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 handwritten digits in the MNIST database provided by `http://yann.lecun.com/exdb/mnist/`. At an abstract level, the architecture looks like:

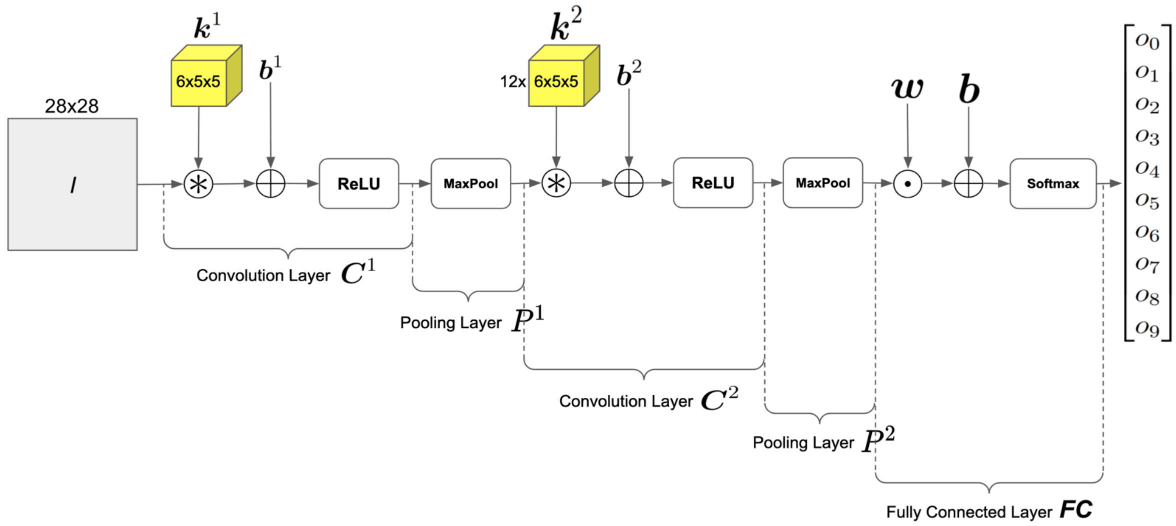

Figure 1: Abstract Architecture

where

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

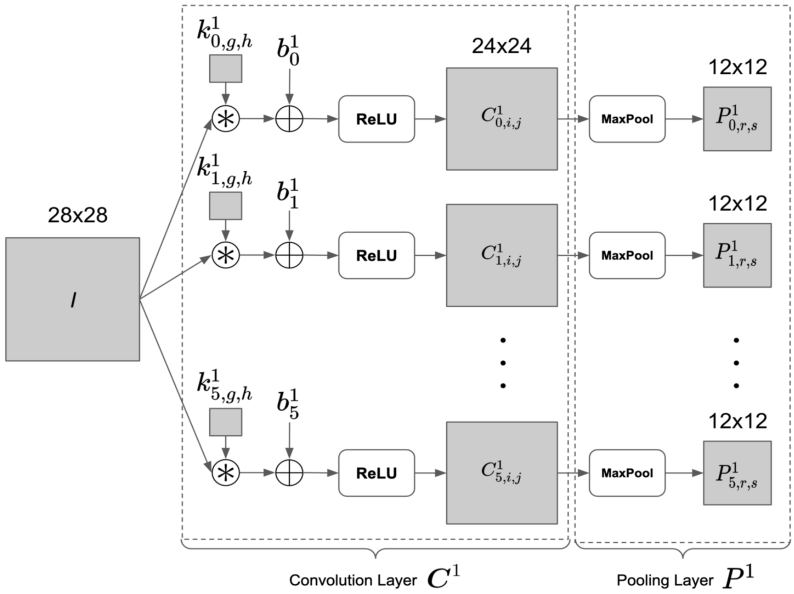

Figure 2: Detailed Architecture - part 1

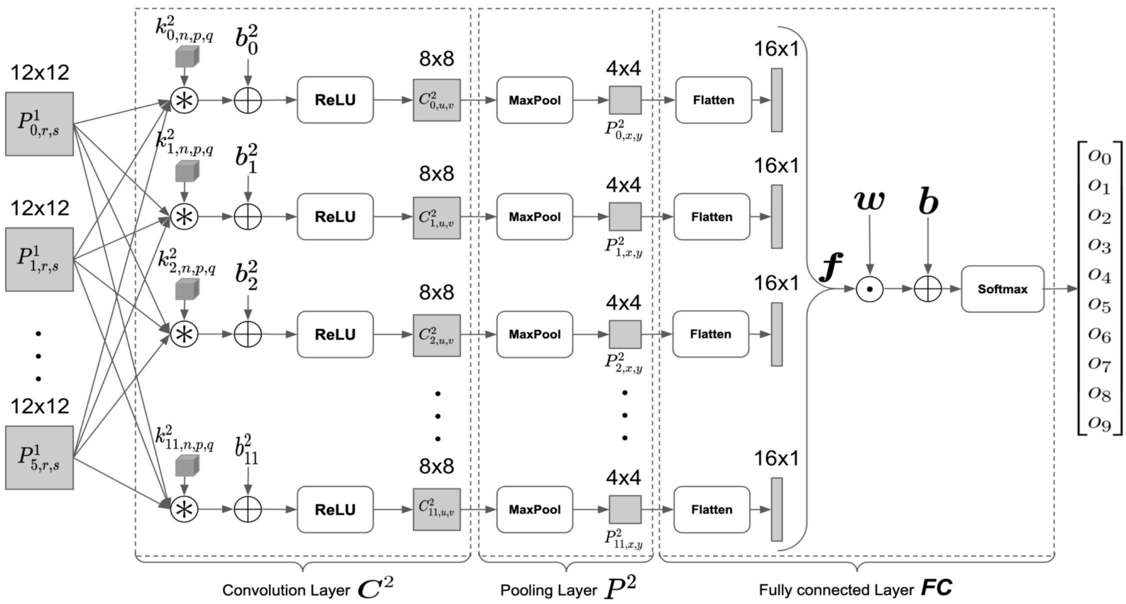

Figure 3: Detailed Architecture - part 2

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 $\boldsymbol{C}^\mathbf{1}$

$$
\begin{aligned}
&S^1_{nij} = \sum_{g=0}^{4}\sum_{h=0}^{4} I_{g+i,h+j}k^1_{ngh} + b^1_n, \\
&C^1_{nij} = \mathrm{ReLU}(S^1_{nij}) = \begin{cases} S^1_{nij} & \text{if } S^1_{nij} > 0 \\ 0 & \text{if } S^1_{nij} \le 0 \end{cases} \\
&n = 0, ..., 5;\ i, j = 0, ..., 23
\end{aligned}
\tag{1}
$$

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

3.2. Pooling Layer $\boldsymbol{P}^\mathbf{1}$

$$
\begin{aligned}
&P^1_{nrs} = \max(C^1_{nij}),\ i = 2r, 2r+1,\ j = 2s, 2s+1, \\
&n = 0, ..., 5;\ r, s = 0, ..., 11
\end{aligned}
\tag{2}
$$

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

3.3. Convolution Layer $\boldsymbol{C}^\mathbf{2}$

$$
