Convolutional Neural Networks

Spring 2020
Neural network architectures

• Full connectivity is a problem for image inputs
  • Scalability: 200x200x3 images imply 120,000 weights per neuron in first hidden layer
  • Overfitting: Too many parameters would lead to overfitting
Convolutional Neural Networks [LeCun 1989]

• Specialized to the case where inputs are images (more generally, data with a grid-like topology)

• Sparse connections, parameter sharing
  • Efficient to train
  • Avoid overfitting

• Generalize across spatial translations of input
  • By sliding “filters” that learn distinct patterns (edges, blobs of color etc.)
Key idea

• Replace matrix multiplication in neural networks with **convolution**

• Everything else remains the same
Edge detection by convolution

Figure 9.6: Efficiency of edge detection. The image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all of the vertically oriented edges in the input image, which can be a useful operation for object detection. Both images are 280 pixels tall. The input image is 320 pixels wide while the output image is 319 pixels wide. This transformation can be described by a convolution kernel containing two elements, and requires $319 \times 280 \times 3 = 267,960$, floating point operations (two multiplications and one addition per output pixel) to compute using convolution. To describe the same transformation with a matrix multiplication would take $320 \times 280 \times 319 \times 280$, or over eight billion, entries in the matrix, making convolution four billion times more efficient for representing this transformation. The straightforward matrix multiplication algorithm performs over sixteen billion floating point operations, making convolution roughly 60,000 times more efficient computationally. Of course, most of the entries of the matrix would be zero. If we stored only the nonzero entries of the matrix, then both matrix multiplication and convolution would require the same number of floating point operations to compute. The matrix would still need to contain $2 \times 319 \times 280 = 178,640$, entries. Convolution is an extremely efficient way of describing transformations that apply the same linear transformation of a small, local region across the entire input. (Photo credit: Paula Goodfellow)
2D Convolution

Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

Sliding filters (kernels)

Fig. Goodfellow et al. 2016
Sparse connectivity

Figure 9.2: Sparse connectivity, viewed from below: We highlight one input unit, \( x_3 \), and also highlight the output units in \( s \) that are affected by this unit. (Top) When \( s \) is formed by convolution with a kernel of width 3, only the output \( x_3 \) is affected by \( x_3 \). (Bottom) When \( s \) is formed by matrix multiplication, connectivity is no longer sparse, so all of the outputs are affected by \( x_3 \).
Sparse connectivity

Figure 9.3: Sparse connectivity, viewed from above: We highlight one output unit, $s_3$, and also highlight the input units in $x$ that affect this unit. These units are known as the receptive field of $s_3$.

(Top) When $s$ is formed by convolution with a kernel of width 3, only the inputs affect $s_3$.

(Bottom) When $s$ is formed by matrix multiplication, connectivity is no longer sparse, so all of the inputs affect $s_3$.

Figure 9.4: The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This effect increases if the network includes architectural features like strided convolution (Figure 9.12) or pooling (Section 9.3). This means that even though direct connections in a convolutional net are very sparse, units in the deeper layers can be indirectly connected to all or most of the input image.

337
Growing receptive fields

Figure 9.4: The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This effect increases if the network includes architectural features like strided convolution (figure 9.12) or pooling (section 9.3). This means that even though direct connections in a convolutional net are very sparse, units in the deeper layers can be indirectly connected to all or most of the input image.
Figure 9.5: Parameter sharing: Black arrows indicate the connections that use a particular parameter in two different models.

(Top) The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Due to parameter sharing, this single parameter is used at all input locations.

(Bottom) The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing so the parameter is used only once.

For every location, we learn only one set. This does not affect the runtime of forward propagation—it is still $O(k \times n)$—but it does further reduce the storage requirements of the model to $k$ parameters. Recall that $k$ is usually several orders of magnitude less than $m$. Since $m$ and $n$ are usually roughly the same size, $k$ is practically insignificant compared to $mn$. Convolution is thus dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency. For a graphical depiction of how parameter sharing works, see figure 9.5.

As an example of both of these first two principles in action, figure 9.6 shows how sparse connectivity and parameter sharing can dramatically improve the efficiency of a linear function for detecting edges in an image.

In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation. To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function $f(x)$ is equivariant to a function $g$ if $f(g(x)) = g(f(x))$. In the case of convolution, if we let $g$ be any function that translates the input, i.e., shifts it, then the convolution function is equivariant to $g$. For example, let $I$ be a function giving image brightness at integer coordinates. Let $g$ be a function 338 Convolution shares the same parameters across all spatial locations

