

## python教程网

- [网站首页](#)
- [python教程](#)
- [python web](#)
- [python爬虫](#)
- [python运维](#)
- [数据分析](#)
- [人工智能](#)
- [少儿编程](#)

## python教程网

一个靠谱的学习python的教程网,小白学python平台

- [首页](#)
- [python教程](#)
- [python web](#)
- [python爬虫](#)
- [python运维](#)
- [数据分析](#)
- [人工智能](#)
- [少儿编程](#)

当前位置: [首页](#) » [python教程](#) » 正文

# PyTorch的自适应池化Adaptive Pooling实例

645 次 2020-12-17 分类: [python教程](#)

### 简介

自适应池化Adaptive Pooling是PyTorch含有的一种池化层, 在PyTorch的中有六种形式:

#### 自适应最大池化Adaptive Max Pooling:

```
torch.nn.AdaptiveMaxPool1d(output_size)
torch.nn.AdaptiveMaxPool2d(output_size)
torch.nn.AdaptiveMaxPool3d(output_size)
```

#### 自适应平均池化Adaptive Average Pooling:

```
torch.nn.AdaptiveAvgPool1d(output_size)
torch.nn.AdaptiveAvgPool2d(output_size)
torch.nn.AdaptiveAvgPool3d(output_size)
```

具体可见[官方文档](#)。

官方给出的例子:

```
>>> # target output size of 5x7
>>> m = nn.AdaptiveMaxPool2d((5, 7))
>>> input = torch.randn(1, 64, 8, 9)
>>> output = m(input)
>>> output.size()
torch.Size([1, 64, 5, 7])

>>> # target output size of 7x7 (square)
>>> m = nn.AdaptiveMaxPool2d(7)
>>> input = torch.randn(1, 64, 10, 9)
>>> output = m(input)
>>> output.size()
torch.Size([1, 64, 7, 7])

>>> # target output size of 10x7
```

```
>>> m = nn.AdaptiveMaxPool2d((None, 7))
>>> input = torch.randn(1, 64, 10, 9)
>>> output = m(input)
>>> output.size()
torch.Size([1, 64, 10, 7])
```

Adaptive Pooling特殊性在于，输出张量的大小都是给定的output\_size。例如输入张量大小为(1, 64, 8, 9)，设定输出大小为(5,7)，通过Adaptive Pooling层，可以得到大小为(1, 64, 5, 7)的张量。

## 原理

若已知池化层的 $kernel\_size$ 、 $padding$ 、 $stride$ 以及输入张量的大小 $input\_size$ ，则输出张量大小 $output\_size$ 为：

$$output\_size = (input\_size + 2 * padding - kernel\_size) / stride + 1$$

则根据上式可得：

$$kernel\_size = (input\_size + 2 * padding) - (output\_size - 1) * stride$$

要想知道Adaptive Pooling的原理，我们需要找到该层的 $kernel\_size$ 、 $padding$ 和 $stride$ 。

通过以下的例子，我们看一下这三个参数的关系：

```
>>> input_size = 9
>>> output_size = 4

>>> input = torch.randn(1, 1, input_size)
>>> input
tensor([[[[ 1.5695, -0.4357, 1.5179, 0.9639, -0.4226, 0.5312, -0.5689, 0.4945, 0.1421]]]])

>>> m1 = nn.AdaptiveMaxPool1d(output_size)
>>> m2 = nn.MaxPool1d(kernel_size=math.ceil(input_size / output_size), stride=math.floor(input_size / output_size), padding=0)
>>> output1 = m1(input)
>>> output2 = m2(input)

>>> output1
tensor([[[[ 1.5695, 1.5179, 0.5312, 0.4945]]]]) torch.Size([1, 1, 4])
>>> output2
tensor([[[[ 1.5695, 1.5179, 0.5312, 0.4945]]]]) torch.Size([1, 1, 4])
```

通过实验发现：

$$stride = \text{floor}(input\_size / output\_size)$$

$$kernel\_size = input\_size - (output\_size - 1) * stride$$

$$padding = 0$$

$\text{ceil}$ 为向上取整， $\text{floor}$ 为向下取整。

下面是Adaptive Average Pooling的C++源码部分。

```
template <typename scalar_t>
static void adaptive_avg_pool2d_out_frame(
    scalar_t *input_p,
    scalar_t *output_p,
    int64_t sized,
    int64_t isizeH,
    int64_t isizeW,
    int64_t osizeH,
    int64_t osizeW,
    int64_t istrideD,
    int64_t istrideH,
    int64_t istrideW)
{
    int64_t d;
    #pragma omp parallel for private(d)
```

```
for (d = 0; d < sizeD; d++)
{
    /* loop over output */
    int64_t oh, ow;
    for(oh = 0; oh < osizeH; oh++)
    {
        int istartH = start_index(oh, osizeH, isizeH);
        int iendH = end_index(oh, osizeH, isizeH);
        int kH = iendH - istartH;

        for(ow = 0; ow < osizeW; ow++)
        {
            int istartW = start_index(ow, osizeW, isizeW);
            int iendW = end_index(ow, osizeW, isizeW);
            int kW = iendW - istartW;

            /* local pointers */
            scalar_t *ip = input_p + d*istrideD + istartH*istrideH + istartW*istrideW;
            scalar_t *op = output_p + d*osizeH*osizeW + oh*osizeW + ow;

            /* compute local average: */
            scalar_t sum = 0;
            int ih, iw;
            for(ih = 0; ih < kH; ih++)
            {
                for(iw = 0; iw < kW; iw++)
                {
                    scalar_t val = *(ip + ih*istrideH + iw*istrideW);
                    sum += val;
                }
            }

            /* set output to local average */
            *op = sum / kW / kH;
        }
    }
}
```

以上这篇PyTorch的自适应池化Adaptive Pooling实例就是小编分享给大家的全部内容了，希望能给大家一个参考，也希望大家多多支持python博客。

来源： [python博客](#) 欢迎分享！


本文链接： <https://www.94e.cn/info/4989>

标签：

[<< 上一篇](#) [下一篇 >>](#)

## ● 相关文章

- 2021-12-31 15:10:35[Python项目实战教程 前言](#)
- 2021-12-20 16:16:40[Python 实现图片色彩转换案例](#)
- 2021-12-20 16:16:38[python初学定义函数](#)
- 2021-12-20 16:16:37[图文详解Python如何导入自己编写的py文件](#)
- 2021-12-20 16:16:36[python二分法查找实例代码](#)
- 2021-12-20 16:16:35[Pyinstaller打包工具的使用以及避坑](#)
- 2021-12-20 16:16:34[Facebook开源一站式服务python时序利器Kats详解](#)
- 2021-12-20 16:16:31[pyCaret效率倍增开源低代码的python机器学习工具](#)
- 2021-12-20 16:16:30[python机器学习使数据更鲜活的可视化工具Pandas\\_Alive](#)
- 2021-12-20 16:16:28[python读写文件with open的介绍](#)

 赞助本站

## 最新python教程

---

- [\[12/31\]Python项目实战教程 前言](#)
- [\[12/20\]Python 实现图片色彩转换案例](#)
- [\[12/20\]python初学定义函数](#)
- [\[12/20\]图文详解Python如何导入自己编写的py文件](#)
- [\[12/20\]python二分法查找实例代码](#)