POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea in Ingegneria Informatica

FACE RECOGNITION WITH CONVOLUTIONAL NEURAL NETWORKS ON LOW POWER ARCHITECTURES

Relatore: Prof. Marco TAGLIASACCHI
Correlatore: Ing. Luca BAROFFIO

Tesi di Laurea di:
Luca BONDI
Matr. 797195

Anno Accademico   2013 / 2014

Contents

List of Figures

List of Tables

Abbreviations

VSN	**V**isual **S**ensor **N**etwork
CNN	**C**onvolutive **N**eural **N**etwork
SVD	**S**ingular **V**alues **D**ecomposition
KLT	**K**arhunen **L**oève **T**ransform
PCA	**P**rincipal **C**omponent **A**nalysis
ATC	**A**nalyze **T**hen **C**ompress
CTA	**C**ompress **T**hen **A**nalyze
SVM	**S**upport **V**ector **M**achine
SoC	**S**ystem **o**n a **C**hip
SDRAM	**S**ynchronous **D**ynamic **R**andom **A**ccess **M**emory
UART	**U**niversal **A**synchronous **R**eceiver **T**ransmitter
SPI	**S**erial **P**eripheral **I**nterface
I2C	**I**nter **I**ntegrated **C**ircuit
I2S	**I**ntegrated **I**nterchip **S**ound
CSI	**C**amera **S**erial **I**nterface
DSI	**D**isplay **S**erial **I**nterface
GPIO	**G**eneral **P**urpose **I**nput **O**utput
DMA	**D**irect **M**emory **A**ccess
GPU	**G**raphical **P**rocessing **U**nit
DSP	**D**igital **S**ignal **P**rocessor
VFP	**V**ector **F**loating-**P**oint

Sommario

La crescente diffusione di dispositivi integrati a basso consumo alimentati a batteria, sia in ambito industriale che scientifico, richiede lo sviluppo di algoritmi di visione artificiale energeticamente efficienti. Gli stessi dispositivi integrati sono inoltre generalmente utilizzati in ambienti problematici, nei quali la larghezza di banda per le comunicazioni risulta fortemente limitata.

Lo stato dell'arte nell'ambito del riconoscimento facciale è rappresentato da algoritmi basati su Reti Neurali Convolutive. Su diversi dataset di volti, fotografati in condizioni ambientali non controlalte, le Reti Neurali Convolutive si sono dimostrate accurate quanto gli esseri umani nel riconoscimento di volti. Applicazioni quali l'autenticazione basata sul volto, il tracciamento degli individui e la sorveglianza a scopi militari possono trarre beneficio da dispositivi integrati, portabili ed autonomi capaci di riconoscere accuratamente un volto con il minor dispendio possibile di energia, anche in condizioni di comunicazione particolarmente avverse.

Questo lavoro ha l'obiettivo di ottimizzare l'implementazione esistente di una Rete Neurale Convolutiva per il riconoscimento facciale al fine di incontrare le necessità dei dispositivi a bassa potenza alimentati a batterie, anche quando operano in condizioni ambientali estreme. La Rete Neurale Convolutiva di partenza viene prima semplificata al fine di ottenerne una versione con un tempo di esecuzione ridotto, ma che allo stesso tempo consenta di mantenere un alto livello di accuratezza. Una architettura di codifica ad-hoc viene quindi sviluppata per comprimere le dimensioni dei dati estratti dalla Rete Convolutiva e mostrare che, anche in condizioni di comunicazione particolarmente difficili, la perdita in termini di accuratezza è minima.

Abstract

The growing diffusion of battery operated low-power computing platforms, both in industrial and scientific environments, calls for the development of energy efficient algorithms for a variety of computer vision applications. Furthermore, these pervasive devices are usually operated in challenging environments, where communications are strongly limited in bandwidth.

The state of the art in face recognition field is represented by Convolutional Neural Networks based algorithms. Convolutional Neural Networks are as accurate as humans on a variety of challenging faces datasets, whose pictures are taken with no constraints on environmental conditions. Applications as face-based authentication, people tracking and military surveillance can benefit of pervasive, portable and autonomous devices capable of performing accurate face recognition with a small amount of energy, even when networking conditions are awful.

This work aims at optimizing an existing Convolutional Neural Network implementation for face recognition to suit the needs of battery operated low-power devices, even when operating in difficult environmental conditions. The existing CNN is first simplified to obtain a faster execution algorithm with the minimum possible loss in recognition accuracy. An ad-hoc coding architecture is then developed to compress the output features of the fast CNN and show that, even with a very limited bandwidth, the recognition accuracy loss is negligible.

Introduction

In the field of computer vision, automatic face recognition is a task that has received attention since the 60s with Bledsoe [2] and in the 70s with Kanade [3], mainly focusing on the idea of finding a model for faces based on points and relationships between them. These techniques were shown to be too sensitive to changes in image conditions. In the 80s Sirovich and Kirby proposed Eigenfaces [4] a compact KLT based representation of human faces. Based on this work, in the 90s Turk and Pentland [5] developed a computationally efficient face tracking and recognition system with images acquired in a controlled environment.

In the last decade a number of face recognition datasets [6], [7], [8] have been created to challenge the development of robust algorithms capable of recognizing faces in unconstrained natural environments. Different methods based on high dimensionality handcrafted features have been proposed in the last years [9], [10], together with alternative methods based on Convolutional Neural Networks and Deep Learning techniques [11], [12]. Deep Convolutional Neural Networks have been proven to perform at the same level as humans in terms of face recognition accuracy [13].

A number of applications could benefit from the development of energy efficient face recognition algorithms, able to run on low-power devices: access control, privacy-critical video surveillance, people tracking, military surveillance. Moving the computational effort of detecting and recognizing faces from a centralized node directly to the camera that acquires the images, allows to avoid bandwidth expensive image or video streaming and reduces the risk of privacy violations due to network intrusions.

The growing number of low-power pervasive computing platforms available nowadays in industrial and scientific applications enforces the development of distributed intelligence Visual Sensor Networks.

A basic Visual Sensor Network consists of a group of nodes, each equipped with a low power embedded processor, an energy source, one or more image sensors and a network adapter for communications [14]. These low power sensor nodes send data to a powerful central node, the sink, which takes care of collecting and processing all the data coming from the sensor nodes or even from other VSNs.