Traditional matrix multiplication does not share any parameters

(Goodfellow 2016)
Edge detection by convolution

Figure 9.6: The image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all of the vertically oriented edges in the input image, which can be a useful operation for object detection. Both images are 280 pixels tall. The input image is 320 pixels wide while the output image is 319 pixels wide. This transformation can be described by a convolution kernel containing two elements, and requires $319 \times 280 \times 3 = 267,960$ floating point operations (two multiplications and one addition per output pixel) to compute using convolution. To describe the same transformation with a matrix multiplication would take $320 \times 280 \times 319 \times 280$, or over eight billion, entries in the matrix, making convolution four billion times more efficient for representing this transformation. The straightforward matrix multiplication algorithm performs over sixteen billion floating point operations, making convolution roughly 60,000 times more efficient computationally. Of course, most of the entries of the matrix would be zero. If we stored only the nonzero entries of the matrix, then both matrix multiplication and convolution would require the same number of floating point operations to compute. The matrix would still need to contain $2 \times 319 \times 280 = 178,640$ entries. Convolution is an extremely efficient way of describing transformations that apply the same linear transformation of a small, local region across the entire input. (Photo credit: Paula Goodfellow)
Convolutional Neural Networks

- A ConvNet is made up of Layers
- Every Layer transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters
- Neurons in a layer will only be connected to a small region of the layer before it
Example ConvNet architecture

Layers: **CONV, RELU, POOL, FC**
Convolutional layer
Connectivity

• An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer.

• Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels).

• If the receptive field (or the filter size) is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights (and +1 bias parameter).

• There are multiple neurons (5 in this example) along the depth, all looking at the same region in the input; these are part of different filters.
Spatial arrangement

- Output volume depends on
  - Depth (Number of filters) $K$
  - Spatial extent of filters (receptive field) $F$
  - Stride $S$
  - Amount of zero-padding $P$
Spatial arrangement

- One spatial dimension (x-axis), one neuron with a receptive field size of $F = 3$, the input size is $W = 5$, and there is zero padding of $P = 1$
- Number of output neurons = $(W−F+2P)/S+1$
- Often $P=(F−1)/2$ when $S=1$; ensures number of output neurons = $W$
Spatial arrangement

• Depth
  • Number of filters
  • Each filter learns to look for a pattern in the input (e.g., first CONV layer filters may activate in the presence of differently oriented edges or blobs of color)
Spatial arrangement

• Stride
  • Step size with which we slide the filters
  • 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) then the filters jump 2 pixels at a time as we slide them around
Spatial arrangement

• Zero-padding
  • Pad the input volume with zeros around the border
  • Allows us to control the spatial size of the output volumes
Parameter sharing

• Assumption
  • If one feature is useful to compute at some spatial position \((x,y)\), then it should also be useful to compute at a different position \((x_2,y_2)\)

• All neurons in the same depth slice use the same weights and bias
Convolution Demos

Example ConvNet architecture

Layers: CONV, RELU, POOL, FC
Max pooling

Reduce the amount of parameters and computation in the network, and hence to also control overfitting
Example ConvNet for CIFAR-10

- **INPUT** [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.

- **CONV** layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.

- **RELU** layer will apply an elementwise activation function, such as the max(0,x). This leaves the size of the volume unchanged ([32x32x12]).

- **POOL** layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].

- **FC** (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score.
CNN Visualization

- Visualize activations
- Visualize filters/kernels
- Visualize inputs maximally activating some neuron or layer
Visualize activations

Activations of first convolution layer (left) and 5th convolution layer of AlexNet.

Source: [http://cs231n.github.io/understanding-cnn/](http://cs231n.github.io/understanding-cnn/)
Visualize filters

First CONV layer filters in AlexNet

Source: [http://cs231n.github.io/understanding-cnn/](http://cs231n.github.io/understanding-cnn/)
Visualize inputs maximizing activation

Maximally activating inputs for 6 neurons of 5th pool layer of AlexNet

Source: http://cs231n.github.io/understanding-cnn/
Acknowledgment

Based in part on material from
• Stanford CS231n http://cs231n.github.io/
• Spring 2019 course
• Deep Learning book
Real-world example

• The [Krizhevsky et al.](#) architecture that won the ImageNet challenge in 2012 accepted images of size [227x227x3].

• On the first Convolutional Layer, it used neurons with receptive field size $F=11$, stride $S=4$ and no zero padding $P=0$.

• Since $(227 - 11)/4 + 1 = 55$, and since the Conv layer had a depth of $K=96$, the Conv layer output volume had size [55x55x96].

• Each of the 55*55*96 neurons in this volume was connected to a region of size [11x11x3] in the input volume.

• Moreover, all 96 neurons in each depth column are connected to the same [11x11x3] region of the input, but of course with different weights.
Real-world example

• Number of parameters
  • Without parameter sharing
    • $55 \times 55 \times 96 = 290,400$ neurons in the first Conv Layer, and each has $11 \times 11 \times 3 = 363$ weights and 1 bias. Together, this adds up to $290400 \times 364 = 105,705,600$ parameters on the first layer of the ConvNet
  • With parameter sharing
    • The first Conv Layer in our example would now have only 96 unique set of weights (one for each depth slice), for a total of $96 \times 11 \times 11 \times 3 = 34,848$ unique weights, or 34,944 parameters (+96 biases).