CSCI 5525 Machine Learning Fall 2019

Lecture 11: Neural Networks (Part 3)

March 2nd, 2020

Lecturer: Steven Wu *Scribe: Steven Wu*

1 Convolutional Neural Networks

We will now study a special type of neural networks–*convolutional neural networks* (CNN)–that is especially powerful for computer vision. Let us start with the mathematical ideas behind CNN.

1.1 Convolutions

A *convolution* of two functions f and g is defined as

$$(f * g)(t) = \int f(a)\, g(t-a) da$$

The first argument f to the convolution is often referred to as the *input*, and the second argument g is called the *kernel, filter, or receptive field*.[1]

Motivating example from [1]. "Suppose we are tracking the location of a spaceship with a sensor. The sensor provides $x(t)$, the position of the spaceship at each time step t. Both x and t are real valued, that is, we can get a different reading from the lasersensor at any instant in time. To obtain a less noisy estimate of the spaceship's position, we would like to average several measurements. Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements. We can do this with a weighting function $w(a)$, where a is the age of a measurement."

$$s(t) = \int x(a)\, w(t-a) da = (x * w)(t)$$

In machine learning, we often use *discrete convolutions*: given two functions (or vectors) $f, g \colon \mathbb{Z} \to \mathbb{R}$

$$(f * g)(t) = \sum_{a=-\infty}^{\infty} f(a)\, g(n-a)$$

We often apply convolution over higher dimensional space. In the case of images, we have two-dimensional convolutions:

$$(f * g)(t, r) = \sum_i \sum_j f(i, j)\, g(t-i, r-j)$$

[1]Many things in math and engineering are called kernels.

Convolutions enjoy the commutative property, which means that we can flip the arguments to the two functions:

$$(f * g)(t, r) = \sum_i \sum_j f(t - i, r - j)\, g(i, j)$$

A related concept is the *cross-correlation*:

$$(f * g)(t, r) = \sum_i \sum_j f(t + i, r + j)\, g(i, j)$$

Many machine learning libraries implement cross-correlation and call it convolution [1]. I personally find cross-correlation more intuitive. See Figure 1 for an illustration.

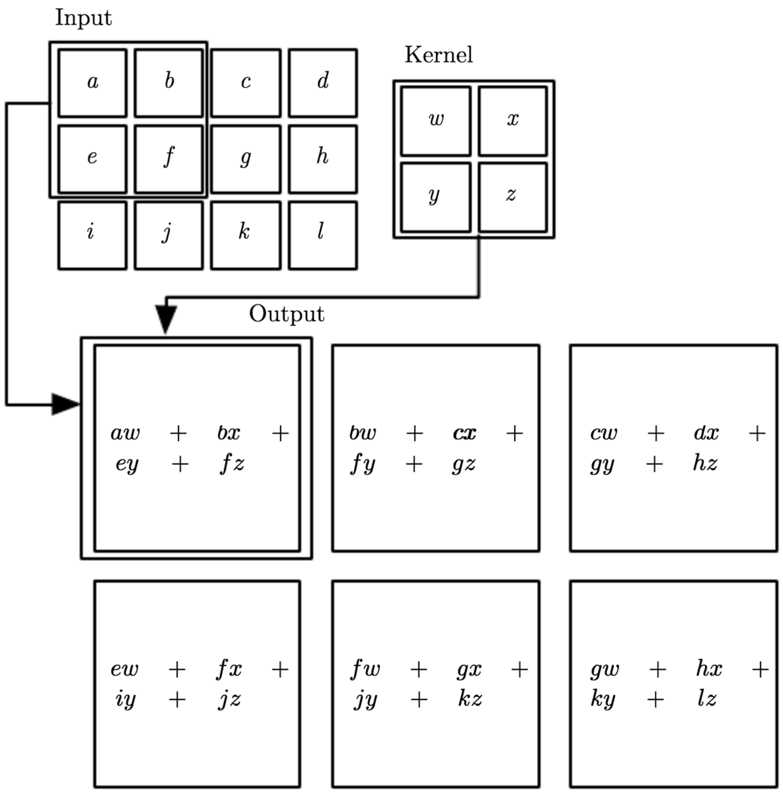

Figure 1: Example of 2D cross-correlation from [1].