In a traditional VSN a sensor node is used to acquire an image or a video, compress it - resorting to JPEG or H.264/AVC - and send it to a sink node through a network infrastructure. The whole elaboration process takes place in the sink node, where features are first extracted from the compressed image or video and then used for specific applications. This workflow is known as Compress-Then-Analyze.

Conversely the Analyze-Then-Compress paradigm [15] proposes another working schema: features extraction is performed directly on sensor nodes and a compact representation of the features is sent to the sink node. This kind of approach is known to be the only feasible way when the available network bandwidth is very limited [16] or when, due to privacy issues, the original signals can't be sent through a network infrastructure. The ATC paradigm also reduces the load at the sink node, thus reducing the cost of the node or allowing more sensor nodes to send data to a single sink node sharing a limited bandwidth communication channel.

Following the idea of the ATC approach, a sensor node could even operate independently from the sink node in all those situations in which, once the training phase of the system is completed at the sink node, a compact library of reference features is sent to each sensor node and is directly used to classify the future signals collected during sensor operations.

The present work shows one possible way to scale and optimize an existing Convolutional Neural Network for Face Recognition with the goal of execute it on a low power computing platform, taking care of reducing the execution time and the amount of data sent to the sink node, while keeping the highest possible accuracy on the task.

Chapter 1 recaps the State of the Art on Visual Sensor Networks and Convolutional Neural Networks for Face Recognition. The *PubFig83* reference dataset and the RaspberryPi platform used in this work are introduced.

Chapter 2 describes in details the method used to reduce and simplify an existing CNN model for facial feature extraction to suit the computational requirements of a low-power computing platform, while preserving the highest possible accuracy on the recognition task.

Chapter 3 shows the development of an ad-hoc coding architecture meant to significantly reduce the rate needed to send the extracted facial features to the sink node, always investigating the impact on task accuracy.

Chapter 4 compares the Analyze-Then-Compress and Compress-Then-Analyze paradigms with respect to the original and the simplified CNN models, showing that the ATC approach is highly competitive with the CTA approach, even in non rate-constrained network conditions.

Finally in Chapter 5 conclusions are drawn and some future works are proposed to continue the investigation on energy and rate efficients algorithms for face and object recognition.

Chapter 1

Review of the State of the Art

1.1 Visual Sensor Networks

A basic Visual Sensor Network consists of

- tiny, battery-operated, visual sensor nodes that integrate an image sensor, an embedded processor and a wireless transceiver
- powerful sink nodes, collecting and processing data from the sensor nodes

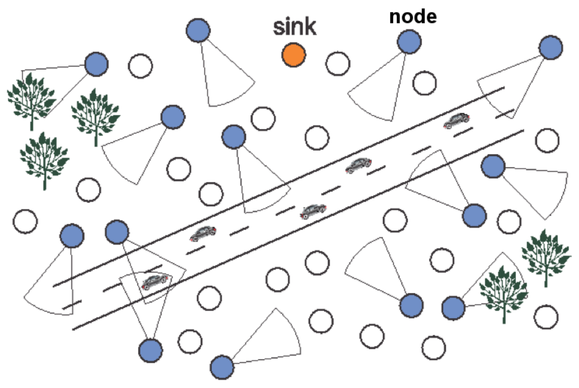


FIGURE 1.1: A Visual Sensor Network schema from [1]

Some applications of Visual Sensor Networks are [17]:

- public and commercial surveillance: VSNs may be used for monitoring public places such as parks, department stores, transport systems, and production sites for infrastructure malfunctioning, accident detection, and crime prevention
- environmental and building monitoring: VSNs are a solution for early detection of landslides, fires, or damages in mountain coasts, historical and archaeological sites
- military surveillance: VSNs can be employed in patrolling national borders, measuring flow of refugees, and assisting battlefield command and control
- smart homes and meeting rooms: VSNs can provide continuous monitoring of kindergarten, patients, or elderly requiring special care. This helps to measure the effectiveness of medical treatments and to detect earlier harmful situations.

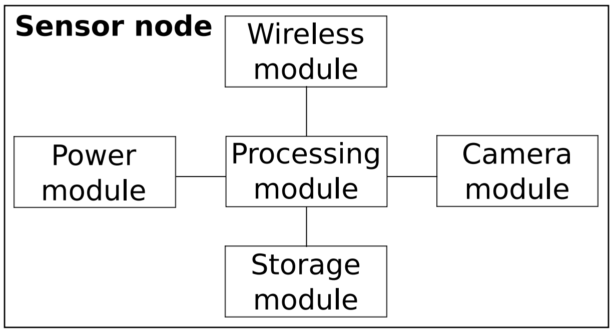


Figure 1.2: Sensor node schema

A sensor node in a VSN is typically composed of an image sensing module, a processing module, a storage module, a wireless communication module, and a power module (figure 1.2). The design of a node depends on the type of components and the way they are interconnected. Some off-the-shelf platforms offer all the system components in a single board, while modular systems are composed of a main board, containing the processing module, to which all the other modules (batteries, energy harvesting modules, digital cameras, wireless transceivers) are connected.

Digital camera modules are typically built with Complementary Metal–Oxide–Semiconductor (CMOS) imaging sensor. Although Charge-Coupled Device (CCD) components were the first used image capture technology, in the last decade CMOS sensors officially surpassed them as the overall image capture technology of choice, especially for challenging environments. This is mainly due to the low power consumption of CMOS (one tenth of

CCD), low cost, and easy integration of all camera functions on a single chip, significantly reducing chip count and board space. Moreover, CMOS sensors offer the same or better sensitivity compared to CCDs.

