

Welcome back! In this discussion, we'll introduce Convolutional Neural Networks (CNNs) through the lens of **weight sharing**, **output shapes** (padding/stride), and **parameter counting**. We'll also cover **pooling** as a common downsampling operation.

## 1. 1D Convolution: Sliding Window, Padding, and Stride

In this question, we'll build intuition for convolution as a **sliding window** operator in 1D.

Let the input be  $\mathbf{x} \in \mathbb{R}^9$  and a kernel (filter) be  $\mathbf{k} \in \mathbb{R}^3$ :

$$\mathbf{k} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_9 \end{bmatrix}.$$

**Conceptual note (Deep Learning convention).** Most deep learning libraries implement “convolution” as **cross-correlation** (no kernel flip). So a length-3 kernel applied at position  $i$  typically produces

$$y_i = k_1 x_i + k_2 x_{i+1} + k_3 x_{i+2}.$$

(a) **No padding, stride 1.**

- (i) What is the output dimension?
- (ii) What is the **first** element of the output?
- (iii) What is the **last** element of the output?

(b) **Padding size 1, stride 2.**

Let padding size 1 mean we pad one zero on the left and one zero on the right:

$$\tilde{\mathbf{x}} = [0, x_1, x_2, \dots, x_9, 0]^\top.$$

- (i) What is the output dimension?
- (ii) What is the **first** element of the output?
- (iii) What is the **second** element of the output?

## 2. Weight Sharing: Convolution as a Linear Layer

Convolution is a **linear operator**. That means we can write it as a matrix multiplication:

$$\mathbf{y} = \mathbf{K}\mathbf{x}$$

(assume no bias).

In this problem, we use the same setup as the previous question part (no padding, stride 1, input length 9, kernel length 3), so the output length is 7.

**Conceptual note (Weight sharing).** Convolution reuses the *same* kernel weights at every spatial location. This is why CNNs can have far fewer parameters than fully-connected layers while still detecting repeated patterns.

- (a) Write down  $\mathbf{K}$  explicitly (as a matrix) such that  $\mathbf{y} = \mathbf{K}\mathbf{x}$  matches the stride-1, no-padding convolution with kernel  $\mathbf{k} = [k_1, k_2, k_3]^\top$ .
- (b) Now suppose we **remove weight sharing**: we still use a sliding window of size 3 at each location, but each location gets its **own** kernel (its own parameters).
- (i) How does the matrix  $\mathbf{K}$  change structurally?
  - (ii) How many weights do we have now (ignoring bias)?

### 3. 2D Convolution: Filtering, Output Shapes, and Parameter Counting

Now we move to 2D convolutions and connect the math to practical tensor shapes and parameter counts.

**Conceptual note (Why CNNs scale).** A convolution layer learns a small bank of filters and applies them everywhere. This gives (1) translation-equivariant feature detection and (2) massive parameter savings versus a fully-connected layer on images.

(a) **2D example (no padding, stride 1).** Compute the output and describe the effect of this filter:

$$\mathbf{k} = \frac{1}{4} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

(b) **Output dimension formula.**

Suppose the input is  $W \times W$ , the kernel is  $K \times K$ .

(i) With stride 1 and no padding, what is  $W'$ ?

(ii) With stride  $s$  and padding  $p$ , what is  $W'$ ?

(c) **Manual multi-channel 2D convolution (stride 1, padding 1).**

Given an input  $\mathbf{X}$  with shape  $[C_{\text{in}}, H, W] = [2, 4, 4]$ , define

$$\mathbf{Z} = \text{conv2d}(\mathbf{X}, \text{kernel\_size} = 3, \text{padding} = 1, \text{stride} = 1, \text{num\_filters} = 8).$$

For this question, focus only on **output channel 0** (i.e., the first filter), and assume **no bias**. Use the standard deep-learning convention (cross-correlation: *no kernel flip*).

The filter for output channel 0 spans both input channels and is given by:

$$W[0, 0, :, :] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad W[0, 1, :, :] = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

The input is:

$$X[0, :, :] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad X[1, :, :] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 4 & 6 & 8 \\ 3 & 4 & 0 & 1 \end{bmatrix}.$$

- (i) What is the shape of *one* conv2d filter (for a single output channel)?  
(Your answer should have 3 dimensions.)
- (ii) What is the shape of  $\mathbf{Z}$ ?  
(Your answer should have 3 dimensions.)
- (iii) Compute  $Z[0, 0, 0]$  and  $Z[0, 0, 1]$ .  
(For sanity, you only need these two output values.)

(d) **Realistic tensor shapes (CNN vs MLP parameter count).**

Input: a  $256 \times 256$  RGB image. We have 32 filters, each with kernel size 5. Use the convention that images are shaped as [channels,height,width].

- (i) What is the input tensor shape?
- (ii) What is the shape of **each** kernel filter? (Both answers to i and ii should have 3 dimensions.)
- (iii) Apply convolution with no padding and stride 2. What is the output tensor shape?
- (iv) How many weights does the CNN layer have (ignore bias)?
- (v) If we used a fully-connected linear layer from the **flattened input** to the **flattened CNN output**, how many weights would that layer have?
- (vi) Suppose we added a bias term to this Conv2d layer. What is the shape of this bias parameter?

