# Contents

# 1

# Convolutional Neural Networks

## 1.1 Convolutional operations

### 1.1.1 Images as matrix

An image can be viewed as a piecewise constant function on a grid. Images with different resolutions can then be viewed as functions on grids of different sizes. The use of such multiple-grids is a main technique used in the standard multigrid method for solving discretized partial differential equations, and it can also be interpreted as a main ingredient used in convolutional neural networks (CNN) for image calssification.

An image can be viewed as a function on a grid [6] on a rectangle domain $\Omega \in \mathcal{R}^2$. Without loss of generality, we assume that the grid, $\mathcal{T}$, is of size

$$m = 2^s, \quad n = 2^t$$

for some integers $s, t \geq 1$. Starting from $\mathcal{T}_1 = \mathcal{T}$, we consider a sequence of coarse grids with $J = \min(s, t)$ (as depicted in Fig. 1.1.1 with $J = 4$):

$$\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_J \tag{1.1}$$

such that $\mathcal{T}_\ell$ consist of $m_\ell \times n_\ell$ grid points, with

$$m_\ell = 2^{s-\ell+1}, \quad n_\ell = 2^{t-\ell+1}. \tag{1.2}$$

Here, please note that each element in this grid can be viewed as a pixel or an image or an element in a matrix.

### 1.1.2 Convolution operation with one channel

For simplicity of exposition, we denote

$$m = m_1 = 2^s, \quad n = n_1 = 2^t. \tag{1.3}$$

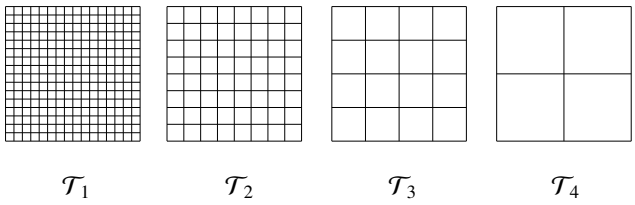



**Fig. 1.1.** multilevel grids for piecewise constant functions (images)

**Definition 1.** *A convolution defined on $\mathbb{R}^{m\times n}$ is a linear mapping $K* : \mathbb{R}^{m\times n} \mapsto \mathbb{R}^{m\times n}$ defined with padding, for any $g \in \mathbb{R}^{m\times n}$ by:*

$$[K * g]_{i,j} = \sum_{p,q=-k}^{k} K_{p,q} g_{i+p,j+q}, \quad i = 1:m, j = 1:n. \tag{1.4}$$

Here we note that the indices for the entries in $K$ are given un a special way. For example, if $k = 1, K \in \mathbb{R}^{3\times 3}$, and

$$K = \begin{pmatrix} K_{-1,-1} & K_{-1,0} & K_{-1,1} \\ K_{0,-1} & K_{0,0} & K_{0,1} \\ K_{1,-1} & K_{1,0} & K_{1,1} \end{pmatrix},$$

for we may have the following 2D Laplacian kernel

$$K = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}. \tag{1.5}$$

The coefficients in (1.17) constitute a kernel matrix

$$K \in \mathbb{R}^{(2k+1)\times(2k+1)}, \tag{1.6}$$

where $k$ is often taken as a small integer. Here padding means how $g_{i+p,j+q}$ is defined when $(i + p, j + q)$ is out of $1 : m$ or $1 : n$. The following three choices are often used

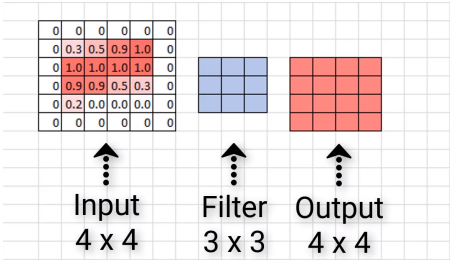
$$g_{i+p,j+q} = \begin{cases} 0, & \text{zero padding,} \\ f_{(i+p) \pmod m,(s+q) \pmod n}, & \text{periodic padding,} \\ f_{|i-1+p|,|j-1+q|}, & \text{reflected padding,} \end{cases} \tag{1.7}$$

if

$$i + p \notin \{1, 2, \ldots, m\} \text{ or } j + q \notin \{1, 2, \ldots, n\}. \tag{1.8}$$

Here $d \pmod m \in \{1, \cdots, m\}$ means the remainder when $d$ is divided by $m$.

