Universidad Rey Juan Carlos

SUPERIOR TECHNICAL SCHOOL OF TELECOMMUNICATION ENGINEERING

Media studies and
Audiovisual Systems engineering

Bachelor's Degree Final Project

Vehicle Detection using Deep Learning

Author: David Pascual Hernández
Tutors: José María Cañas Plaza, Inmaculada Mora Jiménez

Academic year 2016/2017

Contents

List of Figures

List of Tables

Chapter 1

Framework

This chapter serves as a way to introduce the tools that I have employed during the development of this project. All of them are **open-source**. The transparency provided by the open-source platforms is a major advantage because the software can be joined together and adapted to our specific application, which is mainly written in **Python** [1].

JdeRobot

JdeRobot [2] is an open source middleware for robotics and computer vision. It has been designed to simplify the software development in these fields. It's mostly written in C++ language and it's structured like a collection of components (tools and drivers) that communicate to each other through **ICE interfaces** [3]. It is also compatible with **ROS** [4], which allows us to use ROS nodes and JdeRobot components simultaneously. This flexibility makes it very useful for our application. We're going to employ its *cameraserver* driver to capture images from different video sources.

cameraserver

According to JdeRobot documentation, this driver can serve both real cameras and video files. It interacts with other components thanks to the *Camera* interface.

[1] `https://www.python.org/`
[2] `http://jderobot.org`
[3] `https://zeroc.com/products/ice`
[4] `http://www.ros.org/`

In order to use *cameraserver*, we have to properly set its configuration file. These are the parameters that we must specify:

- A network address where the server is going to be listening.
- Parameters related with the video stream: URI, frame rate, image size, format.

Keras

As stated by **Keras** documentation [1]: ”Keras is a high-level neural network library, written in Python and capable of running on top of either TensorFlow or Theano”. TensorFlow and Theano are open-source libraries for numerical computation optimized for GPU and CPU that Keras treats as its ”backends”. This Currently, I'm running Keras on top of **Theano** [5] optimized for CPU, but it's quite easy to switch from one backend to another.

We're going to analyze the main elements that make up a neural network built with Keras, starting with the *model* object, its core component.

Models

Every neural network in Keras is defined as a *model.* For those neural networks which can be built as a stack of *layers* 1.2, Keras provides the ***Sequential model*** object. It is also possible to build more complex models with multiple outputs and shared layers using the Keras *functional API.*

SEQUENTIAL MODEL METHODS!!

Layers

As it has been said before, the models are usually built as a stack of layers. These layers are added to the model using the *.add()* method, inside of which the kind of layer is declared and its particular parameters are set. Several kinds of layers are available, but only the ones that have been used in this project are going to be described:

Convolutional layer This particular layer is the one that turns the neural network into a **convolutional neural network (CNN)**. They are formed by a fixed number of

[5] `http://deeplearning.net/software/theano/index.html`

filters/kernels with a fixed size. These filters are convolved along the input image, generating each one a feature or activation map which will tell us whether the feature learned by that particular filter is present in the input image or not. Keras provides different kinds of convolutional layers depending on the input dimensions: *Conv1D*, *Conv2D* and *Conv3D*. These are the main arguments required by Keras to define a convolutional layer:

- ***filters***: number of filters.
- ***kernel_size***: width and height of the filters.
- ***strides***: how many pixels the filter must be shifted before applying the next convolution. Output size depends on this parameter. It defaults to 1.
- ***padding***: it can be *valid* or *same*. If *valid* mode is set, no padding is applied, resulting in a reduced output. However, if *same* mode is set, the input will be padded with zeros in order to produce an output that preserves the input size. It defaults to *valid*

Pooling layer It shifts a window of a certain size along the input (e.g. a feature map) applying an operation (mean or maximum) that will return a condensed version of it. Depending on the dimensions of the input and the operation applied, Keras provide several pooling layers: MaxPooling1D, MaxPooling2D, MaxPooling3D, AveragePooling1D... The main arguments required by Keras to define these layers are:

- ***pool_size***: size of the window that is shifted along the input. It can also be interpreted as the factor by which the input is going to be downsampled.
- ***strides***: how many pixels the window must be shifted before applying the desired operation.

Dense layer Fully-connected layers in Keras are defined as *Dense layers*. In a **fully-connected layer**, every neuron is connected to every activation (output) of the previous one. The main argument of this layer is:

- ***units***: number of neurons.

Activation layers In Keras models, activations can be declared employing the *.add()* method, or as an argument of the corresponding layer. Keras provides several

activation functions, such as sigmoid, linear, ReLU and softmax. The only argument that must be provided to activation layers is the name of the desired activation function. These are the ones that have been used during the development of the project:

- **ReLU (Rectified Linear Unit)**: This activation function introduces non-linearity right after each convolutional layer, allowing the CNN to learn more complex features. It's defined as:

$$g(z) = \max(0, z) \tag{1.1}$$

- **Softmax** This activation function is very useful in the output layer of classification tasks. It takes a vector of real values z and returns a new vector of real values in the range [0,1]. The N lements of the output vector can be considered probabilities because the softmax function ensures that they sum up to 1. It is defined as follows:

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\Sigma_j \exp(z_j)} \quad \text{for } j = 1, ..., N \tag{1.2}$$

This equations (Goodfellow et al., 2016 [2]) are equivalent to the ones employed by Keras, defined by its backends Theano and Tensorflow.

Flatten layer It "flattens" the input. For instance, it converts the activation maps returned by the convolutional layers into a vector of neurons before being connected to a dense layer. It takes no arguments.

Dropout layer It's considered a regularization layer, because it's main purpose is to avoid over-fitting. Dropout (Srivastava et al., 2014 [3])it's a technique that randomly "switches-off" a fraction of hidden units during training (both forward and backward propagation). This layer, as other regularization layers (i.e. GaussianNoise layer), is only active during training. It's main argument is:

- ***rate***: fraction of units that must be dropped.

Image Preprocessing

Losses

Metrics

Optimizers

Callbacks

Utils

Keras include a module for multiple supplementary tasks called **Utils**. The most important functionality for the project provided by this module is the ***.HDF5Matrix()*** **method**. It reads the **HDF5** [6] datasets that are going to be used as inputs to the neural networks.

Scikit-Learn

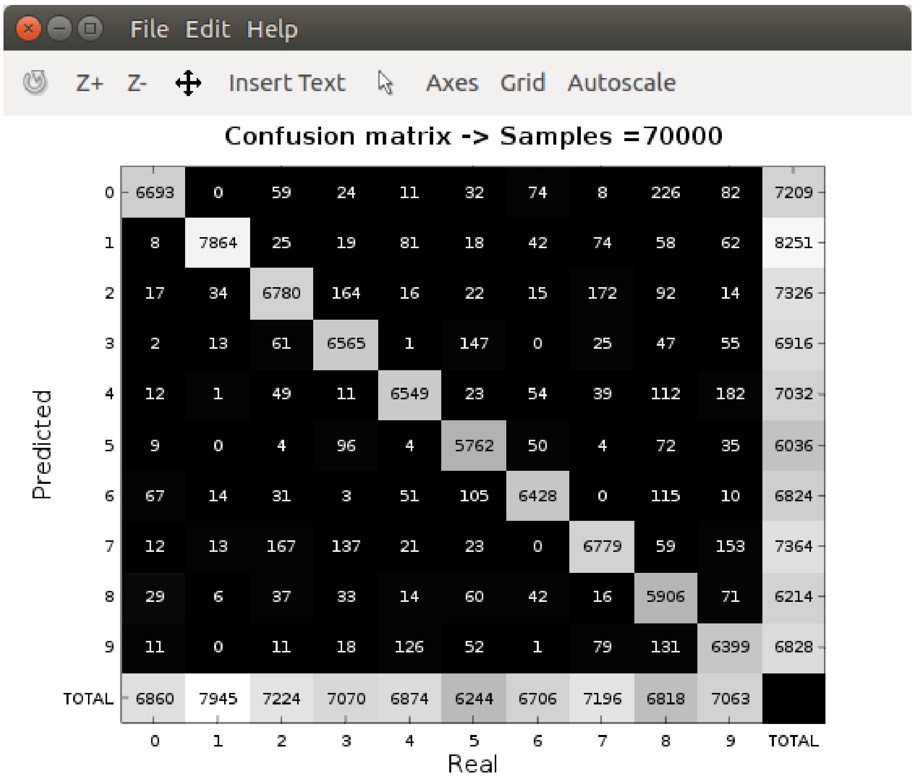
Scikit-Learn [7] is a machine learning library that includes a wide variety of algorithms for clustering, regression and classification. It can be used during the whole machine learning process: preprocessing, training, model selection and evaluation.

We're going to employ Scikit-Learn functions to evaluate the neural networks that we have developed with Keras. Using a tool that is independent from Keras allows us to compare the results achieved by different neural network libraries (e.g. Keras and Caffe).

Octave

GNU Octave [8] is a scientific programming language compatible with Matlab. It provides powerful tools for plotting. We're going to use these tools to visualize the data collected with Scikit-Learn about the performance of our models. An example of a visualization using Octave can be seen in the figure 1.1.

[6] `https://support.hdfgroup.org/HDF5/doc/H5.format.html`
[7] `http://scikit-learn.org/stable/index.html`
[8] `https://www.gnu.org/software/octave/`



Confusion matrix -> Samples =70000

	0	1	2	3	4	5	6	7	8	9	TOTAL
0	6693	0	59	24	11	32	74	8	226	82	7209
1	8	7864	25	19	81	18	42	74	58	62	8251
2	17	34	6780	164	16	22	15	172	92	14	7326
3	2	13	61	6565	1	147	0	25	47	55	6916
4	12	1	49	11	6549	23	54	39	112	182	7032
5	9	0	4	96	4	5762	50	4	72	35	6036
6	67	14	31	3	51	105	6428	0	115	10	6824
7	12	13	167	137	21	23	0	6779	59	153	7364
8	29	6	37	33	14	60	42	16	5906	71	6214
9	11	0	11	18	126	52	1	79	131	6399	6828
TOTAL	6860	7945	7224	7070	6874	6244	6706	7196	6818	7063	

Figure 1.1: Example of a confusion matrix visualization using Octave.

Bibliography

[1] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting.