Convolution1D

[source]

```
keras.layers.convolutional.Convolution1D(nb_filter, filter_length, init='uniform', activation='
```

Convolution operator for filtering neighborhoods of one-dimensional inputs. When using this layer as the first layer in a model, either provide the keyword argument `input_dim` (int, e.g. 128 for sequences of 128-dimensional vectors), or `input_shape` (tuple of integers, e.g. (10, 128) for sequences of 10 vectors of 128-dimensional vectors).

Example

```
# apply a convolution 1d of length 3 to a sequence with 10 timesteps,
# with 64 output filters
model = Sequential()
model.add(Convolution1D(64, 3, border_mode='same', input_shape=(10, 32)))
# now model.output_shape == (None, 10, 64)

# add a new conv1d on top
model.add(Convolution1D(32, 3, border_mode='same'))
# now model.output_shape == (None, 10, 32)
```

Arguments

- **nb_filter**: Number of convolution kernels to use (dimensionality of the output).
- **filter_length**: The extension (spatial or temporal) of each filter.
- **init**: name of initialization function for the weights of the layer (see initializations), or alternatively, Theano function to use for weights initialization. This parameter is only relevant if you don't pass a `weights` argument.
- **activation**: name of activation function to use (see activations), or alternatively, elementwise Theano function. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
- **weights**: list of numpy arrays to set as initial weights.
- **border_mode**: 'valid' or 'same'.
- **subsample_length**: factor by which to subsample output.
- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the main weights matrix.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **activity_regularizer**: instance of ActivityRegularizer, applied to the network output.
- **W_constraint**: instance of the constraints module (eg. maxnorm, nonneg), applied to the main weights matrix.
- **b_constraint**: instance of the constraints module, applied to the bias.

- **bias**: whether to include a bias (i.e. make the layer affine rather than linear).
- **input_dim**: Number of channels/dimensions in the input. Either this argument or the keyword argument `input_shape` must be provided when using this layer as the first layer in a model.
- **input_length**: Length of input sequences, when it is constant. This argument is required if you are going to connect `Flatten` then `Dense` layers upstream (without it, the shape of the dense outputs cannot be computed).

Input shape

3D tensor with shape: `(samples, steps, input_dim)`.

Output shape

3D tensor with shape: `(samples, new_steps, nb_filter)`. `steps` value might have changed due to padding.

Convolution2D

[source]

```
keras.layers.convolutional.Convolution2D(nb_filter, nb_row, nb_col, init='glorot_uniform', acti
```

Convolution operator for filtering windows of two-dimensional inputs. When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the sample axis), e.g. `input_shape=(3, 128, 128)` for 128x128 RGB pictures.

Examples

```
# apply a 3x3 convolution with 64 output filters on a 256x256 image:
model = Sequential()
model.add(Convolution2D(64, 3, 3, border_mode='same', input_shape=(3, 256, 256)))
# now model.output_shape == (None, 64, 256, 256)

# add a 3x3 convolution on top, with 32 output filters:
model.add(Convolution2D(32, 3, 3, border_mode='same'))
# now model.output_shape == (None, 32, 256, 256)
```

Arguments

- **nb_filter**: Number of convolution filters to use.
- **nb_row**: Number of rows in the convolution kernel.
- **nb_col**: Number of columns in the convolution kernel.
- **init**: name of initialization function for the weights of the layer (see initializations), or alternatively, Theano function to use for weights initialization. This parameter is only relevant if you don't pass a `weights` argument.

- **activation**: name of activation function to use (see activations), or alternatively, elementwise Theano function. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
- **weights**: list of numpy arrays to set as initial weights.
- **border_mode**: 'valid' or 'same'.
- **subsample**: tuple of length 2. Factor by which to subsample output. Also called strides elsewhere.
- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the main weights matrix.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **activity_regularizer**: instance of ActivityRegularizer, applied to the network output.
- **W_constraint**: instance of the constraints module (eg. maxnorm, nonneg), applied to the main weights matrix.
- **b_constraint**: instance of the constraints module, applied to the bias.
- **dim_ordering**: 'th' or 'tf'. In 'th' mode, the channels dimension (the depth) is at index 1, in 'tf' mode is it at index 3. It defaults to the `image_dim_ordering` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "th".
- **bias**: whether to include a bias (i.e. make the layer affine rather than linear).

Input shape

4D tensor with shape: `(samples, channels, rows, cols)` if dim_ordering='th' or 4D tensor with shape: `(samples, rows, cols, channels)` if dim_ordering='tf'.

Output shape

4D tensor with shape: `(samples, nb_filter, new_rows, new_cols)` if dim_ordering='th' or 4D tensor with shape: `(samples, new_rows, new_cols, nb_filter)` if dim_ordering='tf'. `rows` and `cols` values might have changed due to padding.

AtrousConvolution2D

[source]

```
keras.layers.convolutional.AtrousConvolution2D(nb_filter, nb_row, nb_col, init='glorot_uniform'
```

Atrous Convolution operator for filtering windows of two-dimensional inputs. A.k.a dilated convolution or convolution with holes. When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the sample axis), e.g. `input_shape=(3, 128, 128)` for 128x128 RGB pictures.

Examples

```
# apply a 3x3 convolution with atrous rate 2x2 and 64 output filters on a 256x256 image:
model = Sequential()
model.add(AtrousConvolution2D(64, 3, 3, atrous_rate=(2,2), border_mode='valid', input_shape=(3,
# now the actual kernel size is dilated from 3x3 to 5x5 (3+(3-1)*(2-1)=5)
# thus model.output_shape == (None, 64, 252, 252)
```

Arguments

- **nb_filter**: Number of convolution filters to use.
- **nb_row**: Number of rows in the convolution kernel.
- **nb_col**: Number of columns in the convolution kernel.
- **init**: name of initialization function for the weights of the layer (see initializations), or alternatively, Theano function to use for weights initialization. This parameter is only relevant if you don't pass a `weights` argument.
- **activation**: name of activation function to use (see activations), or alternatively, elementwise Theano function. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
- **weights**: list of numpy arrays to set as initial weights.
- **border_mode**: 'valid' or 'same'.
- **subsample**: tuple of length 2. Factor by which to subsample output. Also called strides elsewhere.
- **atrous_rate**: tuple of length 2. Factor for kernel dilation. Also called filter_dilation elsewhere.
- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the main weights matrix.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **activity_regularizer**: instance of ActivityRegularizer, applied to the network output.
- **W_constraint**: instance of the constraints module (eg. maxnorm, nonneg), applied to the main weights matrix.
- **b_constraint**: instance of the constraints module, applied to the bias.
- **dim_ordering**: 'th' or 'tf'. In 'th' mode, the channels dimension (the depth) is at index 1, in 'tf' mode is it at index 3. It defaults to the `image_dim_ordering` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "th".
- **bias**: whether to include a bias (i.e. make the layer affine rather than linear).

Input shape

4D tensor with shape: `(samples, channels, rows, cols)` if dim_ordering='th' or 4D tensor with shape: `(samples, rows, cols, channels)` if dim_ordering='tf'.

Output shape

4D tensor with shape: `(samples, nb_filter, new_rows, new_cols)` if dim_ordering='th' or 4D tensor with shape: `(samples, new_rows, new_cols, nb_filter)` if dim_ordering='tf'. `rows` and `cols` values might have changed due to padding.

References

- Multi-Scale Context Aggregation by Dilated Convolutions

Convolution3D

[source]

```
keras.layers.convolutional.Convolution3D(nb_filter, kernel_dim1, kernel_dim2, kernel_dim3, init
```

Convolution operator for filtering windows of three-dimensional inputs. When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the sample axis), e.g. `input_shape=(3, 10, 128, 128)` for 10 frames of 128x128 RGB pictures.

Arguments

- **nb_filter**: Number of convolution filters to use.
- **kernel_dim1**: Length of the first dimension in the convolution kernel.
- **kernel_dim2**: Length of the second dimension in the convolution kernel.
- **kernel_dim3**: Length of the third dimension in the convolution kernel.
- **init**: name of initialization function for the weights of the layer (see initializations), or alternatively, Theano function to use for weights initialization. This parameter is only relevant if you don't pass a `weights` argument.
- **activation**: name of activation function to use (see activations), or alternatively, elementwise Theano function. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
- **weights**: list of Numpy arrays to set as initial weights.
- **border_mode**: 'valid' or 'same'.
- **subsample**: tuple of length 3. Factor by which to subsample output. Also called strides elsewhere.
  - **Note**: 'subsample' is implemented by slicing the output of conv3d with strides=(1,1,1).
- **W_regularizer**: instance of WeightRegularizer (eg. L1 or L2 regularization), applied to the main weights matrix.
- **b_regularizer**: instance of WeightRegularizer, applied to the bias.
- **activity_regularizer**: instance of ActivityRegularizer, applied to the network output.
- **W_constraint**: instance of the constraints module (eg. maxnorm, nonneg), applied to the main weights matrix.
- **b_constraint**: instance of the constraints module, applied to the bias.
- **dim_ordering**: 'th' or 'tf'. In 'th' mode, the channels dimension (the depth) is at index 1, in 'tf' mode is it at index 4. It defaults to the `image_dim_ordering` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "th".
- **bias**: whether to include a bias (i.e. make the layer affine rather than linear).

Input shape

5D tensor with shape: `(samples, channels, conv_dim1, conv_dim2, conv_dim3)` if dim_ordering='th' or 5D tensor with shape: `(samples, conv_dim1, conv_dim2, conv_dim3, channels)` if dim_ordering='tf'.

Output shape

5D tensor with shape: `(samples, nb_filter, new_conv_dim1, new_conv_dim2, new_conv_dim3)` if dim_ordering='th' or 5D tensor with shape: `(samples, new_conv_dim1, new_conv_dim2, new_conv_dim3, nb_filter)` if dim_ordering='tf'. `new_conv_dim1`, `new_conv_dim2` and `new_conv_dim3` values might have changed due to padding.

UpSampling1D

[source]

```
keras.layers.convolutional.UpSampling1D(length=2)
```

Repeat each temporal step `length` times along the time axis.

Arguments

- **length**: integer. Upsampling factor.

Input shape

3D tensor with shape: `(samples, steps, features)`.

Output shape

3D tensor with shape: `(samples, upsampled_steps, features)`.

UpSampling2D

[source]

```
keras.layers.convolutional.UpSampling2D(size=(2, 2), dim_ordering='th')
```

Repeat the rows and columns of the data by size[0] and size[1] respectively.

Arguments

- **size**: tuple of 2 integers. The upsampling factors for rows and columns.

- **dim_ordering**: 'th' or 'tf'. In 'th' mode, the channels dimension (the depth) is at index 1, in 'tf' mode is it at index 3. It defaults to the `image_dim_ordering` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "th".

Input shape

4D tensor with shape: `(samples, channels, rows, cols)` if dim_ordering='th' or 4D tensor with shape: `(samples, rows, cols, channels)` if dim_ordering='tf'.

Output shape

4D tensor with shape: `(samples, channels, upsampled_rows, upsampled_cols)` if dim_ordering='th' or 4D tensor with shape: `(samples, upsampled_rows, upsampled_cols, channels)` if dim_ordering='tf'.

UpSampling3D

[source]

```
keras.layers.convolutional.UpSampling3D(size=(2, 2, 2), dim_ordering='th')
```

Repeat the first, second and third dimension of the data by size[0], size[1] and size[2] respectively.

Arguments

- **size**: tuple of 3 integers. The upsampling factors for dim1, dim2 and dim3.
- **dim_ordering**: 'th' or 'tf'. In 'th' mode, the channels dimension (the depth) is at index 1, in 'tf' mode is it at index 4. It defaults to the `image_dim_ordering` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "th".

Input shape

5D tensor with shape: `(samples, channels, dim1, dim2, dim3)` if dim_ordering='th' or 5D tensor with shape: `(samples, dim1, dim2, dim3, channels)` if dim_ordering='tf'.

Output shape

5D tensor with shape: `(samples, channels, upsampled_dim1, upsampled_dim2, upsampled_dim3)` if dim_ordering='th' or 5D tensor with shape: `(samples, upsampled_dim1, upsampled_dim2, upsampled_dim3, channels)` if dim_ordering='tf'.

ZeroPadding1D

[source]

```
keras.layers.convolutional.ZeroPadding1D(padding=1)
```

Zero-padding layer for 1D input (e.g. temporal sequence).

Arguments

- **padding**: int How many zeros to add at the beginning and end of the padding dimension (axis 1).

Input shape

3D tensor with shape (samples, axis_to_pad, features)

Output shape

3D tensor with shape (samples, padded_axis, features)

ZeroPadding2D

[source]

```
keras.layers.convolutional.ZeroPadding2D(padding=(1, 1), dim_ordering='th')
```

Zero-padding layer for 2D input (e.g. picture).

Arguments

- **padding**: tuple of int (length 2) How many zeros to add at the beginning and end of the 2 padding dimensions (axis 3 and 4).
- **dim_ordering**: 'th' or 'tf'. In 'th' mode, the channels dimension (the depth) is at index 1, in 'tf' mode is it at index 3. It defaults to the `image_dim_ordering` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "th".

Input shape

4D tensor with shape: (samples, depth, first_axis_to_pad, second_axis_to_pad)

Output shape

4D tensor with shape: (samples, depth, first_padded_axis, second_padded_axis)

ZeroPadding3D

[source]

```
keras.layers.convolutional.ZeroPadding3D(padding=(1, 1, 1), dim_ordering='th')
```

Zero-padding layer for 3D data (spatial or spatio-temporal).

Arguments

- **padding**: tuple of int (length 3) How many zeros to add at the beginning and end of the 3 padding dimensions (axis 3, 4 and 5).
- **dim_ordering**: 'th' or 'tf'. In 'th' mode, the channels dimension (the depth) is at index 1, in 'tf' mode is it at index 4. It defaults to the `image_dim_ordering` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "th".

Input shape

5D tensor with shape: (samples, depth, first_axis_to_pad, second_axis_to_pad, third_axis_to_pad)

Output shape

5D tensor with shape: (samples, depth, first_padded_axis, second_padded_axis, third_axis_to_pad)