


- Use Convolutional Neural Networks for Image Analysis

 Module Introduction: Use Convolutional Neural Networks for Image Analysis

Neural Networks for
Images

Formalize Convolution


Review Convolutional Networks

Define the Convolutional Operation

 [Explore Image Convolution](#)

Identify Patterns in Images

 [Explore Pooling Layers](#)

 Visualize a Convolutional Network in Action

Stabilize Training with Batch Normalization


Formalize Batch Normalization

Preserve Information with Residual and Dense Networks

 ConvNet Cheat Sheet

PyTorch Tutorial

Implement a Convolutional Network with PyTorch

 Module Wrap-up: Use Convolutional Neural Networks for Image Analysis

- › Analyze Sequence Data with Neural Sequence Models

› Course Resources

Deep Learning and Neural Networks >

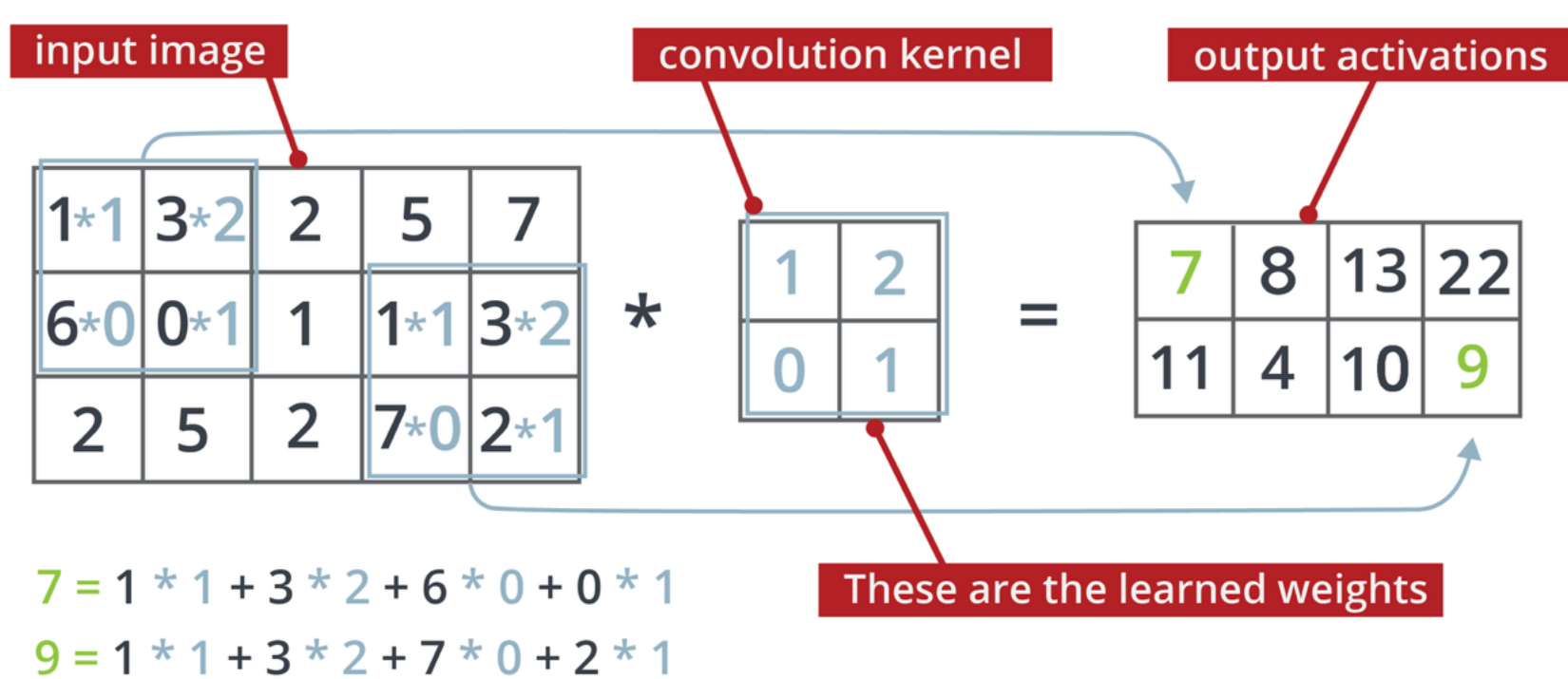
Define the Convolutional Operation

The core of ConvNets are the convolutional layers.

The convolutional layers consist of small convolutional filters (also called convolutional kernels — note, **not** related to the kernel trick) that are learned to detect certain patterns in the images. These small filters are applied to images through the convolution operation (known as cross correlation in signal processing).

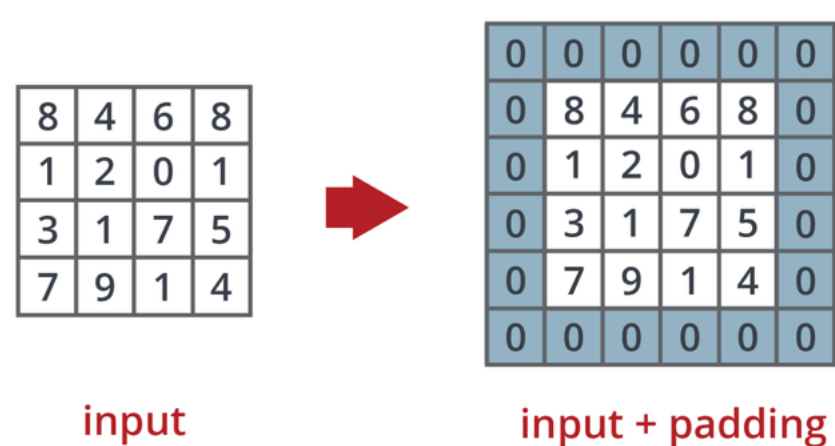
This operation is best illustrated with an example:

Convolution Operation



There are a few design choices when it comes to convolutional layers that we have to make:

- **Number of filters:** You can have multiple filters per layer, each finding their own patterns. For example, you could have a filter that would look for vertical lines and a different filter that would look for horizontal ones, both acting on the same layer. Importantly, what the filters are doing will be learned from the data, but you can decide how many there are. Since more layers means slower training time on a CNN, one can mitigate this by using fewer layers and more filters per layer (i.e., the network is wider and less deep).
- **Size of the convolutional filter:** The most popular size of convolutional filters is 3×3 . This is because it is both a *small* and *odd* numbered.
 - Small filters look at few pixels at a time, so they can only find small, local features. Larger filters look at a bigger "window" of pixels, so they can find larger patterns that are more generic.
 - If the network is deep (i.e., has many layers) small filters are often preferred to reduce the number of parameters; larger filters typically have the advantage that they consider a wider context, but this is less relevant in deep networks, where features in deep layers are influenced from a wide area of input pixels.
- **Whether to pad the images**
 - Applying convolution to images will reduce the size of the image, and sometimes we want to make sure the size of the output layer is the same as the input image so we **pad** the images with zeros around the borders. (Alternatively, one can also pad with pixels from the image, for example by considering the edge of the image as a mirror.)



- **Stride size**
 - **Stride** is the amount by which the filter is moved by as it passes over the image. For a dense representation of the image, we can shift the filter by 1 pixel at a time. To train fewer nodes, we can shift the filter by more pixels — for example, with a stride of the size of the filter. A large stride increases the area of influence for pixels in activations in deeper layers.

