Università degli Studi di Padova

Implementing Deep Networks in Keras

Stefano Ghidoni

Dipartimento di Ingegneria dell'Informazione

Intelligent Autonomous Systems Lab

Agenda

- Keras: what is it?
- Getting started in 30s
- Building blocks: models and layers
- Network training & related tools
- Convolutional Neural Networks (CNNs) in Keras

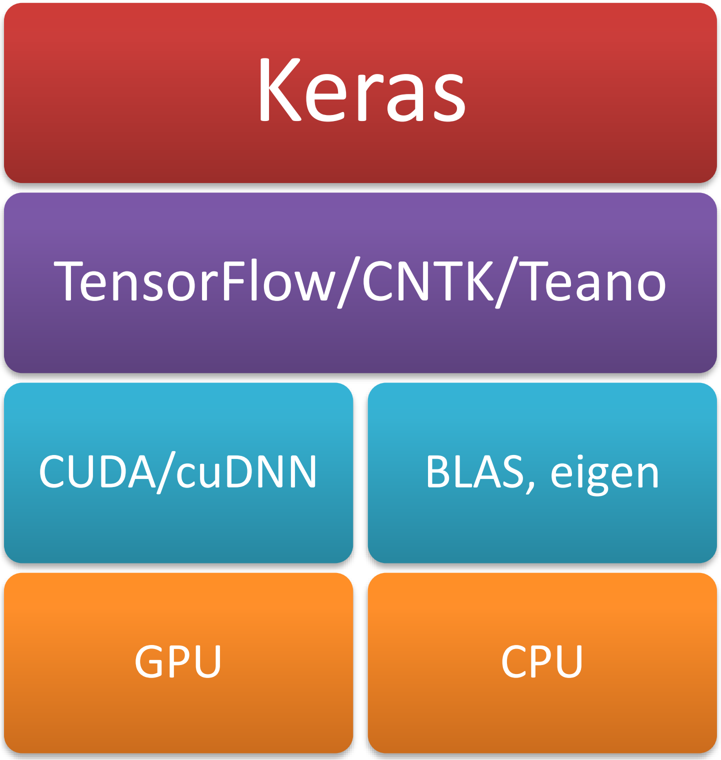
Keras

Simple. Flexible. Powerful.

Get started | API docs | Guides | Examples

Keras in a nutshell

- A simplified interface to TensorFlow
- Backends
  - TensorFlow
  - Theano
  - CNTK
- Supports CPU & GPU



Guiding principles

- User friendly
  - It is easy to do easy things
  - It is possible to do complex things
- Modular
  - Modules: neural layers, cost functions, optimizers, activation functions, regularization schemes
- Extendible
  - Easy to add new modules
- Python-based
  - Interactive environment

Getting started

30s to Keras

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

- Instantiate a sequential model

```
from keras.models import Sequential

model = Sequential()
```

30s to Keras

- Stack some layers

```
from keras.layers import Dense

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

- 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))
```

