Data 188 Introduction to Deep Learning
Spring 2026 Eric Kim

Discussion 05

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 $\mathbf{k} \in \mathbb{R}^3$:

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

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?

Solution: With no padding and stride 1, the kernel of length 3 can start at positions 1 through 7. So the output length is $9 - 3 + 1 = 7$.

First element (kernel aligned to x_1, x_2, x_3):

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

Last element (kernel aligned to x_7, x_8, x_9):

$$y_7 = k_1 x_7 + k_2 x_8 + k_3 x_9.$$

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

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

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

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

Solution: After padding, the effective length is $9 + 2 = 11$. With kernel size 3 and stride 2, the output length is

$$\left\lfloor \frac{11-3}{2} \right\rfloor + 1 = \left\lfloor \frac{8}{2} \right\rfloor + 1 = 4 + 1 = 5.$$

First element uses indices $(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3) = (0, x_1, x_2)$:

$$y_1 = k_1 \cdot 0 + k_2 x_1 + k_3 x_2.$$

Second element shifts by stride 2, using $(\tilde{x}_3, \tilde{x}_4, \tilde{x}_5) = (x_2, x_3, x_4)$:

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

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 $\mathbf{k} = [k_1, k_2, k_3]^\top$.

Solution: Since $y_i = k_1 x_i + k_2 x_{i+1} + k_3 x_{i+2}$ for $i = 1, \ldots, 7$, $\mathbf{K} \in \mathbb{R}^{7\times 9}$ is the Toeplitz (banded) matrix:

$$\mathbf{K} = \begin{bmatrix} k_1 & k_2 & k_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & k_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_1 & k_2 & k_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & k_1 & k_2 & k_3 \end{bmatrix}.$$

(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)?

Solution: (i) The matrix stays **sparse and banded**, but the nonzero entries are no longer the same across rows. Instead of repeating (k_1, k_2, k_3) on every row, each row i has its own triple $(k_1^{(i)}, k_2^{(i)}, k_3^{(i)})$.

Concretely, the linear operator becomes

$$\mathbf{y} = \mathbf{K}_{\text{no-share}}\mathbf{x}, \qquad \mathbf{K}_{\text{no-share}} \in \mathbb{R}^{7\times 9},$$

where

$$\mathbf{K}_{\text{no-share}} = \begin{bmatrix} k_1^{(1)} & k_2^{(1)} & k_3^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & k_1^{(2)} & k_2^{(2)} & k_3^{(2)} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & k_1^{(3)} & k_2^{(3)} & k_3^{(3)} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_1^{(4)} & k_2^{(4)} & k_3^{(4)} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_1^{(5)} & k_2^{(5)} & k_3^{(5)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & k_1^{(6)} & k_2^{(6)} & k_3^{(6)} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & k_1^{(7)} & k_2^{(7)} & k_3^{(7)} \end{bmatrix}.$$

(ii) There are 7 output positions (rows), and each uses 3 parameters, so total weights:

$$7 \times 3 = 21.$$

With weight sharing, we only had 3 parameters.

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:

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

Solution: Output size is $(3-2+1)\times(3-2+1) = 2\times 2$. Each output entry is the average of a 2×2 block:

$$\mathbf{y} = \begin{bmatrix} \frac{1+2+4+5}{4} & \frac{2+3+5+6}{4} \\ \frac{4+5+7+8}{4} & \frac{5+6+8+9}{4} \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 6 & 7 \end{bmatrix}.$$

Effect: a **box blur / local averaging** filter (it smooths the image).

(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'?

Solution: (i) Stride 1, no padding:

$$W' = W - K + 1.$$

(ii) With padding p and stride s:

$$W' = \left\lfloor \frac{W + 2p - K}{s} \right\rfloor + 1.$$

(This assumes $W + 2p \geq K$ so the kernel fits at least once.)

(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

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

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:

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

The input is:

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

(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.)

Solution:

(i) One filter must span all input channels, so its shape is

$$(C_{\text{in}}, K, K) = (2, 3, 3).$$

(Across all 8 filters, the full weight bank is $(8, 2, 3, 3)$.)

