Jurnal Teknologi

Full Paper

FPGA IMPLEMENTATION OF CNN FOR DEFECT CLASSIFICATION ON CMP RING

Ng Wai Kin, Mohd Shahrimie Mohd Asaari*, Bakhtiar Affendi Rosdi, Muhammad Firdaus Akbar

School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Engineering Campus, 14300, Nibong Tebal, Penang, Malaysia

Article history
Received
29 April 2021
Received in revised form
19 July 2021
Accepted
19 July 2021
Published online
20 August 2021

*Corresponding author
mohdshahrimie@usm.my

Abstract

Defect inspection is a crucial part of industrial manufacturing. However, it relies heavily on human effort on manual visual inspection. Various machine vision techniques have been introduced to replace human labour and to improve inspection quality and efficiency. The limitation of these techniques is that the algorithms need to be engineered again with each different use case. In this work, a Convolutional Neural Network (CNN) is used to classify the defects of the Chemical Mechanical Planarization (CMP) ring. The trained CNN model achieved an accuracy of 91% and the time taken for each inference process is around 1800 msec. To achieve computational efficiency, the CNN model is performed on the embedded device. The previous implementation of embedded CNN deploys OpenCL-based high-level synthesis accelerator on a high-end Field Programmable Gate Array (FPGA) board. In this work, the model inference is accelerated by PipeCNN FPGA implementation on Cyclone-VSE DE1-SoC, a low-end embedded FPGA board. Several configurations of hardware parameters are tested to search for the best setup of the FPGA resources. The hardware implementation has improved approximately seven times faster, as the inference time for each classification has improved from 1800 msec to 250 msec. However, the model implemented using the hardware is observed to produce lower inference accuracy as the accuracy drops from 91% to 81%. In conclusion, despite a slight decrease in accuracy, the implementation using FPGA manages to accelerate the inference performance of the CNN model up to 4 frames/sec, confirming the high potential of this approach to be used for high-throughput defect classification on CMP ring.

Keywords: Convolutional Neural Network, Deep learning, Field Programmable Gate Array, Defect Classification, Automatic visual inspection

Abstrak

Pemeriksaan kecacatan ialah perkara yang penting dalam industri pembuatan tetapi ia bergantung kepada usaha manusia dalam pemeriksaan visual secara manual. Kebanyakan teknik penglihatan mesin telah diperkenalkan untuk menggantikan buruh manusia dan pada masa yang sama dapat meningkatkan kualiti dan kecekapan pemeriksaan visual. Penggunaan teknik-teknik ini terbatas dari segi algoritmanya, yang mana ia perlu sentiasa direkabentuk untuk disesuaikan dengan setiap kes penggunaan yang berbeza. Dalam kajian ini, algoritma rangkaian saraf konvolusional (CNN) yang digunakan untuk mengelaskan kecacatan pada cincin pelarasan mekanikal kimia (CMP) telah dibina dengan menggunakan gambar kecacatan cincin CMP. Model yang dilatih telah mencapai ketepatan setinggi 91% dan masa yang diperlukan untuk setiap proses inferensi adalah sekitar 1800 milisaat. Bagi mencapai kecekapan pengiraan, model CNN dilaksanakan pada peranti terbenam. Terdahulu, pelaksanaan CNN tertanam adalah menggunakan pemecut sintesis peringkat tinggi berasaskan OpenCL pada papan tatasusunan get boleh aturcara medan (FPGA) kelas tinggi. Dalam kajian ini, inferensi model dipercepat oleh pelaksanaan PipeCNN FPGA pada Cyclone-VSE DE1-SoC, iaitu papan FPGA terbenam kelas rendah. Beberapa konfigurasi parameter perkakasan diuji untuk mencari persediaan sumber FPGA yang terbaik. Pelaksanaan perkakasan model tersebut telah menghasilkan peningkatan sekitar 700% dari segi prestasi inferensi, kerana masa inferensi untuk setiap pengelasan telah berkurang dari 1800 ms kepada 250 ms ketika dilaksanakan dengan papan DE1-SoC FPGA. Namun begitu, model yang dilaksanakan dengan perkakasan menghasilkan ketepatan inferensi yang lebih rendah apabila ketepatan menurun dari 91% sehingga 81%. Kesimpulannya, walaupun ketepatan menurun, implementasi menggunakan FPGA mampu mempercepatkan prestasi inferensi model CNN yang digunakan untuk klasifikasi kecacatan pada cincin CMP.

