tf.space_to_batch

```
space_to_batch(
    input,
    paddings,
    block_size,
    name=None
)
```

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

See the guide: Tensor Transformations > Slicing and Joining

SpaceToBatch for 4-D tensors of type T.

This is a legacy version of the more general SpaceToBatchND.

Zero-pads and then rearranges (permutes) blocks of spatial data into batch. More specifically, this op outputs a copy of the input tensor where values from the `height` and `width` dimensions are moved to the `batch` dimension. After the zero-padding, both `height` and `width` of the input must be divisible by the block size.

Args:

- `input`: A `Tensor`. 4-D with shape `[batch, height, width, depth]`.
- `paddings`: A `Tensor`. Must be one of the following types: `int32`, `int64`. 2-D tensor of non-negative integers with shape `[2, 2]`. It specifies the padding of the input with zeros across the spatial dimensions as follows:

  ```
    paddings = [[pad_top, pad_bottom], [pad_left, pad_right]]
  ```

  The effective spatial dimensions of the zero-padded input tensor will be:

  ```
    height_pad = pad_top + height + pad_bottom
    width_pad = pad_left + width + pad_right
  ```

  The attr `block_size` must be greater than one. It indicates the block size.

  - Non-overlapping blocks of size `block_size x block size` in the height and width dimensions are rearranged into the batch dimension at each location.
  - The batch of the output tensor is `batch * block_size * block_size`.
  - Both height_pad and width_pad must be divisible by block_size.

  The shape of the output will be:

  ```
  [batch*block_size*block_size, height_pad/block_size, width_pad/block_size,
   depth]
  ```

  Some examples:

  (1) For the following input of shape `[1, 2, 2, 1]` and block_size of 2:

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

  The output tensor has shape `[4, 1, 1, 1]` and value:

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

(2) For the following input of shape `[1, 2, 2, 3]` and block_size of 2:

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

The output tensor has shape `[4, 1, 1, 3]` and value:

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

(3) For the following input of shape `[1, 4, 4, 1]` and block_size of 2:

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

The output tensor has shape `[4, 2, 2, 1]` and value:

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

(4) For the following input of shape `[2, 2, 4, 1]` and block_size of 2:

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

The output tensor has shape `[8, 1, 2, 1]` and value:

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

Among others, this operation is useful for reducing atrous convolution into regular convolution. `block_size`*: An* `int` *that is* `>= 2`. `name`: A name for the operation (optional).

Returns:

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

Last updated November 2, 2017.