

# A gentle explanation of Backpropagation in Convolutional Neural Network

Son Nguyen

February 27, 2020

Recently, I have read some articles about Convolutional Neural Network, for example, [1], [2], and the notes of the Stanford CS class CS231n [3]. These articles explain Convolutional Neural Network's architecture and its layers very well but they didn't include a detailed explanation of Backpropagation in Convolutional Neural Network. After digging the Internet more deeper and wider, I found two articles [4] and [5] explaining the Backpropagation phase pretty deeply but I feel they are still abstract to me. Because I want a more tangible and detailed explanation so I decided to write this article myself. I hope that it is helpful to you.

# 1. Prerequisites

To fully understand this article, I highly recommend you to read the following articles to grasp firmly the foundation of Convolutional Neural Network beforehand:

- <http://cs231n.github.io/convolutional-networks/>
- <https://victorzhou.com/blog/intro-to-cnns-part-1/>

## 2. Architecture

In this article, I will build a real Convolutional Neural Network from scratch to classify hand-written digits in the MNIST database provided by <http://yann.lecun.com/exdb/mnist/>. At an abstract level, the architecture looks like:

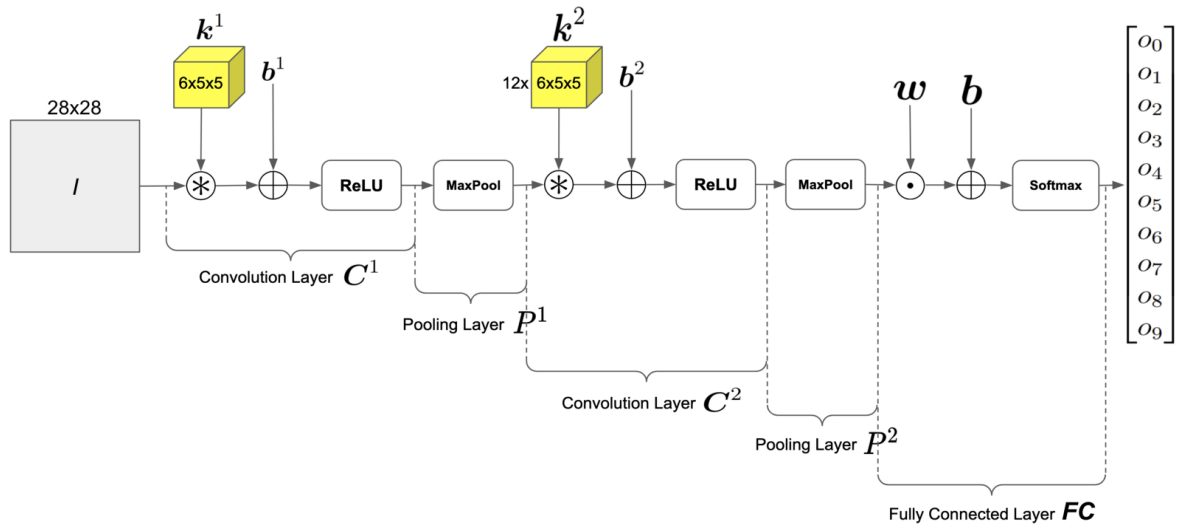


Figure 1: Abstract Architecture

where

- $I$  is a grayscale image with size  $28 \times 28$
- the kernel  $k^1$  is a 3D array with size  $6 \times 5 \times 5$
- the bias  $b^1$  is a 1D array with size 6
- the kernel  $k^2$  is a 4D array with size  $12 \times 6 \times 5 \times 5$
- the bias  $b^2$  is a 1D array with size 12
- the weight  $w$  is a 2D array with size  $10 \times 192$
- the bias  $b$  is a 1D array with size 10
- the output  $O$  is a 1D array with size 10

In the first and second Convolution Layers, I use **ReLU** functions (Rectified Linear Unit) as activation functions. I use **MaxPool** with pool size  $2 \times 2$  in the first and second Pooling Layers. And, I use **Softmax** as an activation function in the Fully Connected Layer.

Zooming in the abstract architecture, we will have a detailed architecture split into two following parts (I split the detailed architecture into 2 parts because it's too long to fit on a single page):

