tf.nn.atrous_conv2d

```
atrous_conv2d(
    value,
    filters,
    rate,
    padding,
    name=None
)
```

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

See the guide: Neural Network > Convolution

Atrous convolution (a.k.a. convolution with holes or dilated convolution).

This function is a simpler wrapper around the more general `tf.nn.convolution`, and exists only for backwards compatibility. You can use `tf.nn.convolution` to perform 1-D, 2-D, or 3-D atrous convolution.

Computes a 2-D atrous convolution, also known as convolution with holes or dilated convolution, given 4-D `value` and `filters` tensors. If the `rate` parameter is equal to one, it performs regular 2-D convolution. If the `rate` parameter is greater than one, it performs convolution with holes, sampling the input values every `rate` pixels in the `height` and `width` dimensions. This is equivalent to convolving the input with a set of upsampled filters, produced by inserting `rate - 1` zeros between two consecutive values of the filters along the `height` and `width` dimensions, hence the name atrous convolution or convolution with holes (the French word trous means holes in English).

More specifically:

```
output[batch, height, width, out_channel] =
    sum_{dheight, dwidth, in_channel} (
        filters[dheight, dwidth, in_channel, out_channel] *
        value[batch, height + rate*dheight, width + rate*dwidth, in_channel]
    )
```

Atrous convolution allows us to explicitly control how densely to compute feature responses in fully convolutional networks. Used in conjunction with bilinear interpolation, it offers an alternative to `conv2d_transpose` in dense prediction tasks such as semantic image segmentation, optical flow computation, or depth estimation. It also allows us to effectively enlarge the field of view of filters without increasing the number of parameters or the amount of computation.

For a description of atrous convolution and how it can be used for dense feature extraction, please see: Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. The same operation is investigated further in Multi-Scale Context Aggregation by Dilated Convolutions. Previous works that effectively use atrous convolution in different ways are, among others, OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks and Fast Image Scanning with Deep Max-Pooling Convolutional Neural Networks. Atrous convolution is also closely related to the so-called noble identities in multi-rate signal processing.

There are many different ways to implement atrous convolution (see the refs above). The implementation here reduces

```
    atrous_conv2d(value, filters, rate, padding=padding)
```

to the following three operations:

```
    paddings = ...
    net = space_to_batch(value, paddings, block_size=rate)
    net = conv2d(net, filters, strides=[1, 1, 1, 1], padding="VALID")
    crops = ...
    net = batch_to_space(net, crops, block_size=rate)
```

Advanced usage. Note the following optimization: A sequence of `atrous_conv2d` operations with identical `rate` parameters, 'SAME' `padding`, and filters with odd heights/ widths:

```
    net = atrous_conv2d(net, filters1, rate, padding="SAME")
    net = atrous_conv2d(net, filters2, rate, padding="SAME")
    ...
    net = atrous_conv2d(net, filtersK, rate, padding="SAME")
```

can be equivalently performed cheaper in terms of computation and memory as:

```
    pad = ...  # padding so that the input dims are multiples of rate
    net = space_to_batch(net, paddings=pad, block_size=rate)
    net = conv2d(net, filters1, strides=[1, 1, 1, 1], padding="SAME")
    net = conv2d(net, filters2, strides=[1, 1, 1, 1], padding="SAME")
    ...
    net = conv2d(net, filtersK, strides=[1, 1, 1, 1], padding="SAME")
    net = batch_to_space(net, crops=pad, block_size=rate)
```

because a pair of consecutive `space_to_batch` and `batch_to_space` ops with the same `block_size` cancel out when their respective `paddings` and `crops` inputs are identical.

Args:

- `value`: A 4-D `Tensor` of type `float`. It needs to be in the default "NHWC" format. Its shape is `[batch, in_height, in_width, in_channels]`.
- `filters`: A 4-D `Tensor` with the same type as `value` and shape `[filter_height, filter_width, in_channels, out_channels]`. `filters`' `in_channels` dimension must match that of `value`. Atrous convolution is equivalent to standard convolution with upsampled filters with effective height `filter_height + (filter_height - 1) * (rate - 1)` and effective width `filter_width + (filter_width - 1) * (rate - 1)`, produced by inserting `rate - 1` zeros along consecutive elements across the `filters`' spatial dimensions.
- `rate`: A positive int32. The stride with which we sample input values across the `height` and `width` dimensions. Equivalently, the rate by which we upsample the filter values by inserting zeros across the `height` and `width` dimensions. In the literature, the same parameter is sometimes called `input stride` or `dilation`.
- `padding`: A string, either `'VALID'` or `'SAME'`. The padding algorithm.
- `name`: Optional name for the returned tensor.

Returns:

A `Tensor` with the same type as `value`. Output shape with ``'VALID`` padding is:

```
[batch, height - 2 * (filter_width - 1),
 width - 2 * (filter_height - 1), out_channels].
```

Output shape with `'SAME'` padding is:

```
[batch, height, width, out_channels].
```

Raises:

- `ValueError`: If input/output depth does not match `filters`' shape, or if padding is other than `'VALID'` or `'SAME'`.

Last updated November 2, 2017.