## 4. Pooling: MaxPool as Downsampling (and Backprop Intuition)

Pooling layers are commonly used to **downsample** spatial dimensions. The most common version is **max pooling**: take the maximum value in each  $K \times K$  window, then move the window by stride  $S$ .

**Conceptual note.** Pooling reduces spatial resolution (smaller height/width), which can reduce compute/memory in later layers. Max pooling also introduces a small amount of invariance to small translations: if a feature shifts slightly within a pooling window, the max can stay the same.

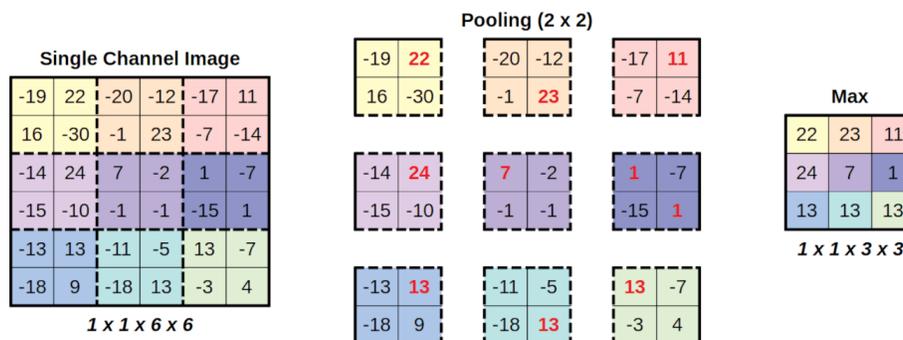
### (a) Compute a maxpool output (shape + values).

Consider a single-channel  $6 \times 6$  input

$$X = \begin{bmatrix} -19 & 22 & -20 & -12 & -17 & 11 \\ 16 & -30 & -1 & 23 & -7 & -14 \\ -14 & 24 & 7 & -2 & 1 & -7 \\ -15 & -10 & -1 & -1 & -15 & 1 \\ -13 & 13 & -11 & -5 & 13 & -7 \\ -18 & 9 & -18 & 13 & -3 & 4 \end{bmatrix}$$

Apply  $2 \times 2$  **max pooling** with stride 2 (no padding).

- i. What is the output shape?
- ii. Compute the output values.



**Figure 1:** Example of  $2 \times 2$  max pooling with stride 2 (single channel).

### (b) General pooling output size formula.

Suppose the input spatial size is  $W \times W$ , pooling window is  $K \times K$ , stride is  $S$ , and padding is  $p$ . What is the output spatial dimension  $W'$ ?

(c) **Backprop intuition through pooling (conceptual).**

Let  $Y = \text{maxpool}(X)$  be max pooling applied independently per window. Given upstream gradient  $\frac{\partial L}{\partial Y}$ , describe  $\frac{\partial L}{\partial X}$  in words or a formula.

Then answer the same for average pooling.

**5. Backprop Through CNN Building Blocks** This optional question is meant as a bridge between CNN operations and reverse-mode automatic differentiation. If a homework asks you to implement backward passes for CNN layers, these patterns show up directly.

**Conceptual note.** In reverse-mode AD, you are given an **upstream gradient** (e.g.,  $\frac{\partial L}{\partial Y}$ ) and compute **downstream gradients** (e.g.,  $\frac{\partial L}{\partial X}$  and parameter gradients). For “routing” operations like max pooling, gradients flow only along the path that determined the output.

(a) **MaxPool backward on one window (numeric).**

Consider a single  $2 \times 2$  window

$$X_{\text{win}} = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}, \quad y = \text{maxpool}(X_{\text{win}}).$$

Suppose the upstream gradient is  $\frac{\partial L}{\partial y} = 7$ .

Compute  $\frac{\partial L}{\partial X_{\text{win}}}$  for: (i) max pooling, and (ii) average pooling.

(b) **1D convolution (cross-correlation) backward (symbolic).**

Let  $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]^\top$  and kernel  $\mathbf{k} = [k_1, k_2, k_3]^\top$ . Define the stride-1, no-padding 1D “convolution” (cross-correlation) output  $\mathbf{y} \in \mathbb{R}^3$ :

$$y_1 = k_1 x_1 + k_2 x_2 + k_3 x_3,$$

$$y_2 = k_1 x_2 + k_2 x_3 + k_3 x_4,$$

$$y_3 = k_1 x_3 + k_2 x_4 + k_3 x_5.$$

Suppose you are given the upstream gradient

$$\frac{\partial L}{\partial \mathbf{y}} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}.$$

Compute:

$$\frac{\partial L}{\partial \mathbf{k}} \quad \text{and} \quad \frac{\partial L}{\partial \mathbf{x}}.$$

**Contributors:**

- Terry Kim.
- Eric Kim.