Convolve4D: A Novelty Approach to Improve Convolutional Process

Suryadiputra Liawatimena\textsuperscript{1,2,3}, Edi Abdurahman\textsuperscript{1}, Agung Trisetyarso\textsuperscript{1}, Antoni Wibowo\textsuperscript{2}, Wiedjaja Atmadja\textsuperscript{2}, Fandy Effendi\textsuperscript{3}, Ivan Sebastian Edbert\textsuperscript{3}

\textsuperscript{1}Computer Science Department, BINUS Graduate Program - Doctor of Computer Science, Bina Nusantara University, Jakarta, Indonesia, 11480
\textsuperscript{2}Computer Engineering Department, Faculty of Engineering, Bina Nusantara University, Jakarta, Indonesia 11480
\textsuperscript{3}Computer Science Department, BINUS Graduate Program – Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia 11480

Corresponding author: suryadi@binus.edu

Abstract. Convolutional neural networks (ConvNet or CNN) are deep learning algorithms that can process input images, assign meaning to various aspects or objects in the image (biases and learnable weight) and recognize one image from another. The bigger kernel size will take more time to process the input. We present a novelty way to use a 4D rank tensor to improve a convolutional process. At the early stage of the Convolve4D development, the edge detection with 3x3 kernel and The Laplacian of Gaussian (LoG) with 5x5 kernel size was used to demonstrate the convolutional process improvement. The Convolve4D needs more elaboration to be used into a CNN algorithm. The advantage of convolve4D is only need 9 loops to calculate 81 outputs, whereas convolve2D need 9 x 9 x 3 x 1 x 7 x 7 = 11,907 loops. The result is 18.5\% shorter when using a 5x5 kernel; it reduces from 0.54 seconds to 0.44 seconds for the edge detection convolution process.

Keywords: Convolutional Layer, Convolutional Neural Network, Tensor, Laplacian of Gaussian, Edge Detection.

1. Introduction
Since Lecun, Bottou, Bengio, & Haffner (1998) introduced Convolutional Neural Network (ConvNet or CNN) with the LeNet to learn and recognize digits. There are many CNNs sprang out, such as AlexNet\textsuperscript{[1]}, VGG\textsuperscript{[2]}, GoogLeNet\textsuperscript{[3]}, ResNet\textsuperscript{[4]} and others more to come. There is a significant increase in accuracy for recognizing numbers and images afterward. As shown in Figure 1, to achieve smaller errors, they use more layers, or the ConvNet became deeper.
Figure 1. The performance and layer numbers of the ConvNets[5]

CNNs are deep learning algorithms that can process input images, assign meaning to various aspects or objects in the image (biases and learnable weight) and recognize one image from another. Much less initial processing is required in ConvNet compared to other classification algorithms. While in the primitive method, the filters are hand-developed and trained, ConvNets can learn these filters and properties. The operation of the ConvNet architecture resembles the patterns of neurons found in the human brain and is inspired by the cortex's visual organization. Each neuron responds to stimuli only in a limited area of the visual field known as the receptive field[6]. According to Agrawal & Mittal (2019), kernel size, and the number of filters significantly impact the network's accuracy. However, the bigger kernel size, it will take more time to process. With the increasing number of layers on ConvNet, the convolution process will be carried out more and more, increasing processing time. In this paper, we propose a novelty approach to improve the convolutional process using a 4-dimensional array.

2. Methods

As shown in Figure 2, the traditional convolutional process is one pixel at a time—the algorithm to compute each cell of the output by multiplying the receptive field with a kernel.

Figure 2. Traditional convolutional process

\[ O_{k,i,j} = \sum_{c=0}^{Ch} \sum_{h=0}^{KH} \sum_{r=0}^{KW} (w_{k,c,h,r} \times x_{i+h,j+r,c}) + b_k \]  

(1)

Where:
- \( c \) is the number of channels for a 2-dimensional image (Red, Green, and Blue channels)
- \( H \) and \( W \) are for the height and width of the image
- \( H_k \) and \( W_k \) are the height and width of the kernel. \( r \) is from 0 up to \( W_k \).
- \( O \) is the output
- \( w \) is the weight
- \( k \) is the kernels
- \( x \) is the input image
- b is the bias

Using the illustration as shown in Figure 2, where \( r = 3, h = 3, c = 3, k = 1, j = 4 \) and \( i = 4 \) then the inner looping will take 432 times. ImageNet is a dataset of more than 15 million high-resolution images that are categorized into 22,000. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) uses a subset from ImageNet of around 1000 images in every 1000 categories. In total, there are 1.2 million training images, 50,000 validation images, and 100,000 test images.[7][8]. If we use a 224x224 resolution for an image, then one convolutional operation will take \( 3 \times 3 \times 3 \times 1 \times 224 \times 224 = 1.330.668 \) steps. That is only for one kernel where the AlexNet uses 96 kernels; it will take 127.744.128 steps. This enormous number is for one convolutional layer where AlexNet has five convolutional layers, as shown in Figure 3.