\begin{aligned}
&S^2_{muv} = \sum_{n=0}^{5}\sum_{p=0}^{4}\sum_{q=0}^{4} P^1_{n,p+u,q+v}k^2_{mnpq} + b^2_m, \\
&C^2_{muv} = \mathrm{ReLU}(S^2_{muv}) = \begin{cases} S^2_{muv} & \text{if } S^2_{muv} > 0 \\ 0 & \text{if } S^2_{muv} \le 0 \end{cases} \\
&m = 0, ..., 11;\ u, v = 0, ..., 7
\end{aligned}
\tag{3}
$$

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

3.4. Pooling Layer $\boldsymbol{P}^\mathbf{2}$

$$
\begin{aligned}
&P^2_{mxy} = \max(C^2_{muv}),\ u = 2x, 2x+1,\ v = 2y, 2y+1, \\
&m = 0, ..., 11;\ x, y = 0, ..., 3
\end{aligned}
\tag{4}
$$

3.5. Fully Connected Layer $\boldsymbol{FC}$

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

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 ($\boldsymbol{k^1}$, $\boldsymbol{b^1}$, $\boldsymbol{k^2}$, and $\boldsymbol{b^2}$). Besides, the weight $\boldsymbol{w}$ and bias $\boldsymbol{b}$ 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`).

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

where *label* is the ground-truth value of the grayscale image $\boldsymbol{I}$, $\ln(O_{label})$ 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 $\boldsymbol{FC}$, the Convolution Layer $\boldsymbol{C^1}$, and the Convolution Layer $\boldsymbol{C^2}$.