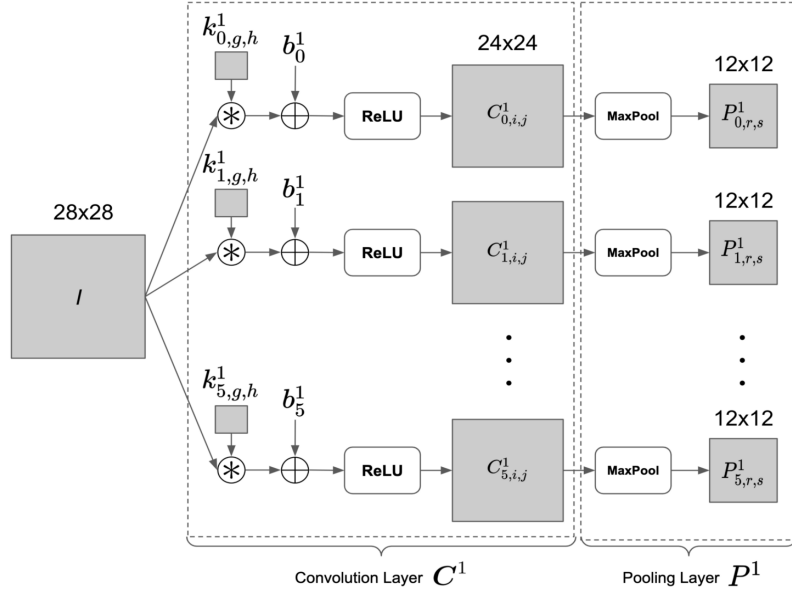


Figure 2: Detailed Architecture - part 1

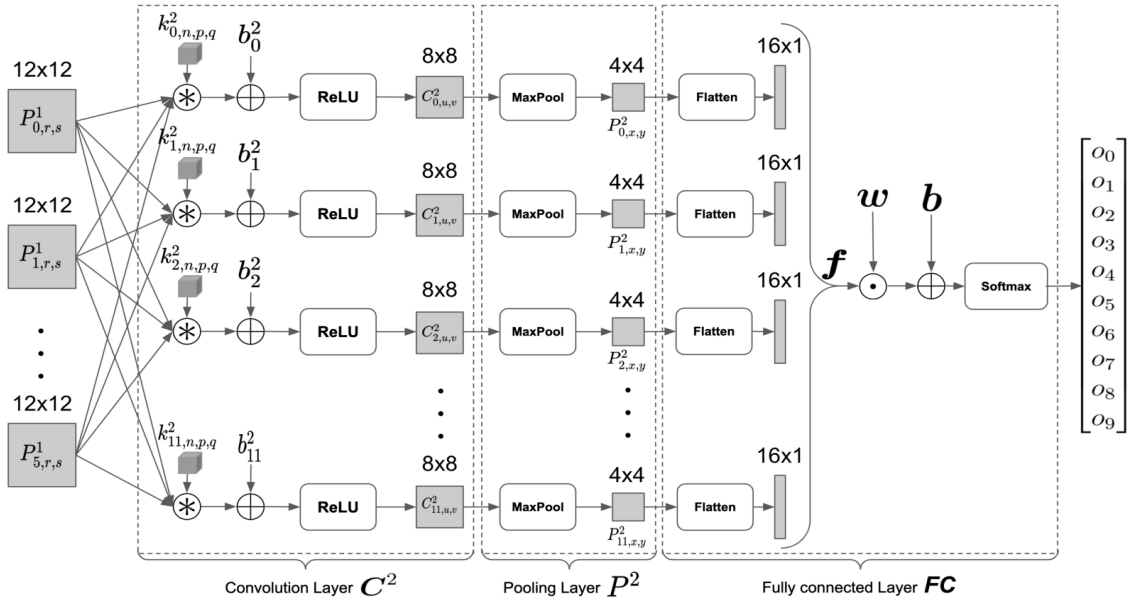


Figure 3: Detailed Architecture - part 2

Like a standard Neural Network, training a Convolutional Neural Network consists of two phases **Feedforward** and **Backpropagation**.

### 3. Feedforward

By definition, the formula of **Feedforward** of each layer is represented as follows.

#### 3.1. Convolution Layer $C^1$

$$\begin{aligned} S_{nij}^1 &= \sum_{g=0}^4 \sum_{h=0}^4 I_{g+i, h+j} k_{ngh}^1 + b_n^1, \\ C_{nij}^1 &= \text{ReLU}(S_{nij}^1) = \begin{cases} S_{nij}^1 & \text{if } S_{nij}^1 > 0 \\ 0 & \text{if } S_{nij}^1 \leq 0 \end{cases} \\ n &= 0, \dots, 5; \quad i, j = 0, \dots, 23 \end{aligned} \quad (1)$$

Where 6 kernels  $k_{0gh}^1, \dots, k_{5gh}^1$  are 2D arrays with size  $5 \times 5$  and we assume that they were rotated  $180^\circ$  beforehand, and **stride** is set to 1.  $\mathbf{S}^1$  is the result of convolution between the grayscale image  $\mathbf{I}$  and 6 kernels  $k_{0gh}^1, \dots, k_{5gh}^1$ . Both  $\mathbf{S}^1$  and  $\mathbf{C}^1$  have the same size of  $6 \times 24 \times 24$ .

#### 3.2. Pooling Layer $P^1$

$$\begin{aligned} P_{nrs}^1 &= \max(C_{nij}^1), \quad i = 2r, 2r+1, \quad j = 2s, 2s+1, \\ n &= 0, \dots, 5; \quad r, s = 0, \dots, 11 \end{aligned} \quad (2)$$

Because we use pool size of  $2 \times 2$ , as a result  $\mathbf{P}^1$  has a size of  $6 \times 12 \times 12$ .

#### 3.3. Convolution Layer $C^2$

$$\begin{aligned} S_{muv}^2 &= \sum_{n=0}^5 \sum_{p=0}^4 \sum_{q=0}^4 P_{n,p+u, q+v}^1 k_{mnpq}^2 + b_m^2, \\ C_{muv}^2 &= \text{ReLU}(S_{muv}^2) = \begin{cases} S_{muv}^2 & \text{if } S_{muv}^2 > 0 \\ 0 & \text{if } S_{muv}^2 \leq 0 \end{cases} \\ m &= 0, \dots, 11; \quad u, v = 0, \dots, 7 \end{aligned} \quad (3)$$

Where 12 kernels  $k_{0npq}^2, \dots, k_{11npq}^2$  are 3D arrays with size  $6 \times 5 \times 5$  and we assume that they were rotated  $180^\circ$  beforehand, and **stride** is set to 1.  $\mathbf{S}^2$  is the result of convolution between  $\mathbf{P}^1$  and 12 kernels  $k_{0npq}^2, \dots, k_{11npq}^2$ . Both  $\mathbf{S}^2$  and  $\mathbf{C}^2$  have the same size of  $12 \times 8 \times 8$ .