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)
```

- Evaluate performance

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

- Generate predictions

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

- The end

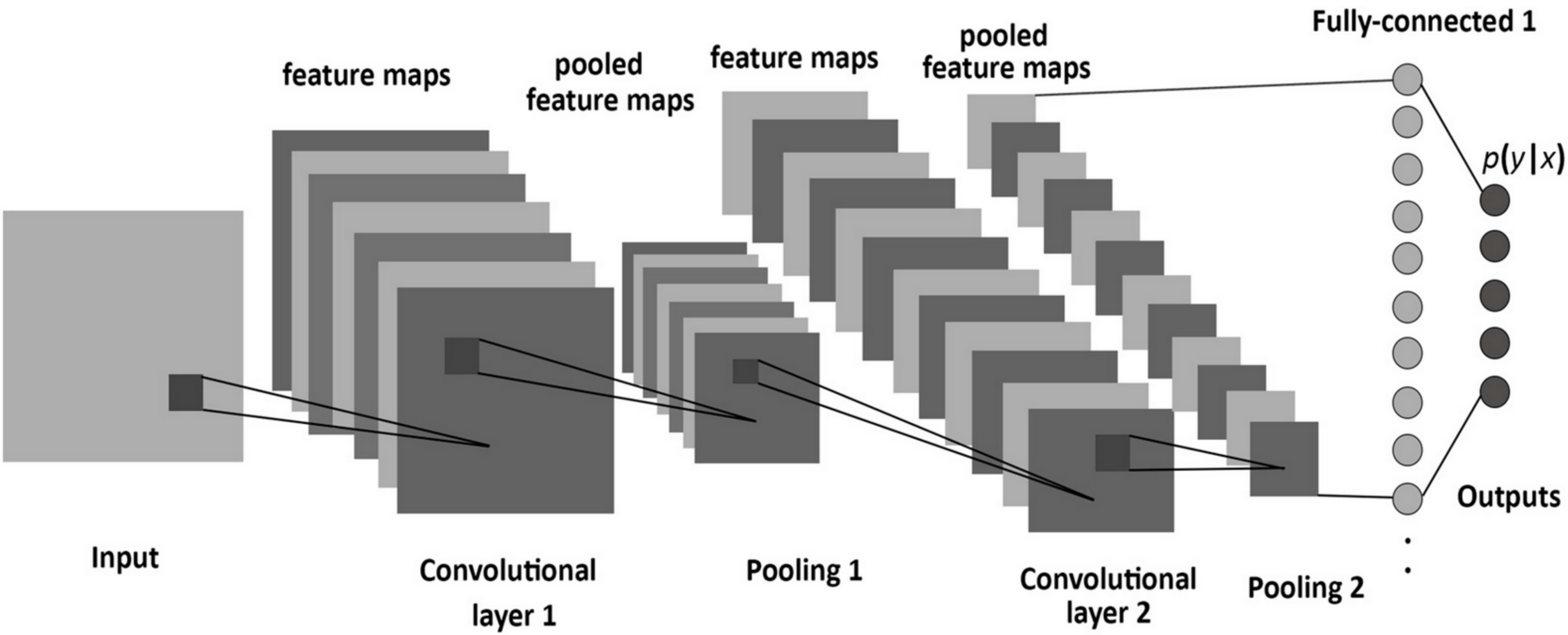
Models

- A way to organize & handle layers
- Sequential model
  - Linear stack of layers
- Model class API
  - Given an input tensor a and an output tensor b, creates all layers required to compute b from a
- Methods
