Convolutional Neural Networks

Rohit Amarnath

February 24, 2024

1 Convolution operation

The convolution operation is what distinguishes this class of model and allows the model to learn spatial features, which traditional neural networks do not learn. Consider image I and a kernel K of size $n \times m$. The convolution operation is defined as

$$S(i,j) = (I * K)(i,j) = \sum_{a=0}^{n} \sum_{b=0}^{m} I(i+a, j+b)K(a,b)$$

This operation is actually the *cross-correlation* function, but most modern libraries implement cross-correlation as the convolution operation. We can think of this operation as computing an element-wise product across every $n \times m$ window of image I. With having many kernels (or filters), the model is able to learn different local features, aiding in visual tasks.

2 Properties

There are three key properties of CNNs: sparse connectivity, parameter sharing, and equivariant representations.

2.1 Sparse connectivity

In a traditional neural network, every neuron in the preceding layer influences every neural in the subsequent layer. Consequently, if we have n inputs in the preceding layer and m outputs in the subsequent layer, we end up having $n \times m$ weights, and generally, the transition between layers is done through matrix multiplication. Images are large, and so transitioning between layers in this way is computationally expensive and unfeasible. Hence, CNNs are sparsely connected, meaning that each input neuron only connects to k neurons in the subsequent layer, greatly reducing the number of weights in transitioning between layers. The convolution operation implements this sparse connectivity since only pixels that are close to each other are *processed* together in the subsequent layers.

More often than not, sparse connectivity is linked with the term receptive field. The receptive field at layer k is the area denoted by $R_k \times R_k$ of the input that each neuron in that layer *is linked to.* Given kernel size K_j at layer j and stride S_i at layer i (where $S_0 = 1$), the receptive field at layer k is

$$R_k = 1 + \sum_{j=1}^{k} (K_j - 1) \prod_{i=0}^{j-1} S_i$$

In a way, the receptive field is a measure of the sparsity of the CNN. A larger receptive field would mean that the neurons can detect changes over a wide region in the input image at the cost of less precise perception, and vice versa.

2.2 Parameter sharing

We mentioned earlier that with sparse connectivity, there would be $k \times n$ parameters across the network. Following in theme with the idea of reducing parameters, we maintain the same weights for a particular kernel across the image and so we have k parameters per kernel (that is, a kernel's weights do not change based on their location). Maintaining the same weights for a kernel vastly reduces the number of weights that the model has to learn, allowing the model to be trained in few samples and reduces training time. Additionally, it also makes feature search insensitive to feature location in the image.

2.3 Equivariant representations

Armed with the knowledge of sparse connectivity and parameter sharing, the last property of CNNs is that they are translationally equivariant. A function $f(x)$ is equivariant to function $g(x)$ if $f(g(x)) = g(f(x))$. If we let f be the convolution operation and g be some translation, then equivariance tells us that applying the convolution operation on the translated image is the same result as applying the convolution operation on the original image and then translating it. This means that regardless of location, the kernels will be able to learn specific features to the image. In the first few convolution layers, the kernels detect edges. Since edges appear everywhere in an image, the same parameters can be used to extract the edge in the top left or the bottom right of the image.

There are two common confusions related to equivariance.

- The convolution operation is only translationally equivariant. It is not, for example, scale or rotation equivariant. There is some research into CNNs that have other types of equivariance.
- Translational invariance is different from equivariance. Translational invariance means that the model produces the same response regardless of how the input is translated, i.e. maintaining notation from earlier $f(g(x)) = f(x)$. The convolutional and pooling layers of CNNs make them invariant to small translations. However, there is some debate on this topic in general that CNNs are not perfectly invariant to translation, but they can be trained to be.

3 Pooling

Formally, a convolutional layer is the combination of three operations: convolution, activation, and pooling. In this section, we will explore what the pooling operation does.

A pooling function replaces the output layer with a summary statistic of nearby outputs. The statistic that is commonly used in most CNNs is the max function; however, average, weighted average or a L^2 norm are other choices. Pooling functions are important as they down-sample feature maps from preceding layers, extract low-level (max pooling) or smooth (average pooling) features, and help make the image representation invariant to small translations. The underlying reason why pooling functions work is not well known besides the fact that experiments tend to show that they work well. The max pooling operation is defined similarly to convolution:

$$P(i,j) = \max_{0 \leq a < n, 0 \leq b < m} I(i+a, j+b)$$

In other words, the pooling layer scans every $n \times m$ window of I and reduces it to the maximum within the window.

