# Package 'keras'

July 30, 2017

**Type** Package

**Title** R Interface to 'Keras'

**Version** 2.0.5

**Description** Interface to 'Keras', a high-level neural networks API which runs on top of 'TensorFlow'. 'Keras' was developed with a focus on enabling fast experimentation, supports both convolution based networks and recurrent networks (as well as combinations of the two), and runs seamlessly on both 'CPU' and 'GPU' devices.

**Encoding** UTF-8

**License** MIT + file LICENSE

**URL** https://github.com/rstudio/keras

**BugReports** https://github.com/rstudio/keras/issues

**Imports** reticulate (>= 1.0), tensorflow (>= 1.3), tfruns, magrittr, methods, R6

**Suggests** ggplot2, testthat, knitr, rmarkdown

**SystemRequirements** TensorFlow >= 1.1 (https://www.tensorflow.org/)

**RoxygenNote** 6.0.1

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** JJ Allaire [aut, cre],
François Chollet [aut, cph],
RStudio [cph, fnd],
Google [ctb, cph, fnd],
Yuan Tang [ctb, cph],
Daniel Falbel [ctb, cph],
Wouter Van Der Bijl [ctb, cph],
Martin Studer [ctb, cph]

**Maintainer** JJ Allaire <jj@rstudio.com>

**Repository** CRAN

**Date/Publication** 2017-07-30 05:16:19 UTC

# R topics documented:

---

`activation_relu` *Activation functions*

---

## Description

Activations functions can either be used through `layer_activation()`, or through the activation argument supported by all forward layers.

## Usage

```
activation_relu(x, alpha = 0, max_value = NULL)

activation_elu(x, alpha = 1)

activation_selu(x, alpha = 1)

activation_hard_sigmoid(x)
```

```
activation_linear(x)

activation_sigmoid(x)

activation_softmax(x, axis = -1)

activation_softplus(x)

activation_softsign(x)

activation_tanh(x)
```

## Arguments

| | |
|---|---|
| x | Tensor |
| alpha | Alpha value |
| max_value | Max value |
| axis | Integer, axis along which the softmax normalization is applied |

## References

- activation_selu(): Self-Normalizing Neural Networks

---

# application_inception_v3

*Inception V3 model, with weights pre-trained on ImageNet.*

---

## Description

Inception V3 model, with weights pre-trained on ImageNet.

## Usage

```
application_inception_v3(include_top = TRUE, weights = "imagenet",
  input_tensor = NULL, input_shape = NULL, pooling = NULL,
  classes = 1000)

inception_v3_preprocess_input(x)
```

## Arguments

| | |
|---|---|
| include_top | whether to include the fully-connected layer at the top of the network. |
| weights | one of NULL (random initialization) or "imagenet" (pre-training on ImageNet). |
| input_tensor | optional Keras tensor to use as image input for the model. |

| | |
|---|---|
| input_shape | optional shape list, only to be specified if include_top is FALSE (otherwise the input shape has to be (299, 299, 3). It should have exactly 3 inputs channels, and width and height should be no smaller than 71. E.g. (150, 150, 3) would be one valid value. |
| pooling | Optional pooling mode for feature extraction when include_top is FALSE.<br>• NULL means that the output of the model will be the 4D tensor output of the last convolutional layer.<br>• avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.<br>• max means that global max pooling will be applied. |
| classes | optional number of classes to classify images into, only to be specified if include_top is TRUE, and if no weights argument is specified. |
| x | Input tensor for preprocessing |

## Details

Do note that the input image format for this model is different than for the VGG16 and ResNet models (299x299 instead of 224x224).

The inception_v3_preprocess_input() function should be used for image preprocessing.

## Value

A Keras model instance.

## Reference

- Rethinking the Inception Architecture for Computer Vision

---

# application_mobilenet *MobileNet model architecture.*

---

## Description

MobileNet model architecture.

## Usage

```
application_mobilenet(input_shape = NULL, alpha = 1, depth_multiplier = 1,
  dropout = 0.001, include_top = TRUE, weights = "imagenet",
  input_tensor = NULL, pooling = NULL, classes = 1000)

mobilenet_preprocess_input(x)

mobilenet_decode_predictions(preds, top = 5)

mobilenet_load_model_hdf5(filepath)
```

## Arguments

| | |
|---|---|
| input_shape | optional shape list, only to be specified if include_top is FALSE (otherwise the input shape has to be (224, 224, 3) (with channels_last data format) or (3, 224, 224) (with channels_first data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value. |
| alpha | controls the width of the network.<br>• If alpha < 1.0, proportionally decreases the number of filters in each layer.<br>• If alpha > 1.0, proportionally increases the number of filters in each layer.<br>• If alpha = 1, default number of filters from the paper are used at each layer. |
| depth_multiplier | depth multiplier for depthwise convolution (also called the resolution multiplier) |
| dropout | dropout rate |
| include_top | whether to include the fully-connected layer at the top of the network. |
| weights | NULL (random initialization) or imagenet (ImageNet weights) |
| input_tensor | optional Keras tensor (i.e. output of layers.Input()) to use as image input for the model. |
| pooling | Optional pooling mode for feature extraction when include_top is FALSE. - NULL means that the output of the model will be the 4D tensor output of the last convolutional layer. - avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor. - max means that global max pooling will be applied. |
| classes | optional number of classes to classify images into, only to be specified if include_top is TRUE, and if no weights argument is specified. |
| x | input tensor, 4D |
| preds | Tensor encoding a batch of predictions. |
| top | integer, how many top-guesses to return. |
| filepath | File path |

## Details

The mobilenet_preprocess_input() function should be used for image preprocessing. To load a saved instance of a MobileNet model use the mobilenet_load_model_hdf5() function. To prepare image input for MobileNet use mobilenet_preprocess_input(). To decode predictions use mobilenet_decode_predictions().

MobileNet is currently only supported with the TensorFlow backend.

## Value

application_mobilenet() and mobilenet_load_model_hdf5() return a Keras model instance. mobilenet_preprocess_input() returns image input suitable for feeding into a mobilenet model. mobilenet_decode_predictions() returns a list of data frames with variables class_name, class_description, and score (one data frame per sample in batch input).

### Reference

- MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.

---

`application_resnet50` *ResNet50 model for Keras.*

---

## Description

ResNet50 model for Keras.

## Usage

```
application_resnet50(include_top = TRUE, weights = "imagenet",
  input_tensor = NULL, input_shape = NULL, pooling = NULL,
  classes = 1000)
```

## Arguments

| | |
|---|---|
| `include_top` | whether to include the fully-connected layer at the top of the network. |
| `weights` | one of `NULL` (random initialization) or "imagenet" (pre-training on ImageNet). |
| `input_tensor` | optional Keras tensor to use as image input for the model. |
| `input_shape` | optional shape list, only to be specified if `include_top` is FALSE (otherwise the input shape has to be `(224, 224, 3)`. It should have exactly 3 inputs channels, and width and height should be no smaller than 197. E.g. `(200, 200, 3)` would be one valid value. |
| `pooling` | Optional pooling mode for feature extraction when `include_top` is `FALSE`.<br>• `NULL` means that the output of the model will be the 4D tensor output of the last convolutional layer.<br>• `avg` means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.<br>• `max` means that global max pooling will be applied. |
| `classes` | optional number of classes to classify images into, only to be specified if `include_top` is TRUE, and if no `weights` argument is specified. |

## Details

Optionally loads weights pre-trained on ImageNet.

The `imagenet_preprocess_input()` function should be used for image preprocessing.

## Value

A Keras model instance.

## Reference

- Deep Residual Learning for ImageRecognition

## Examples

```
## Not run:
library(keras)

# instantiate the model
model <- application_resnet50(weights = 'imagenet')

# load the image
img_path <- "elephant.jpg"
img <- image_load(img_path, target_size = c(224,224))
x <- image_to_array(img)

# ensure we have a 4d tensor with single element in the batch dimension,
# the preprocess the input for prediction using resnet50
dim(x) <- c(1, dim(x))
x <- imagenet_preprocess_input(x)

# make predictions then decode and print them
preds <- model %>% predict(x)
imagenet_decode_predictions(preds, top = 3)[[1]]

## End(Not run)
```

---

# application_vgg *VGG16 and VGG19 models for Keras.*

---

## Description

VGG16 and VGG19 models for Keras.

## Usage

```
application_vgg16(include_top = TRUE, weights = "imagenet",
  input_tensor = NULL, input_shape = NULL, pooling = NULL,
  classes = 1000)

application_vgg19(include_top = TRUE, weights = "imagenet",
  input_tensor = NULL, input_shape = NULL, pooling = NULL,
  classes = 1000)
```

## Arguments

| | |
|---|---|
| include_top | whether to include the 3 fully-connected layers at the top of the network. |
| weights | one of NULL (random initialization) or "imagenet" (pre-training on ImageNet). |

| | |
|---|---|
| input_tensor | optional Keras tensor to use as image input for the model. |
| input_shape | optional shape list, only to be specified if include_top is FALSE (otherwise the input shape has to be (224, 224, 3) It should have exactly 3 inputs channels, and width and height should be no smaller than 48. E.g. (200, 200, 3) would be one valid value. |
| pooling | Optional pooling mode for feature extraction when include_top is FALSE.<br>• NULL means that the output of the model will be the 4D tensor output of the last convolutional layer.<br>• avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.<br>• max means that global max pooling will be applied. |
| classes | optional number of classes to classify images into, only to be specified if include_top is TRUE, and if no weights argument is specified. |

## Details

Optionally loads weights pre-trained on ImageNet.

The imagenet_preprocess_input() function should be used for image preprocessing.

## Value

Keras model instance.

## Reference

- Very Deep Convolutional Networks for Large-Scale ImageRecognition

## Examples

```
## Not run:
library(keras)

model <- application_vgg16(weights = 'imagenet', include_top = FALSE)

img_path <- "elephant.jpg"
img <- image_load(img_path, target_size = c(224,224))
x <- image_to_array(img)
dim(x) <- c(1, dim(x))
x <- imagenet_preprocess_input(x)

features <- model %>% predict(x)

## End(Not run)
```

application_xception *Xception V1 model for Keras.*

## Description

Xception V1 model for Keras.

## Usage

```
application_xception(include_top = TRUE, weights = "imagenet",
  input_tensor = NULL, input_shape = NULL, pooling = NULL,
  classes = 1000)

xception_preprocess_input(x)
```

## Arguments

| | |
|---|---|
| include_top | whether to include the fully-connected layer at the top of the network. |
| weights | one of NULL (random initialization) or "imagenet" (pre-training on ImageNet). |
| input_tensor | optional Keras tensor to use as image input for the model. |
| input_shape | optional shape list, only to be specified if include_top is FALSE (otherwise the input shape has to be (299, 299, 3). It should have exactly 3 inputs channels, and width and height should be no smaller than 71. E.g. (150, 150, 3) would be one valid value. |
| pooling | Optional pooling mode for feature extraction when include_top is FALSE.<br>• NULL means that the output of the model will be the 4D tensor output of the last convolutional layer.<br>• avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.<br>• max means that global max pooling will be applied. |
| classes | optional number of classes to classify images into, only to be specified if include_top is TRUE, and if no weights argument is specified. |
| x | Input tensor for preprocessing |

## Details

On ImageNet, this model gets to a top-1 validation accuracy of 0.790 and a top-5 validation accuracy of 0.945.

Do note that the input image format for this model is different than for the VGG16 and ResNet models (299x299 instead of 224x224).

The xception_preprocess_input() function should be used for image preprocessing.

This application is only available when using the TensorFlow back-end.

### Value

A Keras model instance.

### Reference

- Xception: Deep Learning with Depthwise Separable Convolutions

---

## backend *Keras backend tensor engine*

---

### Description

Obtain a reference to the `keras.backend` Python module used to implement tensor operations.

### Usage

```
backend(convert = TRUE)
```

### Arguments

| | |
|---|---|
| `convert` | `TRUE` to automatically convert Python objects to their R equivalent. If you pass `FALSE` you can do manual conversion using the `py_to_r()` function. |

### Value

Reference to Keras backend python module.

### Note

See the documentation here `https://keras.io/backend/` for additional details on the available functions.

---

## bidirectional *Bidirectional wrapper for RNNs.*

---

### Description

Bidirectional wrapper for RNNs.

### Usage

```
bidirectional(object, layer, merge_mode = "concat", input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| layer | Recurrent instance. |
| merge_mode | Mode by which outputs of the forward and backward RNNs will be combined. One of 'sum', 'mul', 'concat', 'ave', NULL. If NULL, the outputs will not be combined, they will be returned as a list. |
| input_shape | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |
| batch_input_shape | Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors. |
| batch_size | Fixed batch size for layer |
| dtype | The data type expected by the input, as a string (`float32`, `float64`, `int32`...) |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### See Also

Other layer wrappers: `time_distributed`

---

## `callback_csv_logger` *Callback that streams epoch results to a csv file*

---

### Description

Supports all values that can be represented as a string

### Usage

```
callback_csv_logger(filename, separator = ",", append = FALSE)
```

### Arguments

| | |
|---|---|
| filename | filename of the csv file, e.g. 'run/log.csv'. |
| separator | string used to separate elements in the csv file. |
| append | `TRUE`: append if file exists (useful for continuing training). `FALSE`: overwrite existing file, |

## See Also

Other callbacks: callback_early_stopping, callback_lambda, callback_learning_rate_scheduler, callback_model_checkpoint, callback_progbar_logger, callback_reduce_lr_on_plateau, callback_remote_monitor, callback_tensorboard, callback_terminate_on_naan

---

`callback_early_stopping` *Stop training when a monitored quantity has stopped improving.*

---

## Description

Stop training when a monitored quantity has stopped improving.

## Usage

```
callback_early_stopping(monitor = "val_loss", min_delta = 0, patience = 0,
  verbose = 0, mode = c("auto", "min", "max"))
```

## Arguments

| | |
|---|---|
| `monitor` | quantity to be monitored. |
| `min_delta` | minimum change in the monitored quantity to qualify as an improvement, i.e. an absolute change of less than min_delta, will count as no improvement. |
| `patience` | number of epochs with no improvement after which training will be stopped. |
| `verbose` | verbosity mode, 0 or 1. |
| `mode` | one of "auto", "min", "max". In `min` mode, training will stop when the quantity monitored has stopped decreasing; in `max` mode it will stop when the quantity monitored has stopped increasing; in `auto` mode, the direction is automatically inferred from the name of the monitored quantity. |

## See Also

Other callbacks: callback_csv_logger, callback_lambda, callback_learning_rate_scheduler, callback_model_checkpoint, callback_progbar_logger, callback_reduce_lr_on_plateau, callback_remote_monitor, callback_tensorboard, callback_terminate_on_naan

## callback_lambda *Create a custom callback*

### Description

This callback is constructed with anonymous functions that will be called at the appropriate time. Note that the callbacks expects positional arguments, as:

- `on_epoch_begin` and `on_epoch_end` expect two positional arguments: `epoch`, `logs`
- `on_batch_begin` and `on_batch_end` expect two positional arguments: `batch`, `logs`
- `on_train_begin` and `on_train_end` expect one positional argument: `logs`

### Usage

```
callback_lambda(on_epoch_begin = NULL, on_epoch_end = NULL,
  on_batch_begin = NULL, on_batch_end = NULL, on_train_begin = NULL,
  on_train_end = NULL)
```

### Arguments

| | |
|---|---|
| `on_epoch_begin` | called at the beginning of every epoch. |
| `on_epoch_end` | called at the end of every epoch. |
| `on_batch_begin` | called at the beginning of every batch. |
| `on_batch_end` | called at the end of every batch. |
| `on_train_begin` | called at the beginning of model training. |
| `on_train_end` | called at the end of model training. |

### See Also

Other callbacks: `callback_csv_logger`, `callback_early_stopping`, `callback_learning_rate_scheduler`, `callback_model_checkpoint`, `callback_progbar_logger`, `callback_reduce_lr_on_plateau`, `callback_remote_monitor`, `callback_tensorboard`, `callback_terminate_on_naan`

## callback_learning_rate_scheduler *Learning rate scheduler.*

### Description

Learning rate scheduler.

### Usage

```
callback_learning_rate_scheduler(schedule)
```

### Arguments

| | |
|---|---|
| schedule | a function that takes an epoch index as input (integer, indexed from 0) and returns a new learning rate as output (float). |

### See Also

Other callbacks: callback_csv_logger, callback_early_stopping, callback_lambda, callback_model_checkpoint, callback_progbar_logger, callback_reduce_lr_on_plateau, callback_remote_monitor, callback_tensorboard, callback_terminate_on_naan

---

## callback_model_checkpoint

*Save the model after every epoch.*

---

### Description

filepath can contain named formatting options, which will be filled the value of epoch and keys in logs (passed in on_epoch_end). For example: if filepath is weights.{epoch:02d}-{val_loss:.2f}.hdf5, then the model checkpoints will be saved with the epoch number and the validation loss in the filename.

### Usage

```
callback_model_checkpoint(filepath, monitor = "val_loss", verbose = 0,
  save_best_only = FALSE, save_weights_only = FALSE, mode = c("auto",
  "min", "max"), period = 1)
```

### Arguments

| | |
|---|---|
| filepath | string, path to save the model file. |
| monitor | quantity to monitor. |
| verbose | verbosity mode, 0 or 1. |
| save_best_only | if save_best_only=TRUE, the latest best model according to the quantity monitored will not be overwritten. |
| save_weights_only | if TRUE, then only the model's weights will be saved (save_model_weights_hdf5(filepath)), else the full model is saved (save_model_hdf5(filepath)). |
| mode | one of "auto", "min", "max". If save_best_only=TRUE, the decision to overwrite the current save file is made based on either the maximization or the minimization of the monitored quantity. For val_acc, this should be max, for val_loss this should be min, etc. In auto mode, the direction is automatically inferred from the name of the monitored quantity. |
| period | Interval (number of epochs) between checkpoints. |

### For example

if `filepath` is `weights.{epoch:02d}-{val_loss:.2f}.hdf5`,: then the model checkpoints will be saved with the epoch number and the validation loss in the filename.

### See Also

Other callbacks: `callback_csv_logger`, `callback_early_stopping`, `callback_lambda`, `callback_learning_rate_sch`
`callback_progbar_logger`, `callback_reduce_lr_on_plateau`, `callback_remote_monitor`, `callback_tensorboard`,
`callback_terminate_on_naan`

---

## callback_progbar_logger

*Callback that prints metrics to stdout.*

---

### Description

Callback that prints metrics to stdout.

### Usage

```
callback_progbar_logger(count_mode = "samples")
```

### Arguments

| | |
|---|---|
| `count_mode` | One of "steps" or "samples". Whether the progress bar should count samples seens or steps (batches) seen. |

### See Also

Other callbacks: `callback_csv_logger`, `callback_early_stopping`, `callback_lambda`, `callback_learning_rate_sch`
`callback_model_checkpoint`, `callback_reduce_lr_on_plateau`, `callback_remote_monitor`,
`callback_tensorboard`, `callback_terminate_on_naan`

---

## callback_reduce_lr_on_plateau

*Reduce learning rate when a metric has stopped improving.*

---

### Description

Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

### Usage

```
callback_reduce_lr_on_plateau(monitor = "val_loss", factor = 0.1,
  patience = 10, verbose = 0, mode = c("auto", "min", "max"),
  epsilon = 1e-04, cooldown = 0, min_lr = 0)
```

### Arguments

| | |
|---|---|
| monitor | quantity to be monitored. |
| factor | factor by which the learning rate will be reduced. new_lr = lr<br>• factor |
| patience | number of epochs with no improvement after which learning rate will be reduced. |
| verbose | int. 0: quiet, 1: update messages. |
| mode | one of "auto", "min", "max". In min mode, lr will be reduced when the quantity monitored has stopped decreasing; in max mode it will be reduced when the quantity monitored has stopped increasing; in auto mode, the direction is automatically inferred from the name of the monitored quantity. |
| epsilon | threshold for measuring the new optimum, to only focus on significant changes. |
| cooldown | number of epochs to wait before resuming normal operation after lr has been reduced. |
| min_lr | lower bound on the learning rate. |

### See Also

Other callbacks: callback_csv_logger, callback_early_stopping, callback_lambda, callback_learning_rate_sch
callback_model_checkpoint, callback_progbar_logger, callback_remote_monitor, callback_tensorboard,
callback_terminate_on_naan

---

## callback_remote_monitor

*Callback used to stream events to a server.*

---

### Description

Callback used to stream events to a server.

### Usage

```
callback_remote_monitor(root = "http://localhost:9000",
  path = "/publish/epoch/end/", field = "data", headers = NULL)
```

### Arguments

| | |
|---|---|
| root | root url of the target server. |
| path | path relative to root to which the events will be sent. |
| field | JSON field under which the data will be stored. |
| headers | Optional named list of custom HTTP headers. Defaults to: list(Accept = "application/json",Conte Type= "application/json") |

### See Also

Other callbacks: callback_csv_logger, callback_early_stopping, callback_lambda, callback_learning_rate_sch callback_model_checkpoint, callback_progbar_logger, callback_reduce_lr_on_plateau, callback_tensorboard, callback_terminate_on_naan

---

## callback_tensorboard *TensorBoard basic visualizations*

---

### Description

This callback writes a log for TensorBoard, which allows you to visualize dynamic graphs of your training and test metrics, as well as activation histograms for the different layers in your model.

### Usage

```
callback_tensorboard(log_dir = NULL, histogram_freq = 0,
  write_graph = TRUE, write_images = FALSE, embeddings_freq = 0,
  embeddings_layer_names = NULL, embeddings_metadata = NULL)
```

### Arguments

| | |
|---|---|
| log_dir | The path of the directory where to save the log files to be parsed by Tensorboard. The default is NULL, which will use the active run directory (if available) and otherwise will use "logs". |
| histogram_freq | frequency (in epochs) at which to compute activation histograms for the layers of the model. If set to 0, histograms won't be computed. |
| write_graph | whether to visualize the graph in Tensorboard. The log file can become quite large when write_graph is set to TRUE |
| write_images | whether to write model weights to visualize as image in Tensorboard. |
| embeddings_freq | frequency (in epochs) at which selected embedding layers will be saved. |
| embeddings_layer_names | a list of names of layers to keep eye on. If NULL or empty list all the embedding layers will be watched. |
| embeddings_metadata | a named list which maps layer name to a file name in which metadata for this embedding layer is saved. See the details about the metadata file format. In case if the same metadata file is used for all embedding layers, string can be passed. |

## Details

TensorBoard is a visualization tool provided with TensorFlow.

You can find more information about TensorBoard here.

## See Also

Other callbacks: callback_csv_logger, callback_early_stopping, callback_lambda, callback_learning_rate_sch
callback_model_checkpoint, callback_progbar_logger, callback_reduce_lr_on_plateau,
callback_remote_monitor, callback_terminate_on_naan

---

# callback_terminate_on_naan *Callback that terminates training when a NaN loss is encountered.*

---

## Description

Callback that terminates training when a NaN loss is encountered.

## Usage

```
callback_terminate_on_naan()
```

## See Also

Other callbacks: callback_csv_logger, callback_early_stopping, callback_lambda, callback_learning_rate_sch
callback_model_checkpoint, callback_progbar_logger, callback_reduce_lr_on_plateau,
callback_remote_monitor, callback_tensorboard

---

# compile *Configure a Keras model for training*

---

## Description

Configure a Keras model for training

## Usage

```
compile(object, optimizer, loss, metrics = NULL, loss_weights = NULL,
  sample_weight_mode = NULL, ...)
```

## Arguments

| | |
|---|---|
| object | Model object to compile. |
| optimizer | Name of optimizer or optimizer object. |
| loss | Name of objective function or objective function. If the model has multiple outputs, you can use a different loss on each output by passing a dictionary or a list of objectives. The loss value that will be minimized by the model will then be the sum of all individual losses. |
| metrics | List of metrics to be evaluated by the model during training and testing. Typically you will use `metrics='accuracy'`. To specify different metrics for different outputs of a multi-output model, you could also pass a named list such as `metrics=list(output_a = 'accuracy')`. |
| loss_weights | Optional list specifying scalar coefficients to weight the loss contributions of different model outputs. The loss value that will be minimized by the model will then be the *weighted sum* of all indvidual losses, weighted by the `loss_weights` coefficients. |
| sample_weight_mode | If you need to do timestep-wise sample weighting (2D weights), set this to "temporal". `NULL` defaults to sample-wise weights (1D). If the model has multiple outputs, you can use a different `sample_weight_mode` on each output by passing a list of modes. |
| ... | Additional named arguments passed to `tf$Session$run`. |

## See Also

Other model functions: `evaluate_generator`, `evaluate`, `fit_generator`, `fit`, `get_config`, `get_layer`, `keras_model_sequential`, `keras_model`, `pop_layer`, `predict.keras.engine.training.Model`, `predict_generator`, `predict_on_batch`, `predict_proba`, `summary.keras.engine.training.Model`, `train_on_batch`

---

# constraint_maxnorm *MaxNorm weight constraint*

---

## Description

Constrains the weights incident to each hidden unit to have a norm less than or equal to a desired value.

## Usage

```
constraint_maxnorm(max_value = 2, axis = 0)
```

### Arguments

| | |
|---|---|
| max_value | The maximum norm for the incoming weights. |
| axis | The axis along which to calculate weight norms. For instance, in a dense layer the weight matrix has shape input_dim, output_dim, set axis to 0 to constrain each weight vector of length input_dim,. In a convolution 2D layer with dim_ordering="tf", the weight tensor has shape rows, cols, input_depth, output_depth, set axis to c(0, 1, 2) to constrain the weights of each filter tensor of size rows, cols, input_depth. |

### See Also

Dropout: A Simple Way to Prevent Neural Networks from Overfitting Srivastava, Hinton, et al. 2014

Other constraints: constraint_minmaxnorm, constraint_nonneg, constraint_unitnorm

---

## constraint_minmaxnorm *MinMaxNorm weight constraint*

---

### Description

Constrains the weights incident to each hidden unit to have the norm between a lower bound and an upper bound.

### Usage

```
constraint_minmaxnorm(min_value = 0, max_value = 1, rate = 1, axis = 0)
```

### Arguments

| | |
|---|---|
| min_value | The minimum norm for the incoming weights. |
| max_value | The maximum norm for the incoming weights. |
| rate | The rate for enforcing the constraint: weights will be rescaled to yield (1 - rate) * norm + rate * norm.clip(low, high). Effectively, this means that rate=1.0 stands for strict enforcement of the constraint, while rate<1.0 means that weights will be rescaled at each step to slowly move towards a value inside the desired interval. |
| axis | The axis along which to calculate weight norms. For instance, in a dense layer the weight matrix has shape input_dim, output_dim, set axis to 0 to constrain each weight vector of length input_dim,. In a convolution 2D layer with dim_ordering="tf", the weight tensor has shape rows, cols, input_depth, output_depth, set axis to c(0, 1, 2) to constrain the weights of each filter tensor of size rows, cols, input_depth. |

### See Also

Other constraints: constraint_maxnorm, constraint_nonneg, constraint_unitnorm

## constraint_nonneg *NonNeg weight constraint*

### Description

Constrains the weights to be non-negative.

### Usage

```
constraint_nonneg()
```

### See Also

Other constraints: constraint_maxnorm, constraint_minmaxnorm, constraint_unitnorm

## constraint_unitnorm *UnitNorm weight constraint*

### Description

Constrains the weights incident to each hidden unit to have unit norm.

### Usage

```
constraint_unitnorm(axis = 0)
```

### Arguments

| | |
|---|---|
| axis | The axis along which to calculate weight norms. For instance, in a dense layer the weight matrix has shape input_dim, output_dim, set axis to 0 to constrain each weight vector of length input_dim,. In a convolution 2D layer with dim_ordering="tf", the weight tensor has shape rows, cols, input_depth, output_depth, set axis to c(0, 1, 2) to constrain the weights of each filter tensor of size rows, cols, input_depth. |

### See Also

Other constraints: constraint_maxnorm, constraint_minmaxnorm, constraint_nonneg

count_params *Count the total number of scalars composing the weights.*

## Description

Count the total number of scalars composing the weights.

## Usage

```
count_params(object)
```

## Arguments

| | |
|---|---|
| object | Layer or model object |

## Value

An integer count

## See Also

Other layer methods: get_config, get_input_at, get_weights, reset_states

---

create_layer *Create a Keras Layer*

## Description

Create a Keras Layer

## Usage

```
create_layer(layer_class, object, args = list())
```

## Arguments

| | |
|---|---|
| layer_class | Python layer class or R6 class of type KerasLayer |
| object | Object to compose layer with. This is either a keras_model_sequential() to add the layer to, or another Layer which this layer will call. |
| args | List of arguments to layer constructor function |

## Value

A Keras layer

### Note

The `object` parameter can be missing, in which case the layer is created without a connection to an existing graph.

---

`dataset_boston_housing` *Boston housing price regression dataset*

---

### Description

Dataset taken from the StatLib library which is maintained at Carnegie Mellon University.

### Usage

```
dataset_boston_housing(path = "boston_housing.npz", seed = 113L,
  test_split = 0.2)
```

### Arguments

| | |
|---|---|
| `path` | Path where to cache the dataset locally (relative to ~/.keras/datasets). |
| `seed` | Random seed for shuffling the data before computing the test split. |
| `test_split` | fraction of the data to reserve as test set. |

### Value

Lists of training and test data: `train$x, train$y, test$x, test$y`.

Samples contain 13 attributes of houses at different locations around the Boston suburbs in the late 1970s. Targets are the median values of the houses at a location (in k$).

### See Also

Other datasets: `dataset_cifar100`, `dataset_cifar10`, `dataset_imdb`, `dataset_mnist`, `dataset_reuters`

---

`dataset_cifar10` *CIFAR10 small image classification*

---

### Description

Dataset of 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images.

### Usage

```
dataset_cifar10()
```

## Value

Lists of training and test data: `train$x, train$y, test$x, test$y`.

The `x` data is an array of RGB image data with shape (num_samples, 3, 32, 32).

The `y` data is an array of category labels (integers in range 0-9) with shape (num_samples).

## See Also

Other datasets: `dataset_boston_housing`, `dataset_cifar100`, `dataset_imdb`, `dataset_mnist`, `dataset_reuters`

---

`dataset_cifar100` *CIFAR100 small image classification*

---

## Description

Dataset of 50,000 32x32 color training images, labeled over 100 categories, and 10,000 test images.

## Usage

```
dataset_cifar100(label_mode = c("fine", "coarse"))
```

## Arguments

| | |
|---|---|
| `label_mode` | one of "fine", "coarse". |

## Value

Lists of training and test data: `train$x, train$y, test$x, test$y`.

The `x` data is an array of RGB image data with shape (num_samples, 3, 32, 32).

The `y` data is an array of category labels with shape (num_samples).

## See Also

Other datasets: `dataset_boston_housing`, `dataset_cifar10`, `dataset_imdb`, `dataset_mnist`, `dataset_reuters`

dataset_imdb *IMDB Movie reviews sentiment classification*

## Description

Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative). Reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data. This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words".

## Usage

```
dataset_imdb(path = "imdb.npz", num_words = NULL, skip_top = 0L,
  maxlen = NULL, seed = 113L, start_char = 1L, oov_char = 2L,
  index_from = 3L)
```

## Arguments

| | |
|---|---|
| path | Where to cache the data (relative to ~/.keras/dataset). |
| num_words | Max number of words to include. Words are ranked by how often they occur (in the training set) and only the most frequent words are kept |
| skip_top | Skip the top N most frequently occuring words (which may not be informative). |
| maxlen | Truncate sequences after this length. |
| seed | random seed for sample shuffling. |
| start_char | The start of a sequence will be marked with this character. Set to 1 because 0 is usually the padding character. |
| oov_char | Words that were cut out because of the num_words or skip_top limit will be replaced with this character. |
| index_from | Index actual words with this index and higher. |

## Details

As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown word.

## Value

Lists of training and test data: train$x, train$y, test$x, test$y.

The x data includes integer sequences. If the num_words`` argument was specific, the maximum possible index valu
1. If themaxlen“ argument was specified, the largest possible sequence length is maxlen.

The y data includes a set of integer labels (0 or 1).

## See Also

Other datasets: dataset_boston_housing, dataset_cifar100, dataset_cifar10, dataset_mnist, dataset_reuters

---

dataset_mnist *MNIST database of handwritten digits*

---

## Description

Dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images.

## Usage

```
dataset_mnist(path = "mnist.npz")
```

## Arguments

| | |
|---|---|
| path | Path where to cache the dataset locally (relative to ~/.keras/datasets). |

## Value

Lists of training and test data: `train$x`, `train$y`, `test$x`, `test$y`, where `x` is an array of grayscale image data with shape (num_samples, 28, 28) and `y` is an array of digit labels (integers in range 0-9) with shape (num_samples).

## See Also

Other datasets: dataset_boston_housing, dataset_cifar100, dataset_cifar10, dataset_imdb, dataset_reuters

---

dataset_reuters *Reuters newswire topics classification*

---

## Description

Dataset of 11,228 newswires from Reuters, labeled over 46 topics. As with `dataset_imdb()` , each wire is encoded as a sequence of word indexes (same conventions).

## Usage

```
dataset_reuters(path = "reuters.npz", num_words = NULL, skip_top = 0L,
  maxlen = NULL, test_split = 0.2, seed = 113L, start_char = 1L,
  oov_char = 2L, index_from = 3L)

dataset_reuters_word_index(path = "reuters_word_index.pkl")
```

### Arguments

| | |
|---|---|
| path | Where to cache the data (relative to ~/.keras/dataset). |
| num_words | Max number of words to include. Words are ranked by how often they occur (in the training set) and only the most frequent words are kept |
| skip_top | Skip the top N most frequently occuring words (which may not be informative). |
| maxlen | Truncate sequences after this length. |
| test_split | Fraction of the dataset to be used as test data. |
| seed | Random seed for sample shuffling. |
| start_char | The start of a sequence will be marked with this character. Set to 1 because 0 is usually the padding character. |
| oov_char | words that were cut out because of the num_words or skip_top limit will be replaced with this character. |
| index_from | index actual words with this index and higher. |

### Value

Lists of training and test data: train$x, train$y, test$x, test$y with same format as dataset_imdb(). The dataset_reuters_word_index() function returns a list where the names are words and the values are integer. e.g. word_index[["giraffe"]] might return 1234.

[["giraffe"]]: R:[

### See Also

Other datasets: dataset_boston_housing, dataset_cifar100, dataset_cifar10, dataset_imdb, dataset_mnist

---

## evaluate *Evaluate a Keras model*

---

### Description

Evaluate a Keras model

### Usage

```
evaluate(object, x, y, batch_size = 32, verbose = 1, sample_weight = NULL)
```

### Arguments

| | |
|---|---|
| object | Model object to evaluate |
| x | Vector, matrix, or array of training data (or list if the model has multiple inputs). If all inputs in the model are named, you can also pass a list mapping input names to data. |

| | |
|---|---|
| y | Vector, matrix, or array of target data (or list if the model has multiple outputs). If all outputs in the model are named, you can also pass a list mapping output names to data. |
| batch_size | Number of samples per gradient update. |
| verbose | Verbosity mode (0 = silent, 1 = verbose, 2 = one log line per epoch). |
| sample_weight | Optional array of the same length as x, containing weights to apply to the model's loss for each sample. In the case of temporal data, you can pass a 2D array with shape (samples, sequence_length), to apply a different weight to every timestep of every sample. In this case you should make sure to specify sample_weight_mode="temporal" in compile(). |

## Value

Scalar test loss (if the model has a single output and no metrics) or list of scalars (if the model has multiple outputs and/or metrics).

## See Also

Other model functions: compile, evaluate_generator, fit_generator, fit, get_config, get_layer, keras_model_sequential, keras_model, pop_layer, predict.keras.engine.training.Model, predict_generator, predict_on_batch, predict_proba, summary.keras.engine.training.Model, train_on_batch

---

# evaluate_generator *Evaluates the model on a data generator.*

---

## Description

The generator should return the same kind of data as accepted by test_on_batch().

## Usage

```
evaluate_generator(object, generator, steps, max_queue_size = 10)
```

## Arguments

| | |
|---|---|
| object | Model object to evaluate |
| generator | Generator yielding lists (inputs, targets) or (inputs, targets, sample_weights) |
| steps | Total number of steps (batches of samples) to yield from generator before stopping. |
| max_queue_size | maximum size for the generator queue |

## Value

Scalar test loss (if the model has a single output and no metrics) or list of scalars (if the model has multiple outputs and/or metrics). The attribute model$metrics_names will give you the display labels for the scalar outputs.

## See Also

Other model functions: compile, evaluate, fit_generator, fit, get_config, get_layer, keras_model_sequential, keras_model, pop_layer, predict.keras.engine.training.Model, predict_generator, predict_on_batch, predict_proba, summary.keras.engine.training.Model, train_on_batch

---

`fit` *Train a Keras model*

---

## Description

Trains the model for a fixed number of epochs (iterations on a dataset).

## Usage

```
fit(object, x, y, batch_size = 32, epochs = 10, verbose = 1,
  callbacks = NULL, validation_split = 0, validation_data = NULL,
  shuffle = TRUE, class_weight = NULL, sample_weight = NULL,
  initial_epoch = 0, ...)
```

## Arguments

| | |
|---|---|
| `object` | Model to train. |
| `x` | Vector, matrix, or array of training data (or list if the model has multiple inputs). If all inputs in the model are named, you can also pass a list mapping input names to data. |
| `y` | Vector, matrix, or array of target data (or list if the model has multiple outputs). If all outputs in the model are named, you can also pass a list mapping output names to data. |
| `batch_size` | Number of samples per gradient update. |
| `epochs` | Number of times to iterate over the training data arrays. |
| `verbose` | Verbosity mode (0 = silent, 1 = verbose, 2 = one log line per epoch). |
| `callbacks` | List of callbacks to be called during training. |
| `validation_split` | Float between 0 and 1: fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. |
| `validation_data` | Data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. This could be a list (x_val, y_val) or a list (x_val, y_val, val_sample_weights). |
| `shuffle` | `TRUE` to shuffle the training data before each epoch. |

| | |
|---|---|
| class_weight | Optional named list mapping indices (integers) to a weight (float) to apply to the model's loss for the samples from this class during training. This can be useful to tell the model to "pay more attention" to samples from an under-represented class. |
| sample_weight | Optional array of the same length as x, containing weights to apply to the model's loss for each sample. In the case of temporal data, you can pass a 2D array with shape (samples, sequence_length), to apply a different weight to every timestep of every sample. In this case you should make sure to specify sample_weight_mode="temporal" in compile(). |
| initial_epoch | epoch at which to start training (useful for resuming a previous training run). |
| ... | Unused |

## See Also

Other model functions: compile, evaluate_generator, evaluate, fit_generator, get_config, get_layer, keras_model_sequential, keras_model, pop_layer, predict.keras.engine.training.Model, predict_generator, predict_on_batch, predict_proba, summary.keras.engine.training.Model, train_on_batch

---

# fit_generator *Fits the model on data yielded batch-by-batch by a generator.*

---

## Description

The generator is run in parallel to the model, for efficiency. For instance, this allows you to do real-time data augmentation on images on CPU in parallel to training your model on GPU.

## Usage

```
fit_generator(object, generator, steps_per_epoch, epochs = 1, verbose = 1,
  callbacks = NULL, validation_data = NULL, validation_steps = NULL,
  class_weight = NULL, max_queue_size = 10, initial_epoch = 0)
```

## Arguments

| | |
|---|---|
| object | Keras model object |
| generator | A generator (e.g. like the one provided by flow_images_from_directory() or a custom R generator function).<br>The output of the generator must be a list of one of these forms:<br>- (inputs, targets)<br>- (inputs, targets, sample_weights)<br>All arrays should contain the same number of samples. The generator is expected to loop over its data indefinitely. An epoch finishes when steps_per_epoch batches have been seen by the model. |

`steps_per_epoch`
: Total number of steps (batches of samples) to yield from `generator` before declaring one epoch finished and starting the next epoch. It should typically be equal to the number of unique samples if your dataset divided by the batch size.

`epochs`
: integer, total number of iterations on the data.

`verbose`
: verbosity mode, 0, 1, or 2.

`callbacks`
: list of callbacks to be called during training.

`validation_data`
: this can be either:

  - a generator for the validation data
  - a list (inputs, targets)
  - a list (inputs, targets, sample_weights).

`validation_steps`
: Only relevant if `validation_data` is a generator. Total number of steps (batches of samples) to yield from `generator` before stopping.

`class_weight`
: dictionary mapping class indices to a weight for the class.

`max_queue_size`
: maximum size for the generator queue

`initial_epoch`
: epoch at which to start training (useful for resuming a previous training run)

## Value

Training history object (invisibly)

## See Also

Other model functions: `compile`, `evaluate_generator`, `evaluate`, `fit`, `get_config`, `get_layer`, `keras_model_sequential`, `keras_model`, `pop_layer`, `predict.keras.engine.training.Model`, `predict_generator`, `predict_on_batch`, `predict_proba`, `summary.keras.engine.training.Model`, `train_on_batch`

---

# `fit_image_data_generator` *Fit image data generator internal statistics to some sample data.*

---

## Description

Required for `featurewise_center`, `featurewise_std_normalization` and `zca_whitening`.

## Usage

```
fit_image_data_generator(object, x, augment = FALSE, rounds = 1,
  seed = NULL, ...)
```

### Arguments

| | |
|---|---|
| object | image_data_generator() |
| x | array, the data to fit on (should have rank 4). In case of grayscale data, the channels axis should have value 1, and in case of RGB data, it should have value 3. |
| augment | Whether to fit on randomly augmented samples |
| rounds | If augment, how many augmentation passes to do over the data |
| seed | random seed. |
| ... | Unused |

### See Also

Other image preprocessing: flow_images_from_data, flow_images_from_directory, image_load, image_to_array

---

## fit_text_tokenizer *Update tokenizer internal vocabulary based on a list of texts or list of sequences.*

---

### Description

Update tokenizer internal vocabulary based on a list of texts or list of sequences.

### Usage

```
fit_text_tokenizer(object, x, ...)
```

### Arguments

| | |
|---|---|
| object | Tokenizer returned by text_tokenizer() |
| x | Vector/list of strings, or a generator of strings (for memory-efficiency); Alternatively a list of "sequence" (a sequence is a list of integer word indices). |
| ... | Unused |

### Note

Required before using texts_to_sequences(), texts_to_matrix(), or sequences_to_matrix().

### See Also

Other text tokenization: sequences_to_matrix, text_tokenizer, texts_to_matrix, texts_to_sequences_generator, texts_to_sequences

`flow_images_from_data` *Generates batches of augmented/normalized data from image data and labels*

## Description

Generates batches of augmented/normalized data from image data and labels

## Usage