Kata kunci: Rangkaian Neural Konvolusional, Pembelajaran mendalam, Tatasusunan get boleh aturcara medan, Pengelasan kecacatan, Pemeriksaan visual automatik

1.0 INTRODUCTION

Chemical mechanical planarization (CMP) is a high-precision and complex wafer manufacturing process that removes material with chemical and mechanical forces [1]. A standard design of a CMP machine consists of a wafer carrier with a retaining ring (CMP ring) and a rotating polishing pad mounted on a rotatable platen. During the polishing, the CMP ring is used to secure the wafer in the rotating carrier and a downward force is applied to press the wafer against the rotating pad. To ensure that the wafer is always firm at the correct position during the polishing process, the applied CMP ring must be free of irregular defects to reduce the occurrence of micro-scratches on wafers. Therefore, to ensure the high quality and reliability of the CMP ring, surface inspection is one of the vital processes during the production lines.

Automated visual inspection systems [2, 3] are used in industrial manufacturing to detect possible defects on products. In these systems, defect detection is one crucial part to ensure the good quality of the finished product at the end of the manufacturing process through various inspection procedures. Prior to the introduction of machine vision technologies into the industry, surface quality control is done manually and the process is very time consuming, inefficient and might cause serious limitations to the production capacity of a manufacturing system. Automated visual inspection systems are designed to replace human effort in performing such tasks. Classical machine-vision methods have been used for many years and were sufficient to detect manufacturing flaws [4, 5]. However, with the Industry 4.0 paradigm, the trend is moving towards the generalization of the production line. Production lines are required to adapt rapidly to new products [6]. Classical machine-vision methods are not flexible and require the features to be handcrafted according to the respective knowledge domain. If given different tasks, where the object or defect to be recognized is different, development cycles tend to be longer when machine-vision methods must be manually adapted to different products.

Deep learning approaches provide much greater flexibility as the methods can be quickly adapted to new types of products and their respective defects. These approaches are proven to be able to impact the field of visual inspection very strongly as the prediction accuracy is promising and sufficiently reliable to be implemented in industrial manufacturing. In recent years, deep learning has become the most common approach to solve computer vision tasks [6]. Unlike conventional methods, the state-of-the-art deep learning approaches can learn from low-level data about its features and possesses a relatively higher capacity to represent complex structures, thus is gradually eliminating traditional handcrafted engineering methods. Furthermore, deep learning methods need only a reasonable amount of raw data to achieve accurate results and in some cases, the accuracy even exceeds human levels. This makes the deep learning approach practical for the industrial sector since the classification performance is reliable.

The goal of defect classification is to determine the presence of visual defects in digital imaging data and categorize them accordingly. It involves determining what type of defect is present. It is difficult to develop a software program or algorithm to instruct a computer on how to do it. There might be considerable variations in terms of the size of the defect, its orientation in the space, its attitude and its location. The task to efficiently classify surface defects is achievable by using Convolutional Neural Network (CNN) algorithms. CNN algorithms have been providing excellent performance on image classification tasks [7, 8]. However, the general-purpose computers used to execute these algorithms have limited computation resources and strict power consumption constraints [9 - 12]. The implementation of the CNN architecture of an object classification algorithm requires a large number of hardware resources. Therefore, hardware alternatives are needed to attain maximum performance in terms of execution speed. Furthermore, the execution of CNN models requires a large amount of matrix computations and therefore, general-purpose computers are not well adapted. Hardware alternatives such as GPU, FPGA and ASIC are experimented in various studies and have shown improvements to the implemented model in terms of performance.

A study [13] that investigates the power and throughput among CPU, GPU, FPGA and ASIC shows that although GPU is capable of producing high throughput, it requires high power consumption. On the other hand, FPGA can accelerate the inference process of CNN models at much lower power consumption. Some implementations are developed to perform CNN models on FPGA [15, 16, 20 - 22]. Using High-Level Synthesis (HLS) tools is one of the methods to implement a CNN model using FPGA circuits. HLS tools can synthesis coding in hardware description language from high-level language algorithms. This method effectively shortens the development cycle of designing an FPGA hardware circuit.

LeFlow [14] is an open-source tool that utilizes HLS tools to convert algorithms written in Tensorflow into synthesizable hardware. Google's Accelerated Linear Algebra (XLA) compiler is used to generate a Low-Level Virtual Machine-Intermediate Representation (LLVM-IR) computational graph description. The LLVM-IR is then processed using LegUp, an HLS tool, to generate Verilog hardware descriptions. Another common implementation of a

CNN model using FPGA hardware is using Open Computing Language (OpenCL). OpenCL is a framework designed to target program execution across heterogeneous platforms such as CPU, GPU, FPGA, etc. PipeCNN [15] is one of the implemented frameworks applied in OpenCL-based FPGA accelerator for large-scale CNN models.