#### 3.4. Pooling Layer $P^2$

$$\begin{aligned} P_{mxy}^2 &= \max(C_{muv}^2), \quad u = 2x, 2x+1, \quad v = 2y, 2y+1, \\ m &= 0, \dots, 11; \quad x, y = 0, \dots, 3 \end{aligned} \quad (4)$$

### 3.5. Fully Connected Layer $FC$

$$\begin{aligned}
\mathbf{f} &= \text{flatten}(\mathbf{P}^2) \\
S_i &= \sum_{j=0}^{191} w_{ij} f_j + b_i, \\
O_i &= \text{softmax}(S_i) = \frac{e^{S_i}}{\sum_{k=0}^9 e^{S_k}}, \\
i &= 0, \dots, 9
\end{aligned} \tag{5}$$

## 4. Backpropagation

Like a standard Neural Network, in this Backpropagation phase, we also need to find optimal values of parameters so that the loss function  $L$  is minimum. In Convolutional Neural Network, parameters are just kernels and biases ( $\mathbf{k}^1$ ,  $\mathbf{b}^1$ ,  $\mathbf{k}^2$ , and  $\mathbf{b}^2$ ). Besides, the weight  $\mathbf{w}$  and bias  $\mathbf{b}$  are parameters too.

Because in the Fully Connected Layer, we use **Softmax** as an activation function so the most suitable loss function should be a cross entropy ([https://en.wikipedia.org/wiki/Cross\\_entropy](https://en.wikipedia.org/wiki/Cross_entropy)).

$$\begin{aligned}
L &= L(O_0, \dots, O_9), \\
&= L(O_{label}) \\
&= -\ln(O_{label}), \text{ label} \in \{0, \dots, 9\}
\end{aligned} \tag{6}$$

where  $label$  is the ground-truth value of the grayscale image  $\mathbf{I}$ ,  $\ln(O_{label})$  is the natural logarithm of  $O_{label}$ .

To find optimal values of parameters, we will also apply the Stochastic Gradient Descent algorithm. First, we need to derive gradients of parameters in the Fully Connected Layer  $FC$ , the Convolution Layer  $C^1$ , and the Convolution Layer  $C^2$ .

### 4.1. Deriving gradients of parameters in the Fully Connected Layer

In the  $FC$  layer, we need to derive gradients  $\frac{\partial L}{\partial b_i}$  and  $\frac{\partial L}{\partial w_{ij}}$ . From (6), obviously we can say  $L$  is a function of the variable  $O_{label}$ . In addition, from (5) we can also say  $O_{label}$  is a function of multi-variables  $S_i$  ( $i = 0, \dots, 9$ ) and for a specific  $S_i$ , we can say  $S_i$  is a function of a specific  $b_i$  and multi-variables  $w_{ij}$  and  $f_j$  ( $j = 0, \dots, 191$ ). Therefore, we can apply the chain rule to deriving gradients  $\frac{\partial L}{\partial b_i}$  and  $\frac{\partial L}{\partial w_{ij}}$  as follows.

#### 4.1.1. Deriving the gradient $\frac{\partial L}{\partial b_i}$

Applying the chain rule, we can represent

$$\frac{\partial L}{\partial b_i} = \sum_{k=0}^9 \frac{\partial L}{\partial S_k} \frac{\partial S_k}{\partial b_i}, \quad i = 0, \dots, 9 \tag{7}$$

From (5), we can see obviously that

$$\frac{\partial S_k}{\partial b_i} = \begin{cases} 1 & \text{if } k = i \\ 0 & \text{if } k \neq i \end{cases} \tag{8}$$

So we can reduce (7) as follows

$$\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial S_i}, \quad i = 0, \dots, 9 \quad (9)$$

Let's continue developing  $\frac{\partial L}{\partial S_i}$  with the support of the chain rule.

$$\frac{\partial L}{\partial S_i} = \frac{\partial L(O_{label})}{\partial O_{label}} \frac{\partial O_{label}}{\partial S_i} \quad (10)$$

$$\frac{\partial L(O_{label})}{\partial O_{label}} = \frac{\partial(-\ln(O_{label}))}{\partial O_{label}} = -\frac{1}{O_{label}} \quad (11)$$

From (5) we have

$$O_{label} = \frac{e^{S_{label}}}{\sum_{k=0}^9 e^{S_k}} \quad (12)$$

Denote  $T = \sum_{k=0}^9 e^{S_k}$ , we can rewrite  $O_{label}$  as follows

$$O_{label} = \frac{e^{S_{label}}}{T} = e^{S_{label}} T^{-1} \quad (13)$$

So we can develop

$$\frac{\partial O_{label}}{\partial S_i} = \frac{\partial(e^{S_{label}} T^{-1})}{\partial S_i}, \quad i = 0, \dots, 9 \quad (14)$$

Next, let's consider two cases:

**Case 1:**  $i = label$ , then we can rewrite (14) as follows

$$\frac{\partial O_{label}}{\partial S_{label}} = \frac{\partial(e^{S_{label}} T^{-1})}{\partial S_{label}} \quad (15)$$

