

## 卷積神經網路 (五) 卷積層



山與水你和我

體系結構小白，影像處理小白

15 人贊同了該文章

上一篇

山與水你和我：卷積神經網路 (四) 池化層

實現了最大池化的前向反向傳播。

緊接著卷積層的前向與反向傳播。

卷積神經網路的「卷積」可能和數學上的捲積意義不一樣：

- 前者，卷積是在局部視窗內，卷積核和輸入做線性相乘求和；
- 後者，卷積是先對濾波器（卷積核）矩陣做180°的翻轉，再與輸入做線性相乘求和；

前者還有一種說法「協相關Correlation」。這裡不對這二者做嚴格的區分，我只是想實現影像分類這個任務，翻不翻轉不是我需要關心的事情。

### 向前

卷積層：卷積核 $O \times I \times k \times k$ ，偏見 $O$ ；步長 $s$ ；kernel 大小 $k \times k$ (暫時只考慮正方形視窗)；padding 暫時不考慮

輸入： $B \times I \times H \times W$

輸出： $B \times O \times H_2 \times W_2$ ；其中 $H_2 = \frac{H-k}{s} + 1$ ， $W_2 = \frac{W-k}{s} + 1$ 。

個人實現的時候，摒棄卷積的公式，靠想像，流程如下：

- 一個batch,  $B$ 張影像的捲積分開計算，互不影響；
- 首先看有 $O$ 個卷積核，每個卷積核 $I \times k \times k$ ；
- 然後看輸入有 $B$ 張影像的特徵，每張特徵 $I \times H \times W$ ；
- 先考慮0 號卷積核，卷積核 $I \times k \times k$ ，在0 號特徵上做滑動，每次滑動在0 號特徵輸入「一個 $I \times k \times k$ 的內容來，二者做相乘求和，得到一個值，這個值是0號特徵的0 號卷積核的

×

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。

立即登入/註冊

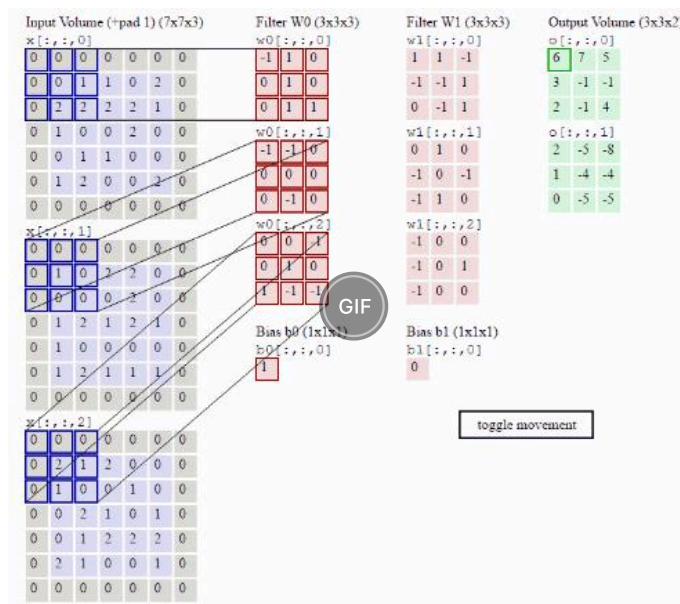


卷積核;

- 再考慮2 號卷積核，在0 號特徵上做滑動，一共有  $H_2 \times W_2$  個這種值，都填入0 號特徵的2 號卷積核；
- .....
- 再考慮  $O - 1$  號卷積核，在0 號特徵上做滑動，一共有  $H_2 \times W_2$  個這種值，都填入0 號特徵的  $O - 1$  號卷積核中；
- 至此，所有捲積核在0 號特徵上的捲積結果都計算完畢，共得到  $O \times H_2 \times W_2$  個值；
- 重複上面的捲積核，在1 號特徵上做滑動，共得到  $O \times H_2 \times W_2$  個值；
- 重複上面的捲積核，在2 號特徵上做滑動，共得到  $O \times H_2 \times W_2$  個值；
- .....
- 重複上面的捲積核，在  $B - 1$  號特徵上做滑動，共得到  $O \times H_2 \times W_2$  個值；
- $O$  個卷積核在  $B$  張特徵輸入的捲積都計算完畢，共得到  $B \times O \times H_2 \times W_2$  個值。

看起來有點複雜，但只要腦海裡過一遍，就十分簡單了。

整個卷積過程可以看下面的圖。

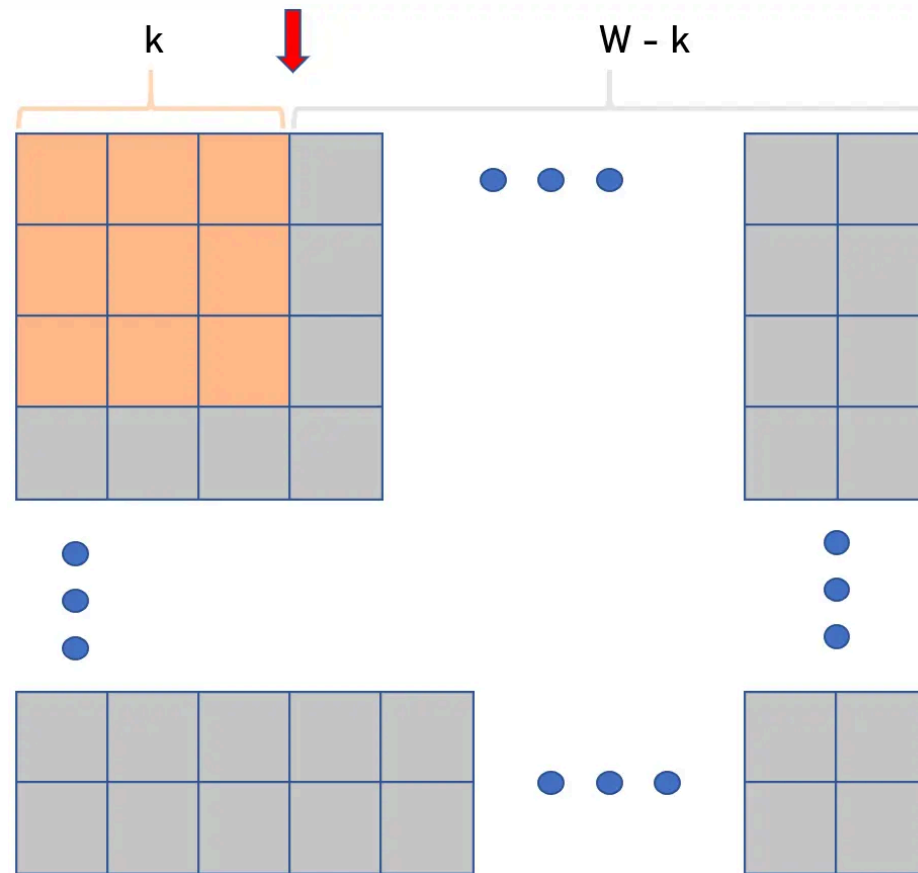


雖然卷積操作不複雜，還是有許多細節，例如：為什麼  $W_2 = \frac{W-k}{s} + 1$ （不考慮padding）？如下圖，橘色區域一定可以得到一個值，所以結果有一個1；剩餘的部分有  $W - k$  個像素的長度，從

登入即可查看超5億專業優質內容

超5千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



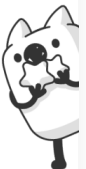


上面的捲積過程可以寫成

```
// 为卷积做准备
const int H_radius = H - radius; // 避免每次循环重新计算 H - radius
const int W_radius = W - radius;
const int window_range = kernel_size * kernel_size; // 卷积核一个二维平面的大小, 用来算偏
const int* const __offset = this->offset.data(); // 获取偏移量指针
// 首先每张图像分开卷积
for(int b = 0; b < batch_size; ++b) {
    // 获取第 b 张图像的起始地址, in_channels X 224 X 224
    data_type* const cur_image_features = input[b]->data;
    for(int o = 0; o < out_channels; ++o) { // 每个卷积核
```

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



```

int cnt = 0; // 记录每次卷积结果存放的位置
for(int x = radius; x < H_radius; x += stride) {
    for(int y = radius; y < W_radius; y += stride) { // 每一次滑动, 卷积核和输入
        data_type sum_value = 0.f;
        const int coord = x * W + y; // 当前点对于这个通道的特征图的位移
        for(int i = 0; i < in_channels; ++i) { // 每个点有多个通道
            const int start = i * length + coord; // 输入的第 i 张特征图在 (x, y)
            const int start_w = i * window_range; // 第 o 个卷积核的第 i 个通道
            for(int k = 0; k < window_range; ++k) // 遍历局部窗口
                sum_value += cur_image_features[start + __offset[k]] * cur_w_p[k];
        }
        sum_value += this->bias[o]; // 别忘记加上 b
        out_ptr[cnt] = sum_value; // 一次线性相乘求和的结果, 放到输出的 cnt 位置
        ++cnt; // 存放的位置 + 1
    }
}
}

```

上面一些細節

- 和之前MaxPool2d 類似，使用了offset 數組，用一次循環找到局部視窗內所有的點，求線性相乘求和；offset 數組的解法和MaxPool2d 類似；
- for 迴圈的o 和b 迴圈可以換一下位置，結果等價；
- bias 是一個 $O$ 維向量 - 其實，在寫之前，我以為的bias 是每次線性相乘求和得到一個值，就有一個對應位置的bias 值，滑動到下一個窗口，相乘求和得到一個值就會再加上另一個bias 值，是我想太多了！我這個想法其實也可以，但是bias 可學習參數的數目必須是固定的，也就是說輸出大小固定，卷積層接收的輸入大小固定。怎麼簡單怎麼來！ $O$ 維向量就可以有不錯的訓練效果了。
- 和之前ReLU, MaxPool 不同，卷積層中輸入通道和輸出通道**不一定相等**！每次卷積核 $I \times k \times k$ 與輸入特徵做局部窗口 $I \times k \times k$ 的線性相乘求和，求的是這 $I$ 個通道所有相乘結果的和。

和之前做ReLU 層、MaxPool2d 層類似，卷積層也存在一些緩衝區，避免每次forward 重新分配空間，如卷積層輸出output 等變量，

```

// 获取输入特征图的信息
const int batch_size = input.size();
const int H = input[0]->H;

```

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



```
// 计算输出的特征图大小
const int out_H = std::floor((H - kernel_size - 2 * padding) / stride) + 1;
const int out_W = std::floor((W - kernel_size - 2 * padding) / stride) + 1;
const int out_length = out_H * out_W; // 输出的特征图，一个通道的输出有多大，111 X 111, 7
// 为卷积做准备
const int radius = int((kernel_size - 1) / 2);
// 如果是第一次经过这一层，分配空间(这里灵活性差点，如果形状不是一样的，可能会崩溃，要重新分配)
if(this->output.empty()) {
    // 分配输出的张量
    this->output.reserve(batch_size);
    for(int b = 0; b < batch_size; ++b) // B X 16 X 111 X 111
        this->output.emplace_back(new Tensor3D(out_channels, out_H, out_W));
    // 辅助变量 offset 只求一遍，虽然方便，但没有局部变量快
    int pos = 0;
    for(int x = -radius; x <= radius; ++x)
        for(int y = -radius; y <= radius; ++y) {
            this->offset[pos] = x * W + y;
            ++pos;
        }
}
```

卷積核在初始化時已經記住一些固有的變量，例如核大小 $k$ ，步長 $s$ ，卷積核數目 $O$ （輸出有幾個通道），每個卷積核處理 $I$ 個通道的線性相乘求和，在建立卷積核時，需要對卷積核可學習參數做權值初始化，比較常用的有常態分佈、平均值分佈、xavier\_init、kaiming\_init等方法，本次實驗使用最簡單的常態分佈，初始化如下：

```
Conv2D::Conv2D(std::string _name, const int _in_channels, const int _out_channels, con
: Layer(_name),
  bias(_out_channels),
  in_channels(_in_channels),
  out_channels(_out_channels),
  kernel_size(_kernel_size),
  stride(_stride),
  params_for_one_kernel(_in_channels * _kernel_size * _kernel_size),
  offset(_kernel_size * _kernel_size) {
    // 验证参数合法性
    assert(_kernel_size & 1 and _kernel_size >= 3 and "卷积核的大小必须是正奇数 !");
    assert(_in_channels > 0 and _out_channels > 0 and _stride > 0);
    // 首先给权值矩阵 weights 和偏置 b 分配空间
    this->weights.reserve(out_channels);
```

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



```

        weights.emplace_back(new Tensor3D(in_channels, kernel_size, kernel_size));
    }
    // 随机初始化, 这里用的是正态分布初始化
    this->seed.seed(212);
    std::normal_distribution<float> engine(0.0, 1.0);
    for(int o = 0; o < out_channels; ++o) bias[o] = engine(this->seed) / random_times;
    for(int o = 0; o < out_channels; ++o) {
        data_type* data_ptr = this->weights[o]->data;
        for(int i = 0; i < params_for_one_kernel; ++i)
            data_ptr[i] = engine(this->seed) / random_times;
    }
}

```

## 落後

前向過程相對簡單。backward 過程就複雜了。

一開始我看劉建平大佬的部落格卷積神經網路(CNN)反向傳播演算法，**很推薦**，入門了解下很合適；也**很不推薦**，裡面的例子只是特殊情況，容易誤導。

因此，我打算不看網路上那些複雜的公式，老實畫圖，看每個輸出，有哪些地方是參與了計算的（有貢獻）。

首先，要先明確一點，之前計算ReLU 層和MaxPool 層的反向傳播，根據從下一層傳回來的梯度  $\delta^l$ ，只計算了傳給上一層的梯度  $\delta^{l-1}$ ，這是因為它們本身沒有可學習參數，但卷積層不同，卷積層具有自己的參數（權值矩陣  $O \times I \times k \times k$  個參數和偏移  $O$  個參數），這些參數也確實參與了輸出的計算中，因此，卷積層梯度的反向傳播由兩部分，已知從下一層傳回來的梯度  $\delta^l$

- 求傳給上一層的梯度  $\delta^{l-1}$
- 求卷積核參數  $W$  和  $bias$  的梯度

先求卷積核參數  $W$  和  $bias$  的梯度，分別假設為  $\delta W$  總和  $\delta b$ ，比較簡單一點。

## $W$ 和 $bias$ 的梯度

考慮一個簡單的例子，只考慮一張圖片的情況，batch\_size = 1，

卷積核：  $1 \times 3 \times 3$

登入即可查看超5億專業優質內容

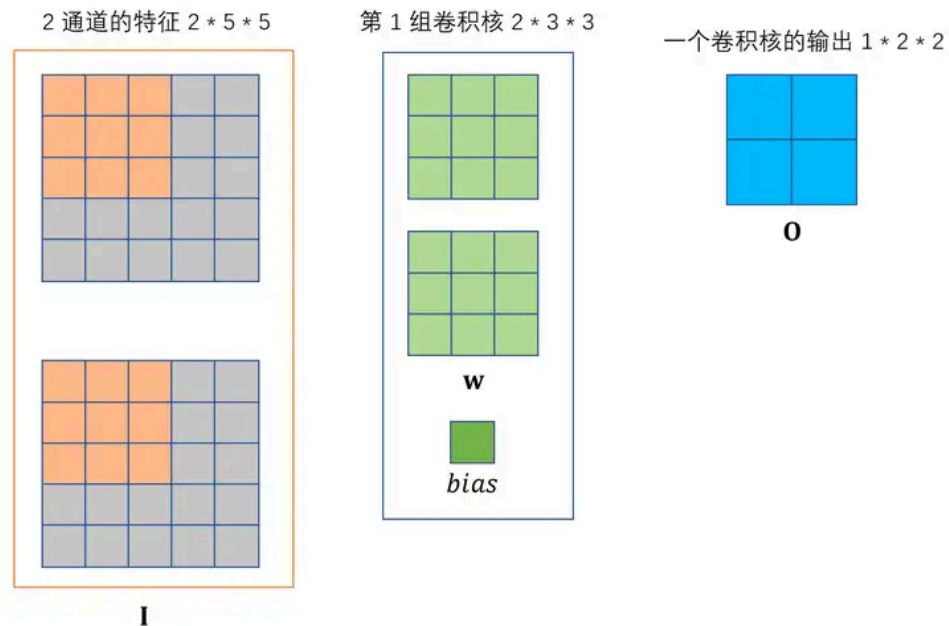
超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



參數：步長  $s = 2$

輸出： $1 \times 2 \times 2$

如下圖

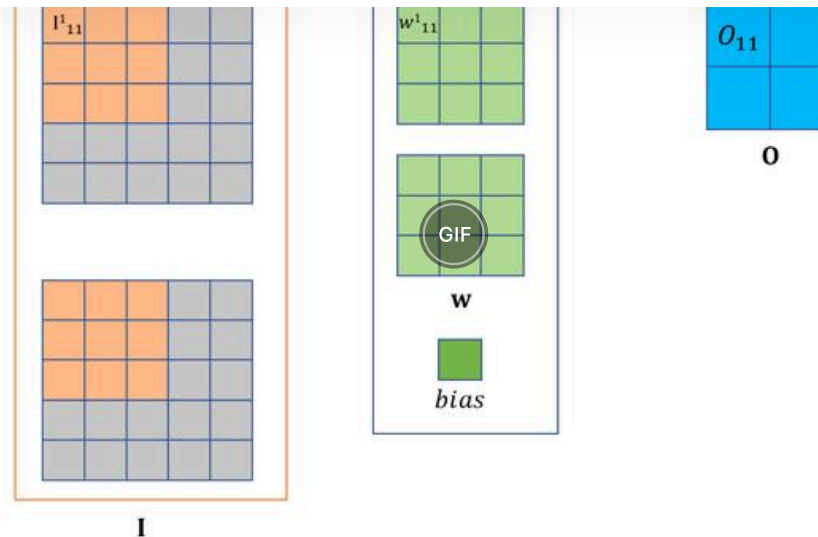


目標是求權值矩陣 $W$ 的梯度，例如求上圖卷積核第一個通道的第一個權值的梯度 $\delta W_{11}^1$ （上標1代表第1個通道，下標11代表第一個權值， $\delta W_{11}^1$ ），就得找卷積核的所有輸出中，有哪些是 $W_{11}^1$ 參與計算得到的，如下

登入即可查看超5億專業優質內容

超5千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。





可以看得出， $W_{11}^1$  參與了輸出  $O^1$  的所有點的計算，即

$$O_{11}^1 = W_{11}^1 * I_{11}^1 + \dots + bias$$

$$O_{12}^1 = W_{11}^1 * I_{13}^1 + \dots + bias$$

$$O_{21}^1 = W_{11}^1 * I_{31}^1 + \dots + bias$$

$$O_{22}^1 = W_{11}^1 * I_{33}^1 + \dots + bias$$

假設從下一層傳回來的梯度是  $\delta^l$ ， $1 \times 2 \times 2$ ，求  $W_{11}^1$  的梯度  $\delta W_{11}^1$ ，就需要對應的梯度相加

$$\delta W_{11}^1 = \delta_{11}^{l1} * I_{11}^1 + \delta_{12}^{l1} * I_{13}^1 + \delta_{21}^{l1} * I_{31}^1 + \delta_{22}^{l1} * I_{33}^1$$

其中  $\delta_{11}^{l1}$  代表第  $l$  層梯度的第 1 個通道，位置 11 的梯度值，其他同理。

同理，再舉個例子，求  $W_{32}^2$  的梯度  $\delta W_{32}^2$ ，

$$O_{11}^1 = W_{32}^2 * I_{32}^2 + \dots + bias$$

$$O_{12}^1 = W_{32}^2 * I_{34}^2 + \dots + bias$$

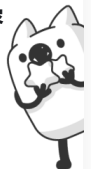
$$O_{21}^1 = W_{32}^2 * I_{52}^2 + \dots + bias$$

$$O_{22}^1 = W_{32}^2 * I_{54}^2 + \dots + bias$$

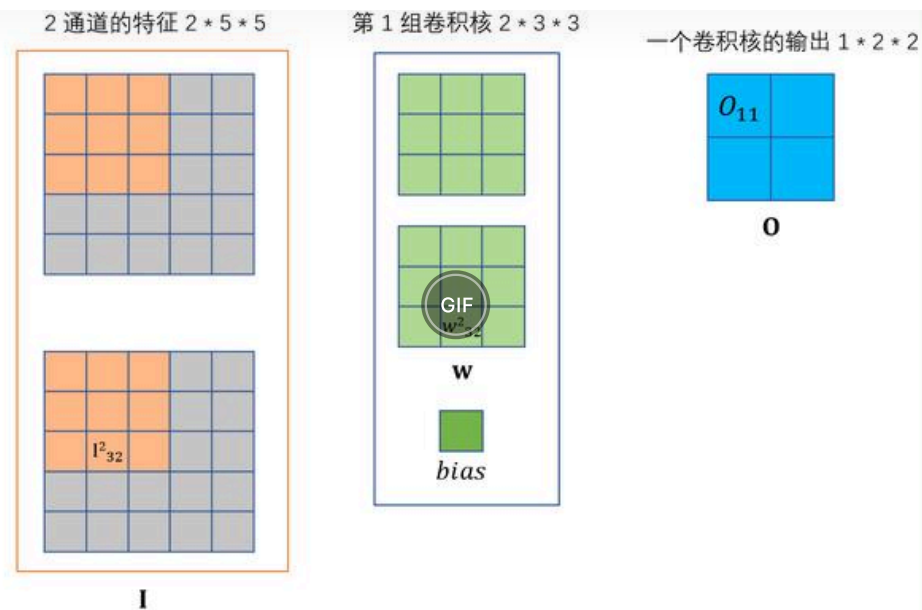
$$\delta W_{32}^2 = \delta_{11}^{l1} * I_{32}^2 + \delta_{12}^{l1} * I_{34}^2 + \delta_{21}^{l1} * I_{52}^2 + \delta_{22}^{l1} * I_{54}^2$$

登入即可查看超5億專業優質內容

超5千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。







根據以上，可以得到**1** 個卷積核參數 **$W$** 的梯度，共有 **$2 \times 3 \times 3$** 個，和1個卷積核的參數量一致。

接下來求對偏置的梯度 **$\delta b$** ，根據上面可以得到，對每個卷積核而言，就一個偏置值 **$bias$** ，且參與了這個卷積核的所有輸出值的計算，因此， **$\delta b$** 需要累加 **$\delta^l$** ， **$1 \times 2 \times 2$** 個值，

$$\delta b = \delta_{11}^l + \delta_{12}^l + \delta_{21}^l + \delta_{22}^l$$

具體實現可參考下面的程式碼幫助理解

```
// 这里默认不记录梯度的历史信息，w, b 之前的梯度全部清空
for(int o = 0; o < out_channels; ++o) this->weights_gradients[o]->set_zero();
for(int o = 0; o < out_channels; ++o) this->bias_gradients[o] = 0;
// 先求 weights, bias 的梯度
for(int b = 0; b < batch_size; ++b) { // 这个 batch 每张图像对应一个梯度，多个梯度取平均
    // 首先，遍历每个卷积核
    for(int o = 0; o < out_channels; ++o) {
        // 第 b 张图像的梯度，找到第 o 个通道的起始地址
        data_type* o_delta = delta[b]->data + o * out_H * out_W;
        // 卷积核的每个 in 通道，分开求
        for(int i = 0; i < in_channels; ++i) {
            // 第 b 张输入，找到第 i 个通道的起始地址
            data_type* in_ptr = __input[b]->data + i * H * W;
```