4.1. Deriving gradients of parameters in the Fully Connected Layer

In the $\boldsymbol{FC}$ 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 ($i = 0, ..., 9$) 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 ($j = 0, ..., 191$). 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

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

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

$$\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial S_i},\ i = 0, ..., 9 \tag{9}$$

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

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

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

From (5) we have

$$O_{label} = \frac{e^{S_{label}}}{\sum\limits_{k=0}^{9} e^{S_k}} \tag{12}$$

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

$$O_{label} = \frac{e^{S_{label}}}{T} = e^{S_{label}} T^{-1} \tag{13}$$

So we can develop

$$\frac{\partial O_{label}}{\partial S_i} = \frac{\partial\left(e^{S_{label}} T^{-1}\right)}{\partial S_i},\ i = 0, ..., 9 \tag{14}$$

Next, let's consider two cases:

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

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

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

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

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

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 $C_2 = e^{S_0} + ... + e^{S_{i-1}} + e^{S_{i+1}} + ... + e^{S_9}$ 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

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

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

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

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

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

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

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

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

$$\mathbf{dLb} = \begin{bmatrix} O_0 \\ . \\ . \\ . \\ O_{label} - 1 \\ . \\ . \\ . \\ O_9 \end{bmatrix} \tag{26}$$

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

Applying the chain rule, we can represent

$$\frac{\partial L}{\partial w_{ij}} = \sum_{k=0}^{9} \frac{\partial L}{\partial S_k} \frac{\partial S_k}{\partial w_{ij}}, \ i = 0, ..., 9, \ j = 0, ..., 191 \tag{27}$$

From (5), we can derive

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

So,

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial S_i} f_j, \ i = 0, ..., 9, \ j = 0, ..., 191 \tag{29}$$

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 **dLw** is a 2D array of $\frac{\partial L}{\partial w_{ij}}$, we can represent

$$\mathbf{dLw} = \begin{bmatrix} \frac{\partial L}{\partial S_0} f_0 \ \cdots\cdots\cdots\cdots \ \frac{\partial L}{\partial S_0} f_{191} \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \frac{\partial L}{\partial S_9} f_0 \ \cdots\cdots\cdots\cdots \ \frac{\partial L}{\partial S_9} f_{191} \end{bmatrix} \tag{30}$$

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 ($P_{n,p+u,q+v}^1, k_{mnpq}^2, b_m^2$) ($n = 0, ..., 5, \ p = 0, ..., 4, \ q = 0, ..., 4$), 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 ($u = 0, ..., 7, \ v = 0, ..., 7$) 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 ($i = 0, ..., 9$) 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 $\frac{\partial L}{\partial b^2_m}$ $(m = 0, ..., 11)$

$$\frac{\partial L}{\partial b^2_m} = \sum_{u=0}^{7}\sum_{v=0}^{7} \frac{\partial L}{\partial S^2_{muv}} \frac{\partial S^2_{muv}}{\partial b^2_m} \tag{31}$$

$$\frac{\partial L}{\partial S^2_{muv}} = \frac{\partial L}{\partial C^2_{muv}} \frac{\partial C^2_{muv}}{\partial S^2_{muv}} \tag{32}$$

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

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^2_{muv} always belongs to a certain block of 4 elements (the C^2_{muv} and 3 other elements of the 3D array $\boldsymbol{C^2}$) and it may or may not be the maximum element out of 4 elements. Considering two following cases:

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

$$\frac{\partial L}{\partial C^2_{muv}} = \frac{\partial L}{\partial C^2_{mu_{max}v_{max}}} = \frac{\partial L}{\partial P^2_{mxy}} \tag{33}$$

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

$$\frac{\partial L}{\partial C^2_{mu_{max}v_{max}}} = \frac{\partial L}{\partial P^2_{mxy}} = \frac{\partial L}{\partial f_k} \tag{34}$$

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

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

From (5), we can derive

$$\frac{\partial S_i}{\partial f_k} = w_{ik} \tag{36}$$

Substitute (36) into (35),

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

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

$$\mathbf{dLf} = \mathbf{w}^T \cdot \mathbf{dLS} \tag{38}$$

Denote **dLP2** is a 3D array of $\frac{\partial L}{\partial \boldsymbol{P^2}}$ with shape = $\boldsymbol{P^2}$.shape = (12, 4, 4), then we can represent

$$\mathbf{dLP2} = \mathbf{dLf}.\text{reshape}(\boldsymbol{P^2}.\text{shape}) \tag{39}$$