```
flow_images_from_data(x, y = NULL, generator = image_data_generator(),
  batch_size = 32, shuffle = TRUE, seed = NULL, save_to_dir = NULL,
  save_prefix = "", save_format = "png")
```

## Arguments

| | |
|---|---|
| `x` | data. Should have rank 4. In case of grayscale data, the channels axis should have value 1, and in case of RGB data, it should have value 3. |
| `y` | labels (can be `NULL` if no labels are required) |
| `generator` | Image data generator to use for augmenting/normalizing image data. |
| `batch_size` | int (default: `32`). |
| `shuffle` | boolean (defaut: `TRUE`). |
| `seed` | int (default: `NULL`). |
| `save_to_dir` | `NULL` or str (default: `NULL`). This allows you to optimally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing). |
| `save_prefix` | str (default: ”). Prefix to use for filenames of saved pictures (only relevant if `save_to_dir` is set). |
| `save_format` | one of "png", "jpeg" (only relevant if save_to_dir is set). Default: "png". |

## Details

Yields batches indefinitely, in an infinite loop.

## Yields

`(x, y)` where `x` is an array of image data and `y` is a array of corresponding labels. The generator loops indefinitely.

## See Also

Other image preprocessing: `fit_image_data_generator`, `flow_images_from_directory`, `image_load`, `image_to_array`

flow_images_from_directory *Generates batches of data from images in a directory (with optional augmented/normalized data)*

## Description

Generates batches of data from images in a directory (with optional augmented/normalized data)

## Usage

```
flow_images_from_directory(directory, generator = image_data_generator(),
  target_size = c(256, 256), color_mode = "rgb", classes = NULL,
  class_mode = "categorical", batch_size = 32, shuffle = TRUE,
  seed = NULL, save_to_dir = NULL, save_prefix = "",
  save_format = "png", follow_links = FALSE)
```

## Arguments

| | |
|---|---|
| directory | path to the target directory. It should contain one subdirectory per class. Any PNG, JPG or BMP images inside each of the subdirectories directory tree will be included in the generator. See thisscript for more details. |
| generator | Image data generator (default generator does no data augmentation/normalization transformations) |
| target_size | integer vectir, default: `c(256, 256)`. The dimensions to which all images found will be resized. |
| color_mode | one of "grayscale", "rbg". Default: "rgb". Whether the images will be converted to have 1 or 3 color channels. |
| classes | optional list of class subdirectories (e.g. `c('dogs', 'cats')`). Default: `NULL`, If not provided, the list of classes will be automatically inferred (and the order of the classes, which will map to the label indices, will be alphanumeric). |
| class_mode | one of "categorical", "binary", "sparse" or `NULL`. Default: "categorical". Determines the type of label arrays that are returned: "categorical" will be 2D one-hot encoded labels, "binary" will be 1D binary labels, "sparse" will be 1D integer labels. If `NULL`, no labels are returned (the generator will only yield batches of image data, which is useful to use `predict_generator()`, `evaluate_generator()`, etc.). |
| batch_size | int (default: `32`). |
| shuffle | boolean (defaut: `TRUE`). |
| seed | int (default: `NULL`). |
| save_to_dir | `NULL` or str (default: `NULL`). This allows you to optimally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing). |

| | |
|---|---|
| save_prefix | str (default: ”). Prefix to use for filenames of saved pictures (only relevant if save_to_dir is set). |
| save_format | one of "png", "jpeg" (only relevant if save_to_dir is set). Default: "png". |
| follow_links | whether to follow symlinks inside class subdirectories (default: FALSE) |

## Details

Yields batches indefinitely, in an infinite loop.

## Yields

(x, y) where x is an array of image data and y is a array of corresponding labels. The generator loops indefinitely.

## See Also

Other image preprocessing: fit_image_data_generator, flow_images_from_data, image_load, image_to_array

---

# get_config *Layer/Model configuration*

---

## Description

A layer config is an object returned from get_config() that contains the configuration of a layer or model. The same layer or model can be reinstantiated later (without its trained weights) from this configuration using from_config(). The config does not include connectivity information, nor the class name (those are handled externally).

## Usage

```
get_config(object)

from_config(config)
```

## Arguments

| | |
|---|---|
| object | Layer or model object |
| config | Object with layer or model configuration |

## Value

get_config() returns an object with the configuration, from_config() returns a re-instantation of hte object.

### Note

Objects returned from `get_config()` are not serializable. Therefore, if you want to save and restore a model across sessions, you can use the `model_to_json()` or `model_to_yaml()` functions (for model configuration only, not weights) or the `save_model_hdf5()` function to save the model configuration and weights to a file.

### See Also

Other model functions: `compile`, `evaluate_generator`, `evaluate`, `fit_generator`, `fit`, `get_layer`, `keras_model_sequential`, `keras_model`, `pop_layer`, `predict.keras.engine.training.Model`, `predict_generator`, `predict_on_batch`, `predict_proba`, `summary.keras.engine.training.Model`, `train_on_batch`

Other layer methods: `count_params`, `get_input_at`, `get_weights`, `reset_states`

---

## get_file *Downloads a file from a URL if it not already in the cache.*

---

### Description

Passing the MD5 hash will verify the file after download as well as if it is already present in the cache.

### Usage

```
get_file(fname, origin, file_hash = NULL, cache_subdir = "datasets",
  hash_algorithm = "auto", extract = FALSE, archive_format = "auto",
  cache_dir = NULL)
```

### Arguments

| | |
|---|---|
| `fname` | Name of the file. If an absolute path `/path/to/file.txt` is specified the file will be saved at that location. |
| `origin` | Original URL of the file. |
| `file_hash` | The expected hash string of the file after download. The sha256 and md5 hash algorithms are both supported. |
| `cache_subdir` | Subdirectory under the Keras cache dir where the file is saved. If an absolute path `/path/to/folder` is specified the file will be saved at that location. |
| `hash_algorithm` | Select the hash algorithm to verify the file. options are 'md5', 'sha256', and 'auto'. The default 'auto' detects the hash algorithm in use. |
| `extract` | True tries extracting the file as an Archive, like tar or zip. |
| `archive_format` | Archive format to try for extracting the file. Options are 'auto', 'tar', 'zip', and None. 'tar' includes tar, tar.gz, and tar.bz files. The default 'auto' is ('tar', 'zip'). None or an empty list will return no matches found. |
| `cache_dir` | Location to store cached files, when `NULL` it defaults to the Keras configuration directory. |

## Value

Path to the downloaded file

---

`get_input_at` *Retrieve tensors for layers with multiple nodes*

---

## Description

Whenever you are calling a layer on some input, you are creating a new tensor (the output of the layer), and you are adding a "node" to the layer, linking the input tensor to the output tensor. When you are calling the same layer multiple times, that layer owns multiple nodes indexed as 1, 2, 3. These functions enable you to retreive various tensor properties of layers with multiple nodes.

## Usage

```
get_input_at(object, node_index)

get_output_at(object, node_index)

get_input_shape_at(object, node_index)

get_output_shape_at(object, node_index)

get_input_mask_at(object, node_index)

get_output_mask_at(object, node_index)
```

## Arguments

| | |
|---|---|
| `object` | Layer or model object |
| `node_index` | Integer, index of the node from which to retrieve the attribute. E.g. `node_index = 1` will correspond to the first time the layer was called. |

## Value

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

## See Also

Other layer methods: `count_params`, `get_config`, `get_weights`, `reset_states`

## get_layer *Retrieves a layer based on either its name (unique) or index.*

### Description

Indices are based on order of horizontal graph traversal (bottom-up) and are 0-based.

### Usage

```
get_layer(object, name = NULL, index = NULL)
```

### Arguments

| | |
|---|---|
| object | Keras model object |
| name | String, name of layer. |
| index | Integer, index of layer (0-based) |

### Value

A layer instance.

### See Also

Other model functions: compile, evaluate_generator, evaluate, fit_generator, fit, get_config, keras_model_sequential, keras_model, pop_layer, predict.keras.engine.training.Model, predict_generator, predict_on_batch, predict_proba, summary.keras.engine.training.Model, train_on_batch

## get_weights *Layer/Model weights as R arrays*

### Description

Layer/Model weights as R arrays

### Usage

```
get_weights(object)

set_weights(object, weights)
```

### Arguments

| | |
|---|---|
| object | Layer or model object |
| weights | Weights as R array |

## See Also

Other model persistence: model_to_json, model_to_yaml, save_model_hdf5, save_model_weights_hdf5

Other layer methods: count_params, get_config, get_input_at, reset_states

---

hdf5_matrix *Representation of HDF5 dataset to be used instead of an R array*

---

## Description

Representation of HDF5 dataset to be used instead of an R array

## Usage

```
hdf5_matrix(datapath, dataset, start = 0, end = NULL, normalizer = NULL)
```

## Arguments

| | |
|---|---|
| datapath | string, path to a HDF5 file |
| dataset | string, name of the HDF5 dataset in the file specified in datapath |
| start | int, start of desired slice of the specified dataset |
| end | int, end of desired slice of the specified dataset |
| normalizer | function to be called on data when retrieved |

## Details

Providing start and end allows use of a slice of the dataset.

Optionally, a normalizer function (or lambda) can be given. This will be called on every slice of data retrieved.

## Value

An array-like HDF5 dataset.

`imagenet_decode_predictions` *Decodes the prediction of an ImageNet model.*

## Description

Decodes the prediction of an ImageNet model.

## Usage

```
imagenet_decode_predictions(preds, top = 5)
```

## Arguments

| | |
|---|---|
| `preds` | Tensor encoding a batch of predictions. |
| `top` | integer, how many top-guesses to return. |

## Value

List of data frames with variables `class_name`, `class_description`, and `score` (one data frame per sample in batch input).

`imagenet_preprocess_input` *Preprocesses a tensor encoding a batch of images.*

## Description

Preprocesses a tensor encoding a batch of images.

## Usage

```
imagenet_preprocess_input(x)
```

## Arguments

| | |
|---|---|
| `x` | input tensor, 4D |

## Value

Preprocessed tensor

image_data_generator *Generate minibatches of image data with real-time data augmentation.*

## Description

Generate minibatches of image data with real-time data augmentation.

## Usage

```
image_data_generator(featurewise_center = FALSE, samplewise_center = FALSE,
  featurewise_std_normalization = FALSE,
  samplewise_std_normalization = FALSE, zca_whitening = FALSE,
  zca_epsilon = 1e-06, rotation_range = 0, width_shift_range = 0,
  height_shift_range = 0, shear_range = 0, zoom_range = 0,
  channel_shift_range = 0, fill_mode = "nearest", cval = 0,
  horizontal_flip = FALSE, vertical_flip = FALSE, rescale = NULL,
  preprocessing_function = NULL, data_format = NULL)
```

## Arguments

| | |
|---|---|
| featurewise_center | set input mean to 0 over the dataset. |
| samplewise_center | set each sample mean to 0. |
| featurewise_std_normalization | divide inputs by std of the dataset. |
| samplewise_std_normalization | divide each input by its std. |
| zca_whitening | apply ZCA whitening. |
| zca_epsilon | Epsilon for ZCA whitening. Default is 1e-6. |
| rotation_range | degrees (0 to 180). |
| width_shift_range | fraction of total width. |
| height_shift_range | fraction of total height. |
| shear_range | shear intensity (shear angle in radians). |
| zoom_range | amount of zoom. if scalar z, zoom will be randomly picked in the range `[1-z, 1+z]`. A sequence of two can be passed instead to select this range. |
| channel_shift_range | shift range for each channels. |
| fill_mode | points outside the boundaries are filled according to the given mode ('constant', 'nearest', 'reflect' or 'wrap'). Default is 'nearest'. |
| cval | value used for points outside the boundaries when fill_mode is 'constant'. Default is 0. |

horizontal_flip
: whether to randomly flip images horizontally.

vertical_flip
: whether to randomly flip images vertically.

rescale
: rescaling factor. If NULL or 0, no rescaling is applied, otherwise we multiply the data by the value provided (before applying any other transformation).

preprocessing_function
: function that will be implied on each input. The function will run before any other modification on it. The function should take one argument: one image (tensor with rank 3), and should output a tensor with the same shape.

data_format
: 'channels_first' or 'channels_last'. In 'channels_first' mode, the channels dimension (the depth) is at index 1, in 'channels_last' mode it is at index 3. 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".

---

`image_load` *Loads an image into PIL format.*

---

## Description

Loads an image into PIL format.

## Usage

```
image_load(path, grayscale = FALSE, target_size = NULL)
```

## Arguments

| | |
|---|---|
| `path` | Path to image file |
| `grayscale` | Boolean, whether to load the image as grayscale. |
| `target_size` | Either `NULL` (default to original size) or integer vector `(img_height, img_width)`. |

## Value

A PIL Image instance.

## See Also

Other image preprocessing: `fit_image_data_generator`, `flow_images_from_data`, `flow_images_from_directory`, `image_to_array`

## image_to_array *Converts a PIL Image instance to a 3d-array.*

### Description

Converts a PIL Image instance to a 3d-array.

### Usage

```
image_to_array(img, data_format = c("channels_last", "channels_first"))
```

### Arguments

| | |
|---|---|
| img | PIL Image instance. |
| data_format | Image data format ("channels_last" or "channels_first") |

### Value

A 3D array.

### See Also

Other image preprocessing: fit_image_data_generator, flow_images_from_data, flow_images_from_directory, image_load

## implementation *Keras implementation*

### Description

Obtain a reference to the Python module used for the implementation of Keras.

### Usage

```
implementation()
```

### Details

There are currently two Python modules which implement Keras:

- keras ("keras")
- tensorflow.contrib.keras ("tensorflow")

This function returns a reference to the implementation being currently used by the keras package. The default implementation is "tensorflow". You can override this by setting the KERAS_IMPLEMENTATION environment variable to "keras".

### Value

Reference to the Python module used for the implementation of Keras.

---

## initializer_constant *Initializer that generates tensors initialized to a constant value.*

---

### Description

Initializer that generates tensors initialized to a constant value.

### Usage

```
initializer_constant(value = 0)
```

### Arguments

| | |
|---|---|
| value | float; the value of the generator tensors. |

### See Also

Other initializers: initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

---

## initializer_glorot_normal *Glorot normal initializer, also called Xavier normal initializer.*

---

### Description

It draws samples from a truncated normal distribution centered on 0 with `stddev = sqrt(2 / (fan_in + fan_out))` where `fan_in` is the number of input units in the weight tensor and `fan_out` is the number of output units in the weight tensor.

### Usage

```
initializer_glorot_normal(seed = NULL)
```

### Arguments

| | |
|---|---|
| seed | Integer used to seed the random generator. |

## References

Glorot & Bengio, AISTATS 2010 http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

## See Also

Other initializers: initializer_constant, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

---

`initializer_glorot_uniform` *Glorot uniform initializer, also called Xavier uniform initializer.*

---

## Description

It draws samples from a uniform distribution within `-limit, limit` where `limit` is `sqrt(6 / (fan_in + fan_out))` where `fan_in` is the number of input units in the weight tensor and `fan_out` is the number of output units in the weight tensor.

## Usage

```
initializer_glorot_uniform(seed = NULL)
```

## Arguments

| | |
|---|---|
| `seed` | Integer used to seed the random generator. |

## References

Glorot & Bengio, AISTATS 2010 http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

## See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

## initializer_he_normal *He normal initializer.*

### Description

It draws samples from a truncated normal distribution centered on 0 with `stddev = sqrt(2 / fan_in)` where `fan_in` is the number of input units in the weight tensor.

### Usage

```
initializer_he_normal(seed = NULL)
```

### Arguments

| | |
|---|---|
| seed | Integer used to seed the random generator. |

### References

He et al., http://arxiv.org/abs/1502.01852

### See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

## initializer_he_uniform *He uniform variance scaling initializer.*

### Description

It draws samples from a uniform distribution within `-limit, limit` where `limit`` issqrt(6 / fan_in)wherefan_in‘ is the number of input units in the weight tensor.

### Usage

```
initializer_he_uniform(seed = NULL)
```

### Arguments

| | |
|---|---|
| seed | Integer used to seed the random generator. |

### References

He et al., http://arxiv.org/abs/1502.01852

## See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

---

# initializer_identity *Initializer that generates the identity matrix.*

---

## Description

Only use for square 2D matrices.

## Usage

```
initializer_identity(gain = 1)
```

## Arguments

| | |
|---|---|
| gain | Multiplicative factor to apply to the identity matrix |

## See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

---

# initializer_lecun_normal *LeCun normal initializer.*

---

## Description

It draws samples from a truncated normal distribution centered on 0 with `stddev <- sqrt(1 / fan_in)` where `fan_in` is the number of input units in the weight tensor..

## Usage

```
initializer_lecun_normal(seed = NULL)
```

## Arguments

| | |
|---|---|
| seed | A Python integer. Used to seed the random generator. |

## References

- Self-Normalizing Neural Networks
- Efficient Backprop

## See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

---

`initializer_lecun_uniform`

*LeCun uniform initializer.*

---

## Description

It draws samples from a uniform distribution within `-limit, limit` where `limit` is `sqrt(3 / fan_in)` where `fan_in` is the number of input units in the weight tensor.

## Usage

```
initializer_lecun_uniform(seed = NULL)
```

## Arguments

| | |
|---|---|
| seed | Integer used to seed the random generator. |

## References

LeCun 98, Efficient Backprop, http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

## See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

## initializer_ones *Initializer that generates tensors initialized to 1.*

### Description

Initializer that generates tensors initialized to 1.

### Usage

```
initializer_ones()
```

### See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_orthogonal, initializer_random_normal, initializer_random_unifor initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

## initializer_orthogonal *Initializer that generates a random orthogonal matrix.*

### Description

Initializer that generates a random orthogonal matrix.

### Usage

```
initializer_orthogonal(gain = 1, seed = NULL)
```

### Arguments

| | |
|---|---|
| gain | Multiplicative factor to apply to the orthogonal matrix. |
| seed | Integer used to seed the random generator. |

### References

Saxe et al., http://arxiv.org/abs/1312.6120

### See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

## initializer_random_normal

*Initializer that generates tensors with a normal distribution.*

### Description

Initializer that generates tensors with a normal distribution.

### Usage

```
initializer_random_normal(mean = 0, stddev = 0.05, seed = NULL)
```

### Arguments

| | |
|---|---|
| mean | Mean of the random values to generate. |
| stddev | Standard deviation of the random values to generate. |
| seed | Integer used to seed the random generator. |

### See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

## initializer_random_uniform

*Initializer that generates tensors with a uniform distribution.*

### Description

Initializer that generates tensors with a uniform distribution.

### Usage

```
initializer_random_uniform(minval = -0.05, maxval = 0.05, seed = NULL)
```

### Arguments

| | |
|---|---|
| minval | Lower bound of the range of random values to generate. |
| maxval | Upper bound of the range of random values to generate. Defaults to 1 for float types. |
| seed | seed |

## See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

---

`initializer_truncated_normal` *Initializer that generates a truncated normal distribution.*

---

## Description

These values are similar to values from an initializer_random_normal() except that values more than two standard deviations from the mean are discarded and re-drawn. This is the recommended initializer for neural network weights and filters.

## Usage

```
initializer_truncated_normal(mean = 0, stddev = 0.05, seed = NULL)
```

## Arguments

| | |
|---|---|
| mean | Mean of the random values to generate. |
| stddev | Standard deviation of the random values to generate. |
| seed | Integer used to seed the random generator. |

## See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_variance_scaling, initializer_zeros

---

`initializer_variance_scaling` *Initializer capable of adapting its scale to the shape of weights.*

---

## Description

With distribution="normal", samples are drawn from a truncated normal distribution centered on zero, with stddev = sqrt(scale / n) where n is:

- number of input units in the weight tensor, if mode = "fan_in"
- number of output units, if mode = "fan_out"
- average of the numbers of input and output units, if mode = "fan_avg"

### Usage

```
initializer_variance_scaling(scale = 1, mode = c("fan_in", "fan_out",
  "fan_avg"), distribution = c("normal", "uniform"), seed = NULL)
```

### Arguments

| | |
|---|---|
| scale | Scaling factor (positive float). |
| mode | One of "fan_in", "fan_out", "fan_avg". |
| distribution | One of "normal", "uniform" |
| seed | Integer used to seed the random generator. |

### Details

With `distribution="uniform"`, samples are drawn from a uniform distribution within `-limit, limit`, with `limit = sqrt(3 * scale / n)`.

### See Also

Other initializers: `initializer_constant`, `initializer_glorot_normal`, `initializer_glorot_uniform`, `initializer_he_normal`, `initializer_he_uniform`, `initializer_identity`, `initializer_lecun_normal`, `initializer_lecun_uniform`, `initializer_ones`, `initializer_orthogonal`, `initializer_random_normal`, `initializer_random_uniform`, `initializer_truncated_normal`, `initializer_zeros`

---

## initializer_zeros *Initializer that generates tensors initialized to 0.*

---

### Description

Initializer that generates tensors initialized to 0.

### Usage

```
initializer_zeros()
```

### See Also

Other initializers: `initializer_constant`, `initializer_glorot_normal`, `initializer_glorot_uniform`, `initializer_he_normal`, `initializer_he_uniform`, `initializer_identity`, `initializer_lecun_normal`, `initializer_lecun_uniform`, `initializer_ones`, `initializer_orthogonal`, `initializer_random_normal`, `initializer_random_uniform`, `initializer_truncated_normal`, `initializer_variance_scaling`

`KerasCallback` *Base R6 class for Keras callbacks*

## Description

Base R6 class for Keras callbacks

## Usage

```
KerasCallback
```

## Format

An R6Class generator object

## Details

The `logs` named list that callback methods take as argument will contain keys for quantities relevant to the current batch or epoch.

Currently, the `fit()` method for sequential models will include the following quantities in the `logs` that it passes to its callbacks:

- `on_epoch_end`: logs include `acc` and `loss`, and optionally include `val_loss` (if validation is enabled in `fit`), and `val_acc` (if validation and accuracy monitoring are enabled).
- `on_batch_begin`: logs include `size`, the number of samples in the current batch.
- `on_batch_end`: logs include `loss`, and optionally `acc` (if accuracy monitoring is enabled).

## Value

KerasCallback.

## Fields

`params` Named list with training parameters (eg. verbosity, batch size, number of epochs...).

`model` Reference to the Keras model being trained.

## Methods

`on_epoch_begin(epoch, logs)` Called at the beginning of each epoch.

`on_epoch_end(epoch, logs)` Called at the end of each epoch.

`on_batch_begin(batch, logs)` Called at the beginning of each batch.

`on_batch_end(batch, logs)` Called at the end of each batch.

`on_train_begin(logs)` Called at the beginning of training.

`on_train_end(logs)` Called at the end of training.

## Examples

```
## Not run:
library(keras)

LossHistory <- R6::R6Class("LossHistory",
  inherit = KerasCallback,

  public = list(

    losses = NULL,

    on_batch_end = function(batch, logs = list()) {
      self$losses <- c(self$losses, logs[["loss"]])
    }
  )
)

## End(Not run)
```

---

`KerasLayer` *Base R6 class for Keras layers*

---

## Description

Base R6 class for Keras layers

## Usage

```
KerasLayer
```

## Format

An R6Class generator object #'

## Value

KerasLayer.

## Methods

`build(input_shape)` Creates the layer weights (must be implemented by all layers that have weights)

`call(inputs,mask)` Call the layer on an input tensor.

`compute_output_shape(input_shape)` Compute the output shape for the layer.

`add_weight(name,shape,dtype,initializer,regularizer,trainable,constraint)` Adds a weight variable to the layer.

keras_model *Keras Model*

## Description

A model is a directed acyclic graph of layers.

## Usage

```
keras_model(inputs, outputs = NULL)
```

## Arguments

| | |
|---|---|
| inputs | Input layer |
| outputs | Output layer |

## See Also

Other model functions: compile, evaluate_generator, evaluate, fit_generator, fit, get_config, get_layer, keras_model_sequential, pop_layer, predict.keras.engine.training.Model, predict_generator, predict_on_batch, predict_proba, summary.keras.engine.training.Model, train_on_batch

## Examples

```
## Not run:
library(keras)

# input layer
inputs <- layer_input(shape = c(784))

# outputs compose input + dense layers
predictions <- inputs %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')

# create and compile model
model <- keras_model(inputs = inputs, outputs = predictions)
model %>% compile(
  optimizer = 'rmsprop',
  loss = 'categorical_crossentropy',
  metrics = c('accuracy')
)

## End(Not run)
```

keras_model_sequential *Keras Model composed of a linear stack of layers*

## Description

Keras Model composed of a linear stack of layers

## Usage

```
keras_model_sequential(layers = NULL, name = NULL)
```

## Arguments

| | |
|---|---|
| layers | List of layers to add to the model |
| name | Name of model |

## Note

The first layer passed to a Sequential model should have a defined input shape. What that means is that it should have received an `input_shape` or `batch_input_shape` argument, or for some type of layers (recurrent, Dense...) an `input_dim` argument.

## See Also

Other model functions: `compile`, `evaluate_generator`, `evaluate`, `fit_generator`, `fit`, `get_config`, `get_layer`, `keras_model`, `pop_layer`, `predict.keras.engine.training.Model`, `predict_generator`, `predict_on_batch`, `predict_proba`, `summary.keras.engine.training.Model`, `train_on_batch`

## Examples

```
## Not run:

library(keras)

model <- keras_model_sequential()
model %>%
  layer_dense(units = 32, input_shape = c(784)) %>%
  layer_activation('relu') %>%
  layer_dense(units = 10) %>%
  layer_activation('softmax')

model %>% compile(
  optimizer = 'rmsprop',
  loss = 'categorical_crossentropy',
  metrics = c('accuracy')
)

## End(Not run)
```

`layer_activation` *Apply an activation function to an output.*

## Description

Apply an activation function to an output.

## Usage

```
layer_activation(object, activation, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `activation` | Name of activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x). |
| `input_shape` | Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model. |
| `batch_input_shape` | Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors. |
| `batch_size` | Fixed batch size for layer |
| `dtype` | The data type expected by the input, as a string (`float32`, `float64`, `int32`...) |
| `name` | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| `trainable` | Whether the layer weights will be updated during training. |
| `weights` | Initial weights for layer. |

## See Also

Other core layers: `layer_activity_regularization`, `layer_dense`, `layer_dropout`, `layer_flatten`, `layer_input`, `layer_lambda`, `layer_masking`, `layer_permute`, `layer_repeat_vector`, `layer_reshape`

Other activation layers: `layer_activation_elu`, `layer_activation_leaky_relu`, `layer_activation_parametric_relu`, `layer_activation_thresholded_relu`

layer_activation_elu *Exponential Linear Unit.*

## Description

It follows: `f(x) = alpha * (exp(x) - 1.0)` for `x < 0`, `f(x) = x` for 'x = 0'.

## Usage

```
layer_activation_elu(object, alpha = 1, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `alpha` | Scale for the negative factor. |
| `input_shape` | Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model. |
| `batch_input_shape` | Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors. |
| `batch_size` | Fixed batch size for layer |
| `dtype` | The data type expected by the input, as a string (`float32`, `float64`, `int32`...) |
| `name` | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| `trainable` | Whether the layer weights will be updated during training. |
| `weights` | Initial weights for layer. |

## See Also

Fast and Accurate Deep Network Learning by Exponential Linear Units(ELUs).

Other activation layers: `layer_activation_leaky_relu`, `layer_activation_parametric_relu`, `layer_activation_thresholded_relu`, `layer_activation`

`layer_activation_leaky_relu` *Leaky version of a Rectified Linear Unit.*

## Description

Allows a small gradient when the unit is not active: `f(x) = alpha * x` for `x < 0`, `f(x) = x` for `x >= 0`.

## Usage

```
layer_activation_leaky_relu(object, alpha = 0.3, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `alpha` | float >= 0. Negative slope coefficient. |
| `input_shape` | Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model. |
| `batch_input_shape` | Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors. |
| `batch_size` | Fixed batch size for layer |
| `dtype` | The data type expected by the input, as a string (`float32`, `float64`, `int32`...) |
| `name` | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| `trainable` | Whether the layer weights will be updated during training. |
| `weights` | Initial weights for layer. |

## See Also

Rectifier Nonlinearities Improve Neural Network AcousticModels.

Other activation layers: `layer_activation_elu`, `layer_activation_parametric_relu`, `layer_activation_threshold`
`layer_activation`

layer_activation_parametric_relu *Parametric Rectified Linear Unit.*

## Description

It follows: `f(x) = alpha * x`` forx < 0, f(x) = xforx >= 0‘, where alpha is a learned array with the same shape as x.

## Usage

```
layer_activation_parametric_relu(object, alpha_initializer = "zeros",
  alpha_regularizer = NULL, alpha_constraint = NULL, shared_axes = NULL,
  input_shape = NULL, batch_input_shape = NULL, batch_size = NULL,
  dtype = NULL, name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `alpha_initializer` | Initializer function for the weights. |
| `alpha_regularizer` | Regularizer for the weights. |
| `alpha_constraint` | Constraint for the weights. |
| `shared_axes` | The axes along which to share learnable parameters for the activation function. For example, if the incoming feature maps are from a 2D convolution with output shape (batch, height, width, channels), and you wish to share parameters across space so that each filter only has one set of parameters, set shared_axes=c(1, 2). |
| `input_shape` | Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model. |
| `batch_input_shape` | Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors. |
| `batch_size` | Fixed batch size for layer |
| `dtype` | The data type expected by the input, as a string (`float32`, `float64`, `int32`...) |
| `name` | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| `trainable` | Whether the layer weights will be updated during training. |
| `weights` | Initial weights for layer. |

## See Also

Delving Deep into Rectifiers: Surpassing Human-Level Performance onImageNet Classification.

Other activation layers: layer_activation_elu, layer_activation_leaky_relu, layer_activation_thresholded_rel
layer_activation

---

`layer_activation_thresholded_relu` *Thresholded Rectified Linear Unit.*

---

## Description

It follows: `f(x) = x` for `x > theta`, `f(x) = 0` otherwise.

## Usage

```
layer_activation_thresholded_relu(object, theta = 1, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `theta` | float >= 0. Threshold location of activation. |
| `input_shape` | Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model. |
| `batch_input_shape` | Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors. |
| `batch_size` | Fixed batch size for layer |
| `dtype` | The data type expected by the input, as a string (`float32`, `float64`, `int32`...) |
| `name` | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| `trainable` | Whether the layer weights will be updated during training. |
| `weights` | Initial weights for layer. |

## See Also

Zero-bias autoencoders and the benefits of co-adapting features.

Other activation layers: layer_activation_elu, layer_activation_leaky_relu, layer_activation_parametric_relu
layer_activation

layer_activity_regularization *Layer that applies an update to the cost function based input activity.*

## Description

Layer that applies an update to the cost function based input activity.

## Usage

```
layer_activity_regularization(object, l1 = 0, l2 = 0, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| l1 | L1 regularization factor (positive float). |
| l2 | L2 regularization factor (positive float). |
| input_shape | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |
| batch_input_shape | Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors. |
| batch_size | Fixed batch size for layer |
| dtype | The data type expected by the input, as a string (float32, float64, int32...) |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

## Input shape

Arbitrary. Use the keyword argument input_shape (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

## Output shape

Same shape as input.

## See Also

Other core layers: layer_activation, layer_dense, layer_dropout, layer_flatten, layer_input, layer_lambda, layer_masking, layer_permute, layer_repeat_vector, layer_reshape

## layer_add *Layer that adds a list of inputs.*

### Description

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also of the same shape).

### Usage

```
layer_add(inputs)
```

### Arguments

| | |
|---|---|
| inputs | A list of input tensors (at least 2). |

### Value

A tensor, the sum of the inputs.

### See Also

Other merge layers: layer_average, layer_concatenate, layer_dot, layer_maximum, layer_multiply

## layer_alpha_dropout *Applies Alpha Dropout to the input.*

### Description

Alpha Dropout is a dropout that keeps mean and variance of inputs to their original values, in order to ensure the self-normalizing property even after this dropout.

### Usage

```
layer_alpha_dropout(object, rate, noise_shape = NULL, seed = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| rate | float, drop probability (as with layer_dropout()). The multiplicative noise will have standard deviation sqrt(rate / (1 - rate)). |
| noise_shape | Noise shape |
| seed | An integer to use as random seed. |

### Details

Alpha Dropout fits well to Scaled Exponential Linear Units by randomly setting activations to the negative saturation value.

### Input shape

Arbitrary. Use the keyword argument `input_shape` (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

### Output shape

Same shape as input.

### References

- Self-Normalizing Neural Networks

### See Also

Other noise layers: `layer_gaussian_dropout`, `layer_gaussian_noise`

---

## `layer_average` *Layer that averages a list of inputs.*

---

### Description

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also of the same shape).

### Usage

```
layer_average(inputs)
```

### Arguments

| | |
|---|---|
| `inputs` | A list of input tensors (at least 2). |

### Value

A tensor, the average of the inputs.

### See Also

Other merge layers: `layer_add`, `layer_concatenate`, `layer_dot`, `layer_maximum`, `layer_multiply`

layer_average_pooling_1d *Average pooling for temporal data.*

## Description

Average pooling for temporal data.

## Usage

```
layer_average_pooling_1d(object, pool_size = 2L, strides = NULL,
  padding = "valid", batch_size = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| pool_size | Integer, size of the max pooling windows. |
| strides | Integer, or NULL. Factor by which to downscale. E.g. 2 will halve the input. If NULL, it will default to pool_size. |
| padding | One of "valid" or "same" (case-insensitive). |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

## Input shape

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

## Output shape

3D tensor with shape: (batch_size, downsampled_steps, features).

## See Also

Other pooling layers: layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d

layer_average_pooling_2d *Average pooling operation for spatial data.*

## Description

Average pooling operation for spatial data.

## Usage

```
layer_average_pooling_2d(object, pool_size = c(2L, 2L), strides = NULL,
  padding = "valid", data_format = NULL, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| pool_size | integer or list of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions. |
| strides | Integer, list of 2 integers, or NULL. Strides values. If NULL, it will default to pool_size. |
| padding | One of "valid" or "same" (case-insensitive). |
| 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

## Input shape

- If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)
- If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)

## Output shape

- If data_format='channels_last': 4D tensor with shape: (batch_size, pooled_rows, pooled_cols, channels)
- If data_format='channels_first': 4D tensor with shape: (batch_size, channels, pooled_rows, pooled_cols

## See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d

---

`layer_average_pooling_3d` *Average pooling operation for 3D data (spatial or spatio-temporal).*

---

## Description

Average pooling operation for 3D data (spatial or spatio-temporal).

## Usage

```
layer_average_pooling_3d(object, pool_size = c(2L, 2L, 2L), strides = NULL,
  padding = "valid", data_format = NULL, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `pool_size` | list of 3 integers, factors by which to downscale (dim1, dim2, dim3). (2, 2, 2) will halve the size of the 3D input in each dimension. |
| `strides` | list of 3 integers, or NULL. Strides values. |
| `padding` | One of `"valid"` or `"same"` (case-insensitive). |
| `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, spatial_dim1, spatial_dim2, spatial_dim3, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, spatial_dim1, spatial_dim2` 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". |
| `batch_size` | Fixed batch size for layer |
| `name` | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| `trainable` | Whether the layer weights will be updated during training. |
| `weights` | Initial weights for layer. |

## Input shape

- If `data_format='channels_last'`: 5D tensor with shape: `(batch_size, spatial_dim1, spatial_dim2, spatial`
- If `data_format='channels_first'`: 5D tensor with shape: `(batch_size, channels, spatial_dim1, spatial_di`

### Output shape

- If `data_format='channels_last'`: 5D tensor with shape: `(batch_size, pooled_dim1, pooled_dim2, pooled_di`
- If `data_format='channels_first'`: 5D tensor with shape: `(batch_size, channels, pooled_dim1, pooled_dim2`

### See Also

Other pooling layers: `layer_average_pooling_1d`, `layer_average_pooling_2d`, `layer_global_average_pooling_1d`, `layer_global_average_pooling_2d`, `layer_global_average_pooling_3d`, `layer_global_max_pooling_1d`, `layer_global_max_pooling_2d`, `layer_global_max_pooling_3d`, `layer_max_pooling_1d`, `layer_max_pooling_2d`, `layer_max_pooling_3d`

---

## layer_batch_normalization

*Batch normalization layer (Ioffe and Szegedy, 2014).*

---

### Description

Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

### Usage

```
layer_batch_normalization(object, axis = -1L, momentum = 0.99,
  epsilon = 0.001, center = TRUE, scale = TRUE,
  beta_initializer = "zeros", gamma_initializer = "ones",
  moving_mean_initializer = "zeros", moving_variance_initializer = "ones",
  beta_regularizer = NULL, gamma_regularizer = NULL,
  beta_constraint = NULL, gamma_constraint = NULL, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `axis` | Integer, the axis that should be normalized (typically the features axis). For instance, after a `Conv2D` layer with `data_format="channels_first"`, set `axis=1` in `BatchNormalization`. |
| `momentum` | Momentum for the moving average. |
| `epsilon` | Small float added to variance to avoid dividing by zero. |
| `center` | If TRUE, add offset of `beta` to normalized tensor. If FALSE, `beta` is ignored. |
| `scale` | If TRUE, multiply by `gamma`. If FALSE, `gamma` is not used. When the next layer is linear (also e.g. `nn.relu`), this can be disabled since the scaling will be done by the next layer. |
| `beta_initializer` | Initializer for the beta weight. |

`gamma_initializer`
: Initializer for the gamma weight.

`moving_mean_initializer`
: Initializer for the moving mean.

`moving_variance_initializer`
: Initializer for the moving variance.

`beta_regularizer`
: Optional regularizer for the beta weight.

`gamma_regularizer`
: Optional regularizer for the gamma weight.

`beta_constraint`
: Optional constraint for the beta weight.

`gamma_constraint`
: Optional constraint for the gamma weight.

`input_shape`
: Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

`batch_input_shape`
: Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.

`batch_size`
: Fixed batch size for layer

`dtype`
: The data type expected by the input, as a string (`float32`, `float64`, `int32`...)

`name`
: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.

`trainable`
: Whether the layer weights will be updated during training.

`weights`
: Initial weights for layer.

## Input shape

Arbitrary. Use the keyword argument `input_shape` (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

## Output shape

Same shape as input.

## References

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

`layer_concatenate` *Layer that concatenates a list of inputs.*

## Description

It takes as input a list of tensors, all of the same shape expect for the concatenation axis, and returns a single tensor, the concatenation of all inputs.

## Usage

```
layer_concatenate(inputs, axis = -1L)
```

## Arguments

| | |
|---|---|
| `inputs` | A list of input tensors (at least 2). |
| `axis` | Concatenation axis. |

## Value

A tensor, the concatenation of the inputs alongside axis `axis`.

## See Also

Other merge layers: `layer_add`, `layer_average`, `layer_dot`, `layer_maximum`, `layer_multiply`

`layer_conv_1d` *1D convolution layer (e.g. temporal convolution).*

## Description

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. If `use_bias` is TRUE, a bias vector is created and added to the outputs. Finally, if `activation` is not `NULL`, it is applied to the outputs as well. When using this layer as the first layer in a model, provide an `input_shape` argument (list of integers or `NULL`, e.g. `(10, 128)` for sequences of 10 vectors of 128-dimensional vectors, or `(NULL, 128)` for variable-length sequences of 128-dimensional vectors.

## Usage

```
layer_conv_1d(object, filters, kernel_size, strides = 1L, padding = "valid",
  dilation_rate = 1L, activation = NULL, use_bias = TRUE,
  kernel_initializer = "glorot_uniform", bias_initializer = "zeros",
  kernel_regularizer = NULL, bias_regularizer = NULL,
  activity_regularizer = NULL, kernel_constraint = NULL,
  bias_constraint = NULL, input_shape = NULL, batch_input_shape = NULL,
  batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| filters | Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution). |
| kernel_size | An integer or list of a single integer, specifying the length of the 1D convolution window. |
| strides | An integer or list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1. |
| padding | One of "valid", "causal" or "same" (case-insensitive). "causal" results in causal (dilated) convolutions, e.g. output[t] does not depend on input[t+1:]. Useful when modeling temporal data where the model should not violate the temporal order. See WaveNet: A GenerativeModel for Raw Audio, section 2.1. |
| dilation_rate | an integer or list of a single integer, specifying the dilation rate to use for dilated convolution. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any strides value != 1. |
| 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 | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |
| batch_input_shape | Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors. |
| batch_size | Fixed batch size for layer |
| dtype | The data type expected by the input, as a string (float32, float64, int32...) |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |

| | |
|---|---|
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

3D tensor with shape: (batch_size, steps, input_dim)

### Output shape

3D tensor with shape: (batch_size, new_steps, filters) steps value might have changed due to padding or strides.

### See Also

Other convolutional layers: layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

---

## layer_conv_2d *2D convolution layer (e.g. spatial convolution over images).*

---

### Description

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is TRUE, a bias vector is created and added to the outputs. Finally, if activation is not NULL, it is applied to the outputs as well. When using this layer as the first layer in a model, provide the keyword argument input_shape (list of integers, does not include the sample axis), e.g. input_shape=c(128, 128, 3) for 128x128 RGB pictures in data_format="channels_last".

### Usage

```
layer_conv_2d(object, filters, kernel_size, strides = c(1L, 1L),
  padding = "valid", data_format = NULL, dilation_rate = c(1L, 1L),
  activation = NULL, use_bias = TRUE,
  kernel_initializer = "glorot_uniform", bias_initializer = "zeros",
  kernel_regularizer = NULL, bias_regularizer = NULL,
  activity_regularizer = NULL, kernel_constraint = NULL,
  bias_constraint = NULL, input_shape = NULL, batch_input_shape = NULL,
  batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| filters | Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution). |
| kernel_size | An integer or 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 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. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1. |
| padding | one of "valid" or "same" (case-insensitive). |
| 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". |
| dilation_rate | an integer or list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1. |
| 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 | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |
| batch_input_shape | Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. |

| | |
|---|---|
| | batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors. |
| batch_size | Fixed batch size for layer |
| dtype | The data type expected by the input, as a string (float32, float64, int32...) |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### 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.

### See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

---

## layer_conv_2d_transpose

*Transposed 2D convolution layer (sometimes called Deconvolution).*

---

### Description

The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution. When using this layer as the first layer in a model, provide the keyword argument input_shape (list of integers, does not include the sample axis), e.g. input_shape=c(128L, 128L, 3L) for 128x128 RGB pictures in data_format="channels_last".

## Usage

```
layer_conv_2d_transpose(object, filters, kernel_size, strides = c(1L, 1L),
  padding = "valid", data_format = NULL, activation = NULL,
  use_bias = TRUE, kernel_initializer = "glorot_uniform",
  bias_initializer = "zeros", kernel_regularizer = NULL,
  bias_regularizer = NULL, activity_regularizer = NULL,
  kernel_constraint = NULL, bias_constraint = NULL, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| filters | Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution). |
| kernel_size | An integer or 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 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. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1. |
| padding | one of "valid" or "same" (case-insensitive). |
| 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`
: Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

`batch_input_shape`
: Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.

`batch_size`
: Fixed batch size for layer

`dtype`
: The data type expected by the input, as a string (`float32`, `float64`, `int32`...)

`name`
: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.

`trainable`
: Whether the layer weights will be updated during training.

`weights`
: Initial weights for layer.

## Input shape

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

## Output shape

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

## References

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

## See Also

Other convolutional layers: `layer_conv_1d`, `layer_conv_2d`, `layer_conv_3d_transpose`, `layer_conv_3d`, `layer_conv_lstm_2d`, `layer_cropping_1d`, `layer_cropping_2d`, `layer_cropping_3d`, `layer_separable_conv_2d`, `layer_upsampling_1d`, `layer_upsampling_2d`, `layer_upsampling_3d`, `layer_zero_padding_1d`, `layer_zero_padding_2d`, `layer_zero_padding_3d`

layer_conv_3d *3D convolution layer (e.g. spatial convolution over volumes).*

## Description

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is TRUE, a bias vector is created and added to the outputs. Finally, if activation is not NULL, it is applied to the outputs as well. When using this layer as the first layer in a model, provide the keyword argument input_shape (list of integers, does not include the sample axis), e.g. input_shape=c(128L, 128L, 128L, 3L) for 128x128x128 volumes with a single channel, in data_format="channels_last".

## Usage

```
layer_conv_3d(object, filters, kernel_size, strides = c(1L, 1L, 1L),
  padding = "valid", data_format = NULL, dilation_rate = c(1L, 1L, 1L),
  activation = NULL, use_bias = TRUE,
  kernel_initializer = "glorot_uniform", bias_initializer = "zeros",
  kernel_regularizer = NULL, bias_regularizer = NULL,
  activity_regularizer = NULL, kernel_constraint = NULL,
  bias_constraint = NULL, input_shape = NULL, batch_input_shape = NULL,
  batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| filters | Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution). |
| kernel_size | An integer or list of 3 integers, specifying the depth, height, and width of the 3D convolution window. Can be a single integer to specify the same value for all spatial dimensions. |
| strides | An integer or list of 3 integers, specifying the strides of the convolution along each spatial dimension. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1. |
| padding | one of "valid" or "same" (case-insensitive). |
| 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, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2 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". |

dilation_rate — an integer or list of 3 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.

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 — Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape — Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size — Fixed batch size for layer

dtype — The data type expected by the input, as a string (float32, float64, int32...)

name — An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.

trainable — Whether the layer weights will be updated during training.

weights — Initial weights for layer.

## Input shape

5D tensor with shape: (samples, channels, conv_dim1, conv_dim2, conv_dim3) if data_format='channels_first' or 5D tensor with shape: (samples, conv_dim1, conv_dim2, conv_dim3, channels) if data_format='channels_last'.

## Output shape

5D tensor with shape: (samples, filters, new_conv_dim1, new_conv_dim2, new_conv_dim3) if data_format='channels_first' or 5D tensor with shape: (samples, new_conv_dim1, new_conv_dim2, new_conv_dim3, if data_format='channels_last'. new_conv_dim1, new_conv_dim2 and new_conv_dim3 values might have changed due to padding.

## See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

---

`layer_conv_3d_transpose` *Transposed 3D convolution layer (sometimes called Deconvolution).*

---

## Description

The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution.

## Usage

```
layer_conv_3d_transpose(object, filters, kernel_size, strides = c(1, 1, 1),
  padding = "valid", data_format = NULL, activation = NULL,
  use_bias = TRUE, kernel_initializer = "glorot_uniform",
  bias_initializer = "zeros", kernel_regularizer = NULL,
  bias_regularizer = NULL, activity_regularizer = NULL,
  kernel_constraint = NULL, bias_constraint = NULL, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `filters` | Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution). |
| `kernel_size` | An integer or list of 3 integers, specifying the width and height of the 3D convolution window. Can be a single integer to specify the same value for all spatial dimensions. |
| `strides` | An integer or list of 3 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. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1. |
| `padding` | one of `"valid"` or `"same"` (case-insensitive). |
| `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, depth, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, depth, 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 — Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape — Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size — Fixed batch size for layer

dtype — The data type expected by the input, as a string (`float32`, `float64`, `int32`...)

name — An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.

trainable — Whether the layer weights will be updated during training.

weights — Initial weights for layer.

## Details

When using this layer as the first layer in a model, provide the keyword argument `input_shape` (list of integers, does not include the sample axis), e.g. `input_shape = list(128, 128, 128, 3)` for a 128x128x128 volume with 3 channels if `data_format="channels_last"`.

## References

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

## See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

---

`layer_conv_lstm_2d` *Convolutional LSTM.*

---

## Description

It is similar to an LSTM layer, but the input transformations and recurrent transformations are both convolutional.

## Usage

```
layer_conv_lstm_2d(object, filters, kernel_size, strides = c(1L, 1L),
  padding = "valid", data_format = NULL, dilation_rate = c(1L, 1L),
  activation = "tanh", recurrent_activation = "hard_sigmoid",
  use_bias = TRUE, kernel_initializer = "glorot_uniform",
  recurrent_initializer = "orthogonal", bias_initializer = "zeros",
  unit_forget_bias = TRUE, kernel_regularizer = NULL,
  recurrent_regularizer = NULL, bias_regularizer = NULL,
  activity_regularizer = NULL, kernel_constraint = NULL,
  recurrent_constraint = NULL, bias_constraint = NULL,
  return_sequences = FALSE, go_backwards = FALSE, stateful = FALSE,
  dropout = 0, recurrent_dropout = 0, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL, input_shape = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `filters` | Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution). |
| `kernel_size` | An integer or list of n integers, specifying the dimensions of the convolution window. |
| `strides` | An integer or list of n integers, specifying the strides of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1. |
| `padding` | One of `"valid"` or `"same"` (case-insensitive). |
| `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, time, ..., channels)` while `channels_first` corresponds to inputs with shape `(batch, time, channels, ...)`. 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". |

`dilation_rate` An integer or list of n integers, specifying the dilation rate to use for dilated convolution. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any `strides` value != 1.

`activation` Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).

`recurrent_activation`
Activation function to use for the recurrent step.

`use_bias` Boolean, whether the layer uses a bias vector.

`kernel_initializer`
Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs..

`recurrent_initializer`
Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state..

`bias_initializer`
Initializer for the bias vector.

`unit_forget_bias`
Boolean. If TRUE, add 1 to the bias of the forget gate at initialization. Use in combination with `bias_initializer="zeros"`. This is recommended in Jozefowicz etal.

`kernel_regularizer`
Regularizer function applied to the `kernel` weights matrix.

`recurrent_regularizer`
Regularizer function applied to the `recurrent_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` weights matrix.

`recurrent_constraint`
Constraint function applied to the `recurrent_kernel` weights matrix.

`bias_constraint`
Constraint function applied to the bias vector.

`return_sequences`
Boolean. Whether to return the last output in the output sequence, or the full sequence.

`go_backwards` Boolean (default FALSE). If TRUE, rocess the input sequence backwards.

`stateful` Boolean (default FALSE). If TRUE, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.

`dropout` Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.

`recurrent_dropout`
Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

| | |
|---|---|
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |
| input_shape | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |

## Input shape

- if data_format='channels_first' 5D tensor with shape: `(samples,time, channels, rows, cols)`
  - if data_format='channels_last' 5D tensor with shape: `(samples,time, rows, cols, channels)`

## References

- Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting
  The current implementation does not include the feedback loop on the cells output

## See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

---

# layer_cropping_1d *Cropping layer for 1D input (e.g. temporal sequence).*

---

## Description

It crops along the time dimension (axis 1).

## Usage

```
layer_cropping_1d(object, cropping = c(1L, 1L), batch_size = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| cropping | int or list of int (length 2) How many units should be trimmed off at the beginning and end of the cropping dimension (axis 1). If a single int is provided, the same value will be used for both. |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

3D tensor with shape (batch, axis_to_crop, features)

### Output shape

3D tensor with shape (batch, cropped_axis, features)

### See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

---

## layer_cropping_2d *Cropping layer for 2D input (e.g. picture).*

---

### Description

It crops along spatial dimensions, i.e. width and height.

### Usage

```
layer_cropping_2d(object, cropping = list(c(0L, 0L), c(0L, 0L)),
  data_format = NULL, batch_size = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| cropping | int, or list of 2 ints, or list of 2 lists of 2 ints.<br>• If int: the same symmetric cropping is applied to width and height.<br>• If list of 2 ints: interpreted as two different symmetric cropping values for height and width: (symmetric_height_crop, symmetric_width_crop).<br>• If list of 2 lists of 2 ints: interpreted as ((top_crop, bottom_crop), (left_crop, right_crop)) |
| 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

4D tensor with shape:

- If `data_format` is `"channels_last"`: `(batch, rows, cols, channels)`
- If `data_format` is `"channels_first"`: `(batch, channels, rows, cols)`

### Output shape

4D tensor with shape:

- If `data_format` is `"channels_last"`: `(batch, cropped_rows, cropped_cols, channels)`
- If `data_format` is `"channels_first"`: `(batch, channels, cropped_rows, cropped_cols)`

### See Also

Other convolutional layers: `layer_conv_1d`, `layer_conv_2d_transpose`, `layer_conv_2d`, `layer_conv_3d_transpose`, `layer_conv_3d`, `layer_conv_lstm_2d`, `layer_cropping_1d`, `layer_cropping_3d`, `layer_separable_conv_2d`, `layer_upsampling_1d`, `layer_upsampling_2d`, `layer_upsampling_3d`, `layer_zero_padding_1d`, `layer_zero_padding_2d`, `layer_zero_padding_3d`

---

## `layer_cropping_3d` *Cropping layer for 3D data (e.g. spatial or spatio-temporal).*

---

### Description

Cropping layer for 3D data (e.g. spatial or spatio-temporal).

### Usage

```
layer_cropping_3d(object, cropping = list(c(1L, 1L), c(1L, 1L), c(1L, 1L)),
  data_format = NULL, batch_size = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

### Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `cropping` | int, or list of 3 ints, or list of 3 lists of 2 ints.<br>• If int: the same symmetric cropping is applied to width and height.<br>• If list of 3 ints: interpreted as two different symmetric cropping values for height and width: `(symmetric_dim1_crop, symmetric_dim2_crop, symmetric_dim3_crop)`.<br>• If list of 3 lists of 2 ints: interpreted as `((left_dim1_crop, right_dim1_crop), (left_dim2_cro` |
| `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, spatial_dim1, spatial_dim2, spatial_dim3, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, spatial_dim1, spatial_dim2` 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". |

| | |
|---|---|
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

5D tensor with shape:

- If data_format is "channels_last": (batch, first_axis_to_crop, second_axis_to_crop, third_axis_to_cr
- If data_format is "channels_first": (batch, depth, first_axis_to_crop, second_axis_to_crop, third_ax

### Output shape

5D tensor with shape:

- If data_format is "channels_last": (batch, first_cropped_axis, second_cropped_axis, third_cropped_ax
- If data_format is "channels_first": (batch, depth, first_cropped_axis, second_cropped_axis, third_cr

### See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

---

## layer_dense *Add a densely-connected NN layer to an output*

---

### Description

Implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is TRUE). Note: if the input to the layer has a rank greater than 2, then it is flattened prior to the initial dot product with kernel.

### Usage

```
layer_dense(object, units, activation = NULL, use_bias = TRUE,
  kernel_initializer = "glorot_uniform", bias_initializer = "zeros",
  kernel_regularizer = NULL, bias_regularizer = NULL,
  activity_regularizer = NULL, kernel_constraint = NULL,
  bias_constraint = NULL, input_shape = NULL, batch_input_shape = NULL,
  batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| units | Positive integer, dimensionality of the output space. |
| activation | Name of activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x). |
| use_bias | 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 weights matrix. |
| bias_constraint | Constraint function applied to the bias vector. |
| input_shape | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |
| batch_input_shape | Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors. |
| batch_size | Fixed batch size for layer |
| dtype | The data type expected by the input, as a string (float32, float64, int32...) |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

## Input and Output Shapes

Input shape: nD tensor with shape: (batch_size, ..., input_dim). The most common situation would be a 2D input with shape (batch_size, input_dim).

Output shape: nD tensor with shape: (batch_size, ..., units). For instance, for a 2D input with shape (batch_size, input_dim), the output would have shape (batch_size, unit).

## See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dropout, layer_flatten, layer_input, layer_lambda, layer_masking, layer_permute, layer_repeat_vector, layer_reshape

## layer_dot — *Layer that computes a dot product between samples in two tensors.*

### Description

Layer that computes a dot product between samples in two tensors.

### Usage

```
layer_dot(inputs, axes, normalize = FALSE)
```

### Arguments

| | |
|---|---|
| inputs | A list of input tensors (at least 2). |
| axes | Integer or list of integers, axis or axes along which to take the dot product. |
| normalize | Whether to L2-normalize samples along the dot product axis before taking the dot product. If set to TRUE, then the output of the dot product is the cosine proximity between the two samples. **kwargs: Standard layer keyword arguments. |

### Value

A tensor, the dot product of the samples from the inputs.

### See Also

Other merge layers: layer_add, layer_average, layer_concatenate, layer_maximum, layer_multiply

## layer_dropout — *Applies Dropout to the input.*

### Description

Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

### Usage

```
layer_dropout(object, rate, noise_shape = NULL, seed = NULL,
  batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| rate | float between 0 and 1. Fraction of the input units to drop. |
| noise_shape | 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch_size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use noise_shape=c(batch_size, 1, features). |
| seed | A Python integer to use as random seed. |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dense, layer_flatten, layer_input, layer_lambda, layer_masking, layer_permute, layer_repeat_vector, layer_reshape

Other dropout layers: layer_spatial_dropout_1d, layer_spatial_dropout_2d, layer_spatial_dropout_3d

---

## layer_embedding *Turns positive integers (indexes) into dense vectors of fixed size.*

---

### Description

For example, list(4L, 20L) -> list(c(0.25, 0.1), c(0.6, -0.2)) This layer can only be used as the first layer in a model.

### Usage

```
layer_embedding(object, input_dim, output_dim,
  embeddings_initializer = "uniform", embeddings_regularizer = NULL,
  activity_regularizer = NULL, embeddings_constraint = NULL,
  mask_zero = FALSE, input_length = NULL, batch_size = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| input_dim | int > 0. Size of the vocabulary, i.e. maximum integer index + 1. |
| output_dim | int >= 0. Dimension of the dense embedding. |
| embeddings_initializer | Initializer for the embeddings matrix. |

embeddings_regularizer
: Regularizer function applied to the embeddings matrix.

activity_regularizer
: activity_regularizer

embeddings_constraint
: Constraint function applied to the embeddings matrix.

mask_zero
: Whether or not the input value 0 is a special "padding" value that should be masked out. This is useful when using recurrent layers, which may take variable length inputs. If this is TRUE then all subsequent layers in the model need to support masking or an exception will be raised. If mask_zero is set to TRUE, as a consequence, index 0 cannot be used in the vocabulary (input_dim should equal size of vocabulary + 1).

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).

batch_size
: Fixed batch size for layer

name
: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.

trainable
: Whether the layer weights will be updated during training.

weights
: Initial weights for layer.

### Input shape

2D tensor with shape: (batch_size, sequence_length).

### Output shape

3D tensor with shape: (batch_size, sequence_length, output_dim).

### References

- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

---

## layer_flatten *Flattens an input*

---

### Description

Flatten a given input, does not affect the batch size.

### Usage

```
layer_flatten(object, batch_size = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dense, layer_dropout, layer_input, layer_lambda, layer_masking, layer_permute, layer_repeat_vector, layer_reshape

---

## layer_gaussian_dropout

*Apply multiplicative 1-centered Gaussian noise.*

---

### Description

As it is a regularization layer, it is only active at training time.

### Usage

```
layer_gaussian_dropout(object, rate, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| rate | float, drop probability (as with Dropout). The multiplicative noise will have standard deviation sqrt(rate / (1 - rate)). |
| input_shape | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |
| batch_input_shape | Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors. |
| batch_size | Fixed batch size for layer |
| dtype | The data type expected by the input, as a string (float32, float64, int32...) |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

Arbitrary. Use the keyword argument `input_shape` (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

### Output shape

Same shape as input.

### References

- Dropout: A Simple Way to Prevent Neural Networks from Overfitting Srivastava, Hinton, et al. 2014

### See Also

Other noise layers: `layer_alpha_dropout`, `layer_gaussian_noise`

---

## `layer_gaussian_noise` *Apply additive zero-centered Gaussian noise.*

---

### Description

This is useful to mitigate overfitting (you could see it as a form of random data augmentation). Gaussian Noise (GS) is a natural choice as corruption process for real valued inputs. As it is a regularization layer, it is only active at training time.

### Usage

```
layer_gaussian_noise(object, stddev, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `stddev` | float, standard deviation of the noise distribution. |
| `input_shape` | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |
| `batch_input_shape` | Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors. |
| `batch_size` | Fixed batch size for layer |
| `dtype` | The data type expected by the input, as a string (`float32`, `float64`, `int32`...) |

| | |
|---|---|
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

Arbitrary. Use the keyword argument input_shape (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

### Output shape

Same shape as input.

### See Also

Other noise layers: layer_alpha_dropout, layer_gaussian_dropout

---

## layer_global_average_pooling_1d

*Global average pooling operation for temporal data.*

---

### Description

Global average pooling operation for temporal data.

### Usage

```
layer_global_average_pooling_1d(object, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

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

### Output shape

2D tensor with shape: (batch_size, channels)

## See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d

---

# layer_global_average_pooling_2d

*Global average pooling operation for spatial data.*

---

## Description

Global average pooling operation for spatial data.

## Usage

```
layer_global_average_pooling_2d(object, data_format = NULL,
  batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

## Input shape

- If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)
- If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)

## Output shape

2D tensor with shape: (batch_size, channels)

## See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d

---

# layer_global_average_pooling_3d

*Global Average pooling operation for 3D data.*

---

## Description

Global Average pooling operation for 3D data.

## Usage

```
layer_global_average_pooling_3d(object, data_format = NULL,
  batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| 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, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

## Input shape

- If data_format='channels_last': 5D tensor with shape: (batch_size, spatial_dim1, spatial_dim2, spatial
- If data_format='channels_first': 5D tensor with shape: (batch_size, channels, spatial_dim1, spatial_di

## Output shape

2D tensor with shape: (batch_size, channels)

## See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d

---

# layer_global_max_pooling_1d

*Global max pooling operation for temporal data.*

---

## Description

Global max pooling operation for temporal data.

## Usage

```
layer_global_max_pooling_1d(object, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

## Input shape

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

## Output shape

2D tensor with shape: (batch_size, channels)

## See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d

`layer_global_max_pooling_2d` *Global max pooling operation for spatial data.*

## Description

Global max pooling operation for spatial data.

## Usage

```
layer_global_max_pooling_2d(object, data_format = NULL, batch_size = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `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". |
| `batch_size` | Fixed batch size for layer |
| `name` | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| `trainable` | Whether the layer weights will be updated during training. |
| `weights` | Initial weights for layer. |

## Input shape

- If `data_format='channels_last'`: 4D tensor with shape: `(batch_size, rows, cols, channels)`
- If `data_format='channels_first'`: 4D tensor with shape: `(batch_size, channels, rows, cols)`

## Output shape

2D tensor with shape: `(batch_size, channels)`

## See Also

Other pooling layers: `layer_average_pooling_1d`, `layer_average_pooling_2d`, `layer_average_pooling_3d`, `layer_global_average_pooling_1d`, `layer_global_average_pooling_2d`, `layer_global_average_pooling_3d`, `layer_global_max_pooling_1d`, `layer_global_max_pooling_3d`, `layer_max_pooling_1d`, `layer_max_pooling_2d`, `layer_max_pooling_3d`

layer_global_max_pooling_3d *Global Max pooling operation for 3D data.*

## Description

Global Max pooling operation for 3D data.

## Usage

```
layer_global_max_pooling_3d(object, data_format = NULL, batch_size = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| 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, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

## Input shape

- If data_format='channels_last': 5D tensor with shape: (batch_size, spatial_dim1, spatial_dim2, spatial
- If data_format='channels_first': 5D tensor with shape: (batch_size, channels, spatial_dim1, spatial_di

## Output shape

2D tensor with shape: (batch_size, channels)

## See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d

layer_gru *Gated Recurrent Unit - Cho et al.*

## Description

Gated Recurrent Unit - Cho et al.

## Usage

```
layer_gru(object, units, activation = "tanh",
  recurrent_activation = "hard_sigmoid", use_bias = TRUE,
  return_sequences = FALSE, go_backwards = FALSE, stateful = FALSE,
  unroll = FALSE, implementation = 0L,
  kernel_initializer = "glorot_uniform",
  recurrent_initializer = "orthogonal", bias_initializer = "zeros",
  kernel_regularizer = NULL, recurrent_regularizer = NULL,
  bias_regularizer = NULL, activity_regularizer = NULL,
  kernel_constraint = NULL, recurrent_constraint = NULL,
  bias_constraint = NULL, dropout = 0, recurrent_dropout = 0,
  input_shape = NULL, batch_input_shape = NULL, batch_size = NULL,
  dtype = NULL, name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| units | Positive integer, dimensionality of the output space. |
| activation | Activation function to use. If you pass NULL, no activation is applied (ie. "linear" activation: a(x) = x). |
| recurrent_activation | Activation function to use for the recurrent step. |
| use_bias | Boolean, whether the layer uses a bias vector. |
| return_sequences | Boolean. Whether to return the last output in the output sequence, or the full sequence. |
| go_backwards | Boolean (default FALSE). If TRUE, process the input sequence backwards and return the reversed sequence. |
| stateful | Boolean (default FALSE). If TRUE, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch. |
| unroll | Boolean (default FALSE). If TRUE, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences. |
| implementation | one of 0, 1, or 2. If set to 0, the RNN will use an implementation that uses fewer, larger matrix products, thus running faster on CPU but consuming more memory. If set to 1, the RNN will use more matrix products, but smaller ones, thus |

running slower (may actually be faster on GPU) while consuming less memory. If set to 2 (LSTM/GRU only), the RNN will combine the input gate, the forget gate and the output gate into a single matrix, enabling more time-efficient parallelization on the GPU.

`kernel_initializer`
: Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs..

`recurrent_initializer`
: Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state..

`bias_initializer`
: Initializer for the bias vector.

`kernel_regularizer`
: Regularizer function applied to the `kernel` weights matrix.

`recurrent_regularizer`
: Regularizer function applied to the `recurrent_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` weights matrix.

`recurrent_constraint`
: Constraint function applied to the `recurrent_kernel` weights matrix.

`bias_constraint`
: Constraint function applied to the bias vector.

`dropout`
: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.

`recurrent_dropout`
: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

`input_shape`
: Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

`batch_input_shape`
: Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.