登入即可查看**超5億**專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



```
// // 遍历的是卷积核的一个通道，求每个参数的梯度
for(int k_x = 0; k_x < kernel_size; ++k_x) {
    for(int k_y = 0; k_y < kernel_size; ++k_y) {
        // 记录一张图像的 W 梯度
        data_type sum_value = 0;
        for(int x = 0; x < out_H; ++x) {
            // delta 在这个通道的第 x 行，每行 out_W 个数
            data_type* delta_ptr = o_delta + x * out_W;
            // 对应的输入 I 在这个通道的第 (x * stride + k_x) 行，每行 W 个数
            // 注意 * stride 每次在 input 中是跳着找的，+ k_x 是找竖直方向上的
            data_type* input_ptr = in_ptr + (x * stride + k_x) * W;
            for(int y = 0; y < out_W; ++y) {
                // 当前 w 的梯度，由参与计算的输入和返回的梯度相乘，累加
                sum_value += delta_ptr[y] * input_ptr[y * stride + k_y];
            }
            // 更新到 weight_gradients，注意除以了 batch_size；这里是 +=，不是 =
            w_ptr[k_x * kernel_size + k_y] += sum_value / batch_size;
        }
    }
}

// 计算 b 的梯度
data_type sum_value = 0;
// 需要计算多个通道输出的梯度
for(int d = 0; d < out_length; ++d) sum_value += o_delta[d];
// 除以 batch_size
bias_gradients[o] += sum_value / batch_size;
}
}
```

細節：

- 上面的例子是只有一張圖， $\text{batch\_size} = 1$ ；如果 $\text{batch\_size} \neq 1$ ，則累加所有圖片的梯度，然後除以 $\text{batch\_size}$ 取平均，這樣可以緩解一些異常梯度的值，使得梯度更準確。
- 計算梯度需要輸入 $I$ ，因此在forward要記錄 $I$ 。
- 看起來過程不難，但for迴圈有7層！寫著寫著有點超乎我想像，仔細捋捋。主要是計算索引要千萬小心，例如最開始的三個定位操作

```
// 第 b 张图像的梯度，找到第 o 个通道的起始地址
data_type* o_delta = delta[b]->data + o * out_H * out_W;
```

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



```
// 第 o 个卷积核，找到第 i 个通道的起始地址
data_type* w_ptr = weights_gradients[o]->data + i * kernel_size * kernel_size;
```

後續的 $k_x, k_y$  循環，遍歷的是卷積核的一個通道，求每個 $\delta W$  參數的梯度

```
// 遍历的是卷积核的一个通道 · 求每个参数的梯度
for(int k_x = 0; k_x < kernel_size; ++k_x) {
    for(int k_y = 0; k_y < kernel_size; ++k_y) {

    }
}
```

求每個參數的梯度的具體運算如下，out\_H, out\_W 遍歷的是輸出的一個通道 $H_2 \times W_2$ ,

```
// 记录一张图像的 w 梯度
data_type sum_value = 0;

for(int x = 0; x < out_H; ++x) {

    // delta 在这个通道的第 x 行 · 每行 out_W 个数
    data_type* delta_ptr = o_delta + x * out_W;

    // 对应的输入 I 在这个通道的第 (x * stride + k_x) 行 · 每行 W 个数 ·
    // 注意 * stride 每次在 input 中是跳着找的, + k_x 是我竖直方向上的偏移量; 下面的 y * s1
    data_type* input_ptr = in_ptr + (x * stride + k_x) * W;

    for(int y = 0; y < out_W; ++y) {
        // 当前 w 的梯度, 由参与计算的输入和返回的梯度相乘 · 累加
        sum_value += delta_ptr[y] * input_ptr[y * stride + k_y];
    }
}

// 更新到 weight_gradients, 注意除以了 batch_size ;
// 这里是 += · 不是 =, 一个 batch 的梯度累加
w_ptr[k_x * kernel_size + k_y] += sum_value / batch_size;
```

稍微有點複雜，不難，就是容易出錯。

到此，對 $W, b$ 的梯度求解完畢

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



給定下一層傳回來的梯度 $\delta^l$ ，解傳給上一層的梯度 $\delta^{l-1}$ 。這也是比較坑的一個地方，劉建平的做法是

这上面9个式子其实可以用一个矩阵卷积的形式表示，即：

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \delta_{11} & \delta_{12} & 0 \\ 0 & \delta_{21} & \delta_{22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{pmatrix} = \begin{pmatrix} \nabla a_{11} & \nabla a_{12} & \nabla a_{13} \\ \nabla a_{21} & \nabla a_{22} & \nabla a_{23} \\ \nabla a_{31} & \nabla a_{32} & \nabla a_{33} \end{pmatrix}$$

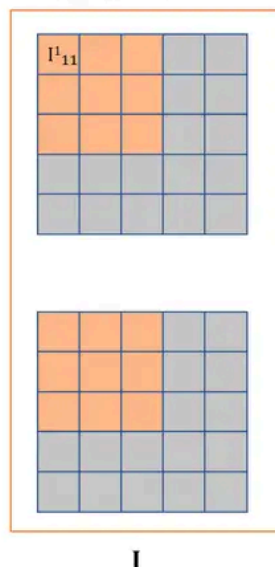
为了符合梯度计算，我们在误差矩阵周围填充了一圈0，此时我们将卷积核翻转后和反向传播的梯度误差进行卷积，

我一開始看的時候，權值矩陣為什麼要翻轉我是看明白了，但還有幾點不明白，① 只填一圈0？  
② 這是步長為1的情況，那如果步長大於1，比如說步長為2時，也是這麼做嗎？③ 最後的捲積步長是多少？1？

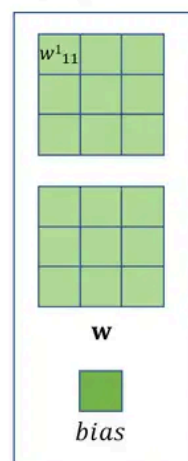
帶著這些疑問，我舉了個例子，畫畫圖，為了方便表示， $\delta^{l-1}$ 記作 $\bar{\delta}$ 。

和之前一樣，例如求 $\bar{\delta}_{11}^1$ ，傳回給上一層第一個通道，座標11的位置的梯度，就得找這個位置的輸入 $I_{11}^1$ 參與了哪些計算？對所有輸出中的哪些值做了貢獻？

