**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
- `layerSpatial_dropout_1d()`, `layerSpatial_dropout_2d()`, `layerSpatial_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_generator()`: Generates batches of augmented/normalized data
- `texts_to_sequences_generator(); texts_to_matrix(); sequences_to_matrix()`: Converts text to a sequence of integers
- `texts_to_matrix(); sequences_to_matrix()`: Converts 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()`: Generates batches of augmented/normalized data from images and labels, or a directory
- `image_data_generator()`: Generates minibatches of image data with real-time data augmentation.
- `fit_image_data_generator()`: Fit image data generator internal statistics to some sample data
- `image_to_array(); image_array_resize(); image_array_save()`: Convert images to arrays

### Callbacks
- `callback_tensorboard()`: TensorBoard basic visualizations
- `callback_early_stopping()`: Stop training when validation loss does not improve
- `callback_learning_rate_scheduler()`: Learning rate scheduler
- `callback_tensorboard()`: TensorBoard basic visualizations

---

**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

---

**ImageNet** 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