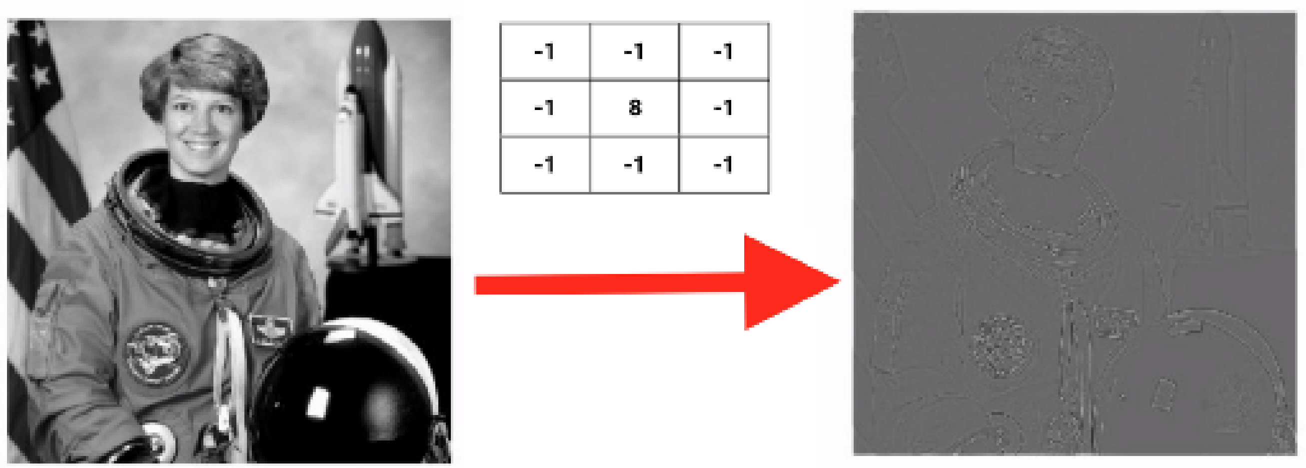
Get started

A Beginner’s Guide to Convolutional Neural Networks (CNNs)

Suhyun Kim
Feb 15 · 7 min read ★

What is a Convolution?

A convolution is how the input is modified by a filter. In convolutional networks, multiple filters are taken to slice through the image and map them one by one and learn different portions of an input image. Imagine a small filter sliding left to right across the image from top to bottom and that moving filter is looking for, say, a dark edge. Each time a match is found, it is mapped out onto an output image.

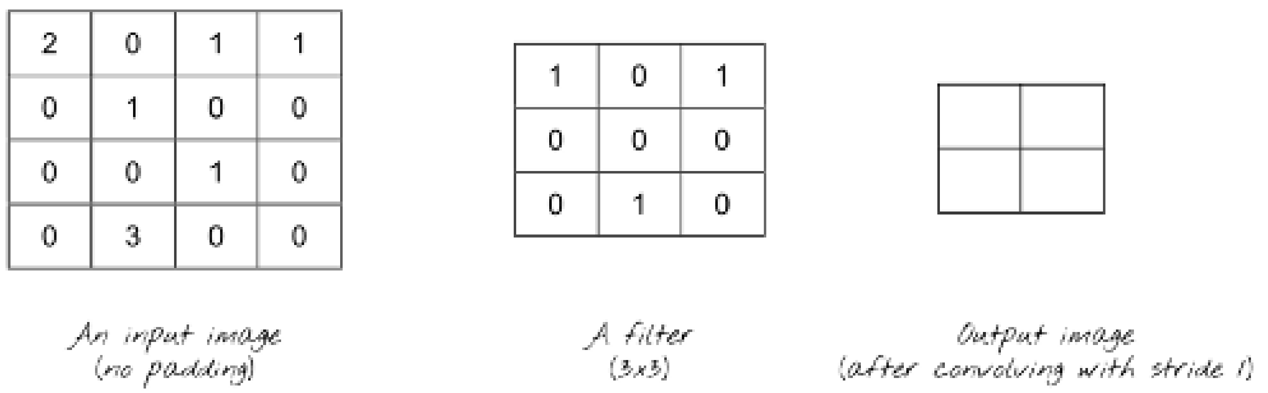


https://www.cs.columbia.edu/education/courses/course/COMSW4995-7/26050/

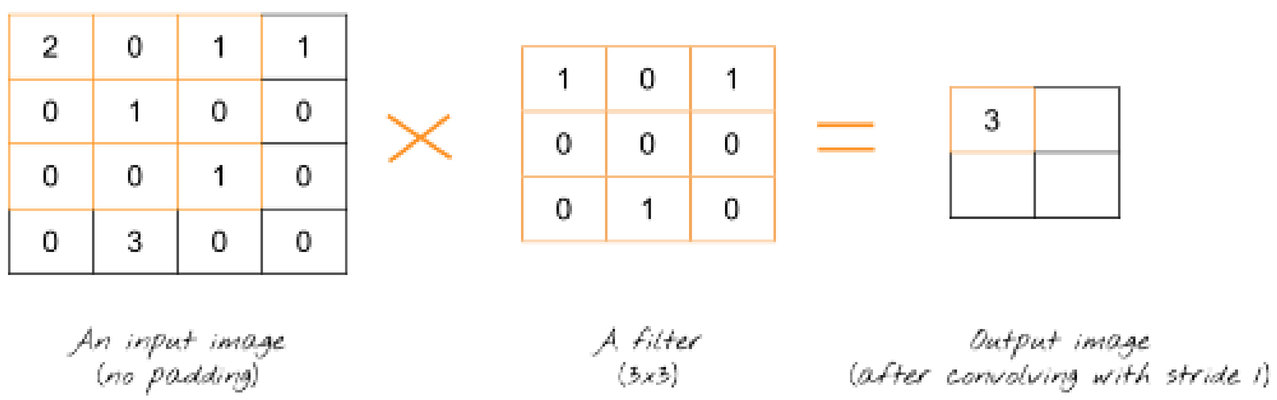
For example, there is a picture of Eileen Collins and the matrix above the red arrow is used as a convolution to detect dark edges. As a result, we see an image where only dark edges are emphasized.

Note that an image is 2 dimensional with width and height. If the image is colored, it is considered to have one more dimension for RGB color. For that reason, 2D

Let’s start with a (4 x 4) input image with no padding and we use a (3 x 3) convolution filter to get an output image.

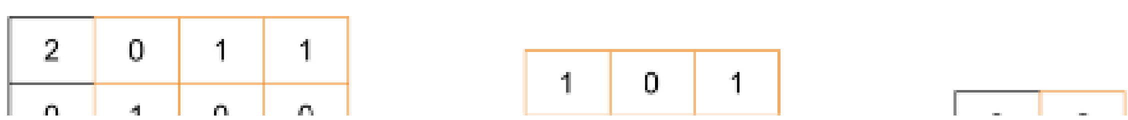


The first step is to multiply the yellow region in the input image with a filter. Each element is multiplied with an element in the corresponding location. Then you sum all the results, which is one output value.

