10-315 Notes
Convolutional Neural Networks

Carnegie Mellon University
Machine Learning Department

Contents

1 Convolutions

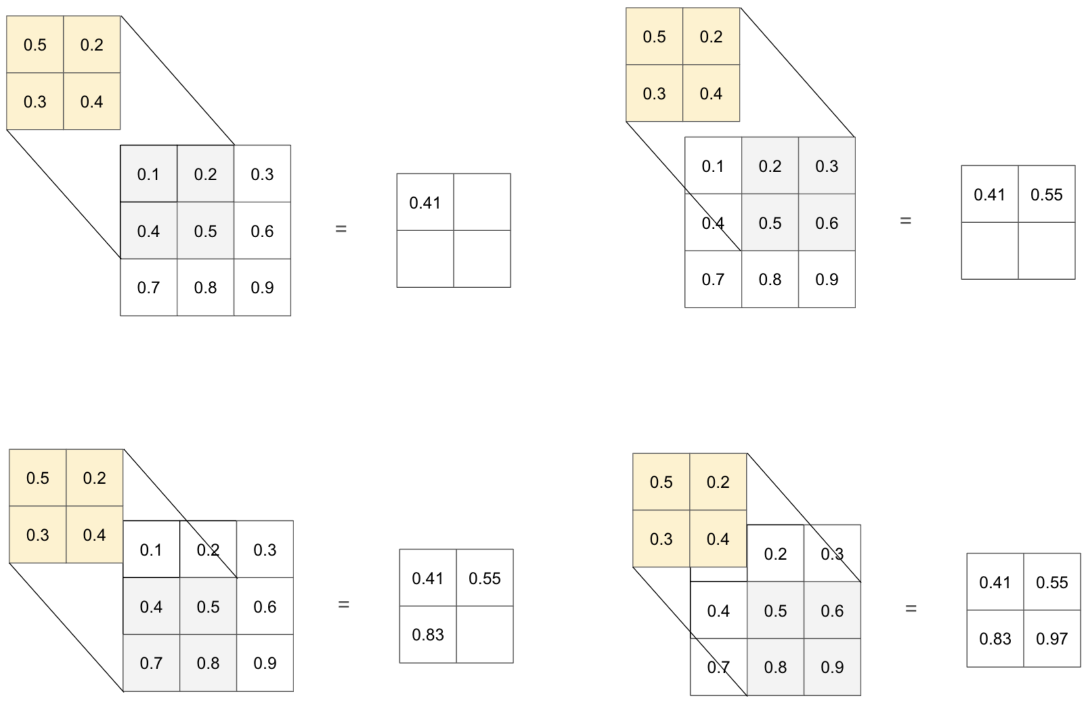
Convolutional neural networks (CNNs) are a category of neural networks that are well-suited for processing image data (although it can also be useful for certain 1-dimensional or N-dimensional data). The distinctive feature of CNNs is the use of convolution layers – so let's dive into what a convolution is.

1.1 Convolutions and Kernels

A convolution is a mathematical operation that involves "sliding" a window across the input data and computing the product between the window and the data at the overlapping points, and then adding the products to get the final output. In the context of 2D data (like images), the window moves from left to right and from top to bottom. This window is usually small, square, and of fixed size (around 3x3 or 5x5) and is called a kernel or filter (in this course, we use these two terms interchangeably, so don't get confused!).

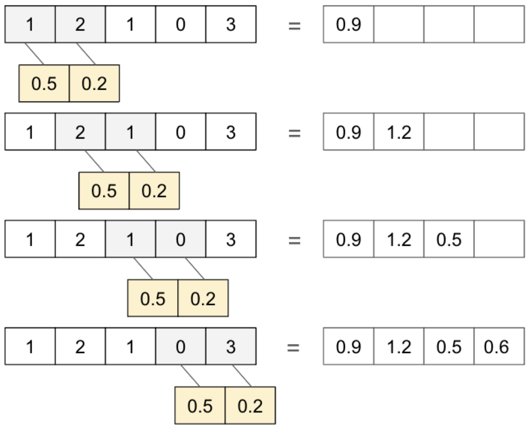
More formally, the equation for a convolution looks like this, where x is the 2-D input over which we are performing the convolution, and w is the 2-D kernel.