Here is a diagram for convolution with one channel (and also stride one).




### 1.1.3 Convolution with stride (one channel)

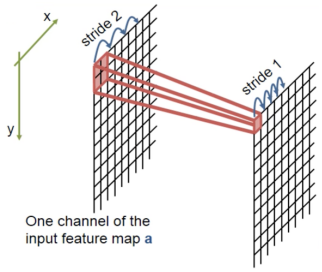
**Definition 2.** *Convolution with stride* 2 *is defined as*

$$[K *_2 g]_{i,j} = \sum_{p,q=-k}^{k} K_{p,q} g_{2i+p-1,2j+q-1}, \quad i = 1 : \lfloor \frac{m+1}{2} \rfloor, j = 1 : \lfloor \frac{n+1}{2} \rfloor. \tag{1.9}$$

We note that, in general, for any given integer $s \geq 1$, a convolution with stride $s$ for $g \in \mathbb{R}^{m\times n}$ can be defined as:

$$[K *_s g]_{i,j} = \sum_{p,q=-k}^{k} K_{p,q} g_{s(i-11)+p+1,s(j-1)+q+1}, \quad i = 1 : \lfloor \frac{m+1}{s} \rfloor, j = 1 : \lfloor \frac{n+1}{s} \rfloor. \tag{1.10}$$

Here $\lfloor \frac{m}{s} \rfloor$ denotes the biggest integer that less than $\frac{m}{s}$. The following is a diagram for stride 2.




**Lemma 1.** *The convolution with stride* 2 *can be written as:*

$$K *_2 g = \mathcal{S}(K * g), \tag{1.11}$$

*where* $\mathcal{S}$ *is a stride operator defined by:*

$$\mathcal{S} : \mathbb{R}^{m\times n} \mapsto \mathbb{R}^{\frac{m+1}{2}\times\frac{n+1}{2}}, \tag{1.12}$$

*with*

$$[\mathcal{S}(g)]_{i,j} = g_{2i-1,2j-1}, \quad i = 1 : \lfloor \frac{m+1}{2} \rfloor, j = 1 : \lfloor \frac{n+1}{2} \rfloor. \tag{1.13}$$

*Example 1.* The so-called average pooling with kernel size $3 \times 3$ and stride 2 means

$$K *_2, \tag{1.14}$$

where

$$K = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}. \tag{1.15}$$

### 1.1.4 Convolutional operations with multi-channel

One important class of linear mapping is the so-called convolution:

$$\theta : \mathbb{R}^{c\times m\times n} \mapsto \mathbb{R}^{h\times m\times n},$$

where $m \times n$ is called the spatial dimension or resolution, $c$ and $h$ are corresponding to input and output channels. The operation is defined by

$$[\theta(f)]_s = \sum_{t=1}^{c} K_{s,t} * [f]_t + b_s \mathbf{1} \in \mathbb{R}^{m\times n}, \quad s = 1 : h, \tag{1.16}$$

where $\mathbf{1} \in \mathbb{R}^{m\times n}$ is a $m \times n$ matrix with all elements being 1, and for $[f]_t \in \mathbb{R}^{m\times n}$ represent for the $t$-th channel

$$[K_{s,t} * [f]_t]_{i,j} = \sum_{p,q=-k}^{k} K_{s,t;p,q} f_{t;i+p,j+q}, \quad i = 1 : m, j = 1 : n. \tag{1.17}$$

The coefficients kernel $K_{s,t}$ in (1.17) constitute a kernel matrix

$$K_{s,t} \in \mathbb{R}^{(2k+1)\times(2k+1)}, \tag{1.18}$$

where $k$ is often taken as small integers.

