DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

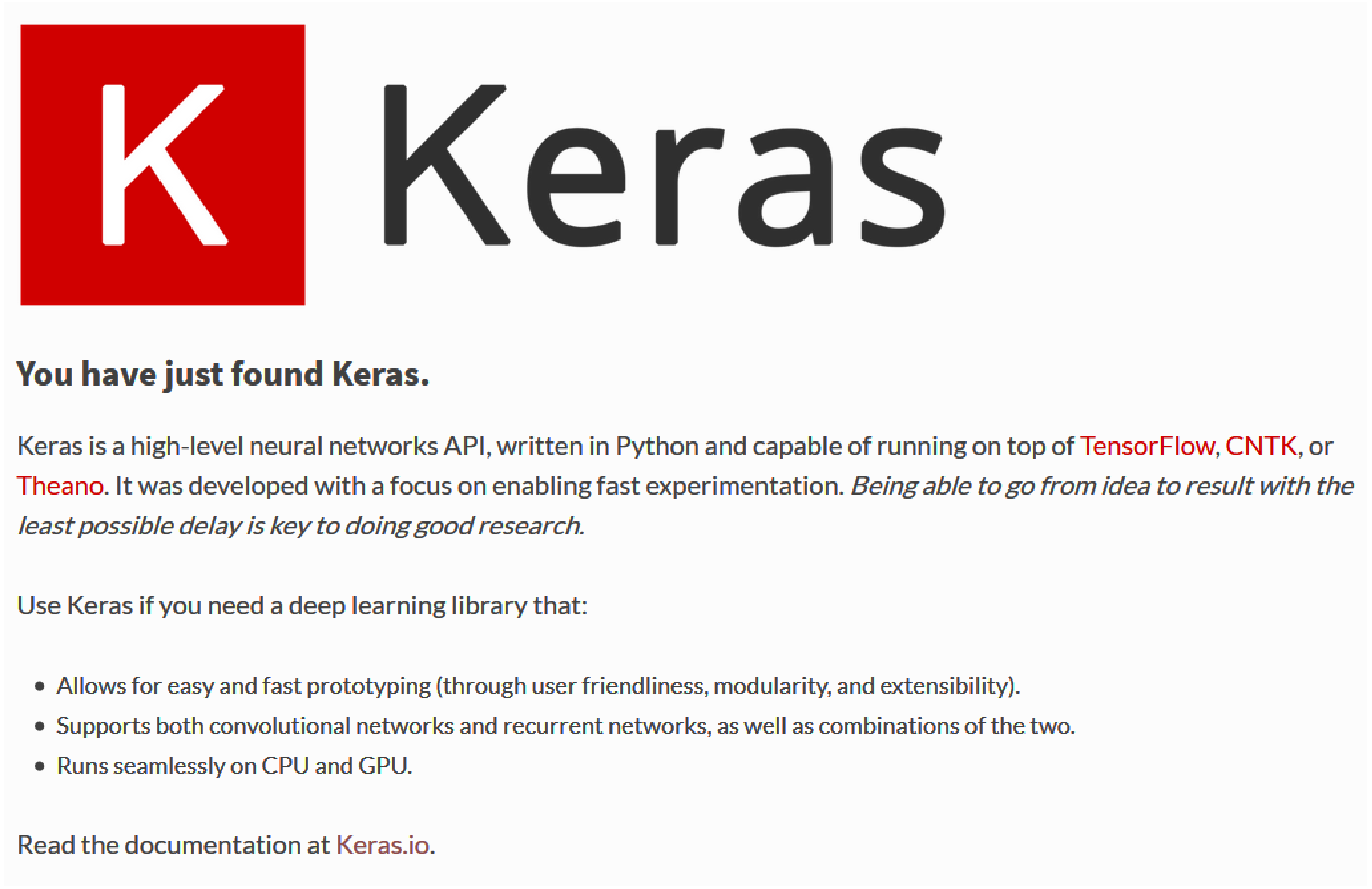
Keras
A deep learning library

Implementing Deep Networks with Keras

Original material: Stefano Ghidoni

Revised : Marco Toldo, Umberto Michieli, Pietro Zanuttigh

Keras



You have just found Keras.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Read the documentation at Keras.io.

Other DL Frameworks

- Tensorflow (most diffused)
- PyTorch (fast growing, especially in research)
- Theano (deprecated)
- Caffe
- CNTK
- …

Guiding Principles

- User friendliness
  - Easy for easy things
  - Possible to do complex things
- Modularity
  - Modules: neural layers, cost functions, optimizers, activation functions, regularization schemes
- Extensibility
  - Easy to add new modules
- Python-based
  - Interactive environment

NB: careful about Keras-TF compatibility

Installing Keras

- Very easy after having installed Tensorflow
  - `pip install keras` (typically already installed)
- The difficult part: *install tensorflow* !

TensorFlow Installation

Two (main) ways of installing

- *anaconda* (especially on Windows it is simpler)
- pip (described on official website)

Anaconda installation (CPU version, simple but slow):

1. Assuming you already have Anaconda/Python 3
2. conda create -n tensorflow_env tensorflow
3. conda activate tensorflow_env

Anaconda installation (GPU version, requires an NVIDIA compatible GPU*):*

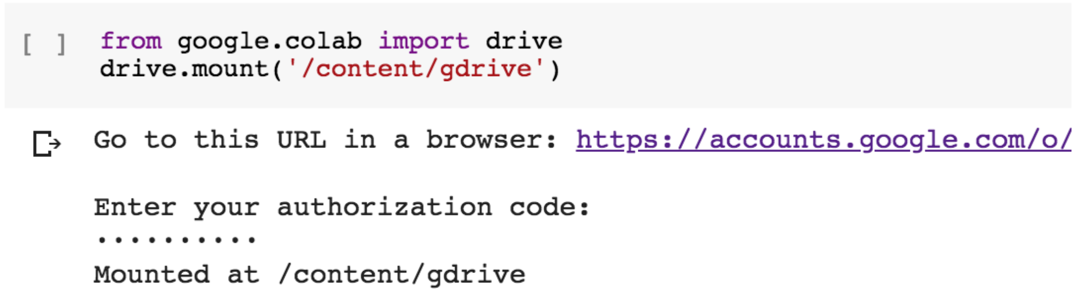
- conda create -n tensorflow_gpuenv tensorflow-gpu
- conda activate tensorflow_gpuenv
- Could require to install (or fix issues with) CUDA and CuDNN libraries

Tensorflow and Jupyter:

- ❑ Jupyter could use a different environment than the TensorFlow one
- ❑ If you have issues install *nb_conda* and switch to the proper kernel form the "kernel" menu

Alternative: use Google Colab

- Requirements:
  - Google account
  - Internet connection

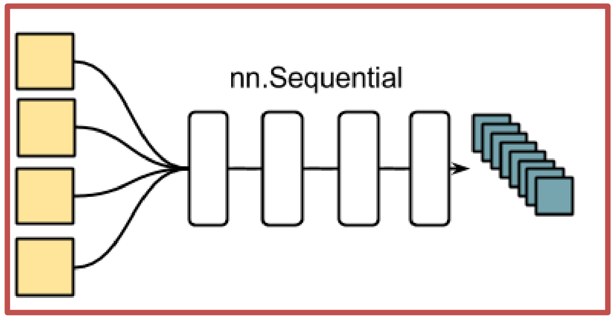


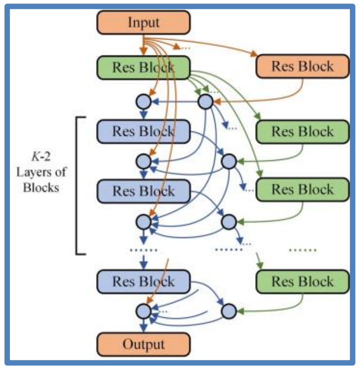
```
[ ]  from google.colab import drive
     drive.mount('/content/gdrive')

     Go to this URL in a browser: https://accounts.google.com/o/

     Enter your authorization code:
     ··········
     Mounted at /content/gdrive
```

- Getting started:
  - Go to your Google Drive and create a new notebook (New -> More -> Google Colaboratory) or upload one
  - Once you have opened it with Google Colaboratory , go to Edit -> Notebook Settings and select GPU
  - Many python libraries available, run `!pip list` to check
  - Mount your Google Drive to access files from it
  - Start coding!

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Sequential vs. Functional Models





- Two ways to build Keras models: *sequential* and *functional*
- The sequential API allows you to create models layer-by-layer
  - Each layer is connected to the previous and next
  - Write your neural network in a few lines of code
  - It does not allow you to create complex models
- The *functional* API allows you to create models that have a lot more flexibility
  - In fact, you can connect each layer to any other layer
  - Creating complex networks becomes possible

The Easy Way

- Instantiate a sequential model
- Stack some layers
- Configure/tune the learning process
- Iterate on the training data
- Evaluate performance
- Generate predictions

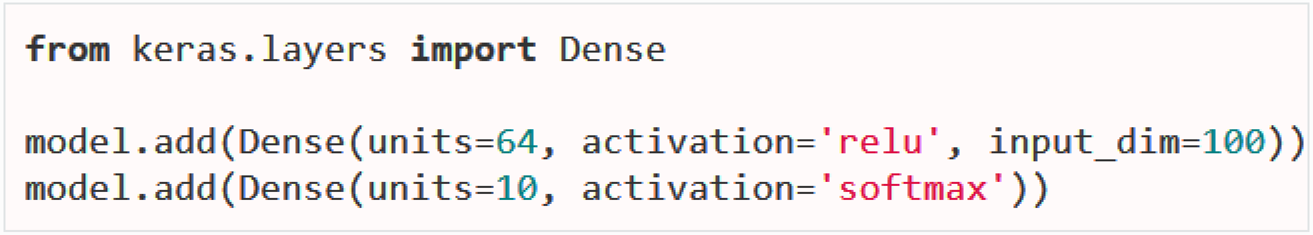
30s to Keras

- Instantiate a sequential model

```
from keras.models import Sequential

model = Sequential()
```

30s to Keras

❑ Stack some layers

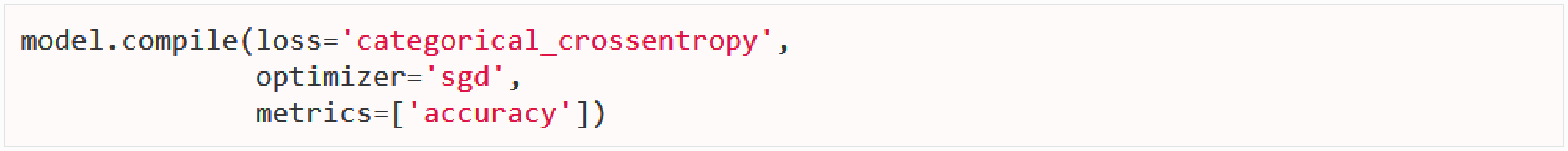


```python
from keras.layers import Dense

model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
```