$$z[i,j] = \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} x[i+u, j+v] \cdot w[u,v]$$



1.2 1D Convolutions

When talking about CNNs, we will primarily focus on 2D convolutions and how they are applied to images, but it is important to note that convolutions can be in any number of dimensions. 1D convolutions are also used in neural networks to process sequential data (text, time series, audio, etc.). 1D convolutions work the same way as 2D, where we have a 1-dimensional array as our input and slide a smaller 1D kernel over the array, computing the dot product at each instance.



1.3 Hand-crafted Kernels for Image Processing

When applied to images, different kernels produce different effects on the input images and can be used to detect features of the input image. Some common effects are blurring, edge detection, or sharpening the image.

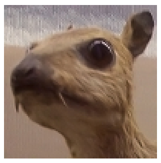
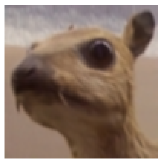
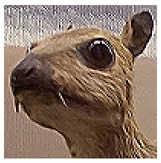
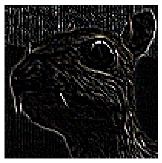
Original	*Gaussian Blur*	*Sharpen*	*Edge Detection*
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
			

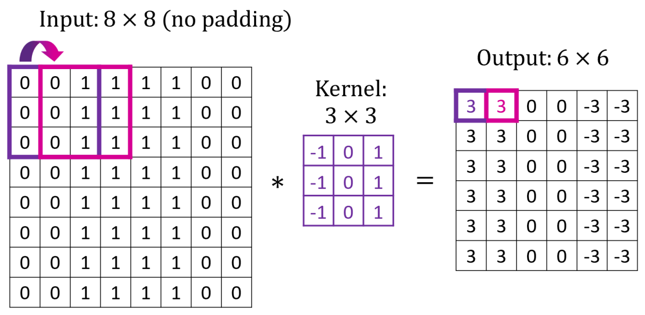
Image source: https://medium.com/@abhishekjainindore24/all-about-convolutions-kernels-features-in-cnn-c656616390a1

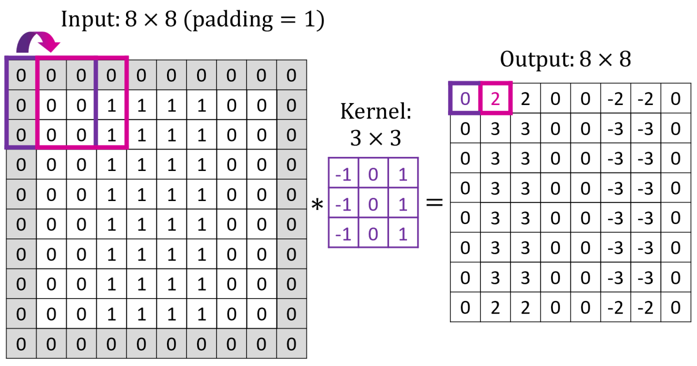
Creating kernels to detect features used to be a manual and time-consuming task, but the breakthrough for convolutional neural networks came when we used deep learning to learn the weight values within the kernels instead, which we will talk about in more detail.

1.4 Padding

Padding refers to adding extra pixels to the outsides of the layer image to make the output dimensions have the desired size after the convolution is applied. The extra pixels can be filled with zeros or with a copy of the data on the border.

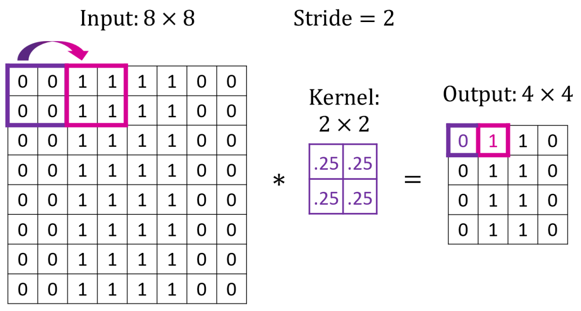
Without padding, the output image will be smaller than the input. Using a stride of 1 with padding will ensure that the output remains the same size as the input image, while increasing the stride will result in smaller outputs.