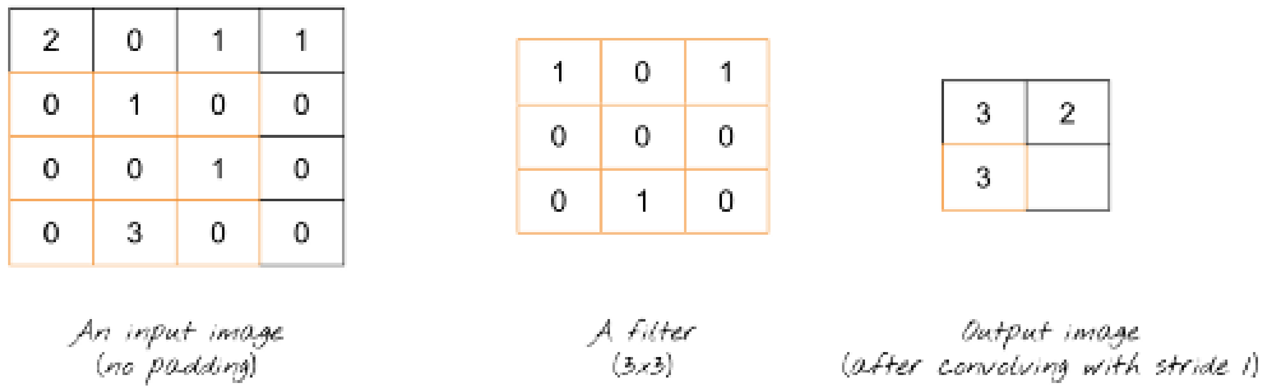


Mathematically, it’s (2 * 1) + (0 * 0) + (1 * 1) + (0 * 0) + (1 * 0) + (0 * 0) + (0 * 0) + (0 * 1) + (1 * 0) = 3

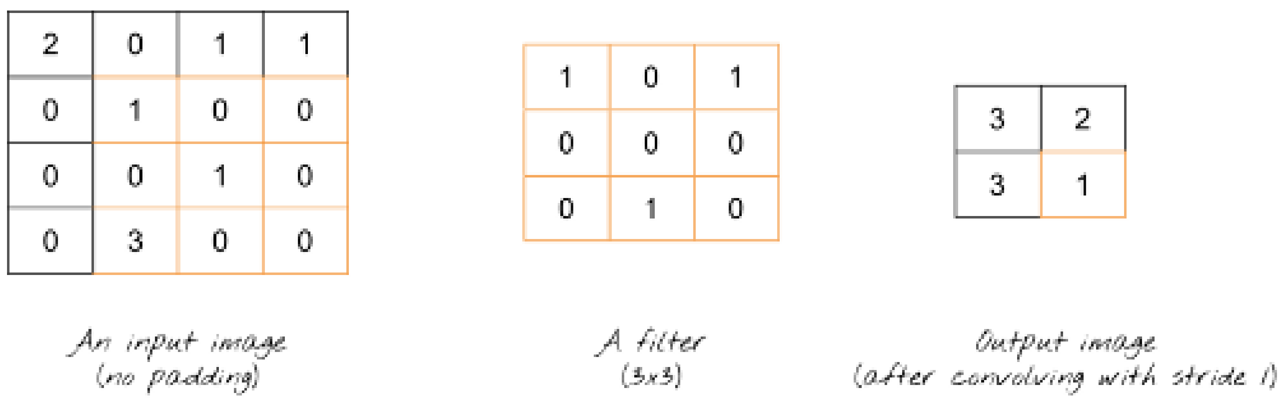
Then, you repeat the same step by moving the filter by one column. And you get the second output.



Notice that you moved the filter by only one column. The step size as the filter slides across the image is called a **stride**. Here, the stride is 1. The same operation is repeated to get the third output. A stride size greater than 1 will always downsize the image. If the size is 1, the size of the image will stay the same.



At last, you are getting the final output.



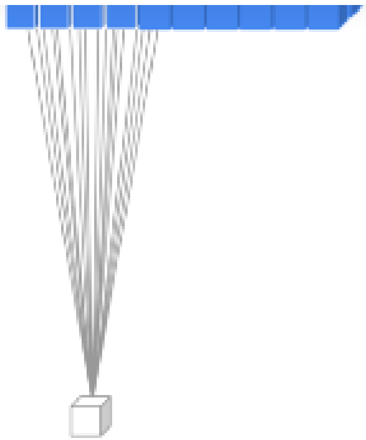
We see that the size of the output image is smaller than that of the input image. In fact, this is true in most cases.

Convolution in 3D

Convolution in 3D is just like 2D, except you are doing the 2d work 3 times, because there are 3 color channels.

