Convolutions for Machine Learning

Glen Woodworth

UAF

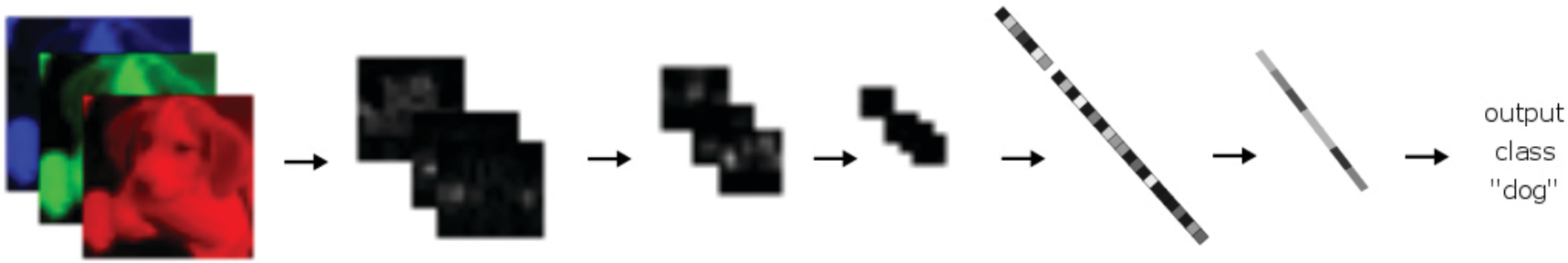
February 4, 2022

Outline

Introduction

- A *convolution*, denoted by $*$, is an operation on two functions, f and k that produces a third function $(f * k)$.
- In machine learning, the first argument, f, is the input, while the second, k, is called a *kernel*. The output is called a *feature map*.
- *Convolutional neural networks* (CNNs) are artificial neural networks that use convolution in place of general matrix multiplication in at least one of their layers.(Goodfellow et al.)

Basic CNN Structure



1D Discrete Convolution

Consider the convolution of $s \in \mathbb{R}^n$ and $k \in \mathbb{R}^m$, $(s * k)(u)$, defined as

$$(s * k)_n = \sum_{j=1}^{m} s_{u-j} k_j,$$

where $u \in \mathbb{Z}$.

To evaluate the convolution over all of s, we iterate this sum over u. This means k slides over a flipped s, and the k weighted sum of $\{s_i : u - m \leq i \leq u - 1\}$ is stored in a vector as $(s * k)(u)$. This can cause a small problem at the edges, which we explore by an example.

Note

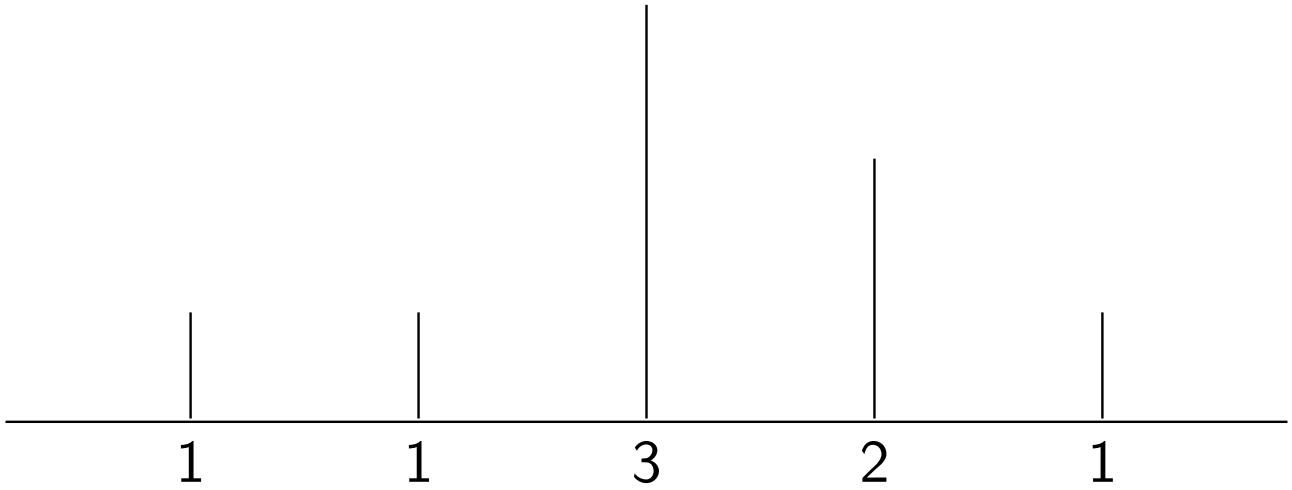
What machine learning convention calls a convolution kernel is what mathematicians call a *cross correlation* kernel. A convolution kernel is the vertical and horizontal reflection of a cross correlation kernel. These reflections are important for other applications. 2D kernels show the difference between the defintions best:

