ICIC Express Letters
Part B: Applications
Volume **13**, Number **7**, July **2022**

ICIC International ©2022 ISSN 2185-2766
pp. 767–776

PERFORMANCE OPTIMIZATION OF MAXPOOL CALCULATION USING 4D RANK TENSOR

Suryadiputra Liawatimena[1,2,3,*], Edi Abdurahman[1], Agung Trisetyarso[1]
Antoni Wibowo[2], Ivan Sebastian Edbert[2], Muhamad Keenan Ario[2]
and Fandy Effendi[2]

[1]Computer Science Department, BINUS Graduate Program – Doctor of Computer Science
[2]Computer Science Department, BINUS Graduate Program – Master of Computer Science
[3]Computer Engineering Department, Faculty of Engineering
Bina Nusantara University
Jl. K. H. Syahdan No. 9, Kemanggisan, Palmerah, Jakarta 11480, Indonesia
{ edia; ivan.edbert; muhamad.ario; fandy.effendi }@binus.ac.id; { atrisetyarso; anwibowo }@binus.edu
*Corresponding author: suryadi@binus.edu

Received September 2021; accepted December 2021

Abstract. *Convolutional neural networks (CNN) are deep learning algorithms usually used to process input images by focusing on different aspects or objects in order to recognize one image from another. They also have a pooling layer which is normally used to handle and avoid unnecessarily complicated calculations in order to speed up the processing time. This research, therefore, proposes a novelty approach to improving the performance of the pooling layer process using a 4D rank tensor. The layer was generally formed using a 4 × 4 window size or 2 × 2 filter which was later operated on each slice of the input. Meanwhile, the maxpool calculation with the 4D rank tensor requires more elaboration for its application in the CNN algorithm. This method is faster by 221.01 milliseconds or 27.99%.*
Keywords: Convolutional neural network, Pooling layer, Maxpool, 4D rank tensor

1. **Introduction.** Convolutional neural network is one of the machine learning approaches usually applied for visual object recognition [1,2]. It has fewer connections and parameters and is also easy to train when compared to the other standard neural networks [3]. This method relies on and uses many convolution processes in 2D (two dimensions) through the application of arrays. Moreover, CNN has become efficient in studying data due to its ability to extract low-level features as well as to detect and classify objects [4]. Some of its inherent architectures include LeNet [5,6], AlexNet [3], ZFNet [7], GoogleNet [8], VGGNet [9], and ResNet [10]. Several studies have, however, been conducted to develop new models with better accuracy in recognizing numbers and images. Image is the main way for people to obtain original information [11]. Furthermore, it has been discovered that CNN requires less initial processing compared to other classification algorithms and its architecture is also divided into two parts which are the feature extraction layer and fully connected layer. The feature extraction layer is the process of encoding an image into a number that represents the image. This involved the utilization of the first convolutional layer to remove several low-level features such as the edges, corners, and lines from the input image [12]. An example of this is the pooling layer which is also known as subsampling and down-sampling which has the ability to discard 75% of information without reducing and affecting the content [13]. Pooling operation also shrinks feature map resolution and preserves critical discriminant information required for recognition.

DOI: 10.24507/icicelb.13.07.767

Furthermore, this method reduces the number of neural connections and the size of the feature map [14] and also discards irrelevant image details.

Figure 1 shows the process involved in the reduction of the spatial dimension which further leads to the loss of information. Meanwhile, kernel size and filter number can significantly affect the performance [16]. A convolutional layer is usually followed by a pooling layer. Now, the model formed becomes deeper and more complex. Of course, as the number of layers increases, it will take more time to process. Therefore, this research proposes an approach to improving the performance of the pooling layer process using a 4-dimensional array through the Halide concept [17]. This method was later tested with an input test image. In this study, researcher will increase the speed, especially modification in max pooling method.

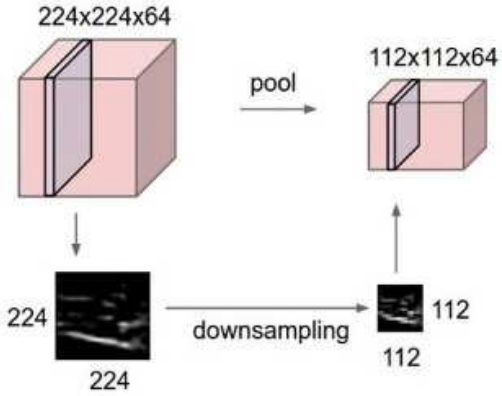


FIGURE 1. Downsampling with pooling layer [15]