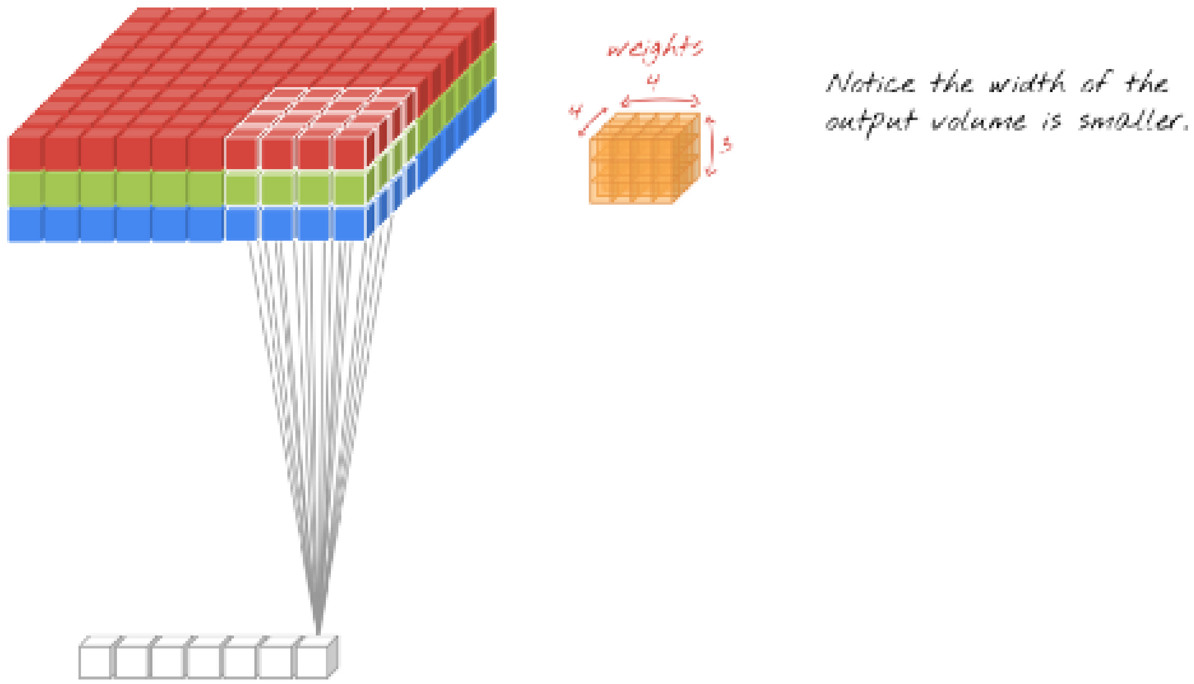
M

Get started



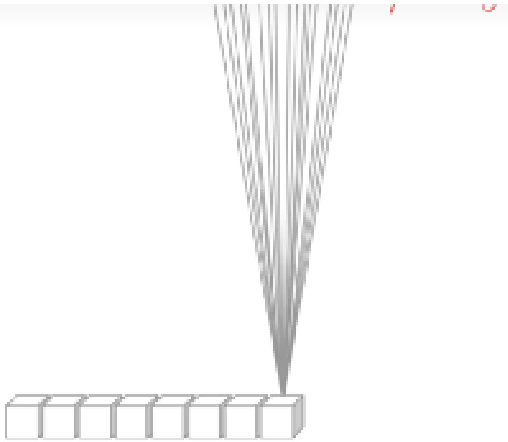
https://twitter.com/martin_gorner

Normally, the width of the output gets smaller, just like the size of the output in 2D case.