$$\text{ML Convolution} : \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \text{Math Convolution} : \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix}$$

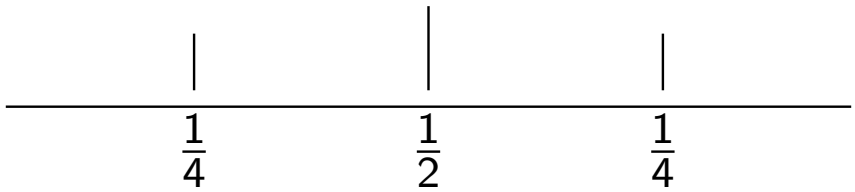
I will use the machine learning convention (no flip).

1D Discrete Convolution: Example

Suppose our input is $s = [1, 1, 3, 2, 1]$ and we want to convolve it with an averaging kernel, $k = [\frac{1}{4}, \frac{1}{2}, \frac{1}{4}]$.



s



k

1D Discrete Convolution: Example

For $u = 4, 5, 6$ $s = [1, 1, 3, 2, 1]$ and $k = [\frac{1}{4}, \frac{1}{2}, \frac{1}{4}]$, we get

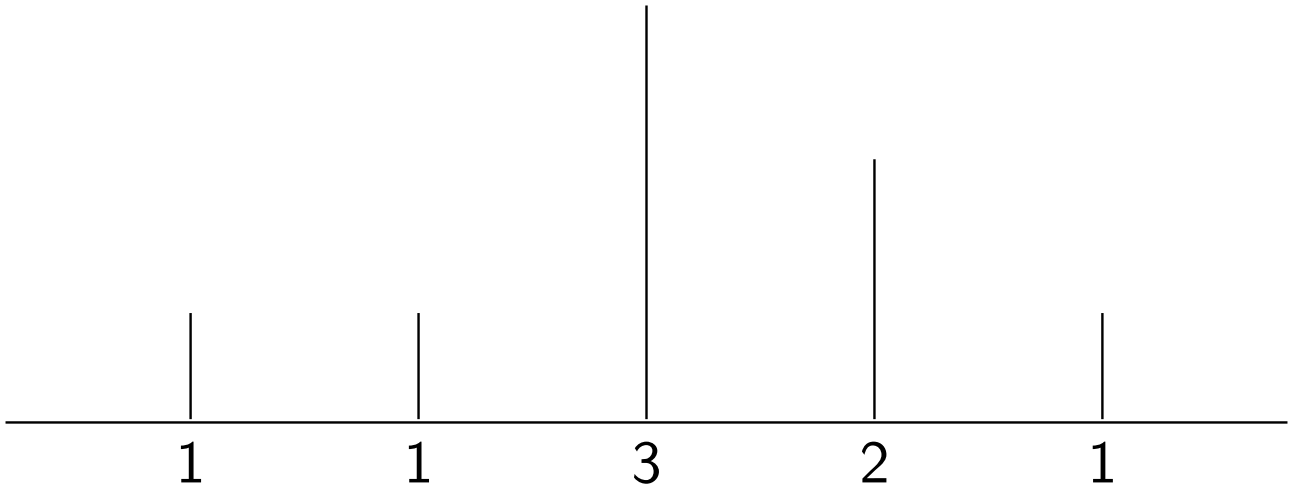
$$(s * k)(4) = \sum_{j=1}^{3} s_{4-j} k_j = s_3 k_1 + s_2 k_2 + s_1 k_3 = 3\frac{1}{4} + 1\frac{1}{2} + 1\frac{1}{4} = \frac{3}{2}$$