2. Background Information and Related Works.

2.1. Parallel in Halide.

Halide is a domain-specific language (DSL) designed for fast image processing and computational photograph [18]. It allows quick operation without any benchmarking, considers a single level of tiling and fusion, and also combines with fixed heuristic choices for several purposes such as parallelism, vectorization, and unrolling. The main idea in Halide is the separation of a program into the algorithm which specifies the data to be computed, scheduled, and dictates the order of computation and storage. Moreover, the algorithm is expressed as a purely functional and feed-forward pipeline of arithmetic operation on multidimensional grids. This helps the user not to worry about the low-level optimizations while writing the high-level algorithm. Furthermore, Halide is also required to express iterative or recursive computations such as summation, histogram, and scan [17].

Halide has recently enabled the implementation of a high-performance image processing pipeline. Li et al. conducted an experiment [18] to extend the ability of this method to automatically and efficiently compute the gradients of arbitrary Halide programming using reverse-mode automatic differentiation. Moreover, this method writes codes in few lines while PyTorch and CUDA need more codes. Halide programming language has also proven to the trial and writing of different schedules easy towards achieving high performance. Adams et al. also fused several parallel loops into a single one to avoid the slight overhead involved in nested parallelism [19].

2.2. Tensor.

Tensors are uniform multidimensional arrays with a uniform type [20] and their numbers are represented by the modes or ways and orders [21]. It is important to note that scalar mathematics is when the required index is 0, a vector is when it is 1, a matrix is when it is 2, and a tensor is when the index is 3 or more as shown in Table 1. The different types of tensors are, however, presented in Figure 2.

TABLE 1. Scalar, vector, matrix, tensor

Index	Computer science	Mathematics
0	(0D) number	Scalar
1	(1D) array	Vector
2	(2D) array	Matrix
n	(nD) array	Tensor

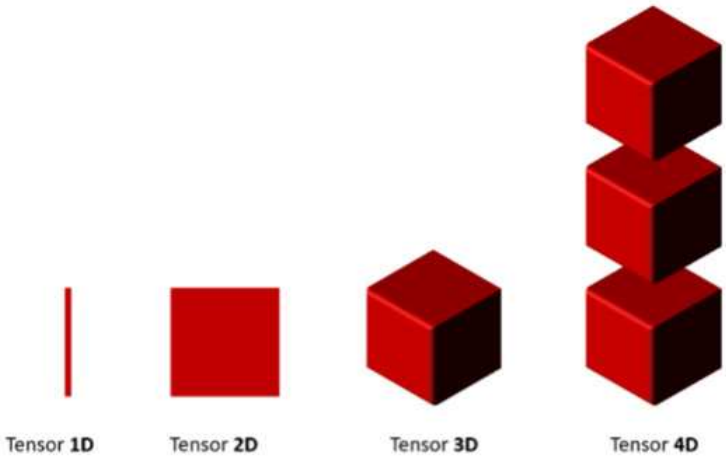


FIGURE 2. Tensors 1D, 2D, 3D, and 4D

Some of the vocabularies usually used in tensors include [20]

- Shape: the length of each of the axes of a tensor
- Rank: number of tensor axes
- Axis or dimension: a particular dimension of the tensor
- Size: the total number of items in the tensor or the product shape of the tensor

Sparse tensors have been observed to be increasingly used in data analysis and machine learning. A high-performance vector product known as SpTV is one of the most fundamental operations to process sparse tensors [21]. It is also important to note that tensors can be unfolded and represented as matrices, and their sparsity has the ability to influence load balance.

Liawatimena et al. [22] proposed a novel way to improve the convolutional process using a 4-dimensional array using the concepts of Halide and 4D rank tensor. This method produced fewer loops and this means it is faster compared to the traditional way of convolution by saving approximately 18.5% of the time.

2.3. **Big O notation.** Big O notation is a mathematical notation which describes the limiting function's behavior when its argument tends towards a specific value or infinity [23]. It was invented as Bachmann-Landau or asymptotic notation by Paul Bachmann, Edmund Landau, and others. The notation is commonly referred to as "Big O squared", and written as $O(n^2)$ notation where n is used to represent the size of the input.

Algorithm analysis is an essential part of computational complexity theory [24] which provides theoretical estimates for resources such as space and time devoted to solving a given computational problem. However, time complexity describes the amount of time it takes an algorithm or a program to complete its process or execution and is usually expressed using Big Omega (Ω) and Big Theta (Θ). Figure 3 shows the complexity of Big O [25] which is marked with different colors where red means very complex and green means not complex.

