Conv1D

[source]

```
keras.layers.Conv1D(filters, kernel_size, strides=1, padding='valid', dilation_rate=1, activation=None, us
```

1次元の畳み込みレイヤー（例えば時間的な畳込み）．

このレイヤーは畳み込みカーネルを生成します．これはレイヤーの入力を単一の空間的（または時間的）次元で畳み込んで，出力のテンソルを作ります． `use_bias` をTrueにすると，バイアスベクトルが出力に加えられます． `activation` が `None` でない場合，指定した活性化関数が出力に適用されます．

このレイヤーを第一層に使う場合，キーワード引数として `input_shape` （整数のタプルか `None` ．例えば10個の128次元ベクトルの場合ならば `(10, 128)` ，任意個数の128次元ベクトルの場合は `(None, 128)` ）を指定してください．

引数

- **filters**: 整数，出力空間の次元（つまり畳み込みにおける出力フィルタの数）．
- **kernel_size**: 整数か単一の整数からなるタプル/リストで，1次元の畳み込みウィンドウの長さを指定します．
- **strides**: 整数か単一の整数からなるタプル/リストで，畳み込みのストライドの長さを指定します. strides value != 1とすると `dilation_rate` value != 1と指定できません．
- **padding**: `"valid"` ， `"same"` ， `"causal"` のいずれか（大文字小文字の区別はしない）． `"valid"` はパディングを行いません． `"same"` は元の入力と同じ長さを出力がもつように入力にパディングします． `"causal"` はcausal（dilated）畳み込み．例えば，output[t]はinput[t+1]に依存しません．時間的順序を無視すべきでない時系列データをモデリングする際に有効です．WaveNet: A Generative Model for Raw Audio, section 2.1を参照して下さい．
- **dilation_rate**: 整数か単一の整数からなるタプル/リストで，dilated convolutionで使われる膨張率を指定します． 現在， `dilation_rate` value != 1 とすると， `strides` value != 1を指定することはできません．
- **activation**: 使用する活性化関数の名前（activationsを参照）， 何も指定しなければ，活性化は一切適用されません（つまり"線形"活性 `a(x) = x` ）．
- **use_bias**: 真理値で，バイアスベクトルを加えるかどうかを指定します．
- **kernel_initializer**: `kernel` の重み行列の初期値を指定します． （initializersを参照）
- **bias_initializer**: バイアスベクトルの初期値を指定します． （initializersを参照）
- **kernel_regularizer**: `kernel` の重みに適用させるRegularizerを指定します． （regularizerを参照）
- **bias_regularizer**: バイアスに適用させるRegularizerを指定します． （regularizerを参照）
- **activity_regularizer**: 出力テンソルに適用させるRegularizerを指定します． （regularizerを参照）
- **kernel_constraint**: カーネルの行列に適用させるConstraintを指定します． （constraintを参照）
- **bias_constraint**: バイアスベクトルに適用させるConstraintを指定します． （constraintを参照）

入力のshape

shapeが `(batch_size, steps, input_dim)` の3階テンソル．

出力のshape

shapeが `(batch_size, new_steps, nb_filter)` の3階テンソル． `steps` はパディングにより変わっている可能性があります．

Conv2D

[source]

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate
```

2次元の畳み込みレイヤー（例えばイメージに対する空間的畳み込み）.

このレイヤーは畳み込みカーネルを生成します．これはレイヤーの入力を畳み込んで，出力のテンソルを作ります． `use_bias` をTrueにすると，バイアスベクトルが出力に加えられます． `activation` が `None` でない場合，指定した活性化関数が出力に適用されます.

このレイヤーをモデルの第1層に使うときはキーワード引数 `input_shape` （整数のタプル，サンプル軸を含まない）を指定してください． 例えば， `data_format="channels_last"` のとき，128x128 RGB画像では `input_shape=(128, 128, 3)` となります.

引数

- **filters**: 整数で，出力空間の次元（つまり畳み込みにおける出力フィルタの数）.
- **kernel_size**: 整数か2つの整数からなるタプル/リストで，2次元の畳み込みウィンドウの幅と高さを指定します. 単一の整数の場合は全ての次元に対して同じ値を指定します．単一の整数の場合は正方形のカーネルになります.
- **strides**: 整数か2つの整数からなるタプル/リストで畳み込みの縦と横のストライドをそれぞれ指定できます．単一の整数の場合は幅と高さが同様のストライドになります．strides value != 1とすると `dilation_rate` value != 1と指定できません.
- **padding**: `"valid"` か `"same"` のどちらかを指定します.
- **data_format**: 文字列で， `"channels_last"` （デフォルト）か `"channels_first"` のどちらかを指定します．これは入力における次元の順序です． `"channels_last"` の場合，入力のshape は `"(batch, height, width, channels)"` となり， `"channels_first"` の場合 は `"(batch, channels, height, width)"` となります．デフォルトはKerasの設定ファイル `~/.keras/keras.json` の `image_data_format` の値です．一度も値を変更していなければ，"channels_last"になります.
- **dilation_rate**: 整数か2つの整数からなるタプル/リストで，dilated convolutionで使われる膨張率を指定します． 現在， `dilation_rate` value != 1 とすると，strides value != 1を指定することはできません.
- **activation**: 使用する活性化関数の名前（activationsを参照）， 何も指定しなければ，活性化は一切適用されません（つまり"線形"活性 `a(x) = x` ）.
- **use_bias**: 真理値で，バイアスベクトルを加えるかどうかを指定します.
- **kernel_initializer**: `kernel` の重み行列の初期値を指定します． （initializersを参照）
- **bias_initializer**: バイアスベクトルの初期値を指定します． （initializersを参照）
- **kernel_regularizer**: `kernel` の重み行列に適用させるRegularizerを指定します． （regularizerを参照）
- **bias_regularizer**: バイアスベクトルに適用させるRegularizerを指定します． （regularizerを参照）
- **activity_regularizer**: 出力テンソルに適用させるRegularizerを指定します． （regularizerを参照）
- **kernel_constraint**: カーネルの行列に適用させるConstraintを指定します． （constraintを参照）
- **bias_constraint**: バイアスベクトルに適用させるConstraintを指定します． （constraintを参照）

入力のshape

data_format='channels_first'の場合， `(batch_size, channels, rows, cols)` の4階テンソル.
data_format='channels_last'の場合， `(batch_size, rows, cols, channels)` の4階テンソルになります.

出力のshape

data_format='channels_first'の場合，`(samples, channels, rows, cols)`の4階テンソル．
data_format='channels_last'の場合，`(samples, rows, cols, channels)`の4階テンソルになります．`rows`と`cols`値はパディングにより変わっている可能性があります．

SeparableConv2D

[source]

```
keras.layers.SeparableConv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dept
```

Depthwiseな2次元separable畳み込み層．

separable畳み込み演算は，depthwiseの空間的な畳み込み（各入力チャネルに別々に作用する）を実行し，続いてpointwiseに畳み込みを行い，両者の出力チャネルを混合します．`depth_multiplier`は，出力チャネルを生成するための入力チャネルの数を指定します．

separable畳み込み演算はひとつのカーネルをふたつの小さなカーネルに分解する方法として，直感的に理解することができます．もしくはInception blockの極端な例として考えることもできます．

引数

- **filters**: 整数で，出力空間の次元（つまり畳み込みにおける出力フィルタの数）．
- **kernel_size**: 整数か2つの整数からなるタプル/リストで，2次元の畳み込みウィンドウの幅と高さを指定します．単一の整数の場合は全ての次元に対して同じ値を指定します．
- **strides**: 整数か2つの整数からなるタプル/リストで畳み込みの縦と横のストライドをそれぞれ指定できます．単一の整数の場合は幅と高さが同様のストライドになります．strides value != 1とすると`dilation_rate` value != 1と指定できません．
- **padding**: `"valid"`か`"same"`のどちらかを指定します．
- **data_format**: 文字列で，`"channels_last"`（デフォルト）か`"channels_first"`のどちらかを指定します．これは入力における次元の順序です．`"channels_last"`の場合，入力のshapeは`"(batch, height, width, channels)"`となり，`"channels_first"`の場合は`"(batch, channels, height, width)"`となります．デフォルトはKerasの設定ファイル`~/.keras/keras.json`の`image_data_format`の値です．一度も値を変更していなければ，"channels_last"になります．
- **depth_multiplier**: 各入力チャネルに対するdepthwiseな畳み込みチャネルの数．深さ方向畳み込みチャネルの出力総数は，`filters_in * depth_multiplier`に等しくなります．
- **activation**: 使用する活性化関数の名前（activationsを参照），何も指定しなければ，活性化は一切適用されません（つまり"線形"活性`a(x) = x`）．
- **use_bias**: 真理値で，バイアスベクトルを加えるかどうかを指定します．
- **depthwise_initializer**: カーネルの重み行列の初期値をdepthwiseに指定します．（initializersを参照）
- **pointwise_initializer**: カーネルの重み行列の初期値をpointwiseに指定します．（initializersを参照）
- **bias_initializer**: バイアスベクトルの初期値を指定します．（initializersを参照）
- **depthwise_regularizer**: 重み行列に対し，"depthwise"に適用させるRegularizerを指定します．（regularizerを参照）
- **pointwise_regularizer**: 重み行列に対し，"pointwise"に適用させるRegularizerを指定します．（regularizerを参照）
- **bias_regularizer**: バイアスベクトルに適用させるRegularizerを指定します．（regularizerを参照）
- **activity_regularizer**: 出力テンソルに適用させるRegularizerを指定します．（regularizerを参照）
- **depthwise_constraint**: カーネル行列に対し，"depthwise"に適用させるConstraintを指定します．（constraintを参照）
- **depthwise_constraint**: カーネル行列に対し，"pointwise"に適用させるConstraintを指定します．（constraintを参照）
- **bias_constraint**: バイアスベクトルに適用させるConstraintを指定します．（constraintを参照）

入力のshape

data_format='channels_first'の場合， `(batch_size, channels, rows, cols)` の4階テンソル．
data_format='channels_last'の場合， `(batch_size, rows, cols, channels)` の4階テンソルになります．

出力のshape

data_format='channels_first'の場合， `(batch, channels, rows, cols)` の4階テンソル．
data_format='channels_last'の場合， `(batch, rows, cols, channels)` の4階テンソルになります． `rows` と `cols` 値はパディングにより変わっている可能性があります．

Conv2DTranspose [source]

```
keras.layers.Conv2DTranspose(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, acti
```

transposed畳み込みレイヤー（Deconvolutionとも呼ばれます）．

一般的に，transposed畳み込み演算は通常の畳み込みに対して逆の演算を行いたい時に使われます．つまり，なんらかの畳み込み演算の出力を，接続パターンを保ちながら入力の形に変換する層です．

このレイヤーをモデルの第一層に使うときはキーワード引数 `input_shape` （整数のタプル，サンプル軸を含まない）を指定してください． 例えば `data_format="channels_last"` のとき，128x128 RGB画像では `input_shape=(128, 128, 3)` となります．

引数

- **filters**: 整数で，出力空間の次元（つまり畳み込みにおける出力フィルタの数）．
- **kernel_size**: 整数か2つの整数からなるタプル/リストで，2次元の畳み込みウィンドウの幅と高さを指定します. 単一の整数の場合は全ての次元に対して同じ値を指定します．
- **strides**: 整数か2つの整数からなるタプル/リストで畳み込みの縦と横のストライドをそれぞれ指定できます．単一の整数の場合は幅と高さが同様のストライドになります．strides value != 1とすると `dilation_rate` value != 1と指定できません．
- **padding**: `"valid"` か `"same"` のどちらかを指定します．
- **data_format**: 文字列で， `"channels_last"` （デフォルト）か `"channels_first"` のどちらかを指定します．これは入力における次元の順序です． `"channels_last"` の場合，入力のshape は `"(batch, height, width, channels)"` となり， `"channels_first"` の場合は `"(batch, channels, height, width)"` となります．デフォルトはKerasの設定ファイル `~/.keras/keras.json` の `image_data_format` の値です．一度も値を変更していなければ，"channels_last"になります．
- **dilation_rate**: 整数か2つの整数からなるタプル/リストで，dilated convolutionで使われる膨張率を指定します． 現在， `dilation_rate` value != 1 とすると，strides value != 1を指定することはできません．
- **activation**: 使用する活性化関数の名前（activationsを参照）， 何も指定しなければ，活性化は一切適用されません（つまり"線形"活性 `a(x) = x` ）．
- **use_bias**: 真理値で，バイアスベクトルを加えるかどうかを指定します．
- **kernel_initializer**: `kernel` の重み行列の初期値を指定します． （initializersを参照）
- **bias_initializer**: バイアスベクトルの初期値を指定します． （initializersを参照）
- **kernel_regularizer**: `kernel` の重み行列に適用させるRegularizerを指定します． （regularizerを参照）
- **bias_regularizer**: バイアスベクトルに適用させるRegularizer関数を指定します． （regularizerを参照）
- **activity_regularizer**: 出力テンソルに適用させるRegularizer関数を指定します． （regularizerを参照）

- **kernel_constraint**: カーネルの行列に適用させるConstraint関数を指定します．（Constraintを参照）
- **bias_constraint**: バイアスベクトルに適用させるConstraint関数を指定します．（Constraintを参照）

入力のshape

data_format='channels_first'の場合，`(batch, channels, rows, cols)`の4階テンソル．
data_format='channels_last'の場合，`(batch, rows, cols, channels)`の4階テンソルになります．

出力のshape

data_format='channels_first'の場合，`(batch, filters, new_rows, new_cols)`の4階テンソル．
data_format='channels_last'の場合，`(batch, new_rows, new_cols, channels)`の4階テンソルになります．`rows`と`cols`値はパディングにより変わっている可能性があります．

参考文献

- A guide to convolution arithmetic for deep learning
- Deconvolutional Networks

Conv3D

[source]

```
keras.layers.Conv3D(filters, kernel_size, strides=(1, 1, 1), padding='valid', data_format=None, dilation_r
```

3次元入力をフィルターする畳み込み演算（例えば高さを含めた空間の畳込み）．

このレイヤーは畳み込みカーネルを生成します．これはレイヤーの入力を畳み込んで，出力のテンソルを作ります．`use_bias`をTrueにすると，バイアスベクトルが出力に加えられます．`activation`が`None`でない場合，指定した活性化関数が出力に適用されます．

このレイヤーをモデルの第一層に使うときはキーワード引数`input_shape`（整数のタプル，サンプル軸を含まない）を指定してください．例えば`data_format="channels_last"`の場合，シングルチャネルの128x128x128の立体は`input_shape=(128, 128, 128, 1)`です．

引数

- **filters**: 整数で，出力空間の次元（つまり畳み込みにおける出力フィルタの数）．
- **kernel_size**: 整数か3つの整数からなるタプル/リストで，3次元の畳み込みウィンドウの幅と高さを指定します．単一の整数の場合は全ての次元に対して同じ値を指定します．
- **strides**: 整数か3つの整数からなるタプル/リストで畳み込みの縦と横のストライドをそれぞれ指定できます．単一の整数の場合は幅と高さが同様のストライドになります．strides value != 1とすると`dilation_rate` value != 1と指定できません．
- **padding**: `"valid"`か`"same"`のどちらかを指定します．
- **data_format**: 文字列で，`"channels_last"`（デフォルト）か`"channels_first"`のどちらかを指定します．これは入力における次元の順序です．`"channels_last"`の場合，入力のshapeは`(batch, spatial_dim1, spatial_dim2, spatial_dim3, channels)`となり，`"channels_first"`の場合は`(batch, channels, spatial_dim1, spatial_dim2, spatial_dim3)`となります．デフォルトはKerasの設定ファイル`~/.keras/keras.json`の`image_data_format`の値です．一度も値を変更していなければ，"channels_last"になります．

- **dilation_rate**: 整数か3つの整数からなるタプルで，dilated convolutionで使われる膨張率を指定します． 現在，`dilation_rate` value != 1 とすると，strides value != 1を指定することはできません．
- **activation**: 使用する活性化関数の名前（activationsを参照），何も指定しなければ，活性化は一切適用されません（つまり"線形"活性`a(x) = x`）．
- **use_bias**: 真理値で，バイアスベクトルを加えるかどうかを指定します．
- **kernel_initializer**: `kernel`の重み行列の初期値を指定します．（initializersを参照）
- **bias_initializer**: バイアスベクトルの初期値を指定します．（initializersを参照）
- **kernel_regularizer**: `kernel`の重み行列に適用させるRegularizerを指定します．（regularizerを参照）
- **bias_regularizer**: バイアスベクトルに適用させるRegularizer関数を指定します．（regularizerを参照）
- **activity_regularizer**: 出力テンソルに適用させるRegularizer関数を指定します．（regularizerを参照）
- **kernel_constraint**: カーネルの行列に適用させるConstraint関数を指定します．（Constraintを参照）
- **bias_constraint**: バイアスベクトルに適用させるConstraint関数を指定します．（Constraintを参照）

入力のshape

data_format='channels_first'の場合，`(samples, channels, conv_dim1, conv_dim2, conv_dim3)`の5階テンソル． data_format='channels_last'の場合，`(samples, conv_dim1, conv_dim2, conv_dim3, channels)`の5階テンソルになります．

出力のshape

data_format='channels_first'の場合，`(samples, channels, conv_dim1, conv_dim2, conv_dim3)`の5階テンソル． data_format='channels_last'の場合，`(samples, conv_dim1, conv_dim2, conv_dim3, channels)`の5階テンソルになります．`conv_dim1`，`conv_dim2`，`conv_dim3`の値はパディングにより変わっている可能性があります．

Cropping1D [source]

```
keras.layers.Cropping1D(cropping=(1, 1))
```

一次元の入力をクロップする（切り落とす）層（例えば時間の配列）．

クロップは時間軸に沿って実行されます(axis 1).

引数

- **cropping**: 整数か長さ2の整数のタプルで，クロップしたいユニットの数を指定します．2つの整数からなるタプルで指定した場合は，それぞれ両側からクロップします．1つの整数の場合は，両側から同数のユニットをクロップします．

入力のShape

`(batch, axis_to_crop, features)`の3階テンソル．

出力のShape

`(batch, cropped_axis, features)`の3階テンソル．

Cropping2D [source]

```
keras.layers.Cropping2D(cropping=((0, 0), (0, 0)), data_format=None)
```

二次元の入力をクロップする（切り落とす）層（例えば画像）.

クロップは幅と高さに対して実行されます.

引数

- **cropping**: 整数, タプル（2つの整数）, タプル（2つの整数）のタプルのいずれか.
  - 整数: 幅と高さに対称なクロップが実行されます.
  - タプル（2つの整数）: 幅と高さでそれぞれ対称なクロップが実行されます. `(symmetric_height_crop, symmetric_width_crop)`
  - タプルのタプル: 四辺それぞれにクロップが実行されます. `(top_crop, bottom_crop), (left_crop, right_crop))`
- **data_format**: `"channels_last"`（デフォルト）か`"channels_first"`を指定します. `"channels_last"`の場合，入力は`"(batch, height, width, channels)"`. `"channels_first"`の場合は`"(batch, channels, height, width)"`となります. デフォルトはKerasの設定ファイル`~/.keras/keras.json`の`image_data_format`の値です. 一度も値を変更していなければ，"channels_last"になります.

入力のShape

`data_format`が`"channels_last"`の場合，`(batch, rows, cols, channels)`. `"channels_first"`の場合，`(batch, channels, rows, cols)`の4階テンソル.

出力のShape

`data_format`が`"channels_last"`の場合，`(batch, cropped_rows, cropped_cols, channels)`. `"channels_first"`の場合，`(batch, channels, cropped_rows, cropped_cols)`の4階テンソル.

例

```
# Crop the input 2D images or feature maps
model = Sequential()
model.add(Cropping2D(cropping=((2, 2), (4, 4)),
                     input_shape=(28, 28, 3)))
# now model.output_shape == (None, 24, 20, 3)
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Cropping2D(cropping=((2, 2), (2, 2))))
# now model.output_shape == (None, 20, 16. 64)
```

Cropping3D [source]

```
keras.layers.Cropping3D(cropping=((1, 1), (1, 1), (1, 1)), data_format=None)
```

三次元の入力をクロップする（切り落とす）層（例えば空間や時空間）.

引数

- **cropping**: 整数, タプル（2つの整数）, タプル（2つの整数）のタプル, のいずれか.
  - 整数: 3つの軸に対して対称なクロップが実行されます.

- タプル（3つの整数）：3つの軸に対して，それぞれ対称なクロップが実行されます．
  `(symmetric_dim1_crop, symmetric_dim2_crop, symmetric_dim3_crop)`
  - タプルのタプル: 六面それぞれにクロップが実行されます．
    `((left_dim1_crop, right_dim1_crop), (left_dim2_crop, right_dim2_crop), (left_dim3_crop, right_dim3_crop))`

- **data_format**: `"channels_last"`（デフォルト）か`"channels_first"`を指定します．`"channels_last"`の場合，入力は`"(batch, spatial_dim1, spatial_dim2, spatial_dim3, channels)"`．`"channels_first"`の場合は`"(batch, channels, spatial_dim1, spatial_dim2, spatial_dim3)"`となります．デフォルトはKerasの設定ファイル`~/.keras/keras.json`の`image_data_format`の値です．一度も値を変更していなければ，"channels_last"になります．

入力のShape

`data_format`が`"channels_last"`の場合，`(batch, first_cropped_axis, second_cropped_axis, third_cropped_axis, depth)`．`"channels_first"`の場合，`(batch, depth, first_cropped_axis, second_cropped_axis, third_cropped_axis)`の5階テンソル．

出力のShape

`data_format`が`"channels_last"`の場合，`(batch, first_axis_to_crop, second_axis_to_crop, third_axis_to_crop, depth)`．
`"channels_first"`の場合，`(batch, depth, first_axis_to_crop, second_axis_to_crop, third_axis_to_crop)`の5階テンソル．

UpSampling1D [source]

```
keras.layers.UpSampling1D(size=2)
```

1次元の入力に対するアップサンプリングレイヤー．

時間軸方向にそれぞれの時間ステップを`size`回繰り返します．

引数

- **size**: 整数．upsampling係数．

入力のshape

`(batch, steps, features)`の3階テンソル．

出力のshape

`(batch, upsampled_steps, features)`の3階テンソル．

UpSampling2D [source]

```
keras.layers.UpSampling2D(size=(2, 2), data_format=None)
```

2次元の入力に対するアップサンプリングレイヤー.

データの行と列をそれぞれsize[0]及びsize[1]回繰り返します.

引数

- **size**: 整数か2つの整数のタプル．行と列のupsampling係数.
- **data_format**: `"channels_last"`（デフォルト）か`"channels_first"`を指定します.`"channels_last"`の場合，入力のshapeは`"(batch, height, width, channels)"`となり，`"channels_first"`の場合は`"(batch, channels, height, width)"`となります．デフォルトはKerasの設定ファイル`~/.keras/keras.json`の`image_data_format`の値です．一度も値を変更していなければ，"channels_last"になります.

入力のshape

data_format='channels_last'の場合，`(batch, rows, cols, channels)`の4階テンソル.
data_format='channels_first'の場合，`(batch, channels, rows, cols)`の4階テンソルになります.

出力のshape

data_format='channels_first'の場合，`(batch, channels, upsampled_rows, upsampled_cols)`の4階テンソル． data_format='channels_last'の場合，`(batch, upsampled_rows, upsampled_cols, channels)`の4階テンソルになります.

UpSampling3D

[source]

```
keras.layers.UpSampling3D(size=(2, 2, 2), data_format=None)
```

3次元の入力に対するアップサンプリングレイヤー.

データの1番目，2番目，3番目の次元をそれぞれsize[0]，size[1]，size[2]だけ繰り返す.

引数

- **size**: 3つの整数のタプル．dim1，dim2，dim3のアップサンプリング係数.
- **data_format**: `"channels_last"`（デフォルト）か`"channels_first"`を指定します.`"channels_last"`の場合，入力のshapeは`"(batch, spatial_dim1, spatial_dim2, spatial_dim3, channels)"`となり，`"channels_first"`の場合は`"(batch, channels, spatial_dim1, spatial_dim2, spatial_dim3)"`となります．デフォルトはKerasの設定ファイル`~/.keras/keras.json`の`image_data_format`の値です．一度も値を変更していなければ，"channels_last"になります.

入力のshape

data_format='channels_last'の場合，`(batch, channels, spatial_dim1, spatial_dim2, spatial_dim3)`の5階テンソル． data_format='channels_first'の場合，
`(batch, spatial_dim1, spatial_dim2, spatial_dim3, channels)`の5階テンソルになります.

出力のshape

data_format='channels_last'の場合，
`(batch, upsampled_dim1, upsampled_dim2, upsampled_dim3, channels)`の5階テンソル．
data_format='channels_first'の場合，
`(batch, channels, upsampled_dim1, upsampled_dim2, upsampled_dim3)`の5階テンソルになります．

ZeroPadding1D

[source]

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

一次元入力（例，時系列）に対するゼロパディングレイヤー．

引数

- **padding**: 整数，タプル（2つの整数）のいずれか．
  - 整数: パディング次元(axis 1)の始めと終わりにいくつのゼロを加えるか．
  - （長さ2の）整数のタプル: 始めと終わりにそれぞれいくつのゼロを加えるか． `(left_pad, right_pad)`

入力のshape

`(batch, axis_to_pad, features)`の3階テンソル．

出力のshape

`(batch, padded_axis, features)`の3階テンソル．

ZeroPadding2D

[source]

```
keras.layers.ZeroPadding2D(padding=(1, 1), data_format=None)
```

2次元入力(例，画像)のためのゼロパディングレイヤー

このレイヤーは画像のテンソルの上下左右にゼロの行と列を追加します．

引数

- **padding**: 整数，タプル（2つの整数），タプル（2つの整数）のタプル．
  - 整数: 幅と高さにたいして対称なパディングを実行する．
  - タプル: 幅と高さ，それぞれに対して対称なパディングを実行する．
    `(symmetric_height_pad, symmetric_width_pad)`
  - タプルのタプル: 四辺それぞれにパディング． `((top_pad, bottom_pad), (left_pad, right_pad))`
- **data_format**: `"channels_last"` （デフォルト）か `"channels_first"` を指定します．`"channels_last"` の場合，入力は `"(batch, height, width, channels)"` ． `"channels_first"` の場合は `"(batch, channels, height, width)"` となります． デフォルトはKerasの設定ファイル `~/.keras/keras.json` の `image_data_format` の値です．一度も値を変更していなければ，"channels_last"になります．

入力のShape

`data_format` が `"channels_last"` の場合， `(batch, rows, cols, channels)` ． `"channels_first"` の場合， `(batch, channels, rows, cols)` の4階テンソル．

出力のShape

`data_format`が`"channels_last"`の場合，`(batch, padded_rows, padded_cols, channels)`.
`"channels_first"`の場合，`(batch, channels, padded_rows, padded_cols)`の4階テンソル.

ZeroPadding3D [source]

```
keras.layers.ZeroPadding3D(padding=(1, 1, 1), data_format=None)
```

3次元データ(空間及び時空間)のためのゼロパディングレイヤー.

引数

- **cropping**: 整数, タプル（2つの整数），タプル（2つの整数）のタプル，のいずれか.
  - 整数: 3つの軸に対して対称なパディングが実行されます.
  - タプル（3つの整数）: 3つの軸に対して，それぞれ対称なパディングが実行されます.
    `(symmetric_dim1_pad, symmetric_dim2_pad, symmetric_dim3_pad)`
  - タプルのタプル: 六面それぞれにパディングが実行されます.
    `((left_dim1_pad, right_dim1_pad), (left_dim2_pad, right_dim2_pad), (left_dim3_pad, right_dim3_pad))`
- **data_format**: `"channels_last"`（デフォルト）か`"channels_first"`を指定します．`"channels_last"`の場合，入力は`"(batch, spatial_dim1, spatial_dim2, spatial_dim3, channels)"`．`"channels_first"`の場合は`"(batch, channels, spatial_dim1, spatial_dim2, spatial_dim3)"`となります．デフォルトはKerasの設定ファイル`~/.keras/keras.json`の`image_data_format`の値です．一度も値を変更していなければ，"channels_last"になります.

入力のShape

`data_format`が`"channels_last"`の場合，`(batch, first_axis_to_pad, second_axis_to_pad, third_axis_to_pad, depth)`．`"channels_first"`の場合，`(batch, depth, first_axis_to_pad, second_axis_to_pad, third_axis_to_pad)`の5階テンソル.

出力のShape

`data_format`が`"channels_last"`の場合，`(batch, first_padded_axis, second_padded_axis, third_axis_to_pad, depth)`.
`"channels_first"`の場合，`(batch, depth, first_padded_axis, second_padded_axis, third_axis_to_pad)`の5階テンソル.