# tf.space_to_depth

```
space_to_depth(
    input,
    block_size,
    name=None,
    data_format='NHWC'
)
```

Defined in `tensorflow/python/ops/array_ops.py`.

See the guide: Tensor Transformations > Slicing and Joining

SpaceToDepth for tensors of type T.

Rearranges blocks of spatial data, into depth. More specifically, this op outputs a copy of the input tensor where values from the `height` and `width` dimensions are moved to the `depth` dimension. The attr `block_size` indicates the input block size.

- Non-overlapping blocks of size `block_size x block size` are rearranged into depth at each location.
- The depth of the output tensor is `block_size * block_size * input_depth`.
- The Y, X coordinates within each block of the input become the high order component of the output channel index.
- The input tensor's height and width must be divisible by block_size.

The `data_format` attr specifies the layout of the input and output tensors with the following options: "NHWC": `[ batch, height, width, channels ]` "NCHW": `[ batch, channels, height, width ]` "NCHW_VECT_C": `qint8 [ batch, channels / 4, height, width, channels % 4 ]`

It is useful to consider the operation as transforming a 6-D Tensor. e.g. for data_format = NHWC, Each element in the input tensor can be specified via 6 coordinates, ordered by decreasing memory layout significance as: n,oY,bY,oX,bX,iC (where n=batch index, oX, oY means X or Y coordinates within the output image, bX, bY means coordinates within the input block, iC means input channels). The output would be a transpose to the following layout: n,oY,oX,bY,bX,iC

This operation is useful for resizing the activations between convolutions (but keeping all data), e.g. instead of pooling. It is also useful for training purely convolutional models.

For example, given an input of shape `[1, 2, 2, 1]`, data_format = "NHWC" and block_size = 2:

```
x = [[[[1], [2]],
      [[3], [4]]]]
```

This operation will output a tensor of shape `[1, 1, 1, 4]`:

```
[[[[1, 2, 3, 4]]]]
```

Here, the input has a batch of 1 and each batch element has shape `[2, 2, 1]`, the corresponding output will have a single element (i.e. width and height are both 1) and will have a depth of 4 channels (1 * block_size * block_size). The output element shape is `[1, 1, 4]`.

For an input tensor with larger depth, here of shape `[1, 2, 2, 3]`, e.g.

```
x = [[[[1, 2, 3], [4, 5, 6]],
      [[7, 8, 9], [10, 11, 12]]]]
```

This operation, for block_size of 2, will return the following tensor of shape `[1, 1, 1, 12]`

```
[[[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]]]]
```

Similarly, for the following input of shape `[1 4 4 1]`, and a block size of 2:

```
x = [[[[1],   [2],  [5],  [6]],
      [[3],   [4],  [7],  [8]],
      [[9],  [10], [13],  [14]],
      [[11], [12], [15],  [16]]]]
```

the operator will return the following tensor of shape `[1 2 2 4]`:

```
x = [[[[1, 2, 3, 4],
       [5, 6, 7, 8]],
      [[9, 10, 11, 12],
       [13, 14, 15, 16]]]]
```

## Args:

- `input`: A `Tensor`.
- `block_size`: An `int` that is `>= 2`. The size of the spatial block.
- `data_format`: An optional `string` from: `"NHWC", "NCHW", "NCHW_VECT_C"`. Defaults to `"NHWC"`.
- `name`: A name for the operation (optional).

## Returns:

A `Tensor`. Has the same type as `input`.

---

*Last updated November 2, 2017.*