  - compile (configures the model for training)
  - fit (training on a dataset)
  - evaluate (evaluation at test time)

CNNs

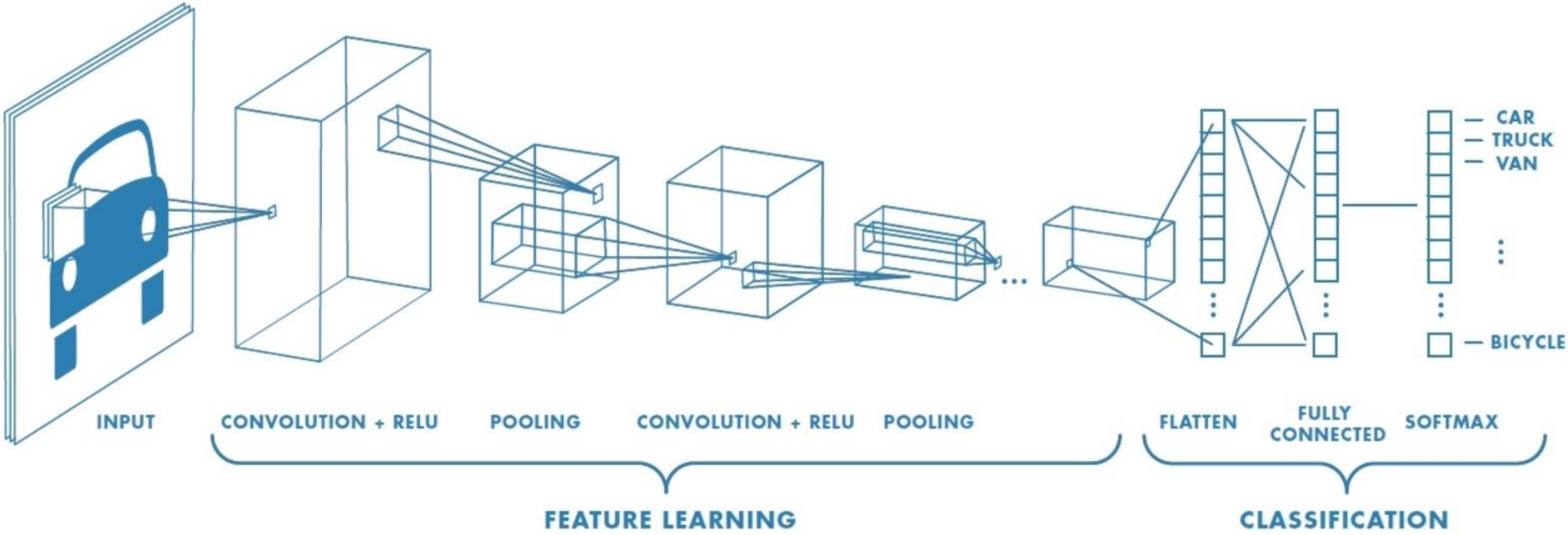
CNN – recall

- Typical CNN



CNN – recall

- Typical CNN

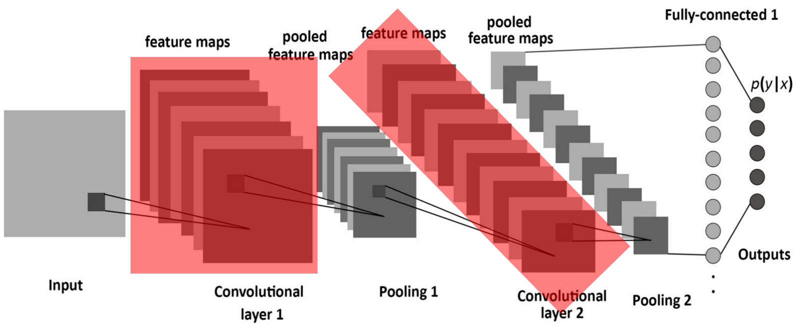


CNN – layers

- Convolutional Neural Network
- Typical layers:
  - Convolutional + ReLU
  - Pooling
  - Fully connected
  - Activations (Softmax)

Layers in Keras

- Convolutional layers
  - 1D (typically temporal)
  - 2D (typically spatial)