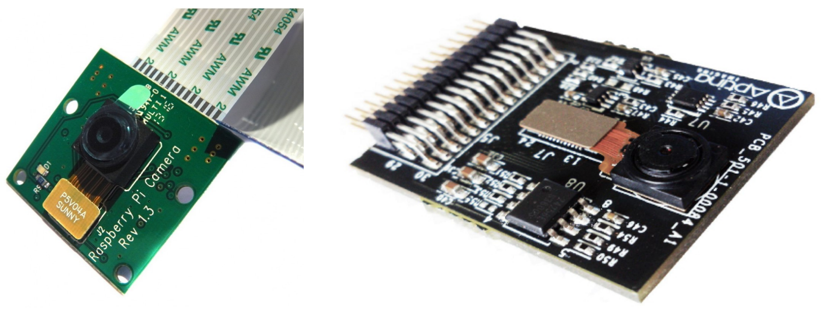


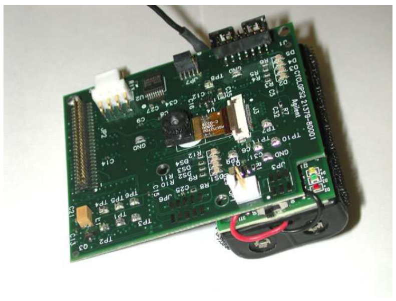
FIGURE 1.3: Camera modules for RaspberryPi and BeagleBone platforms

Processing modules are divided in three main categories. Some nodes are equipped with relatively powerful General Purpose Processor and large storage components (directly on the processing module or as separate storage modules) to achieve high performance. The drawback is the high energy dissipation, in the order of several Watts. Other nodes use lightweight microcontrollers (MCU) and specialized reconfigurable hardware components to address critical parts, as Complex Programmable Logic Devices (CPLD) or Field Programmable Gate Array (FPGA). The last type of nodes uses 32-bit low power processors (less than 1W) operating at relatively high frequencies.

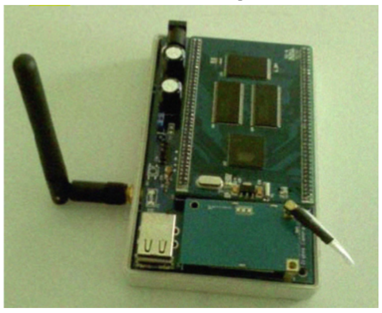
Popular wireless technologies available for different bandwidth requirements, distance range, price, network topology are, power consumption:

- Bluetooth - IEEE 802.15.1: up to 230.4 Kbps, low power consumption
- ZigBee or similar - IEEE 802.15.4: up to 250 Kbps, low power consumption
- WiFi - IEEE 802.11: up to 150Mbps depending on the standard, high power consumption

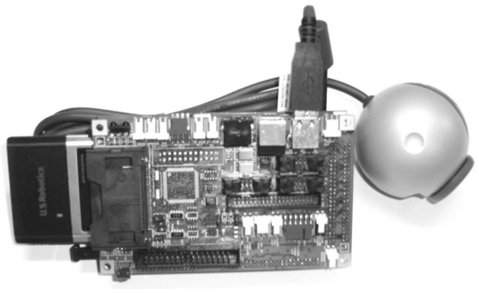
Energy modules are typically made of battery packs. Energy harvesting techniques are used to convert different forms of ambient energy (solar power, thermal energy, wind energy, kinetic energy) into electricity to power pervasive devices. However only photovoltaic technologies in good environmental conditions are able to give the necessary amount of power for visual sensor nodes to operate without battery draining.



Cyclops: Low power 8-bit ATMEL ATmega12 RISC MCU with TinyOS [18]



Vision Mote : Medium power 32-bit Atmel 9261 ARM 9 CPU with Linux OS [19]



Panoptes: High power Intel StrongARM 206 MHz with Linux OS [20]

Figure 1.4: Examples of sensor nodes for Visual Sensor Networks

From the software side, sensor nodes may run general purpose OS, application specific OS or no OS at all. Linux, together with its distributions, is a widespread general purpose OS for VSNs nodes. It provides flexibility to modify its components to fit the application needs. Programming and prototyping is easy at a greater expense of energy due to OS overhead compared to application specifics OS. TinyOS and Nano-RK are two examples of VSN specific OS, requiring minimal hardware to run. Some sensors could even have no OS at all, with a finite state machine firmware that performs resource management. This kind of approach has great performance but requires a longer development time.

1.1.1 RaspberryPi

The reference sensor node chosen for this work is a a Raspberry Pi model B.

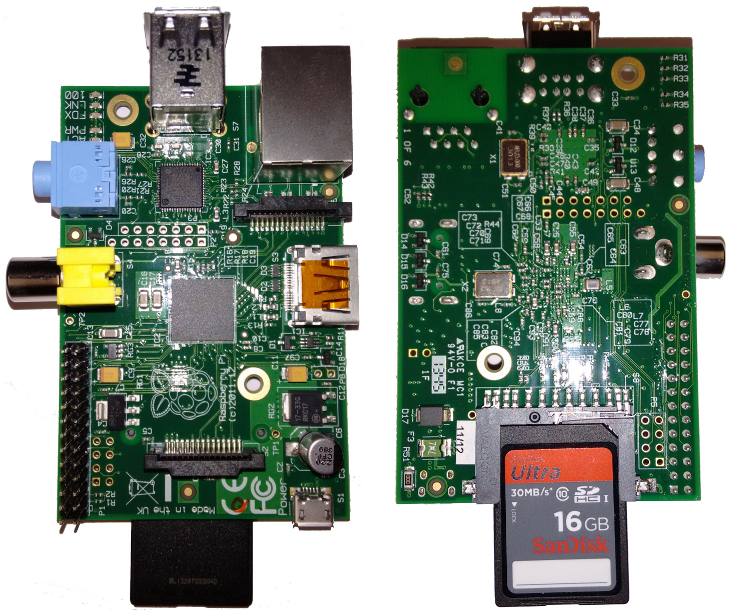


FIGURE 1.5: RaspberryPi Model B

The Raspberry Pi is a single board computer based on a BCM2835 SoC [21]. The board is equipped with:

- BCM2835 SoC
- Full size SD card slot
- HDMI output port
- Composite video out
- Two USB type A ports
- 26 pin expansion header exposing GPIO, I2C, SPI, UART peripherals
- 3.5mm stereo audio output jack
- Camera interface port (CSI-2)
- LCD display interface port (DSI)

- microUSB type B connector (power only)
- Ethernet port 10/100 Mbit/s

The BCM2835 is configured with:

- ARMv6 architecture ARM1176JZF-S processor [22] working at 700MHz
- Vectorized Floating Point Unit v2 (VFPv2)
- Dual Core VideoCore IV©Multimedia Co-Processor
- Low power, high performance OpenGL-ES©1.1/2.0 VideoCore GPU
- 512Mb of SDRAM
- UART, I2C and SPI/I2S peripherals
- DMA controller
- General Purpose I/O
- USB 2.0 host controller

The running OS is a non real time Linux Debian version optimized for RaspberryPi, named Raspbian[1], with kernel version 3.12.28+.