Denote  $e^{S_{label}} = T - C_1$ , where  $C_1 = e^{S_0} + \dots + e^{S_{label-1}} + e^{S_{label+1}} + \dots + e^{S_9}$  and we can regard  $C_1$  as a constant because  $C_1$  is independent of  $S_{label}$ . Then,

$$\begin{aligned} \frac{\partial O_{label}}{\partial S_{label}} &= \frac{\partial((T - C_1)T^{-1})}{\partial S_{label}} \\ &= \frac{\partial(1 - C_1 T^{-1})}{\partial S_{label}} \\ &= -C_1 \frac{\partial T^{-1}}{\partial S_{label}} \\ &= -C_1 \frac{\partial T^{-1}}{\partial T} \frac{\partial T}{\partial S_{label}} \\ &= -C_1 (-1 T^{-2}) \frac{\partial T}{\partial S_{label}} \\ &= C_1 T^{-2} \frac{\partial T}{\partial S_{label}} \end{aligned} \quad (16)$$

$$\frac{\partial T}{\partial S_{label}} = \frac{\partial(e^{S_{label}} + C_1)}{\partial S_{label}} = \frac{\partial e^{S_{label}}}{\partial S_{label}} = e^{S_{label}} \quad (17)$$

Substitute (17) into (16),

$$\begin{aligned}
\frac{\partial O_{label}}{\partial S_{label}} &= C_1 T^{-2} e^{S_{label}} \\
&= (C_1 T^{-1})(e^{S_{label}} T^{-1}) \\
&= (T - e^{S_{label}}) T^{-1} (e^{S_{label}} T^{-1}) \\
&= (1 - e^{S_{label}} T^{-1})(e^{S_{label}} T^{-1}) \\
&= (1 - O_{label}) O_{label}
\end{aligned} \tag{18}$$

**Case 2:**  $i \neq label$ , then we can regard  $e^{S_{label}}$  as a constant and therefore,

$$\begin{aligned}
\frac{\partial O_{label}}{\partial S_i} &= e^{S_{label}} \frac{\partial(T^{-1})}{\partial S_i} \\
&= e^{S_{label}} \frac{\partial(T^{-1})}{\partial T} \frac{\partial T}{\partial S_i} \\
&= -e^{S_{label}} T^{-2} \frac{\partial T}{\partial S_i}
\end{aligned} \tag{19}$$

From (5), we can represent  $T = C_2 + e^{S_i}$ , where  $C_2 = e^{S_0} + \dots + e^{S_{i-1}} + e^{S_{i+1}} + \dots + e^{S_9}$  and we can regard  $C_2$  as a constant because  $C_2$  is independent of  $S_i$ . Then,

$$\frac{\partial T}{\partial S_i} = \frac{\partial(C_2 + e^{S_i})}{\partial S_i} = \frac{\partial e^{S_i}}{\partial S_i} = e^{S_i} \tag{20}$$

Substitute (20) into (19),

$$\frac{\partial O_{label}}{\partial S_i} = -e^{S_{label}} T^{-2} e^{S_i} = -(e^{S_{label}} T^{-1})(e^{S_i} T^{-1}) = -O_{label} O_i \tag{21}$$

Combine **Case 1** with **Case 2**, we have

$$\frac{\partial O_{label}}{\partial S_i} = \begin{cases} (1 - O_{label}) O_{label} & \text{if } i = label \\ -O_{label} O_i & \text{if } i \neq label \end{cases}, \quad i = 0, \dots, 9 \tag{22}$$

Substitute (22) and (11) into (10),

$$\frac{\partial L}{\partial S_i} = \begin{cases} O_{label} - 1 & \text{if } i = label \\ O_i & \text{if } i \neq label \end{cases}, \quad i = 0, \dots, 9 \tag{23}$$

Denote  $\mathbf{dLS}$  is an array of  $\frac{\partial L}{\partial S_i}$ , we can represent

$$\mathbf{dLS} = \begin{bmatrix} \frac{\partial L}{\partial S_0} \\ \vdots \\ \frac{\partial L}{\partial S_9} \end{bmatrix} \tag{24}$$

Substitute (23) into (9), finally we obtain

$$\frac{\partial L}{\partial b_i} = \begin{cases} O_{label} - 1 & \text{if } i = label \\ O_i & \text{if } i \neq label \end{cases}, \quad i = 0, \dots, 9 \tag{25}$$

Denote  $\mathbf{dLb}$  is an array of  $\frac{\partial L}{\partial b_i}$ , we can represent

$$\mathbf{dLb} = \begin{bmatrix} O_0 \\ \cdot \\ \cdot \\ \cdot \\ O_{label} - 1 \\ \cdot \\ \cdot \\ \cdot \\ O_9 \end{bmatrix} \quad (26)$$

#### 4.1.2. Deriving the gradient $\frac{\partial L}{\partial w_{ij}}$

Applying the chain rule, we can represent

$$\frac{\partial L}{\partial w_{ij}} = \sum_{k=0}^9 \frac{\partial L}{\partial S_k} \frac{\partial S_k}{\partial w_{ij}}, \quad i = 0, \dots, 9, \quad j = 0, \dots, 191 \quad (27)$$

From (5), we can derive

$$\frac{\partial S_k}{\partial w_{ij}} = \frac{\partial(\sum_{t=0}^{191} w_{kt} f_t + b_k)}{\partial w_{ij}} = \begin{cases} f_j & \text{if } k = i \\ 0 & \text{if } k \neq i \end{cases} \quad (28)$$

So,

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial S_i} f_j, \quad i = 0, \dots, 9, \quad j = 0, \dots, 191 \quad (29)$$