A work proposed in [16] shows that the implemented method is effectively optimized CNN-based object detection algorithms on FPGA platforms. The architecture used in the design was formed with a convolutional kernel and a fully convolutional kernel. The object detection algorithms were evaluated across multiple hardware platforms, including x86 CPU, ARM CPU, FPGA and GPU. In [17], FPGAs are used to accelerate the CNN to perform image classification of different plant leaves. The work has developed an FPGA-based system which able to identify plants through leaf venations. The work adopted PipeCNN and OpenCL's methodology as the main frameworks to develop the hardware design.

This study aims to build a CNN defect classification model to recognize the type of defects on the CMP ring surface. Apart from that, this study focuses on the hardware accelerator implementation based on the FPGA platform. The performance prior to and after the FPGA implementation is evaluated to determine the significance of the hardware accelerator.

2.0 METHODOLOGY

This section covers the overall flow of the project. Generally, the project is divided into two major phases. In the first phase, the focus is on building a custom CNN model for defects classification on images of CMP ring. The latter phase involves implementing and testing the developed classification model on hardware module by exploiting FPGA as the CNN accelerator.

2.1 Data Preparation

The dataset preparation begins with collecting the defect images on the CMP ring. There are three common types of defects on CMP rings: scratch, dent, and burr. Scratch and dent are the defects found on the top-view of a CMP ring, while burr is the defect found on the side-view images of a CMP ring. Therefore, both the top-view image and side-view image are required for classifying the defect patterns on the CMP ring. The CMP ring images with no defect from both sides and top surfaces are also considered for reference purposes. Overall, three classes of images are classified from the top surface (i.e., scratch, dent and no defect) and two classes from the side surface (i.e., burr and no defect). For each class, there are 20 images collected, and therefore in total, the dataset consists of 100 images.

Figure 1 shows the image of a CMP ring and the examples of surface images with the defect and no defect pattern occur on top and side-view of the CMP ring. The collected images are scarce in numbers and relatively insufficient for training the CNN model. Therefore, the standard method of image augmentation is applied to reach a considerably ample amount of training data and improve data variations. The size of the original dataset is improved to reach 1000 images, where the original images are randomly augmented based on horizontal flipping, cropping and padding. Out of these 1000 images, 75% are assigned as training images, 15% are assigned as validation, and 10% are assigned as testing data.

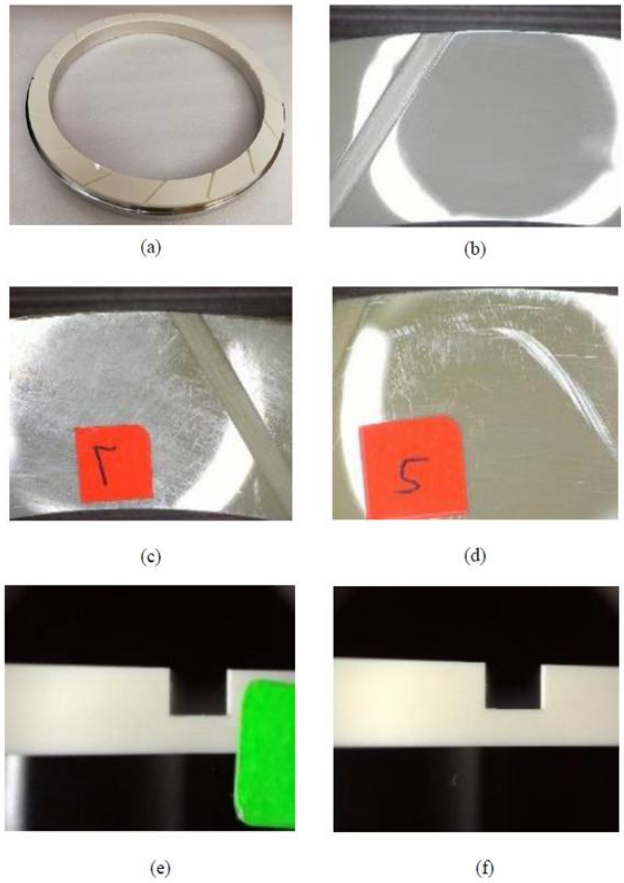

Figure 1 Images of CMP ring (a) and some examples of surface defects from the top (b-d) and side views (e-f)

2.2 CNN Model

CNN is a class of deep neural network models that has made a great breakthrough in image classification and retrieval [16 - 18], target detection [19, 23, 24] and so on. In CNN, the network is made of a sequence of layers, and every layer of CNN transforms one volume of activations to another through a differential function. In this work, AlexNet CNN architecture is used to develop the classification model of the surface defect on the CMP ring.

AlexNet is considered because of its obvious advantages, namely superior performance, less training parameters and strong robustness [18]. The architecture of AlexNet is comprised of five convolutional layers, three pooling layers, two fully connected hidden layers and one fully connected output layer as depicted in Figure 2.