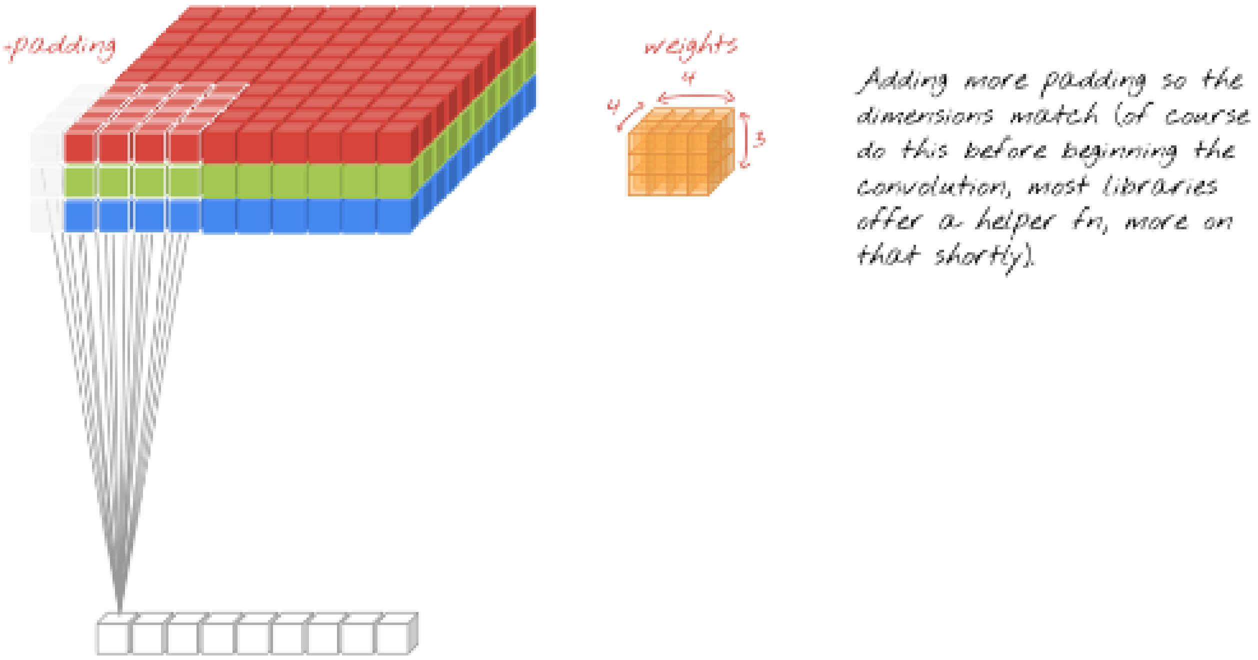
https://twitter.com/martin_gorner

If you want to keep the output image at the same width and height without decreasing the filter size, you can add padding to the original image with zero's and make a convolution slice through the image.



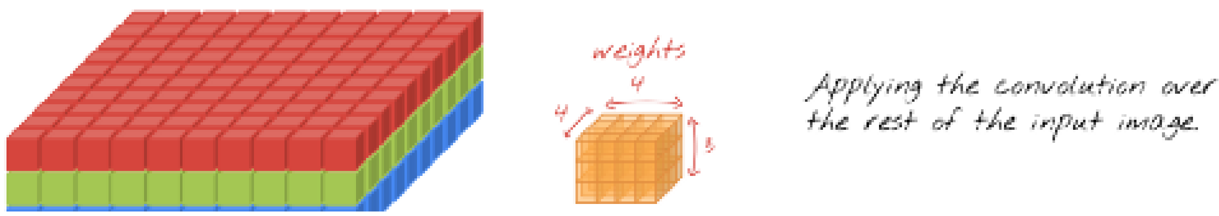
https://twitter.com/martin_gorner

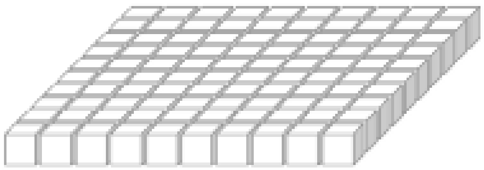
We can apply more padding!