Because we already calculated  $\frac{\partial L}{\partial S_i}$  in (23), so we can calculate  $\frac{\partial L}{\partial w_{ij}}$  easily. Now denote  $\mathbf{dLw}$  is a 2D array of  $\frac{\partial L}{\partial w_{ij}}$ , we can represent

$$\mathbf{dLw} = \begin{bmatrix} \frac{\partial L}{\partial S_0} f_0 & \dots & \frac{\partial L}{\partial S_0} f_{191} \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ \frac{\partial L}{\partial S_9} f_0 & \dots & \frac{\partial L}{\partial S_9} f_{191} \end{bmatrix} \quad (30)$$

## 4.2. Deriving gradients of parameters in the Convolution Layer $C^2$

For a specific  $b_m^2$  (or  $k_{mnpq}^2$ ), from the equations in the **Feedforward** section, we can see obviously that every  $S_{muv}^2$  is a function of 301 variables ( $P_{n,p+u,q+v}^1, k_{mnpq}^2, b_m^2$ ) ( $n = 0, \dots, 5, p = 0, \dots, 4, q = 0, \dots, 4$ ), and every  $C_{muv}^2$  is a function of one variable  $S_{muv}^2$ , and every  $P_{mxy}^2$  is a function of 4 variables  $C_{muv}^2$  ( $u = 2x, 2x + 1, v = 2y, 2y + 1$ ), and every  $f_k$  ( $k = 16m + 4x + y$ ) is equal to one  $P_{mxy}^2$ . If the variable  $b_m^2$  (or  $k_{mnpq}^2$ ) changes a bit, it will affect all  $S_{muv}^2$  ( $u = 0, \dots, 7, v = 0, \dots, 7$ ) and then like the domino effect, all  $C_{muv}^2$  and all  $P_{mxy}^2$  will be affected. Next, only 16  $f_k$  elements ( $k = 16m, 16m + 15$ ) will be affected. Next, all  $S_i$  ( $i = 0, \dots, 9$ ) will be effected. Finally, the change will cause  $L$  changes accordingly. Based on these nested relationships and applying the chain rule, we can derive  $\frac{\partial L}{\partial b_m^2}$  and  $\frac{\partial L}{\partial k_{mnpq}^2}$  as follows.



#### 4.2.1. Deriving the gradient $\frac{\partial L}{\partial b_m^2}$ ( $m = 0, \dots, 11$ )

$$\frac{\partial L}{\partial b_m^2} = \sum_{u=0}^7 \sum_{v=0}^7 \frac{\partial L}{\partial S_{muv}^2} \frac{\partial S_{muv}^2}{\partial b_m^2} \quad (31)$$

$$\frac{\partial L}{\partial S_{muv}^2} = \frac{\partial L}{\partial C_{muv}^2} \frac{\partial C_{muv}^2}{\partial S_{muv}^2} \quad (32)$$

Let's calculate  $\frac{\partial L}{\partial C_{muv}^2}$ .

Because of how **MaxPool** (pool size =  $2 \times 2$ ) works (please review <https://victorzhou.com/blog/intro-to-cnns-part-2/#4-backprop-max-pooling>), one specific  $C_{muv}^2$  always belongs to a certain block of 4 elements (the  $C_{muv}^2$  and 3 other elements of the 3D array  $\mathbf{C}^2$ ) and it may or may not be the maximum element out of 4 elements. Considering two following cases:

**Case 1:**  $C_{muv}^2$  is the maximum element out of 4 elements. Denote  $u = u_{max}$ ,  $v = v_{max}$ . Then,  $C_{muv}^2 = C_{mu_{max}v_{max}}^2$  and  $P_{mxy}^2 = C_{mu_{max}v_{max}}^2$  ( $x = \text{floor}(u_{max}/2)$ ,  $y = \text{floor}(v_{max}/2)$ ).

$$\frac{\partial L}{\partial C_{muv}^2} = \frac{\partial L}{\partial C_{mu_{max}v_{max}}^2} = \frac{\partial L}{\partial P_{mxy}^2} \quad (33)$$

Furthermore,  $f_k = P_{mxy}^2$  ( $k = 16m + 4x + y$ ). It follows that

$$\frac{\partial L}{\partial C_{mu_{max}v_{max}}^2} = \frac{\partial L}{\partial P_{mxy}^2} = \frac{\partial L}{\partial f_k} \quad (34)$$

Let's calculate  $\frac{\partial L}{\partial f_k}$ :

$$\frac{\partial L}{\partial f_k} = \sum_{i=0}^9 \frac{\partial L}{\partial S_i} \frac{\partial S_i}{\partial f_k} \quad (35)$$

From (5), we can derive

$$\frac{\partial S_i}{\partial f_k} = w_{ik} \quad (36)$$

Substitute (36) into (35),

$$\frac{\partial L}{\partial f_k} = \sum_{i=0}^9 \frac{\partial L}{\partial S_i} w_{ik} \quad (37)$$

Denote  $\mathbf{dLf}$  is a 1D array of  $\frac{\partial L}{\partial f_k}$  with shape = (192), then we can represent

$$\mathbf{dLf} = \mathbf{w}^T \cdot \mathbf{dLS} \quad (38)$$

Denote  $\mathbf{dLP}^2$  is a 3D array of  $\frac{\partial L}{\partial \mathbf{P}^2}$  with shape =  $\mathbf{P}^2.\text{shape} = (12, 4, 4)$ , then we can represent

$$\mathbf{dLP}^2 = \mathbf{dLf}.\text{reshape}(\mathbf{P}^2.\text{shape}) \quad (39)$$

