagram correspond to different outputs. Each output is the processing result of input local information, reflecting the local feature information. And the same input characteristic diagram uses the same convolution kernel for characteristic processing, which is the weight sharing mechanism in convolution network [17]. Convolution operation can be calculated by sliding the convolution kernel on the input characteristic diagram. The formula of convolution operation is shown in Formula 1, where out($f_0, x, y$) is the value corresponding to the position $(x, y)$ in the $f_0$ output characteristic diagram, and W is convolution kernel weight matrix, b is bias of convolution layer, k is convolution kernel size, and S is the sliding step length of convolution kernel.

\[
out(f_0, x, y) = \sum_{n_i}^{N-1} \sum_{k_x=0}^{K_x-1} \sum_{k_y=0}^{K_y-1} W(f_0, n_i, k_x, k_y) \ast \text{in}(n_i, S \ast x + k_x, S \ast y + k_y) + b(f_0)
\]  

3.3. Pooling Layer
Pooling layer is usually deployed between successive convolution layers, which can reduce the amount of data and parameters and the over-fitting of the network by feature selection and information filtering of the input characteristic diagram [18]. The most commonly used sampling methods are Max pooling and Average pooling.

3.4. Full-Connected Layer
Full-connected layer is deployed at the end of CNNs, which is used to process the output characteristic of convolutional layer to reach the final result. Compared with convolutional layer, all neurons between the two layers in full-connected layer have more weight connections. The core computation process of full-connected layer is matrix multiplication as shown in Formula 2, where $x$ is the input vector of the full-connected layer, $W$ is the weight matrix, $b$ is the offset of the full-connected layer, and $y$ is the output vector of the full-connected layer.

\[
y = W \ast x + b
\]
Activation functions are usually deployed on the output of convolution layer or full-connected layer, aimed at increasing the nonlinearity of neural network and strengthening the expression ability of neural network to approach any function [19]. Commonly-used activation functions include Sigmod, Tanh, ReLU and so on [20].

3.5. Output Layer
Output layer of CNNs is usually an output classifier to distinguish and classify the output features. The final output layer of the convolutional neural network is. SoftMax, a common classifier, is used for multi-classification tasks, as demonstrated in Formula 3. $X$ is an n-dimensional vector, corresponding to n classifications and Softmax calculates the probability of N classifications.

$$
\sigma(X)_j = \frac{e^{x_j}}{\sum_{i=1}^{N} e^{x_i}}
$$

4. Extended Coprocessor Interface of Hummingbird E203

4.1. Introduction to Hummingbird E203
Hummingbird E203, a 32-bit open-source processor based on RISC-V architecture and two-stage pipeline, is developed by research team of China. It is designed for scenarios with low power consumption and small area, and suitable to replace the traditional 8051 core or arm Cortex-M series core in IoT. The characteristics of Hummingbird E203 are as follows:

- It can be configured as RV32IMAC or RV32EMAC architecture, and employs multi-cycle hardware multiplier and divider based on area optimization;
- It supplies with private ITCM (Instruction Tightly Coupled Memory) and DTCM (Data Tightly Coupled Memory), realizing separate storage of instruction and data and improving storage access performance;
- It provides instruction set extended interface with good scalability;
- It employs synthetic RTL code abided by robust Verilog 2001 syntax.

4.2. Instruction Set Extended Modes of Hummingbird E203
RISC-V instruction set architecture is a flexible and extended architecture. In order to facilitate users to extend RISC-V, RISC-V architecture predefines four groups of custom instruction types (custom-0, custom-1, custom-2 and custom-3) in 32-bit instructions. Each custom has its own opcode, as shown in table 1. Users can use these four instruction types to expand into custom coprocessor instructions.