Python interpreter version 2.7.3 is used with the following modules:

- numpy 1.9.0 scientific computing package for Python, linked to ATLAS 3.8.4 linear algebra libraries[2]
- scipy 0.14.0 mathematics, science, and engineering package for Python[3]
- numexpr 2.4 fast numerical array expression evaluator for Python[4]
- skimage 0.10.1 image processing package for Python[5]

[1]http://www.raspbian.org
[2]http://www.numpy.org/
[3]http://www.scipy.org/
[4]https://github.com/pydata/numexpr
[5]http://scikit-image.org/

- PIL 1.1.7 Python Imaging Library from the Pillow fork[6]

RaspberryPi Model B is natively equipped with an Ethernet port for network communications, but no wireless module nor camera are available. An ad-hoc HD camera module[7] is available from RaspberryPi Foundation, but also generic USB webcam can be used, paying attention to driver availability and compatibility for the kernel in use.

Some examples of wireless devices, for different radio protocols, known to work with Raspbian are:

- IEEE 802.11b/g/n - Edimax EW7811-Un[8] USB adapter
- IEEE 802.15.1 - Hama Bluetooth 3.0+EDR[9] USB adapter
- IEEE 802.15.4 - Dresden Elektronik RaspBee[10] UART adapter

FIGURE 1.6: Wireless USB and UART adapters for RaspberryPi

As in the camera case, care must be taken to select a device whose drivers are available and stable for the kernel version in use.

1.2 Convolutional Neural Networks for Face Recognition

Cox and Pinto in [23] describe a large-scale feature search approach to the problem of unconstrained face recognition, based on the Labeled Faces in the Wild challenge set [8]. The best performing CNN model (HT-L3-1st) is composed of an input layer ($Layer^0$) followed by three structurally identical layers ($Layer^1$, $Layer^2$, $Layer^3$).

[6]http://pillow.readthedocs.org/
[7]http://www.raspberrypi.org/products/camera-module/
[8]http://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/global/wireless_adapters_n150/ew-7811un
[9]https://de.hama.com/00049237/hama-nano-bluetooth-usb-adapter-version-30+edrclass2_eng
[10]http://www.dresden-elektronik.de/funktechnik/solutions/wireless-light-control/raspbee/

Layer^0 takes as input an RGB image, converts it into a greyscale image, then locally normalizes the image to get $\boldsymbol{N^0}$.

$$Layer^0 : \textbf{Input RGB image} \xrightarrow{\text{Grayscale}} \xrightarrow{\text{Local normalization}} \boldsymbol{N}^0$$

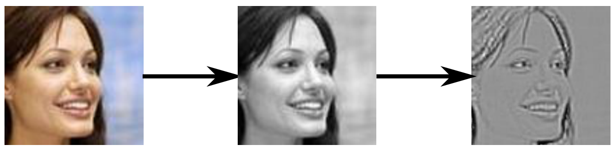


Figure 1.7: CNN Layer 0: image transformation

The general structure for *Layer*^{l} with $l \geq 1$ is

$$Layer^l : \boldsymbol{N}^{l-1} \xrightarrow{\text{Filter-bank correlation}} \boldsymbol{F}^l \xrightarrow{\text{Activation}} \boldsymbol{A}^l \xrightarrow{\text{Local pooling}} \boldsymbol{P}^l \xrightarrow{\text{Local normalization}} \boldsymbol{N}^l$$

where:

- Filter-bank correlation locally transforms the input maps to generate a greater number of output maps
- Activation acts in a non-linear fashion on the single coefficients
- Pooling performs a map-by-map local smoothing followed by downsampling
- Normalization works along all the maps to locally normalize the coefficients

A qualitative result of the feature maps transformation trough the three layers is shown in figure 1.8.

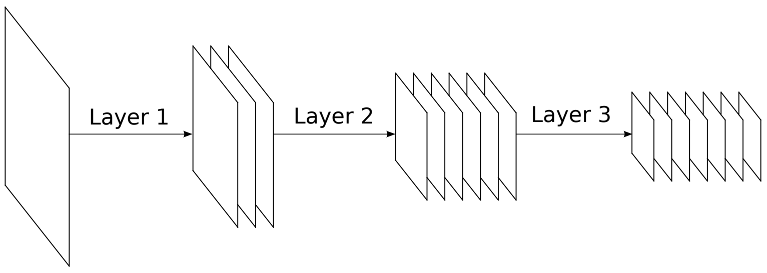


Figure 1.8: CNN: feature maps transformation

The Python implementation of the CNN by Pinto et al. available at [24] is used as reference in this work.

1.2.1 Filter-bank correlation

The input $\boldsymbol{N}^{l-1}$ for $Layer^l, l \geq 1$ is linearly filtered using a bank of k^l filters to produce a stack of k^l feature maps, called $\boldsymbol{F}^l$. Each filter Φ_i^l has shape $f_s^l \times f_s^l \times f_d^l$, where $f_s^l \in \{3, 5, 7, 9\}$ is the filter neighborhood and

$$f_d^l = \begin{cases} k^{l-1} & l > 1 \\ 1 & l = 1 \end{cases}$$

is the stack dimension of the input layer feature maps. The filtering operation for $Layer^l$ is denoted

$$\boldsymbol{F}^l = \textbf{Filter}\left(\boldsymbol{N}^{l-1}, \boldsymbol{\Phi}^l\right)$$

where $\boldsymbol{\Phi}^l$ is the $f_s^l \times f_s^l \times f_d^l \times k^l$ filterbank and each filter, denoted as $\Phi_i^l \ \forall i \in \{1, 2, \ldots, k^l\}$, has shape $f_s^l \times f_s^l \times f_d^l$. Each output map F_i^l is the result of a three dimensional correlation between N^{l-1} and Φ_i^l sliding along the first and second dimensions of N^{l-1}

$$F_i^l = N^{l-1} \otimes \Phi_i^l \ \forall i \in \{1, 2, \ldots, k^l\}$$

For an input feature map $\boldsymbol{N}^{l-1}$ of shape $B \times B \times f_d^l$ the output feature map $\boldsymbol{F}^l$ has shape $B - (f_s^l - 1) \times B - (f_s^l - 1) \times k^l$.

The number of output maps for the filter k^l is chosen in a different pool for each layer of the network:

- $k^1 \in \{16, 32, 64\}$
- $k^2 \in \{16, 32, 64, 128\}$
- $k^3 \in \{16, 32, 64, 128, 256\}$

A peculiar characteristic of this network is that the filter weights are not trained with feedforward and back-propagation, as usually done in neural networks, but instead they are randomly drawn from a uniform distribution between -1 and 1. For each filter F_i^l the mean of the random coefficients is removed and the filter is normalized with respect to the L_2 norm of the unbiased coefficients.