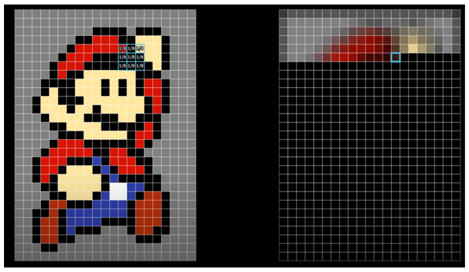


1.5 Stride

Stride represents the amount by which we slide the kernel across the input data. For example, a stride of 1 would mean that we move the kernel by one pixel each time, while a stride of 2 means that we would move it by 2 pixels. Typically, the same stride is applied across both rows and across columns.



1.6 Additional Resources



If you want to learn more about how convolutions work or if you're more of a visual learner, this video provides a great overview of convolutions!
3Blue 1Brown: But what is a convolution?

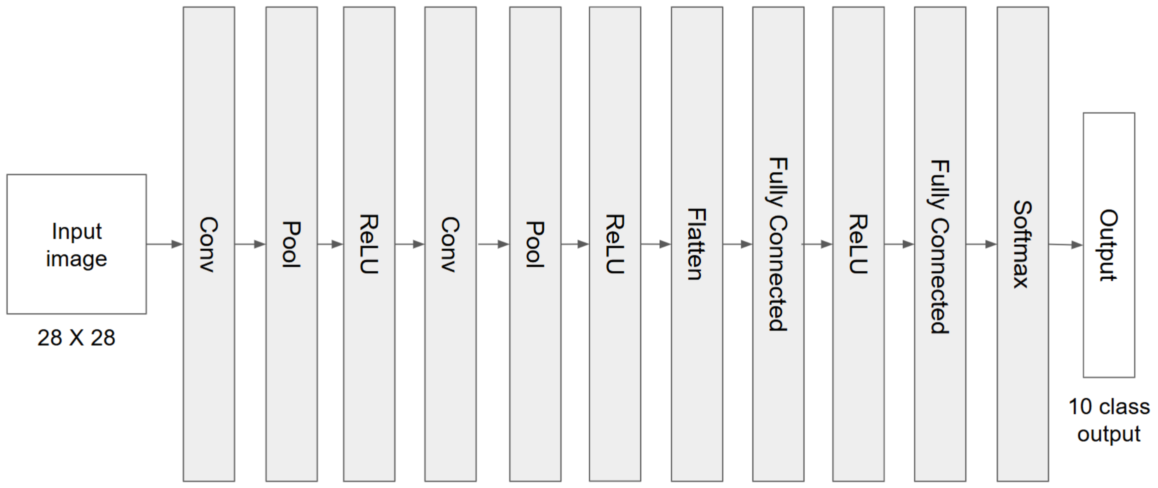
2 Convolutional Neural Networks

CNNs are made up of four main types of layers:

1. Convolutional Layer
2. Pooling Layer
3. Fully Connected Layer
4. Activation Layers (e.g. ReLU)

The pooling layer and the ReLU layer do not have any parameters (although the pooling layer does have hyperparameters).

CNNs can be used to process both black and white images with a single channel, as well as images with multiple channels (such as RGB images with 3 channels). In the next sections, we will focus on single-channel networks, but the same concepts can be extended to consider multiple channels. Below is a simple CNN architecture that takes in MNIST images (28×28 in size) and outputs a score for each of the 10 classes for classification.



2.1 Convolution Layer

The convolution layer is the first layer of a CNN and consists of a set of filters that are learned during the training process. During the forward pass, we perform the convolution operation for each filter. Each filter will have weights associated with it and a single bias term that is added to the convolution. With multiple filters, each one produces a different feature map which are stacked to product the output of the layer. The weights and bias of each filter adjust during backpropagation. Intuitively, each of these filters will learn different patterns in the image (such as edges or certain textures). The convolutional layer is often followed by a ReLU activation which adds some nonlinearities, and could have additional convolutional or pooling layers after it.