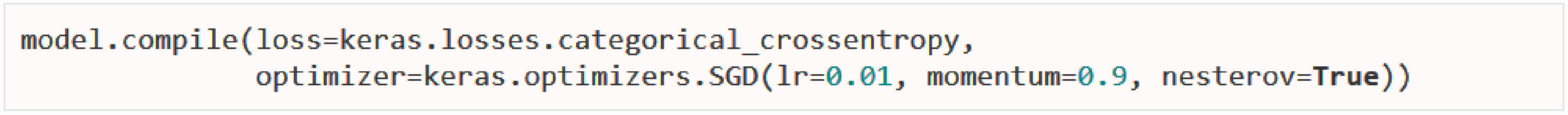
- Add one layer after the other (order matters !)
- Specify input for first layer (for the others it is the output of the previous one)

30s to Keras

❑ Configure/tune the learning process



```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```



```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True))
```

- Notice the difference between loss (used for NN optimization) and metrics
- Can use pre-defined settings ('..') or manually specify optimizer parameters (…*modularity* and *extensibility*…)

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

30s to Keras

- Iterate on the training data

```
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

```
model.train_on_batch(x_batch, y_batch)
```

- fit: runs the whole training procedure
- train_on_batch: perform a single step

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

30s to Keras

- Evaluate performance

```
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
```

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

30s to Keras

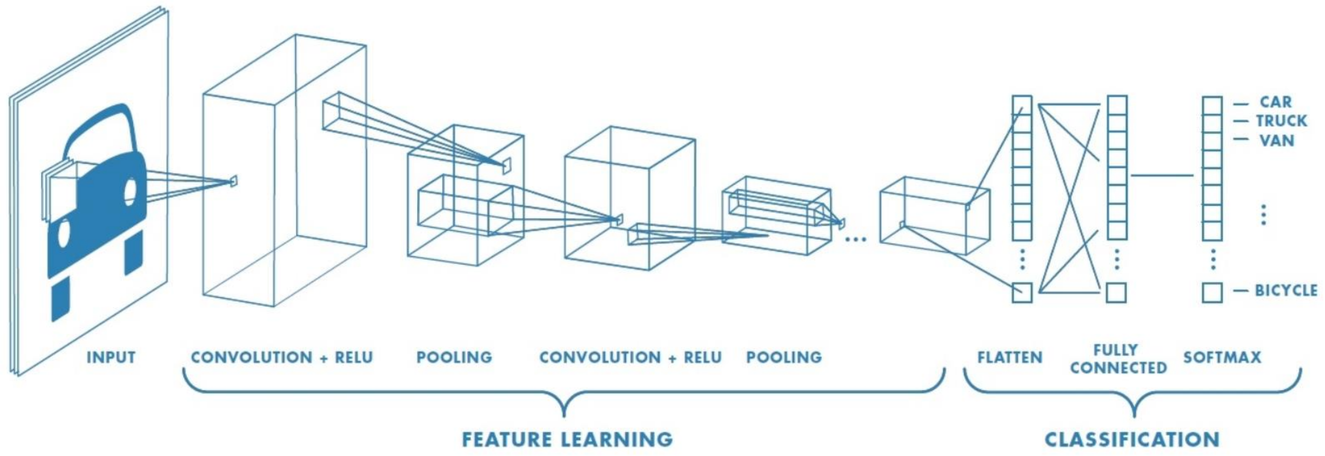
- Generate predictions

```
classes = model.predict(x_test, batch_size=128)
```

- The end
  - Simple but limited... More advanced example with functional model in the notebook

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Keras: Build a CNN

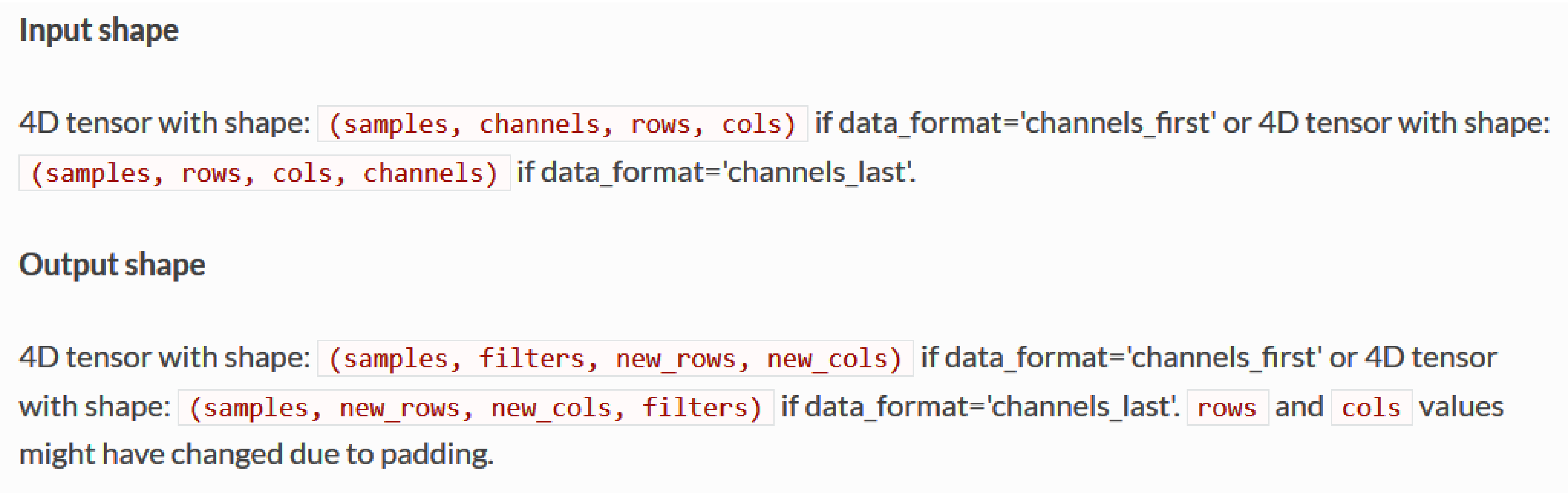


Typical layers in a Convolutional Neural Network

1. Convolutional
2. Pooling
3. Fully connected
4. Activations (ReLU, Softmax, …)

I/O: Tensors

Tensor: typed multi-dimensional array (From TensorFlow)



