卷積神經網路（四）池化層

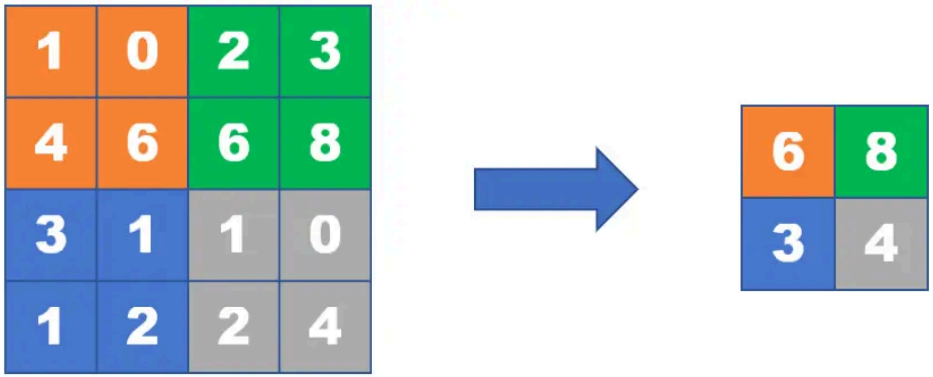
山與水你和我
體系結構小白，影像處理小白

上一篇

山與水你與我：卷積神經網路（三）ReLU 層

ReLU 層的前向反向傳播，接下來寫下同樣簡單的池化層。

以MaxPool2d 為例，如下圖某通道，步長2，核大小2 x 2 的最大池化過程，



步長2，核大小2 x 2

forward 前向過程，每個大小為2×2的局部窗口，選取最大值作為輸出。

由於參與了下一層運算的是6, 8, 3, 4 這4 個數，因此回傳給上一層的梯度只有6，8，3，4對應的四個位置有梯度，其餘地方梯度為0，如下圖

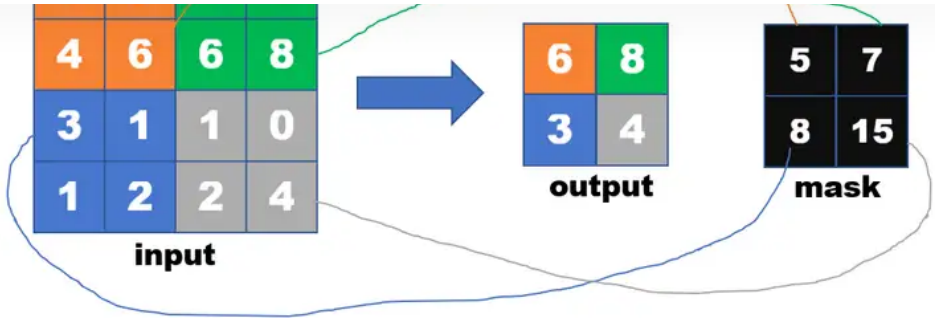
梯度回傳

上面只是舉例$H \times W$的最大池化過程，實際forward 接收輸入為$B \times C \times H \times W$，不同影像之間互相獨立做池化；同一個影像，不同通道之間也是相互獨立做池化。

向前

和上一次ReLU 層類似，如果後面需要backward，forward 需要記錄一些中間變數用於反向傳播等，因此有以下幾個主要變數需要存儲

- output, 作為forward 的輸出，雖然backward 用不到，但避免每次forward 重新分配空間，需要分配一個緩衝區；
- delta_output，回傳給上一層的梯度δ^{l-1}，和ReLU 層就地修改delta 不一樣，最大池化的δ^{l-1}, δ^l尺寸不一致，不能就地修改，同上分配一個緩衝區，將δ^{l-1}指標數組傳回給上一層
- mask，記錄哪些地方是有梯度的，第一個vector 數組定位這個batch 的第幾張圖的特徵，第二個vector 記錄這個特徵中所有「最大值」在input 中的位置，略微有點繞口，如下圖，mask 第2 個值7，記錄的是第二個最大值在Input 中的索引是7，對應Input 中的值為 8



面具

由上，MaxPool2d 類別聲明如下

```
class MaxPool2D : public Layer {
private:
    // 这一层的固有属性
    const int kernel_size; // 池化的局部窗口直径
    const int step;        // 池化的步长，一般和 kernel_size 相等
    // 缓冲区，避免每次重新分配的
    std::vector<tensor> output;  // 记录上一次输出
    std::vector<tensor> delta_output; // 返回的 delta
    std::vector< std::vector<int> > mask; // 记录哪些位置是有梯度回传的，第 b 张图，每张
    std::vector<int> offset;  // 偏移量指针
};
```

前向傳播過程的函數如下

```
std::vector<tensor> MaxPool2D::forward(const std::vector<tensor>& input) {
    // 获取输入信息
    const int batch_size = input.size();
    const int H = input[0]->H;
    const int W = input[0]->W;
    const int C = input[0]->C;
    // 计算输出的大小
    const int out_H = std::floor(((H - kernel_size) / step)) + 1;
    const int out_W = std::floor(((W - kernel_size) / step)) + 1;
    // 第一次经过该池化层
```

```
    // 给反向传播的 delta 分配空间
    if(not no_grad) {
        this->delta_output.reserve(batch_size);
        for(int b = 0;b < batch_size; ++b)this->delta_output.emplace_back(new Tens
        // mask 对 batch 的每一张图都分配空间
        this->mask.reserve(batch_size);
        for(int b = 0;b < batch_size; ++b) this->mask.emplace_back(std::vector<int
    }
    // 第一次经过这一层, 根据 kernel_size 计算 offset
    int pos = 0;
    for(int i = 0;i < kernel_size; ++i)
        for(int j = 0;j < kernel_size; ++j)
            this->offset[pos++] = i * W + j;
}
const int out_length = out_H * out_W;
int* mask_ptr = nullptr;
// 如果存在 backward, 每次 forward 要记得把 mask 全部填充为 0
if(not no_grad) {
    const int mask_size = C * out_length;
    for(int b = 0;b < batch_size; ++b) {
        int* const mask_ptr = this->mask[b].data();
        for(int i = 0;i < mask_size; ++i) mask_ptr[i] = 0;
    }
}
// 开始池化
const int length = H * W;
const int H_kernel = H - kernel_size;
const int W_kernel = W - kernel_size;
const int window_range = kernel_size * kernel_size; // 局部窗口要遍历的数据长度
for(int b = 0;b < batch_size; ++b) { // batch 的每一张图像对应的特征图分开池化
    // 16 X 111 X 111 → 16 X 55 X 55
    for(int i = 0;i < C; ++i) {  // 每个通道
        // 现在我拿到了第 b 张图的第 i 个通道, 一个指向内容大小 55 X 55 的指针
        data_type* const cur_image_features = input[b]->data + i * length;
        // 第 b 个输出的第 i 个通道的, 同样是指向内容大小 55 X 55 的指针
        data_type* const output_ptr = this->output[b]->data + i * out_length;
        // 记录第 b 个输出, 记录有效点在 111 X 111 这个图上的位置, 一共有 55 X 55 个值
        if(not no_grad) mask_ptr = this->mask[b].data() + i * out_length;
        int cnt = 0;  // 当前池化输出的位置
        for(int x = 0; x <= H_kernel; x += step) {
            data_type* const row_ptr = cur_image_features + x * W; // 获取这个通道
```

同意 8

```
                    for(int k = 1; k < window_range; ++k) { // 从 1 开始因为 0 已经比过
                        data_type comp = row_ptr[y + offset[k]];
                        if(max_value < comp) {
                            max_value = comp;
                            max_index = offset[k];
                        }
                    }
                    // 局部最大值填到输出的对应位置上
                    output_ptr[cnt] = max_value;
                    // 如果后面要 backward, 记录 mask
                    if(not no_grad) {
                        max_index += x * W + y; // 第 i 个通道, i * out_H * out_W 为起点
                        mask_ptr[cnt] = i * length + max_index;
                    }
                    ++cnt;
                }
            }
        }
    }
    return this->output;
}
```


若干細節

- 最大池化的輸出大小如何計算？公式以高為例 $\frac{H-kernel\_size+2*padding}{stride}+1$，這個公式隨便畫畫圖就可以證明是可行的，但由於本次實驗不考慮padding，預設都是0；
- 這裡有一個offset 變量，用來計算偏移量。局部視窗假設是4x4，做池化，每個視窗如果做兩次 for 迴圈稍微會慢點，如果記錄這個局部視窗距離左上角的偏移量，一個for 迴圈即可完成遍歷；偏移量的計算$i*W+j$，表示從左上角往下數i 行的第j 個資料的偏移量；
- 如果需要backward，每次forward 都要將mask 置為0，避免上一次的mask 結果影響之後的計算；
- 池化的遍歷方法，首先定位到$B\times C$，然後二維特徵圖找到每一個局部窗口，按照步長stride 移動，窗口最大值放在輸出的「對應位置」上，這個「對應位置」是藉助一個變數cnt 不斷更新的。

落後

```
// 反向传播
std::vector<tensor> MaxPool2D::backward(std::vector<tensor>& delta) {
    // 获取输入的梯度信息
    const int batch_size = delta.size();
    // 回传给上一层的梯度先填 0
    for(int b = 0;b < batch_size; ++b) this->delta_output[b]->set_zero();
    // batch 每张图像，根据 mask 标记的位置，把 delta 中的值填到 delta_output 中去
    const int total_length = delta[0]->get_length();  // delta 和 mask 尺寸一致
    for(int b = 0;b < batch_size; ++b) {
        int* mask_ptr = this->mask[b].data();
        // 获取 delta 第 b 张输出传回来的梯度起始地址
        data_type* const src_ptr = delta[b]->data;
        // 获取返回到输入的梯度，第 b 张梯度的起始地址
        data_type* const res_ptr = this->delta_output[b]->data;
        for(int i = 0;i < total_length; ++i)
            res_ptr[mask_ptr[i]] = src_ptr[i]; // res_ptr 在有效位置 mask_ptr[i] 上填“
    }
    return this->delta_output;
}
```

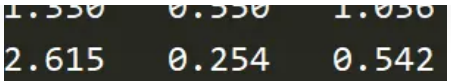
驗證

假設MaxPool2d 的輸入，在某張影像的第3 個頻道的內容如下，6×6

-0.170	0.137	-0.427	0.699	-0.469	-0.523
0.036	-0.434	0.423	1.875	0.413	-0.573
0.067	-1.014	0.550	0.133	0.880	1.036
1.330	-1.025	-0.112	-1.763	0.882	0.181
-0.367	2.615	0.254	-1.258	-0.470	0.542
0.708	0.814	-0.118	0.020	0.535	-0.233

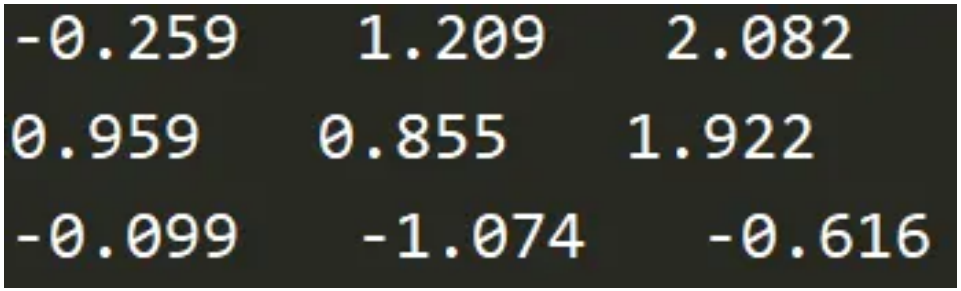
MaxPool 的輸入

經過步長為2，視窗大小2 的最大池化後，得到輸出3×3



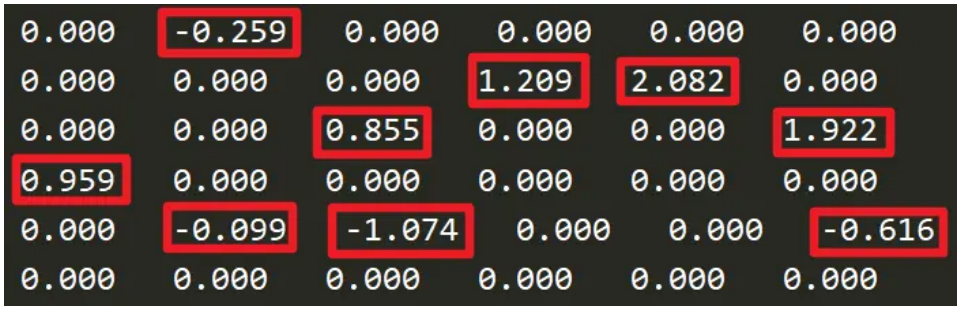
MaxPool 的輸出

backward 時，從下一層返回的梯度3×3



從下一層返回的梯度

得到返回給上一層的梯度6×6



返回給上一層的梯度

正確！

即使換成7×7的輸入，視窗大小3，步長2，依舊正確。

程式碼

卷積神經網路（CNN） 深度學習（Deep Learning） C++

同意 8

寫下你的評論...

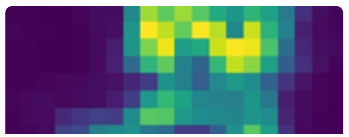
還沒有評論，發表第一個評論吧

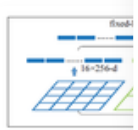
文章被以下專欄收錄

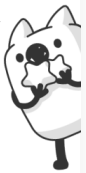
電腦視覺基礎
最簡單的捲積神經網路、ReLU、BN、目標偵測等

推薦閱讀

深度學習入門-卷積神經網路（二）池化層







嚴忻愷 無雙譜 人類之奴 劉冬煜 發表於人工智慧圖...

空間大小。一般來說，池化的視窗

同意 8