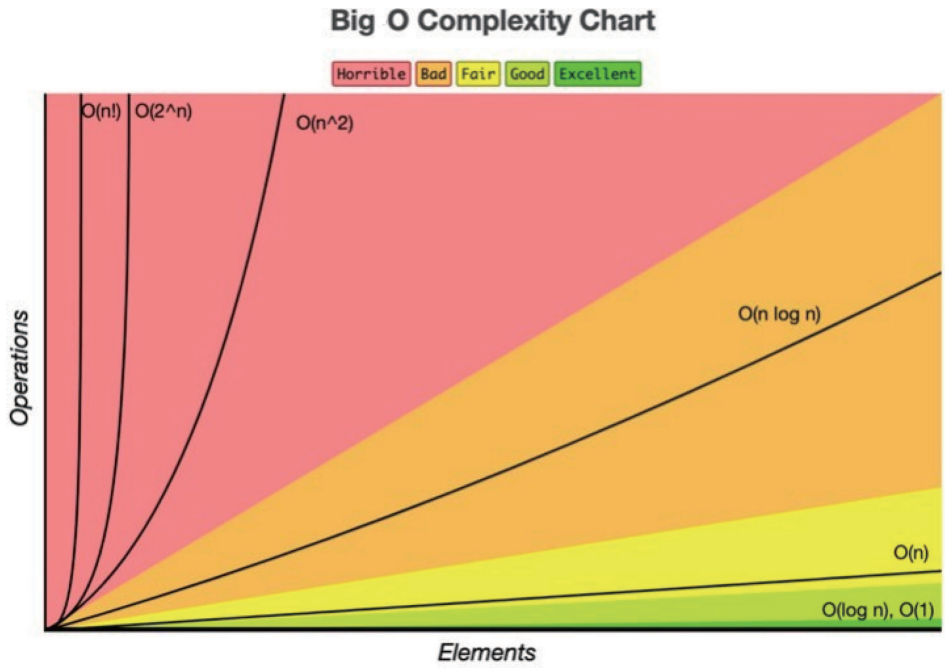


FIGURE 3. (color online) Big O complexity graph

3. **Method.** According to Giusti et al. [26], the benefit of using a pooling layer is associated with its ability to handle complications and avoid unnecessary calculations in order to speed up the processing time. This process is needed to produce a faster result for object detection and image classification. However, the most essential aspect of CNN modeling is the selection of the types of pooling layers to be used [27] due to its ability to act on each stack in the feature map and reduces its size. It is generally formed using a 2 $\times$ 2 filter applied with a stride of 2 and operates on each slice of the input. For example, an image of size 224 $\times$ 224 pixels is expected to have an output size of 112 $\times$ 112 using a 2 $\times$ 2 pooling size with a stride of 2 as shown in Figure 1. Moreover, the input image dimension is represented as W $\times$ H $\times$ D where W is width, H is the height, and D is the depth. The image dimensions produced after the application of the pooling layer are, therefore, shown in Equations (1) and (2).

$$NewWidth = \frac{(InputWidth - FilterWidth)}{Stride} + 1 \tag{1}$$

$$NewHeight = \frac{(InputHeight - FilterHeight)}{Stride} + 1 \tag{2}$$

Figure 4 shows the traditional method of maxpool in the pooling layer which uses 4 $\times$ 4 size considered as the four maxpools of a 2 $\times$ 2 filter. Meanwhile, the method proposed was based on the multiplication from this traditional maxpool.

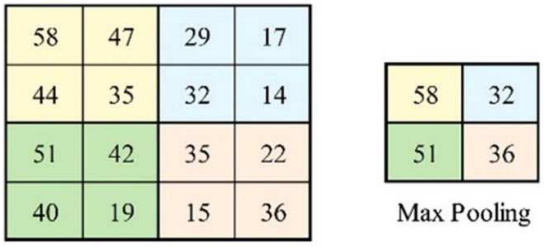


FIGURE 4. Output feature after max-pooling application

Figure 5 shows the multiplication of the traditional method into 16×16 size to produce 16 maxpools with a 2×2 filter. The number of values produced from the process increased four times compared to the traditional method. Moreover, the traditional Maxpool2D process is indicated in Figure 6 and the layer with one input/output feature map and pseudocode is observed. A 2×2 maxpooling kernel extracts the maximum value from a 2×2 region on feature map 1 to feature map 2 and discards other values assuming the pooling layer is connected to one input and one output feature map. This paper proposes Maxpool2D4DT algorithm as shown in Figure 6.