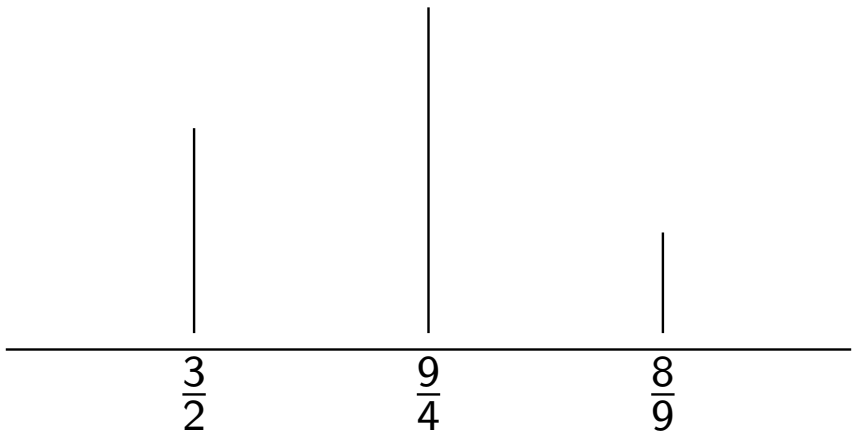
$$(s * k)(5) = \sum_{j=1}^{3} s_{5-j} k_j = s_4 k_1 + s_3 k_2 + s_2 k_3 = 2\frac{1}{4} + 3\frac{1}{2} + 1\frac{1}{4} = \frac{9}{4}$$

$$(s * k)(6) = \sum_{j=1}^{3} s_{6-j} k_j = s_5 k_1 + s_4 k_2 + s_3 k_3 = 1\frac{1}{4} + 2\frac{1}{2} + 3\frac{1}{4} = \frac{8}{9}$$

Picking $u = 1, 2, 3$ or $u > 6$ is not defined, but this biases our convolution toward the center values. Notice that s_3 is the only element from s to appear in each one.

1D Discrete Convolution Example: Output





1D Padding and Looping

When we don't want the output of our convolution to be smaller than the input, or when we want to avoid center bias, we can use padding or looping.

Padding means adding extra values on each side. In our example, we could add one zero on each side,

$$s_p = [0, 1, 1, 3, 2, 1, 0].$$

Looping refers to placing a copy of s before and after s, so that the last values come before the first, and vice-versa. In our example, we get