Hummingbird E203 processor uses custom instruction to extend the coprocessor, and its instruction coding format is shown in figure 3. The opcode coding section is used to distinguish the types of custom-0, custom-1, custom-2 and custom-3; rd represents the purpose register code; rs1 and rs2 represent the source operand register code; xd, xs1 and xs2 represent whether the general registers corresponding to rd, rs1 and rs2 need to be accessed (read or write), respectively; funct7 represents the specific operation type of the instruction, which can be used as additional coding space to code more instructions.

![Figure 3. EAI instruction coding format.](image)

In the assembler, the user-defined instructions can be realized by using the pseudo instruction .insn, as follows:

```
.insn r opcode, func3, func7, rd, rs1, rs2
```
.insn is used to inform the compiler that the current instruction is in the form of .insn. r is used to specify that the instruction type is R-type. opcode, func3, func7, rd, rs1 and rs2 correspond to the bit fields in the instruction format of figure 3, and func3 corresponds to xs1, xs2 and xd.

Table 1. 32-Bit instruction opcode table based on RISC-V architecture (inst[1:0]==11).

| inst[4:2] | inst[6:5] | opcode  | func3  | func7  | rd | func3 corresponds to xs1, xs2 and xd. |
|----------|----------|---------|--------|--------|----|--------------------------------------|
| 00       | -000     | LOAD    | MIC    | MEM    | OP | IMM                                 |
| 01       | 001      | STORE   | AMO    | OP     | LUI| OP-IMM-32                            |
| 10       | 010      | MADD    | NMADD  | D      | reserved | CUSTOM-2/rv128                        |
| 11       | 111      | BRANCH  | JAL    | SYSTEM | reserved | CUSTOM-3/rv128 >80b                   |

4.3. EAI Interface of Hummingbird E203

EAI (Extension Accelerator Interface) interface is used to extend the coprocessor in the open source Hummingbird E203 processor, and the position of EAI in the pipeline structure is shown in figure 4. The decoding unit of processor decodes the opcode of the instruction at EXU level, and the information will be distributed through the request channel of EAI interface if belonging to the user-defined instructions. The coprocessor further decodes the instructions after receiving the effective instructions, performs specific operations according to the relevant information of the instructions and returns the results to the processor core through the feedback channel.

In order to expand the applications of coprocessor, Hummingbird E203 also supports storage resource sharing between coprocessor and main processor. A special access interface is reserved for the EAI coprocessor in the LSU module of the main processor, thus, the coprocessor based on the EAI interface can access the main processor. The addressable data storage resources include ITCM, DTCM, system memory bus, system equipment bus and fast IO interface.

5. Implementation of Convolution Coprocessor

The convolution operation often takes up 90% of the operation time of the whole convolution neural network, which may greatly affect the performance of the network. Therefore, we will adopt the hummingbird coprocessor expansion method to accelerate the operation of the convolution layer.

5.1. Hardware Architecture of Convolution Coprocessor

The hardware implementation block of convolution coprocessor is presented in figure 5, mainly consisting of control module, address generation module, multiply-accumulate module and output saturation module. The control module is responsible for the whole operation process, including instruction decoding, storage of parameter, reading memory data, operation execution and storage of output result; the address generation module is used to calculate the storage address of the required input data and corresponding output data; the multiply-accumulate module is the main calculation module of convolution operation; the output saturation module will limit the scope of output data.

5.2. Instruction Definition of Convolutional Coprocessor

In this paper, seven extended instructions are defined for the implementation of convolution coprocessor, where JJ_INIT_CH, JJ_INIT_IM, JJ_INIT_FS, JJ_INIT_PW, JJ_INIT_IMADDR and JJ_INIT_BIAS are used to initialize convolution parameters, and JJ_LOOP instruction is employed to execute convolution operations. The instructions used for parameter initialization are one clock instructions. When receiving the corresponding instructions, the operands will be read out and sent to the register for subsequent computation. The JJ_LOOP instruction that performs convolution operation.
is a variable multicycle instruction, of which the number of execution cycles is determined by the relevant parameters of convolution operation. The specific definitions of each instruction are shown in table 2.