Here a more compact notation for multi-channel convolution can be written as

$$\theta(f) = K * f + \mathbf{b} \tag{1.19}$$

where

$$f = \begin{pmatrix} [f]_1 \\ [f]_2 \\ \vdots \\ [f]_c \end{pmatrix}, \quad K = \begin{pmatrix} K_{1,1} & K_{1,2} & \cdots & K_{1,c} \\ K_{2,1} & K_{2,2} & \cdots & K_{2,c} \\ \vdots & \vdots & \ddots & \vdots \\ K_{h,1} & K_{h,2} & \cdots & K_{h,c} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1\mathbf{1} \\ b_2\mathbf{1} \\ \vdots \\ b_h\mathbf{1} \end{pmatrix} = b \otimes \mathbf{1}. \tag{1.20}$$

Furthermore, we have the following natural extension of convolution with stride for multi-channel by

$$[\theta(f)]_s = \sum_{t=1}^{c} K_{s,t} *_2 [f]_t + b_s \mathbf{1} \in \mathbb{R}^{\tilde{m}\times\tilde{n}}, \quad s = 1:h, \tag{1.21}$$

where

$$\tilde{m} = \lfloor \frac{m+1}{2} \rfloor, \quad \tilde{n} = \lfloor \frac{n+1}{2} \rfloor, \tag{1.22}$$

### 1.1.5 Pooling operation in CNNs

Finally, we introduce another type of important operation in CNNs – pooling. The key purpose for pooling operator is to reduce the spatial resolution of images (features) in a typical CNN models. Basically, pooling is an operator

$$T : \mathbb{R}^{c_1\times m_1\times n_1} \mapsto \mathbb{R}^{c_2\times m_2\times n_2}. \tag{1.23}$$

where

$$m_2 = \lfloor \frac{m+1}{s} \rfloor, \quad n_2 = \lfloor \frac{n+1}{s} \rfloor, \tag{1.24}$$

for any choice of $c_2 \geq 1$. Here $s$ is also called the stride in pooling operations. There are generally two types of pooling

*Convolution with stride s as pooling*

In this case, it often happens that

$$T = R*_s, \quad (s = 2 \text{ for the main case }). \tag{1.25}$$

Here $R$ can be learned or fixed such as average pooling as we discussed before.

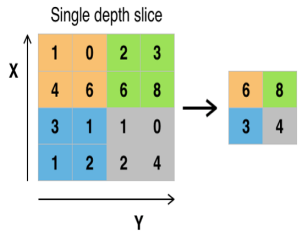
*Nonlinear pooling*

The most commonly used nonlinear pooling is called max-pooling, a max pooling with kernel size $(2k+1)\times(2k+1)$ and stride $s$ is is defined as

$$[R_{\max}(f)]_{t;i,j} = \max\{f_{t;si+p-1,sj+q-1} \mid -k \leq p,q \leq k\}, \tag{1.26}$$

here $t$ means channel and $c_2 = c_1$ in this case.

Here is an example for max-pooling with kernel size $2\times 2$ and stride 2.



## 1.2 Examples of convolution filters and performance

In this section, we will give a brief description how convolution operations are used for image processing. One useful description can be found in the following link:

http://aishack.in/tutorials/image-convolution-examples/

Convolutions is a technique for general signal processing. People studying electrical/electronics will tell you the near infinite sleepless nights these convolutions have given them. Entire books have been written on this topic. And the questions and theorems that need to be proved are [insurmountable]. But for computer vision, we'll just deal with some simple things.

A convolution lets you do many things, like calculate derivatives, detect edges, apply blurs, etc. A very wide variety of things. And all of this is done with a "convolution kernel".

### 1.2.1 Calculation with convolutions

The most direct way to compute a convolution would be to use multiple for loops. But that causes a lot of repeated calculations. And as the size of the image and kernel increases, the time to compute the convolution increases too (quite drastically).

Techniques haves been developed to calculate convolutions rapidly. One such technique is using the Discrete Fourier Transform. It converts the entire convolution operation into a simple multiplication. Fortunately, you don't need to know the math to do this in OpenCV. It automatically decides whether to do it in frequency domain (after the DFT) or not.

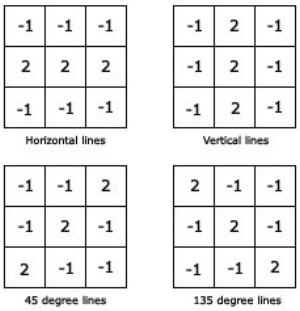
### 1.2.2 Image convolution examples

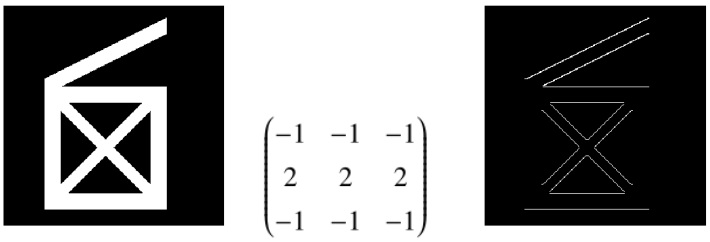
A convolution is very useful for signal processing in general. There is a lot of complex mathematical theory available for convolutions. For digital image processing, you don't have to understand all of that. You can use a simple matrix as an image convolution kernel and do some interesting things!