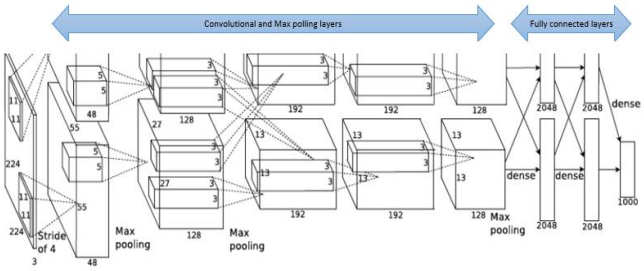

Figure 2 The framework of Alex-Net CNN architecture [18]

At the lower layers of AlexNet, the model learns feature extractors that resembled traditional filters. The fully connected layer at the last part of the AlexNet outputs to a 1000-way softmax that performs computation on a distribution for the 1000 class labels. In this work, Caffe deep learning framework is used to develop the CNN model. The default setting of AlextNet architecture is used, except that the original AlexNet that target for classification task involving 1000 classes are modified accordingly to match our task. Therefore, the last output layer is changed to 5, as we only target five defect types as described in Section 2.1. Within the Caffe framework, a solver definition is used to optimize the generated model during the training phase. The parameters in the solver definition include the number of test iteration, base learning rate, training and validation batch sizes, display and snapshot, which are all set to the default setting. During the training process, the training loss, validation loss and validation accuracy are recorded for every 100 iterations. Training loss is the probability that indicates how good or bad the learning model performs prediction based on the validation set. The smaller the training loss, the higher the probability of the model to predict correctly. The loss is calculated based on the cross-entropy measure

$$H(p,q) = -\sum_{c=1}^{M}(q_c \cdot log\ p_c) \quad (1)$$

where, M is the number of classes, p is model prediction and q is the class label.

2.3 FPGA Implementation of CNN Model

The CNN model is implemented on the FPGA development board (Cyclone-VSE DE1-SoC) using an OpenCL code and a host program. These components are built using the OpenCL development environment. The OpenCL code, which defines multiple parallel compute units in the form of kernel functions, is compiled and synthesized to run on the FPGA accelerator. In this implementation, the considered kernel function is based on the PipeCNN framework [15]. This kernel utilizes pipelined CNN functional kernels to achieve improved throughput in inference computation. The core part of the convolution layer is a 3-Dimensional multiply-accumulate operation that can be defined by

$$D_o(f_o,y,x) = \sum_{f_i=1}^{C_l}\sum_{k_y=0}^{K-1}\sum_{k_x=0}^{K-1} W_l(f_o,f_i,k_y,k_x) \cdot D_i(f_i,y+k_y,x+k_x) \quad (2)$$

where, $D_i(f_i,y,x)$ and $D_o(f_o,y,x)$ denotes the neuron at position (x,y) in the input feature map f_i and output feature map f_o, while $W_l(f_o,f_i,y,x)$ represents the corresponding weights in the l-th layer that gets convolved with f_i. In fully connected layers, computation of each output neuron is done by using the weighted summation of all input neurons as

$$D_o(f_o) = \sum_{f_i=0}^{C_l} W_l(f_o,f_i) \cdot D_i(f_i) \quad (3)$$

The architecture of PipeCNN consists of four major kernels, linked together by OpenCL extension channel or pipes as shown in Figure 3. The Convolution (Conv.) kernel performs both the 3-D multiply-accumulate operation of Eq. (2) and the inner product operation of Eq. (3). Subsampling is executed by the pooling kernel on the data streams generated by the Conv. kernel. MemRD and MemWR are the two data mover kernels involved in transferring the feature weights and data to and from the global memory, respectively. The cascaded kernels form a deep computation pipeline that allows the execution of a series of basic CNNs operations, hence can omit the process of storing interlayer data back to global memory. The Local Response Normalization (LRN) function is established separately from the pipeline since its input can be the data from adjacent feature maps or the same feature map, which requires a multitude of memory access patterns [15].