Input shape

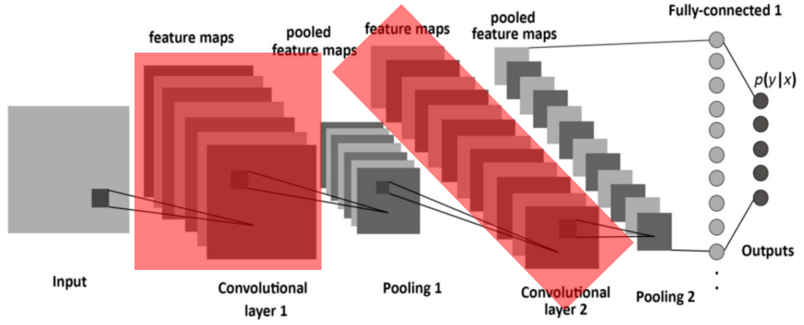
4D tensor with shape: `(samples, channels, rows, cols)` if data_format='channels_first' or 4D tensor with shape: `(samples, rows, cols, channels)` if data_format='channels_last'.

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.

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

2D Convolutional Layers in Keras



```
model.add(Conv2D(32, (3, 3), input_shape=(28,28,1)))
```

```
model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1),
                 activation='relu',
                 input_shape=input_shape))
```