`batch_size`
: Fixed batch size for layer

`dtype`
: The data type expected by the input, as a string (`float32`, `float64`, `int32`...)

`name`
: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.

`trainable`
: Whether the layer weights will be updated during training.

`weights`
: Initial weights for layer.

### Statefulness in RNNs

You can set RNN layers to be 'stateful', which means that the states computed for the samples in one batch will be reused as initial states for the samples in the next batch. This assumes a one-to-one mapping between samples in different successive batches.

To enable statefulness:

- Specify `stateful=TRUE` in the layer constructor.
- Specify a fixed batch size for your model. For sequential models, pass `batch_input_shape = c(...)` to the first layer in your model. For functional models with 1 or more Input layers, pass `batch_shape = c(...)` to all the first layers in your model. This is the expected shape of your inputs *including the batch size*. It should be a vector of integers, e.g. `c(32, 10, 100)`.
- Specify `shuffle = FALSE` when calling fit().

To reset the states of your model, call `reset_states()` on either a specific layer, or on your entire model.

### Initial State of RNNs

You can specify the initial state of RNN layers symbolically by calling them with the keyword argument `initial_state`. The value of `initial_state` should be a tensor or list of tensors representing the initial state of the RNN layer.

You can specify the initial state of RNN layers numerically by calling `reset_states` with the keyword argument `states`. The value of `states` should be a numpy array or list of numpy arrays representing the initial state of the RNN layer.

### References

- On the Properties of Neural Machine Translation:Encoder-Decoder Approaches
- EmpiricalEvaluation of Gated Recurrent Neural Networks on SequenceModeling
- A Theoretically GroundedApplication of Dropout in Recurrent NeuralNetworks

### See Also

Other recurrent layers: `layer_lstm`, `layer_simple_rnn`

---

## `layer_input` *Input layer*

---

### Description

Layer to be used as an entry point into a graph.

### Usage

```
layer_input(shape = NULL, batch_shape = NULL, name = NULL, dtype = NULL,
  sparse = FALSE, tensor = NULL)
```

### Arguments

| | |
|---|---|
| shape | Shape, not including the batch size. For instance, shape=c(32) indicates that the expected input will be batches of 32-dimensional vectors. |
| batch_shape | Shapes, including the batch size. For instance, batch_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors. |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| dtype | The data type expected by the input, as a string (float32, float64, int32...) |
| sparse | Boolean, whether the placeholder created is meant to be sparse. |
| tensor | Existing tensor to wrap into the Input layer. If set, the layer will not create a placeholder tensor. |

### Details

It can either wrap an existing tensor (pass an input_tensor argument) or create its a placeholder tensor (pass arguments input_shape or batch_input_shape as well as input_dtype).

### Value

A tensor

### See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dense, layer_dropout, layer_flatten, layer_lambda, layer_masking, layer_permute, layer_repeat_vector, layer_reshape

---

## layer_lambda *Wraps arbitrary expression as a layer*

---

### Description

Wraps arbitrary expression as a layer

### Usage

```
layer_lambda(object, f, output_shape = NULL, mask = NULL,
  arguments = NULL, input_shape = NULL, batch_input_shape = NULL,
  batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| f | The function to be evaluated. Takes input tensor as first argument. |
| output_shape | Expected output shape from the function (not required when using TensorFlow back-end). |
| mask | mask |
| arguments | optional named list of keyword arguments to be passed to the function. |
| input_shape | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |
| batch_input_shape | Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors. |
| batch_size | Fixed batch size for layer |
| dtype | The data type expected by the input, as a string (`float32`, `float64`, `int32`...) |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

Arbitrary. Use the keyword argument input_shape (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

### Output shape

Arbitrary (based on tensor returned from the function)

### See Also

Other core layers: `layer_activation`, `layer_activity_regularization`, `layer_dense`, `layer_dropout`, `layer_flatten`, `layer_input`, `layer_masking`, `layer_permute`, `layer_repeat_vector`, `layer_reshape`

---

## `layer_locally_connected_1d` *Locally-connected layer for 1D inputs.*

---

### Description

`layer_locally_connected_1d()` works similarly to `layer_conv_1d()` , except that weights are unshared, that is, a different set of filters is applied at each different patch of the input.

## Usage

```
layer_locally_connected_1d(object, filters, kernel_size, strides = 1L,
  padding = "valid", data_format = NULL, activation = NULL,
  use_bias = TRUE, kernel_initializer = "glorot_uniform",
  bias_initializer = "zeros", kernel_regularizer = NULL,
  bias_regularizer = NULL, activity_regularizer = NULL,
  kernel_constraint = NULL, bias_constraint = NULL, batch_size = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `filters` | Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution). |
| `kernel_size` | An integer or list of a single integer, specifying the length of the 1D convolution window. |
| `strides` | An integer or list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1. |
| `padding` | Currently only supports `"valid"` (case-insensitive). `"same"` may be supported in the 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. |
| `batch_size` | Fixed batch size for layer |

| | |
|---|---|
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

3D tensor with shape: (batch_size, steps, input_dim)

### Output shape

3D tensor with shape: (batch_size, new_steps, filters) steps value might have changed due to padding or strides.

### See Also

Other locally connected layers: layer_locally_connected_2d

---

## layer_locally_connected_2d

*Locally-connected layer for 2D inputs.*

---

### Description

layer_locally_connected_2d works similarly to layer_conv_2d(), except that weights are unshared, that is, a different set of filters is applied at each different patch of the input.

### Usage

```
layer_locally_connected_2d(object, filters, kernel_size, strides = c(1L, 1L),
  padding = "valid", data_format = NULL, activation = NULL,
  use_bias = TRUE, kernel_initializer = "glorot_uniform",
  bias_initializer = "zeros", kernel_regularizer = NULL,
  bias_regularizer = NULL, activity_regularizer = NULL,
  kernel_constraint = NULL, bias_constraint = NULL, batch_size = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| filters | Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution). |
| kernel_size | An integer or 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 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. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.

padding — Currently only supports "valid" (case-insensitive). "same" may be supported in the 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, width, height, channels) while channels_first corresponds to inputs with shape (batch, channels, width, height). 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.

batch_size — Fixed batch size for layer

name — An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.

trainable — Whether the layer weights will be updated during training.

weights — Initial weights for layer.

## 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.

## See Also

Other locally connected layers: layer_locally_connected_1d

---

layer_lstm *Long-Short Term Memory unit - Hochreiter 1997.*

---

## Description

For a step-by-step description of the algorithm, see thistutorial.

## Usage

```
layer_lstm(object, units, activation = "tanh",
  recurrent_activation = "hard_sigmoid", use_bias = TRUE,
  return_sequences = FALSE, go_backwards = FALSE, stateful = FALSE,
  unroll = FALSE, implementation = 0L,
  kernel_initializer = "glorot_uniform",
  recurrent_initializer = "orthogonal", bias_initializer = "zeros",
  unit_forget_bias = TRUE, kernel_regularizer = NULL,
  recurrent_regularizer = NULL, bias_regularizer = NULL,
  activity_regularizer = NULL, kernel_constraint = NULL,
  recurrent_constraint = NULL, bias_constraint = NULL, dropout = 0,
  recurrent_dropout = 0, input_shape = NULL, batch_input_shape = NULL,
  batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| units | Positive integer, dimensionality of the output space. |
| activation | Activation function to use. If you pass NULL, no activation is applied (ie. "linear" activation: a(x) = x). |
| recurrent_activation | Activation function to use for the recurrent step. |
| use_bias | Boolean, whether the layer uses a bias vector. |
| return_sequences | Boolean. Whether to return the last output in the output sequence, or the full sequence. |
| go_backwards | Boolean (default FALSE). If TRUE, process the input sequence backwards and return the reversed sequence. |
| stateful | Boolean (default FALSE). If TRUE, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch. |
| unroll | Boolean (default FALSE). If TRUE, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences. |

`implementation` one of 0, 1, or 2. If set to 0, the RNN will use an implementation that uses fewer, larger matrix products, thus running faster on CPU but consuming more memory. If set to 1, the RNN will use more matrix products, but smaller ones, thus running slower (may actually be faster on GPU) while consuming less memory. If set to 2 (LSTM/GRU only), the RNN will combine the input gate, the forget gate and the output gate into a single matrix, enabling more time-efficient parallelization on the GPU.

`kernel_initializer`
Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs..

`recurrent_initializer`
Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state..

`bias_initializer`
Initializer for the bias vector.

`unit_forget_bias`
Boolean. If TRUE, add 1 to the bias of the forget gate at initialization. Setting it to true will also force `bias_initializer="zeros"`. This is recommended in Jozefowicz etal.

`kernel_regularizer`
Regularizer function applied to the `kernel` weights matrix.

`recurrent_regularizer`
Regularizer function applied to the `recurrent_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` weights matrix.

`recurrent_constraint`
Constraint function applied to the `recurrent_kernel` weights matrix.

`bias_constraint`
Constraint function applied to the bias vector.

`dropout` Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.

`recurrent_dropout`
Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

`input_shape` Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

`batch_input_shape`
Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.

`batch_size` Fixed batch size for layer

| | |
|---|---|
| dtype | The data type expected by the input, as a string (float32, float64, int32...) |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Statefulness in RNNs

You can set RNN layers to be 'stateful', which means that the states computed for the samples in one batch will be reused as initial states for the samples in the next batch. This assumes a one-to-one mapping between samples in different successive batches.

To enable statefulness:

- Specify stateful=TRUE in the layer constructor.
- Specify a fixed batch size for your model. For sequential models, pass batch_input_shape = c(...) to the first layer in your model. For functional models with 1 or more Input layers, pass batch_shape = c(...) to all the first layers in your model. This is the expected shape of your inputs *including the batch size*. It should be a vector of integers, e.g. c(32, 10, 100).
- Specify shuffle = FALSE when calling fit().

To reset the states of your model, call reset_states() on either a specific layer, or on your entire model.

### Initial State of RNNs

You can specify the initial state of RNN layers symbolically by calling them with the keyword argument initial_state. The value of initial_state should be a tensor or list of tensors representing the initial state of the RNN layer.

You can specify the initial state of RNN layers numerically by calling reset_states with the keyword argument states. The value of states should be a numpy array or list of numpy arrays representing the initial state of the RNN layer.

### References


- Long short-term memory (original 1997 paper)
- Supervised sequence labeling with recurrent neural networks
- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks


### See Also

Other recurrent layers: layer_gru, layer_simple_rnn

Other recurrent layers: layer_gru, layer_simple_rnn

`layer_masking` *Masks a sequence by using a mask value to skip timesteps.*

## Description

For each timestep in the input tensor (dimension #1 in the tensor), if all values in the input tensor at that timestep are equal to `mask_value`, then the timestep will be masked (skipped) in all downstream layers (as long as they support masking). If any downstream layer does not support masking yet receives such an input mask, an exception will be raised.

## Usage

```
layer_masking(object, mask_value = 0, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `mask_value` | float, mask value |
| `input_shape` | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |
| `batch_input_shape` | Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors. |
| `batch_size` | Fixed batch size for layer |
| `dtype` | The data type expected by the input, as a string (`float32`, `float64`, `int32`...) |
| `name` | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| `trainable` | Whether the layer weights will be updated during training. |
| `weights` | Initial weights for layer. |

## See Also

Other core layers: `layer_activation`, `layer_activity_regularization`, `layer_dense`, `layer_dropout`, `layer_flatten`, `layer_input`, `layer_lambda`, `layer_permute`, `layer_repeat_vector`, `layer_reshape`

## layer_maximum *Layer that computes the maximum (element-wise) a list of inputs.*

### Description

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also of the same shape).

### Usage

```
layer_maximum(inputs)
```

### Arguments

| | |
|---|---|
| inputs | A list of input tensors (at least 2). |

### Value

A tensor, the element-wise maximum of the inputs.

### See Also

Other merge layers: layer_add, layer_average, layer_concatenate, layer_dot, layer_multiply

## layer_max_pooling_1d *Max pooling operation for temporal data.*

### Description

Max pooling operation for temporal data.

### Usage

```
layer_max_pooling_1d(object, pool_size = 2L, strides = NULL,
  padding = "valid", batch_size = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| pool_size | Integer, size of the max pooling windows. |
| strides | Integer, or NULL. Factor by which to downscale. E.g. 2 will halve the input. If NULL, it will default to pool_size. |
| padding | One of "valid" or "same" (case-insensitive). |
| batch_size | Fixed batch size for layer |

| | |
|---|---|
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

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

### Output shape

3D tensor with shape: (batch_size, downsampled_steps, features).

### See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_2d, layer_max_pooling_3d

---

## layer_max_pooling_2d *Max pooling operation for spatial data.*

---

### Description

Max pooling operation for spatial data.

### Usage

```
layer_max_pooling_2d(object, pool_size = c(2L, 2L), strides = NULL,
  padding = "valid", data_format = NULL, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| pool_size | integer or list of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions. |
| strides | Integer, list of 2 integers, or NULL. Strides values. If NULL, it will default to pool_size. |
| padding | One of "valid" or "same" (case-insensitive). |

| | |
|---|---|
| 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

## Input shape

- If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)
- If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)

## Output shape

- If data_format='channels_last': 4D tensor with shape: (batch_size, pooled_rows, pooled_cols, channels)
- If data_format='channels_first': 4D tensor with shape: (batch_size, channels, pooled_rows, pooled_cols

## See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_3d

---

# layer_max_pooling_3d *Max pooling operation for 3D data (spatial or spatio-temporal).*

---

## Description

Max pooling operation for 3D data (spatial or spatio-temporal).

## Usage

```
layer_max_pooling_3d(object, pool_size = c(2L, 2L, 2L), strides = NULL,
  padding = "valid", data_format = NULL, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| pool_size | list of 3 integers, factors by which to downscale (dim1, dim2, dim3). (2, 2, 2) will halve the size of the 3D input in each dimension. |
| strides | list of 3 integers, or NULL. Strides values. |
| padding | One of "valid" or "same" (case-insensitive). |
| 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, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

- If data_format='channels_last': 5D tensor with shape: (batch_size, spatial_dim1, spatial_dim2, spatial
- If data_format='channels_first': 5D tensor with shape: (batch_size, channels, spatial_dim1, spatial_di

### Output shape

- If data_format='channels_last': 5D tensor with shape: (batch_size, pooled_dim1, pooled_dim2, pooled_di
- If data_format='channels_first': 5D tensor with shape: (batch_size, channels, pooled_dim1, pooled_dim2

### See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_2d

---

## layer_multiply *Layer that multiplies (element-wise) a list of inputs.*

---

### Description

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also of the same shape).

### Usage

```
layer_multiply(inputs)
```

### Arguments

| | |
|---|---|
| inputs | A list of input tensors (at least 2). |

### Value

A tensor, the element-wise product of the inputs.

### See Also

Other merge layers: layer_add, layer_average, layer_concatenate, layer_dot, layer_maximum

---

## layer_permute *Permute the dimensions of an input according to a given pattern*

---

### Description

Permute the dimensions of an input according to a given pattern

### Usage

```
layer_permute(object, dims, input_shape = NULL, batch_input_shape = NULL,
  batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL,
  weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| dims | List of integers. Permutation pattern, does not include the samples dimension. Indexing starts at 1. For instance, (2, 1) permutes the first and second dimension of the input. |
| input_shape | Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model. |
| batch_input_shape | Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors. |
| batch_size | Fixed batch size for layer |
| dtype | The data type expected by the input, as a string (float32, float64, int32...) |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input and Output Shapes

Input shape: Arbitrary

Output shape: Same as the input shape, but with the dimensions re-ordered according to the specified pattern.

### Note

Useful for e.g. connecting RNNs and convnets together.

### See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dense, layer_dropout, layer_flatten, layer_input, layer_lambda, layer_masking, layer_repeat_vector, layer_reshape

---

## layer_repeat_vector *Repeats the input n times.*

---

### Description

Repeats the input n times.

### Usage

```
layer_repeat_vector(object, n, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| n | integer, repetition factor. |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

2D tensor of shape (num_samples, features).

### Output shape

3D tensor of shape (num_samples, n, features).

### See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dense, layer_dropout, layer_flatten, layer_input, layer_lambda, layer_masking, layer_permute, layer_reshape

`layer_reshape` *Reshapes an output to a certain shape.*

## Description

Reshapes an output to a certain shape.

## Usage

```
layer_reshape(object, target_shape, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `target_shape` | List of integers, does not include the samples dimension (batch size). |
| `input_shape` | Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model. |
| `batch_input_shape` | Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors. |
| `batch_size` | Fixed batch size for layer |
| `dtype` | The data type expected by the input, as a string (`float32`, `float64`, `int32`...) |
| `name` | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| `trainable` | Whether the layer weights will be updated during training. |
| `weights` | Initial weights for layer. |

## Input and Output Shapes

Input shape: Arbitrary, although all dimensions in the input shaped must be fixed.

Output shape: `(batch_size,) + target_shape`.

## See Also

Other core layers: `layer_activation`, `layer_activity_regularization`, `layer_dense`, `layer_dropout`, `layer_flatten`, `layer_input`, `layer_lambda`, `layer_masking`, `layer_permute`, `layer_repeat_vector`

layer_separable_conv_2d *Depthwise separable 2D convolution.*

## Description

Separable convolutions consist in first performing a depthwise spatial convolution (which acts on each input channel separately) followed by a pointwise convolution which mixes together the resulting output channels. The depth_multiplier argument controls how many output channels are generated per input channel in the depthwise step. Intuitively, separable convolutions can be understood as a way to factorize a convolution kernel into two smaller kernels, or as an extreme version of an Inception block.

## Usage

```
layer_separable_conv_2d(object, filters, kernel_size, strides = c(1L, 1L),
  padding = "valid", data_format = NULL, depth_multiplier = 1L,
  activation = NULL, use_bias = TRUE,
  depthwise_initializer = "glorot_uniform",
  pointwise_initializer = "glorot_uniform", bias_initializer = "zeros",
  depthwise_regularizer = NULL, pointwise_regularizer = NULL,
  bias_regularizer = NULL, activity_regularizer = NULL,
  depthwise_constraint = NULL, pointwise_constraint = NULL,
  bias_constraint = NULL, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| filters | Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution). |
| kernel_size | An integer or 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 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. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1. |
| padding | one of "valid" or "same" (case-insensitive). |
| 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". |

depth_multiplier
: The number of depthwise convolution output channels for each input channel. The total number of depthwise convolution output channels will be equal to `filterss_in * depth_multiplier`.

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.

depthwise_initializer
: Initializer for the depthwise kernel matrix.

pointwise_initializer
: Initializer for the pointwise kernel matrix.

bias_initializer
: Initializer for the bias vector.

depthwise_regularizer
: Regularizer function applied to the depthwise kernel matrix.

pointwise_regularizer
: Regularizer function applied to the depthwise kernel matrix.

bias_regularizer
: Regularizer function applied to the bias vector.

activity_regularizer
: Regularizer function applied to the output of the layer (its "activation")..

depthwise_constraint
: Constraint function applied to the depthwise kernel matrix.

pointwise_constraint
: Constraint function applied to the pointwise kernel matrix.

bias_constraint
: Constraint function applied to the bias vector.

batch_size
: Fixed batch size for layer

name
: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.

trainable
: Whether the layer weights will be updated during training.

weights
: Initial weights for layer.

## Input shape

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

## Output shape

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

## See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

---

`layer_simple_rnn` *Fully-connected RNN where the output is to be fed back to input.*

---

## Description

Fully-connected RNN where the output is to be fed back to input.

## Usage

```
layer_simple_rnn(object, units, activation = "tanh", use_bias = TRUE,
  return_sequences = FALSE, go_backwards = FALSE, stateful = FALSE,
  unroll = FALSE, implementation = 0L,
  kernel_initializer = "glorot_uniform",
  recurrent_initializer = "orthogonal", bias_initializer = "zeros",
  kernel_regularizer = NULL, recurrent_regularizer = NULL,
  bias_regularizer = NULL, activity_regularizer = NULL,
  kernel_constraint = NULL, recurrent_constraint = NULL,
  bias_constraint = NULL, dropout = 0, recurrent_dropout = 0,
  input_shape = NULL, batch_input_shape = NULL, batch_size = NULL,
  dtype = NULL, name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `units` | Positive integer, dimensionality of the output space. |
| `activation` | Activation function to use. If you pass `NULL`, no activation is applied (ie. "linear" activation: `a(x) = x`). |
| `use_bias` | Boolean, whether the layer uses a bias vector. |
| `return_sequences` | Boolean. Whether to return the last output in the output sequence, or the full sequence. |
| `go_backwards` | Boolean (default FALSE). If TRUE, process the input sequence backwards and return the reversed sequence. |
| `stateful` | Boolean (default FALSE). If TRUE, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch. |
| `unroll` | Boolean (default FALSE). If TRUE, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences. |

`implementation` one of 0, 1, or 2. If set to 0, the RNN will use an implementation that uses fewer, larger matrix products, thus running faster on CPU but consuming more memory. If set to 1, the RNN will use more matrix products, but smaller ones, thus running slower (may actually be faster on GPU) while consuming less memory. If set to 2 (LSTM/GRU only), the RNN will combine the input gate, the forget gate and the output gate into a single matrix, enabling more time-efficient parallelization on the GPU.

`kernel_initializer`
Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs..

`recurrent_initializer`
Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state..

`bias_initializer`
Initializer for the bias vector.

`kernel_regularizer`
Regularizer function applied to the `kernel` weights matrix.

`recurrent_regularizer`
Regularizer function applied to the `recurrent_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` weights matrix.

`recurrent_constraint`
Constraint function applied to the `recurrent_kernel` weights matrix.

`bias_constraint`
Constraint function applied to the bias vector.

`dropout` Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.

`recurrent_dropout`
Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

`input_shape` Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

`batch_input_shape`
Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.

`batch_size` Fixed batch size for layer

`dtype` The data type expected by the input, as a string (`float32`, `float64`, `int32`...)

`name` An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.

`trainable` Whether the layer weights will be updated during training.

`weights` Initial weights for layer.

### Statefulness in RNNs

You can set RNN layers to be 'stateful', which means that the states computed for the samples in one batch will be reused as initial states for the samples in the next batch. This assumes a one-to-one mapping between samples in different successive batches.

To enable statefulness:

- Specify `stateful=TRUE` in the layer constructor.
- Specify a fixed batch size for your model. For sequential models, pass `batch_input_shape = c(...)` to the first layer in your model. For functional models with 1 or more Input layers, pass `batch_shape = c(...)` to all the first layers in your model. This is the expected shape of your inputs *including the batch size*. It should be a vector of integers, e.g. `c(32, 10, 100)`.
- Specify `shuffle = FALSE` when calling fit().

To reset the states of your model, call `reset_states()` on either a specific layer, or on your entire model.

### Initial State of RNNs

You can specify the initial state of RNN layers symbolically by calling them with the keyword argument `initial_state`. The value of `initial_state` should be a tensor or list of tensors representing the initial state of the RNN layer.

You can specify the initial state of RNN layers numerically by calling `reset_states` with the keyword argument `states`. The value of `states` should be a numpy array or list of numpy arrays representing the initial state of the RNN layer.

### References

- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

### See Also

Other recurrent layers: `layer_gru`, `layer_lstm`

---

## `layer_spatial_dropout_1d` *Spatial 1D version of Dropout.*

---

### Description

This version performs the same function as Dropout, however it drops entire 1D feature maps instead of individual elements. If adjacent frames within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, `layer_spatial_dropout_1d` will help promote independence between feature maps and should be used instead.

### Usage

```
layer_spatial_dropout_1d(object, rate, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| rate | float between 0 and 1. Fraction of the input units to drop. |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

3D tensor with shape: `(samples, timesteps, channels)`

### Output shape

Same as input

### References

- Efficient Object Localization Using ConvolutionalNetworks

### See Also

Other dropout layers: `layer_dropout`, `layer_spatial_dropout_2d`, `layer_spatial_dropout_3d`

---

## layer_spatial_dropout_2d

*Spatial 2D version of Dropout.*

---

### Description

This version performs the same function as Dropout, however it drops entire 2D feature maps instead of individual elements. If adjacent pixels within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, `layer_spatial_dropout_2d` will help promote independence between feature maps and should be used instead.