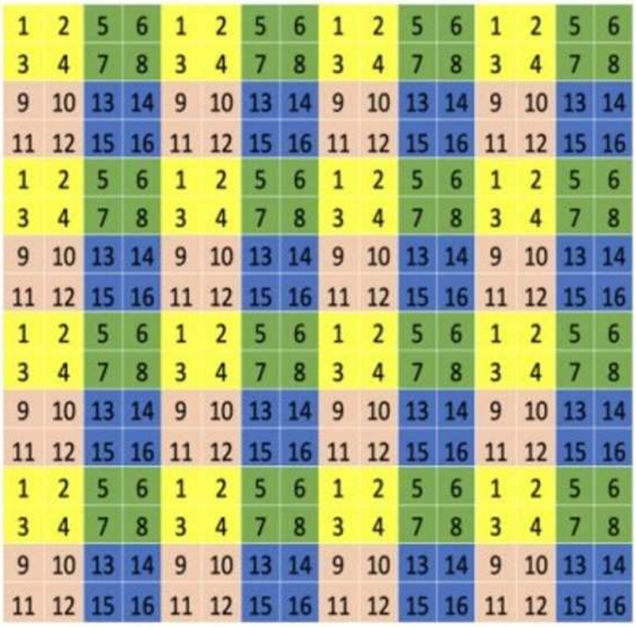


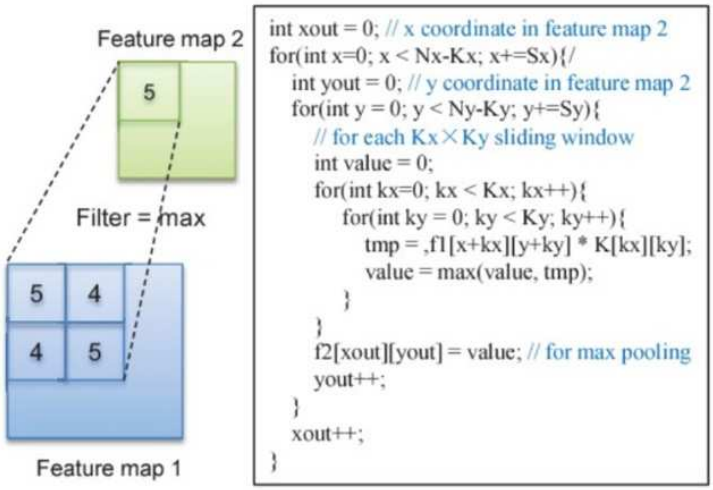
FIGURE 5. 16 maxpool in a 2-axis tensor shape $[16, 16]$

A method was proposed to calculate the maxpool and a tile block consisting of 16 maxpool 4D tensor was called in line with $[1, 4, 4, 2, 2]$. This size was explained using its 1 column maxpool, 4 rows maxpool, a depth of 4, and a filter size of 2×2 as indicated in Figure 7.

The calculation process of the proposed method is presented in Figures 8 and 9 and the first four rows and four columns were observed to be maxed pool with a 2×2 filter with a 2-axis shape tensor. This was called a one maxpool. A traditional maxpool process with a 2×2 filter usually produces 4 maximum values, but the proposed method produced 16 values in one single process.

The method was called Maxpool2D4DT which means maxpool with 2×2 filter and 4D-ranked tensor model. However, the difference between this method and the traditional maxpool algorithm is the number of iterations. For example, there is a 16×16 window size with a filter of 2×2, and the traditional maxpool needs at least 16 iterations to obtain 16 values while the method developed requires just one single process to obtain these values. This means it speeds up the process to obtain maximum values using only one iteration. Moreover, it does not need to re-calculate the remaining columns and this also shows it is very faster than the usual method. It is important to note that the method was designed based on a Halide concept known as the parallelization function.