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,
dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform',
bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None, bias_constraint=None)
```

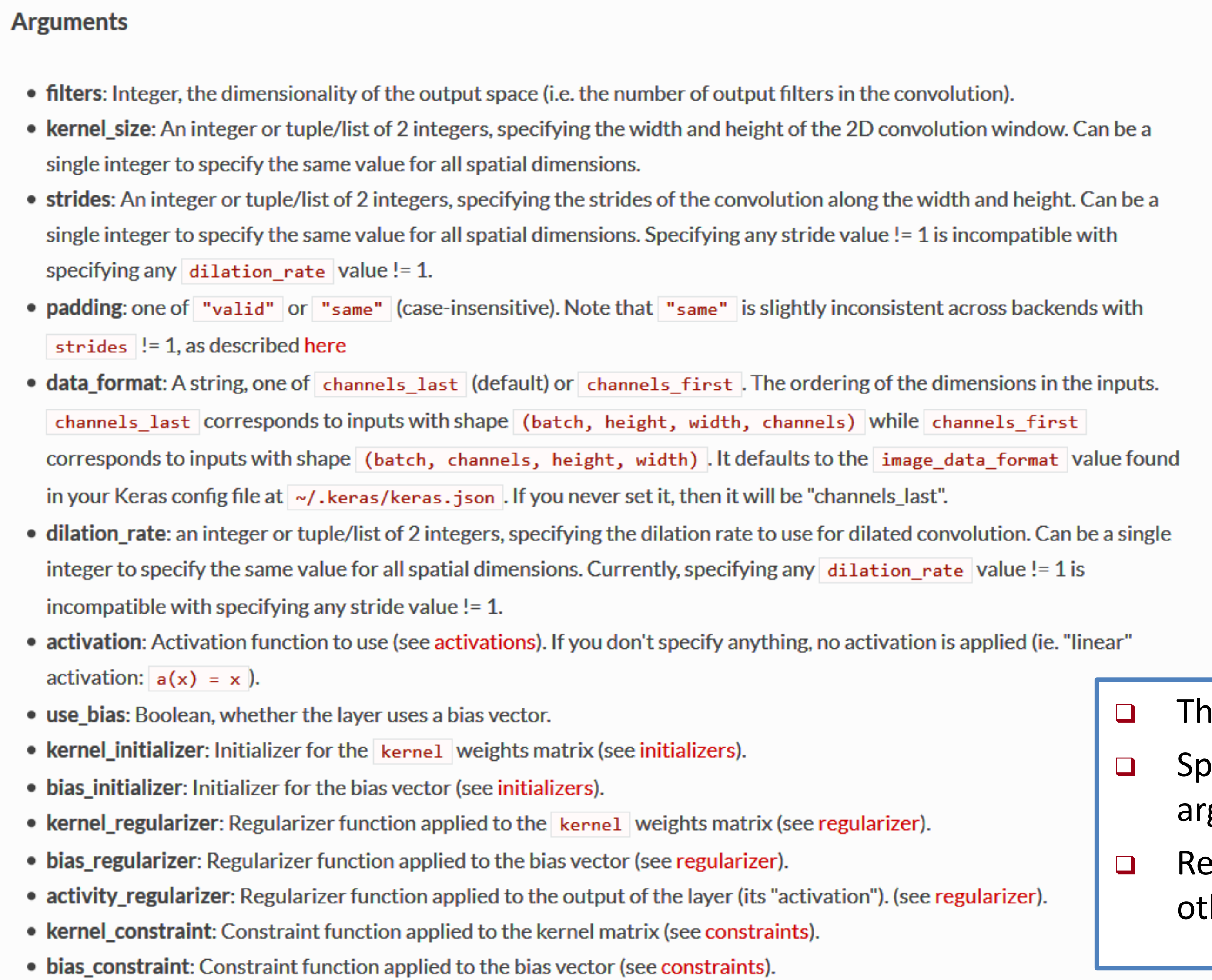
2D convolution layer (e.g. spatial convolution over images).

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 `None`, 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` (tuple of integers, does not include the sample axis), e.g. `input_shape=(128, 128, 3)` for 128x128 RGB pictures in `data_format="channels_last"`.

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Conv2D Arguments



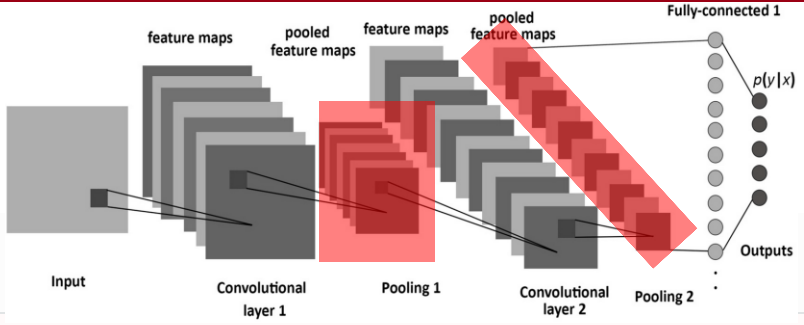
Arguments

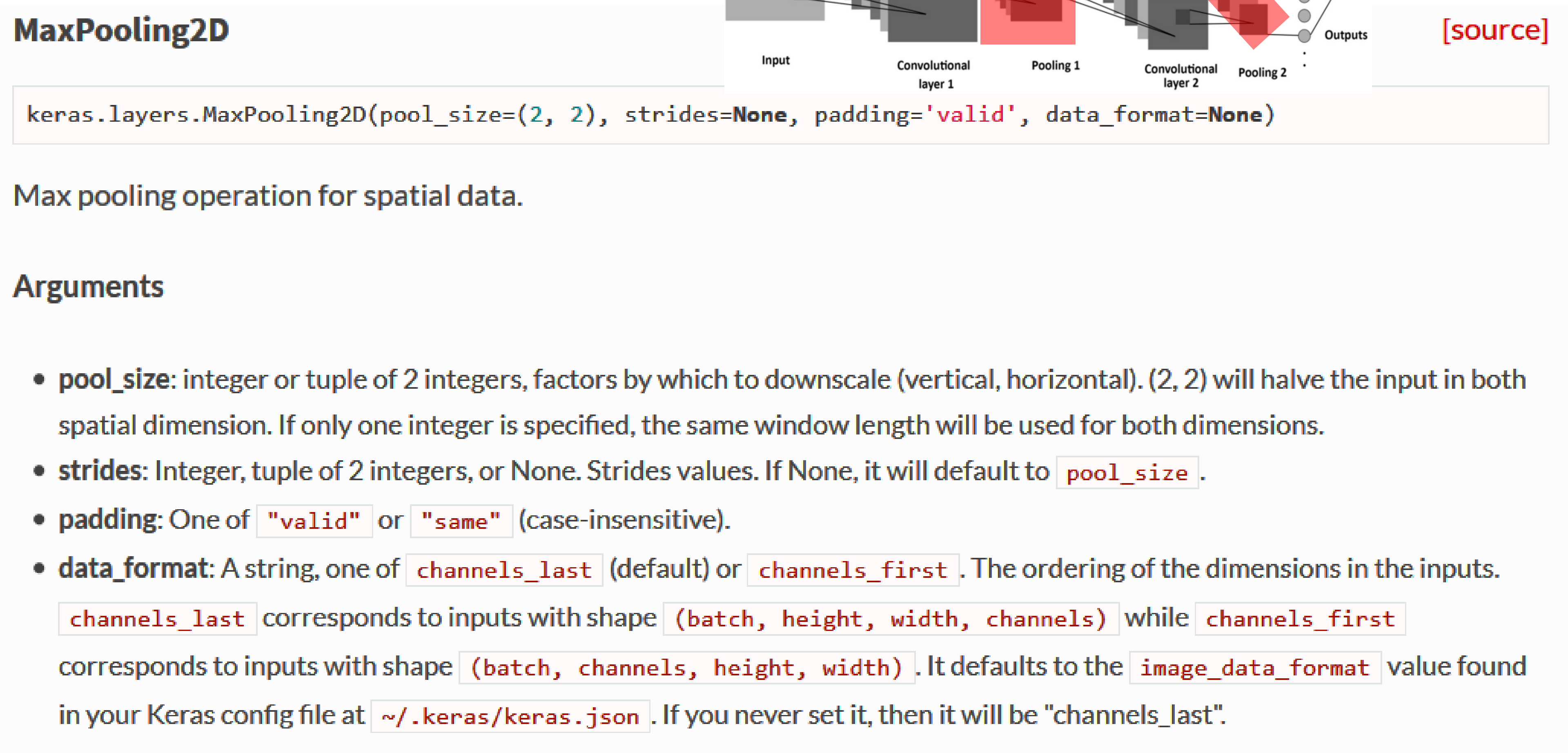
- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: one of `"valid"` or `"same"` (case-insensitive). Note that `"same"` is slightly inconsistent across backends with `strides` != 1, as described here
- **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 tuple/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 (see activations). 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 (see initializers).
- **bias_initializer**: Initializer for the bias vector (see initializers).
- **kernel_regularizer**: Regularizer function applied to the `kernel` weights matrix (see regularizer).
- **bias_regularizer**: Regularizer function applied to the bias vector (see regularizer).
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).
- **kernel_constraint**: Constraint function applied to the kernel matrix (see constraints).
- **bias_constraint**: Constraint function applied to the bias vector (see constraints).

- The argument list is huge!
- Specify only the desired arguments
- Rely on default values for the others

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Pooling Layers





MaxPooling2D

[source]

