International Journal of Computer Applications Technology and Research
Volume 13–Issue 06, 18 – 23, 2024, ISSN:-2319–8656
DOI:10.7753/IJCATR1306.1003

Real Time 2D Convolution and Max Pooling Process

Panca Mudjirahardjo
Dept. of Electrical Engineering
Faculty of Engineering
Universitas Brawijaya
Malang, Indonesia

Abstract: In Convolutional Neural Network (CNN) consists of process of 2D convolution and max pooling. 2D convolution is performed to express the shape of an object in an image. Max pooling is one way to reduce the spatial dimensions of an input volume. They are together to create an object's feature. As we know, the feature extraction is an important part in classification and detection task. A good feature can distinguish the shape of one object from another. It will increase the classification and detection accuracy. In this paper, researcher will build and observe the real time 2-D convolution and max pooling process for feature extraction.

Keywords: real time; 2D convolution; max pooling; feature extraction; CNN

1. INTRODUCTION

As we delve into the inner workings of Convolutional Neural Networks, we encounter two fundamental processes: 2D Convolution and Max Pooling. These operations play a crucial role in extracting meaningful features from input images, paving the way for robust pattern recognition and classification. 2D Convolution and Max Pooling process, which are fundamental operations in Convolutional Neural Networks (CNNs) used for feature extraction and down-sampling.

2D Convolution, a cornerstone of CNNs. Imagine a pristine canvas representing our input image, brimming with pixels waiting to reveal their secrets. In the world of convolution, kernels—small, learnable filters—act as brushes, sweeping across the image to uncover distinctive patterns.

As the kernel traverses the image, it performs a dot product between its weights and the corresponding pixel values in each local region. This process yields a new value—a feature—that encapsulates the essence of the underlying structure within that region. With each stroke of the kernel, the canvas transforms, revealing hidden edges, textures, and shapes.

Max Pooling, like a magnifying glass, scans through the feature maps generated by convolution, selecting the most salient elements within localized regions. It achieves this by selecting the maximum value within each window, discarding irrelevant details and retaining only the most prominent features. Through this process, our canvas undergoes a subtle transformation, shrinking in size while amplifying the significance of its contents. A conceptual model of CNN is shown in Figure 1 [1]. Creating of feature extraction is shown in Figure 2 [2].

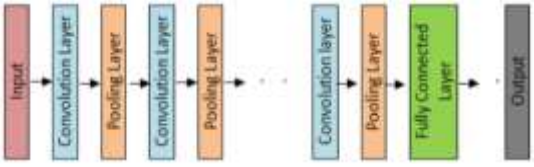


Figure 1. Conceptual model of CNN [1]

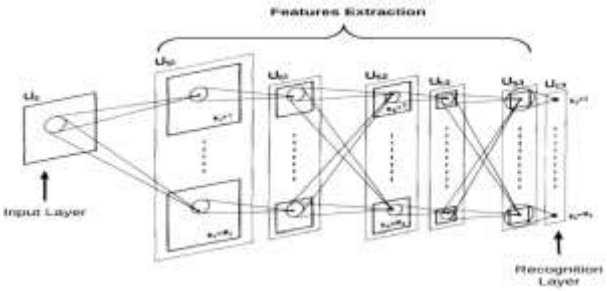


Figure 2. Schematic diagram illustrating the interconnections between layers in the neocognitron [2]

2. THE PROPOSED METHOD

The experimental method can be shown in Figure 3. A Camera is a device to capture the object(s) image. Then, the RGB input image is converted into grayscale. The contrast limited adaptive histogram equalization (CLAHE) is performed to the grayscale image to improve the image's contrast. This CLAHE image is filtered with a filter kernel in 2D convolution process, to express how the shape of the input image is modified by a filter. Finally, the max pooling is performed to reduce the spatial dimensions of an input volume.

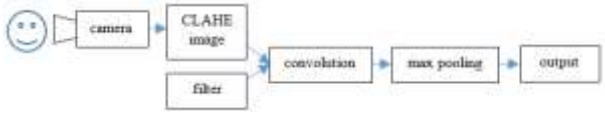


Figure 3. The experimental method

2.1 Contrast Limited Adaptive Histogram Equalization (CLAHE)

Adaptive histogram equalization (AHE) is a computer image processing technique used to improve contrast in images. It differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image [1][3][4].

International Journal of Computer Applications Technology and Research
Volume 13–Issue 06, 18 – 23, 2024, ISSN:-2319–8656
DOI:10.7753/IJCATR1306.1003

However, AHE has a tendency to over amplify noise in relatively homogeneous regions of an image. A variant of adaptive histogram equalization called contrast limited adaptive histogram equalization (CLAHE) prevents this by limiting the amplification. The one implementation of CLAHE is used for improve the visibility level of foggy image or video [4].

In CLAHE, the contrast amplification in the vicinity of a given pixel value is given by the slope of the transformation function. This is proportional to the slope of the neighborhood cumulative distribution function (CDF) and therefore to the value of the histogram at that pixel value. CLAHE limits the amplification by clipping the histogram at a predefined value before computing the CDF. This limits the slope of the CDF and therefore of the transformation function. The value at which the histogram is clipped, the so-called clip limit, depends on the normalization of the histogram and thereby on the size of the neighborhood region. Common values limit the resulting amplification to between 3 and 4.

It is advantageous not to discard the part of the histogram that exceeds the clip limit but to redistribute it equally among all histogram bins [5-8].

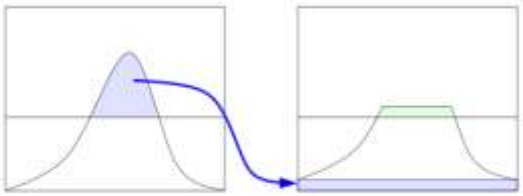


Figure 4. The histogram distribution in CLAHE [5]

The redistribution will push some bins over the clip limit again (region shaded green in the Figure 4), resulting in an effective clip limit that is larger than the prescribed limit and the exact value of which depends on the image. If this is undesirable, the redistribution procedure can be repeated recursively until the excess is negligible.

The CLAHE algorithm has three major parts: tile generation, histogram equalization, and bilinear interpolation. The input image is first divided into sections. Each section is called a tile. Histogram equalization is then performed on each tile using a pre-defined clip limit. Histogram equalization consists of five steps: histogram computation, excess calculation, excess distribution, excess redistribution, and scaling and mapping using a cumulative distribution function (CDF). The histogram is computed as a set of bins for each tile. Histogram bin values higher than the clip limit are accumulated and distributed into other bins. CDF is then calculated for the histogram values. CDF values of each tile are scaled and mapped using the input image pixel values. The resulting tiles are stitched together using bilinear interpolation, to generate an output image with improved contrast.

To increase image contrast, use the CLAHE algorithm as below. Grayscale and color photos can both be processed using this approach.

CLAHE algorithm steps are as follows [9]:

Step 1: Input image
Step 2: Segment input images into tiles
Step 3: Compute histogram for each tiles
Step 4: Apply TFM to compute clip limit
Step 5: Limit the contrast based on computed clip limit
Step 6: Check for enhanced image
Step 7: Enhanced image

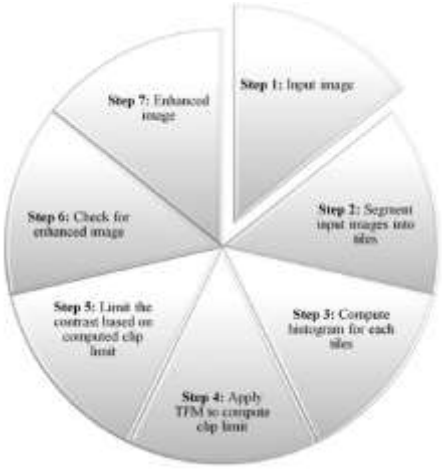


Figure 5. Steps followed in CLAHE algorithm [9].

Figure 5 illustrates the steps to be followed in the CLAHE algorithm. Prior to creating a histogram for each context region, a given input image is first separated into context regions. So that various portions of the image may be easily linked, a mapping function is used to produce an image mapping. The image noise is subsequently reduced using an interpolation approach. This enables us to lessen the noise in particular regions of the image. Although the method denoises the image, it does not do so fully.

2.2 Filter Kernel

A filter kernel, often simply referred to as a kernel or a filter, is a small matrix of weights used in convolutional operations, particularly in image processing and computer vision tasks. The kernel acts as a window or a template that is systematically applied to an input image to perform operations such as feature extraction, blurring, sharpening, edge detection, and more.

Key characteristics of a filter kernel include:

1. **Size**: The size of the kernel determines the spatial extent of the features it detects or the type of operation it performs. Common sizes include 3x3, 5x5, and 7x7 kernels, although larger or smaller kernels can also be used depending on the specific task.
2. **Weights**: Each element in the kernel matrix represents a weight that determines the contribution of the corresponding pixel in the input image to the output result. These weights are often learned during the training process in neural networks or manually defined for specific image processing tasks.
3. **Center**: The center element of the kernel matrix is typically aligned with the pixel being processed in the input image during convolution operations. The weights of the kernel are applied to the surrounding pixels in the input image to compute the output value for that pixel.
4. **Functionality**: The values in the kernel matrix define a mathematical operation that is applied to the input image. For example, in edge detection, the kernel may be designed to highlight areas of rapid intensity change, while in blurring, the kernel may apply a smoothing effect by averaging neighboring pixel values.

Examples of commonly used filter kernels include:

1. **Identity Kernel**: A 3x3 kernel with a center value of 1 and all other values set to 0. It preserves the original image without any modification.

2. **Gaussian Kernel**: A 2D Gaussian distribution used for blurring or smoothing images. It assigns higher weights to central pixels and lower weights to surrounding pixels, resulting in a blur effect.
3. **Sobel Kernels**: A pair of 3x3 kernels used for edge detection. One kernel highlights vertical edges, while the other highlights horizontal edges.
4. **Laplacian Kernel**: A 3x3 kernel used for edge detection and image sharpening by emphasizing regions of rapid intensity change.

Filter kernels are versatile tools that enable a wide range of image processing and feature extraction tasks, playing a crucial role in the success of convolutional operations in various computer vision applications. Some of filters are shown in Figure 6 [10].

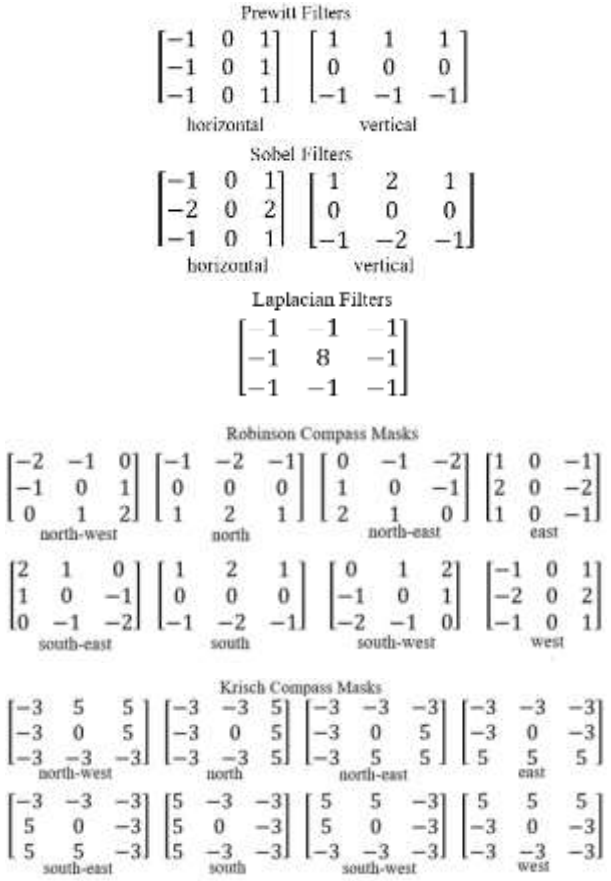


Figure 6 Some of filter kernel [10].

2.3 2D Convolution

2D convolution is a fundamental operation in image processing and computer vision, widely used in Convolutional Neural Networks (CNNs) for feature extraction. It involves applying a filter, also known as a kernel or a mask, to an input image to produce a feature map that highlights specific patterns or features.

Here's how 2D convolution works:

1. **Input Image**: The process begins with an input image represented as a two-dimensional grid of pixels, where each pixel contains grayscale intensity values or color channels (e.g., red, green, blue).
2. **Filter or Kernel**: A filter or kernel is a small matrix of weights that slides over the input image. The size of the kernel determines the spatial extent of the features it detects. For example, a 3x3 kernel captures local features, while larger kernels capture more global patterns.
3. **Convolution Operation**: The kernel is convolved with the input image by sliding it over the image and computing the element-wise multiplication between the kernel and the corresponding pixel values in the image patch covered by the kernel. These products are summed up to produce a single value, which becomes the corresponding pixel value in the output feature map.
4. **Stride and Padding**: The kernel moves across the input image with a specified step size called the stride. Padding may also be applied to the input image to preserve its spatial dimensions during convolution, ensuring that the output feature map has the same spatial dimensions as the input image.
5. **Output Feature Map**: As the kernel slides over the input image, it computes the convolution operation at each position, generating a new grid of values known as the output feature map. Each value in the feature map represents a local feature or pattern detected in the corresponding region of the input image.

The convolution operation captures various types of features such as edges, textures, and shapes present in the input image. By learning appropriate filter weights during training, convolutional layers in CNNs can automatically extract hierarchical representations of visual features, enabling the network to perform tasks like image classification, object detection, and segmentation.

2D convolutions, a convolution generalized to matrices, are useful in computer vision for a variety of reasons, including edge detection and convolutional neural networks. Their exact usage will not be discussed here, and instead we will discuss an efficient way to calculate a 2D convolution with the FFT we have already developed. We have a “data” matrix, representing an image, and we have a kernel matrix, which is the matrix we imagine sliding over the image. This is also known as a filter [11][12][13].

For 2D convolutions, the result is slightly ambiguous depending on how one defines it. We will use scipy’s definition, where to calculate the value of the convolution at a particular point, we imagine the bottom right corner of the kernel placed over that point.

We define the 2D convolution between an image x of size $M \times N$ and a kernel h of size $H \times W$ as follows (similar to the 1D case, we assume both matrices are padded with 0’s):

$$(x * h)[i,j] = \sum_{k=0}^{i} \sum_{l=0}^{j} x[k][l]h[i-k][j-l] \quad (1)$$

This operation is also symmetric, so what we call the image and the kernel is essentially arbitrary (by convention, the kernel is the smaller matrix). The resulting matrix is going to be of size $(M + H - 1) \times (N + W - 1)$ from the same logic as the 1D case. Thus, the time it takes to compute the convolution is $O(MNHW)$. We can, however, take advantage of a trick if the kernel has a certain property.

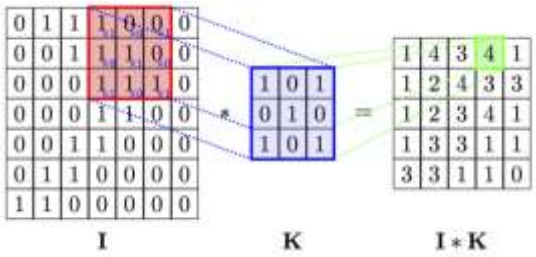


Figure 7. A convolution process [11]

In summary, 2D convolution plays a crucial role in extracting meaningful features from input images, providing the foundation for advanced computer vision algorithms and applications. Its ability to capture local patterns and spatial relationships enables machines to perceive and interpret visual information with remarkable accuracy and efficiency.

2.4 Pooling Layer

The pooling layers are used to sub-sample the feature maps (produced after convolution operations), i.e. it takes the larger size feature maps and shrinks them to lower sized feature maps. While shrinking the feature maps it always preserve the most dominant features (or information) in each pool steps. The pooling operation is performed by specifying the pooled region size and the stride of the operation, similar to convolution operation [1].

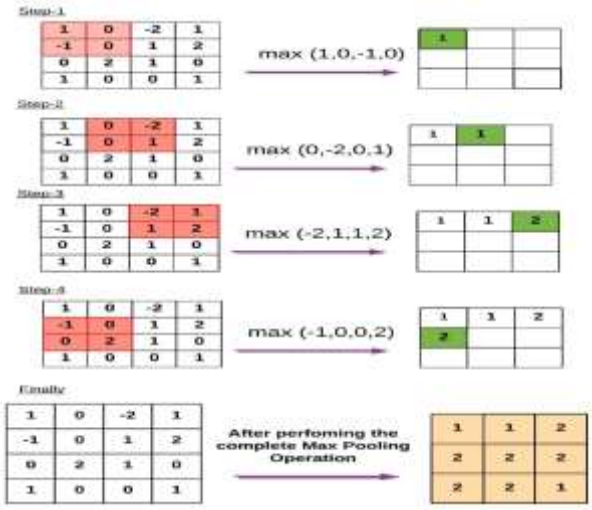


Figure 8. Illustrating the max pooling process [1]

There are different types of pooling techniques are used in different pooling layers such as max pooling, min pooling, average pooling, gated pooling, tree pooling, etc. Max Pooling is the most popular and mostly used pooling technique.

The main drawback of pooling layer is that it sometimes decreases the overall performance of CNN. The reason behind this is that pooling layer helps CNN to find whether a specific feature is present in the given input image or not without caring about the correct position of that feature [1].

Typically, the size of the pooling window is 3×3, and the stride with which the window is moved is also 2 pixels, as shown in Figure 7. This setup reduces the size of the input by half, both in height and width, effectively reducing the total number of pixels by 75%.

Max pooling offers several benefits in the context of CNNs [8]:

Feature Invariance: Max pooling helps the model to become invariant to the location and orientation of features. This means that the network can recognize an object in an image no matter where it is located.

Dimensionality Reduction: By down sampling the input, max pooling significantly reduces the number of parameters and computations in the network, thus speeding up the learning process and reducing the risk of overfitting.

Noise Suppression: Max pooling helps to suppress noise in the input data. By taking the maximum value within the window, it emphasizes the presence of strong features and diminishes the weaker ones.

3. THE EXPERIMENTAL RESULT

In this section, we explain our experimental result. We use an input image captured by a camera. The image size is 640×480 pixels. This experiment is performed using programming language C++ and openCV library.

The programming code to convert the RGB input image into grayscale is:

```
cvtColor(imgOriginal,imgGrey,COLOR_BGR2GRAY);
```

The programming code to convert the grayscale image into CLAHE image with clip limit 4, are:

```
Ptr<CLAHE> clahe = createCLAHE();
clahe->setClipLimit(4);
clahe->apply(imgGrey,imgClahe);
```

To create the filter kernel 3×3 is as follows:

```
kernelPFH = (Mat_<int>(3,3) << -1, 0, 1, -1, 0, 1, -1, 0, 1);
  //Prewitt filter horizontal
```

Then the convolution process is performed in filter2D(src,dst,ddepth,kernel,anchor,delta,BORDER_DEFAULT) as:

```
filter2D(imgClahe,output, -1 , kernel, Point(-1, -1), 0, 4);
```

where the arguments denote:

- *src* : source image.
- *dst* : destination image.
- *ddepth* : the depth of *dst*. A negative value (such as -1) indicates that the depth is same as the source.
- *kernel* : the kernel to be scanned through the image.
- *anchor* : the position of the anchor relative to its kernel. The location *Point(-1,-1)* indicates the center by default.
- *delta* : a value to be added to each pixel during the correlation. By default it is 0.
- *BORDER_DEFAULT* : we let this value by default.

The result images are depicted in Figure 9 and 10. Figure 9 is processing of static image, to compare the convolved image with grayscale input and CLAHE input. It is shown in Figure 9(e) more texture than (b). Figure 10 is processing of the real time frame that captured by a camera, with the image size of 640×480 pixels. It shows the convolved and pooled image.

4. CONCLUSION

2D convolution serves as a powerful tool for capturing spatial patterns and extracting hierarchical features from input images. By convolving learnable filter kernels across the image grid, convolutional layers effectively detect edges, textures, shapes, and other visual motifs, enabling the network to learn rich representations of the input data. Through the process of convolution, raw pixel values are transformed into higher-level features that encode essential information about the underlying structure of the input image.

On the other hand, max pooling serves as a selective down-sampling mechanism that preserves the most salient features while discarding irrelevant details. By systematically scanning through feature maps and retaining only the maximum values within localized regions, max pooling effectively reduces the spatial dimensions of the input data, making subsequent layers more computationally efficient and robust to variations in input size and position.

The experiment has been conducted to observe the result of convolution and max pooling process in real time. The experiment results show the pooled image still has a similar pattern to the input image, i.e. the convolved image. The pooled image becomes an object's feature to be fed to the classifier.

5. REFERENCES

[1] Ghosh, A., Sufian, A., Sultana, F., Chakrabarti, A., De, Debashis. 2020. *Fundamental Concepts of Convolutional Neural Network*. 10.1007/978-3-030-32644-9_36.

[2] K. Fukushima. Neocognitron. 1980. *A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. Biological Cybernetics, 36(4):193-202.

[3] S. M. Pizer, R. E. Johnston, J. P. Ericksen, B. C. Yankaskas and K. E. Muller, "Contrast-limited adaptive histogram equalization: speed and effectiveness," [1990] Proceedings of the First Conference on Visualization in Biomedical Computing, Atlanta, GA, USA, 1990, pp. 337-345.

[4] G. Yadav, S. Maheshwari and A. Agarwal, "Contrast limited adaptive histogram equalization based enhancement for real time video system," 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 2014, pp. 2392-2397.