$$s_l = [...3, 2, 1, \underline{1, 1, 3, 2, 1}, 1, 1, 3, ...],$$

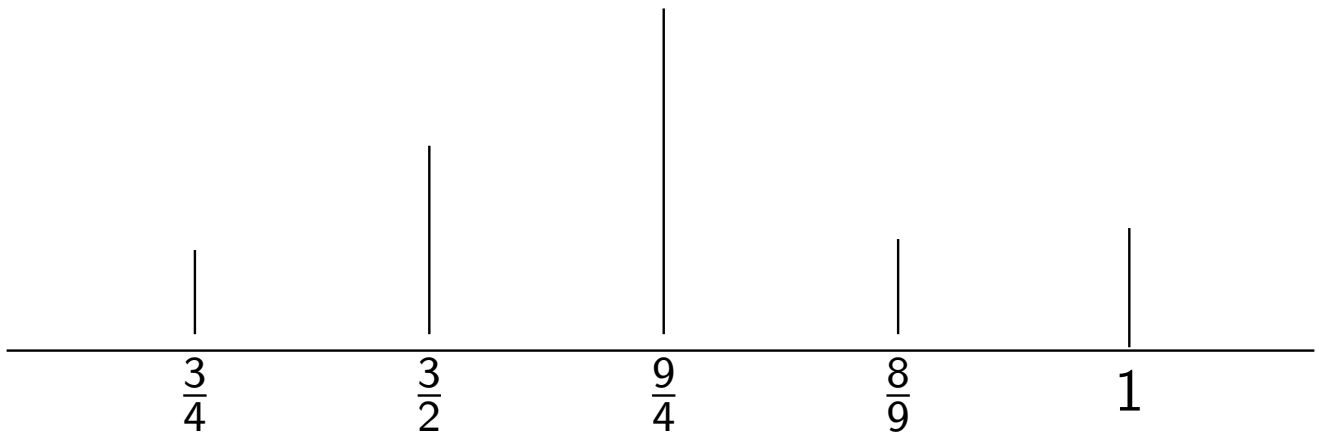
where the original s is underlined.

1D Padding Example

Using $s_p = [0, 1, 1, 3, 2, 1, 0]$ and $k = [\frac{1}{4}, \frac{1}{2}, \frac{1}{4}]$, we can now find $(s_p * k)(3)$ and $(s_p * k)(7)$ so that $(s_p * k)$ is the same size as s.

$$(s * k)(3) = \sum_{j=1}^{3} s_p(3-j)k_j = 1\frac{1}{4} + 1\frac{1}{2} + 0\frac{1}{4} = \frac{3}{4}$$

$$(s * k)(7) = \sum_{j=1}^{3} s_p(7-j)k_j = 0\frac{1}{4} + 1\frac{1}{2} + 2\frac{1}{4} = 1.$$



Discrete 2D Convolution: Edge detection

In order to apply a convolution to an image, we need a two dimensional version. Rather than define this using a complicated sum, we look at matrices. Suppose we have a 6x6, black and white image, A, that we wish to detect the vertical edges of. One simple way to do this is to convolve A with a 3x3 kernel K_v.