```
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)
```

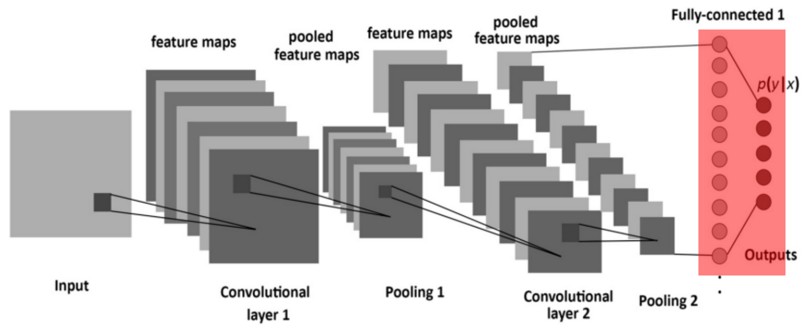
Max pooling operation for spatial data.

Arguments

- **pool_size**: integer or tuple 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, tuple of 2 integers, or None. Strides values. If None, 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".

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Dense (Fully Connected) Layers



```
keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform',
bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None, bias_constraint=None)
```

Just your regular densely-connected NN layer.

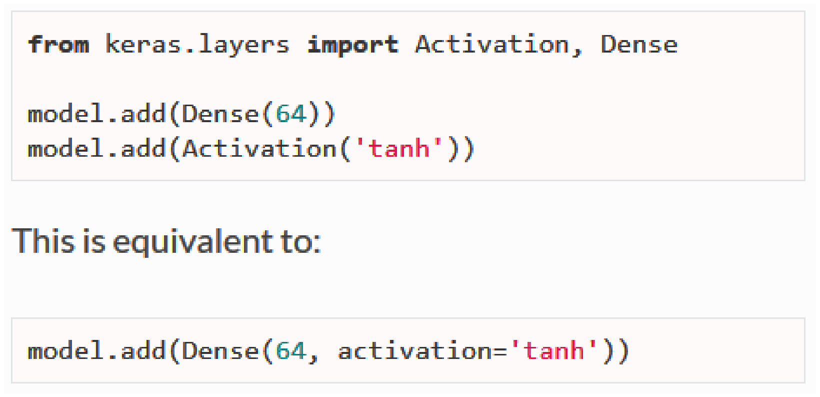
`Dense` implements the operation: `output = activation(dot(input, kernel) + bias)` where `activation` is the element-wise activation function passed as the `activation` argument, `kernel` is a weights matrix created by the layer, and `bias` is a bias vector created by the layer (only applicable if `use_bias` 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`.

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Activations and Softmax

- Activations (e.g. ReLU, tanh, …)
  - As standalone layers
  - Embedded in forward layers
- Softmax activation often at the end of the last fully connected layer



```
from keras.layers import Activation, Dense

model.add(Dense(64))
model.add(Activation('tanh'))
```

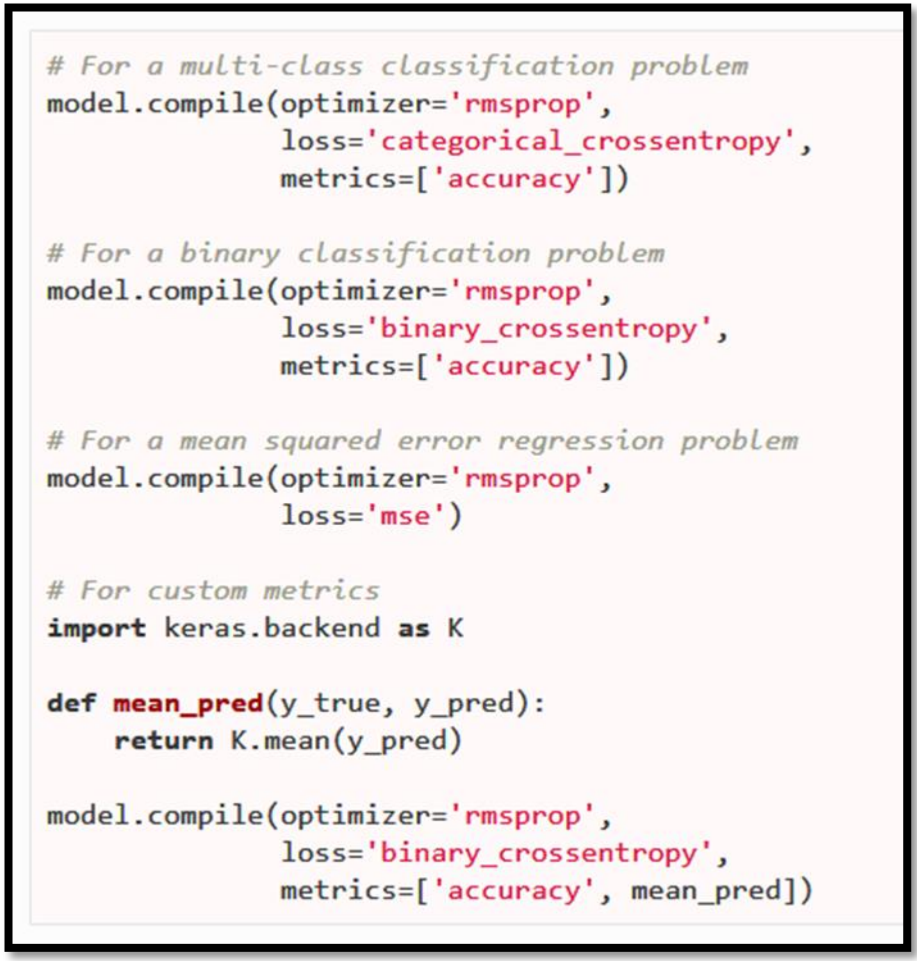
This is equivalent to:

```
model.add(Dense(64, activation='tanh'))
```

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

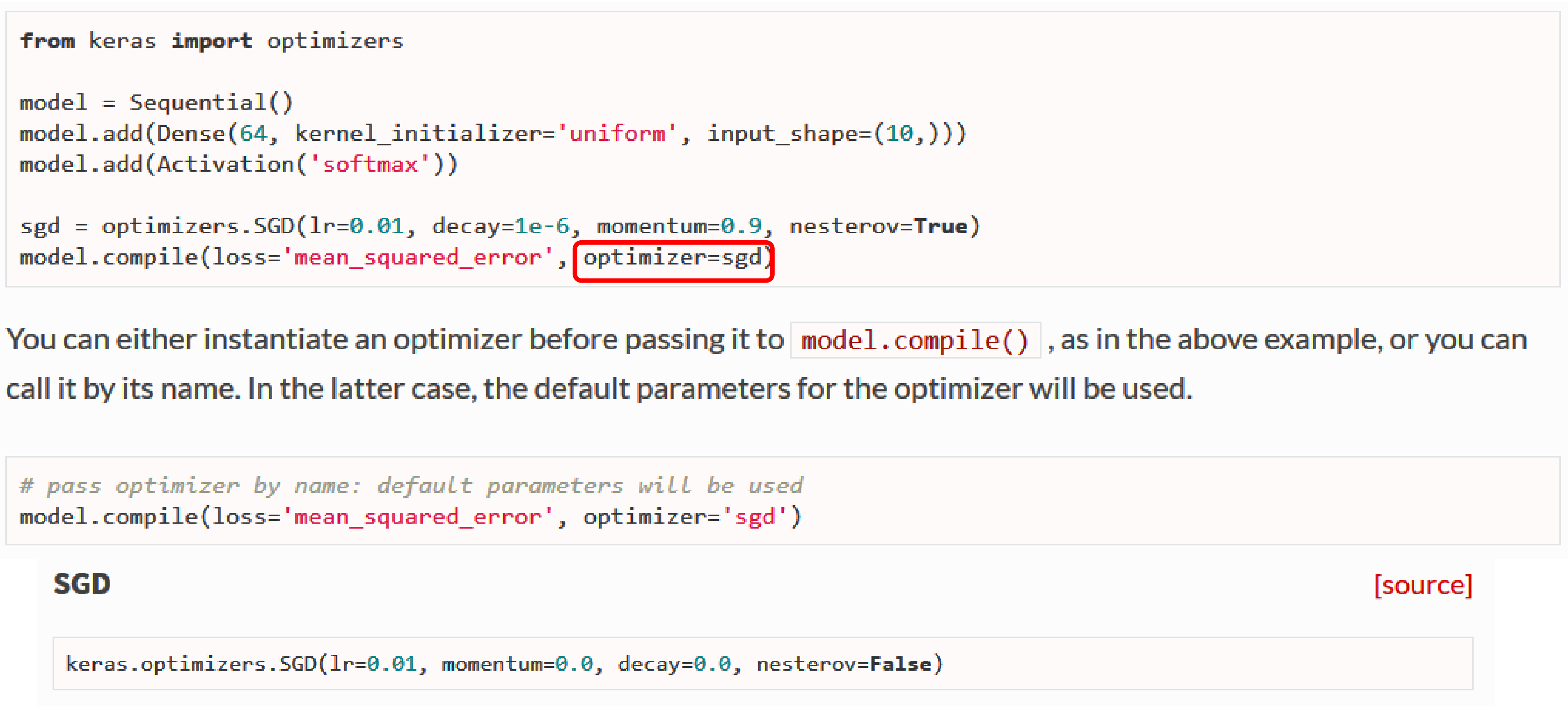
Compiling a Model

- ❑ Compiling a model means configuring the learning process
- ❑ A model can be compiled by providing
  - An optimizer
  - A loss function
  - A list of metrics
- ❑ The choices depend on the considered problem
  - Binary classification
  - Multi-class classification
  - Regression



```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')

# For custom metrics
import keras.backend as K

def mean_pred(y_true, y_pred):
    return K.mean(y_pred)

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy', mean_pred])
```

Optimizers



```
from keras import optimizers

model = Sequential()
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(Activation('softmax'))

sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

You can either instantiate an optimizer before passing it to `model.compile()` , as in the above example, or you can call it by its name. In the latter case, the default parameters for the optimizer will be used.

```
# pass optimizer by name: default parameters will be used
model.compile(loss='mean_squared_error', optimizer='sgd')
```

SGD

[source]

```
keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
```

Loss Function (1)

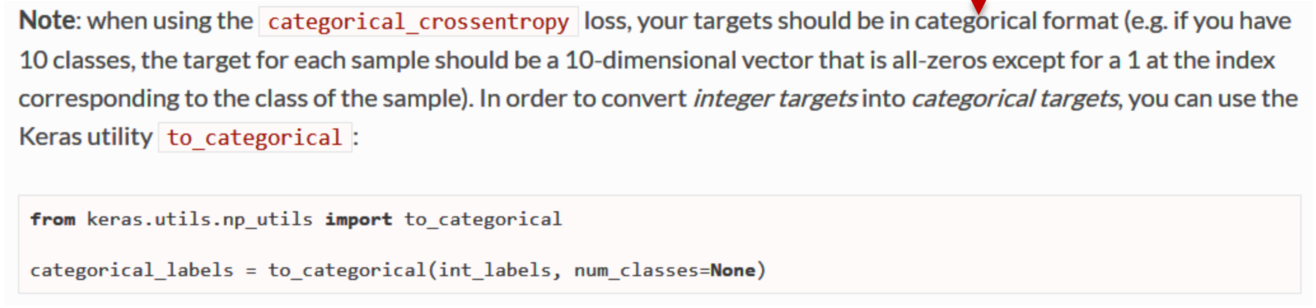
- Loss Function
  - a.k.a. objective function
  - a.k.a. optimization score function
- Measures the distance between actual and desired output
  - Drives the training process
- Depends on the task / output type
  - Classification vs regression

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

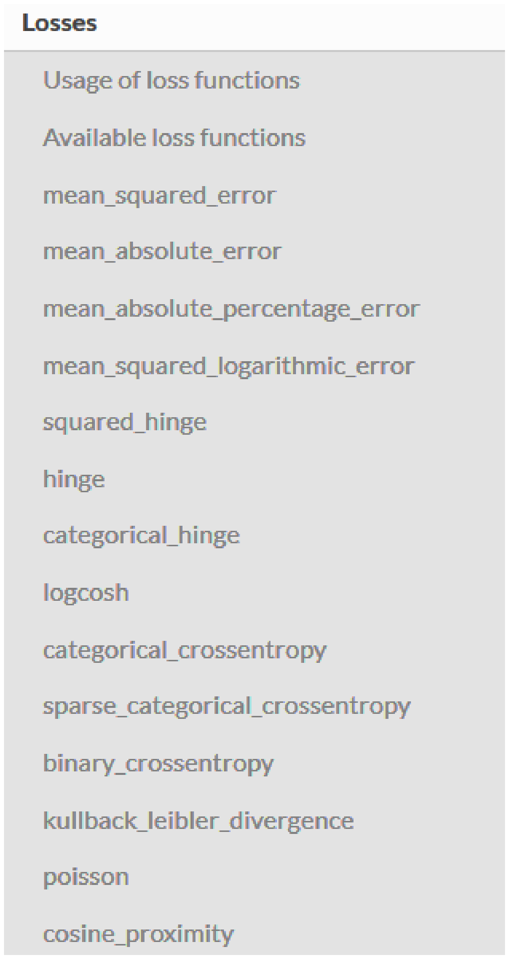
Loss Function (2)

- Can be selected from a set of predefined functions, or user-defined
- Good starting points
  - Classification: categorical/binary cross entropy
  - Regression: mean square error (L2)

one-hot representation


Note: 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 except 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 `to_categorical`:

```
from keras.utils.np_utils import to_categorical

categorical_labels = to_categorical(int_labels, num_classes=None)
```

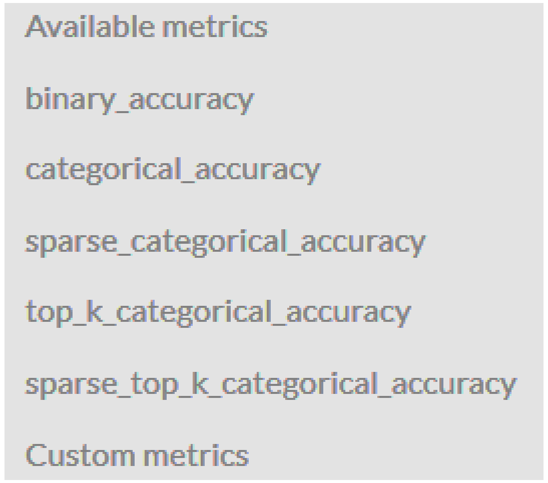

Losses

Usage of loss functions
Available loss functions
mean_squared_error
mean_absolute_error
mean_absolute_percentage_error
mean_squared_logarithmic_error
squared_hinge
hinge
categorical_hinge
logcosh
categorical_crossentropy
sparse_categorical_crossentropy
binary_crossentropy
kullback_leibler_divergence
poisson
cosine_proximity

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Metrics

"A metric function is similar to a loss function, except that the results from evaluating a metric are not used when training the model"



Available metrics

binary_accuracy

categorical_accuracy

sparse_categorical_accuracy

top_k_categorical_accuracy

sparse_top_k_categorical_accuracy

Custom metrics

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Resources

- Getting started with Keras
  - https://keras.io/getting-started/sequential-model-guide/
  - https://blog.keras.io/keras-as-a-simplified-interface-to-tensorflow-tutorial.html
- Focus on optimization algorithms
  - https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f
- Metrics
  - https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/
- Selected MNIST example
  - https://yashk2810.github.io/Applying-Convolutional-Neural-Network-on-the-MNIST-dataset/
  - https://elitedatascience.com/keras-tutorial-deep-learning-in-python
  - https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py