### Usage

```
layer_spatial_dropout_2d(object, rate, data_format = NULL,
  batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| rate | float between 0 and 1. Fraction of the input units to drop. |
| data_format | 'channels_first' or 'channels_last'. In 'channels_first' mode, the channels dimension (the depth) is at index 1, in 'channels_last' mode is it at index 3. 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### 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

Same as input

### References

- Efficient Object Localization Using ConvolutionalNetworks

### See Also

Other dropout layers: layer_dropout, layer_spatial_dropout_1d, layer_spatial_dropout_3d

---

## layer_spatial_dropout_3d

*Spatial 3D version of Dropout.*

---

### Description

This version performs the same function as Dropout, however it drops entire 3D feature maps instead of individual elements. If adjacent voxels within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, layer_spatial_dropout_3d will help promote independence between feature maps and should be used instead.

### Usage

```
layer_spatial_dropout_3d(object, rate, data_format = NULL,
  batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| rate | float between 0 and 1. Fraction of the input units to drop. |
| data_format | 'channels_first' or 'channels_last'. In 'channels_first' mode, the channels dimension (the depth) is at index 1, in 'channels_last' mode is it at index 4. 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

5D tensor with shape: (samples, channels, dim1, dim2, dim3) if data_format='channels_first' or 5D tensor with shape: (samples, dim1, dim2, dim3, channels) if data_format='channels_last'.

### Output shape

Same as input

### References

- Efficient Object Localization Using ConvolutionalNetworks

### See Also

Other dropout layers: layer_dropout, layer_spatial_dropout_1d, layer_spatial_dropout_2d

---

## layer_upsampling_1d *Upsampling layer for 1D inputs.*

---

### Description

Repeats each temporal step size times along the time axis.

### Usage

```
layer_upsampling_1d(object, size = 2L, batch_size = NULL, name = NULL,
  trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| size | integer. Upsampling factor. |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

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

### Output shape

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

### See Also

Other convolutional layers: `layer_conv_1d`, `layer_conv_2d_transpose`, `layer_conv_2d`, `layer_conv_3d_transpose`, `layer_conv_3d`, `layer_conv_lstm_2d`, `layer_cropping_1d`, `layer_cropping_2d`, `layer_cropping_3d`, `layer_separable_conv_2d`, `layer_upsampling_2d`, `layer_upsampling_3d`, `layer_zero_padding_1d`, `layer_zero_padding_2d`, `layer_zero_padding_3d`

---

## `layer_upsampling_2d` *Upsampling layer for 2D inputs.*

---

### Description

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

[[0]]: R:[0 [[1]]: R:[1

### Usage

```
layer_upsampling_2d(object, size = c(2L, 2L), data_format = NULL,
  batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| size | int, or list of 2 integers. The upsampling factors for rows and columns. |

| | |
|---|---|
| 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

4D tensor with shape:

- If data_format is "channels_last": (batch, rows, cols, channels)
- If data_format is "channels_first": (batch, channels, rows, cols)

### Output shape

4D tensor with shape:

- If data_format is "channels_last": (batch, upsampled_rows, upsampled_cols, channels)
- If data_format is "channels_first": (batch, channels, upsampled_rows, upsampled_cols)

### See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

---

## layer_upsampling_3d *Upsampling layer for 3D inputs.*

---

### Description

Repeats the 1st, 2nd and 3rd dimensions of the data by size[[0]], size[[1]] and size[[2]] respectively.

[[0]: R:[0 [[1]: R:[1 [[2]: R:[2

### Usage

```
layer_upsampling_3d(object, size = c(2L, 2L, 2L), data_format = NULL,
  batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| size | int, or list of 3 integers. The upsampling factors for dim1, dim2 and dim3. |
| 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, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

5D tensor with shape:

- If data_format is "channels_last": (batch, dim1, dim2, dim3, channels)
- If data_format is "channels_first": (batch, channels, dim1, dim2, dim3)

### Output shape

5D tensor with shape:

- If data_format is "channels_last": (batch, upsampled_dim1, upsampled_dim2, upsampled_dim3, channels)
- If data_format is "channels_first": (batch, channels, upsampled_dim1, upsampled_dim2, upsampled_dim3

### See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

---

## layer_zero_padding_1d *Zero-padding layer for 1D input (e.g. temporal sequence).*

---

### Description

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

### Usage

```
layer_zero_padding_1d(object, padding = 1L, batch_size = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| padding | int, or list of int (length 2)<br>• If int: How many zeros to add at the beginning and end of the padding dimension (axis 1).<br>• If list of int (length 2): How many zeros to add at the beginning and at the end of the padding dimension ((left_pad, right_pad)). |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Input shape

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

### Output shape

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

### See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_2d, layer_zero_padding_3d

---

## layer_zero_padding_2d *Zero-padding layer for 2D input (e.g. picture).*

---

### Description

This layer can add rows and columns of zeros at the top, bottom, left and right side of an image tensor.

### Usage

```
layer_zero_padding_2d(object, padding = c(1L, 1L), data_format = NULL,
  batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| object | Model or layer object |
| padding | int, or list of 2 ints, or list of 2 lists of 2 ints.<br>• If int: the same symmetric padding is applied to width and height.<br>• If list of 2 ints: interpreted as two different symmetric padding values for height and width: (symmetric_height_pad, symmetric_width_pad).<br>• If list of 2 lists of 2 ints: interpreted as ((top_pad, bottom_pad), (left_pad, right_pad)) |
| 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". |
| batch_size | Fixed batch size for layer |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

## Input shape

4D tensor with shape:

- If data_format is "channels_last": (batch, rows, cols, channels)
- If data_format is "channels_first": (batch, channels, rows, cols)

## Output shape

4D tensor with shape:

- If data_format is "channels_last": (batch, padded_rows, padded_cols, channels)
- If data_format is "channels_first": (batch, channels, padded_rows, padded_cols)

## See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_3d

`layer_zero_padding_3d` *Zero-padding layer for 3D data (spatial or spatio-temporal).*

## Description

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

## Usage

```
layer_zero_padding_3d(object, padding = c(1L, 1L, 1L), data_format = NULL,
  batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

## Arguments

| | |
|---|---|
| `object` | Model or layer object |
| `padding` | int, or list of 3 ints, or list of 3 lists of 2 ints.<br>• If int: the same symmetric padding is applied to width and height.<br>• If list of 3 ints: interpreted as three different symmetric padding values: `(symmetric_dim1_pad, symmetric_dim2_pad, symmetric_dim3_pad)`.<br>• If list of 3 lists of 2 ints: interpreted as `((left_dim1_pad, right_dim1_pad), (left_dim2_pad, r` |
| `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, spatial_dim1, spatial_dim2, spatial_dim3, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, spatial_dim1, spatial_dim2` 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". |
| `batch_size` | Fixed batch size for layer |
| `name` | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| `trainable` | Whether the layer weights will be updated during training. |
| `weights` | Initial weights for layer. |

## Input shape

5D tensor with shape:

- If `data_format` is `"channels_last"`: `(batch, first_axis_to_pad, second_axis_to_pad, third_axis_to_pad`
- If `data_format` is `"channels_first"`: `(batch, depth, first_axis_to_pad, second_axis_to_pad, third_axis`

## Output shape

5D tensor with shape:

- If `data_format` is `"channels_last"`: `(batch, first_padded_axis, second_padded_axis, third_axis_to_pad`
- If `data_format` is `"channels_first"`: `(batch, depth, first_padded_axis, second_padded_axis, third_axis`

## See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d

---

`loss_mean_squared_error` *Model loss functions*

---

## Description

Model loss functions

## Usage

```
loss_mean_squared_error(y_true, y_pred)

loss_mean_absolute_error(y_true, y_pred)

loss_mean_absolute_percentage_error(y_true, y_pred)

loss_mean_squared_logarithmic_error(y_true, y_pred)

loss_squared_hinge(y_true, y_pred)

loss_hinge(y_true, y_pred)

loss_categorical_hinge(y_true, y_pred)

loss_logcosh(y_true, y_pred)

loss_categorical_crossentropy(y_true, y_pred)

loss_sparse_categorical_crossentropy(y_true, y_pred)

loss_binary_crossentropy(y_true, y_pred)

loss_kullback_leibler_divergence(y_true, y_pred)

loss_poisson(y_true, y_pred)

loss_cosine_proximity(y_true, y_pred)
```

### Arguments

| | |
|---|---|
| y_true | True labels (Tensor) |
| y_pred | Predictions (Tensor of the same shape as y_true) |

### Details

Loss functions are to be supplied in the loss parameter of the compile() function.

Loss functions can be specified either using the name of a built in loss function (e.g. 'loss = binary_crossentropy'), a reference to a built in loss function (e.g. 'loss = loss_binary_crossentropy()') or by passing an artitrary function that returns a scalar for each data-point and takes the following two arguments:

- y_true True labels (Tensor)
- y_pred Predictions (Tensor of the same shape as y_true)

The actual optimized objective is the mean of the output array across all datapoints.

### Categorical Crossentropy

When using the categorical_crossentropy loss, your targets should be in categorical format (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional vector that is all-zeros expect for a 1 at the index corresponding to the class of the sample). In order to convert integer targets into categorical targets, you can use the Keras utility function to_categorical():

```
categorical_labels <- to_categorical(int_labels, num_classes = NULL)
```

### See Also

compile()

---

## make_sampling_table *Generates a word rank-based probabilistic sampling table.*

---

### Description

This generates an array where the ith element is the probability that a word of rank i would be sampled, according to the sampling distribution used in word2vec. The word2vec formula is: p(word) = min(1, sqrt(word.frequency/sampling_factor) / (word.frequency/sampling_factor)) We assume that the word frequencies follow Zipf's law (s=1) to derive a numerical approximation of frequency(rank): frequency(rank) ~ 1/(rank * (log(rank) + gamma) + 1/2 - 1/(12*rank)) where gamma is the Euler-Mascheroni constant.

### Usage

```
make_sampling_table(size, sampling_factor = 1e-05)
```

### Arguments

| | |
|---|---|
| size | int, number of possible words to sample. |
| sampling_factor | the sampling factor in the word2vec formula. |

### Value

An array of length size where the ith entry is the probability that a word of rank i should be sampled.

### Note

The word2vec formula is: p(word) = min(1, sqrt(word.frequency/sampling_factor) / (word.frequency/sampling_factor))

### See Also

Other text preprocessing: pad_sequences, skipgrams, text_hashing_trick, text_one_hot, text_to_word_sequence

---

## metric_binary_accuracy

*Model performance metrics*

---

### Description

Model performance metrics

### Usage

```
metric_binary_accuracy(y_true, y_pred)

metric_binary_crossentropy(y_true, y_pred)

metric_categorical_accuracy(y_true, y_pred)

metric_categorical_crossentropy(y_true, y_pred)

metric_cosine_proximity(y_true, y_pred)

metric_hinge(y_true, y_pred)

metric_kullback_leibler_divergence(y_true, y_pred)

metric_mean_absolute_error(y_true, y_pred)

metric_mean_absolute_percentage_error(y_true, y_pred)
```

```
metric_mean_squared_error(y_true, y_pred)

metric_mean_squared_logarithmic_error(y_true, y_pred)

metric_poisson(y_true, y_pred)

metric_sparse_categorical_crossentropy(y_true, y_pred)

metric_squared_hinge(y_true, y_pred)

metric_top_k_categorical_accuracy(y_true, y_pred, k = 5)

metric_sparse_top_k_categorical_accuracy(y_true, y_pred, k = 5)
```

## Arguments

| | |
|---|---|
| y_true | True labels (tensor) |
| y_pred | Predictions (tensor of the same shape as y_true). |
| k | An integer, number of top elements to consider. |

## Custom Metrics

You can provide an arbitrary R function as a custom metric. Note that the y_true and y_pred parameters are tensors, so computations on them should use backend tensor functions. For example:

```
# create metric using backend tensor functions
K <- backend()
metric_mean_pred <- function(y_true, y_pred) {
  K$mean(y_pred)
}

model
  optimizer = optimizer_rmsprop(),
  loss = loss_binary_crossentropy,
  metrics = c('accuracy',
              'mean_pred' = metric_mean_pred)
)
```

Note that a name ('mean_pred') is provided for the custom metric function. This name is used within training progress output.

Documentation on the available backend tensor functions can be found at https://rstudio.github.io/keras/articles/backend.html#backend-functions.

## Note

Metric functions are to be supplied in the metrics parameter of the compile() function.

## model_to_json *Model configuration as JSON*

### Description

Save and re-load models configurations as JSON. Note that the representation does not include the weights, only the architecture.

### Usage

```
model_to_json(object)

model_from_json(json, custom_objects = NULL)
```

### Arguments

| | |
|---|---|
| object | Model object to save |
| json | JSON with model configuration |
| custom_objects | Optional named list mapping names to custom classes or functions to be considered during deserialization. |

### See Also

Other model persistence: get_weights, model_to_yaml, save_model_hdf5, save_model_weights_hdf5

## model_to_yaml *Model configuration as YAML*

### Description

Save and re-load models configurations as YAML Note that the representation does not include the weights, only the architecture.

### Usage

```
model_to_yaml(object)

model_from_yaml(yaml, custom_objects = NULL)
```

### Arguments

| | |
|---|---|
| object | Model object to save |
| yaml | YAML with model configuration |
| custom_objects | Optional named list mapping names to custom classes or functions to be considered during deserialization. |

## See Also

Other model persistence: get_weights, model_to_json, save_model_hdf5, save_model_weights_hdf5

---

`normalize` *Normalize a matrix or nd-array*

---

## Description

Normalize a matrix or nd-array

## Usage

```
normalize(x, axis = -1, order = 2)
```

## Arguments

| | |
|---|---|
| x | Matrix or array to normalize |
| axis | Axis along which to normalize |
| order | Normalization order (e.g. 2 for L2 norm) |

## Value

A normalized copy of the array.

---

`optimizer_adadelta` *Adadelta optimizer.*

---

## Description

Adadelta optimizer as described in ADADELTA: An Adaptive Learning RateMethod.

## Usage

```
optimizer_adadelta(lr = 1, rho = 0.95, epsilon = 1e-08, decay = 0,
  clipnorm = NULL, clipvalue = NULL)
```

## Arguments

| | |
|---|---|
| lr | float >= 0. Learning rate. |
| rho | float >= 0. Decay factor. |
| epsilon | float >= 0. Fuzz factor. |
| decay | float >= 0. Learning rate decay over each update. |
| clipnorm | Gradients will be clipped when their L2 norm exceeds this value. |
| clipvalue | Gradients will be clipped when their absolute value exceeds this value. |

### Note

It is recommended to leave the parameters of this optimizer at their default values.

### See Also

Other optimizers: optimizer_adagrad, optimizer_adamax, optimizer_adam, optimizer_nadam, optimizer_rmsprop, optimizer_sgd

---

## optimizer_adagrad *Adagrad optimizer.*

---

### Description

Adagrad optimizer as described in Adaptive Subgradient Methods for OnlineLearning and StochasticOptimization.

### Usage

```
optimizer_adagrad(lr = 0.01, epsilon = 1e-08, decay = 0,
  clipnorm = NULL, clipvalue = NULL)
```

### Arguments

| | |
|---|---|
| lr | float >= 0. Learning rate. |
| epsilon | float >= 0. Fuzz factor. |
| decay | float >= 0. Learning rate decay over each update. |
| clipnorm | Gradients will be clipped when their L2 norm exceeds this value. |
| clipvalue | Gradients will be clipped when their absolute value exceeds this value. |

### Note

It is recommended to leave the parameters of this optimizer at their default values.

### See Also

Other optimizers: optimizer_adadelta, optimizer_adamax, optimizer_adam, optimizer_nadam, optimizer_rmsprop, optimizer_sgd

## optimizer_adam *Adam optimizer*

### Description

Adam optimizer as described in Adam - A Method for StochasticOptimization.

### Usage

```
optimizer_adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999,
  epsilon = 1e-08, decay = 0, clipnorm = NULL, clipvalue = NULL)
```

### Arguments

| | |
|---|---|
| lr | float >= 0. Learning rate. |
| beta_1 | The exponential decay rate for the 1st moment estimates. float, 0 < beta < 1. Generally close to 1. |
| beta_2 | The exponential decay rate for the 2nd moment estimates. float, 0 < beta < 1. Generally close to 1. |
| epsilon | float >= 0. Fuzz factor. |
| decay | float >= 0. Learning rate decay over each update. |
| clipnorm | Gradients will be clipped when their L2 norm exceeds this value. |
| clipvalue | Gradients will be clipped when their absolute value exceeds this value. |

### Note

Default parameters follow those provided in the original paper.

### See Also

Other optimizers: optimizer_adadelta, optimizer_adagrad, optimizer_adamax, optimizer_nadam, optimizer_rmsprop, optimizer_sgd

## optimizer_adamax *Adamax optimizer*

### Description

Adamax optimizer from Section 7 of the Adam paper. It is a variant of Adam based on the infinity norm.

### Usage

```
optimizer_adamax(lr = 0.002, beta_1 = 0.9, beta_2 = 0.999,
  epsilon = 1e-08, decay = 0, clipnorm = NULL, clipvalue = NULL)
```

### Arguments

| | |
|---|---|
| lr | float >= 0. Learning rate. |
| beta_1 | The exponential decay rate for the 1st moment estimates. float, 0 < beta < 1. Generally close to 1. |
| beta_2 | The exponential decay rate for the 2nd moment estimates. float, 0 < beta < 1. Generally close to 1. |
| epsilon | float >= 0. Fuzz factor. |
| decay | float >= 0. Learning rate decay over each update. |
| clipnorm | Gradients will be clipped when their L2 norm exceeds this value. |
| clipvalue | Gradients will be clipped when their absolute value exceeds this value. |

### See Also

Other optimizers: optimizer_adadelta, optimizer_adagrad, optimizer_adam, optimizer_nadam, optimizer_rmsprop, optimizer_sgd

---

## optimizer_nadam *Nesterov Adam optimizer*

---

### Description

Much like Adam is essentially RMSprop with momentum, Nadam is Adam RMSprop with Nesterov momentum. See Incorporating Nesterov Momentum intoAdam.

### Usage

```
optimizer_nadam(lr = 0.002, beta_1 = 0.9, beta_2 = 0.999,
  epsilon = 1e-08, schedule_decay = 0.004, clipnorm = NULL,
  clipvalue = NULL)
```

### Arguments

| | |
|---|---|
| lr | float >= 0. Learning rate. |
| beta_1 | The exponential decay rate for the 1st moment estimates. float, 0 < beta < 1. Generally close to 1. |
| beta_2 | The exponential decay rate for the 2nd moment estimates. float, 0 < beta < 1. Generally close to 1. |
| epsilon | float >= 0. Fuzz factor. |
| schedule_decay | Schedule deacy. |
| clipnorm | Gradients will be clipped when their L2 norm exceeds this value. |
| clipvalue | Gradients will be clipped when their absolute value exceeds this value. |

## Details

Default parameters follow those provided in the paper. It is recommended to leave the parameters of this optimizer at their default values.

## See Also

On the importance of initialization and momentum in deeplearning.

Other optimizers: optimizer_adadelta, optimizer_adagrad, optimizer_adamax, optimizer_adam, optimizer_rmsprop, optimizer_sgd

---

# optimizer_rmsprop *RMSProp optimizer*

---

## Description

RMSProp optimizer

## Usage

```
optimizer_rmsprop(lr = 0.001, rho = 0.9, epsilon = 1e-08, decay = 0,
  clipnorm = NULL, clipvalue = NULL)
```

## Arguments

| | |
|---|---|
| lr | float >= 0. Learning rate. |
| rho | float >= 0. Decay factor. |
| epsilon | float >= 0. Fuzz factor. |
| decay | float >= 0. Learning rate decay over each update. |
| clipnorm | Gradients will be clipped when their L2 norm exceeds this value. |
| clipvalue | Gradients will be clipped when their absolute value exceeds this value. |

## Note

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

This optimizer is usually a good choice for recurrent neural networks.