It is important to note that as the kernel moves across the image, the weights remain fixed – this is known as parameter sharing.

Some of the hyperparameters that need to be set for this layer are the size of the filter, the number of filters, stride, and padding.

2.2 Pooling Layer

Pooling or downsampling is a technique used to downsample the image or reduce its spatial dimensions while retaining relevant information. It helps to reduce the complexity of the model by reducing the number of parameters and computation needed, and also controls overfitting. Similarly to a convolution, it involves sliding a window over the image. But in this case, the window does not have any weights but just summarizes or aggregates the value within that region. Two commonly used pooling operations are max pooling and average pooling.

1. **Max pooling:** Selects the element with the highest/maximum value within the pooling region.
2. **Average/Mean pooling:** Takes the average value of the elements within the pooling region.

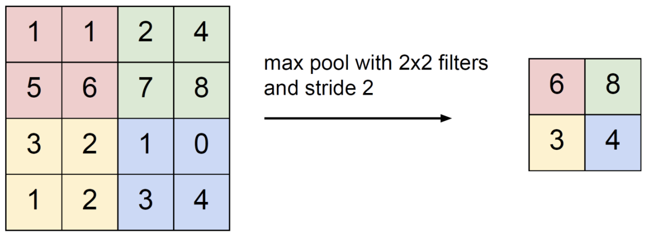


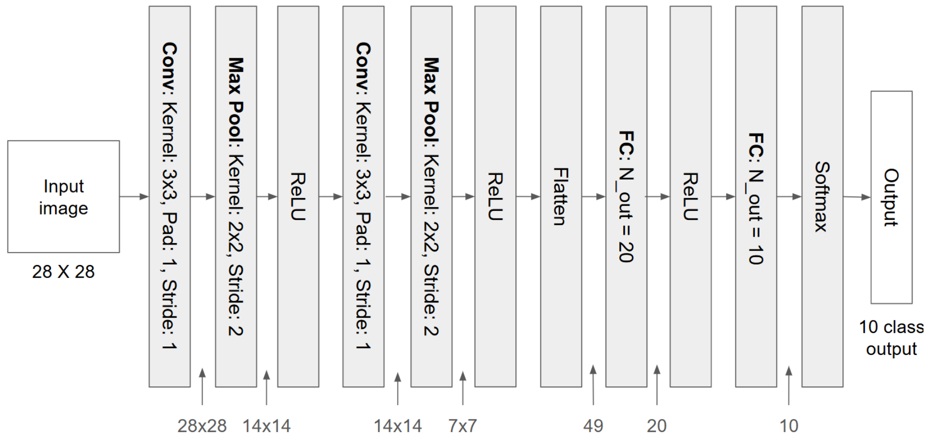
Image source: Stanford CS 231n, Spring 2017

2.3 Fully Connected Layer

After a series of conv-pool-activation layers, CNN architectures typically finish with a series of fully connected layers (interleaved with activation layers). In order to convert from the 2D representation used in the convolution layers, the output of the final convolution or pooling layer is flattened to a 1D vector and before being passed to the fully connected layers.

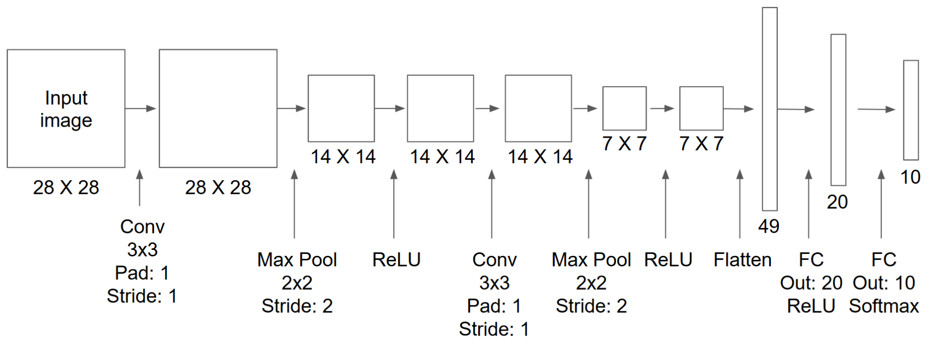
2.4 Output sizes: Single channel version