1.2.2 Activation

The output of the filtering stage $\boldsymbol{F}^l$ is subject to non-linear threshold and saturation.

$$\boldsymbol{A}^l = \textbf{Activate}\left(\boldsymbol{F}^l\right)$$

The activation function is defined as

$$\textbf{Activate}(x) = \begin{cases} \gamma_{\max}^l & \text{if } x > \gamma_{\max}^l \\ \gamma_{\min}^l & \text{if } x < \gamma_{\min}^l \\ x & \text{otherwise} \end{cases}$$

where $\gamma_{\min}^l \in \{-\infty, 0\}$ is the threshold level and $\gamma_{\max}^l \in \{1, +\infty\}$ is the saturation level. The activation function acts element-wise, thus the shape of $\boldsymbol{A}^l$ is the same of $\boldsymbol{F}^l$.

1.2.3 Local pooling

The output of the activation stage $\boldsymbol{A}^l$ is subject to a two step pooling operation:

- A map-wise smoothing operation of order $p^l \in \{1, 2, 10\}$ within a neighborhood of shape $a^l \times a^l$ ($a^l \in \{3, 5, 7, 9\}$)
- A downsampling operation with fixed stride $\alpha = 2$ resulting in a downsamplig factor of 4

Each map $\boldsymbol{P}_i^l$ $\forall i \in \{1, 2, \ldots, k^l\}$ is defined as:

$$\boldsymbol{P}_i^l = \textbf{Downsample}_\alpha\left(\sqrt[p^l]{\left(A_i^l\right)^{p^l} \odot \mathbf{1}_{a^l \times a^l}}\right)$$

where

- A_i^l is a map of shape $B \times B$ from the activation stage output $\boldsymbol{A}^l$ of shape $B \times B \times k^l$.
- $\odot$ is the 2-dimensional correlation function between $\left(A_i^l\right)^{p^l}$ and $\mathbf{1}_{a^l \times a^l}$
- $\mathbf{1}_{a^l \times a^l}$ is a $a^l \times a^l$ matrix of ones.

For $p^l = 1$ the result of the smoothing operation is equivalent to blurring with a box-car filter, while with $p^l = 10$ the smoothing operation is similar to a *max* operation (*softmax*).

For an input $\boldsymbol{A}^l$ of shape $B \times B \times k^l$ the output of the local pooling operation $\boldsymbol{P}^l$ has shape $\lfloor \left(B - (a^l - 1)\right)/2 \rfloor \times \lfloor \left(B - (a^l - 1)\right)/2 \rfloor \times k^l$.

1.2.4 Local normalization

The final processing stage of each layer is a local normalization operation across space and maps. The local normalization function is defined as

$$\boldsymbol{N}^l = \textbf{Normalize}\left(\boldsymbol{P}^l\right) = \begin{cases} \rho^l \cdot \boldsymbol{P}^l & \text{if } \rho^l \cdot \left|\left|\boldsymbol{P}^l \otimes \mathbf{1}_{b^l \times b^l \times k^l}\right|\right|_2 < \tau^l \\ \frac{\boldsymbol{P}^l}{\left|\left|\boldsymbol{P}^l \otimes \mathbf{1}_{b^l \times b^l \times k^l}\right|\right|_2} & \text{otherwise} \end{cases}$$

where

- $\rho^l \in \{10^{-1}, 10^0, 10^1\}$ is a stretching value
- $\tau^l \in \{10^{-1}, 10^0, 10^1\}$ is a threshold value
- $b^l \in \{3, 5, 7, 9\}$ is the neighborhood size
- $||x||_2$ is the L_2 norm of x
- $\mathbf{1}_{b^l \times b^l \times k^l}$ is a $b^l \times b^l \times k^l$ array of ones
- $\otimes$ is a 3-dimensional correlation operation sliding across the first two dimensions of $\boldsymbol{P}^l$

For an input $\boldsymbol{P}^l$ of shape $B \times B \times k^l$ the output of the local normalization operation $\boldsymbol{N}^l$ has shape $B - (b^l - 1) \times B - (b^l - 1) \times k^l$.

Note that for the input $Layer^0$, k^l is equal to 1.

1.3 The PubFig83 dataset

The reference dataset used in this work is the *PubFig83* aligned dataset, created by Pinto et al. in [12]. The *PubFig83* dataset is a subset of the *PubFig* dataset defined in [25], a collection of 60000 publicly available photos of 200 celebrities (300 photos per individual on average). Both the development and the evaluation *PubFig* sets have been processed with the OpenCV face detector to prune the faces not successfully detected. The remaining images contained many duplicates of the same original photos, with different type of compression, color spaces, sizes, digitally edited background. To remove those duplicates, a correlation based score on the face central region has been used to rank couple of images for each subject. The 4% most correlated couples among all the subjects were marked as duplicates and one of the images has been removed. At the end of this process only 83 subjects with more than 100 non duplicated photos were preserved to build the *PubFig83* dataset. The final result is a dataset containing 13838 face images for 83 subjects. Each subject counts from a minimum of 100 to a maximum of 300 color images at 100 x 100 pixels resolution. The aligned version of the *PubFig83* is an eye-aligned version of the original *PubFig83* dataset. Some samples from the *PubFig83* aligned dataset are shown in figure 1.9

1.3.1 Accuracy evaluation protocol

The evaluation protocol proposed for the *PubFig83* dataset is a 10-fold cross validation method, with 90 training samples and 10 testing samples per category.

For each subject 10 splits of 100 images are randomly generated from the available pictures. Each split is further randomly divided in a training subset of 90 pictures and

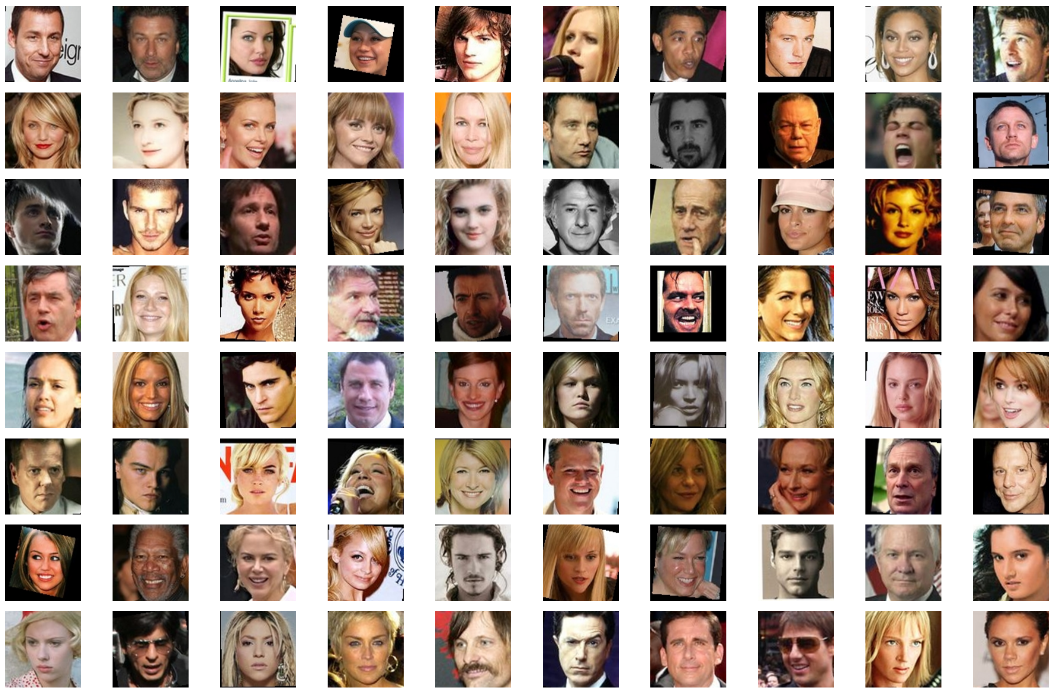


FIGURE 1.9: *PubFig83* aligned sample images

a testing subset of 10 pictures. For the *PubFig83* dataset the resulting training and test sets are composed respectively of $83 \cdot 90 = 7470$ and $83 \cdot 10 = 830$ images.

The classifier used to evaluate the accuracy is a linear Support Vector Machine with regularization parameter fixed to 10^5 and precomputed train and test kernel.

Given a train set M_{train} of shape $N_{\text{train}} \times P$, where N_{train} is the number of training samples and P is the number of features, and a test set M_{test} of shape $N_{\text{test}} \times P$ the precomputed train and test kernels are $M_{train} \cdot M_{train}^T$ and $M_{test} \cdot M_{train}^T$, respectively of shapes $N_{\text{train}} \times N_{\text{train}}$ and $N_{\text{test}} \times N_{\text{train}}$. Both the kernels are divided by the trace of the train kernel.

To support the multi-class problem of face recognition the one-versus-all approach is used to train a different SVM for each category, using the precomputed training kernel. The training samples corresponding to the considered category are labeled as +1, all the others are labeled as -1. The SVM is then trained using the precomputed train kernel and the separating hyperplane is determined. The distances of the test samples from the separating hyperplane are calculated on the precomputed test kernel trough the decision function.

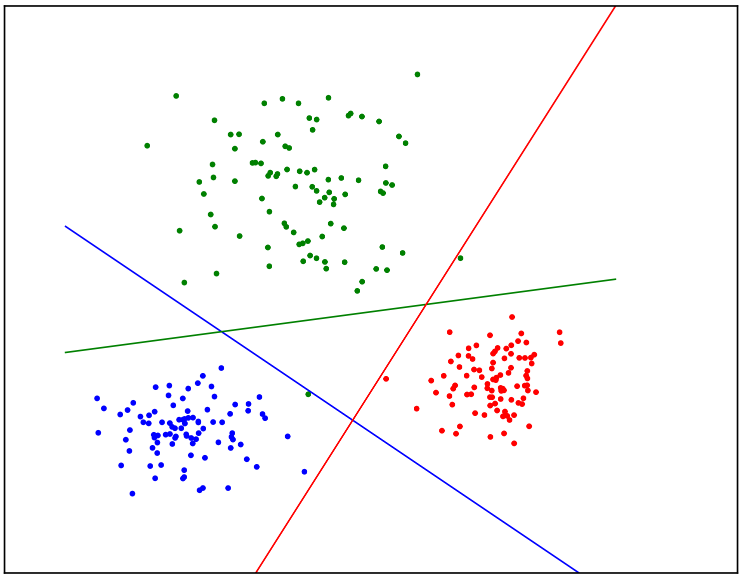


FIGURE 1.10: SVM one-vs-all example: Each line represents the separating hyperplane generated by a binary linear SVM trained with the samples of the same color of the line labeled +1 and all the others labeled -1

Every SVM, one for each category, gives a distance measure for each test sample. To each samples is then assigned the category corresponding to the highest output SVM.

The accuracy of a single split is calculated as the percentage of correctly classified test samples. The global accuracy is the mean accuracies over the 10 splits .

Chapter 2

Time-accuracy optimization

The best performance 3-layer CNN model (HT-L3-1st) presented in [23] is the starting point for a model search aimed at finding a good compromise between processing time on the RaspberryPi and accuracy on the *PubFig83* dataset. Since the processing time is proportional to the amount of energy required by the algorithm, the following work can be thought as an energy-accuracy optimization process.

To comply with the naming scheme of the reference Python implementation available at [24] the 3-layer CNN model HT-L3-1st is called *fg11-ht-l3-1*[1]. Its parameters are shown in table 2.1.

		Layer 0	**Layer 1**	**Layer 2**	**Layer 3**
fbcorr	shape	-	3 x 3	5 x 5	5 x 5
	filt num	-	64	128	256
threshold	th min	-	0	0	0
	th max	-	∞	∞	∞
lpool	shape	-	7 x 7	5 x 5	7 x 7
	order	-	1	1	10
	stride	-	2	2	2
lnorm	shape	9 x 9	5 x 5	7 x 7	3 x 3
	stretch	10.0	0.1	1.0	10.0
	threshold	1.0	1.0	1.0	1.0

TABLE 2.1: *fg11-ht-l3-1* model: parameters

[1]*fg11-ht*: model class; *l3*: three layers; *1*: top-1 model

The *fg11-ht-l3-1* model needs input images of 200 x 200 pixels, thus requiring an initial image upsampling before using the CNN to extract the features.

The original implementation of the neural network has been modified to allow the timing of network layers and operations. The chosen timing method is based on system time and thus is influenced by other processes running on the platform. However, since the RaspberryPi has no scheduled cronjobs nor active services, this timing system is reliable when compared to the dedicated Python timing modules (Profile, cProfile) and it has a very small overhead.