**Case 2:**  $C_{muv}^2$  is not the maximum element out of 4 elements. Then, if  $C_{muv}^2$  changes a super small quantity, that change will not affect  $P_{mxy}^2$  ( $x = \text{floor}(u/2)$ ,  $y = \text{floor}(v/2)$ ) at all. For example, given the following block of 4  $C_{muv}^2$  elements (as shown in Figure 4)

Suppose that  $u = 3, v = 2$ , then  $C_{muv}^2 = 0.31$  (the yellow cell) and obviously we have  $u_{max} = 2, v_{max} = 3$ , and  $C_{mu_{max}v_{max}}^2 = 0.32$ . So  $x = \text{floor}(u/2) = \text{floor}(3/2) = 1, y = \text{floor}(v/2) = \text{floor}(2/2) = 1$ , and thus  $P_{mxy}^2 = P_{m11}^2 = C_{mu_{max}v_{max}}^2 = 0.32$ . If we increase  $C_{muv}^2$  by  $10^{-9}$ , that

			$v$							
			0	1	2	3	4	5	6	7
0										
1										
2										
$u$ 3										
4										
5										
6										
7										

Figure 4: A sample  $\mathbf{C}^2[m]$

is  $C_{muv}^2 = C_{muv}^2 + 10^{-9} = 0.31 + 10^{-9}$ . The new  $C_{muv}^2$  still doesn't affect the  $P_{mxy}^2$  at all. As a result, the change doesn't affect the loss function  $L$ . From this observation, we can state that in this case

$$\frac{\partial L}{\partial C_{muv}^2} = 0 \quad (40)$$

Combining **Case 1** with **Case 2**, we can express

$$\frac{\partial L}{\partial C_{muv}^2} = \begin{cases} \frac{\partial L}{\partial P_{mxy}^2}, & (x = \text{floor}(u/2), y = \text{floor}(v/2)) \text{ if } C_{muv}^2 \text{ is the maximum element} \\ & \text{out of 4 elements.} \\ 0 & \text{otherwise} \end{cases} \quad (41)$$

Denote  $\mathbf{dLC2}$  is a 3D array of  $\frac{\partial L}{\partial C_{muv}^2}$  elements ( $\mathbf{dLC2}[m, u, v] = \frac{\partial L}{\partial C_{muv}^2}$ ), with shape =  $\mathbf{C}^2.\text{shape}$ . We can form a procedure to calculate  $\mathbf{dLC2}[m, u, v]$  as follows:

- In the **Feedforward** phase, we cache  $(m, u_{max}, v_{max})$  into a 3D array  $\mathbf{I2}$  (with shape =  $\mathbf{P}^2.\text{shape}$ ) like this:

$$\mathbf{I2}[m, x, y] = \begin{bmatrix} u_{max} \\ v_{max} \end{bmatrix}, \quad x = u_{max}/2, \quad y = v_{max}/2 \quad (42)$$

- Calculate  $\mathbf{dLP2}$  using equations (38) and (39).
- Initialize  $\mathbf{dLC2} = 0$  (all  $\mathbf{dLC2}$ 's elements are zero).
- Loop through all triple  $(m, x, y)$ ,  $m = 0, \dots, 11$ ;  $x, y = 0, \dots, 3$ :

$$\begin{aligned} & \text{Retrieve } u_{max}, v_{max} \text{ from } \mathbf{I2}[m, x, y], \\ & \mathbf{dLC2}[m, u_{max}, v_{max}] = \mathbf{dLP2}[m, x, y] \end{aligned} \quad (43)$$

So every  $\mathbf{dLC2}[m, u, v] = \frac{\partial L}{\partial C_{muv}^2}$  is calculated.

Next, denote **dC2S2** is a 3D array of  $\frac{\partial C_{muv}^2}{\partial S_{muv}^2}$  elements ( $\mathbf{dC2S2}[m, u, v] = \frac{\partial C_{muv}^2}{\partial S_{muv}^2}$ ), then just in the **Feedforward** phase we can also calculate **dC2S2** $[m, u, v]$  using (3) as follows:

$$\mathbf{dC2S2}[m, u, v] = \frac{\partial C_{muv}^2}{\partial S_{muv}^2} = \begin{cases} 1 & \text{if } S_{muv}^2 > 0 \\ 0 & \text{if } S_{muv}^2 \leq 0 \end{cases} \quad (44)$$

So we can rewrite (32) as follows:

$$\mathbf{dLS2}[m, u, v] = \frac{\partial L}{\partial S_{muv}^2} = \mathbf{dLC2}[m, u, v] \mathbf{dC2S2}[m, u, v] \quad (45)$$

where, **dLS2** is a 3D array of  $\frac{\partial L}{\partial S_{muv}^2}$  elements.

