Smart Area CNR Pisa - Smart Parking - Technical Report

Giuseppe Amato, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro, Claudio Vairo

Introduction

The Smart Parking application has the purpose of determining the number and the position of the available slots in the parking lot of the CNR Area in Pisa. To achieve this goal, nine smart cameras (video cameras with computational capabilities) are mounted on top of the roof in front of the parking to be monitored, and visual computing algorithms have been developed in order to recognize whether a parking slot is empty or occupied by a car.

The information produced by the cameras is sent to a central database located in a cloud-computing environment by means of a RESTful web service.

Hardware

Raspberry PI and Camera Module

We used the Rapsberry Pi 2 model B equipped with a camera module (see Figure 1) as platform for acquiring and analyzing video information of the parking. They are equipped with:

- BCM2836 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM DDR2
- 32GB micro SD card for storage

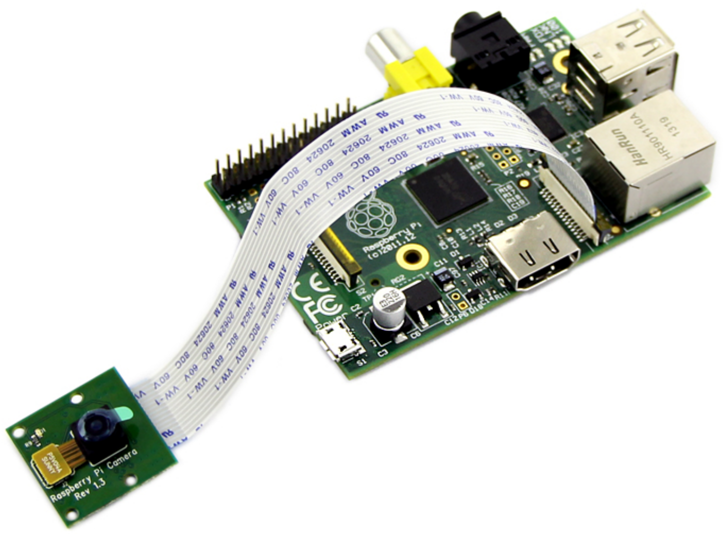


Figure 1 - Rasberry Pi with camera module

The camera module is a 5MP fixed-focus camera that supports 1080p30, 720p60 and VGA90 video modes, as well as stills capture. The view angles of the camera are 53.50° horizontally and 41.41° vertically. The allowed resolutions are reported in Table 1.

Size	**Aspect Ratio**	**Frame Rate**
2592x1944	4:3	1-15fps
1920x1080	16:9	1-30fps
1296x972	4:3	1-42fps
1296x730	16:9	1-49fps
640x480	4:3	42.1-60fps

Table 1 - Raspberry Pi camera module resolutions

We put the Raspberry Pi in outdoor boxes (see Figure 2) that have been installed on top of the roof of the building in front of the parking lot. Each box is provided with a heater/blower temperature control system, and each Raspberry Pi is connected to the network by an Ethernet cable.
We deployed nine cameras in order to cover half of the parking lot of the rear part of the CNR area. Other smart cameras installed by another group working to the project cover the other half of the parking lot.

Figure 2 - Outdoor box

Software

We use the OpenCV[1] library to elaborate the images acquired by the cameras and a deep learning approach to determine whether a parking slot is available or not. We then communicate the parking slot updates by invoking a RESTful web service responsible of storing the updates in the cloud and of providing the information about the parking lot to the user.

Deep Learning

Deep Learning (DL) [1] is a branch of Artificial Intelligence that aims at developing techniques that allow computers to learn complex perception tasks such as seeing and hearing at human levels of performance. It provides near-human level accuracy in image classification, object detection, speech recognition, natural language processing, vehicle and pedestrian detection, and more.
The traditional approach to the classification problem [2], [3], [4], [5], uses ad-hoc functions to extract particular features from the image that are considered to be indicative of certain objects. For example, hard corners and straight edges might be believed to indicate the presence of manmade objects in the scene. The outputs of these feature extraction functions are then fed to a classification function, which determines if a particular object has been detected in the image. However, this approach leads to weak and false-alarm prone detectors and present the following problems:

- It is very hard to think of robust, reliable features, which map to specific object types.
- It is a massive task to find the right combination of features for every type of object to classify.
- It is very difficult to design functions that are robust to translations, rotations and scaling of objects in the image.

All these problems make very hard developing high accuracy object detectors and classifiers for a broad range of objects.

[1] http://opencv.org/

The Deep Learning approach, on the other hand, exploits a large number of ground-truth labeled images to discover which features and combinations of features are most discriminative for each class of objects to be recognized and builds a combined feature extraction and classification model (this phase is called *training* the model).
The model thus obtained can be deployed and it is not only able to classify specific objects it was trained on, but it is also able to recognize previously unseen similar objects.
The Deep Learning approach is shown in Figure 3 (borrowed from the NVIDIA Deep Learning web course).

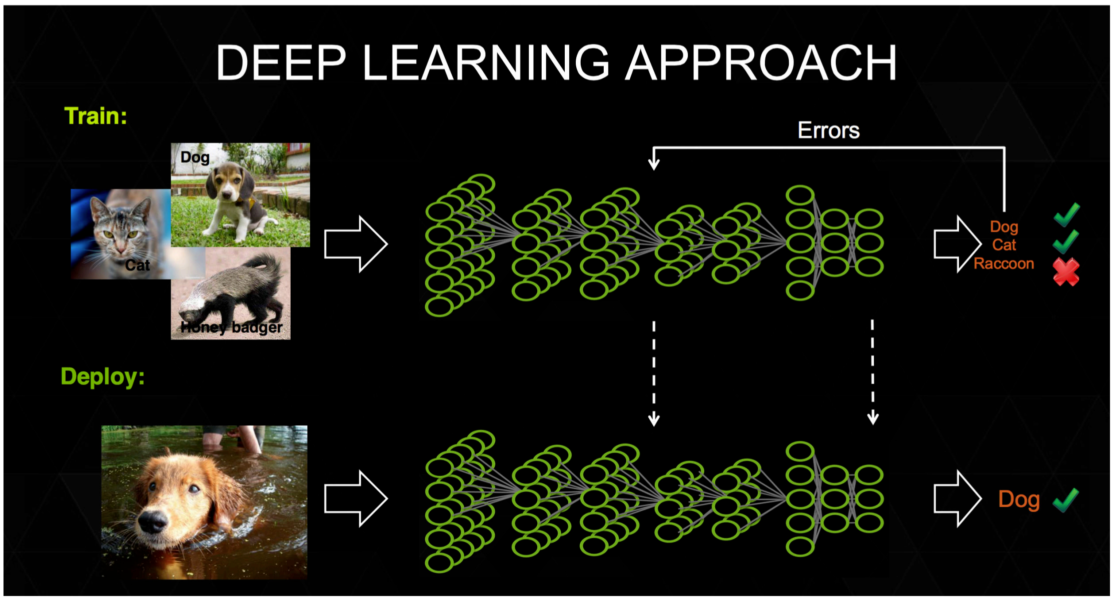


Figure 3 - Deep Learning Approach

A large set of images, which compose the training set, is fed to a neural network composed of a possibly large number of hidden layers (whence the term "deep"). Each of these hidden layer perform mathematical computation on the input and produce an output that is fed as input to the following layer. The final outputs of this network are the classes for which the network has been trained on.
The training phase is usually extremely computationally expensive and it may take a long time. On the other hand, once the network has been trained and the classifier has been initialized accordingly, the run time phase of prediction is quite fast and efficient.

Our Solution

Dataset

We collected a large number of screenshots of the parking lot in different days, from different points of view, with different occlusion patterns, and in different weather conditions. We then built a mask for each parking slot in order to split the original image in several smaller images (we refer to these small images as

patches), one for each parking slot (see Figure 4). Each of these patches is a square of size proportional to the distance from the camera, the nearest are bigger then the farthest.
We built a dataset of about 10K patches and this constitutes the training set of our deep learning network. The training set captures different situations of luminosity, including partial occlusion due to obstacles (lamps, trees or other cars) and partial or global shadowed cars (see Figure 4). This allows the generated classifier to be able to distinguish almost every situation that can be found at run time. Of course total occlusions and nightly situations cannot be addressed with this approach.

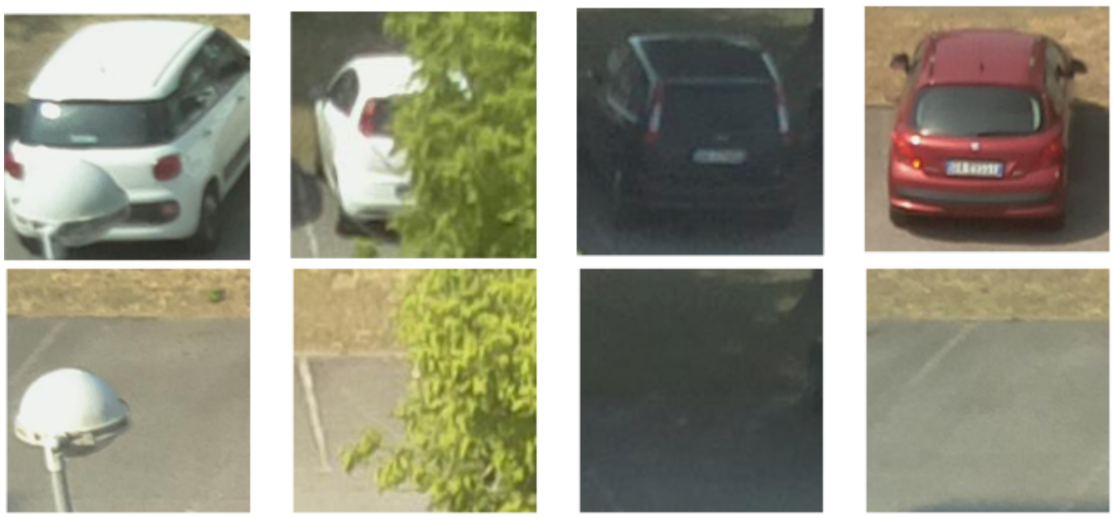
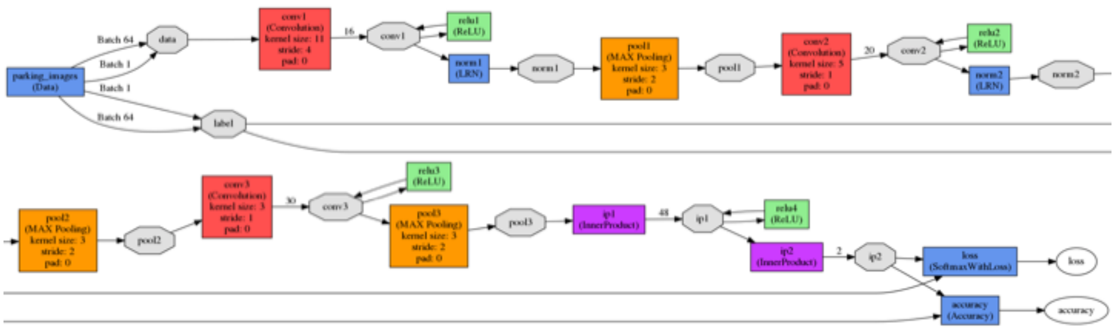


Figure 4 - Training set samples

Caffe Framework

We used the Caffe framework[2] for Deep Learning to train the neural network and initialize the classifier. Caffe is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. It is very fast due to its highly optimized C/CUDA backend, which integrates GPU acceleration. It provides command line, python and MATLAB interfaces and it is very easy to use since it allows defining the network and the training/testing phases by writing simple human-readable text files.
The neural network that we adopted is presented in Figure 5. It is composed of 14 levels, 5 of which trainable (3 convolutions and 2 fully-connected). It is a smaller network compared to the typical Deep Learning neural networks, since we need that the classifier will be able to predict only two classes: busy and available.

[2] http://caffe.berkeleyvision.org/


parking_images (Data)
Batch 64
Batch 1
Batch 1
Batch 64
data
label
conv1 (Convolution) kernel size: 11 stride: 4 pad: 0
16
conv1
relu1 (ReLU)
norm1 (LRN)
norm1
pool1 (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool1
conv2 (Convolution) kernel size: 5 stride: 1 pad: 0
20
conv2
relu2 (ReLU)
norm2 (LRN)
norm2
pool2 (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool2
conv3 (Convolution) kernel size: 3 stride: 1 pad: 0
30
conv3
relu3 (ReLU)
pool3 (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool3
ip1 (InnerProduct)
48
ip1
relu4 (ReLU)
ip2 (InnerProduct)
2
ip2
loss (SoftmaxWithLoss)
loss
accuracy (Accuracy)
accuracy
Figure 5 – The neural network used in our solution

The training phase generates the weights of the neural network that we used to initialize the classifier provided by Caffe according to our application scenario. To this purpose, we implemented a JNI interface to use the C interface provided by Caffe in order to instantiate and use the neural network. We then created a library containing the classifier initialized with the trained neural network, and we integrated it in our software that has been eventually deployed on the Raspberry Pi.

Algorithm

The software we implemented first loads the trained neural network generated by Caffe, then it loads the masks to detect the individual parking slots and to split the image in the patches. Please note that each camera has its own set of masks because each camera sees a different portion of the parking lot with different angles of view. However, this is a one-time process to be done only when the camera is installed and we implemented a semi-automatic software tool to generate the masks. With this tool, the user can load an image captured by the camera and by simply clicking in the center of each parking slot, the corresponding mask will be automatically generated with the proper size.

The rest of the computation is organized in two main run-cycles, one smaller of 15 seconds called *slot monitoring cycle* and one larger of one minute, composed of four slot monitoring cycles, called *slot update cycle*.
In the slot monitoring cycle the status of each slot is measured and compared to the previous consistent state. The slot update cycle determines if there has been an actual slot status change according to the values read in the four slot monitoring cycles. In particular, if a slot status change is detected in at least three slot monitoring cycles, then we can conclude that the slot status has been actually updated. This is done to avoid failures due to temporary maneuvering cars, or detection errors.

An iteration of the slot monitoring cycle performs the following operations:

1. Capture the current frame.
2. Split the frame in the set of patches according to the masks for that camera.
3. Perform the prediction for each patch by interrogating the classifier.
4. As a result, a matrix, containing the score for both the *busy* and *available* classes for each slot, is returned.

5. Compare the occupancy level for each slot with the status of the previous consistent state for that slot. If a parking slot status change is detected, a counter (called *counterUpdate)* for that slot is incremented.

If the slot update cycle determines a change in the status of a parking slot (i.e. the *counterUpdate* has a value greater or equal to 3), a *slot status update message* is sent to the RESTful web service responsible of storing the information about the parking lot in the cloud. The message is formatted in a JSON format according to the scheme reported in Table 2 and it is sent by means of a POST primitive.

Name	Format	Meaning
DATE	String	Timestamp of the status update
STATUS	int	New status of the slot (0,1)
MAC	String	MAC address of the camera generating the update
POSITION	int	Position of the updated slot
CONFIDENCE	int	Confidence level of the new status (0-100)
WEIGHT	double	Accuracy of the camera for that slot (0-1)

Table 2 - Structure of the slot status update message

Results

We performed some preliminary tests by placing the camera both on the roof of the building (Figure 6) and behind the window of an office (Figure 7) with a lower angle of view and an oblique perspective.
In both cases, we obtained quite good prediction results from the deep learning classifier, considering that trees, lamps, other cars and shadows occlude some cars.

Acknowledgments

This work has been funded by the DIITET Department of CNR, in the framework of the ”Renewable Energy and ICT for Sustainability Energy” project.

References

[1]. Y. Bengio. Learning deep architectures for A.I. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.
[2]. T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence*, IEEE Transactions on, 24(7):971–987, 2002.
[3]. V. Ojansivu and J. Heikkila. Blur insensitive texture classification using local phase quantization. In *Image and signal processing*, pages 236–243. Springer, 2008.

[4]. J. Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74,

1999.

[5]. E. Rahtu, J. Heikkila¨, V. Ojansivu, and T. Ahonen. Local phase quantization for blur-insensitive image analysis. *Image and Vision Computing*, 30(8):501–512, 2012.

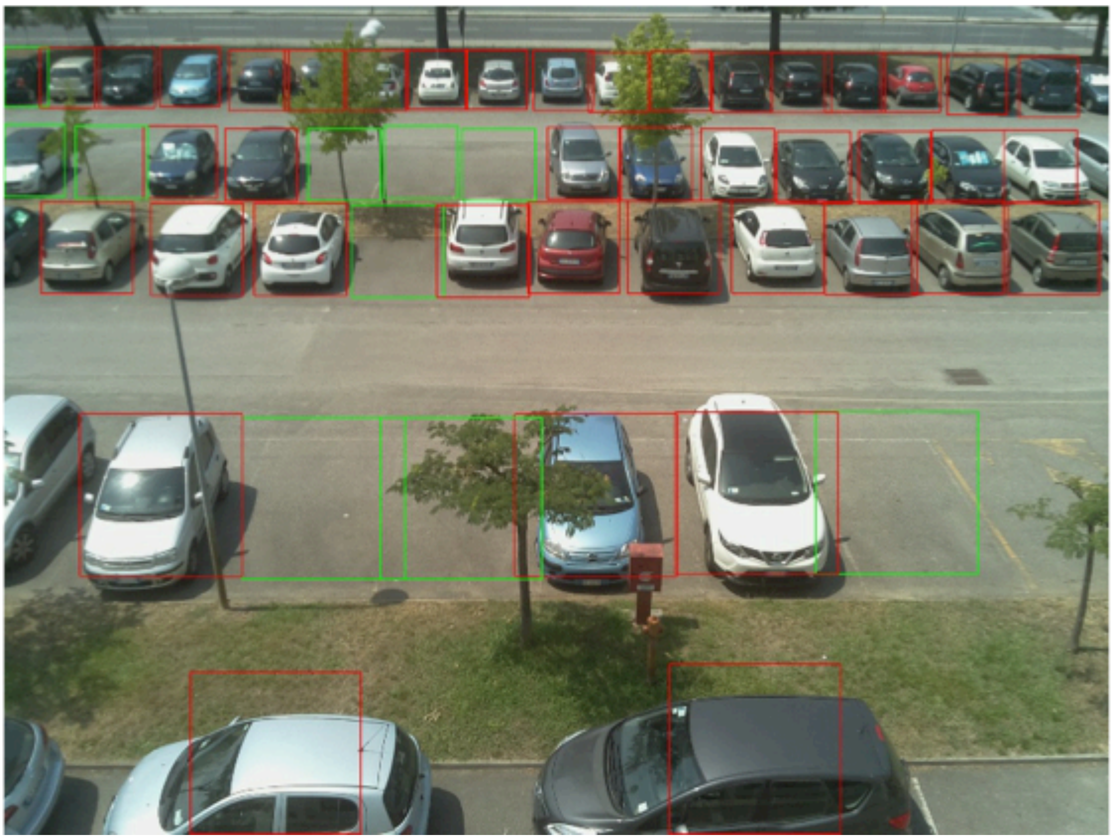


Figure 6 - Parking slot status detection example from roof

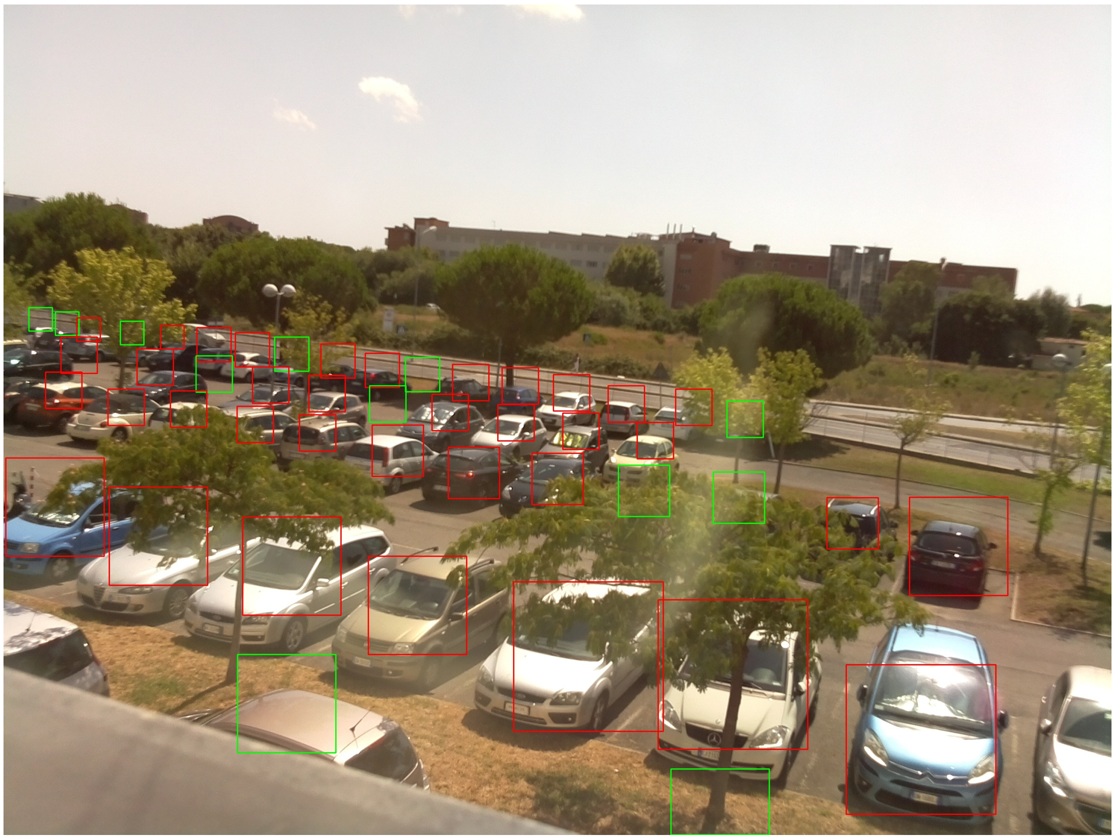


Figure 7 - Parking slot status detection example from office