A smaller version of the *PubFig83* aligned dataset has been created selecting one image from each category. This smaller dataset is used to average the timing results on the RaspberryPi while keeping a reasonably fast execution of the time profiling procedures.

2.1 Three-Layers CNN performance and simplification

The execution time of the *fg11-ht-l3-1* model is profiled using the time-enabled neural network implementation. The results are shown in table 2.2.

	Time [s]	**Time %**
Layer 0	00.06	00.34
lnorm	00.06	00.34
Layer 1	03.76	21.00
fbcorr	00.69	03.84
lpool	02.90	16.22
lnorm	00.16	00.90
Layer 2	09.42	52.69
fbcorr	**08.44**	**47.23**
lpool	00.93	05.22
lnorm	00.04	00.22
Layer 3	04.64	25.96
fbcorr	**04.20**	**23.51**
lpool	00.42	02.36
lnorm	00.01	00.08
Tot	***17.88***	***100.00***

Table 2.2: *fg11-ht-l3-1* model: time profiling

The total time needed for the *fg11-ht-l3-1* CNN execution is 17.88s. The two most demanding operations are the filter-bank correlations at 2nd and 3rd layers. In order to reduce the huge amount time needed to compute the features, the 3rd layer of the network is removed and the initial image upsampling is avoided. The newly obtained model is called *fg11-ht-l2-s1*[2]

2.2 Stage-wise Dual-Layers CNN optimization

The accuracy of the *fg11-ht-l2-s1* model on the *PubFig83* dataset is 80.64%. The execution time of the model is profiled and the result is shown in table 2.3.

	Time [s]	**Time %**
Layer 0	00.02	00.57
lnorm	00.02	00.57
Layer 1	00.83	23.86
fbcorr	00.16	04.58
lpool	**00.63**	**18.21**
lnorm	00.04	01.01
Layer 2	02.62	75.55
fbcorr	**02.46**	**70.90**
lpool	00.15	04.34
lnorm	00.01	00.27
Tot	***03.47***	***100.00***

Table 2.3: *fg11-ht-l2-s1* model: time profiling

The total time needed for the *fg11-ht-l2-s1* CNN execution is 3.47s. The two most time demanding operation are the filter-bank correlation in the second layer and the local pooling in the first layer. Following the idea of brute-force model search presented in [12], the CNN parameters are modified to exploit the effect of the network structure modifications on the time-accuracy plane. Since the parameters space it's too big to be completely explored, a handcrafted 5-step search is performed in the following sections. The performance of the three layer network model are kept as reference benchmark for accuracy and timing during the whole model search process.

[2]two layers; first model search

2.2.1 Model search: First step

Starting from the *fg11-ht-l2-s1* model a first model search is performed varying:

- The number of filters in the first layer (k^1): $\{16, 32, 64\}$
- The number of filters in the second layer (k^2): $\{16, 32, 64, 128\}$

Each model is evaluated both in its execution time, on a RaspberryPi, and in its accuracy, on a test PC, following the *PubFig83* evaluation protocol. The 12 resulting models time-accuracy performance are shown in figure 2.1.

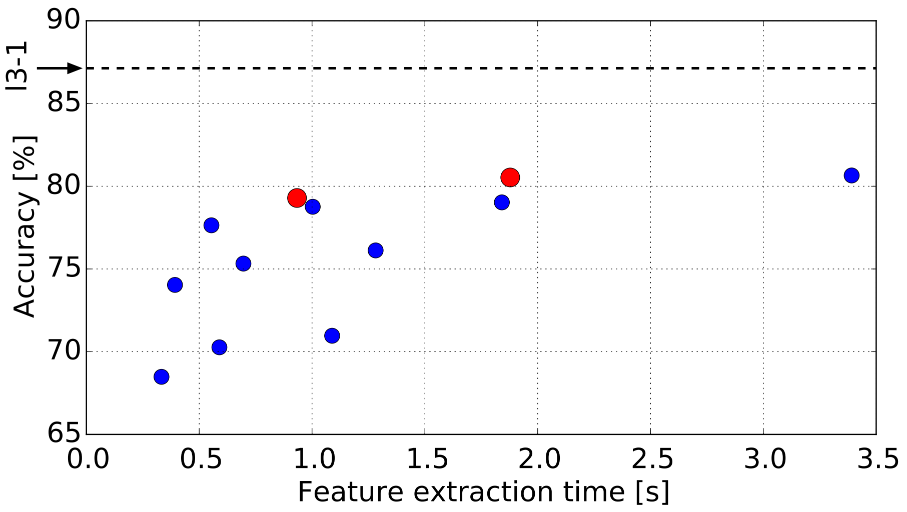


Figure 2.1: First model search: results

In order to continue the model search, the two red-dots models are chosen as good compromise between accuracy and time. Their parameters and time-accuracy performance are shown in table 2.4.

k^1	k^2	Time [s]	Accuracy [%]
16	128	0.93	79.28
32	128	1.88	80.53

Table 2.4: First model search: selected models

Reducing the number of filters and removing the third layer reduces significantly the computation time, at the cost of losing at least 10% points in accuracy.

2.2.2 Model search: Second step

Starting from the results of the first model search, a second model search is performed. The search-base model *fg11-ht-l2-s2* is the same as the first search-base model (*fg11-ht-l2-s1*) but with 16 filters in the first layer, as shown in table 2.5.

		Layer 0	**Layer 1**	**Layer 2**
fbcorr	shape	-	3 x 3	5 x 5
	filt num	-	16	128
threshold	th min	-	0	0
	th max	-	∞	∞
lpool	shape	-	7 x 7	5 x 5
	order	-	1	1
	stride	-	2	2
lnorm	shape	9 x 9	5 x 5	7 x 7
	stretch	10.0	0.1	1.0
	threshold	1.0	1.0	1.0

Table 2.5: *fg11-ht-l2-s2* model: parameters

The second model search is performed varying:

- The number of filters in the first layer (k^1): $\{16, 32\}$
- The normalization stretching in all layers ($\rho^l, l \in \{0, 1, 2\}$): $\{0.1, 1, 10\}$

The resulting 54 models time-accuracy performance are shown in figure 2.2.