Convolutions as feature extraction. With different choicese of filters, the convolution operation can perform different types of feature extractions, including edge detection, sharpening, and blurring. See Figure 2 for the example of edge detection with filter

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

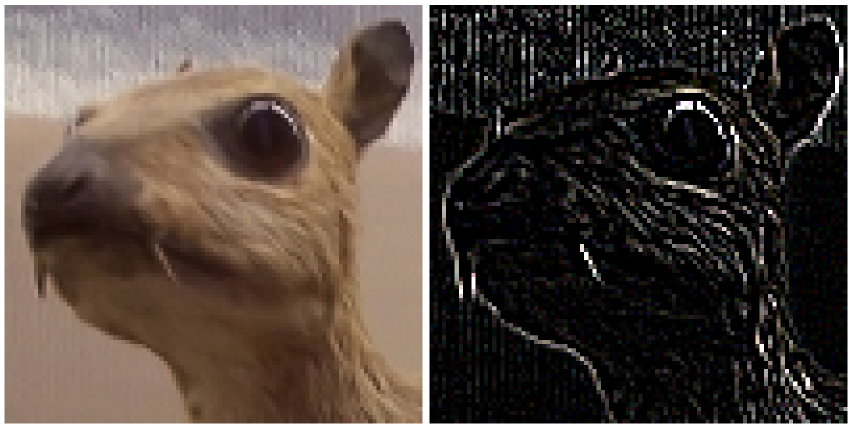

Figure 2: Edge detection example (image source).

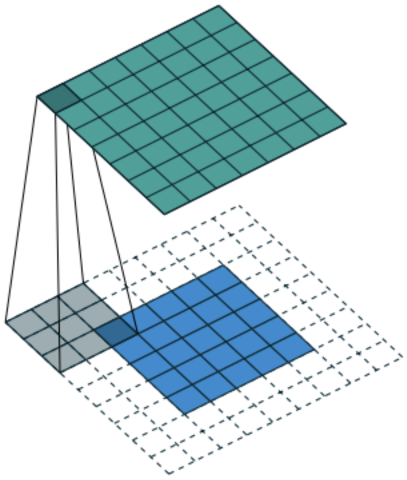

Figure 3: Padding

Check out this page on Wikipedia for more examples. We can use multiple filters to extract different features.

1.2 Convolution Parameters

Variants. There are also variants of on how we apply the convolutional mapping:

- **Padding**: surround the input matrix with zeroes (Figure 3).
- **Strides**: shifting the kernel window more than one unit at a time (Figure 4).
- **Dilation**: convolution applied input with defined gaps, using filter of larger size (Figure 5).

Check out more animations here in this github repo.

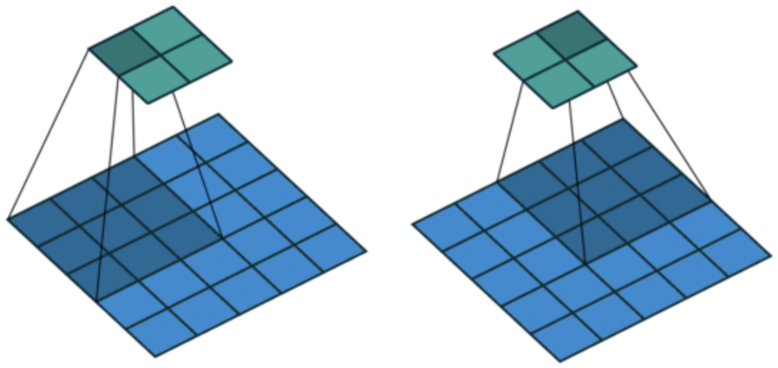
Figure 4: Strided convolution

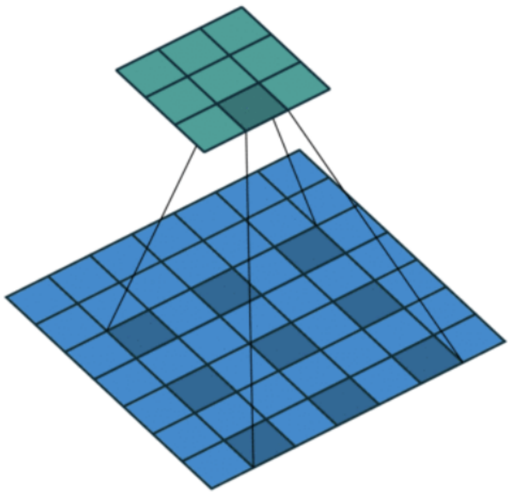
Figure 5: Dilation

Reduction on number of parameters. Recall that for fully-connected (FC) layers, we need to keep track of a weight coefficient W_{ij} for every pair of nodes i and j across the two layers. In contrast, for convolutional layers, the number of parameters is much smaller–since we only need to maintain the weights in the kernel matrix and in some cases, a bias term.

1.3 Convolutional Neural Network (CNN) Architecture

A typical CNN will combine convolutional layers with non-linearity mappings (typically ReLU) and pooling layers (typically max pooling), which provides downsampling and compression. Average pooling also has similar effects. See Figure 7.