[5] S. M. Pizer, E. P. Amburn, J. D. Austin, et al.: Adaptive Histogram Equalization and Its Variations. Computer Vision, Graphics, and Image Processing 39. 1987. Pp. 355-368.

[6] K. Zuiderveld: Contrast Limited Adaptive Histogram Equalization. In: P. Heckbert: Graphics Gems IV, Academic Press 1994, ISBN 0-12-336155-9.

[7] T. Sund & A. Moystad: Sliding window adaptive histogram equalization of intra-oral radiographs: effect on diagnostic quality. Dentomaxillofac Radiol. 2006 May;35(3):133-8.

[8] G. R. Vidhya and H. Ramesh, Effectiveness of contrast limited adaptive histogram equalization technique on multispectral satellite imagery, Proc. Int. Conf. Video Image Process., Dec. 2017. pp. 234-239.

[9] Venkatesh S., John De Britto C., Subhashini P., Somasundaram K. *Image Enhancement and Implementation of CLAHE Algorithm and Bilinear Interpolation*. Cybernetics and systems: an International Journal. 2022.

[10] *Filters in convolutional neural networks*, 2022. https://blog.paperspace.com/filters-in-convolutional-neural-networks/.

[11] Stephen Huan, 2020, Fast Fourier Transform and 2D Convolutions.

[12] Vincent Mazet. *Convolution*. Basics of Image Processing — (Université de Strasbourg), 2020-2024 https://vincmazet.github.io/bip/filtering/convolution.html

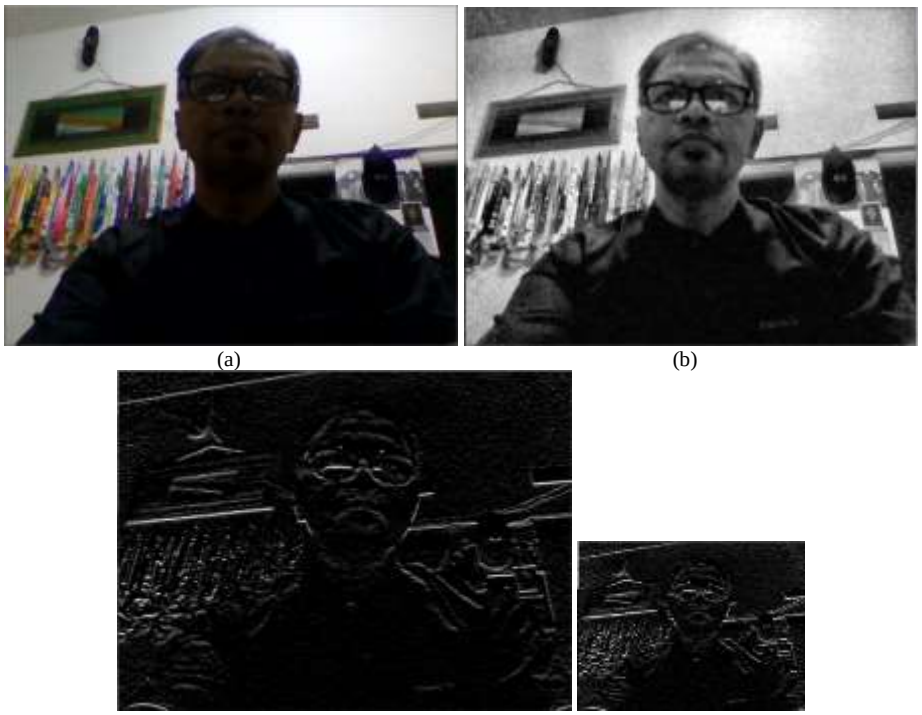
[13] Pupeikis, Rimantas. (2022). Revised 2D convolution. 10.13140/RG.2.2.11346.68809.

(a) (b) (c)

International Journal of Computer Applications Technology and Research
Volume 13–Issue 06, 18 – 23, 2024, ISSN:-2319–8656
DOI:10.7753/IJCATR1306.1003

(d) (e)

Figure 9. Original and convolved images (a) the original image (b) grayscale image (c) the convolved output of (b) (d) CLAHE image (e) the convolved output of (d)



(a) (b)

(c) (d)

Figure 10. Processing of real time frame (a) original image, size of 640×480 pixels (b) CLAHE image (c) convolved image (d) max pooled image of (c), size of 320×240 pixels