4. **Results.** The method developed was run using python and the process was compared with the usual method. The results presented in Figure 10 show the output size was 208×208 which is half of the original size [28,29]. This was, therefore, followed by edge detection and comparison with the traditional method.




```
for maxpoolRow in range(0, totalRow, tileMaxpool):
    for maxpoolCol in range(0, totalCol, tileMaxpool):

        if maxpoolCol == 516:
            print ("\n LAST COLUMN!!")

        # Get a tileBlock consists 16x16
        lastPart = image[maxpoolRow:maxpoolRow + tileMaxpool, maxpoolCol:maxpoolCol + tileMaxpool]
        blr, blc = np.shape(lastPart)  # get the shape of the lastPart
        if (blr != tileMaxpool) or (blc != tileMaxpool):  # Check if we are the last block
            aBlock = np.zeros((tileMaxpool,tileMaxpool))
            aBlock[0:blr, 0:blc] = lastPart    # copy the lastPart into 16x16
        else:
            aBlock = lastPart

        if wantDebug:
            print("\nmaxPool #{}, maxpoolRow {},  maxpoolCol {}, aBlock = \n {}".format(m, maxpoolRow, maxpoolCol, aBlock))

        # We process each cell. 16 values are calculated at a time
        for cellRow in range(cellMaxpool):
            for cellCol in range(cellMaxpool):
                if wantDebug:
                    print("\nmaxpoolRow {}, maxpoolCol {}, cellRow {}, cellCol {}".format(maxpoolRow, maxpoolCol, cellRow, cellCol))
                for row in range(filterMaxpool):
                    for col in range(filterMaxpool):
                        # Take oneCell, find itsMax
                        sourceRow  = (row * filterMaxpool) + (cellMaxpool * cellRow)
                        sourceRow2 = (row * filterMaxpool) + (cellMaxpool * (cellRow+1))
                        sourceCol  = (col * filterMaxpool) + (cellMaxpool * cellCol)  # 0, 4, 8, 12
                        sourceCol2 = (col * filterMaxpool) + (cellMaxpool * (cellCol+1))  # 0, 4, 8, 12416
                        oneCell = aBlock[sourceRow:sourceRow2, sourceCol:sourceCol2]

                        targetRow = maxpoolRow//2 + (row * cellMaxpool + cellRow)
                        targetCol = maxpoolCol//2 + (col * cellMaxpool + cellCol)
                        itsMax = np.max(aBlock[sourceRow:sourceRow2, sourceCol:sourceCol2])
                        if wantDebug:
                            print("maxPool #{}, oneCell[{},{}] = \n {} \ntemp[{},{}] = {}".format(m, sourceRow, sourceCol, oneCell, targetRow, targetCol, itsMax))
                        temp[targetRow, targetCol] = itsMax
                        m = m + 1

out2 = temp[0:Nh, 0:Nw]
print("\ntotal maxPool2dt4d = ", m)
print(out2.shape)
return out2
```

FIGURE 6. Maxpool2D and Maxpool2D4DT algorithm

Figure 11 shows the Maxpool2D and Maxpool2D4DT process time and in average the Maxpool2D4DT 221.01 ms or 27.99% faster than the Maxpool2D. Maxpool2D4DT is faster because our method calculates the value faster with fewer iterations than Maxpool2D. Our method can get the value faster because we have calculated the value before. As we have mentioned before in Section 3, our method does not re-calucalte the column that already has been calculated. This concept is called parallelization in Halide.

5. **Conclusion.** This research describes the process of developing max pooling using a python programming language. It was discovered that the proposed method was able to save up to 27.99% of the time for several resolution input images when compared with the original maxpool method. This means it is faster because there is no need to focus on the remaining columns. It is also important to note that the difference of 546.09 milliseconds

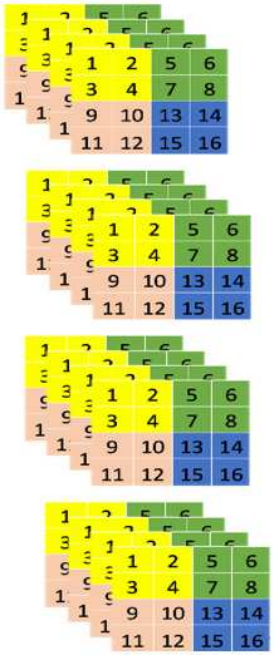


FIGURE 7. Tile block of a 16 maxpool in 4-axis shape

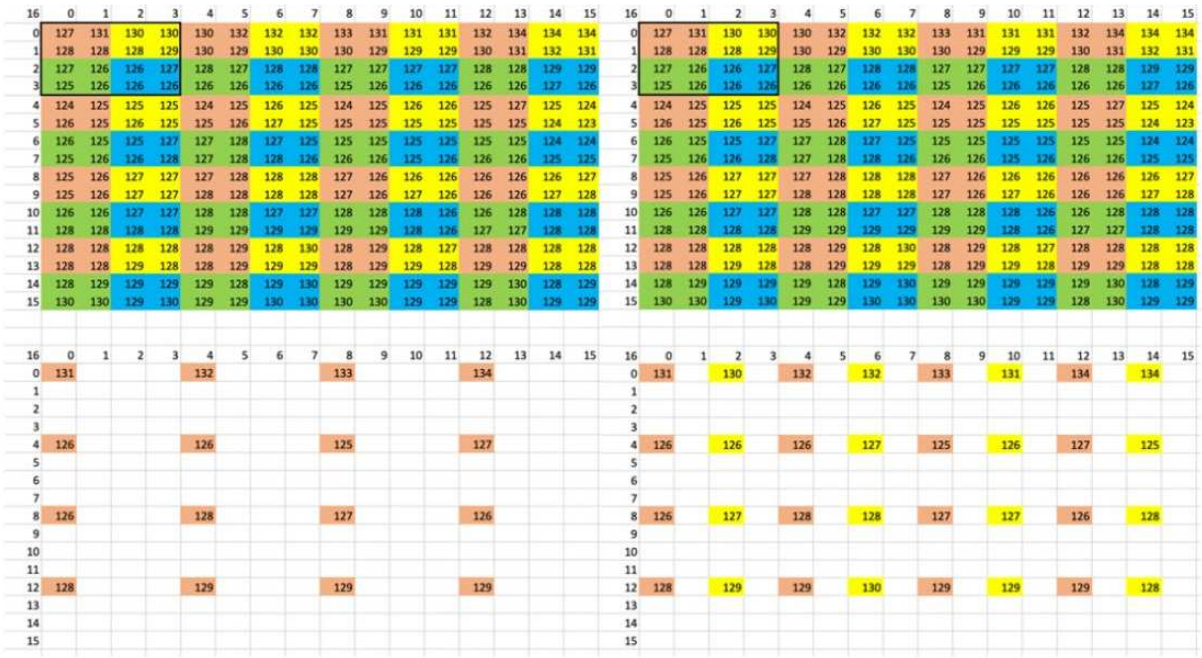


FIGURE 8. (color online) Maxpool2D4DT processes red and yellow cell

was recorded for one-time use of the maxpool layer. Meanwhile, the training model usually requires thousands of iterations depending on the amount of the data and the accuracy to be achieved. This means a model normally requires more than one maxpool layer and each convolutional layer is usually followed by a pooling layer. The model is also observed to be currently getting deeper, thereby, making the number of layers reach hundreds. Therefore, it is recommended that further studies increase the calculation at the inner looping from 16 to 64 values at a single-cell process in order to calculate 256 values altogether in a block.

