Lecture 5 Notes

Deep Learning AI335

1 Convolution [1, 2]

A *convolution* of two functions f and g, denoted $f * g$, is defined as the integral of the product of the two functions after one is reflected about the y-axis and shifted:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)(t - \tau)\, d\tau. \tag{1}$$

The integral is evaluated for all values of shift, producing the convolution function.

Note 1. *The term convolution refers to both the result function and to the process of computing it.*

1.1 Discrete Convolution

For two sequences f and g, the discrete convolution of f and g is given by:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n - m]. \tag{2}$$

We usually refer to f as the *input sequence* and g as the *kernel* or *filter*.

In practice, both f and g are usually finite, which limits the summation to the bounds of the larger sequence (usually the input sequence):

$$(f * g)[n] = \sum_{m=0}^{M-1} f[m]g[n - m], \tag{3}$$

where M is the length of f.

Stride

Stride refers to the number of positions the kernel is shifted after each convolution operation. We can rewrite Equation 2 to include stride:

$$(f * g)[n] = \sum_{m=0}^{M-1} f[m]g[n \cdot s - m], \tag{4}$$

where s is the stride.

A stride of 1 means the kernel moves one position at a time (the standard case), while a stride of 2 means the kernel moves two positions at a time, and so on. Increasing the stride reduces the spatial dimensions of the output.

For an input sequence f of size M, and a kernel g of size N, the size L of the convolution $f * g$ can be calculated as:

$$L = \left\lfloor \frac{M - N}{s} \right\rfloor + 1, \tag{5}$$

where $\lfloor \cdot \rfloor$ denotes the floor function.

Padding

Padding refers to adding extra values (usually zeros) to the borders of the input. This is done to control the output size and to handle boundary effects. There are several types of padding:

1. **Valid padding** - This actually refers to no padding. The output size is smaller than the input size, as calculated by the Equation 5. This is also called *narrow* convolution.
2. **Same padding** - Padding is added to the input sequence such that the size of the output size is equal to the input size.
3. **Full padding** - Padding is added such that the output size is $M + N - 1$. This is also called *wide* convolution.

We can incorporate padding into Equation 5:

$$L = \left\lfloor \frac{M + 2p - N}{s} \right\rfloor + 1, \tag{6}$$

where p is the number of padding elements added to the input sequence.

Discrete Convolution Examples

Consider the following sequences:

$$f = \{3, 5, 6, 7, 8\},$$
$$g = \{2, 1, 2\}.$$

Let's consider different values of stride and padding and see how they affect the convolution.

No padding ($p = 0$) and stride $s = 1$

First we find the size of the output $f * g$ using Equation 6:

$$L = \left\lfloor \frac{5 + 2(0) - 3}{1} \right\rfloor + 1 = 3$$

This gives us the valid range of values to calculate $f * g$ for:

$$\text{range}(3) = \{0, 1, 2\}$$

We can now use Equation 2 to compute the convolutions:

$$(f * g)[n] = \begin{cases} (2 \times 3) + (1 \times 5) + (2 \times 6), & n = 0 \\ (2 \times 5) + (1 \times 6) + (2 \times 7), & n = 1 \\ (2 \times 6) + (1 \times 7) + (2 \times 8), & n = 2 \end{cases}.$$

The convolution result can then be written as:

$$f * g = [23, 30, 35].$$

No padding ($p = 0$) and stride $s = 2$

First, we find the output size L using stride $s = 2$:

$$L = \left\lfloor \frac{5 + 2(0) - 3}{2} \right\rfloor + 1 = 2$$

This gives us the valid range of values to calculate $f * g$ for:

$$\text{range}(2) = \{0, 1\}$$

Computing the convolutions:

$$(f * g)[n] = \begin{cases} (2 \times 3) + (1 \times 5) + (2 \times 6), & n = 0 \\ (2 \times 6) + (1 \times 7) + (2 \times 8), & n = 1 \end{cases}.$$

The convolution result can then be written as:

$$f * g = [23, 35].$$

Same padding and stride $s = 1$

To find the value of padding to use we solve for p in equation 6:

$$L = \left\lfloor \frac{M + 2p - N}{s} \right\rfloor + 1,$$

$$p = \frac{s(L-1) + N - M}{2}.$$

We can then use that equation to find the value of p that achieves same padding by setting $L = M$ (output length = input length):

$$p = \frac{1(5-1) + 3 - 5}{2} = 1.$$

With $p = 1$, the padded input sequence becomes:

$$f = \{0, 3, 5, 6, 7, 8, 0\}.$$

This gives us the valid range of values to calculate $f * g$ for:

$$\text{range}(5) = \{0, 1, 2, 3, 4\}.$$

We can now compute the convolutions:

$$(f * g)[n] = \begin{cases} (2 \times 0) + (1 \times 3) + (2 \times 5), & n = 0 \\ (2 \times 3) + (1 \times 5) + (2 \times 6), & n = 1 \\ (2 \times 5) + (1 \times 6) + (2 \times 7), & n = 2 \\ (2 \times 6) + (1 \times 7) + (2 \times 8), & n = 3 \\ (2 \times 7) + (1 \times 8) + (2 \times 0), & n = 4 \end{cases}.$$

The convolution result can then be written as:

$$f * g = [13, 23, 30, 35, 22].$$

Same padding and stride $s = 2$

First, we find p:

$$p = \frac{2(5-1)+3-5}{2} = 3.$$

With $p = 1$, the padded input sequence becomes:

$$f = \{0, 0, 0, 3, 5, 6, 7, 8, 0, 0, 0\}.$$

Our valid range is the same as the last example:

$$\text{range}(5) = \{0, 1, 2, 3, 4\}.$$

Computing the convolutions:

$$(f * g)[n] = \begin{cases} (2\times 0) + (1\times 0) + (2\times 0), & n = 0 \\ (2\times 0) + (1\times 3) + (2\times 5), & n = 1 \\ (2\times 5) + (1\times 6) + (2\times 7), & n = 2 \\ (2\times 7) + (1\times 8) + (2\times 0), & n = 3 \\ (2\times 0) + (1\times 0) + (2\times 0), & n = 4 \end{cases}.$$

The convolution result can then be written as:

$$f * g = [0, 13, 30, 22, 0].$$

1.2 Discrete Convolution in 2-Dimensions

We can extend the discrete convolution to two dimensions, this operation can be represented using matrices. For an input matrix $\mathbf{F} \in \mathbb{R}^{M\times N}$ and a kernel matrix $\mathbf{G} \in \mathbb{R}^{P\times Q}$ The convolution $\mathbf{F} * \mathbf{G}$ is defined as:

$$(\mathbf{F} * \mathbf{G})[x,\, y] \;=\; \sum_{i=0}^{P-1} \sum_{j=0}^{Q-1} \mathbf{F}[\, x\cdot s + i,\; y\cdot s + j\,]\, \mathbf{G}[\, i,\, j\,], \tag{7}$$

where s is the stride.

The size of the output matrix $\mathbf{F} * \mathbf{G}$ is given by:

$$\big(M_{\text{out}},\, N_{\text{out}}\big) \;=\; \Big(\Big\lfloor \frac{M + 2p - P}{s} \Big\rfloor + 1,\; \Big\lfloor \frac{N + 2p - Q}{s} \Big\rfloor + 1 \Big), \tag{8}$$

where p is the number of padding elements added to the input matrix.

2 Convolutional Networks (ConvNets) [3, 4]

Convolutional (Neural) Networks (CNNs) are are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

The ConvNet architecture is explicitly made to handle images in the form of 3-dimensional data. For a ConvNet the input is a tensor **I** with *shape*: [input_depth, input_width, input_height]. More formally this is:

$$\mathbf{I} \in \mathbb{R}^{D_{\text{in}} \times W_{\text{in}} \times H_{\text{in}}}.$$

The input tensor **I** holds the holds the pixel values for an image in each of it's color channels.

