

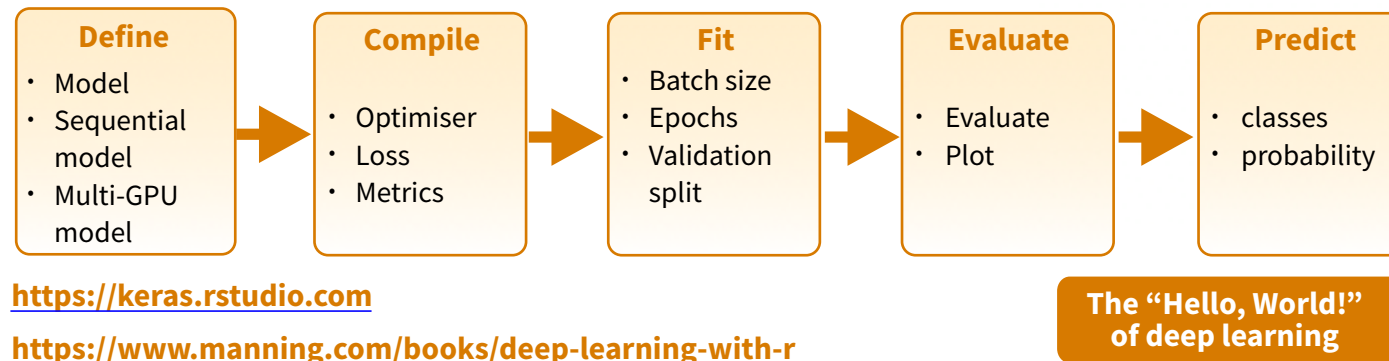
# Deep Learning with Keras :: CHEAT SHEET



## Intro

**Keras** is a high-level neural networks API developed with a focus on enabling fast experimentation. It supports multiple back-ends, including TensorFlow, CNTK and Theano.

TensorFlow is a lower level mathematical library for building deep neural network architectures. The **keras** R package makes it easy to use Keras and TensorFlow in R.



## INSTALLATION

The **keras** R package uses the Python **keras** library. You can install all the prerequisites directly from R.

[https://keras.rstudio.com/reference/install\\_keras.html](https://keras.rstudio.com/reference/install_keras.html)

```
library(keras)
install_keras()
```

See ?install\_keras for GPU instructions

This installs the required libraries in an Anaconda environment or virtual environment 'r-tensorflow'.

## Working with keras models

### DEFINE A MODEL

**keras\_model()** Keras Model

**keras\_model\_sequential()** Keras Model composed of a linear stack of layers

**multi\_gpu\_model()** Replicates a model on different GPUs

### COMPILE A MODEL

**compile(object, optimizer, loss, metrics = NULL)**  
Configure a Keras model for training

### FIT A MODEL

**fit(object, x = NULL, y = NULL, batch\_size = NULL, epochs = 10, verbose = 1, callbacks = NULL, ...)**  
Train a Keras model for a fixed number of epochs (iterations)

**fit\_generator()** Fits the model on data yielded batch-by-batch by a generator

**train\_on\_batch() test\_on\_batch()** Single gradient update or model evaluation over one batch of samples

### EVALUATE A MODEL

**evaluate(object, x = NULL, y = NULL, batch\_size = NULL)** Evaluate a Keras model

**evaluate\_generator()** Evaluates the model on a data generator

### PREDICT

**predict()** Generate predictions from a Keras model

**predict\_proba()** and **predict\_classes()**  
Generates probability or class probability predictions for the input samples

**predict\_on\_batch()** Returns predictions for a single batch of samples

**predict\_generator()** Generates predictions for the input samples from a data generator

### OTHER MODEL OPERATIONS

**summary()** Print a summary of a Keras model

**export\_savedmodel()** Export a saved model

**get\_layer()** Retrieves a layer based on either its name (unique) or index

**pop\_layer()** Remove the last layer in a model

**save\_model\_hdf5(); load\_model\_hdf5()** Save/Load models using HDF5 files

**serialize\_model(); unserialize\_model()**  
Serialize a model to an R object

**clone\_model()** Clone a model instance

**freeze\_weights(); unfreeze\_weights()**  
Freeze and unfreeze weights

### CORE LAYERS

**layer\_input()** Input layer

**layer\_dense()** Add a densely-connected NN layer to an output

**layer\_activation()** Apply an activation function to an output

**layer\_dropout()** Applies Dropout to the input

**layer\_reshape()** Reshapes an output to a certain shape

**layer\_permute()** Permute the dimensions of an input according to a given pattern

**layer\_repeat\_vector()** Repeats the input n times

**layer\_lambda(object, f)** Wraps arbitrary expression as a layer

**layer\_activity\_regularization()**  
Layer that applies an update to the cost function based input activity

**layer\_masking()** Masks a sequence by using a mask value to skip timesteps

**layer\_flatten()** Flattens an input

### # input layer: use MNIST images

```
mnist <- dataset_mnist()
x_train <- mnist$train$x; y_train <- mnist$train$y
x_test <- mnist$test$x; y_test <- mnist$test$y
```

### # reshape and rescale

```
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
x_train <- x_train / 255; x_test <- x_test / 255
```

```
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

### # defining the model and layers

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu',
              input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')
```

### # compile (define loss and optimizer)

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

### # train (fit)

```
model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
model %>% evaluate(x_test, y_test)
model %>% predict_classes(x_test)
```

# More layers

## CONVOLUTIONAL LAYERS



**layer\_conv\_1d()** 1D, e.g. temporal convolution



**layer\_conv\_2d\_transpose()**  
Transposed 2D (deconvolution)

**layer\_conv\_2d()** 2D, e.g. spatial convolution over images



**layer\_conv\_3d\_transpose()**  
Transposed 3D (deconvolution)  
**layer\_conv\_3d()** 3D, e.g. spatial convolution over volumes