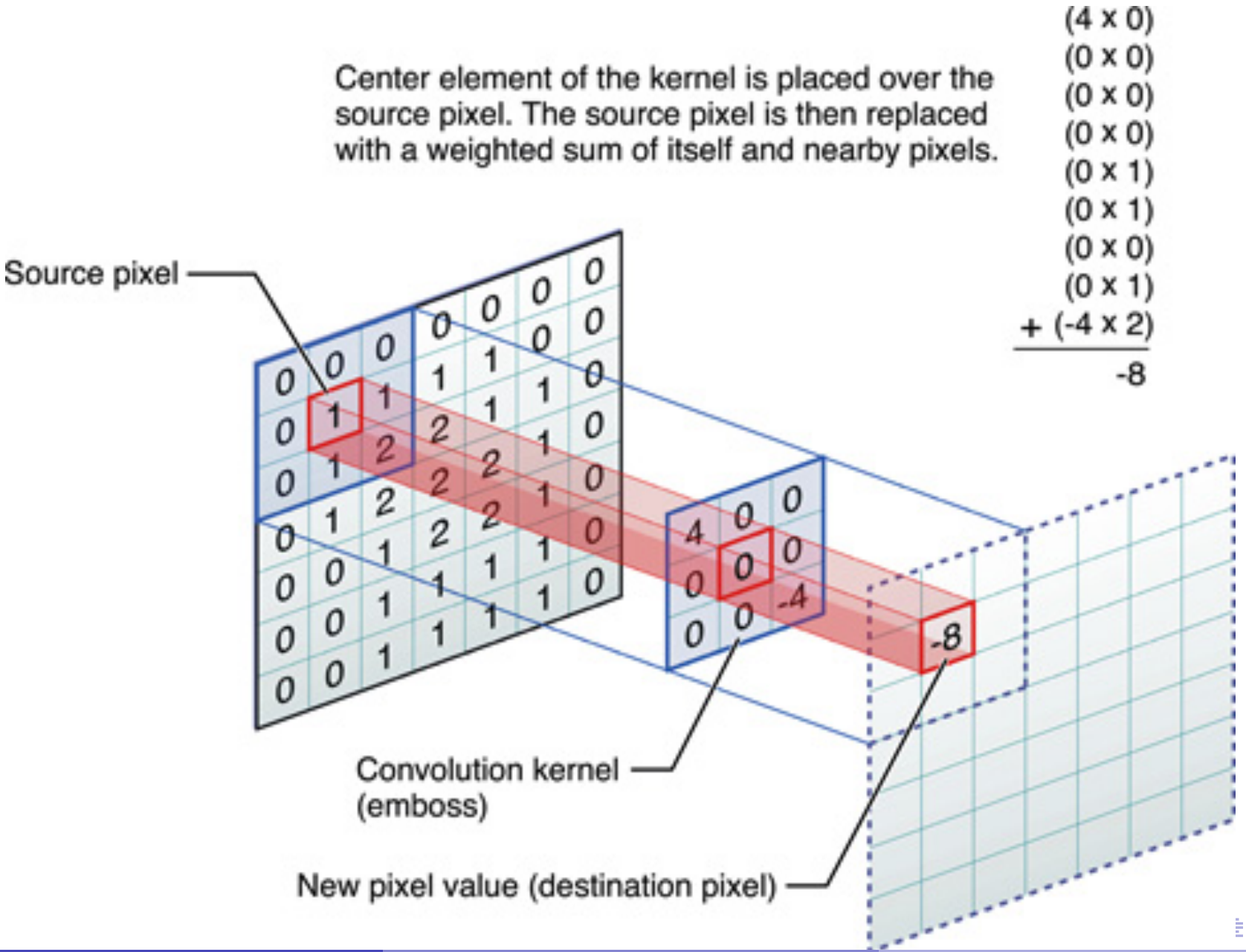
$$A = \begin{pmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{pmatrix}, \; K_v = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

Computing Discrete 2D Convolution

To compute $(A * K_v)$, we place K_v in the top left corner of A, multiplying term-by-term the values of this corner of A times the values of K_v and summing them (NOT matrix multiplication!) We slide K_v over 1 spot, and iterate until we have covered all of A.

$$A * K_v = \begin{pmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{pmatrix}.$$

Visualizing a Convolution



Discrete 2D Convolution: Padding

If the output is restricted to only positions where the kernel lies entirely within the image, this is called a *valid* convolution (Goodfellow et al.).

If the kernel will slide partially off of the input image, but we don't want to restrict the output, padding is used.

Padding A with zeros creates an 8x8:

$$A_p = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Discrete 2D Convolution: Striding

Another way to change the size of the output of a convolution is called *striding*. Usually, a stride is 1, moving the kernel 1 pixel at a time. A stride of 2, for example, means that the kernel skips over a pixel each time it moves. As an example of a stride of 2 with K_v, we use an odd-sized matrix so that the convolution is valid.

$$A_s * K_v = \begin{pmatrix} 10 & 10 & 10 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 \end{pmatrix} * \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 30 \\ 0 & 30 \end{pmatrix}.$$