From (3), we can calculate

$$\frac{\partial S_{muv}^2}{\partial b_m^2} = 1 \quad (46)$$

Finally, substitute (46) and (45) into (31), we obtain

$$\frac{\partial L}{\partial b_m^2} = \sum_{u=0}^7 \sum_{v=0}^7 \mathbf{dLS2}[m, u, v] \quad (47)$$

#### 4.2.2. Deriving the gradient $\frac{\partial L}{\partial k_{mnpq}^2}$

$$\frac{\partial L}{\partial k_{mnpq}^2} = \sum_{u=0}^7 \sum_{v=0}^7 \frac{\partial L}{\partial S_{muv}^2} \frac{\partial S_{muv}^2}{\partial k_{mnpq}^2} \quad (48)$$

From (3), we can calculate

$$\frac{\partial S_{muv}^2}{\partial k_{mnpq}^2} = P_{n,p+u,q+v}^1 \quad (49)$$

Substitute (49) into (48), we obtain

$$\frac{\partial L}{\partial k_{mnpq}^2} = \sum_{u=0}^7 \sum_{v=0}^7 \frac{\partial L}{\partial S_{muv}^2} P_{n,p+u,q+v}^1 \quad (50)$$

### 4.3. Deriving gradients of parameters in the Convolution Layer $C^1$

#### 4.3.1. Deriving the gradient $\frac{\partial L}{\partial b_n^1}$ ( $n = 0, \dots, 5$ )

$$\frac{\partial L}{\partial b_n^1} = \sum_{i=0}^{23} \sum_{j=0}^{23} \frac{\partial L}{\partial S_{nij}^1} \frac{\partial S_{nij}^1}{\partial b_n^1} \quad (51)$$

$$\frac{\partial L}{\partial S_{nij}^1} = \frac{\partial L}{\partial C_{nij}^1} \frac{\partial C_{nij}^1}{\partial S_{nij}^1} \quad (52)$$

$$\frac{\partial L}{\partial C_{nij}^1} = \begin{cases} \frac{\partial L}{\partial P_{nrs}^1}, & (r = \text{floor}(i/2), s = \text{floor}(j/2)) \text{ if } C_{nij}^1 \text{ is the maximum element} \\ & \text{out of 4 elements.} \\ 0 & \text{otherwise} \end{cases} \quad (53)$$

Let's calculate  $\frac{\partial L}{\partial P_{nrs}^1}$ .

From (3) and the architecture diagrams, we can see that a specific  $P_{nrs}^1$  may affect all  $S_{mu\nu}^2$ . Therefore,

$$\frac{\partial L}{\partial P_{nrs}^1} = \sum_{m=0}^{11} \sum_{u=0}^7 \sum_{v=0}^7 \frac{\partial L}{\partial S_{mu\nu}^2} \frac{\partial S_{mu\nu}^2}{\partial P_{nrs}^1} \quad (54)$$

We only need to calculate  $\frac{\partial S_{mu\nu}^2}{\partial P_{nrs}^1}$  because  $\frac{\partial L}{\partial S_{mu\nu}^2}$  was already calculated in (45).

From (3), we see that for a specific triple  $(m, u, v)$ , there is a 3D region of  $P_{nrs}^1$  elements ( $n = 0, \dots, 5$ ,  $r = u, \dots, u + 4$ ,  $s = v, \dots, v + 4$ ) contributing to  $S_{mu\nu}^2$ , and  $\frac{\partial S_{mu\nu}^2}{\partial P_{nrs}^1} = k_{mnpq}^2$ ,  $p = r - u$ ,  $q = s - v$ . If an element  $P_{nrs}^1$  is outside that 3D region, it doesn't contribute to  $S_{mu\nu}^2$ , and therefore  $\frac{\partial S_{mu\nu}^2}{\partial P_{nrs}^1} = 0$ . So denote **dS2P1** is a 6D array of  $\frac{\partial S_{mu\nu}^2}{\partial P_{nrs}^1}$  elements ( $\mathbf{dS2P1}[n, r, s, m, u, v] = \frac{\partial S_{mu\nu}^2}{\partial P_{nrs}^1}$ ) with  $\mathbf{dS2P1.shape} = \mathbf{P}^1.shape + \mathbf{C}^2.shape = (6, 12, 12, 12, 8, 8)$ , we can calculate  $\mathbf{dS2P1}[n, r, s, m, u, v]$  just in the **Feedforward** phase using the following procedure:

- Initialize  $\mathbf{dS2P1} = 0$
- Loop through all triple  $(m, u, v)$ ,  $m = 0, \dots, 11$ ,  $u = 0, \dots, 7$ ,  $v = 0, \dots, 7$ :

$$\mathbf{dS2P1}[0 : 6, u : (u + 5), v : (v + 5), m, u, v] = \mathbf{k}^2[m] \quad (55)$$

Note that in (55), **dS2P1** and  $\mathbf{k}^2$  are Numpy arrays and  $\mathbf{k}^2[m]$  is a 3D array of  $k_{mnpq}^2$ ,  $n = 0, \dots, 5$ ;  $p, q = 0, \dots, 4$ .

Substitute (45) and  $\mathbf{dS2P1}[n, r, s, m, u, v]$  calculated above into (54), we obtain

$$\frac{\partial L}{\partial P_{nrs}^1} = \sum_{m=0}^{11} \sum_{u=0}^7 \sum_{v=0}^7 \mathbf{dLS2}[m, u, v] \mathbf{dS2P1}[n, r, s, m, u, v] \quad (56)$$

Denote **dLP1** is a 3D array of  $\frac{\partial L}{\partial P_{nrs}^1}$  elements ( $\mathbf{dLP1}[n, r, s] = \frac{\partial L}{\partial P_{nrs}^1}$ ) and **dLC1** is a 3D array of  $\frac{\partial L}{\partial C_{nij}^1}$  elements ( $\mathbf{dLC1}[n, i, j] = \frac{\partial L}{\partial C_{nij}^1}$ ). And, if just in the **Feedforward** phase, we cache triple  $(n, i_{max}, j_{max})$  at which  $C_{ni_{max}j_{max}}^1$  is the maximum element out of 4 elements into a 3D array **I1**

$$\mathbf{I1}[n, r, s] = \begin{bmatrix} i_{max} \\ j_{max} \end{bmatrix}, \quad r = i_{max}/2, \quad s = j_{max}/2 \quad (57)$$