Table 2. Instruction list of convolution coprocessing.

| Coprocessing instructions | Introduction | Code description | Opcode | Xd | Xs1 | Xs2 | Funct7 |
|---------------------------|--------------|------------------|--------|----|-----|-----|--------|
| JJ_INIT_CH                | Set the number of input and output tensor channels | custom0 | 0 | 1 | (The number of input tensor channels) | 1 | (The number of output tensor channels) | 3 |
| JJ_INIT_IM                | Set the size of input and output tensor | custom0 | 0 | 1 | (The size of input tensor) | 1 | (The size of output tensor) | 4 |
| JJ_INIT_FS                | Set filter core size and step size | custom0 | 0 | 1 | (Filter core size) | 1 | (Step size) | 5 |
| JJ_INIT_PW                | Set filling size and the initial address of filter weight data | custom0 | 0 | 1 | (Filling size) | 1 | (The initial address of filter weight data) | 6 |
| JJ_INIT_IMADDAR           | Set the initial address of input and output data | custom0 | 0 | 1 | (The initial address of input data) | 1 | (The initial address of output data) | 7 |
| JJ_INIT_BIAS              | Set bias and the initial address of biased data | custom0 | 0 | 1 | (The high 16 bits output the right offset, and the low 16 bits output the left offset) | 1 | (The initial address of biased data) | 11 |
| JJ_LOOP                   | Perform convolution operation | custom0 | 1 | 0 | 0 | 0 | 8 |
5.3. Programming Model of Convolution Coprocessor
The programming model of convolution coprocessor is shown in figure 6. In the software execution process, the convolution coprocessor needs to be initialized by parameter configuration and other instructions, and then the convolution operation is carried out by the JJ_LOOP instruction. After the execution of the instructions, the output result has been written to the corresponding memory.

6. Experimental Results
We evaluate the convolution coprocessor (Section V) on Hummingbird E203.

![Figure 6. Programming model of convolution coprocessor.](image)

6.1. Setup
We have customized a special FPGA prototype development board and JTAG debugger for the Hummingbird E203 SoC. The customized FPGA development board for Hummingbird E203 is based on a low-cost Xilinx XC7A75T-2FGG484I development board, equipped with onboard FPGA USB JTAG downloader, equipped with, independent MCU_FLASH chip and FPGA_FLASH chip, and the Hummingbird JTAG debugger interface (MCU_JTAG) and so on, as shown in figure 7. The development board can not only be used for circuit design, but also used as an MCU SoC prototype development board for developing embedded software because it preburns the Hummingbird E203 SoC. We use Ubuntu 16.04 Linux operating system, and install Xilinx Vivado software to the system. Download the E200_opensource project to the Linux system, and use the command (git clone https://github.com/SI-RISCV/e200_opensource.git) to enter the fpga directory under the e200_opensource directory folder. Running the command (make install CORE=e203) to compile for the e203 kernel. Use the JTAG debugger to connect the host PC to the FPGA development board. Download the HBird-E-SDK project to the native Linux system by using the command (git clone https://github.com/SI-RISCV/hbird-e-sdk). Additionally, the GNU tool chain is needed to compile software programs.

6.2. Convolution Coprocessor Based on EAI Interface
To verify the correctness of convolution coprocessor based on EAI interface in function and quantify the performance of its hardware implementation, this paper refers to the software algorithm implementation of convolution operation in ARM CMSIS-NN software library and transplants it into the Hummingbird E203 platform as the reference scheme for comparison test of convolution coprocessor. Convolution function interfaces are shown in table 3.
Figure 7. Hummingbird FPGA development board.