https://twitter.com/martin_gorner

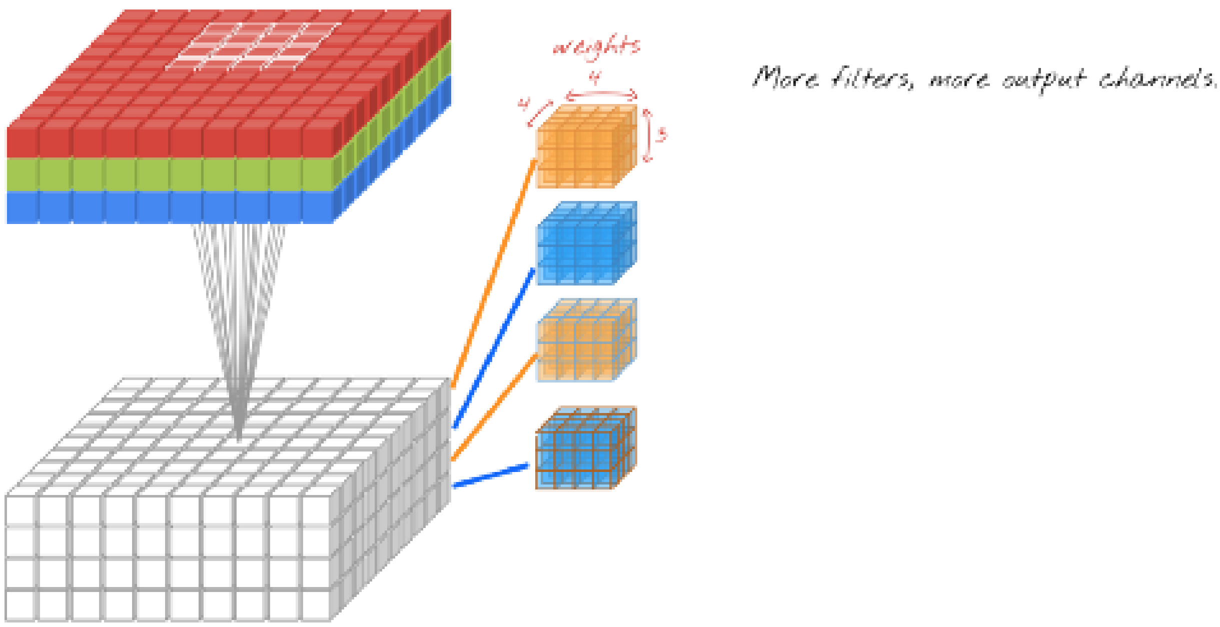
Once you're done, this is what the result would look like:





https://twitter.com/martin_gorner

As you add more filters, it increases the depth of the output image. If you have the depth of 4 for the output image, 4 filters were used. Each layer corresponds to one filter and learns one set of weights. It does not change between steps as it slides across the image.



