

tf.keras.layers.LocallyConnected2D

Contents

Class LocallyConnected2D

Properties

activity_regularizer

dtype

Class **LocallyConnected2D**Inherits From: [Layer](#)Defined in [tensorflow/python/keras/_impl/keras/layers/local.py](#).

Locally-connected layer for 2D inputs.

The **LocallyConnected2D** layer works similarly to the **Conv2D** layer, except that weights are unshared, that is, a different set of filters is applied at each different patch of the input.

Examples:

```
# apply a 3x3 unshared weights convolution with 64 output filters on a
32x32 image
# with `data_format="channels_last"`:
model = Sequential()
model.add(LocallyConnected2D(64, (3, 3), input_shape=(32, 32, 3)))
# now model.output_shape == (None, 30, 30, 64)
# notice that this layer will consume (30*30)*(3*3*3*64) + (30*30)*64
parameters

# add a 3x3 unshared weights convolution on top, with 32 output filters:
model.add(LocallyConnected2D(32, (3, 3)))
# now model.output_shape == (None, 28, 28, 32)
```

Arguments:

- **filters**: Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions.
- **padding**: Currently only support **"valid"** (case-insensitive). **"same"** will be supported in future.
- **data_format**: A string, one of **channels_last** (default) or **channels_first**. The ordering of the dimensions in the inputs. **channels_last** corresponds to inputs with shape **(batch, height, width, channels)** while **channels_first** corresponds to inputs with shape **(batch, channels, height, width)**. It defaults to the **image_data_format** value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be **"channels_last"**.

- `activation` : Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).
- `use_bias` : Boolean, whether the layer uses a bias vector.
- `kernel_initializer` : Initializer for the `kernel` weights matrix.
- `bias_initializer` : Initializer for the bias vector.
- `kernel_regularizer` : Regularizer function applied to the `kernel` weights matrix.
- `bias_regularizer` : Regularizer function applied to the bias vector.
- `activity_regularizer` : Regularizer function applied to the output of the layer (its "activation")..
- `kernel_constraint` : Constraint function applied to the kernel matrix.
- `bias_constraint` : Constraint function applied to the bias vector.

Input shape: 4D tensor with shape: `(samples, channels, rows, cols)` if `data_format='channels_first'` or 4D tensor with shape: `(samples, rows, cols, channels)` if `data_format='channels_last'`.

Output shape: 4D tensor with shape: `(samples, filters, new_rows, new_cols)` if `data_format='channels_first'` or 4D tensor with shape: `(samples, new_rows, new_cols, filters)` if `data_format='channels_last'`. `rows` and `cols` values might have changed due to padding.

Properties

`activity_regularizer`

Optional regularizer function for the output of this layer.

`dtype`

`graph`

`input`

Retrieves the input tensor(s) of a layer.

Only applicable if the layer has exactly one input, i.e. if it is connected to one incoming layer.

Returns:

Input tensor or list of input tensors.

Raises:

- `AttributeError` : if the layer is connected to more than one incoming layers.

Raises:

- `RuntimeError` : If called in Eager mode.
- `AttributeError` : If no inbound nodes are found.

`input_mask`

Retrieves the input mask tensor(s) of a layer.

Only applicable if the layer has exactly one inbound node, i.e. if it is connected to one incoming layer.

Returns:

Input mask tensor (potentially None) or list of input mask tensors.

Raises:

- `AttributeError` : if the layer is connected to more than one incoming layers.

input_shape

Retrieves the input shape(s) of a layer.

Only applicable if the layer has exactly one input, i.e. if it is connected to one incoming layer, or if all inputs have the same shape.

Returns:

Input shape, as an integer shape tuple (or list of shape tuples, one tuple per input tensor).

Raises:

- `AttributeError` : if the layer has no defined input_shape.
- `RuntimeError` : if called in Eager mode.

losses

name

non_trainable_variables

non_trainable_weights

output

Retrieves the output tensor(s) of a layer.

Only applicable if the layer has exactly one output, i.e. if it is connected to one incoming layer.

Returns:

Output tensor or list of output tensors.

Raises:

- `AttributeError` : if the layer is connected to more than one incoming layers.
- `RuntimeError` : if called in Eager mode.

output_mask

Retrieves the output mask tensor(s) of a layer.

Only applicable if the layer has exactly one inbound node, i.e. if it is connected to one incoming layer.

Returns:

Output mask tensor (potentially None) or list of output mask tensors.

Raises:

- `AttributeError` : if the layer is connected to more than one incoming layers.

output_shape

Retrieves the output shape(s) of a layer.

Only applicable if the layer has one output, or if all outputs have the same shape.

Returns:

Output shape, as an integer shape tuple (or list of shape tuples, one tuple per output tensor).

Raises:

- `AttributeError` : if the layer has no defined output shape.
- `RuntimeError` : if called in Eager mode.

scope_name

trainable_variables

trainable_weights

updates

variables

Returns the list of all layer variables/weights.

Returns:

A list of variables.

weights

Returns the list of all layer variables/weights.

Returns:

A list of variables.

Methods

`__init__`

```
__init__(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

`__call__`

```
__call__(
    inputs,
    **kwargs
)
```

Wrapper around `self.call()`, for handling internal references.

If a Keras tensor is passed: - We call `self._add_inbound_node()`. - If necessary, we **build** the layer to match the shape of the input(s). - We update the `_keras_history` of the output tensor(s) with the current layer. This is done as part of `_add_inbound_node()`.

Arguments:

- `inputs`: Can be a tensor or list/tuple of tensors.
- `**kwargs`: Additional keyword arguments to be passed to `call()`.

Returns:

Output of the layer's `call` method.

Raises:

- `ValueError`: in case the layer is missing shape information for its **build** call.

`__deepcopy__`

```
__deepcopy__(memo)
```

add_loss

```
add_loss(  
    losses,  
    inputs=None  
)
```

Add loss tensor(s), potentially dependent on layer inputs.

Some losses (for instance, activity regularization losses) may be dependent on the inputs passed when calling a layer. Hence, when reusing a same layer on different inputs **a** and **b**, some entries in **layer.losses** may be dependent on **a** and some on **b**. This method automatically keeps track of dependencies.

The **get_losses_for** method allows to retrieve the losses relevant to a specific set of inputs.

Arguments:

- **losses**: Loss tensor, or list/tuple of tensors.
- **inputs**: Optional input tensor(s) that the loss(es) depend on. Must match the **inputs** argument passed to the **__call__** method at the time the losses are created. If **None** is passed, the losses are assumed to be unconditional, and will apply across all dataflows of the layer (e.g. weight regularization losses).

Raises:

- **RuntimeError**: If called in Eager mode.

add_update

```
add_update(  
    updates,  
    inputs=None  
)
```

Add update op(s), potentially dependent on layer inputs.

Weight updates (for instance, the updates of the moving mean and variance in a BatchNormalization layer) may be dependent on the inputs passed when calling a layer. Hence, when reusing a same layer on different inputs **a** and **b**, some entries in **layer.updates** may be dependent on **a** and some on **b**. This method automatically keeps track of dependencies.

The **get_updates_for** method allows to retrieve the updates relevant to a specific set of inputs.

This call is ignored in Eager mode.

Arguments:

- **updates**: Update op, or list/tuple of update ops.
- **inputs**: Optional input tensor(s) that the update(s) depend on. Must match the **inputs** argument passed to the **__call__** method at the time the updates are created. If **None** is passed, the updates are assumed to be unconditional, and will apply across all dataflows of the layer.

add_variable

```
add_variable(  
    name,  
    shape,  
    dtype=None,  
    initializer=None,  
    regularizer=None,  
    trainable=True,  
    constraint=None  
)
```

Adds a new variable to the layer, or gets an existing one; returns it.

Arguments:

- `name` : variable name.
- `shape` : variable shape.
- `dtype` : The type of the variable. Defaults to `self.dtype` or `float32`.
- `initializer` : initializer instance (callable).
- `regularizer` : regularizer instance (callable).
- `trainable` : whether the variable should be part of the layer's "trainable_variables" (e.g. variables, biases) or "non_trainable_variables" (e.g. BatchNorm mean, stddev).
- `constraint` : constraint instance (callable).

Returns:

The created variable.

Raises:

- `RuntimeError` : If called in Eager mode with regularizers.

add_weight

```
add_weight(  
    name,  
    shape,  
    dtype=None,  
    initializer=None,  
    regularizer=None,  
    trainable=True,  
    constraint=None  
)
```

Adds a weight variable to the layer.

Arguments:

- `name` : String, the name for the weight variable.
- `shape` : The shape tuple of the weight.
- `dtype` : The dtype of the weight.
- `initializer` : An Initializer instance (callable).

- `regularizer` : An optional Regularizer instance.
- `trainable` : A boolean, whether the weight should be trained via backprop or not (assuming that the layer itself is also trainable).
- `constraint` : An optional Constraint instance.

Returns:

The created weight variable.

apply

```
apply(  
    inputs,  
    *args,  
    **kwargs  
)
```

Apply the layer on a input.

This simply wraps `self.__call__` .

Arguments:

- `inputs` : Input tensor(s).
- `*args` : additional positional arguments to be passed to `self.call` .
- `**kwargs` : additional keyword arguments to be passed to `self.call` .

Returns:

Output tensor(s).

build

```
build(input_shape)
```

call

```
call(inputs)
```

compute_mask

```
compute_mask(  
    inputs,  
    mask=None  
)
```

Computes an output mask tensor.

Arguments:

- `inputs` : Tensor or list of tensors.
- `mask` : Tensor or list of tensors.

Returns:

None or a tensor (or list of tensors, one per output tensor of the layer).

count_params

```
count_params()
```

Count the total number of scalars composing the weights.

Returns:

An integer count.

Raises:

- `ValueError` : if the layer isn't yet built (in which case its weights aren't yet defined).

from_config

```
from_config(  
    cls,  
    config  
)
```

Creates a layer from its config.

This method is the reverse of `get_config`, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by `Container`), nor weights (handled by `set_weights`).

Arguments:

- `config` : A Python dictionary, typically the output of `get_config`.

Returns:

A layer instance.

get_config

```
get_config()
```

get_input_at

```
get_input_at(node_index)
```

Retrieves the input tensor(s) of a layer at a given node.

Arguments:

- `node_index` : Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A tensor (or list of tensors if the layer has multiple inputs).

Raises:

- `RuntimeError` : If called in Eager mode.

get_input_mask_at

```
get_input_mask_at(node_index)
```

Retrieves the input mask tensor(s) of a layer at a given node.

Arguments:

- `node_index` : Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A mask tensor (or list of tensors if the layer has multiple inputs).

get_input_shape_at

```
get_input_shape_at(node_index)
```

Retrieves the input shape(s) of a layer at a given node.

Arguments:

- `node_index` : Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A shape tuple (or list of shape tuples if the layer has multiple inputs).

Raises:

- `RuntimeError` : If called in Eager mode.

get_losses_for

```
get_losses_for(inputs)
```

Retrieves losses relevant to a specific set of inputs.

Arguments:

- `inputs`: Input tensor or list/tuple of input tensors. Must match the `inputs` argument passed to the `__call__` method at the time the losses were created. If you pass `inputs=None`, unconditional losses are returned, such as weight regularization losses.

Returns:

List of loss tensors of the layer that depend on `inputs`.

Raises:

- `RuntimeError`: If called in Eager mode.

get_output_at

```
get_output_at(node_index)
```

Retrieves the output tensor(s) of a layer at a given node.

Arguments:

- `node_index`: Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A tensor (or list of tensors if the layer has multiple outputs).

Raises:

- `RuntimeError`: If called in Eager mode.

get_output_mask_at

```
get_output_mask_at(node_index)
```

Retrieves the output mask tensor(s) of a layer at a given node.

Arguments:

- `node_index`: Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A mask tensor (or list of tensors if the layer has multiple outputs).

get_output_shape_at

```
get_output_shape_at(node_index)
```

Retrieves the output shape(s) of a layer at a given node.

Arguments:

- `node_index` : Integer, index of the node from which to retrieve the attribute. E.g. `node_index=0` will correspond to the first time the layer was called.

Returns:

A shape tuple (or list of shape tuples if the layer has multiple outputs).

Raises:

- `RuntimeError` : If called in Eager mode.

get_updates_for

```
get_updates_for(inputs)
```

Retrieves updates relevant to a specific set of inputs.

Arguments:

- `inputs` : Input tensor or list/tuple of input tensors. Must match the `inputs` argument passed to the `__call__` method at the time the updates were created. If you pass `inputs=None`, unconditional updates are returned.

Returns:

List of update ops of the layer that depend on `inputs`.

Raises:

- `RuntimeError` : If called in Eager mode.

get_weights

```
get_weights()
```

Returns the current weights of the layer.

Returns:

Weights values as a list of numpy arrays.

set_weights

```
set_weights(weights)
```

Sets the weights of the layer, from Numpy arrays.

Arguments:

- `weights` : a list of Numpy arrays. The number of arrays and their shape must match number of the dimensions of the weights of the layer (i.e. it should match the output of `get_weights`).

Raises:

- `ValueError` : If the provided weights list does not match the layer's specifications.

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated November 2, 2017.

Stay Connected

Blog
GitHub
Twitter

Support

Issue Tracker
Release Notes
Stack Overflow

English

[Terms](#) | [Privacy](#)