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

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

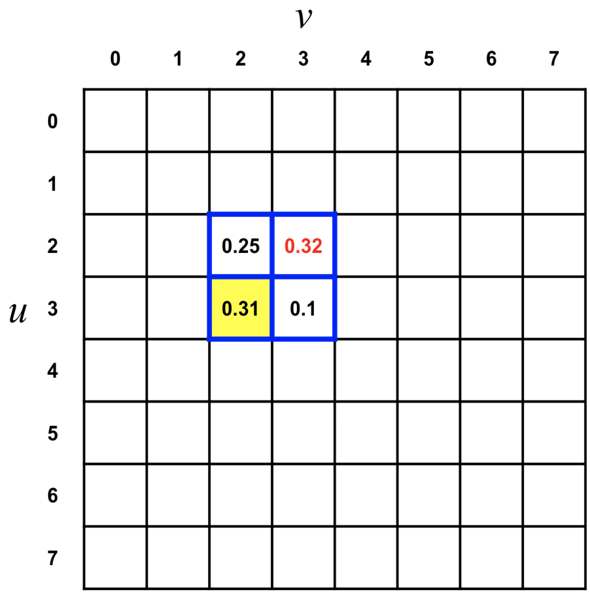

Figure 4: A sample $\boldsymbol{C^2}$[m]

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

$$\frac{\partial L}{\partial C^2_{muv}} = 0 \tag{40}$$

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

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

Denote **dLC2** is a 3D array of $\frac{\partial L}{\partial C^2_{muv}}$ elements $\left(\mathbf{dLC2}[m,u,v] = \frac{\partial L}{\partial C^2_{muv}}\right)$, with shape $=$ $\boldsymbol{C^2}$.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 **I2** (with shape $=$ $\boldsymbol{P^2}$.shape) like this:

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

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

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

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

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

$$\mathbf{dC2S2}[m,u,v] = \frac{\partial C^2_{muv}}{\partial S^2_{muv}} = \begin{cases} 1 \text{ if } S^2_{muv} > 0 \\ 0 \text{ if } S^2_{muv} \le 0 \end{cases} \tag{44}$$

So we can rewrite (32) as follows:

$$\mathbf{dLS2}[m,u,v] = \frac{\partial L}{\partial S^2_{muv}} = \mathbf{dLC2}[m,u,v]\mathbf{dC2S2}[m,u,v] \tag{45}$$

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

From (3), we can calculate

$$\frac{\partial S^2_{muv}}{\partial b^2_m} = 1 \tag{46}$$

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

$$\frac{\partial L}{\partial b^2_m} = \sum_{u=0}^{7}\sum_{v=0}^{7} \mathbf{dLS2}[m,u,v] \tag{47}$$

4.2.2. Deriving the gradient $\frac{\partial L}{\partial k^2_{mnpq}}$

$$\frac{\partial L}{\partial k^2_{mnpq}} = \sum_{u=0}^{7}\sum_{v=0}^{7} \frac{\partial L}{\partial S^2_{muv}}\frac{\partial S^2_{muv}}{\partial k^2_{mnpq}} \tag{48}$$

From (3), we can calculate

$$\frac{\partial S^2_{muv}}{\partial k^2_{mnpq}} = P^1_{n,p+u,q+v} \tag{49}$$

Substitute (49) into (48), we obtain

$$\frac{\partial L}{\partial k^2_{mnpq}} = \sum_{u=0}^{7}\sum_{v=0}^{7} \frac{\partial L}{\partial S^2_{muv}} P^1_{n,p+u,q+v} \tag{50}$$

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

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

$$\frac{\partial L}{\partial b^1_n} = \sum_{i=0}^{23}\sum_{j=0}^{23} \frac{\partial L}{\partial S^1_{nij}}\frac{\partial S^1_{nij}}{\partial b^1_n} \tag{51}$$

$$\frac{\partial L}{\partial S^1_{nij}} = \frac{\partial L}{\partial C^1_{nij}}\frac{\partial C^1_{nij}}{\partial S^1_{nij}} \tag{52}$$