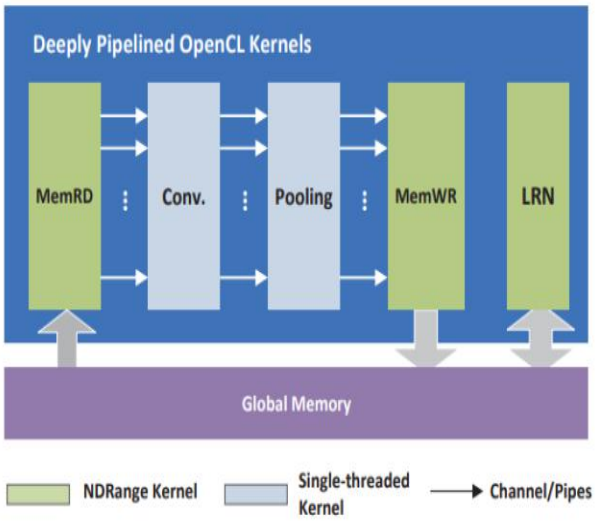

Figure 3 CNN Accelerator architecture in PipeCNN [15]

To use the OpenCL development environment on a host computer, the Intel FPGA software development kit (SDK) for OpenCL is needed and being setup beforehand. In addition, since the DE1-SoC board is an ARM-based device, the Intel SoC FPGA Embedded Design Suite (EDS) is required to compile an ARM-based host program. Minicom, the terminal emulator used to communicate with the board, is then installed in the host computer and configured according to the communication specification of the board.

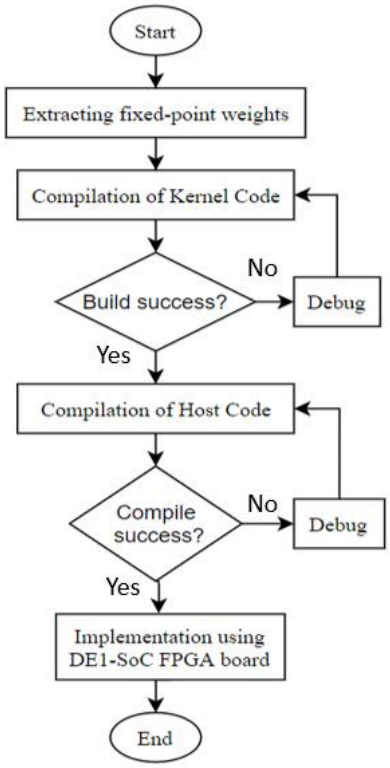

Figure 4 Process flow of CNN implementation on DE1-SoC FPGA

The workflow to implement the CNN model on DE1-SoC FPGA is shown in Figure 4. The trained CNN model weights are first quantized with 8-bit precisions before implementing on the DE1-SoC board. For the weights quantization, the fixed-point weights are assumed to be in the form of $N \times 2^{-m}$, where N is an integer with *n*-bit word length and *m* is fractional bits of the quantized weights. The process of extracting and quantizing the weights of the CNN model is done by using Matlab. The Fixed-Point Toolbox in Matlab is used to quantize the extracted weights according to the word lengths and fractional bit length. Finally, all the quantized weights and biases are combined and written in a single binary file.

The FPGA kernel is then developed using Quartus and OpenCL SDK. During the build process, the hardware resources utilization is displayed to the command log by including the -v option in the 'aoc' command in the OpenCL SDK. This is done to ensure that the hardware resource utilization does not exceed the available resources in the DE1-SoC FPGA board and causing the build to fail. Then, the host program is cross-compiled using the GCC cross-compiler of the Intel SoC EDS to facilitate the inference process. The host program first loads the trained CNN model and then loads the test images from given paths and resizes them to fit the model's input. Finally, the built kernel and compiled host program is transferred to the DE1-SoC board. The FPGA module on the SoC board is reprogrammed to execute inference on test images.

3.0 RESULTS AND DISCUSSION

The model is trained and tested using the allocated proportion of the dataset, as explained in Section 2.1. The training loss, validation loss, and validation accuracy are recorded at intervals of 100 iterations to obtain the best possible performance. A learning curve is plotted to visualize the model performance parameters. The training process is halted after 1600 iterations to prevent the model from overfitting as the training dataset is relatively scarce in number. Figure 5 shows the plot of the model loss and accuracy obtained from the independent training and validation images. After 400 iterations, the model has reached the accuracy of 88% and has been fluctuating at a very small interval between 75% and 90% and maintains since then. At 900 iterations, the validation loss is recorded at the lowest at around 0.22, and the validation accuracy reached the highest compared to other iteration numbers. Moving towards to 2000 iterations, the training loss seems to stabilize to reach low saturation level of 0.18; however, validation loss deviated to a higher loss, approximately at 0.4. On the other hand, the validation accuracy stabilized and did not recorded new improvement. the training results obtained at 900 iterations have the highest validation accuracy

with the lowest validation loss, this model is selected for the the final implementation model.