2 通道的特征 2 \* 5 \* 5



第 1 组卷积核 2 \* 3 \* 3



一个卷积核的输出 1 \* 2 \* 2



額，很遺憾， $I_{11}^1$ 只參與了一次計算

登入即可查看超5億專業優質內容

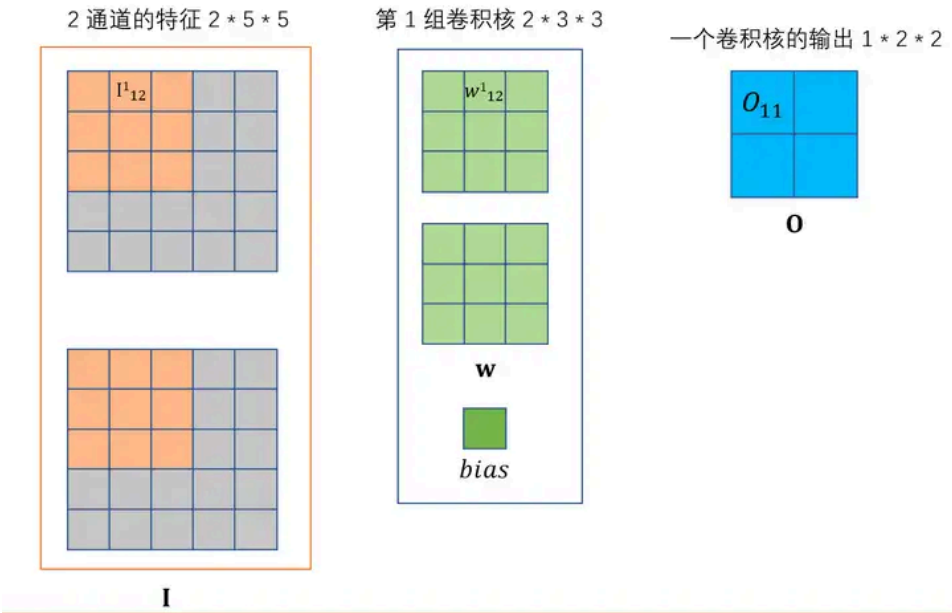
超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



回給上一層的梯度，這個位置的值 $\bar{\delta}_{11}^1$ 為

$$\bar{\delta}_{11}^1 = \delta_{11}^1 * w_{11}^1$$

換一個位置，找 $I_{12}^1$ 參與了哪些計算



也是只有一次計算

$$O_{11}^1 = \dots + w_{12}^1 * I_{12}^1 + \dots + bias,$$

$$\text{因此, } \bar{\delta}_{12}^1 = \delta_{11}^1 * w_{12}^1$$

再換一個位置，找 $I_{13}^1$ 參與了哪些計算

登入即可查看超5億專業優質內容

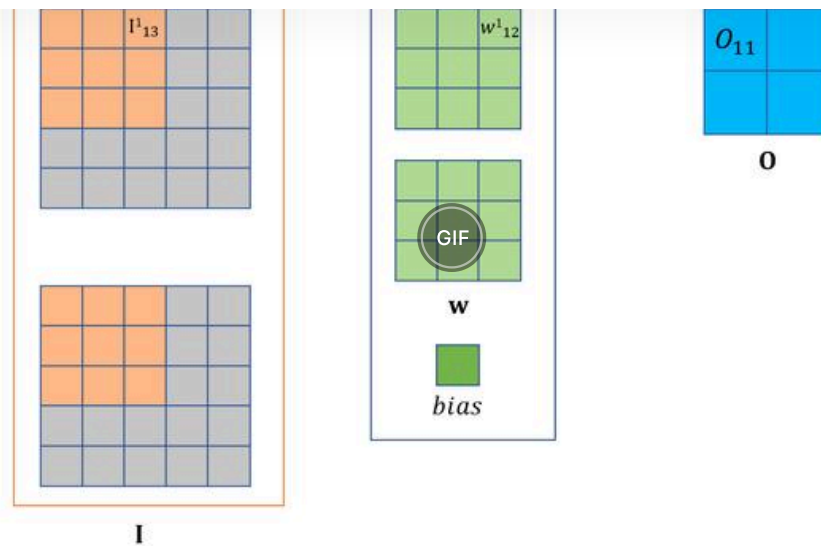
超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



知乎

首發於  
電腦視覺基礎

切換模式



$I_{13}^1$  參與了兩次計算

$$O_{11}^1 = \dots + \mathbf{w}_{13}^1 * I_{13}^1 + \dots + \mathbf{bias}$$

$$O_{12}^1 = \dots + \mathbf{w}_{11}^1 * I_{13}^1 + \dots + \mathbf{bias}$$

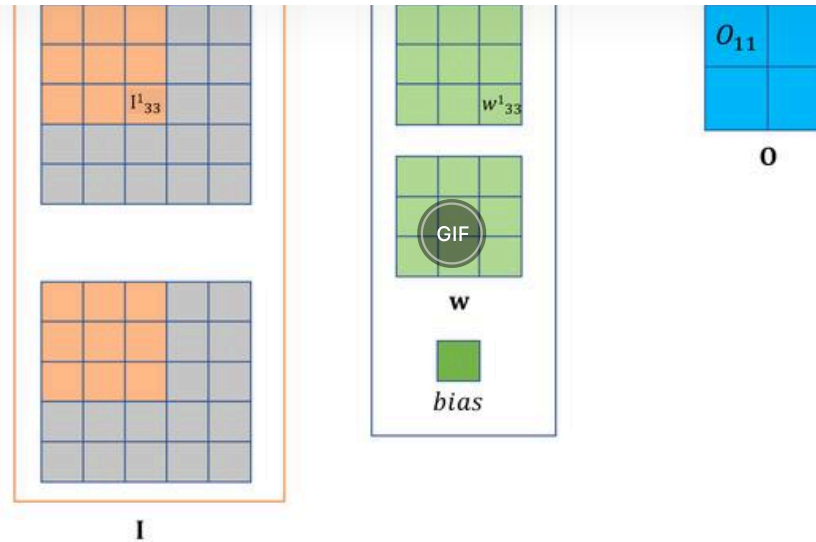
因此,  $\delta_{13}^1 = \delta_{11}^1 * \mathbf{w}_{13}^1 + \delta_{12}^1 * \mathbf{w}_{11}^1$

繼續找其他位置參與的計算, 例如  $I_{33}^1$

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。





r 如上圖可  $I^1_{33}$  共參與了4 次計算,

$$O^1_{11} = \dots + W^1_{33} * I^1_{33} + \dots + bias$$

$$O^1_{12} = \dots + W^1_{31} * I^1_{33} + \dots + bias$$

$$O^1_{21} = \dots + W^1_{13} * I^1_{33} + \dots + bias$$

$$O^1_{22} = \dots + W^1_{11} * I^1_{33} + \dots + bias$$

$$\text{由此, } \bar{\delta}^1_{33} = \delta^{l1}_{11} * W^1_{33} + \delta^{l1}_{12} * W^1_{31} + \delta^{l1}_{21} * W^1_{13} + \delta^{l1}_{22} * W^1_{11},$$

從這個例子也可以看得出, 確實有翻轉的意思

。

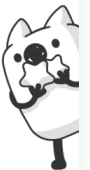
經過本人的總結, 求  $\bar{\delta}$  這一步驟等價於另一個卷積:

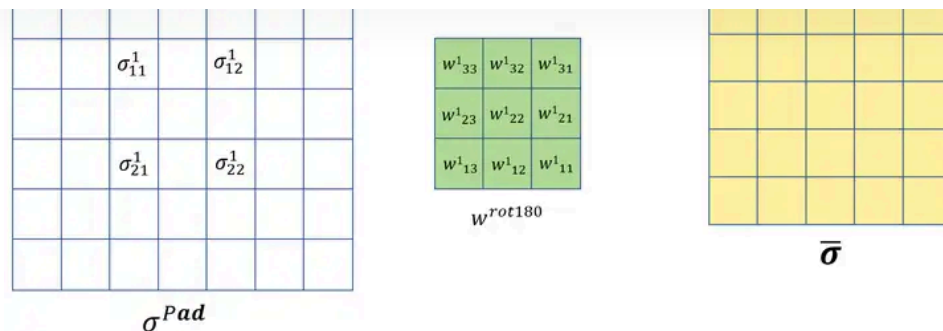
- 將從下一層傳回來的梯度  $\delta^l$  (為方便記作  $\delta$ ), 外圍填充  $k - p - 1$  行的0 ( $p=0$ , padding), 資料之間填入  $s - 1$  行的0, 得到  $\delta^{Pad}$ ,  $7 \times 7$
- 將權值矩陣  $2 \times 3 \times 3$  在  $3 \times 3$  這個維度平面做180 °的翻轉, 得到  $W^{rot180}$  如下圖
- $W^{rot180}$  對  $\delta^{Pad}$  做步長為1的捲積, 輸出  $\bar{\delta}$ ,  $5 \times 5$ ,

如下圖示意

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。





對上圖做一些簡單的運算，即可發現，跟之前舉的幾個例子都吻合！（實際上可以參考轉置卷積的forward 過程）。

回到一開始，劉建平大佬博客中的例子，例子沒錯，但只是特殊情況，還有很多細節沒有說明白，

这上面9个式子其实可以用一个矩阵卷积的形式表示，即：

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \delta_{11} & \delta_{12} & 0 \\ 0 & \delta_{21} & \delta_{22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{pmatrix} = \begin{pmatrix} \nabla a_{11} & \nabla a_{12} & \nabla a_{13} \\ \nabla a_{21} & \nabla a_{22} & \nabla a_{23} \\ \nabla a_{31} & \nabla a_{32} & \nabla a_{33} \end{pmatrix}$$

为了符合梯度计算，我们在误差矩阵周围填充了一圈0，此时我们将卷积核翻转后和反向传播的梯度误差进行卷积，

回顧我之前的三個疑問，① 只填一圈0？② 這是步長為1 的情況，那如果步長大於1，比如說步長為2 時，也是這麼做嗎？③ 最後的捲積步長是多少？1？

答① 外圍填充k - p - 1 圈0 ② 步長s 不為1 時，內部填充s - 1 圈0 ③ 最後的捲積步長是1。

問題好像解決了？

嚴格來說，是的。按照上面的說法， $\delta^l$ 做padding，卷積核參數矩陣最後兩維 ( $k \times k$ ) 做180°的翻轉，二者做步長為1 的捲積即可。

但是，本人注意到一個問題， $\delta^{Pad}$ 有很大一部分都是0，如果步長為2，則大約只有不到25 % ( $2 * 2 = 4$ ) 的計算是有效的（後續卷積很多都是0）；如果步長為3，則大約不到11% ( $3 * 3 = 9$ ) 的計算是有效的，這在計算和儲存上來說都是浪費！

拋棄嗎？

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。





輸出時，將輸出替換成 $\delta^l$ ，求輸出改成了求對輸入的梯度，可參考下面的實現，對forward 做了略微修改即可用

```
// delta_output 初始化为 0, 这一步不是多余的
for(int o = 0; o < batch_size; ++o) this->delta_output[o]->set_zero();
// 翻转 180, padding 那个太难了, 下面直接采用最笨的方法, 用卷积来定位每个输入对应的参与计算的
const int radius = (kernel_size - 1) / 2;
const int H_radius = H - radius;
const int W_radius = W - radius;
const int window_range = kernel_size * kernel_size;
// 多个 batch 的梯度分开算
for(int b = 0; b < batch_size; ++b) {
    // in_channels X 224 X 224
    data_type* const cur_image_features = this->delta_output[b]->data;
    // 16 个卷积核的输出, 16 x 111 x 111
    for(int o = 0; o < out_channels; ++o) { // 每个卷积核
        data_type* const out_ptr = delta[b]->data + o * out_length; // 第 o 个卷积核会得
        data_type* const cur_w_ptr = this->weights[o]->data; // in_channels x 3 x 3
        int cnt = 0; // 记录每次卷积结果存放的位置
        for(int x = radius; x < H_radius; x += stride) {
            for(int y = radius; y < W_radius; y += stride) { // 遍历图像平面每一个点
                data_type sum_value = 0.f;
                const int coord = x * W + y; // 当前点对于这个通道的特征图的位移
                for(int i = 0; i < in_channels; ++i) { // 每个点有多个通道
                    const int start = i * length + coord; // 输入的第 i 张特征图在 (x, y)
                    const int start_w = i * window_range; // 第 o 个卷积核的第 i 个通道
                    for(int k = 0; k < window_range; ++k) { // 遍历局部窗口
                        // sum_value += cur_image_features[start + offset[k]] * cur_w_ptr[start_w + k]
                        cur_image_features[start + offset[k]] += cur_w_ptr[start_w + k]
                    }
                }
                ++cnt; // 用来定位当前是输出的第几个数
            }
        }
    }
}
// 返回
return this->delta_output;
```