(ii) With $H = W = 4$, $K = 3$, stride 1, padding 1:

$$H' = W' = \left\lfloor \frac{4 + 2 \cdot 1 - 3}{1} \right\rfloor + 1 = 4,$$

so

$$\mathbf{Z} \text{ has shape } (8, 4, 4).$$

(iii) Padding adds a 1-pixel border of zeros around each input channel.
Entry $Z[0,0,0]$**:** the top-left 3×3 padded patches are

$$X[0]_{\text{pad}}[0:3, 0:3] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}, \quad X[1]_{\text{pad}}[0:3, 0:3] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

Channel 0 contribution:

$$\langle W[0,0,:,:],\ X[0]_{\text{pad}}[0:3, 0:3] \rangle = (1 \cdot 0) + (2 \cdot 1) + (1 \cdot 2) = 0 + 2 + 2 = 4.$$

Channel 1 contribution (all-ones filter $\Rightarrow$ sum of patch):

$$\langle W[0,1,:,:],\ X[1]_{\text{pad}}[0:3, 0:3] \rangle = 2.$$

Thus

$$Z[0,0,0] = 4 + 2 = 6.$$

Entry $Z[0,0,1]$**:** shift one step right (still on the top row):

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

Channel 0 contribution:

$$\langle W[0,0,:,:],\ X[0]_{\text{pad}}[0:3,1:4]\rangle = (1\cdot 1)+(2\cdot 2)+(1\cdot 3) = 1+4+3 = 8.$$

Channel 1 contribution (sum of patch):

$$\langle W[0,1,:,:],\ X[1]_{\text{pad}}[0:3,1:4]\rangle = 3.$$

Thus

$$Z[0,0,1] = 8+3 = 11.$$

(d) **Realistic tensor shapes (CNN vs MLP parameter count).**
Input: a 256×256 RGB image. We have 32 filters, each with kernel size 5. Use the convention that images are shaped as $[\texttt{channels}, \texttt{height}, \texttt{width}]$.
(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?

Solution: (i) Input tensor shape:

$$(3,\ 256,\ 256).$$

(ii) Each kernel must span all input channels:

$$(3,\ 5,\ 5).$$

(Across all 32 filters, the full weight bank would be $(32, 3, 5, 5)$ in PyTorch.)
(iii) Output spatial size:

$$W' = \left\lfloor \frac{256-5}{2} \right\rfloor + 1 = \left\lfloor \frac{251}{2} \right\rfloor + 1 = 125+1 = 126.$$

Output tensor shape (channels = number of filters):

$$(32,\ 126,\ 126).$$

(iv) CNN weights (ignore bias):

$$32 \times (3\cdot 5\cdot 5) = 32 \times 75 = 2400.$$

(v) Fully-connected layer weights: input dim $= 3\cdot 256\cdot 256 = 196{,}608$, output dim $= 32\cdot 126\cdot 126 = 508{,}032$. So the weight matrix has

$$196{,}608 \times 508{,}032 = 99{,}883{,}155{,}456$$

weights (about 9.99×10^{10}), which is enormous compared to 2400.
(vi) One bias per output channel:

$$[\texttt{chans\_out}] = [32]$$

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 6×6 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 2×2 **max pooling** with stride 2 (no padding).

i. What is the output shape?

ii. Compute the output values.

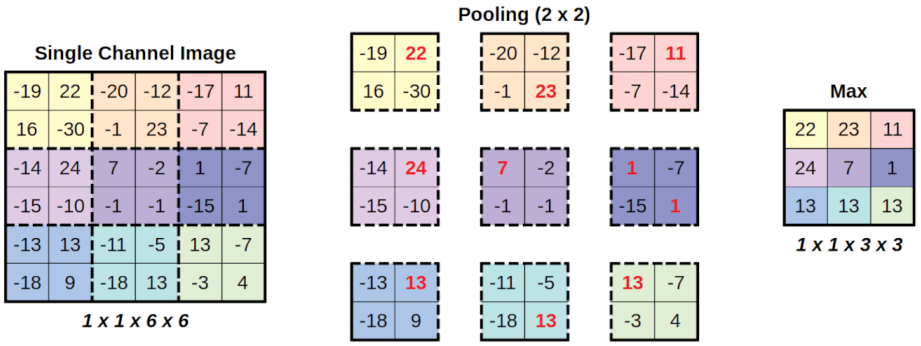

Figure 1: Example of 2×2 max pooling with stride 2 (single channel).

Solution: Output shape:

$$6 \times 6 \xrightarrow{K=2,\, S=2} 3 \times 3.$$

Break into non-overlapping 2×2 windows:

Top row of windows:

$$\max \begin{bmatrix} -19 & 22 \\ 16 & -30 \end{bmatrix} = 22, \quad \max \begin{bmatrix} -20 & -12 \\ -1 & 23 \end{bmatrix} = 23, \quad \max \begin{bmatrix} -17 & 11 \\ -7 & -14 \end{bmatrix} = 11.$$

Middle row:

$$\max\begin{bmatrix}-14 & 24\\ -15 & -10\end{bmatrix}=24,\quad \max\begin{bmatrix}7 & -2\\ -1 & -1\end{bmatrix}=7,\quad \max\begin{bmatrix}1 & -7\\ -15 & 1\end{bmatrix}=1.$$

Bottom row:

$$\max\begin{bmatrix}-13 & 13\\ -18 & 9\end{bmatrix}=13,\quad \max\begin{bmatrix}-11 & -5\\ -18 & 13\end{bmatrix}=13,\quad \max\begin{bmatrix}13 & -7\\ -3 & 4\end{bmatrix}=13.$$

So

$$Y=\begin{bmatrix}22 & 23 & 11\\ 24 & 7 & 1\\ 13 & 13 & 13\end{bmatrix}.$$

(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'?