### 1.2.3 Line detection by 1D Laplacian

With image convolutions, you can easily detect lines. Here are four convolutions to detect horizontal, vertical and lines at 45 degrees:

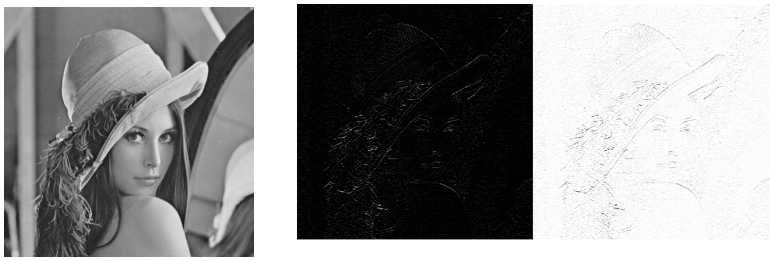
Here's 0,90,45,135 lines detection that I got on an image:



$$\begin{pmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{pmatrix}$$

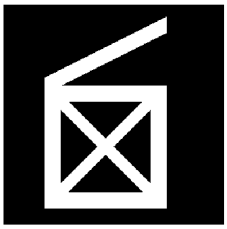
(a) image (b) filter (c) result



(d) image (e) result

**Fig. 1.2.** A horizontal line detection done with convolutions

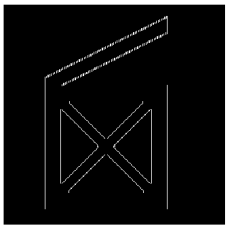
In Lena,the black background is the original result, the white background is obtained by subtracting the original result from 255, the same below.



(a) image

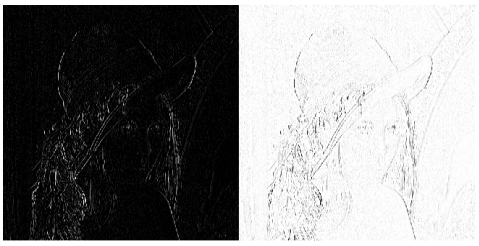
$$\begin{pmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{pmatrix}$$

(b) filter



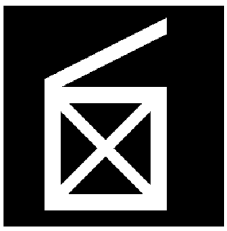
(c) result

(d) image



(e) result

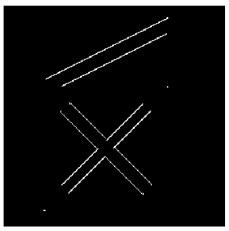
**Fig. 1.3.** A vertical line detection done with convolutions



(a) image

$$\begin{pmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{pmatrix}$$

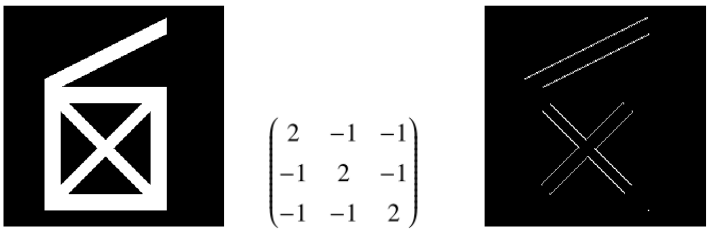
(b) filter



(c) result

(d) image (e) result

**Fig. 1.4.** A 45 degress line detection done with convolutions



$$\begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix}$$

(a) image (b) filter (c) result

(d) image (e) result

**Fig. 1.5.** A 135 degress line detection done with convolutions

### 1.2.4 Edge detection by 2D Laplacian operator

The laplacian is the second derivative of the image. It is extremely sensitive to noise, so it isn't used as much as other operators. Unless, of course you have specific requirements.