![Figure 3. The AlexNet Architecture [1]](image)

Tensor is a multidimensional matrix whose contents are of the same type. When the required index is 0, it is called a scalar, an index of 1 is called a vector, an index of 2 is called a matrix, and an index of more than three is called a tensor, as shown in Table 1. There is an idea to take advantage of a tensor with a 4-dimensional array.

| Indexes Required | Computer Science | Mathematics |
|------------------|------------------|-------------|
| 0                | (0D)number       | Scalar      |
| 1                | (1D) Array       | Vector      |
| 2                | (2D) Array       | Matrix      |
| n                | (n D) Array      | Tensor      |

![Figure 4 shows a tensor with a 4-Dimensional array. We create from a 3x3 kernel into a 9x9 called a TileKernel, then reshape it into a 4-Dimensional array, see Figure 5. The upper cube is for row 0, the middle cube is for row 1, and the bottom cube is for row 2; in each cube, there are three layers for column 0 to column 2. In each layer, there is a 2-Dimension matrix of 3x3 kernel.](image)
As shown in Figure 5, we found the kernel when the column are equal to 0, 3, 6 in Covolve2D, we can process at once use the TileKernel. To demonstrate the work of our Convolve4D we use the edge detection kernel3 as np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]]) and kernel5[10].

```python
# Edge Detection Kernel 3x3
kernel3 = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])

# Edge Detection Kernel Lg6 5x5
kernel5 = np.array([[9, 0, -1, 0, 0],
                    [9, -2, -2, -2, -1],
                    [-1, -2, 16, -2, -1],
                    [-1, -2, -2, -1, 0],
                    [9, 0, -1, 0, 0]])
```

Figure 6. The Kernels and the Convolve4D process
As shown in Figure 6, the Convolve4D process are as follows: when the column is 0, we compute 9 output together for (0,0), (0,3), (0,6), (3,0), (3,3), (3,6), (6,0), 6,3) and (6,6) in one loop. Then we move the TileKernel by one column and compute (0,1), (0,4), (0,7), (3,1), (3,4), (3,7), (6,1), 6,4) and (6,7). Next, the TileKernel at column is 2, we compute (0,2), (0,5), (0,8), (3,2), (3,5), (3,8), (6,2), 6,5) and (6,8). After that, we move the column to 9, and the process repeats with the previous pattern. When the column is at $x_{ImgPadded} - x_{KernShape} - 1$, we compute the last column output. After that, we increase the row with 1 and reset the column with 0. The same concept with column operation, after finish calculates output at the row 2, we jump the row to 9. The advantage of convolve4D is only need 9 loops to calculate 81 outputs, whereas convolve2D need $9 \times 9 \times 3 \times 1 \times 7 \times 7 = 11,907$ loops.

Our program is shown in Figure 7.

3. Results
We run the python program to compare the convolve2D and convolve4D on a Windows 10 operating system notebook using Intel Core i7 with 16 MB RAM; we get the result shown in Figure 8, and the convolutional performances in Figure 9.
Figure 8. Input image, Convolve2D result, and Convolve4D result

```plaintext
D:\Python37\python.exe D:\learn\belajarconv2d\main28.py
Image size = 345 x 345
ImagePadded size = 349 x 349
Process convolve2D, 3x3 = 0.51 seconds
Process 39 blocks with total 353 pixels
Process convolve4D, 3x3 = 0.47 seconds

======================
Image size = 345 x 345
ImagePadded size = 349 x 349
Process convolve2D, 5x5 = 0.54 seconds
Process 14 blocks with total 354 pixels
Process convolve4D = 0.44 seconds
```

Figure 9. The convolutional performances

From the Python program output, we summarize the result in Table 2. The larger the kernel size, the Convolve2D method takes the longer the process, and the Convolve4D method takes less time. There was a time savings of 7.8% up to 18.5% for a 3x3 and 5x5 kernel, respectively.

Table 2. Comparison of convolution methods process time

| Kernel Size | Convolve2D (seconds) | Convolve4D (seconds) | Difference (seconds) | Difference (%) |
|-------------|----------------------|----------------------|----------------------|----------------|
| 3x3         | 0.51                 | 0.47                 | 0.04                 | 7.8%           |
| 5x5         | 0.54                 | 0.44                 | 0.10                 | 18.5%          |

4. Conclusion

This paper described the development process of the Convolve4D using the Python program. With a 345x345 resolution input image, the Convolve4D can improve a convolutional process by 18.5%. Further work, the research will include the stride parameter in the program. Currently, the stride is equal to 1.
5. References

[1] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems*, pp. 1097-1105, 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv.org*, p. arXiv:1409.1556, 2014.

[3] K. Z. A. Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv*, p. arXiv:1409.1556, 2014.

[4] K. He, X. Zhang, S. Ren and J. Sun, "Very deep convolutional networks for large-scale image recognition," in *the IEEE conference on computer vision and pattern recognition*, 2016.

[5] J. Wang, J. Zhang, W. Bao, X. Zhu, B. Cao and P. S. Yu, "Not just privacy: Improving performance of private deep learning in mobile cloud," in *The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[6] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks," Towards Data Science, 16 December 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed 12 October 2020].

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2009, 2009.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," in *International Journal of Computer Vision (IJCV)*, 2015.

[9] Anonymous, "Introduction to Tensors," 2019. [Online]. Available: https://www.tensorflow.org/guide/tensor.

[10] A. Campilho, "Edge Detection," 2008. [Online]. Available: http://web.fe.up.pt/~campilho/PDI/NOTES/EdgeDetection.pdf.

[11] 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," *The Computer Vision*, p. 405–412, 2019.

[12] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of IEEE*, 1998.