4 Variations of Convolution

4.1 Stride

Before we talk about stride, a word on padding: padding refers to adding a border of zeros of a particular layer. Generally, after each convolution, there is a marginal decrease in the size of an image, which leads to information loss near the borders. With padding, we are able to preserve the information near the borders, improving model accuracy.

Stride refers to the number of rows and columns traversed per slide. Perhaps, it's wrong to say that strided convolution is a *type* of convolution since the normal convolution operation assumes a stride of 1. If we assume that a particular layer is of dimension $n \times n$, padding size p, kernel size k and stride s, then the the output layer's dimension would be

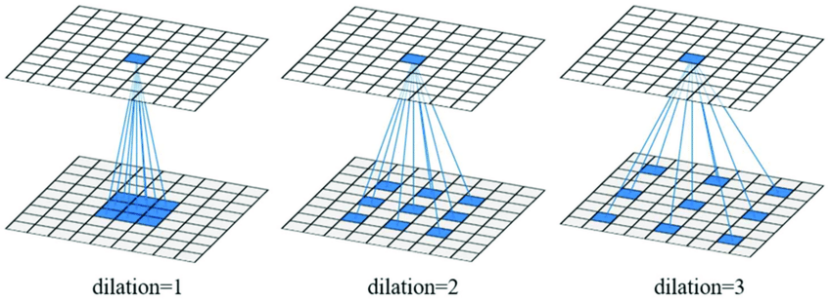
$$\left\lfloor \frac{n + 2p - k}{s} \right\rfloor + 1$$

Notice that the stride reduces the dimensionality of a layer, meaning that it is a viable down-sampling technique. It has been shown that strided convolution performs better than maxpooling, which is largely attributed to the

fact that strided convolution can learn the necessary features and downsample meanwhile pooling is a fixed operation. The only issue with strided convolution is the increase in parameters that it introduces, which is not a problem in modern usecases.

4.2 Dilated

Dilation refers to the number of rows and columns skipped in the convolution operation itself. In normal convolution, there is a dilation of 1. A dilation of d means that $d-1$ cells are skipped while computing the convolution. This is best illustrated with the below image.



Notice that for $d=2$, only one cell is skipped, but when $d=3$, two cells are skipped. Notice that for dilation d and kernel size k, the receptive field one layer up is $((k-1)\cdot d+1)\times((k-1)\cdot d+1)$. Hence, dilated convolutions allow for an exponential expansion of the receptive field while being computationally efficient.

4.3 Tiled

5 Other Convolution Details

Given an input image of $H\times W$, padding (p_H, p_W), kernel size $k_H\times k_W$, stride (s_H, s_W) and dilation (d_H, d_W), the output of the convolution operation has dimensions

$$H' = \left\lfloor \frac{H+2p_H-(k_H-1)\cdot d_H-1}{s_H}+1 \right\rfloor$$
$$W' = \left\lfloor \frac{W+2p_W-(k_W-1)\cdot d_W-1}{s_W}+1 \right\rfloor$$

Generally, padding, kernel size, stride and dilation lengths are equal across the height and width. Additionally, it is common for padding sizes to be odd.

Most of our discussion of the convolution operation has assumed that there is only one channel. This is not always the case: RGB images have three channels. When performing convolution with multiple channels, the kernels become three dimensional (third dimension matching the number of channels), and the convolutions are summed across the channels. Let C_I be the number of input channels. Assuming one output channel, the convolution across multiple channels can be formulated as follows:

$$S = B + \sum_{c=0}^{C_I-1} (I * K_c)$$

where C_I is the number of input channel, B is the bias term and $*$ refers to the cross-correlation operation we specified earlier. If we had several output channels, then we would have to perform the above operation with a different set of kernel weights. In practice, specifying the number of output channels indicates the number of kernels a particular convolutional layer uses.