最重要的是如下兩行

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



第一行是正常forward 計算輸出；

第二行，輸入的每個位置（start + offset[k]）的值，和哪些參數（cur\_w\_ptr[start\_w + k]）做了運算，得到了什麼輸出，反向傳播時從輸出回傳了梯度（out\_ptr[cnt]），對應參數和梯度做相乘，相乘結果累加在start + offset[k] 這個位置上的梯度（cur\_image\_features）。

這樣會錯嗎？

實際我一開始這麼寫的時候，心裡也是很忐忑的。還好，最後訓練跑起來的時候，看到不斷下降的損失，我應該是寫對了！老天保佑！

這麼寫，沒有之前padding 導致的無意義計算，不多不少！

雖然借助forward 來做backward 對了，但需要注意兩點，

- forward 不是萬能的，在這裡卷積層的反向傳播可以，只是因為計算上不存在前後依賴的情況，這種情況在計算圖上可以體現出來，後面實現batch norm 再討論。
- 計算好了 $W, b$ 的梯度，對 $W, b$ 的權重更新必須放在求 $\delta^{l-1}$ 後面，因為參與計算的是更新之前的 $W, b$ ，而不是更新之後的 $W, b$ 。

## 梯度更新

至此，卷積層求好了梯度，對應Pytorch 中的.backward() 運算。

由於卷積層存在自己的可學習參數，和之前ReLU、MaxPool 不一樣，梯度下降做優化需要更新這些參數，對應Pytorch 中的optimizer.step()，暫時不考慮動量、Adam 等優化器，目前先使用SGD 優化器方法，如下：

$$\omega = \omega - lr \cdot \delta$$

其中 $\omega$ 是每個參數， $lr$ 是學習率， $\delta$ 是這個參數所得到的梯度，具體實作比較簡單，如下

```
// 更新参数
void Conv2D::update_gradients(const data_type learning_rate=1e-3) {
    // 必须确保之前计算过梯度
    assert(not this->weights_gradients.empty());
    // 把梯度更新到 w 和 b
    for(int o = 0; o < out_channels; ++o) {
```

向前

落後

和的梯度

向前傳播一層的梯度

收起

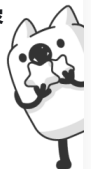
梯度更新

程式碼

參考

登入即可查看超5億專業優質內容


超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



```
data_type* wg_ptr = weights_gradients[o]->data;
// 逐個通道做權值更新
for(int i = 0; i < params_for_one_kernel; ++i)
    w_ptr[i] -= learning_rate * wg_ptr[i];
// 更新 bias
bias[o] -= learning_rate * bias_gradients[o];
}
}
```

如果考慮動量優化器，則還需要保存上一次的梯度信息，存儲消耗要更大，日後再說。

程式碼

<https://github.com/hermosayhl/CNN>  
號  github.com/hermosayhl/CNN

參考

1. [卷積神經網路\(CNN\)反向傳播演算法- 劉建平Pinard - 部落格園](#)

編輯於2022-02-19 18:35

卷積神經網路 (CNN)   C++   反向傳播演算法

號  贊同15   號    號  7 則評論   號  分享   號  喜歡   號  收藏   號  申請轉載   號  ...

寫下你的評論...

7 則評論

預設   最新



亦冰

大佬，當卷積核大小是偶數時，好像會有問題，請問如何修改程式碼。多謝

2023-10-09

號  回覆   號  喜歡



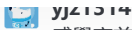
山與水你和我 作者

你好，我這個程式碼只支援邊長為奇數的捲積核🙄，不好意思。我這個計畫已經遺棄了，如果你有興趣，請找更好的資料吧🙄，謝謝

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。





yj41514  
感覺寫前傳的推理框架已經夠難了，反向也寫，太硬核了大佬😂

2023-04-20

號 ● 回覆 號 ♥ 喜歡



9iM ▸ 山與水你和我

大佬，請問有了解的其他關於訓練的項目嗎？

2023-10-09

號 ● 回覆 號 ♥ 1



山與水你和我 [作者] ▸ 9iM

你好，訓練的項目我還真不知道😂，我其實也很少接觸這些。訓練的話，可以看看oneflow？還有挖一挖😂NVIDIA的倉庫

2023-10-09

號 ● 回覆 號 ♥ 喜歡

展開其他2 則回復號 >

文章被以下專欄收錄



電腦視覺基礎

最簡單的卷積神經網路、ReLU、BN、目標偵測等

推薦閱讀



【綜述】神經網路中不同類型的捲積層

PPRP

卷積神經網路結構-卷積層

卷積層的作用是透過卷積操作抽象出影像特徵訊息，卷積層通常包含多個卷積核，每個卷積核都對應一個特徵映射。卷積層的參數包括卷積核滑動的步長和卷積核的大小。在特徵圖進行卷積過程中...

AI高級人... 發表於深度人臉識...

卷積神經網路學習筆記

1.0引例在前面的文章中我們講了神經網路的作用，而在本篇文章中我們要講卷積神經網路，通常我們對一個事物的了解是從名詞開始的，也有人說知識的詛咒，其實本質上就是連名詞都不理解，更別...

船長



別怕，"卷積

圖靈的貓

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。