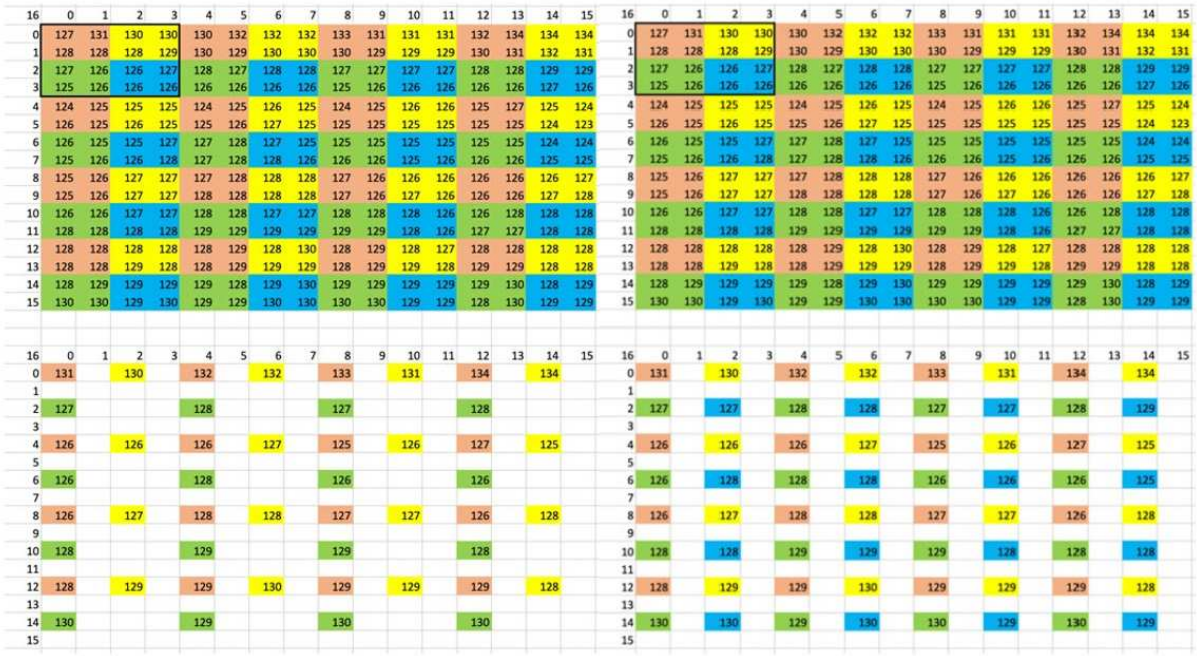


FIGURE 9. (color online) Maxpool2D4DT processes green and blue cell

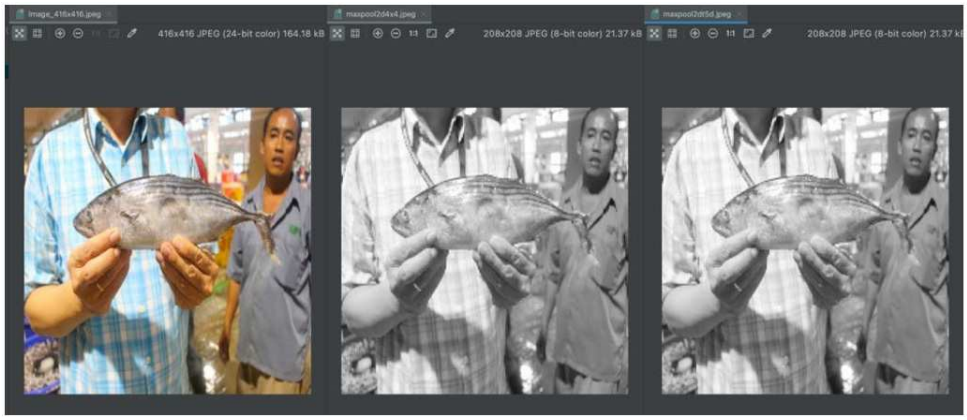


FIGURE 10. Maxpool2D and Maxpool2D4DT output

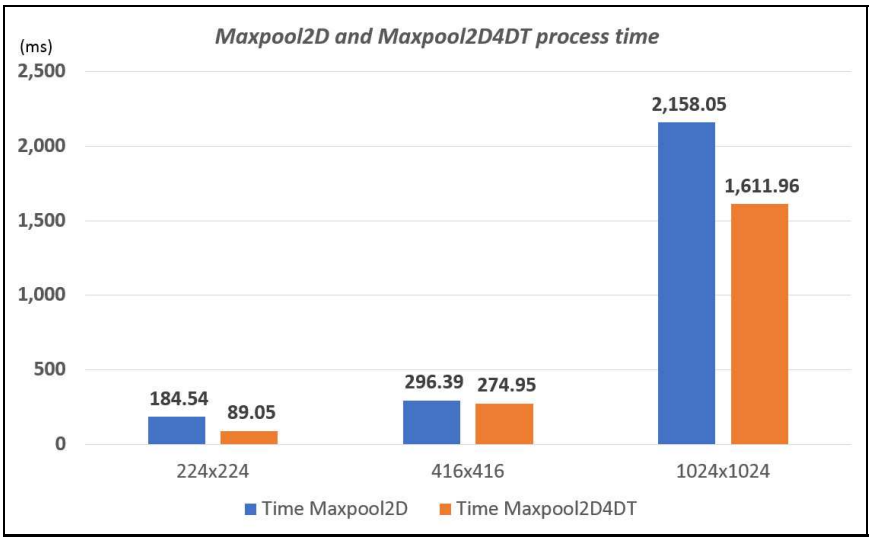


FIGURE 11. Maxpool2D and Maxpool2D4DT process time

REFERENCES

[1] S. Albawi, T. A. M. Mohammed and S. Alzawi, *Layers of a Convolutional Neural Network*, IEEE, 2017.

[2] Anderies, B. A. Jabar, R. Yunanda and A. A. S. Gunawan, The development of a smart door decision system, based on pir sensor, embedded face recognition and server request using TTGO ESP 32, *ICIC Express Letters, Part B: Applications*, vol.12, no.10, pp.965-970, 2021.

[3] A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet classification with deep convolutional neural networks, *Communications of the ACM*, vol.60, no.6, pp.84-90, 2017.

[4] Y. LeCun, Y. Bengio and G. Hinton, Deep learning, *Nature*, vol.521, pp.436-444, 2015.

[5] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition, *Proc. of IEEE*, vol.86, no.11, pp.2278-2323, 1998.

[6] W. Cui, Q. Lu, A. M. Qureshi, W. Li and K. Wu, An adaptive LeNet-5 model for anomaly detection, *Inf. Secur. J.*, vol.30, no.1, pp.19-29, 2021.