## See Also

Other optimizers: optimizer_adadelta, optimizer_adagrad, optimizer_adamax, optimizer_adam, optimizer_nadam, optimizer_sgd

## optimizer_sgd *Stochastic gradient descent optimizer*

### Description

Stochastic gradient descent optimizer with support for momentum, learning rate decay, and Nesterov momentum.

### Usage

```
optimizer_sgd(lr = 0.01, momentum = 0, decay = 0, nesterov = FALSE,
  clipnorm = NULL, clipvalue = NULL)
```

### Arguments

| | |
|---|---|
| lr | float >= 0. Learning rate. |
| momentum | float >= 0. Parameter updates momentum. |
| decay | float >= 0. Learning rate decay over each update. |
| nesterov | boolean. Whether to apply Nesterov momentum. |
| clipnorm | Gradients will be clipped when their L2 norm exceeds this value. |
| clipvalue | Gradients will be clipped when their absolute value exceeds this value. |

### Value

Optimizer for use with compile.

### See Also

Other optimizers: optimizer_adadelta, optimizer_adagrad, optimizer_adamax, optimizer_adam, optimizer_nadam, optimizer_rmsprop

## pad_sequences *Pads each sequence to the same length (length of the longest sequence).*

### Description

Pads each sequence to the same length (length of the longest sequence).

### Usage

```
pad_sequences(sequences, maxlen = NULL, dtype = "int32", padding = "pre",
  truncating = "pre", value = 0)
```

### Arguments

| | |
|---|---|
| sequences | List of lists where each element is a sequence |
| maxlen | int, maximum length |
| dtype | type to cast the resulting sequence. |
| padding | 'pre' or 'post', pad either before or after each sequence. |
| truncating | 'pre' or 'post', remove values from sequences larger than maxlen either in the beginning or in the end of the sequence |
| value | float, value to pad the sequences to the desired value. |

### Details

If maxlen is provided, any sequence longer than maxlen is truncated to maxlen. Truncation happens off either the beginning (default) or the end of the sequence. Supports post-padding and pre-padding (default).

### Value

Array with dimensions (number_of_sequences, maxlen)

### See Also

Other text preprocessing: make_sampling_table, skipgrams, text_hashing_trick, text_one_hot, text_to_word_sequence

---

## plot.keras_training_history
*Plot training history*

---

### Description

Plots metrics recorded during training.

### Usage

```
## S3 method for class 'keras_training_history'
plot(x, y, metrics = NULL,
  method = c("auto", "ggplot2", "base"), smooth = TRUE, ...)
```

### Arguments

| | |
|---|---|
| x | Training history object returned from fit(). |
| y | Unused. |
| metrics | One or more metrics to plot (e.g. c('loss', 'accuracy')). Defaults to plotting all captured metrics. |

| | |
|---|---|
| method | Method to use for plotting. The default "auto" will use **ggplot2** if available, and otherwise will use base graphics. |
| smooth | Whether a loess smooth should be added to the plot, only available for the ggplot2 method. If the number of epochs is smaller than ten, it is forced to false. |
| ... | Additional parameters to pass to the plot() method. |

---

## pop_layer *Remove the last layer in a model*

---

### Description

Remove the last layer in a model

### Usage

```
pop_layer(object)
```

### Arguments

| | |
|---|---|
| object | Keras model object |

### See Also

Other model functions: compile, evaluate_generator, evaluate, fit_generator, fit, get_config, get_layer, keras_model_sequential, keras_model, predict.keras.engine.training.Model, predict_generator, predict_on_batch, predict_proba, summary.keras.engine.training.Model, train_on_batch

---

## predict.keras.engine.training.Model *Generate predictions from a Keras model*

---

### Description

Generates output predictions for the input samples, processing the samples in a batched way.

### Usage

```
## S3 method for class 'keras.engine.training.Model'
predict(object, x, batch_size = 32,
  verbose = 0, ...)
```

### Arguments

| | |
|---|---|
| object | Keras model |
| x | Input data (vector, matrix, or array) |
| batch_size | Integer |
| verbose | Verbosity mode, 0 or 1. |
| ... | Unused |

### Value

vector, matrix, or array of predictions

### See Also

Other model functions: compile, evaluate_generator, evaluate, fit_generator, fit, get_config, get_layer, keras_model_sequential, keras_model, pop_layer, predict_generator, predict_on_batch, predict_proba, summary.keras.engine.training.Model, train_on_batch

---

## predict_generator *Generates predictions for the input samples from a data generator.*

---

### Description

The generator should return the same kind of data as accepted by predict_on_batch().

### Usage

```
predict_generator(object, generator, steps, max_queue_size = 10,
  verbose = 0)
```

### Arguments

| | |
|---|---|
| object | Keras model object |
| generator | Generator yielding batches of input samples. |
| steps | Total number of steps (batches of samples) to yield from generator before stopping. |
| max_queue_size | Maximum size for the generator queue. |
| verbose | verbosity mode, 0 or 1. |

### Value

Numpy array(s) of predictions.

### Raises

ValueError: In case the generator yields data in an invalid format.

## See Also

Other model functions: compile, evaluate_generator, evaluate, fit_generator, fit, get_config, get_layer, keras_model_sequential, keras_model, pop_layer, predict.keras.engine.training.Model, predict_on_batch, predict_proba, summary.keras.engine.training.Model, train_on_batch

---

# predict_on_batch *Returns predictions for a single batch of samples.*

---

## Description

Returns predictions for a single batch of samples.

## Usage

```
predict_on_batch(object, x)
```

## Arguments

| | |
|---|---|
| object | Keras model object |
| x | Input data (vector, matrix, or array) |

## Value

array of predictions.

## See Also

Other model functions: compile, evaluate_generator, evaluate, fit_generator, fit, get_config, get_layer, keras_model_sequential, keras_model, pop_layer, predict.keras.engine.training.Model, predict_generator, predict_proba, summary.keras.engine.training.Model, train_on_batch

---

# predict_proba *Generates probability or class probability predictions for the input samples.*

---

## Description

Generates probability or class probability predictions for the input samples.

## Usage

```
predict_proba(object, x, batch_size = 32, verbose = 0)

predict_classes(object, x, batch_size = 32, verbose = 0)
```

### Arguments

| | |
|---|---|
| object | Keras model object |
| x | Input data (vector, matrix, or array) |
| batch_size | Integer |
| verbose | Verbosity mode, 0 or 1. |

### Details

The input samples are processed batch by batch.

### See Also

Other model functions: compile, evaluate_generator, evaluate, fit_generator, fit, get_config, get_layer, keras_model_sequential, keras_model, pop_layer, predict.keras.engine.training.Model, predict_generator, predict_on_batch, summary.keras.engine.training.Model, train_on_batch

---

## regularizer_l1 *L1 and L2 regularization*

---

### Description

L1 and L2 regularization

### Usage

```
regularizer_l1(l = 0.01)

regularizer_l2(l = 0.01)

regularizer_l1_l2(l1 = 0.01, l2 = 0.01)
```

### Arguments

| | |
|---|---|
| l | Regularization factor. |
| l1 | L1 regularization factor. |
| l2 | L2 regularization factor. |

## reset_states *Reset the states for a layer*

### Description

Reset the states for a layer

### Usage

```
reset_states(object)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |

### See Also

Other layer methods: count_params, get_config, get_input_at, get_weights

## save_model_hdf5 *Save/Load models using HDF5 files*

### Description

Save/Load models using HDF5 files

### Usage

```
save_model_hdf5(object, filepath, overwrite = TRUE,
  include_optimizer = TRUE)

load_model_hdf5(filepath, custom_objects = NULL, compile = TRUE)
```

### Arguments

| | |
|---|---|
| object | Model object to save |
| filepath | File path |
| overwrite | Overwrite existing file if necessary |
| include_optimizer | If TRUE, save optimizer's state. |
| custom_objects | Mapping class names (or function names) of custom (non-Keras) objects to class/functions |
| compile | Whether to compile the model after loading. |

## Details

The following components of the model are saved:

- The model architecture, allowing to re-instantiate the model.
- The model weights.
- The state of the optimizer, allowing to resume training exactly where you left off. This allows you to save the entirety of the state of a model in a single file.

Saved models can be reinstantiated via `load_model()`. The model returned by `load_model()` is a compiled model ready to be used (unless the saved model was never compiled in the first place or `compile = FALSE` is specified.

## See Also

Other model persistence: `get_weights`, `model_to_json`, `model_to_yaml`, `save_model_weights_hdf5`

---

`save_model_weights_hdf5` *Save/Load model weights using HDF5 files*

---

## Description

Save/Load model weights using HDF5 files

## Usage

```
save_model_weights_hdf5(object, filepath, overwrite = TRUE)

load_model_weights_hdf5(object, filepath, by_name = FALSE)
```

## Arguments

| | |
|---|---|
| `object` | Model object to save/load |
| `filepath` | Path to the file |
| `overwrite` | Whether to silently overwrite any existing file at the target location |
| `by_name` | Whether to load weights by name or by topological order. |

## Details

The weight file has:

- `layer_names` (attribute), a list of strings (ordered names of model layers).
- For every layer, a `group` named `layer.name`
- For every such layer group, a group attribute `weight_names`, a list of strings (ordered names of weights tensor of the layer).
- For every weight in the layer, a dataset storing the weight value, named after the weight tensor.

For load_model_weights(), if by_name is FALSE (default) weights are loaded based on the network's topology, meaning the architecture should be the same as when the weights were saved. Note that layers that don't have weights are not taken into account in the topological ordering, so adding or removing layers is fine as long as they don't have weights.

If by_name is TRUE, weights are loaded into layers only if they share the same name. This is useful for fine-tuning or transfer-learning models where some of the layers have changed.

## See Also

Other model persistence: get_weights, model_to_json, model_to_yaml, save_model_hdf5

---

# sequences_to_matrix *Convert a list of sequences into a matrix.*

---

## Description

Convert a list of sequences into a matrix.

## Usage

```
sequences_to_matrix(tokenizer, sequences, mode = c("binary", "count", "tfidf",
  "freq"))
```

## Arguments

| | |
|---|---|
| tokenizer | Tokenizer |
| sequences | List of sequences (a sequence is a list of integer word indices). |
| mode | one of "binary", "count", "tfidf", "freq". |

## Value

A matrix

## See Also

Other text tokenization: fit_text_tokenizer, text_tokenizer, texts_to_matrix, texts_to_sequences_generator, texts_to_sequences

skipgrams *Generates skipgram word pairs.*

## Description

Takes a sequence (list of indexes of words), returns list of `couples` (word_index, other_word index) and `labels` (1s or 0s), where label = 1 if 'other_word' belongs to the context of 'word', and label=0 if 'other_word' is randomly sampled

## Usage

```
skipgrams(sequence, vocabulary_size, window_size = 4, negative_samples = 1,
  shuffle = TRUE, categorical = FALSE, sampling_table = NULL)
```

## Arguments

| | |
|---|---|
| `sequence` | a word sequence (sentence), encoded as a list of word indices (integers). If using a `sampling_table`, word indices are expected to match the rank of the words in a reference dataset (e.g. 10 would encode the 10-th most frequently occuring token). Note that index 0 is expected to be a non-word and will be skipped. |
| `vocabulary_size` | int. maximum possible word index + 1 |
| `window_size` | int. actually half-window. The window of a word wi will be `[i-window_size, i+window_size+1]` |
| `negative_samples` | float >= 0. 0 for no negative (=random) samples. 1 for same number as positive samples. etc. |
| `shuffle` | whether to shuffle the word couples before returning them. |
| `categorical` | bool. if FALSE, labels will be integers (eg. `[0, 1, 1 .. ]`), if TRUE labels will be categorical eg. `[[1,0],[0,1],[0,1] .. ]`<br>[[1,0]: R:[1,0 [0,1]: R:0,1 [0,1]: R:0,1 |
| `sampling_table` | 1D array of size `vocabulary_size` where the entry i encodes the probabibily to sample a word of rank i. |

## Value

List of `couples`, `labels` where:

- `couples` is a list of 2-element integer vectors: `[word_index, other_word_index]`.
- `labels` is an integer vector of 0 and 1, where 1 indicates that `other_word_index` was found in the same window as `word_index`, and 0 indicates that `other_word_index` was random.
- if `categorical` is set to `TRUE`, the labels are categorical, ie. 1 becomes `[0,1]`, and 0 becomes `[1, 0]`.

## See Also

Other text preprocessing: `make_sampling_table`, `pad_sequences`, `text_hashing_trick`, `text_one_hot`, `text_to_word_sequence`

## summary.keras.engine.training.Model *Print a summary of a Keras model*

### Description

Print a summary of a Keras model

### Usage

```
## S3 method for class 'keras.engine.training.Model'
summary(object,
  line_length = getOption("width"), positions = NULL, ...)
```

### Arguments

| | |
|---|---|
| object | Keras model instance |
| line_length | Total length of printed lines |
| positions | Relative or absolute positions of log elements in each line. If not provided, defaults to c(0.33, 0.55, 0.67, 1.0). |
| ... | Unused |

### See Also

Other model functions: compile, evaluate_generator, evaluate, fit_generator, fit, get_config, get_layer, keras_model_sequential, keras_model, pop_layer, predict.keras.engine.training.Model, predict_generator, predict_on_batch, predict_proba, train_on_batch

## texts_to_matrix *Convert a list of texts to a matrix.*

### Description

Convert a list of texts to a matrix.

### Usage

```
texts_to_matrix(tokenizer, texts, mode = c("binary", "count", "tfidf",
  "freq"))
```

### Arguments

| | |
|---|---|
| tokenizer | Tokenizer |
| texts | Vector/list of texts (strings). |
| mode | one of "binary", "count", "tfidf", "freq". |

### Value

A matrix

### See Also

Other text tokenization: fit_text_tokenizer, sequences_to_matrix, text_tokenizer, texts_to_sequences_generat
texts_to_sequences

---

## texts_to_sequences *Transform each text in texts in a sequence of integers.*

---

### Description

Only top "num_words" most frequent words will be taken into account. Only words known by the tokenizer will be taken into account.

### Usage

```
texts_to_sequences(tokenizer, texts)
```

### Arguments

| | |
|---|---|
| tokenizer | Tokenizer |
| texts | Vector/list of texts (strings). |

### See Also

Other text tokenization: fit_text_tokenizer, sequences_to_matrix, text_tokenizer, texts_to_matrix, texts_to_sequences_generator

---

## texts_to_sequences_generator *Transforms each text in texts in a sequence of integers.*

---

### Description

Only top "num_words" most frequent words will be taken into account. Only words known by the tokenizer will be taken into account.

### Usage

```
texts_to_sequences_generator(tokenizer, texts)
```

### Arguments

| | |
|---|---|
| tokenizer | Tokenizer |
| texts | Vector/list of texts (strings). |

### Value

Generator which yields individual sequences

### See Also

Other text tokenization: fit_text_tokenizer, sequences_to_matrix, text_tokenizer, texts_to_matrix, texts_to_sequences

---

## text_hashing_trick *Converts a text to a sequence of indexes in a fixed-size hashing space.*

---

### Description

Converts a text to a sequence of indexes in a fixed-size hashing space.

### Usage

```
text_hashing_trick(text, n, hash_function = NULL,
  filters = "!\"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n", lower = TRUE,
  split = " ")
```

### Arguments

| | |
|---|---|
| text | Input text (string). |
| n | Dimension of the hashing space. |
| hash_function | if NULL uses python hash function, can be 'md5' or any function that takes in input a string and returns a int. Note that hash is not a stable hashing function, so it is not consistent across different runs, while 'md5' is a stable hashing function. |
| filters | Sequence of characters to filter out. |
| lower | Whether to convert the input to lowercase. |
| split | Sentence split marker (string). |

### Details

Two or more words may be assigned to the same index, due to possible collisions by the hashing function.

### Value

A list of integer word indices (unicity non-guaranteed).

## See Also

Other text preprocessing: make_sampling_table, pad_sequences, skipgrams, text_one_hot, text_to_word_sequence

---

`text_one_hot` *One-hot encode a text into a list of word indexes in a vocabulary of size n.*

---

## Description

One-hot encode a text into a list of word indexes in a vocabulary of size n.

## Usage

```
text_one_hot(text, n, filters = "!\"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n",
  lower = TRUE, split = " ")
```

## Arguments

| | |
|---|---|
| `text` | Input text (string). |
| `n` | Size of vocabulary (integer) |
| `filters` | Sequence of characters to filter out. |
| `lower` | Whether to convert the input to lowercase. |
| `split` | Sentence split marker (string). |

## Value

List of integers in `[1, n]`. Each integer encodes a word (unicity non-guaranteed).

## See Also

Other text preprocessing: make_sampling_table, pad_sequences, skipgrams, text_hashing_trick, text_to_word_sequence

`text_tokenizer` *Text tokenization utility*

## Description

Vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf...

## Usage

```
text_tokenizer(num_words = NULL,
  filters = "!\"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n", lower = TRUE,
  split = " ", char_level = FALSE)
```

## Arguments

| | |
|---|---|
| `num_words` | the maximum number of words to keep, based on word frequency. Only the most common `num_words` words will be kept. |
| `filters` | a string where each element is a character that will be filtered from the texts. The default is all punctuation, plus tabs and line breaks, minus the ' character. |
| `lower` | boolean. Whether to convert the texts to lowercase. |
| `split` | character or string to use for token splitting. |
| `char_level` | if `TRUE`, every character will be treated as a token |

## Details

By default, all punctuation is removed, turning the texts into space-separated sequences of words (words maybe include the ' character). These sequences are then split into lists of tokens. They will then be indexed or vectorized. `0` is a reserved index that won't be assigned to any word.

## Attributes

The tokenizer object has the following attributes:

- `word_counts` — named list mapping words to the number of times they appeared on during fit. Only set after `fit_text_tokenizer()` is called on the tokenizer.
- `word_docs` — named list mapping words to the number of documents/texts they appeared on during fit. Only set after `fit_text_tokenizer()` is called on the tokenizer.
- `word_index` — named list mapping words to their rank/index (int). Only set after `fit_text_tokenizer()` is called on the tokenizer.
- `document_count` — int. Number of documents (texts/sequences) the tokenizer was trained on. Only set after `fit_text_tokenizer()` is called on the tokenizer.

## See Also

Other text tokenization: fit_text_tokenizer, sequences_to_matrix, texts_to_matrix, texts_to_sequences_genera
texts_to_sequences

---

`text_to_word_sequence` *Convert text to a sequence of words (or tokens).*

---

## Description

Convert text to a sequence of words (or tokens).

## Usage

```
text_to_word_sequence(text,
  filters = "!\"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n", lower = TRUE,
  split = " ")
```

## Arguments

| | |
|---|---|
| `text` | Input text (string). |
| `filters` | Sequence of characters to filter out. |
| `lower` | Whether to convert the input to lowercase. |
| `split` | Sentence split marker (string). |

## Value

Words (or tokens)

## See Also

Other text preprocessing: make_sampling_table, pad_sequences, skipgrams, text_hashing_trick,
text_one_hot

---

`time_distributed` *Apply a layer to every temporal slice of an input.*

---

## Description

The input should be at least 3D, and the dimension of index one will be considered to be the temporal dimension.

## Usage

```
time_distributed(object, layer, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

### Arguments

| | |
|---|---|
| object | Model or layer object |
| layer | A layer instance. |
| input_shape | Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model. |
| batch_input_shape | Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors. |
| batch_size | Fixed batch size for layer |
| dtype | The data type expected by the input, as a string (float32, float64, int32...) |
| name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided. |
| trainable | Whether the layer weights will be updated during training. |
| weights | Initial weights for layer. |

### Details

Consider a batch of 32 samples, where each sample is a sequence of 10 vectors of 16 dimensions. The batch input shape of the layer is then (32, 10, 16), and the input_shape, not including the samples dimension, is (10, 16). You can then use time_distributed to apply a layer_dense to each of the 10 timesteps, independently.

### See Also

Other layer wrappers: bidirectional

---

## to_categorical *Converts a class vector (integers) to binary class matrix.*

---

### Description

Converts a class vector (integers) to binary class matrix.

### Usage

```
to_categorical(y, num_classes = NULL)
```

### Arguments

| | |
|---|---|
| y | Class vector to be converted into a matrix (integers from 0 to num_classes). |
| num_classes | Total number of classes. |

### Details

E.g. for use with loss_categorical_crossentropy().

### Value

A binary matrix representation of the input.

---

train_on_batch *Single gradient update or model evaluation over one batch of samples.*

---

### Description

Single gradient update or model evaluation over one batch of samples.

### Usage

```
train_on_batch(object, x, y, class_weight = NULL, sample_weight = NULL)

test_on_batch(object, x, y, sample_weight = NULL)
```

### Arguments

| | |
|---|---|
| object | Keras model object |
| x | input data, as an array or list of arrays (if the model has multiple inputs). |
| y | labels, as an array. |
| class_weight | named list mapping classes to a weight value, used for scaling the loss function (during training only). |
| sample_weight | sample weights, as an array. |

### Value

Scalar training or test loss (if the model has no metrics) or list of scalars (if the model computes other metrics). The property model$metrics_names will give you the display labels for the scalar outputs.

### See Also

Other model functions: compile, evaluate_generator, evaluate, fit_generator, fit, get_config, get_layer, keras_model_sequential, keras_model, pop_layer, predict.keras.engine.training.Model, predict_generator, predict_on_batch, predict_proba, summary.keras.engine.training.Model

# Index