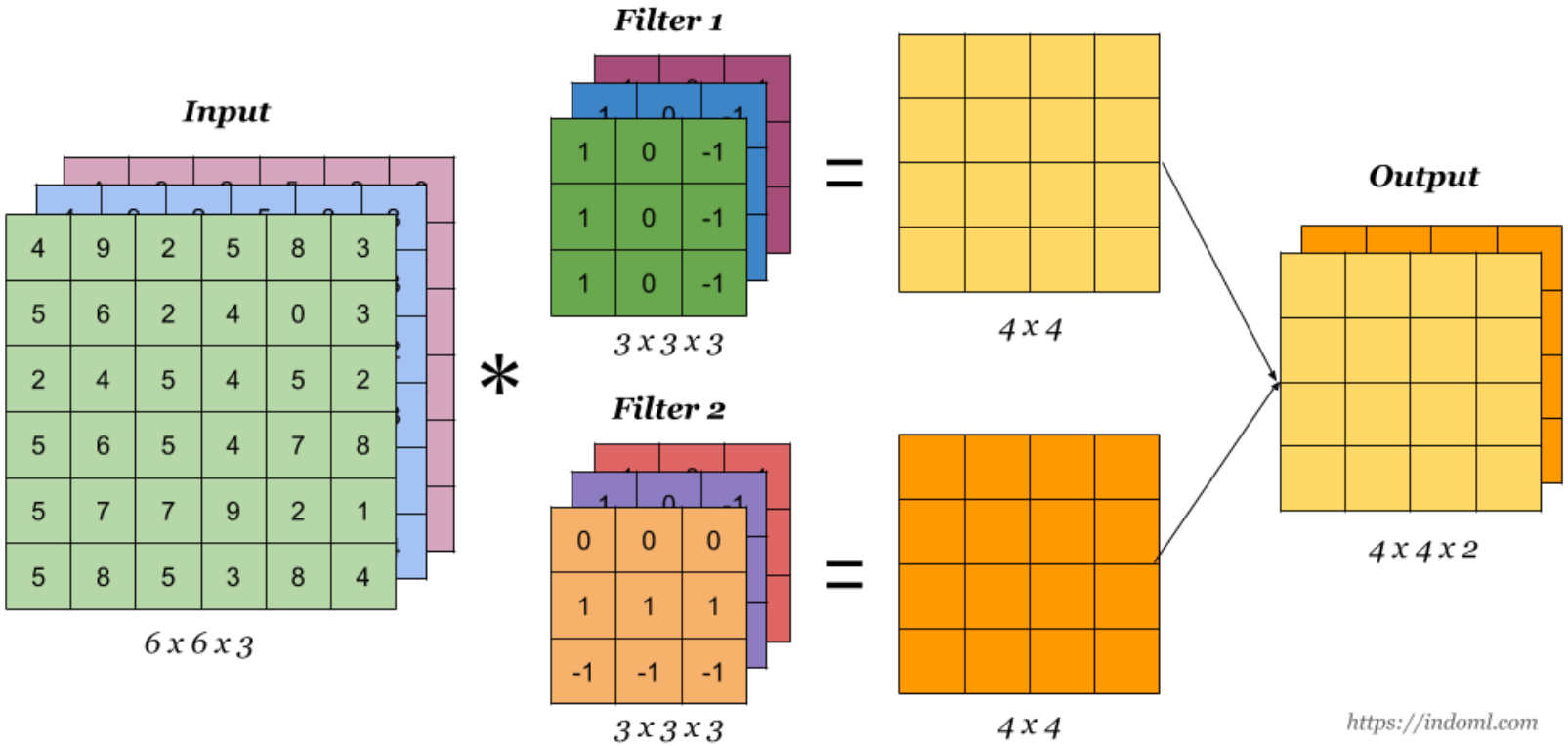
Discrete 2D Convolutions: Output Size

The size of the output depends on the input size (nxn), kernel size (fxf), padding (p), and striding (s. Here is a formula to determine the output size (from Andrew Ng):

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor x \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

Discrete 2D Convolution: RGB Image

A digital image is created using discrete image pixels with different ratios of red, green, and blue (RGB). This is represented as a tensor. For a 6x6 RGB image, there are really 3 6x6 matrices of pixel values (3rd order 6x6 tensor). Each order of the tensor is called a *channel*. To convolve such an image, we need to apply a kernel to each one.



