

## Bing

Sure, I can give you an example of how the adaptive average pooling reduces the size of the input tensor  $x$ . Adaptive average pooling is a type of pooling operation that applies an average pooling over an input signal with a variable output size. The output size is specified by the user, and the pooling kernel and stride are automatically adjusted to fit the input size. This way, the output size is always consistent regardless of the input size.

For example, let's say the input tensor  $x$  has a shape of  $(1, 3, 4, 4)$ , which means it has one batch, three channels, and four units in each spatial dimension. The input tensor  $x$  can be visualized as a  $3 \times 4 \times 4$  cube, where each cell represents a feature value. Here is an example of the input tensor  $x$ :

```
x = torch.tensor([[[[ 0.1, 0.2, 0.3, 0.4], [ 0.5, 0.6, 0.7, 0.8], [ 0.9, 1.0, 1.1, 1.2], [ 1.3, 1.4, 1.5, 1.6]],
[[ 1.7, 1.8, 1.9, 2.0],
[ 2.1, 2.2, 2.3, 2.4],
[ 2.5, 2.6, 2.7, 2.8],
[ 2.9, 3.0, 3.1, 3.2]],
[[ 3.3, 3.4, 3.5, 3.6],
[ 3.7, 3.8, 3.9, 4.0],
[ 4.1, 4.2, 4.3, 4.4],
[ 4.5, 4.6, 4.7, 4.8]]]])
```

Now, let's say we want to apply an adaptive average pooling on the input tensor  $x$ , and reduce the spatial dimensions to  $1 \times 1 \times 1$ . We can use the `nn.AdaptiveAvgPool3d` function from PyTorch to do this. We need to specify the output size as  $(1, 1, 1)$ , and the function will automatically calculate the pooling kernel and stride to fit the input size. Here is an example of how to use the function:

```
m = nn.AdaptiveAvgPool3d((1, 1, 1)) output = m(x)
```

The output tensor will have a shape of  $(1, 3, 1, 1, 1)$ , which means it has one batch, three channels, and one unit in each spatial dimension. The output tensor can be visualized as a  $3 \times 1 \times 1 \times 1$  cube, where each cell represents a feature value. Here is an example of the output tensor:

```
output = torch.tensor([[[[ 0.85]],
[[ 2.55]],
[[ 4.25]]]])
```

The output tensor is obtained by applying an average pooling over each channel of the input tensor. For example, the first cell of the output tensor, 0.85, is the average of all the values in the first channel of the input tensor, which are 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, and 1.6. Similarly, the second cell of the output tensor, 2.55, is the average of all the values in the second channel of the input tensor, and so on.

By applying the adaptive average pooling, we have reduced the size of the input tensor from  $(1, 3, 4, 4)$  to  $(1, 3, 1, 1, 1)$ . This can save some computation and memory resources, and also capture the global context information of the input. However, this also means that we have lost some spatial details and resolution of the input, which may affect the performance of downstream tasks, such as segmentation or detection.

I hope this example helps you understand how the adaptive average pooling reduces the size of the input tensor  $x$ . 😊

: [AdaptiveAvgPool3d — PyTorch 2.2 documentation](#)