**layer\_conv\_lstm\_2d()**  
Convolutional LSTM

**layer\_separable\_conv\_2d()**  
Depthwise separable 2D



**layer\_upsampling\_1d()**  
**layer\_upsampling\_2d()**  
**layer\_upsampling\_3d()**  
Upsampling layer



**layer\_zero\_padding\_1d()**  
**layer\_zero\_padding\_2d()**  
**layer\_zero\_padding\_3d()**  
Zero-padding layer



**layer\_cropping\_1d()**  
**layer\_cropping\_2d()**  
**layer\_cropping\_3d()**  
Cropping layer

## POOLING LAYERS



**layer\_max\_pooling\_1d()**  
**layer\_max\_pooling\_2d()**  
**layer\_max\_pooling\_3d()**  
Maximum pooling for 1D to 3D



**layer\_average\_pooling\_1d()**  
**layer\_average\_pooling\_2d()**  
**layer\_average\_pooling\_3d()**  
Average pooling for 1D to 3D



**layer\_global\_max\_pooling\_1d()**  
**layer\_global\_max\_pooling\_2d()**  
**layer\_global\_max\_pooling\_3d()**  
Global maximum pooling



**layer\_global\_average\_pooling\_1d()**  
**layer\_global\_average\_pooling\_2d()**  
**layer\_global\_average\_pooling\_3d()**  
Global average pooling



## ACTIVATION LAYERS



**layer\_activation(object, activation)**  
Apply an activation function to an output



**layer\_activation\_leaky\_relu()**  
Leaky version of a rectified linear unit



**layer\_activation\_parametric\_relu()**  
Parametric rectified linear unit



**layer\_activation\_thresholded\_relu()**  
Thresholded rectified linear unit



**layer\_activation\_elu()**  
Exponential linear unit

## DROPOUT LAYERS



**layer\_dropout()**  
Applies dropout to the input



**layer\_spatial\_dropout\_1d()**  
**layer\_spatial\_dropout\_2d()**  
**layer\_spatial\_dropout\_3d()**  
Spatial 1D to 3D version of dropout

## RECURRENT LAYERS



**layer\_simple\_rnn()**  
Fully-connected RNN where the output is to be fed back to input

**layer\_gru()**  
Gated recurrent unit - Cho et al

**layer\_cudnn\_gru()**  
Fast GRU implementation backed by CuDNN

**layer\_lstm()**  
Long-Short Term Memory unit - Hochreiter 1997

**layer\_cudnn\_lstm()**  
Fast LSTM implementation backed by CuDNN

## LOCALLY CONNECTED LAYERS

**layer\_locally\_connected\_1d()**  
**layer\_locally\_connected\_2d()**  
Similar to convolution, but weights are not shared, i.e. different filters for each patch

# Preprocessing

## SEQUENCE PREPROCESSING

**pad\_sequences()**  
Pads each sequence to the same length (length of the longest sequence)

**skipgrams()**  
Generates skipgram word pairs

**make\_sampling\_table()**  
Generates word rank-based probabilistic sampling table

## TEXT PREPROCESSING

**text\_tokenizer()** Text tokenization utility

**fit\_text\_tokenizer()** Update tokenizer internal vocabulary

**save\_text\_tokenizer(); load\_text\_tokenizer()**  
Save a text tokenizer to an external file

**texts\_to\_sequences(); texts\_to\_sequences\_generator()**  
Transforms each text in texts to sequence of integers

**texts\_to\_matrix(); sequences\_to\_matrix()**  
Convert a list of sequences into a matrix

**text\_one\_hot()** One-hot encode text to word indices

**text\_hashing\_trick()**  
Converts a text to a sequence of indexes in a fixed-size hashing space

**text\_to\_word\_sequence()**  
Convert text to a sequence of words (or tokens)

## IMAGE PREPROCESSING

**image\_load()** Loads an image into PIL format.

**flow\_images\_from\_data()**  
**flow\_images\_from\_directory()**  
Generates batches of augmented/normalized data from images and labels, or a directory

**image\_data\_generator()** Generate minibatches of image data with real-time data augmentation.

**fit\_image\_data\_generator()** Fit image data generator internal statistics to some sample data

**generator\_next()** Retrieve the next item

**image\_to\_array(); image\_array\_resize()**  
**image\_array\_save()** 3D array representation



# Pre-trained models

Keras applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

**application\_xception()**  
**xception\_preprocess\_input()**  
Xception v1 model

**application\_inception\_v3()**  
**inception\_v3\_preprocess\_input()**  
Inception v3 model, with weights pre-trained on ImageNet

**application\_inception\_resnet\_v2()**  
**inception\_resnet\_v2\_preprocess\_input()**  
Inception-ResNet v2 model, with weights trained on ImageNet

**application\_vgg16(); application\_vgg19()**  
VGG16 and VGG19 models

**application\_resnet50()** ResNet50 model

**application\_mobilenet()**  
**mobilenet\_preprocess\_input()**  
**mobilenet\_decode\_predictions()**  
**mobilenet\_load\_model\_hdf5()**  
MobileNet model architecture

## IMAGENET

[ImageNet](https://www.image-net.org/) is a large database of images with labels, extensively used for deep learning

**imagenet\_preprocess\_input()**  
**imagenet\_decode\_predictions()**  
Preprocesses a tensor encoding a batch of images for ImageNet, and decodes predictions

# Callbacks

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.

**callback\_early\_stopping()** Stop training when a monitored quantity has stopped improving  
**callback\_learning\_rate\_scheduler()** Learning rate scheduler  
**callback\_tensorboard()** TensorBoard basic visualizations