Solution: Pooling uses the same output-size formula as convolution (just no learnable weights):

$$W' = \left\lfloor \frac{W + 2p - K}{S} \right\rfloor + 1.$$

(For no padding, $p = 0$.)

(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.

Solution: **Max pooling:** In each pooling window, the output is the max of the window. During backprop, the upstream gradient for that output location is routed *only* to the input entry that achieved the max (the argmax). All other entries in the window get zero gradient.

Formally, for each output position (i, j) with window $\mathcal{W}_{ij}$ and argmax location $(u^*, v^*) \in \mathcal{W}_{ij}$:

$$\frac{\partial L}{\partial X_{uv}} = \begin{cases} \frac{\partial L}{\partial Y_{ij}}, & (u,v) = (u^*, v^*) \\ 0, & \text{otherwise (within that window).} \end{cases}$$

(Assuming unique max; ties require a tie-breaking rule.)

Average pooling: The output is the mean of the window, so the upstream gradient is evenly distributed across the K^2 entries:

$$\frac{\partial L}{\partial X_{uv}} = \frac{1}{K^2}\frac{\partial L}{\partial Y_{ij}} \quad \text{for all } (u,v) \in \mathcal{W}_{ij}.$$

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 2×2 window

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

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.

Solution: **Max pooling:** $y = \max(X_{\text{win}}) = 3$ comes from the bottom-right entry, so all gradient goes there:

$$\frac{\partial L}{\partial X_{\text{win}}} = \begin{bmatrix} 0 & 0 \\ 0 & 7 \end{bmatrix}.$$

Average pooling: $y = \frac{1}{4}\sum X_{ij}$, so each entry gets $7/4$:

$$\frac{\partial L}{\partial X_{\text{win}}} = \begin{bmatrix} \frac{7}{4} & \frac{7}{4} \\ \frac{7}{4} & \frac{7}{4} \end{bmatrix}.$$

(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 $\mathbf{y} \in \mathbb{R}^3$:

$$\begin{aligned} y_1 &= k_1 x_1 + k_2 x_2 + k_3 x_3, \\ y_2 &= k_1 x_2 + k_2 x_3 + k_3 x_4, \\ y_3 &= k_1 x_3 + k_2 x_4 + k_3 x_5. \end{aligned}$$

Suppose you are given the upstream gradient

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

Compute:

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

Solution: Use the chain rule. Each parameter k_j affects multiple outputs due to weight sharing.

Gradient w.r.t. kernel:

$$\begin{aligned}
\frac{\partial L}{\partial k_1} &= \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial k_1} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial k_1} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial k_1} \\
&= a \cdot x_1 + b \cdot x_2 + c \cdot x_3, \\
\frac{\partial L}{\partial k_2} &= \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial k_2} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial k_2} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial k_2} \\
&= a \cdot x_2 + b \cdot x_3 + c \cdot x_4, \\
\frac{\partial L}{\partial k_3} &= \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial k_3} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial k_3} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial k_3} \\
&= a \cdot x_3 + b \cdot x_4 + c \cdot x_5.
\end{aligned}$$

Gradient w.r.t. input: each x_i contributes to the outputs whose receptive fields include it:

$$\begin{aligned}
\frac{\partial L}{\partial x_1} &= \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x_1} = a \cdot k_1, \\
\frac{\partial L}{\partial x_2} &= \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x_2} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x_2} = a \cdot k_2 + b \cdot k_1, \\
\frac{\partial L}{\partial x_3} &= \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x_3} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x_3} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial x_3} = a \cdot k_3 + b \cdot k_2 + c \cdot k_1, \\
\frac{\partial L}{\partial x_4} &= \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x_4} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial x_4} = b \cdot k_3 + c \cdot k_2, \\
\frac{\partial L}{\partial x_5} &= \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial x_5} = c \cdot k_3.
\end{aligned}$$

(Notice how the middle input x_3 receives contributions from all three upstream gradients because it appears in all three convolution windows.)

Contributors:

- Terry Kim.
- Eric Kim.