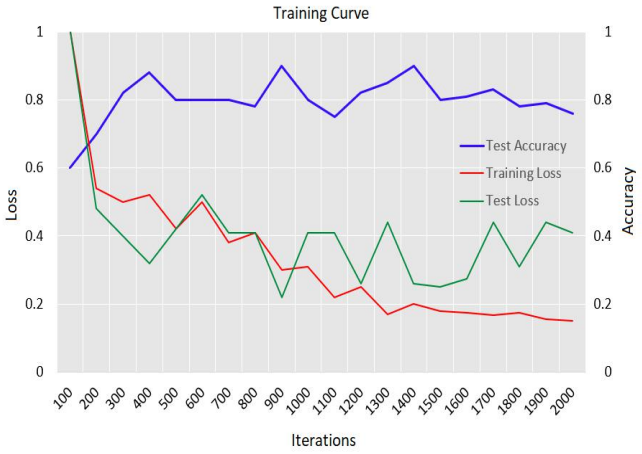

Figure 5 The model loss and accuracy plot obtained from the independent set of training and validation data

Table 1 shows the overall and breakdown of the classification accuracy of the selected CNN model when performed on five different classes of CMP ring images from the test dataset. The evaluation of the inference time for the software implementation is measured as the time interval between when an input image is loaded and when the CNN model produces classification results. This inference process is executed on 2.5 GHz Intel Core i5-7200 processor with 8 GB RAM machine. The average inference time for the model is around 1.828 seconds. This means that the software implementation executes the inference at the rate of 0.55 fps, making it virtually impossible to be deployed for industrial application that requires real-time processing.

Table 1 Classification result of the CNN model based on software implementation

Defect Class	True Positive	Actual Positive	Accuracy (%)
No Defect (Top)	20	20	100.0
Scratch	19	20	95.0
Dent	18	20	90.0
No Defect (Side)	17	20	85.0
Burr	17	20	85.0
Overall	91	100	91.0

The CNN model trained earlier is compiled into the kernel code and deployed on the DE1-SoC FPGA board, and the hardware inference is performed on the same test dataset. After testing with all the test images, the host program determines the correct predictions to calculate the test accuracy and display on the serial console. Table 2 summarizes the results of the hardware inference, where a final test accuracy of 81% is achieved. From Table 2, it is evident that the model accuracy decreases by 10% from 91% to 81% when switching from software implementation to hardware implementation. One of the reasons that cause this to happen can be related to the weights used in both implementations. Software implementation uses floating-point weights, while the FPGA uses fixed-point weights. The weights used in FPGA are quantized to 8-bit. This is done to save on the hardware resources needed for the CNN computation. This can cause the model accuracy to decrease as the CNN model relies heavily on the matrix computations using the network weights.

Table 2 Classification result of the CNN model based on hardware implementation

Defect Class	True Positive	Actual Positive	Accuracy (%)
No Defect (Top)	18	20	90.0
Scratch	17	20	85.0
Dent	16	20	80.0
No Defect (Side)	15	20	75.0
Burr	15	20	75.0
Overall	81	100	81.0

The implementation of the CNN model using PipeCNN requires the hardware parameters such as VEC_SIZE and LANE_NUM to be configured as part of the hardware setup of the FPGA resources. The different combinations of these parameters are experimented to determine the best possible configurations to obtain the best inference performance. Table 3 shows the results of FPGA inference tested on seven different configurations of PipeCNN parameters. The DE1-SoC FPGA board consisted of approximately 85,000 logic elements. The logic element utilization must not exceed the logic elements available to implement the CNN model using the FPGA hardware resources. Therefore, it is justifiable that the configuration E and F have failed to build as the logic utilization is too high. However, configurations B and G, with logic utilization of 96% and 87%, respectively also failed despite the well-constrained logic utilization. This is due to the usage of logic array blocks exceeds the available resources in the DE1-SoC FPGA. The successful state of hardware build is with the parameter configuration of A, C and D, which are then considered for further analysis. All three of the configurations yielded the same test accuracy, which is 81% when tested using the test images. In terms of the runtime, configuration A takes an enormous amount of time to execute the inference. This configuration probably did not utilize any parallelism and the input data is not buffered. On the other hand, C and D yielded relatively reasonable inference performances, with each takes 450 msec and 250 msec, respectively. At the same test accuracy, configuration D outperforms configuration C with around 200 msec shorter inference time. Hence, configuration D is chosen to

be implemented for hardware deployment. Therefore, for hardware implementation using the DE1-SoC FPGA board, the model has achieved a test accuracy of 81% and an inference speed of 4 fps.

Table 3 Performance of hardware inference based on different configurations of PipeCNN parameters

Conf	VEC SIZE	LANE NUM	Logic Utilization (%)	Build Status	Test Accuracy (%)	Inference time per image (msec)
A	4	8	77	S	81	5000
B	4	12	96	F	-	-
C	8	4	67	S	81	450
D	8	8	80	S	81	250
E	8	12	102	F	-	-
F	12	8	120	F	-	-
G	16	8	87	F	-	-

S = Success, F = Fail, Conf = Configuration

Referring to Tables 1 and 3, it is evident that the model accuracy decreases by 10% from 91% to 81% when switching from software implementation to hardware implementation. One of the reasons that cause this to happen is related to the weights used in both implementations. Software implementation uses floating-point weights while the FPGA uses fixed-point weights. The weights used in FPGA are quantized to 8-bit. This is done to save on the hardware resources needed for the CNN computation. It was expected that the configuration A, C and D are affected by similar floating-point weights patterns. This situation can cause the model accuracy to decrease as the CNN model relies heavily on the matrix computations using the network weights.

The accuracy of the developed CNN model can be further improved if more ground truth images are available. In difficult situations where the image data is not abundant, the accuracy of the CNN model will reduce significantly. Implementing the CNN model using FPGA is constrained by the board's hardware resources as the DE1-SoC FPGA board only contains around 85,000 logic elements. This limits the acceleration of the implemented model as the optimization of the model is greatly dependent on the available hardware resources to be utilized. It is also observed that there is a decline in the classification accuracy when the CNN model is implemented using FPGA due to the adaptation of fixed-point model weights into the hardware.

4.0 CONCLUSION

In this paper, a CNN model for classifying surface defects on CMP ring is developed and performed on hardware implementation. At the software level, the model has achieved a classification accuracy of 91%. When the model is implemented using the DE1-SoC FPGA board, the test accuracy slightly decreased to 81%. Despite the decrease in test accuracy, the hardware implementation demonstrated a much better inference performance in terms of inference speed. The hardware implementation has gained an improvement of around seven times faster than the software implementation. The improvement is significant as the model implemented in the hardware approach can of performing image classification at a rate of 4 fps. This shows that the FPGA implementation can accelerate computation-intensive CNN model inference and, therefore, allow a much efficient classification process. In the future, different hardware platforms with more hardware resources can also be considered to achieve better performance improvement in terms of both inference speed and accuracy. Apart from the classification task, a detection framework can also be integrated to detect the defects of CMP rings by exploring the most recent CNN algorithms, such as the Faster R-CNN algorithm.

Acknowledgement

The authors fully acknowledged School of Electrical and Electronic Engineering, Universiti Sains Malaysia for the support in term of hardware equipment and lab facility which makes this research work viable and effective.

References

[1] Sundararajan, S., Thakurta, D. G., Schwendeman, D. W., Murarka, S. P., and Gill W. N. 1999. Two-Dimensional Wafer-Scale Chemical Mechanical Planarization Models Based on Lubrication Theory and Mass Transport. *Journal of The Electrochemical Society*. 146 (2): 761-766. DOI: https://iopscience.iop.org/article/10.1149/1.1391678.

[2] Chin, R. T., and Harlow, C. A. 1982. Automated Visual Inspection: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 4(6): 557-573. DOI: https://doi.org/10.1006/cviu.1995.1017.

[3] Huang, S. H. and Pan, Y. C. 2015. Automated Visual Inspection in the Semiconductor Industry: A Survey. *Computers in Industry*. 66 (2015): 1-10. DOI: https://doi.org/10.1016/j.compind.2014.10.006.

[4] Tabernik, D., Šela, S., Skvarč, J., Skočaj, D. 2020. Segmentation-based Deep-learning Approach for Surface-defect Detection. *Journal of Intelligent Manufacturing*. 31(3): 759-776. DOI: https://link.springer.com/article/10.1007/s10845-019-01476-x.

[5] Wang, C. C., Jiang, B. C., Lin, J.Y. and Chu, C. C. 2013. Machine Vision-based Defect Detection in IC Images Using The Partial Information Correlation Coefficient. *IEEE Transactions on Semiconductor Manufacturing*. 26(3): 378-384. DOI: https://ieeexplore.ieee.org/document/6513319.

[6] Oztemel. E. and Gursev, S. 2020. Literature Review of Industry 4.0 and Related Technologies. *Journal of Intelligent Manufacturing*. 31(1): 127-182. DOI: https://link.springer.com/article/10.1007/s10845-019-01476-x.

[7] Zeiler, M. D. and Fergus, R. 2014. Visualizing and Understanding Convolutional Networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T., Editors. Computer Vision –

European Conference on Computer Vision. *Lecture Notes in Computer Science*. Springer. 818-833.
DOI: https://link.springer.com/chapter/10.1007/978-3-319-10590-1_53.

[8] Hijazi, S., Kumar, R., and Rowen, C. 2015. Using Convolutional Neural Networks for Image Recognition. *Technical Report*.
Online: http://ip.cadence.com/uploads/901/cnn-wp-pdf.

