

Cq_CNN_Intro

2021-10-10 19:04:01

Contents

Some sources	1
Overview	2
Convolution and Cross-correlation	2
CS231n Notes	2
Conv Layer	4
Pooling Layer	6
Comprehensive Guide Notes	6
Reference Test	10
ReferenceHead2	10
testReferences	10

Some sources

Sources:

A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way |
by Sumit Saha | Towards Data Science

vdumoulin/conv_arithmetic: A technical report on convolution arithmetic in
the context of deep learning

Convolutional neural network - Wikipedia

Convolution - Wikipedia

CS231n Convolutional Neural Networks for Visual Recognition

Convolutional Neural Networks, Explained | by Mayank Mishra | Towards Data
Science

An Intuitive Explanation of Convolutional Neural Networks – the data science blog

Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks | by Adam Geitgey | Medium

Overview

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolutional networks are a specialized type of neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Here is an inline note.¹

Convolution and Cross-correlation

In mathematics (in particular, functional analysis), **convolution** is a mathematical operation on two functions (f and g) that produces a third function ($f * g$) that expresses how the shape of one is modified by the other. The term convolution refers to both the result function and to the process of computing it. It is defined as the integral of the product of the two functions after one is reversed and shifted. The integral is evaluated for all values of shift, producing the convolution function. Some features of convolution are similar to cross-correlation.

Cross-correlation

In signal processing, **cross-correlation** is a measure of similarity of two series as a function of the displacement of one relative to the other. This is also known as a *sliding dot product* or *sliding inner-product*. It is commonly used for searching a long signal for a shorter, known feature.

CS231n Notes

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product

¹Inlines notes are easier to write, since you don't have to pick an identifier and move down to type the note.

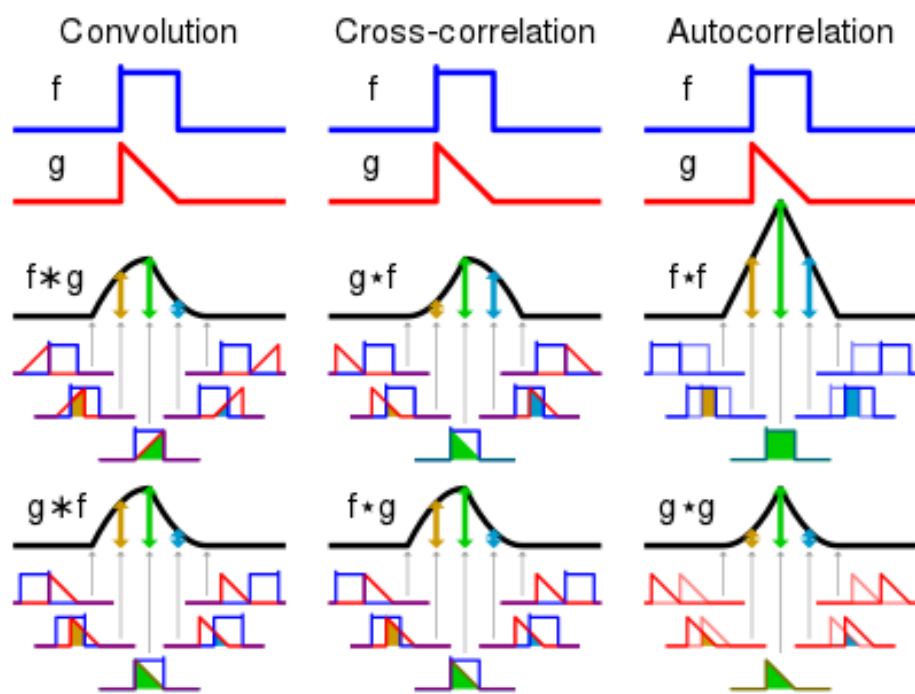


Figure 1: first fig text.

and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

Neural Networks receive an input (a single vector), and transform it through a series of *hidden layers*. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the “output layer” and in classification settings it represents the class scores.

For example, an image of more respectable size, e.g. $200 \times 200 \times 3$, would lead to neurons that have $200 \times 200 \times 3 = 120,000$ weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Conv Layer

The CONV layer’s parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size $5 \times 5 \times 3$ (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer.

Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

If you’re a fan of the brain/neuron analogies, every entry in the 3D output volume can also be interpreted as an output of a neuron that looks at only a small region in the input and shares parameters with all neurons to the left and right spatially (since these numbers all result from applying the same filter).

Example 1. For example, suppose that the input volume has size $[32 \times 32 \times 3]$, (e.g. an RGB CIFAR-10 image). If the receptive field (or the filter size) is 5×5 , then each neuron in the Conv Layer will have weights to a $[5 \times 5 \times 3]$ region in the input volume, for a total of $5 \times 5 \times 3 = 75$ weights (and +1 bias parameter). Notice

that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.

Three hyperparameters control the size of the output volume: the **depth**, **stride** and **zero-padding**. We discuss these next:

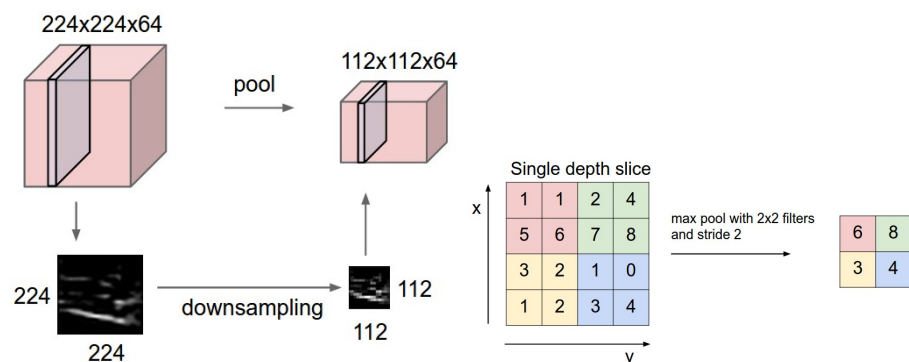
1. First, the **depth** of the output volume is a hyperparameter: it corresponds to the number of filters we would like to use, each learning to look for something different in the input. For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a **depth column** (some people also prefer the term *fibre*).
2. Second, we must specify the **stride** with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.
3. As we will soon see, sometimes it will be convenient to pad the input volume with zeros around the border. The size of this **zero-padding** is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes.

1x1 convolution. As an aside, several papers use 1x1 convolutions, as first investigated by Network in Network. Some people are at first confused to see 1x1 convolutions especially when they come from signal processing background. Normally signals are 2-dimensional so 1x1 convolutions do not make sense (it's just pointwise scaling). However, in ConvNets this is not the case because one must remember that we operate over 3-dimensional volumes, and that the filters always extend through the full depth of the input volume. For example, if the input is $[32 \times 32 \times 3]$ then doing 1x1 convolutions would effectively be doing 3-dimensional dot products (since the input depth is 3 channels).

Dilated convolutions. A recent development (e.g. see paper by Fisher Yu and Vladlen Koltun) is to introduce one more hyperparameter to the CONV layer called the *dilation*. So far we've only discussed CONV filters that are contiguous. However, it's possible to have filters that have spaces between each cell, called dilation. As an example, in one dimension a filter w of size 3 would compute over input x the following: $w[0]*x[0] + w[1]*x[1] + w[2]*x[2]$. This is dilation of 0. For dilation 1 the filter would instead compute $w[0]*x[0] + w[1]*x[2] + w[2]*x[4]$; In other words there is a gap of 1 between the applications. This can be very useful in some settings to use in conjunction with 0-dilated filters because it allows you to merge spatial information across the inputs much more aggressively with fewer layers.

Pooling Layer

General pooling. In addition to max pooling, the pooling units can also perform other functions, such as *average pooling* or even *L2-norm pooling*. Average pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice.



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size $[224 \times 224 \times 64]$ is pooled with filter size 2, stride 2 into output volume of size $[112 \times 112 \times 64]$. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2×2 square).

Getting rid of pooling. Many people dislike the pooling operation and think that we can get away without it. For example, Striving for Simplicity: The All Convolutional Net proposes to discard the pooling layer in favor of architecture that only consists of repeated CONV layers. To reduce the size of the representation they suggest using larger stride in CONV layer once in a while. Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs). It seems likely that future architectures will feature very few to no pooling layers.

Comprehensive Guide Notes

A digital image is a binary representation of visual data. It contains a series of pixels arranged in a grid-like fashion that contains pixel values to denote how bright and what color each pixel should be.

Essentially, every image can be represented as a matrix of pixel values.

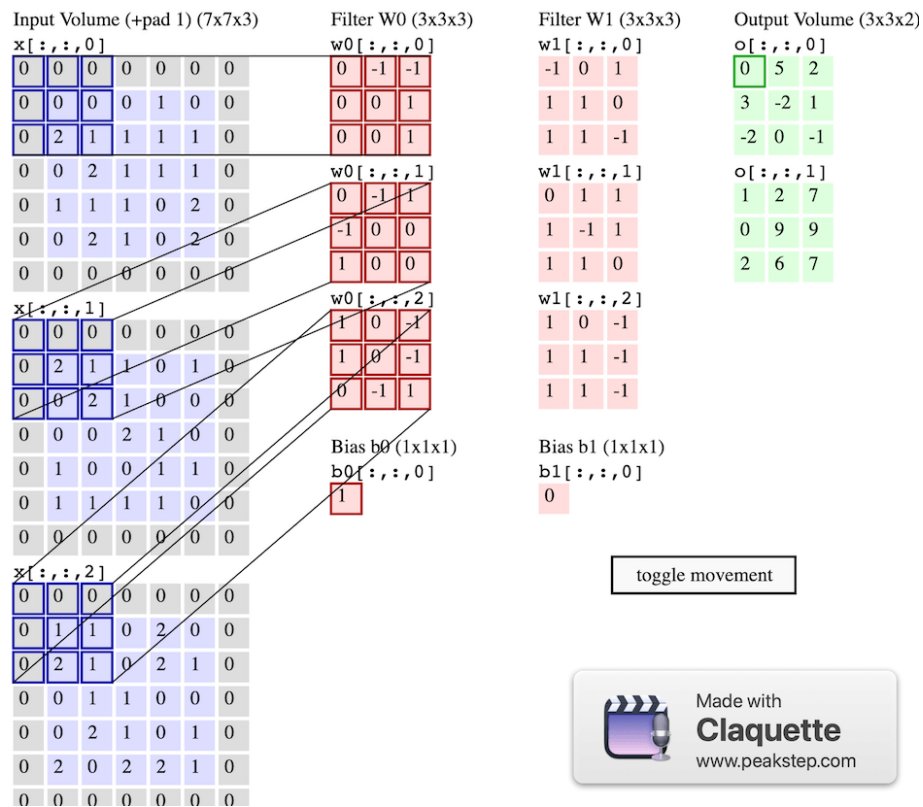


Figure 2: This is a gif.

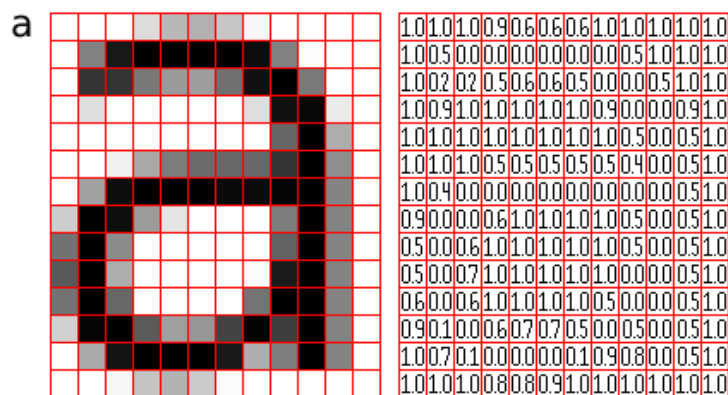
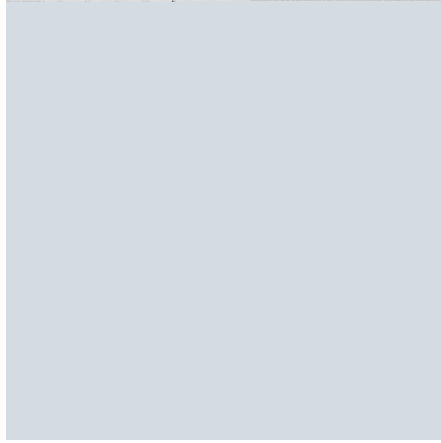
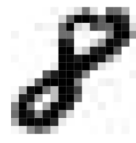


Figure 3: Representation of image as a grid of pixels.