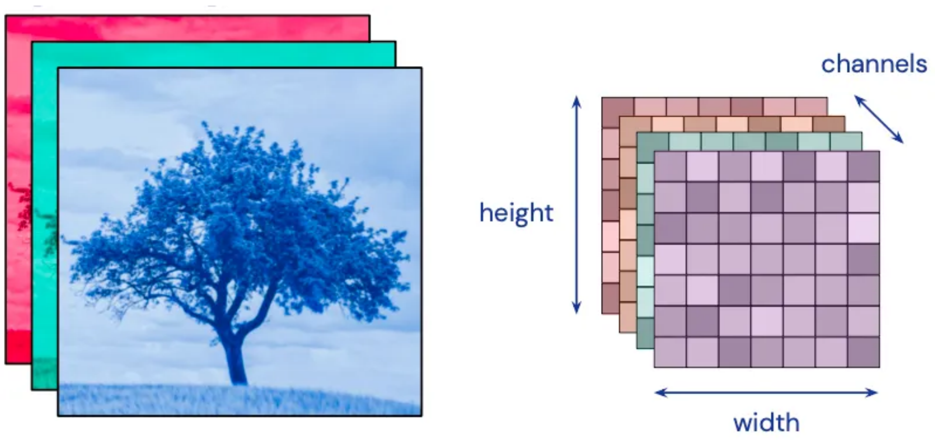

Figure 1: How an image is represented as a tensor.

Note 2. *There is also an extra dimension in the input for the batch, but it is left out for simplicity.*

2.1 Convolution in ConvNets

Given an input tensor **I**, consider performing a convolution with a kernel/filter **K** of shape [kernel_width, kernel_height, input_depth]:

$$\mathbf{K} \in \mathbb{R}^{D_{\text{input}} \times W_{\text{kernel}} \times H_{\text{kernel}}}.$$

We can 'divide' the input tensor into regions r each of the same size as the kernel tensor. A region is sometimes referred to as a *receptive field*. The computation being performed at each region is:

$$(\mathbf{I} * \mathbf{K})[x,\, y] \;=\; \sum_{d=0}^{D_{\text{input}}-1} \; \sum_{w=0}^{W_{\text{kernel}}-1} \; \sum_{h=0}^{H_{\text{kernel}}-1} \mathbf{I}\big[x \cdot s + w,\; y \cdot s + h,\; d\big]\, \mathbf{K}\big[d,\, w,\, h\big] \;+\; b, \tag{9}$$

where b is a bias term, and s is the stride.

Each region r corresponds to a single value x, y in the output $\mathbf{I} * \mathbf{K}$. The output is sometimes referred to as the *activation/feature map*.

Note 2. *The operation described in Equation 9 constitutes a single neuron in a ConvNet.*

Usually we want to perform convolution with a set of kernels rather than a single kernel, with each kernel producing it's own feature map. To achieve this, we add an extra dimension to represent the number of filters in the kernel tensor:

$$\mathbf{K} \in \mathbb{R}^{D_{\text{output}} \times D_{\text{input}} \times W_{\text{kernel}} \times H_{\text{kernel}}}.$$

The operation being performed now becomes:

$$(\mathbf{I} * \mathbf{K})[x,\ y,\ z] \ = \sum_{d_{\text{out}}=0}^{D_{\text{output}}-1} \sum_{d_{\text{in}}=0}^{D_{\text{input}}-1} \sum_{w=0}^{W_{\text{kernel}}-1} \sum_{h=0}^{H_{\text{kernel}}-1} \mathbf{I}\big[x{\cdot}s{+}w,\ y{\cdot}s{+}h,\ d\big]\, \mathbf{K}\big[d_{\text{out}},\ d_{\text{in}},\ w,\ h\big] + b[d_{\text{out}}], \tag{10}$$

Note 3. *The operation being performed in Equation 10 constitutes a single layer in a ConvNet.*

Convolutional Layer in a ConvNet Example

Consider a Conv layer defined as:

- Input tensor with shape [3, 32, 32]
- 6 kernels of shape [3, 5, 5]
- No padding and stride $s = 1$

Lets first consider the output shape for a single kernel. Using Equation 8:

$$\big(W_{\text{out}},\ H_{\text{out}}\big) \ = \ \left(\left\lfloor \frac{W_{\text{in}} + 2p - W_{\text{kernel}}}{s} \right\rfloor + 1,\ \left\lfloor \frac{H_{\text{in}} + 2p - H_{\text{kernel}}}{s} \right\rfloor + 1 \right),$$

$$\big(W_{\text{out}},\ H_{\text{out}}\big) \ = \ \left(\left\lfloor \frac{32 + 2(0) - 5}{1} \right\rfloor + 1,\ \left\lfloor \frac{32 + 2(0) - 5}{1} \right\rfloor + 1 \right) = (28,\ 28).$$

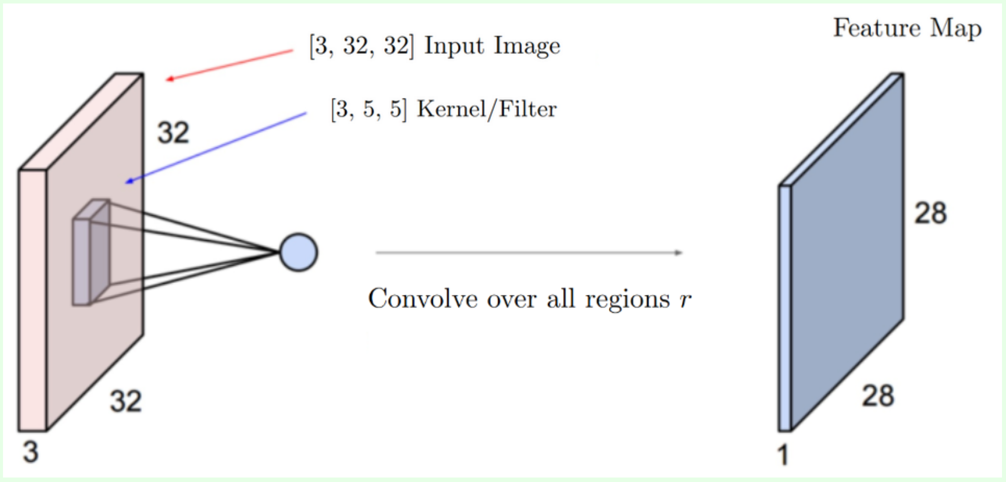

Figure 2

Since we have 6 filters, each producing their own activation map, the output shape will be [6, 28, 28].

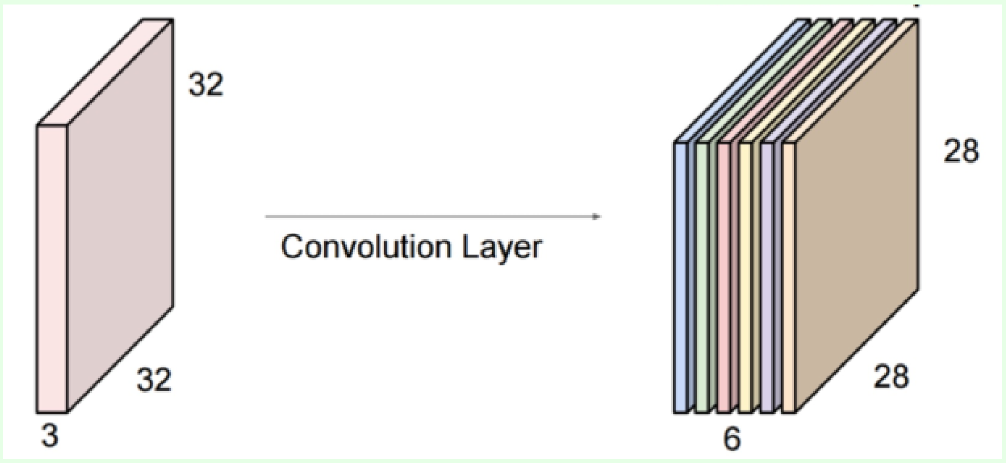

Figure 3

Lets consider how many parameters are in this layer:

Each kernel has parameters:

$$3 \times 5 \times 5 = 75,$$

each kernel also has a bias term b so that makes 76.