Convolution Image Example: Horizontal and vertical edge filters

Kernels used: Vertical and horizontal Sobel filters (Kim,Casper)

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

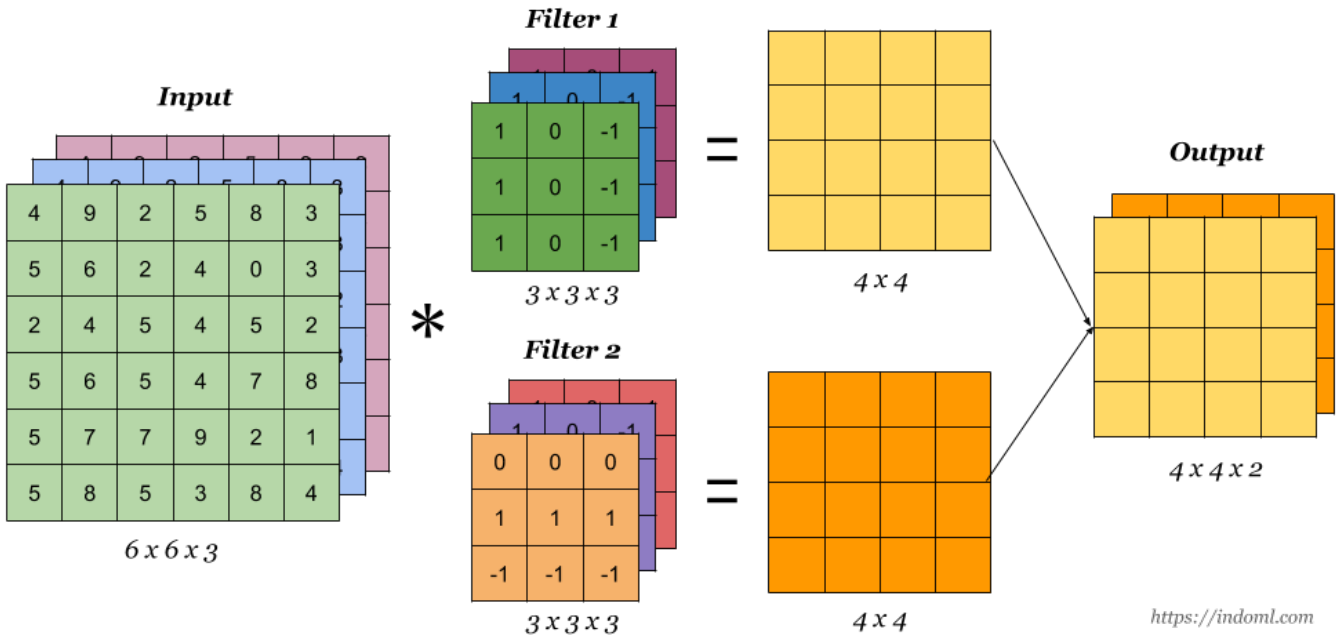
Convolutional Neural Networks

- CNN net training finds the weights in the kernel that is applied to the image
  - CNNs typically have *sparse interactions* because the kernel is smaller than the input
  - For example, an image might have millions of pixels, a CNN can detect meaningful features using kernels with tens or hundreds of pixels. (Goodfellow et al.)
  - Since the kernel is typically smaller than the input, CNNs store less parameters, regardless of input size
  - Using