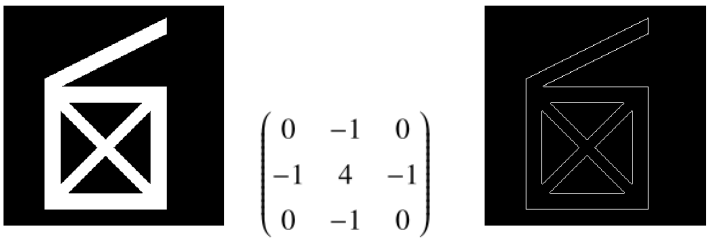
| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

The laplacian operator

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

The laplacian operator (include diagonals)

Here's the result with the convolution kernel without diagonals:



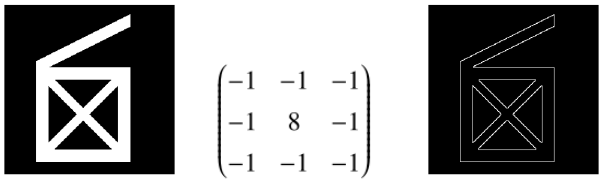
$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

(a) image (b) filter (c) result

(d) image (e) result

**Fig. 1.6.** A laplace operator done with convolutions

The result with the convolution kernel with diagonals:



$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

(a) image (b) filter (c) result

(d) image (e) result

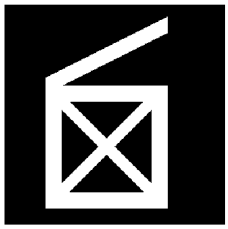
**Fig. 1.7.** A laplace operator include diagonals done with convolutions

### 1.2.5 The Laplacian of Gaussian

The laplacian alone has the disadvantage of being extremely sensitive to noise. So, smoothing the image before a laplacian improves the results we get. This is done with a 5x5 image convolution kernel.

| | | | | |
|---|---|---|---|---|
| 0 | 0 | -1 | 0 | 0 |
| 0 | -1 | -2 | -1 | 0 |
| -1 | -2 | 16 | -2 | -1 |
| 0 | -1 | -2 | -1 | 0 |
| 0 | 0 | -1 | 0 | 0 |

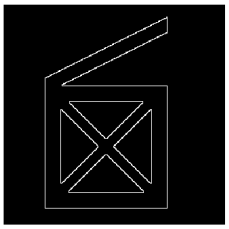
The result on applying this image convolution was:



(a) image

$$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

(b) filter



(c) result

(d) image

(e) result

**Fig. 1.8.** A Laplacian of Gaussian operator done with convolutions

### 1.2.6 Other examples with ReLU activation

(a) input image

$$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

(b) filter

(c) convolution result

(d) result after ReLU

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

(e) filter

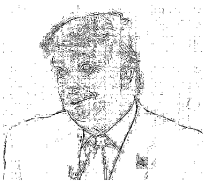
(f) result after average

(g) image

$$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

(h) filter

(i) filter



(j) ReLU

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

(k) filter

(l) average

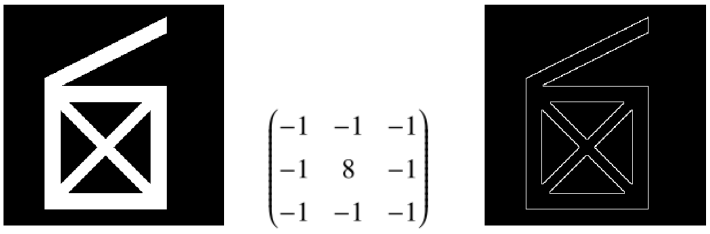
### 1.2.7 Some other examples

#### Edge detection

The above kernels are in a way edge detectors. Only thing is that they have separate components for horizontal and vertical lines. A way to ”combine” the results is to merge the convolution kernels. The new image convolution kernel looks like this:

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

Below result I got with edge detection:



$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

(m) image (n) filter (o) result

(p) image (q) result

**Fig. 1.9.** A edge detection done with convolutions

**The Sobel Edge Operator**

The above operators are very prone to noise. The Sobel edge operators have a smoothing effect, so they're less affected to noise. Again, there's a horizontal component and a vertical component.