To feed an image into our neural network, we simply treat the 18x18 pixel image as an array of 324 numbers:

 = `[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0, 0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2, 55, 233, 40, 0, 0, 10, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 107, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0, 38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 4, 0, 163, 225, 251, 255, 229, 120, 0, 0, 0, 0, 11, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 10, 14, 6, 0, 0, 9, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 145, 248, 228, 116, 235, 255, 141, 34, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32, 0, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`

We need to give our neural network understanding of *translation invariance* — an “8” is an “8” no matter where in the picture it shows up.

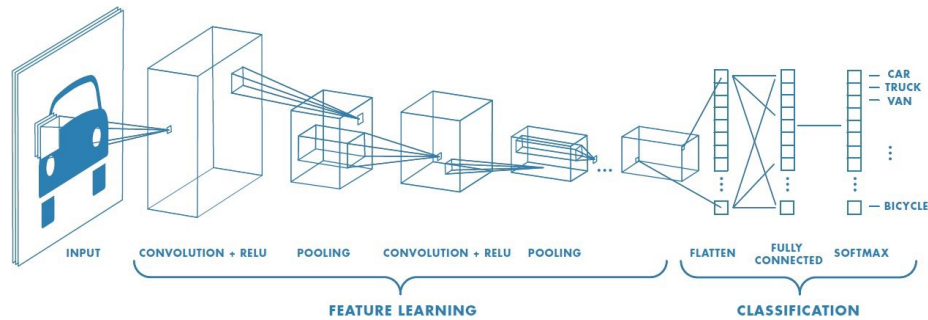


Figure 4: A general CNN flow.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

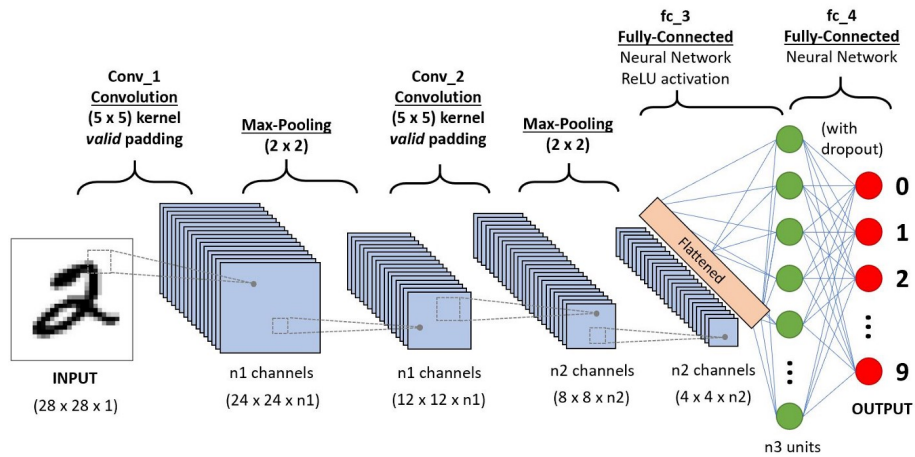


Figure 5: A CNN sequence to classify handwritten digits.

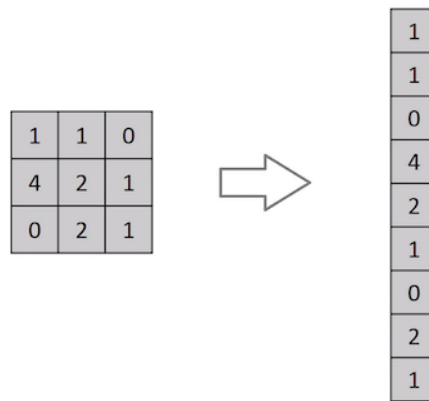


Figure 6: Flattening of a 3x3 image matrix into a 9x1 vector.

Reference Test

This is to fig1 fig. 1 and fig2 fig. 2 over. This is not working.

Try this #fig:firstFig over.

And second gif fig #fig:giffig over.

This way of referencing figs is actually working, except the figure number is not auto generated, we have to manually set a text.

ReferenceHead2

And some test

testReferences

So here is the way what Max C. Foo(2021){% ref mcf-2021 %} do.

{% references %} [mcf-2021] Max C. Foo. A way to write an article.[J] Journal of Kelaideng University Samwin School. 2021.3 300-321. {% endreferences %}