[9] Ghaffari, A. and Savaria, Y. 2020. CNN2GATE: Toward Designing a General Framework for Implementation of Convolutional Neural Networks on FPGA.
DOI: https://arxiv.org/abs/2004.04641.

[10] Lian, X., Liu, Z., Song, Z., Dai, J., Zhou, W., and Ji, X. 2019. High-performance FPGA-based CNN Accelerator with Block-Floating-Point Arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 27(8): 1874-1885.
DOI: 10.1109/TVLSI.2019.2913958.

[11] Shi, J., Tian, X., Zheng, Z., and Zhang, T. 2020. Application Research of CNN Accelerator Design Based on FPGA in ADAS. *IOP Conference Series: Materials Science and Engineering*. 768(7): 072014.
DOI: https://iopscience.iop.org/article/10.1088/1757-899X/768/7/072014/meta.

[12] Liu, B., Zou, D., Feng, L., Feng, S., Fu, P., and Li, J. 2019. An FPGA-based CNN Accelerator Integrating Depthwise Separable Convolution. *Electronics*. 8(3): *281*.
DOI: https://doi.org/10.3390/electronics8030281.

[13] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa R., Bates, S., Bhatia. S., Boden, N. *et al.* 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. *Proceedings of the 44th Annual International Symposium on Computer Architecture*, *Toronto, Canada*.
DOI: https://dl.acm.org/doi/10.1145/3079856.3080246.

[14] Noronha, D. H., Salehpour, B., and Wilton Steven, J. E. 2018. LeFlow: Enabling Flexible FPGA High-level Synthesis of Tensorflow Deep Neural Networks. *5th International Workshop on FPGAs for Software Programmers*. Dublin.
DOI: https://arxiv.org/abs/1807.05317.

[15] Dong, W., Jianjing, A., and Ke Xu. 2016. PipeCNN: An OpenCL-based FPGA Accelerator for Large-scale Convolution Neuron Networks.
DOI: https://arxiv.org/abs/1611.02450.

[16] Zhao, R., Niu X., Wu Y., Luk W., and Liu, Q. 2017. Optimizing CNN-based Object Detection Algorithms on Embedded FPGA Platforms. In: Wong S., Beck A., Bertels K., and Carro L., Editors. *Applied Reconfigurable Computing. ARC 2017. Lecture Notes in Computer Science*. Springer. 255-267.
DOI: https://link.springer.com/chapter/10.1007/978-3-319-56258-2_22.

[17] Linsangan, N. B., and Pangantihon, Jr R. S. 2018. FPGA-Based Plant Identification through Leaf Veins. *Proceedings of the 2018 5th International Conference on Biomedical and Bioinformatics Engineering*, *Okinawa, Japan*.
DOI: https://dl.acm.org/doi/10.1145/3301879.3301905.

[18] 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*, *Lake Tahoe, Nevada, USA*.
DOI: https://dl.acm.org/doi/10.1145/3065386.

[19] Wang, J., and Zheng, T., Lei, P., and Bai, X. 2019. A Hierarchical Convolution Neural Network (CNN)-based Ship Target Detection Method in Spaceborne SAR Imagery. *Remote Sensing*. 11(6): 620.
DOI: https://doi.org/10.3390/rs11060620.

[20] Phan-Xuan, H., Le-Tien, T. and Nguyen-Tan, S. 2019. FPGA Platform Applied for Facial Expression Recognition System Using Convolutional Neural Networks. *Procedia Computer Science*. 151: 651-658.
DOI: 10.1016/j.procs.2019.04.087.

[21] Anderson, J. H., Brown, S. D., Canis, A. and Choi, J. 2013. High-level Synthesis with Legup: A Crash Course for Users and Researchers. *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate, Monterey California USA*.
DOI: https://doi.org/10.1145/2435264.2435269.

[22] Suda, N., Chandra, V., Dasika, G., Mohanty, A., Ma, Y., Vrudhula, S., and Seo, J. S., and Cao, Y. 2016. Throughput-Optimized Opencl-based FPGA Accelerator for Large-scale Convolutional Neural Networks. *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, *Monterey California USA*.
DOI: 10.1145/2847263.2847276.

[23] Cai, Z, and Fan, Q., and Feris, R. S., and Vasconcelos N. 2016. A Unified Multi-scale Deep Convolutional Neural Network for Fast Object Detection. *European Conference on Computer Vision*.
DOI: https://link.springer.com/chapter/10.1007/978-3-319-46493-0_22.

[24] Kim, H., Lee, Y., Yim, B., Park, E. and Kim, H. 2016. On-Road Object Detection Using Deep Neural Network. *IEEE International Conference on Consumer Electronics-Asia, Seoul, South Korea*.
DOI: https://ieeexplore.ieee.org/document/7804765.