Table 3. Convolution function interfaces.

| Type               | Description                                                      |
|--------------------|------------------------------------------------------------------|
| riscv_status       | Input tensor pointer (The initial address of input data)         |
| const q7_t *       | dim_im_in                                                        |
| const uint16_t     | ch_im_in                                                         |
| const q7_t *       | wt                                                               |
| const uint16_t     | ch_im_out                                                        |
| const uint16_t     | dim_kernel                                                       |
| const uint16_t     | padding                                                          |
| const uint16_t     | stride                                                           |
| const q7_t *       | bias                                                             |
| const uint16_t     | bias_shift                                                       |
| const uint16_t     | out_shift                                                        |
| q7_t *             | Im_out                                                           |
| const uint16_t     | dim_im_out                                                       |
| q15_t *            | bufferA                                                          |
| q7_t *             | bufferB                                                          |

This experiment runs on Hummingbird development board, provides a certain amount of random input data for convolution operation and convolution coprocessor, and then compares the output data as well as running time of the two types of implementations. The parameter setting is presented in Table 4.

Table 4. Parameter-setting.

| dim_im_in | ch_im_in | ch_im_out | dim_kernel | padding | stride | bias_shift | out_shift | dim_im_out |
|-----------|----------|-----------|------------|---------|--------|------------|-----------|------------|
| 8         | 3        | 3         | 3          | 0       | 1      | 2          | 3         | 6          |

The pure software and coprocessor have the same random input. The convolution coprocessor has the same output result as the convolution operation realized by pure software, and achieves more than 20 times speedup over pure software implementation when the input data amount is small, as shown in Table 5.

Table 5. Cycle counts.

| Pure software implementation | Coprocessor implementation | Speedup ratio |
|------------------------------|----------------------------|---------------|
| 135963                       | 6079                       | 22.3          |
6.3. Application of Cifar-10 Image Classification

Cifar-10 data set classification is an open benchmark in machine learning, of which the task is to classify a group of 32 x 32-pixel RGB images. We also test the CNN in the practical applications using Cifar-10 image classification, including implementation of pure software algorithm and convolution coprocessor. The performance of the two types of implementation will be compared by the experimental results. The specific implementation process is presented in figure 8.

Table 6 shows the results of the convolution coprocessor-based and pure software algorithm-based implementations. Both of them can correctly complete the Cifar-10 image recognition and classification. Obviously, it can be seen from the run time that the convolution coprocessor has a certain acceleration effect.

![Figure 8. Specific implementation process.](image)

| Processing stages     | Coprocessor implementation | Pure software      | Speedup ratio |
|-----------------------|----------------------------|--------------------|---------------|
| Convolution1          | 4675254                    | 94595789           | 20.23         |
| ReLU1                 | 196627                     | 229393             | 1.17          |
| Pool1                 | 3424505                    | 3424505            | 1             |
| Convolution2          | 5629428                    | 74600939           | 13.25         |
| ReLU2                 | 24588                      | 28682              | 1.17          |
| Pool2                 | 423328                     | 423328             | 1             |
| Convolution3          | 1201082                    | 16861265           | 14.04         |
| ReLU3                 | 12304                      | 14350              | 1.17          |
| Pool3                 | 207353                     | 207353             | 1             |
| Fullconnect1          | 123102                     | 123102             | 1             |
| Softmax1              | 510                        | 510                | 1             |
| Total                 | 15918081                   | 190509216          | 11.97         |
7. Conclusions
In this paper, RISE-V processor is innovatively introduced into the realization of CNN, which considers the high efficiency and flexibility of CNN in embedded system. Combined with the great extensible characteristic, this coprocessor fully makes use of the complementary advantages of software and hardware development and improve the speedup ratio, which is a promising candidate applied to more application scenarios.

Acknowledgements
This research was supported by the Advanced Technology Program Foundation of National Grid (5442A1190007).