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

Let's calculate $\frac{\partial L}{\partial P^1_{nrs}}$.
From (3) and the architecture diagrams, we can see that a specific P^1_{nrs} may affect all S^2_{muv}. Therefore,

$$\frac{\partial L}{\partial P^1_{nrs}} = \sum_{m=0}^{11} \sum_{u=0}^{7} \sum_{v=0}^{7} \frac{\partial L}{\partial S^2_{muv}} \frac{\partial S^2_{muv}}{\partial P^1_{nrs}} \tag{54}$$

We only need to calculate $\frac{\partial S^2_{muv}}{\partial P^1_{nrs}}$ because $\frac{\partial L}{\partial S^2_{muv}}$ was already calculated in (45).
From (3), we see that for a specific triple (m, u, v), there is a 3D region of P^1_{nrs} elements $(n = 0, ..., 5,\ r = u, ..., u + 4,\ s = v, ..., v + 4)$ contributing to S^2_{muv}, and $\frac{\partial S^2_{muv}}{\partial P^1_{nrs}} = k^2_{mnpq},\ p = r - u,\ q = s - v$. If an element P^1_{nrs} is outside that 3D region, it doesn't contribute to S^2_{muv}, and therefore $\frac{\partial S^2_{muv}}{\partial P^1_{nrs}} = 0$. So denote **dS2P1** is a 6D array of $\frac{\partial S^2_{muv}}{\partial P^1_{nrs}}$ elements $\left(\mathbf{dS2P1}[n, r, s, m, u, v] = \frac{\partial S^2_{muv}}{\partial P^1_{nrs}}\right)$ with **dS2P1**.shape = $\boldsymbol{P^1}$.shape + $\boldsymbol{C^2}$.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 $(m, u, v),\ m = 0, ..., 11,\ u = 0, ..., 7,\ v = 0, ..., 7$:

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

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

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

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

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

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

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

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

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

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

$$\mathbf{dC1S1}[n, i, j] = \frac{\partial C^1_{nij}}{\partial S^1_{nij}} = \begin{cases} 1 \text{ if } S^1_{nij} > 0 \\ 0 \text{ if } S^2_{nij} \leq 0 \end{cases} \tag{59}$$

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

$$\frac{\partial L}{\partial S^1_{nij}} = \mathbf{dLC1}[n,i,j]\mathbf{dC1S1}[n,i,j] \tag{60}$$

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

$$\frac{\partial S^1_{nij}}{\partial b^1_n} = 1 \tag{61}$$

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

$$\boxed{\frac{\partial L}{\partial b^1_n} = \sum_{i=0}^{23}\sum_{j=0}^{23} \mathbf{dLC1}[n,i,j]\mathbf{dC1S1}[n,i,j]} \tag{62}$$

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

$$\frac{\partial L}{\partial k^1_{ngh}} = \sum_{i=0}^{23}\sum_{j=0}^{23} \frac{\partial L}{\partial S^1_{nij}} \frac{\partial S^1_{nij}}{\partial k^1_{ngh}} \tag{63}$$

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

$$\frac{\partial S^1_{nij}}{\partial k^1_{ngh}} = I_{n,g+i,h+j} \tag{64}$$

Substitute (64) into (63), we obtain

$$\frac{\partial L}{\partial k^1_{ngh}} = \sum_{i=0}^{23}\sum_{j=0}^{23} \frac{\partial L}{\partial S^1_{nij}} I_{n,g+i,h+j} \tag{65}$$

4.4. Update parameters

$$\boxed{\begin{aligned}
&k^1_{ngh} = k^1_{ngh} - \eta\tfrac{\partial L}{\partial k^1_{ngh}},\ n = 0, ..., 5;\ g, h = 0, ..., 4,\\
&b^1_n = b^1_n - \eta\tfrac{\partial L}{\partial b^1_n},\ n = 0, ..., 5,\\
&k^2_{mnpq} = k^2_{mnpq} - \eta\tfrac{\partial L}{\partial k^2_{mnpq}},\ m = 0, ..., 11,\ n = 0, ..., 5;\ p, q = 0, ..., 4,\\
&b^2_m = b^2_m - \eta\tfrac{\partial L}{\partial b^2_m},\ m = 0, ..., 11,\\
&w_{ij} = w_{ij} - \eta\tfrac{\partial L}{\partial w_{ij}},\ i = 0, ..., 9,\ j = 0, ..., 191,\\
&b_i = b_i - \eta\tfrac{\partial L}{\partial b_i},\ i = 0, ..., 9.
\end{aligned}} \tag{66}$$

Where η is the learning rate.