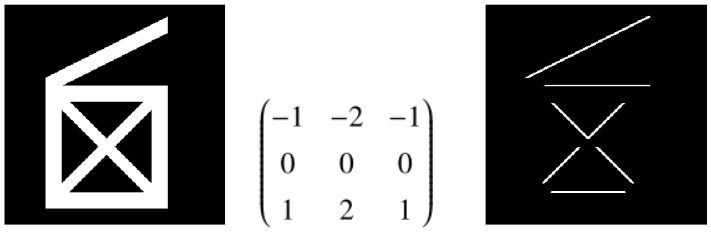
| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

Horizontal

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

Vertical

On applying horizontal component in image , the result was:



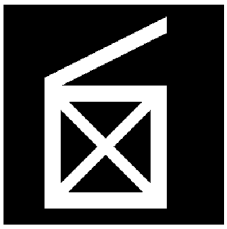
$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

(a) image (b) filter (c) result

(d) image (e) result

**Fig. 1.10.** A horizontal sobel edge operator done with convolutions

On applying vertical component in image , the result was:



(a) image

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

(b) filter

(c) result

(d) image

(e) result

**Fig. 1.11.** A vertical sobel edge operator done with convolutions

### Simple box blur

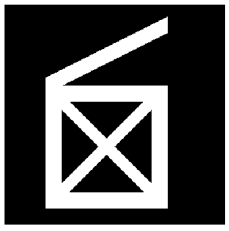
[1] Here's a first and simplest. This convolution kernel has an averaging effect. So you end up with a slight blur. The image convolution kernel is:

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

Note that the sum of all elements of this matrix is 1.0. This is important. If the sum is not exactly one, the resultant image will be brighter or darker.

Here's a blur that I got on an image:

[1] The following examples are from the website, http://aishack.in/tutorials/image-convolution-examples/



(a) image

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

(b) filter

(c) result

(d) image

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

(e) filter

(f) result

**Fig. 1.12.** A simple blur done with convolutions

### Gaussian blur

Gaussian blur has certain mathematical properties that makes it important for computer vision. And you can approximate it with an image convolution. The image convolution kernel for a Gaussian blur is:

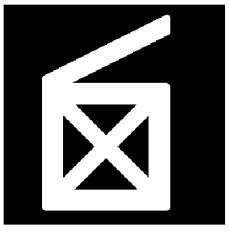
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 5 | 0 | 0 | 0 |
| 0 | 5 | 18 | 32 | 18 | 5 | 0 |
| 0 | 18 | 64 | 100 | 64 | 18 | 0 |
| 5 | 32 | 100 | 100 | 100 | 32 | 5 |
| 0 | 18 | 64 | 100 | 64 | 18 | 0 |
| 0 | 5 | 18 | 32 | 18 | 5 | 0 |
| 0 | 0 | 0 | 5 | 0 | 0 | 0 |

Here's a result that I got:

(a) image

$$\begin{pmatrix} 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 5 & 18 & 32 & 18 & 5 & 0 \\ 0 & 18 & 64 & 100 & 64 & 18 & 0 \\ 5 & 32 & 100 & 100 & 100 & 32 & 5 \\ 0 & 18 & 64 & 100 & 64 & 18 & 0 \\ 0 & 5 & 18 & 32 & 18 & 5 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \end{pmatrix}$$

(b) filter



(c) result

(d) image

$$\begin{pmatrix} 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 5 & 18 & 32 & 18 & 5 & 0 \\ 0 & 18 & 64 & 100 & 64 & 18 & 0 \\ 5 & 32 & 100 & 100 & 100 & 32 & 5 \\ 0 & 18 & 64 & 100 & 64 & 18 & 0 \\ 0 & 5 & 18 & 32 & 18 & 5 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \end{pmatrix}$$

(e) filter

(f) result

**Fig. 1.13.** A Gaussian blur done with convolutions

### 1.2.8 Summary

You got to know about some important operations that can be approximated using an image convolution. You learned the exact convolution kernels used and also saw an example of how each operator modifies an image. I hope this helped!

# References


[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[2] J. He, Y. Chen, and J. Xu. Constrained linear data-feature mapping for image classification. *arXiv preprint arXiv:1911.10428*, 2019.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[5] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[9] J. Xu and L. Zikatanov. Algebraic multigrid methods. *Acta Numerica*, 26:591–721, 2017.