```
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"`.

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).

Conv2D arguments

- The argument list is huge!
- Rely on default values (our best friends)
  - Specify only the desired arguments

```
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)
```

- Code examples:

```
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))
```

Conv2D I/O

- Tensors & I/O
  - Tensor: typed multi-dimensional array (From TensorFlow)

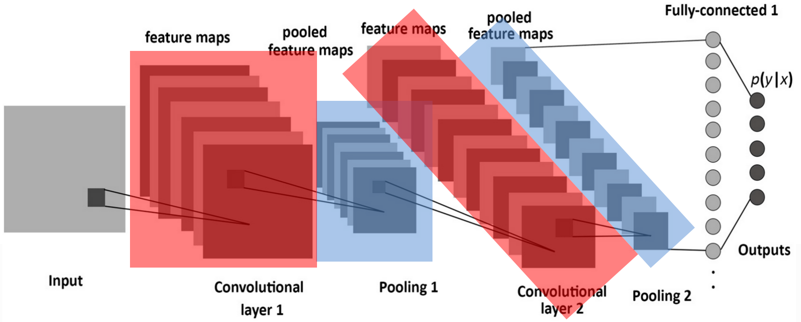
Input shape

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.

- Pooling layers (1D and 2D)



MaxPooling2D

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

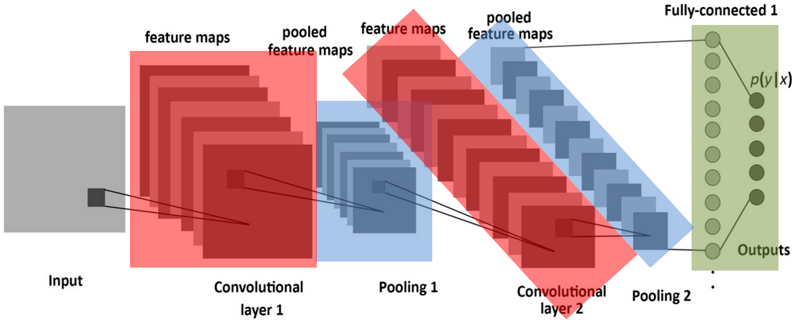
Max pooling operation for spatial data.

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".

Layers in Keras

- Dense layers
  - AKA fully connected




```
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`.

Layers in Keras

- Activations
  - Standalone layers
  - Embedded in forward layers
- Softmax activation often at the end of the last FC 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'))
```

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
- Depending on the problem at hand
  - Binary classification
  - Multi-class classification
  - Regression

Compiling a model

- The compile function – examples

```
# 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

- Several optimizers are provided
- Example:

```
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 (Stochastic Gradient Descent)

- SGD is a good starting point
  - Default values for every parameter!

SGD [source]

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

Stochastic gradient descent optimizer.

Includes support for momentum, learning rate decay, and Nesterov momentum.

Arguments

- **lr**: float >= 0. Learning rate.
- **momentum**: float >= 0. Parameter that accelerates SGD in the relevant direction and dampens oscillations.
- **decay**: float >= 0. Learning rate decay over each update.
- **nesterov**: boolean. Whether to apply Nesterov momentum.

Loss function

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

Loss function

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

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

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)
```

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

Working environment

Colab

- Colab is similar to Jupyter, but:
  - A tool for online programming
  - A remote GPU is available
  - Remote environment (e.g., files you have access to)
  - Hard to tune/install new libs
    - Keras is available!

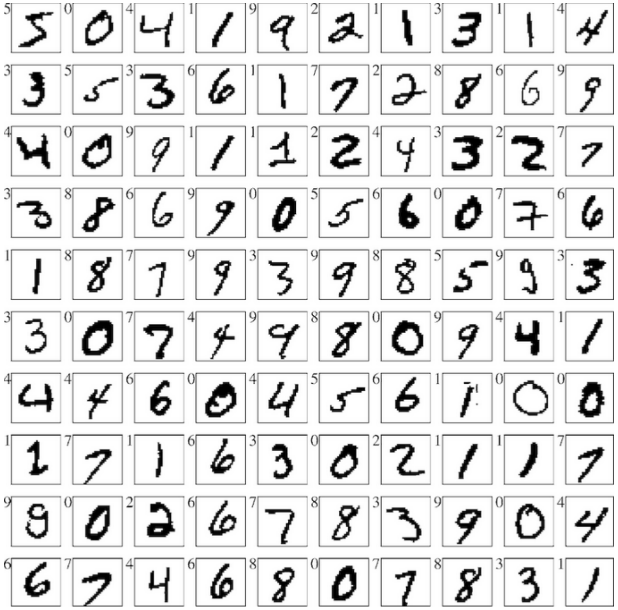
MNIST classification in Colab

- The next slides present the code needed to train a CNN on the MNIST dataset
- Try on your own
- An example is available at:

https://colab.research.google.com/github/AviatorMoser/keras-mnist-tutorial/blob/master/MNIST%20in%20Keras.ipynb

Training example – MNIST

- MNIST database of handwritten digits
  - Training set: 60k examples
  - Test set: 10k examples
- ML's hello world



http://yann.lecun.com/exdb/mnist/

Training example – MNIST

- Imports

```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
```

Training example – MNIST

- Set training parameters + read dataset + reshape

```
batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

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.