Depending on the hyperparameters that we use in the network architecture, the convolution and pooling functions can produce output feature maps that are of different dimensions from the input. Now, we can add hyperparameters to our simple CNN from above. Each of the conv layers uses a 3×3 kernel with stride 1 and padding 1. The pooling layers use max pooling with a 2×2 kernel and a stride of 2. The fully connected layers have 20 and 10 output neurons, respectively. Let us see how we can determine the output sizes at each layer.



Input image
28 X 28
Conv: Kernel: 3x3, Pad: 1, Stride: 1
Max Pool: Kernel: 2x2, Stride: 2
ReLU
Conv: Kernel: 3x3, Pad: 1, Stride: 1
Max Pool: Kernel: 2x2, Stride: 2
ReLU
Flatten
FC: N_out = 20
ReLU
FC: N_out = 10
Softmax
Output
10 class output
28x28
14x14
14x14
7x7
49
20
10

Alternative view (boxes represent data rather than layer functions)



For a convolution layer, the output shape is influenced by the input image size ($W \times H$), the filter size ($K \times K$), stride (S), and padding (P):

- W: Input image width
- H: Input image height
- K: Kernel width and height (these are often the same)
- S: Stride (often the same vertically and horizontally)
- P: Padding applied to each side (often the same vertically and horizontally)

The output size,$W' \times H'$, is basically just how many times can the sliding kernel fit within the padded input. Specifically:

$$W' = \lfloor (W + 2P - K)/S + 1 \rfloor$$
$$H' = \lfloor (H + 2P - K)/S + 1 \rfloor$$

The output size for a pooling layer follows the same formula as convolution.

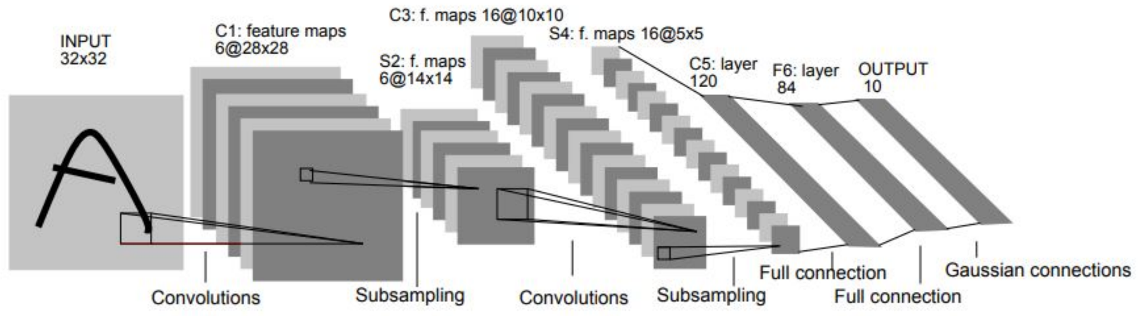
2.5 Counting Parameters: Single channel version

Now that we know the input and output size of our layers, let's take a look at the number of parameters in each layer of a CNN. Consider the input size as $W \times H$ and the filter size $K \times K$.

1. Convolution Layer: This layer will need to learn the weights and biases for the filters. So the number of parameters in each filter will come out to $K^2 + 1$ (1 for the bias term).
2. Pooling Layer: While this layer has a filter associated with it, there are no weights/biases associated with it. Since the filter only performs a fixed operation over the pixels, there are no parameters associated with the pooling layer.
3. Fully Connected Layer: We've already looked at fully connected layers: If the current layer has N neurons and the previous layer has M neurons, the total number of parameters will be $N(M + 1)$

2.6 Up Next: Multiple Channels

The next step towards understanding convolutional layers is adding multiple feature channels to the convolution layers. In the classic LeNet5 CNN architecture figure below, each convolution layer uses multiple filters, and the output feature maps are concatenated to form the input to the next layer. We will discuss how this works in more detail during lecture!


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections