tf.nn.with_space_to_batch

```
with_space_to_batch(
    input,
    dilation_rate,
    padding,
    op,
    filter_shape=None,
    spatial_dims=None,
    data_format=None
)
```

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

See the guide: Neural Network > Morphological filtering

Performs `op` on the space-to-batch representation of `input`.

This has the effect of transforming sliding window operations into the corresponding "atrous" operation in which the input is sampled at the specified `dilation_rate`.

In the special case that `dilation_rate` is uniformly 1, this simply returns:

op(input, num_spatial_dims, padding)

Otherwise, it returns:

batch_to_space_nd(op(space_to_batch_nd(input, adjusted_dilation_rate, adjusted_paddings), num_spatial_dims, "VALID") adjusted_dilation_rate, adjusted_crops),

where:

adjusted_dilation_rate is an int64 tensor of shape [max(spatial_dims)], adjusted_{paddings,crops} are int64 tensors of shape [max(spatial_dims), 2]

defined as follows:

We first define two int64 tensors `paddings` and `crops` of shape `[num_spatial_dims, 2]` based on the value of `padding` and the spatial dimensions of the `input`:

If `padding = "VALID"`, then:

paddings, crops = required_space_to_batch_paddings(input_shape[spatial_dims], dilation_rate)

If `padding = "SAME"`, then:

dilated_filter_shape = filter_shape + (filter_shape - 1) * (dilation_rate - 1)

paddings, crops = required_space_to_batch_paddings(input_shape[spatial_dims], dilation_rate, [(dilated_filter_shape - 1) // 2, dilated_filter_shape - 1 - (dilated_filter_shape - 1) // 2])

Because `space_to_batch_nd` and `batch_to_space_nd` assume that the spatial dimensions are contiguous starting at the second dimension, but the specified `spatial_dims` may not be, we must adjust `dilation_rate`, `paddings` and `crops` in order to be usable with these operations. For a given dimension, if the block size is 1, and both the starting and ending padding and crop amounts are 0, then space_to_batch_nd effectively leaves that dimension alone, which is what is needed for dimensions not part of `spatial_dims`. Furthermore, `space_to_batch_nd` and `batch_to_space_nd` handle this case efficiently for any number of leading and trailing dimensions.

For 0 <= i < len(spatial_dims), we assign:

adjusted_dilation_rate[spatial_dims[i] - 1] = dilation_rate[i] adjusted_paddings[spatial_dims[i] - 1, :] = paddings[i, :] adjusted_crops[spatial_dims[i] - 1, :] = crops[i, :]

All unassigned values of `adjusted_dilation_rate` default to 1, while all unassigned values of `adjusted_paddings` and `adjusted_crops` default to 0.

Note in the case that `dilation_rate` is not uniformly 1, specifying "VALID" padding is equivalent to specifying `padding = "SAME"` with a filter_shape of `[1]*N`.

Advanced usage. Note the following optimization: A sequence of `with_space_to_batch` operations with identical (not uniformly 1) `dilation_rate` parameters and "VALID" padding

net = with_space_to_batch(net, dilation_rate, "VALID", op_1) ... net = with_space_to_batch(net, dilation_rate, "VALID", op_k)

can be combined into a single `with_space_to_batch` operation as follows:

def combined_op(converted_input, num_spatial_dims, _): result = op_1(converted_input, num_spatial_dims, "VALID") ... result = op_k(result, num_spatial_dims, "VALID")

net = with_space_to_batch(net, dilation_rate, "VALID", combined_op)

This eliminates the overhead of `k-1` calls to `space_to_batch_nd` and `batch_to_space_nd`.

Similarly, a sequence of `with_space_to_batch` operations with identical (not uniformly 1) `dilation_rate` parameters, "SAME" padding, and odd filter dimensions

net = with_space_to_batch(net, dilation_rate, "SAME", op_1, filter_shape_1) ... net = with_space_to_batch(net, dilation_rate, "SAME", op_k, filter_shape_k)

can be combined into a single `with_space_to_batch` operation as follows:

def combined_op(converted_input, num_spatial_dims, _): result = op_1(converted_input, num_spatial_dims, "SAME") ... result = op_k(result, num_spatial_dims, "SAME")

net = with_space_to_batch(net, dilation_rate, "VALID", combined_op)

Args:

- `input`: Tensor of rank > max(spatial_dims).
- `dilation_rate`: int32 Tensor of *known* shape [num_spatial_dims].
- `padding`: str constant equal to "VALID" or "SAME"
- `op`: Function that maps (input, num_spatial_dims, padding) -> output
- `filter_shape`: If padding = "SAME", specifies the shape of the convolution kernel/pooling window as an integer Tensor of shape [>=num_spatial_dims]. If padding = "VALID", filter_shape is ignored and need not be specified.
- `spatial_dims`: Monotonically increasing sequence of `num_spatial_dims` integers (which are >= 1) specifying the spatial dimensions of `input` and output. Defaults to: `range(1, num_spatial_dims+1)`.
- `data_format`: A string or None. Specifies whether the channel dimension of the `input` and output is the last dimension (default, or if `data_format` does not start with "NC"), or the second dimension (if `data_format` starts with "NC"). For N=1, the valid values are "NWC" (default) and "NCW". For N=2, the valid values are "NHWC" (default) and "NCHW". For N=3, the valid values are "NDHWC" (default) and "NCDHW".

Returns:

The output Tensor as described above, dimensions will vary based on the op provided.

Raises:

- `ValueError` : if `padding` is invalid or the arguments are incompatible.
- `ValueError` : if `spatial_dims` are invalid.

Last updated November 2, 2017.