Then we can calculate  $\mathbf{dLC1}[n, i, j]$  as follows:

- Initialize  $\mathbf{dLC1} = 0$ .
- Loop through all triple  $(n, r, s)$ ,  $n = 0, \dots, 5$ ;  $r, s = 0, \dots, 11$ :

$$\begin{aligned} &\text{Retrieve } i_{max}, j_{max} \text{ from } \mathbf{I1}[n, r, s], \\ &\mathbf{dLC1}[n, i_{max}, j_{max}] = \mathbf{dLP1}[n, r, s] \end{aligned} \quad (58)$$

Next, denote **dC1S1** is a 3D array of  $\frac{\partial C_{nij}^1}{\partial S_{nij}^1}$  elements ( $\mathbf{dC1S1}[n, i, j] = \frac{\partial C_{nij}^1}{\partial S_{nij}^1}$ ), then just in the **Feedforward** phase we can also calculate  $\mathbf{dC1S1}[n, i, j]$  using (1) as follows:

$$\mathbf{dC1S1}[n, i, j] = \frac{\partial C_{nij}^1}{\partial S_{nij}^1} = \begin{cases} 1 & \text{if } S_{nij}^1 > 0 \\ 0 & \text{if } S_{nij}^2 \leq 0 \end{cases} \quad (59)$$

Substitute the calculated  $\mathbf{dLC1}[n, i, j] = \frac{\partial L}{\partial C_{nij}^1}$  and  $\mathbf{dC1S1}[n, i, j] = \frac{\partial C_{nij}^1}{\partial S_{nij}^1}$  into (52), we obtain

$$\frac{\partial L}{\partial S_{nij}^1} = \mathbf{dLC1}[n, i, j] \mathbf{dC1S1}[n, i, j] \quad (60)$$

Next, from (1), we can see obviously that

$$\frac{\partial S_{nij}^1}{\partial b_n^1} = 1 \quad (61)$$

Finally, substitute (61) and (60) into (51), we obtain

$$\frac{\partial L}{\partial b_n^1} = \sum_{i=0}^{23} \sum_{j=0}^{23} \mathbf{dLC1}[n, i, j] \mathbf{dC1S1}[n, i, j] \quad (62)$$

**4.3.2. Deriving the gradient**  $\frac{\partial L}{\partial k_{ngh}^1}$  ( $n = 0, \dots, 5; g, h = 0, \dots, 4$ )

$$\frac{\partial L}{\partial k_{ngh}^1} = \sum_{i=0}^{23} \sum_{j=0}^{23} \frac{\partial L}{\partial S_{nij}^1} \frac{\partial S_{nij}^1}{\partial k_{ngh}^1} \quad (63)$$

$\frac{\partial L}{\partial S_{nij}^1}$  was calculated in (60). So the rest of work is to calculate  $\frac{\partial S_{nij}^1}{\partial k_{ngh}^1}$  and we can calculate it easily using (1):

$$\frac{\partial S_{nij}^1}{\partial k_{ngh}^1} = I_{n,g+i,h+j} \quad (64)$$

Substitute (64) into (63), we obtain

$$\frac{\partial L}{\partial k_{ngh}^1} = \sum_{i=0}^{23} \sum_{j=0}^{23} \frac{\partial L}{\partial S_{nij}^1} I_{n,g+i,h+j} \quad (65)$$

## 4.4. Update parameters

$$\begin{aligned} k_{ngh}^1 &= k_{ngh}^1 - \eta \frac{\partial L}{\partial k_{ngh}^1}, \quad n = 0, \dots, 5; \quad g, h = 0, \dots, 4, \\ b_n^1 &= b_n^1 - \eta \frac{\partial L}{\partial b_n^1}, \quad n = 0, \dots, 5, \\ k_{mnpq}^2 &= k_{mnpq}^2 - \eta \frac{\partial L}{\partial k_{mnpq}^2}, \quad m = 0, \dots, 11, \quad n = 0, \dots, 5; \quad p, q = 0, \dots, 4, \\ b_m^2 &= b_m^2 - \eta \frac{\partial L}{\partial b_m^2}, \quad m = 0, \dots, 11, \\ w_{ij} &= w_{ij} - \eta \frac{\partial L}{\partial w_{ij}}, \quad i = 0, \dots, 9, \quad j = 0, \dots, 191, \\ b_i &= b_i - \eta \frac{\partial L}{\partial b_i}, \quad i = 0, \dots, 9. \end{aligned} \quad (66)$$

Where  $\eta$  is the learning rate.

## References

- [1] <https://victorzhou.com/blog/intro-to-cnns-part-1/>
- [2] Alejandro Escontrela. *Convolutional Neural Networks from the ground up*.
- [3] <http://cs231n.github.io/convolutional-networks/>
- [4] <http://cbelwal.blogspot.com/2018/05/part-i-backpropagation-mechanics-for.html>
- [5] Zhifei Zhang. *Derivation of Backpropagation in Convolutional Neural Network (CNN)*. University of Tennessee, Knoxville, TN, October 18, 2016.  
<https://pdfs.semanticscholar.org/5d79/11c93ddcb34cac088d99bd0cae9124e5dcd1.pdf>