Training example – MNIST

- Type and format conversion, normalization

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

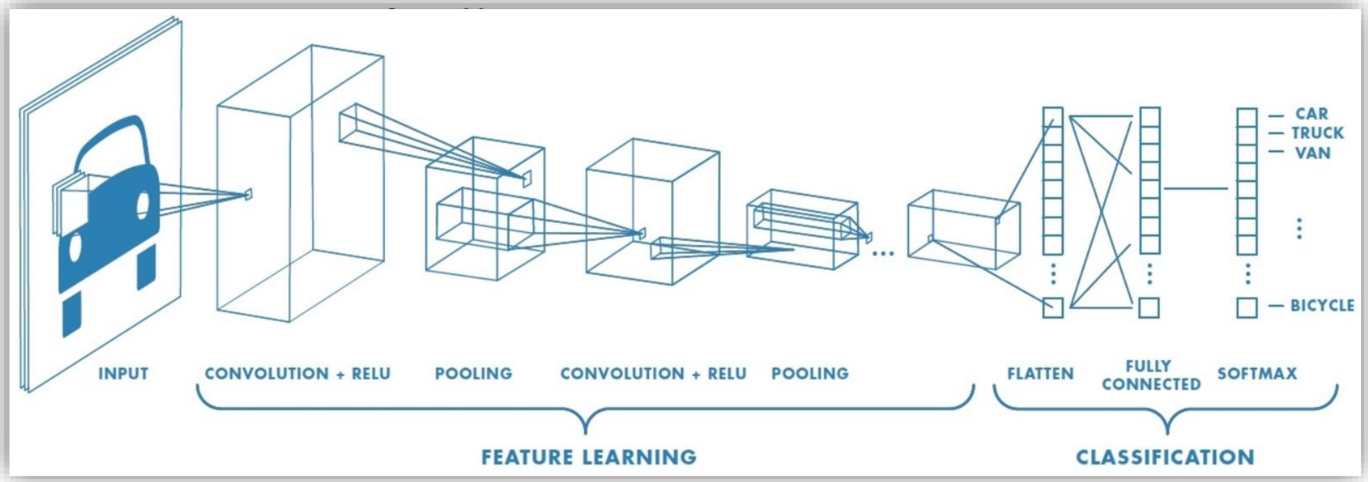
Training example – MNIST

- Model building

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Training example – MNIST

- Model building




```
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Training example – MNIST

- Compilation, fitting and output

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Resources

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
- 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

Resources

- Dropout
  - https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/
- Transfer learning
  - https://medium.com/@14prakash/transfer-learning-using-keras-d804b2e04ef8
  - http://cv-tricks.com/keras/fine-tuning-tensorflow/

Resources

- Code examples: https://keras.io/examples/
  - Including transfer learning examples

Code examples

Our code examples are short (less than 300 lines of code), focused demonstrations of vertical deep learning workflows.

All of our examples are written as Jupyter notebooks and can be run in one click in Google Colab, a hosted notebook environment that requires no setup and runs in the cloud. Google Colab includes GPU and TPU runtimes.

★ = Good starter example

Computer Vision

Image classification

- ★ Image classification from scratch
- ★ Simple MNIST convnet
- ★ Image classification via fine-tuning with EfficientNet
- Image classification with Vision Transformer
- Image Classification using BigTransfer (BiT)
- Classification using Attention-based Deep Multiple Instance Learning
- Image classification with modern MLP models
- A mobile-friendly Transformer-based model for image classification
- Pneumonia Classification on TPU
- Compact Convolutional Transformers
- Image classification with ConvMixer
- Image classification with EANet (External Attention Transformer)

Università degli Studi di Padova

Implementing Deep Networks in Keras

Stefano Ghidoni