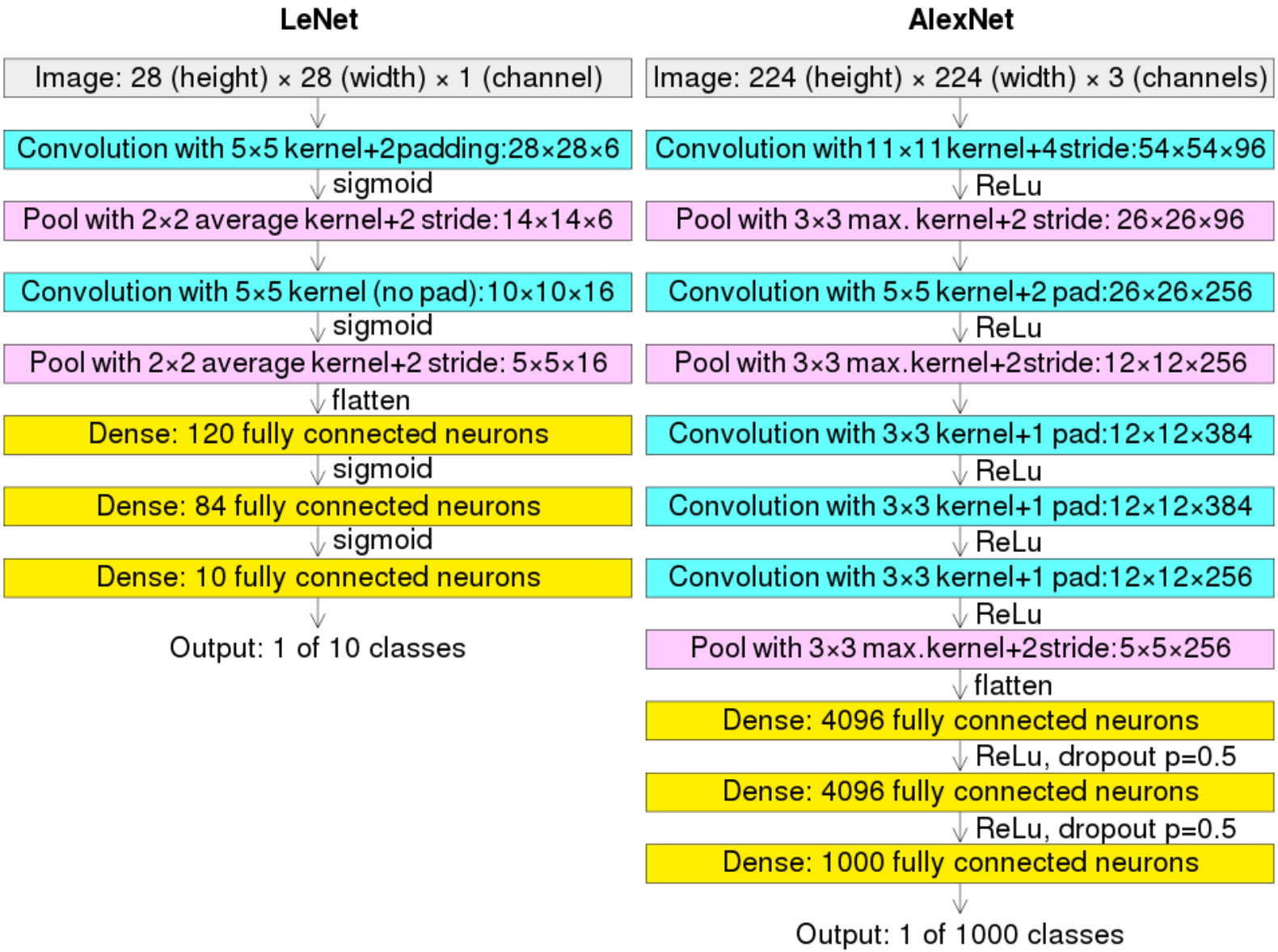
A Convolutional Layer

The most basic convonlutional layer (that no one would use) applies the convolution shown below. Training the layer goes as follows:

- Each filter starts with randomized values
- Compute convolution, add a bias
- ReLu is applied
- Flatten into a long vector, apply logistic regression
- Back-propogation updates the values in the kernels and the biases



CNN Architecture



Further Reading

- Higham C., Higham D. (2018) Deep Learning: An Introduction for Applied Mathematicians, SIAM
- Kim, S., Casper, R. (2013). Applications of convolution in image processing with MATLAB. University of Washington, 1-20.
- Goodfellow, I., Benjio Y., Courville A. Deep Learning. MIT
- Convolutional Neural Networks Course by Andrew Ng, youtube
- Convolutions in image processing by Grant Sanderson, youtube