References
[1] Zhou Z, Chen X, Li E, Zeng L, Luo K and Zhang J 2019 Edge intelligence: Paving the last mile of artificial intelligence with edge computing Proceedings of the IEEE (107) 1738-1762.
[2] Strigl D, Kofler K and Podlipnig S 2010 Performance and scalability of GPU-based convolutional neural networks 18th Parallel, Distributed and Network-Based Processing pp 317-324.
[3] Zhang C, Li P, Sun G, Guan Y, Xiao B and Cong J 2015 Optimizing FPGA-based Accelerator design for deep convolutional neural networks Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays pp 161-170.
[4] Chen Y, Luo T, Liu S, Zhang S, He L, Wang J, Li L, Chen T, Xu Z and Sun N 2014 Dadiannaoo: A machine-learning supercomputer 47th Annual IEEE/ACM International Symposium on Microarchitecture pp 609-622.
[5] Nurvitadhi E, Venkatessh G, Sim J, Marr D, Huang R, Hock J O G, Liew YT, Srivatsan K, Moss D J M, Subhaschandra S and Boudoukh S 2017 Can FPGAs beat GPUs in accelerating next-generation deep neural networks? Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays pp 5-14.
[6] Gray J 2016 Grviphalanx: A massively parallel RISC-V FPGA accelerator accelerator IEEE Symposium on FPGAs for Custom Computing Machines pp 17-20.
[7] Waterman A, Lee Y, Avizienis R, Cook H, Patterson D A and Asanovic K 2013 The RISC-V instruction set IEEE Hot Chips Symposium pp 1-1.
[8] Yin J Y, Yang J, Yang H B and Du Z X 2019 A CNN accelerator on embedded FPGA using dynamic reconfigurable coprocessor Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing pp 1-5.
[9] Sankaradas M, Jakkula V, Cadambi S, Chakradhar S, Durukanovic I, Cosatto E and Graf H P 2009 A massively parallel coprocessor for convolutional neural networks 20th IEEE International Conference on Application-specific Systems, Architectures and Processors pp 53-60.
[10] Clere S R, Sethumadhavan S and Varghese K 2018 FPGA based reconfigurable coprocessor for deep convolutional neural network training Proceedings of IEEE 21st Euromicro Conference on Digital System Design pp 381-388.
[11] Wijeratne S, Jayaweera S, Dananjaya M and Pasqual A 2018 Reconfigurable co-processor architecture with limited numerical precision to accelerate deep convolutional neural networks IEEE 29th International Conference on Application-Specific Systems, Architectures and Processors pp 1-7.
[12] Li Z H, Hu W and Chen S 2019 Design and implementation of CNN custom processor based on RISC-V architecture IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems pp 1945-1950.
[13] Lou W Q, Wang C, Gong L and Zhou X H 2019 RV-CNN: Flexible and efficient instruction set for CNNs based on RISC-V processors International Symposium on Advanced Parallel Processing Technologies pp 3-14.
[14] Hubel D H and Wiesel T N 1968 Receptive fields and functional architecture of monkey striate cortex *The Journal of Physiology* (195) 215-243.

[15] LeCun Y, Boser B, Denker J S, Henderson D, Howard R E, Hubbard W and Jackel L D 1989 Handwritten digit recognition with a back-propagation network *Neural Computation* (1) 541-551.

[16] Krizhevsky A, Sutskever I and Hinton G E 2012 ImageNet classification with deep convolutional neural networks *Proceedings of the 25th International Conference on Neural Information Processing Systems* pp 1097-1105.

[17] Uchida K, Tanaka M and Okutomi M 2018 Non-blind image restoration based on convolutional neural network 2018 *IEEE 7th Global Conference on Consumer Electronics* pp 40-44

[18] Liu L, Shen C and Van Den Hengel A 2015 The treasure beneath convolutional layers: Cross-convolutional-layer pooling for image classification *Computer Vision and Pattern Recognition* pp 4749-4757.

[19] Naresh Babu K V and Edla D R 2017 New algebraic activation function for multi-layered feed forward neural networks *Iete Journal of Research* (63) 71-79.

[20] Roman A, Basu A, Mianjy P and Mukherjee A 2016 understanding deep neural networks with rectified linear units *International Conference on Learning Representations* pp 1-17.