Since there are 6 kernels:

$$76 \times 6 = 456$$

We can also find the number of neurons in this layer:

Each pixel in the feature map corresponds to a neuron:

$$28 \times 28 = 784.$$

For 6 feature maps that makes:

$$784 \times 6 = 4704.$$

We can observe the *parameter sharing* characteristic of ConvNets since we have more neurons than parameters.

2.2 ConvNet Architecture

ConvNets are usually structured as a sequence of layers. Every layer of a ConvNet transforms one volume of activations to another through a differentiable function. There are three main types of layers in the architecture: Convolutional Layers, Non-Linear Layers, Pooling Layers, and Fully-Connected Layers.

Non-Linear Layers

Non-linear layers in ConvNets work similarly to regular neural networks. We apply the activation function element wise for every pixel in the feature maps. Non-linear layers have no effect on shape of the feature maps.

Pooling Layers

Pooling layers serve to reduce the dimensions of the feature maps, this is known as *downsampling.* Pooling layers operate independently on every depth slice of the input. Some common pooling techniques are:

- **Max Pooling**:

$$\mathbf{Y}[x,\,y,\,d] \;=\; \max_{\substack{0\le i<P\\0\le j<Q}} \mathbf{X}[\,x\cdot s+i,\;y\cdot s+j,\;d\,], \tag{11}$$

  where P and Q are the spatial dimensions of the pooling kernel, s is the pooling stride, and d indexes over the depth.

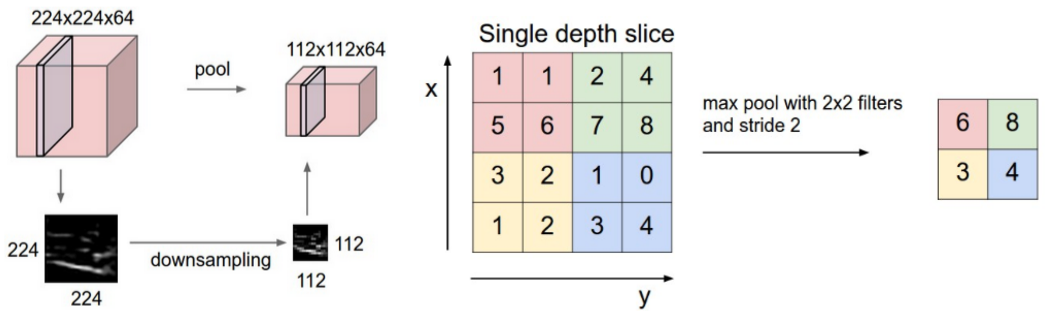

Figure 4: Max pooling example.

- **Average Pooling**:

$$\mathbf{Y}[x,\,y,\,d] \;=\; \frac{1}{P\times Q}\sum_{\substack{0\le i<P\\0\le j<Q}} \mathbf{X}[\,x\cdot s+i,\;y\cdot s+j,\;d\,]. \tag{12}$$

  Instead of taking the maximum value within the pooling region, it computes the mean.

Note 4. *Pooling layers introduce an element of translational invariance, as nearby features in the spatial domain get "compressed" into a single representative value.*

Note 5. *In some deep learning frameworks pooling is also used across depth layers to reduce dimensionality in that direction.*

Max pooling is used much more widely than average pooling, as it works better in practice. Within max pooling we rarely deviate from a 2×2 or 3×3 filter, and stride $s = 2$. Pooling with larger filters may be too destructive to features.