[7] L. Fu, Y. Feng, Y. Majeed et al., Kiwifruit detection in field images using Faster R-CNN with ZFNet, *IFAC-PapersOnLine*, vol.51, no.17, pp.45-50, 2018.

[8] Z. Zhong, L. Jin and Z. Xie, High performance offline handwritten Chinese character recognition using GoogLeNet and directional feature maps, *Proc. of Int. Conf. Doc. Anal. Recognition (ICDAR)*, pp.846-850, 2015.

[9] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, *arXiv.org*, arXiv: 1409.1556, 2015.

[10] Z. Wu, C. Shen and A. van den Hengel, Wider or deeper: Revisiting the ResNet model for visual recognition, *Pattern Recognit.*, vol.90, pp.119-133, 2019.

[11] J. Si, W. Sun and Y. Cheng, Image denoising using low rank matrix completion via bilinear generalized approximate message passing, *International Journal of Innovative Computing, Information and Control*, vol.16, no.5, pp.1547-1558, 2020.

[12] R. Kumar, S. Joshi and A. Dwivedi, CNN-SSPSO: A hybrid and optimized CNN approach for peripheral blood cell image recognition and classification, *Int. J. Pattern Recognit. Artif. Intell.*, vol.35, no.5, 2021.

[13] N. Akhtar and U. Ragavendran, Interpretation of intelligence in CNN-pooling processes: A methodological survey, *Neural Comput. Appl.*, vol.32, no.3, pp.879-898, 2020.

[14] J. Chen, Z. Hua, J. Wang and S. Cheng, A convolutional neural network with dynamic correlation pooling, *The 13th Int. Conf. Comput. Intell. Secur.*, no.2, pp.1-4, 2017.

[15] A. Karpathy, *CS231n Convolutional Neural Networks for Visual Recognition*, https://cs231n.github.io/convolutional-networks/, 2018.

[16] A. Agrawal and N. Mittal, Using CNN for facial expression recognition: A study of the effects of kernel size and number of filters on accuracy, *Vis. Comput.*, vol.2, 2019.

[17] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand and S. Amarasinghe, Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines, *Proc. of ACM SIGPLAN Conf. Program. Lang. Des. Implement.*, pp.519-530, 2013.

[18] T. M. Li, M. Gharbi, A. Adams, F. Durand and J. Ragan-Kelley, Differentiable programming for image processing and deep learning in Halide, *ACM Trans. Graph.*, vol.37, no.4, 2018.

[19] A. Adams et al., Learning to optimize Halide with tree search and random programs, *ACM Trans. Graph.*, vol.38, no.4, 2019.

[20] A. Singh, *Introduction to Tensors*, https://www.tensorflow.org/guide/tensor, 2019.

[21] Y. Chen, G. Xiao, M. T. Ozsu, C. Liu, A. Y. Zomaya and T. Li, AeSpTV: An adaptive and efficient framework for sparse tensor-vector product kernel on a high-performance computing platform, *IEEE Trans. Parallel Distrib. Syst.*, vol.31, no.10, pp.2329-2345, 2020.

[22] S. Liawatimena, E. Abdurahman, A. Trisetyarso et al., Convolve4D: A novelty approach to improve convolutional process, *IOP Conf. Ser. Earth Environ. Sci.*, vol.794, no.1, DOI: 10.1088/1755-1315/794/1/012107, 2021.

[23] S. Huang, *What is Big O Notation Explained: Space and Time Complexity*, https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/, 2020.

[24] Randerson, *Algorithm Analysis & Time Complexity Simplified*, https://randerson112358.medium.com/algorithm-analysis-time-complexity-simplified-cd39a81fec71, 2017.

[25] E. Rowell, *Know Thy Complexities!*, https://www.bigocheatsheet.com/, 2012.

[26] A. Giusti, D. C. Cireoan, J. Masci, L. M. Gambardella and J. Schmidhuber, Fast image scanning with deep max-pooling convolutional neural networks, *Proc. of 2013 IEEE Int. Conf. Image Process (ICIP2013)*, pp.4034-4038, 2013.

[27] C. Y. Lee, P. W. Gallagher and Z. Tu, Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree, *Proc. of the 19th Int. Conf. Artif. Intell. Stat. (AISTATS2016)*, pp.464-472, 2016.
[28] J. Redmon and A. Farhadi, YOLO9000: Better, faster, stronger, *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.7263-7271, 2017.
[29] J. Redmon and A. Farhadi, *YOLO v.3*, Technical Report, pp.1-6, 2018.