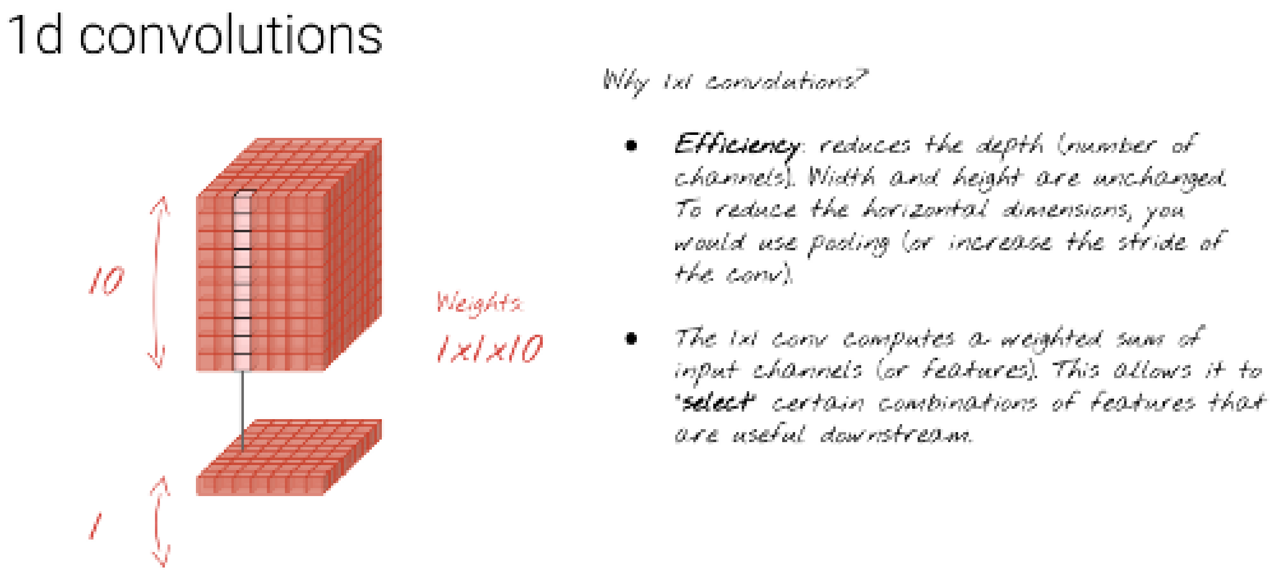
https://twitter.com/martin_gorner

An output channel of the convolutions is called a **feature map**. It encodes the presence or absence, and degree of presence of the feature it detects. Notice that unlike the 2D filters from before, each filter connects to **every** input channel. (question? what does it mean by each filter connects to every input channel unlike 2D?) This means they can compute sophisticates features. Initially, by looking at R, G, B channels, but after, by looking at combinations of learned features such as various edges, shapes, textures and semantic features.

because the convolution is a feature detector and if it's detecting a dark edge and the image is moved to the bottom, then dark edges will not be detected until the convolution is moved down.

Special Case — 1D Convolution

1D convolution is covered here, because it's usually under-explained, but it has noteworthy benefits.



https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd

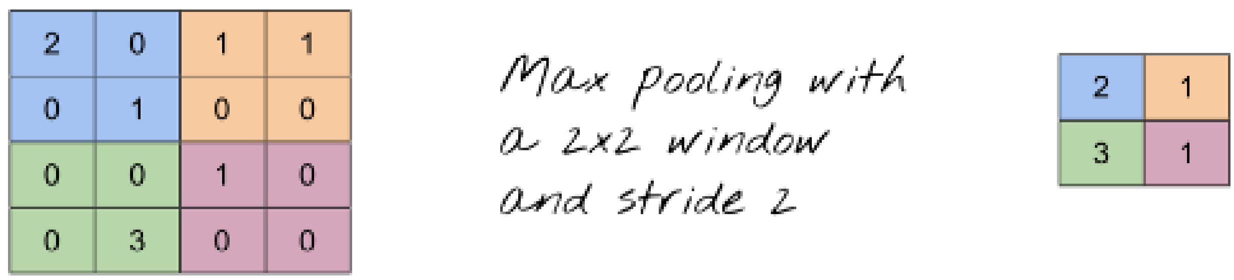
They are used to reduce the depth (number of channels). Width and height are unchanged in this case. If you want to reduce the horizontal dimensions, you would use pooling, increase the stride of the convolution, or don't add paddings. The 1D convolutions computes a weighted sum of input channels or features, which allow **selecting certain combinations of features that are useful downstream. 1D convolution compresses because there is only one It has a same effect of**

Pooling

Note that pooling is a separate step from convolution. Pooling is used to reduce the image size of width and height. Note that the depth is determined by the number of channels. As the name suggests, all it does is it picks the maximum value in a certain

Get started

Max pooling is used to reduce the image size by mapping the size of a given window into a single result by taking the maximum value of the elements in the window.