A typical CNN for computer vision has the following architecure:

$$\text{Input } x \to \left[(\text{Conv} \to \text{ReLU})^N \to \text{(Max) Pooling}\right]^M \to [\text{FC} \to \text{ReLU}]^P \to \text{FC}$$

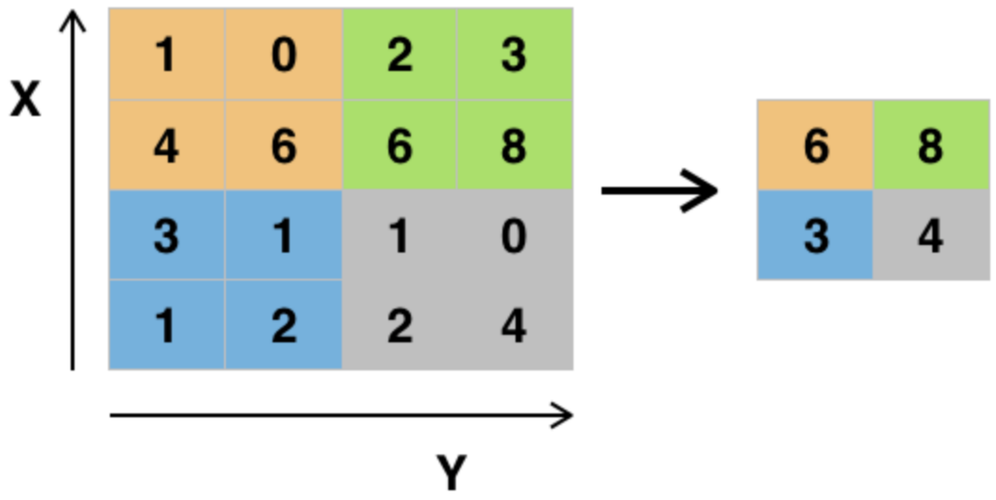

Figure 6: Max pooling layer compresses the repsentation further.

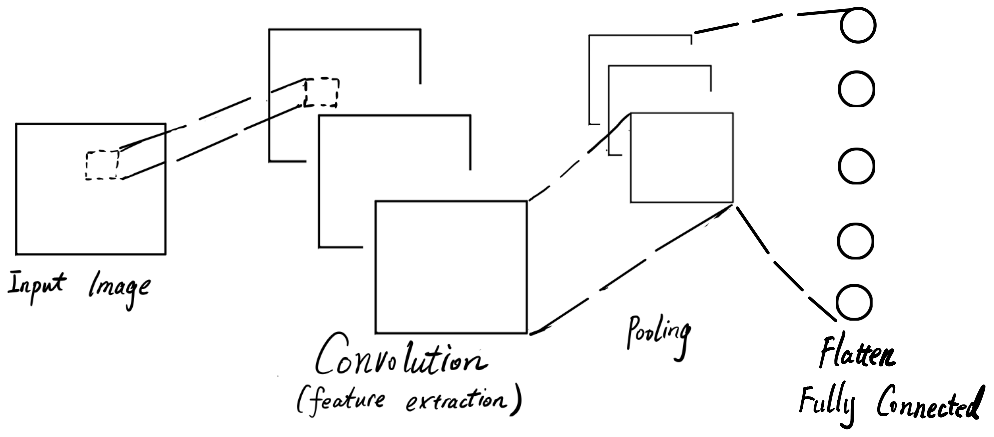

Figure 7: Combination of convolution, pooling, and fully connected layers.

2 Training and Optimization

There are many alternatives to SGD. See more examples here.

Nesterov Acceleration Initially: let $v_0 = 0$ and w_0 be arbitrary. The update step:

$$v_{i+1} = w_i - \eta \nabla_w F(w)$$
$$w_{i+1} = v_{i+1} + \frac{i+1}{i+4}(v_{i+1} - v_i)$$

Newton methods Newton iteration:

$$w' = w - \eta \left(\nabla^2 F(w)\right)^{-1} \nabla F(w)$$

An advantage of this method is that it does not have a learning rate η, which saves some tuning. For certain cases, this converges faster in terms of number of iterations. However, per-iteration computation involves computing the Hessian matrix, which can be prohibitively expensive.

2.1 Training tricks

Random initialization. Initializing all the weights to be zero is a bad idea in general, since all the neurons will be computing the same functions even after gradient descent udpates. Common practice is to randomly initialize the weights.

Batch normalization A useful technique that seems to accelerate training of neural networks is *batch normalization*, which performs a standardization of the node outputs. Standardization is a method of rescaling the feature. For each feature j and a batch of data $x_1, x_2, \ldots x_m \in \mathbb{R}^d$, we can transform the feature as

$$\tilde{x}_{ij} = \frac{x_{ij} - \overline{x}_j}{\sigma_j}$$

where $\overline{x}_j = \frac{1}{m}\sum_i x_{ij}$ and $\sigma_j^2 = \frac{1}{m}\sum_i x_{ij}^2$. The rescaled feature vectors will then have mean zero, and unit variance in each coordinate. This is a useful technique for training linear models for linear regression or logistic regression. Sometimes we replace σ_j in the denominator by $\sqrt{\sigma_j^2 + \epsilon}$ for some small value of ϵ. Batch normalization is simply applying the standardization method to the input to the activation function at each node.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.