Fully-Connected Layers

A *fully-connected (FC) layer* can be understood by first flattening the input tensor into a vector:

$$\mathbf{x} \in \mathbb{R}^D, \tag{13}$$

where $D = D_{\text{out}} \times W_{\text{out}} \times H_{\text{out}}$ if the tensor has shape $[D_{\text{out}},\, W_{\text{out}},\, H_{\text{out}}]$. The layer then computes:

$$\mathbf{y} = \mathbf{W}\,\mathbf{x} \;+\; \mathbf{b}, \tag{14}$$

where $\mathbf{W} \in \mathbb{R}^{K \times D}$ is the weight matrix (with K being the output dimension, e.g., number of classes in classification), and $\mathbf{b} \in \mathbb{R}^K$ is a bias vector.

Fully connected layers typically have a lot of parameters. E.g., an output of shape [7, 7, 512] mapped to 10 classes would require a fully connected layer with 250,880 parameters.

Fully-Connected Layers as Convolutions

An FC can be viewed as a convolution whose spatial filter size matches the entire input.

For example:

An FC layer with output dimension $K = 4096$ that accepts a $7 \times 7 \times 512$ input can be replaced by a Conv layer with:

- **Kernel size** - [4096, 512, 7, 7]
- **Padding** - $p = 0$
- **Stride** - $s = 0$

This layer would result an output of shape $[1 \times 1 \times 4096]$. In practice, this involves reshaping the FC weights/biases into $[4096 \times 512 \times 7 \times 7]$ filters plus biases.

Why do this? Once an FC layer is converted to a CONV layer, the network can be applied to *larger* inputs in a *single forward pass*, producing outputs (class scores or feature maps) at multiple spatial positions. This *sliding-window* effect is much more efficient than cropping the larger image into multiple patches and running the original FC-based network on each patch separately.

2.3 Layer Patterns

A typical ConvNet stacks only a few layer types: **CONV**, **POOL**, and **FC**, with **RELU** activations inserted between these layers. A concise way to depict this is:

$$\text{INPUT} \;\rightarrow\; [\![\, \text{CONV} \rightarrow \text{RELU}\,]^{N} \rightarrow \text{POOL?}]^{M} \;\rightarrow\; [\text{FC} \rightarrow \text{RELU}]^{K} \;\rightarrow\; \text{FC}.$$

where $N \geq 0$, $M \geq 0$, and $K \geq 0$ are small integers indicating how many times each pattern repeats; POOL? indicates that some blocks may omit pooling.

Filter Sizing

Instead of a single large filter (e.g., 7×7), stacking multiple smaller CONV layers increases expressiveness (due to extra non-linearities) and reduces parameters. For instance, one 7×7 filter has $49C^2$ parameters (if input/output channels each have depth C), while three stacked 3×3 filters only have $27C^2$ total.

Layer Sizing

Common practice uses 3×3 filters with stride 1 and same padding, this preserves the spatial dimensions of the input. POOL layers (often 2×2) then handle downsampling. Larger strides or filters are used sparingly to avoid excessive loss of detail.

3 Transfer Learning [5]

Transfer learning leverages *pretrained networks*—often trained on large datasets such as ImageNet—to avoid building ConvNets from scratch on smaller or domain-specific datasets. This typically involves one of two strategies:

- **Fixed Feature Extractor -** Remove the final classification layer from a pretrained network (e.g., the 1000-way ImageNet FC layer). The rest of the networ essentially acts as a feature extractor. Train a lightweight classifier (e.g., SVM or Softmax) on the extracted features. This method avoids overfitting, especially when the dataset is small.
- **Fine-Tuning the Network -** Initialize training with pretrained weights, train layers (or a subset of the layers) on the new dataset. This method achieves higher accuracy on larger or closely related datasets.

References

[1] Wikipedia Contributors. Convolution, n.d. Accessed: 2024-12-29.

[2] Czech Technical University in Prague. Convolution and convolutional integrals, n.d. Accessed: 2024-12-29.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[4] Stanford University. Cs231n: Convolutional neural networks for visual recognition, 2017. Accessed: 2024-12-29.

[5] Stanford University. Cs231n: Transfer learning, 2017. Accessed: 2024-12-29.