While perusing the internet, I came across an interesting way to implement the convolution operation using matrix multiplication. It is best explained practically. The following example will implement stride-2 convolution on a 4×4 image I with kernel K of size 2×2. Consider the following matrices:

$$I = \begin{bmatrix} I_{1,1} & I_{1,2} & I_{1,3} & I_{1,4} \\ I_{2,1} & I_{2,2} & I_{2,3} & I_{2,4} \\ I_{3,1} & I_{3,2} & I_{3,3} & I_{3,4} \\ I_{4,1} & I_{4,2} & I_{4,3} & I_{4,4} \end{bmatrix}$$

$$K = \begin{bmatrix} K_{1,1} & K_{1,2} \\ K_{2,1} & K_{2,2} \end{bmatrix}$$

Now, consider the following transformation on I and K.

$$I' = \begin{bmatrix} I_{1,1} & I_{1,2} & I_{2,1} & I_{2,2} \\ I_{1,3} & I_{1,4} & I_{2,3} & I_{2,4} \\ I_{3,1} & I_{3,2} & I_{4,1} & I_{4,2} \\ I_{3,3} & I_{3,4} & I_{4,3} & I_{4,4} \end{bmatrix}$$

$$K' = \begin{bmatrix} K_{1,1} \\ K_{1,2} \\ K_{2,1} \\ K_{2,2} \end{bmatrix}$$

Notice that $I'K'$ is a 4×1 matrix, which can be reshaped into a 2×2 matrix. The result is the stride-2 convolution operation on image I with kernel K. Perhaps not the most elegant way to compute the convolution operation; nevertheless, I thought it was an interesting approach!

6 Backpropagation

6.1 Convolution Layer

Denote $h^{(l)}$ to be the l-th layer of the CNN. The convolution operation between layers l and $l+1$ is defined as

$$z_{i,j}^{(l+1)} = b^{(l)} + \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} w_{x,y}^{(l)} \cdot h_{i+x,j+y}^{(l)}$$

$$h_{i,j}^{(l+1)} = \sigma(z_{i,j}^{(l1)})$$

where σ is the activation function of the l-th layer. We will be focusing on finding the gradients with respect to the l-th layer. Notice that once we know how to compute the gradient with respect to one layer, the remaining layers follow the same procedure. Let the objective function be $\mathcal{L}$. Assume that $\frac{\partial \mathcal{L}}{\partial h_{i,j}^{(l)}}$ is known (it will be clear soon why this is needed). We are to calculate the gradient with respect to $w_{x,y}^{(l)}, b^{(l)}$ and $h_{i,j}^{(l)}$.

6.1.1 $w_{x,y}^{l}$

By the chain rule, we have

$$\frac{\partial \mathcal{L}}{\partial w_{x,y}^{(l)}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial \mathcal{L}}{\partial z_{i,j}^{(l+1)}} \frac{\partial z_{i,j}^{(l+1)}}{w_{x,y}^{(l)}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial \mathcal{L}}{\partial z_{i,j}^{(l+1)}} h_{i+x,j+y}^{(l)}$$

Note that

$$\frac{\partial \mathcal{L}}{\partial z_{i,j}^{(l+1)}} = \frac{\partial \mathcal{L}}{h_{i,j}^{(l+1)}} \frac{\partial h_{i,j}^{(l+1)}}{\partial z_{i,j}^{(l+1)}} = \frac{\partial \mathcal{L}}{h_{i,j}^{(l+1)}} \cdot \sigma'(z_{i,j}^{(l+1)})$$

Thus,

$$\boxed{\frac{\partial \mathcal{L}}{\partial w_{x,y}^{(l)}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial \mathcal{L}}{h_{i,j}^{(l+1)}} \cdot \sigma'(z_{i,j}^{(l+1)}) \cdot h_{i+x,j+y}^{(l)}}$$

6.1.2 $b^{(l)}$

Follow much of what was done above. Note that $\frac{\partial z_{i,j}^{(l+1)}}{\partial b^{(l)}} = 1$. Thus,

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{x,y}^{(l)}} &= \sum_{i=0}^{N-m}\sum_{j=0}^{N-m} \frac{\partial \mathcal{L}}{\partial z_{i,j}^{(l+1)}} \frac{\partial z_{i,j}^{(l+1)}}{b^{(l)}} \\ &= \boxed{\sum_{i=0}^{N-m}\sum_{j=0}^{N-m} \frac{\partial \mathcal{L}}{h_{i,j}^{(l+1)}} \cdot \sigma'(z_{i,j}^{(l+1)})}\end{aligned}$$

6.1.3 $h_{i,j}^{(l)}$

6.2 Maxpooling Layer