http://cs231n.github.io/convolutional-networks/

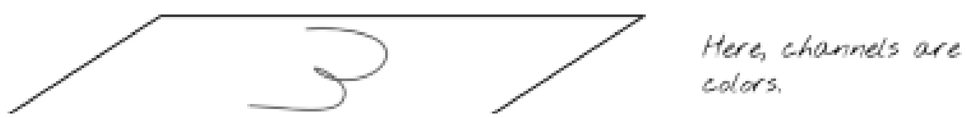
Average-Pooling

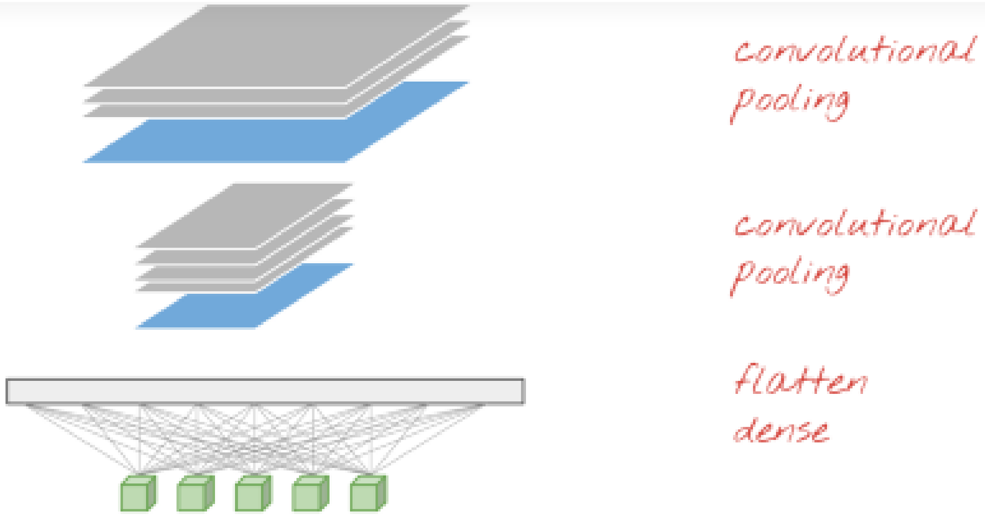
It’s same as max-pooling except that it averages the windows instead of picking the maximum value.

http://cs231n.github.io/convolutional-networks/

Common Set-up

In order to implement CNNs, most successful architecture uses one or more stacks of convolution + pool layers with relu activation, followed by a flatten layer then one or two dense layers.





As we move through the network, feature maps become smaller spatially, and increase in depth. Features become increasingly abstract and lose spatial information. For example, the network understands that the image contained an eye, but it is not sure where it was.

Here's an example of a typical CNN network in Keras.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(28, 28, 1),  rows, cols, color channels
                 padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Here's the result when you do model.summary()

```
Layer (type)              Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)         (None, 28, 28, 32)        320
_________________________________________________________________
```

```
dense_1 (Dense)          (None, 128)         401536
_________________________________________________________________
dense_2 (Dense)          (None, 10)          1290
=================================================================
Total params: 421,612
```

Let's break those layers down and see how we get those parameter numbers.

Conv2d_1

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
                                   input type    # of filters or depth
_________________________________________________________________
```

Filter size (3 x 3) * input depth (1) * # of filters (32) + Bias 1/filter (32) = 320. Here, the input depth is 1, because it's for MNIST black and white data. Note that in tensorflow by default every convolution layer has bias added.

Max_pooling2d_1

```
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
```

Pooling layers don't have parameters

Conv2d_2

```
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
                                              input depth
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496
                                              # of filters or depth
_________________________________________________________________
```

Filter size (3 x 3) * input depth (32) * # of filters (64) + Bias, 1 per filter (64) = 18496

It unstacks the volume above it into an array.

https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd

Dense_1

flatten_1 (Flatten)	(None, 3136)	0
dense_1 (Dense)	(None, 128)	401536

Input Dimension (128) * Output Dimension (10) + One bias per output neuron (10) = 1290

Summary

Convolutional Neural Network (CNN) is a class of deep neural network (DNN) which is widely used for computer vision or NLP. During the training process, the network's building blocks are repeatedly altered in order for the network to reach optimal performance and to classify images and objects as accurately as possible.

Sources

This tutorial is based on lectures from the Applied Deep Learning course at Columbia University by Joshua Gordon. Awesome 3d Images are from Martin Gorner .

196 claps

WRITTEN BY

Suhyun Kim

https://suhyunkim.net/

Follow

Towards Data Science

Sharing concepts, ideas, and codes.

Follow

See responses (1)

More From Medium

More from Towards Data Science

The 5 Sampling Algorithms every Data Scientist need to know

Rahul Agarwal in Towards Data Science
Jul 21 · 5 min read ★

606

More from Towards Data Science

Hands On Bayesian Statistics with Python, PyMC3 & ArviZ

Susan Li in Towards Data Science

Get started

Towards Data Science

12 Things I Learned During My First Year as a Machine Learning Engineer

Daniel Bourke in Towards Data Science
Jul 6 · 11 min read ★

6.1K