The two clusters of points clearly show the difference between the models using 16 filters in the first level (leftmost) and the ones using 32 filters in the first level (rightmost). The normalization stretching doesn't influence too much the timing but it strongly influences the accuracy. Following these considerations, the two red-dots models are used as search-base models for the third (*fg11-ht-l2-s3*) and the fourth (*fg11-ht-l2-s4*) steps of the model search. Their time-accuracy performance are shown in table 2.6.

New model	k^1	ρ^0	ρ^1	ρ^2	**Time** [s]	**Accuracy** [%]
fg11-ht-l2-s3	16	10	10	10	0.95	81.67
fg11-ht-l2-s4	32	10	10	1	1.87	83.12

Table 2.6: Second model search: selected models

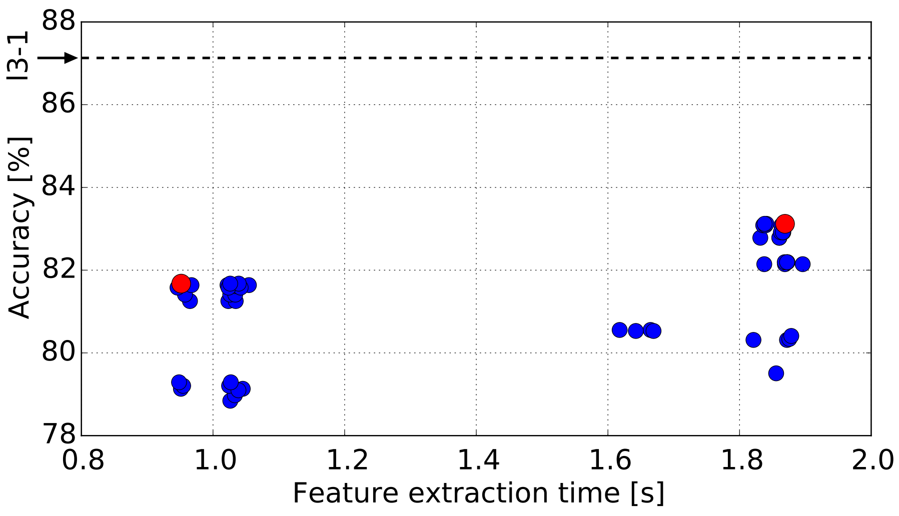


FIGURE 2.2: Second model search: results

2.2.3 Model search: Third step

The parameters of the third step search-base model *fg11-ht-l2-s3* are shown in table 2.7.

		Layer 0	**Layer 1**	**Layer 2**
fbcorr	shape	-	3 x 3	5 x 5
	filt num	-	16	128
threshold	th min	-	0	0
	th max	-	∞	∞
lpool	shape	-	7 x 7	5 x 5
	order	-	1	1
	stride	-	2	2
lnorm	shape	9 x 9	5 x 5	7 x 7
	stretch	10.0	10.0	1.0
	threshold	1.0	1.0	1.0

TABLE 2.7: *fg11-ht-l2-s3* model: parameters

The third model search is performed varying:

- The pooling neighborhood size in all layers ($a^l, l \in \{1, 2\}$): $\{3, 5, 7, 9\}$
- The pooling exponent in all layers ($p^l, l \in \{1, 2\}$): $\{1, 2, 10\}$

The resulting 144 models time-accuracy performance are shown in figure 2.3.

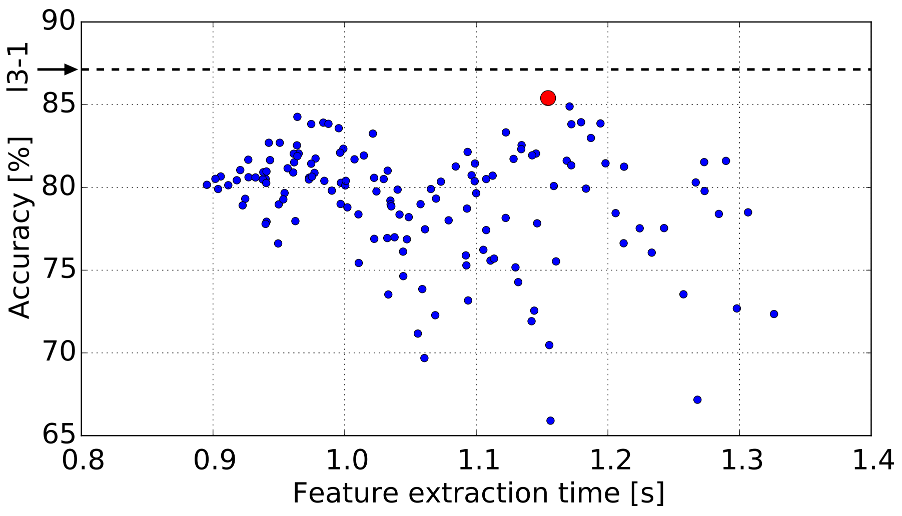


FIGURE 2.3: Third model search: results

Since all the models have 16 filters in the first layer, the average execution time is around 1s. The time difference is mostly due to the varying neighborhood size of the pooling operations. A larger neighborhood causes a greater cropping of the image and thus a reduction of features first and second dimensions in the successive operations.

The red-dot model is selected as search-base model for the fifth step of the model search (*fg1-ht-l2-s5*). Its time-accuracy performance are shown in table 2.8.

New model	a^1	p^1	a^2	p^2	**Time** [s]	**Accuracy** [%]
fg11-ht-l2-s5	5	1	5	10	1.15	85.40

TABLE 2.8: Third model search: selected model

2.2.4 Model search: Fourth step

The parameters of the fourth step search-base model *fg11-ht-l2-s4* are shown in table 2.9.

The fourth model search is performed varying the same parameters as in the third step:

- The pooling neighborhood in all layers ($a^l, l \in \{1, 2\}$): $\{3, 5, 7, 9\}$
- The pooling exponent in all layers ($p^l, l \in \{1, 2\}$): $\{1, 2, 10\}$

		Layer 0	**Layer 1**	**Layer 2**
fbcorr	shape	-	3 x 3	5 x 5
	filt num	-	32	128
threshold	th min	-	0	0
	th max	-	∞	∞
lpool	shape	-	7 x 7	5 x 5
	order	-	1	1
	stride	-	2	2
lnorm	shape	9 x 9	5 x 5	7 x 7
	stretch	10.0	10.0	1.0
	threshold	1.0	1.0	1.0

TABLE 2.9: *fg11-ht-l2-s4* model: parameters

The resulting 144 models time-accuracy performance are shown in figure 2.4.

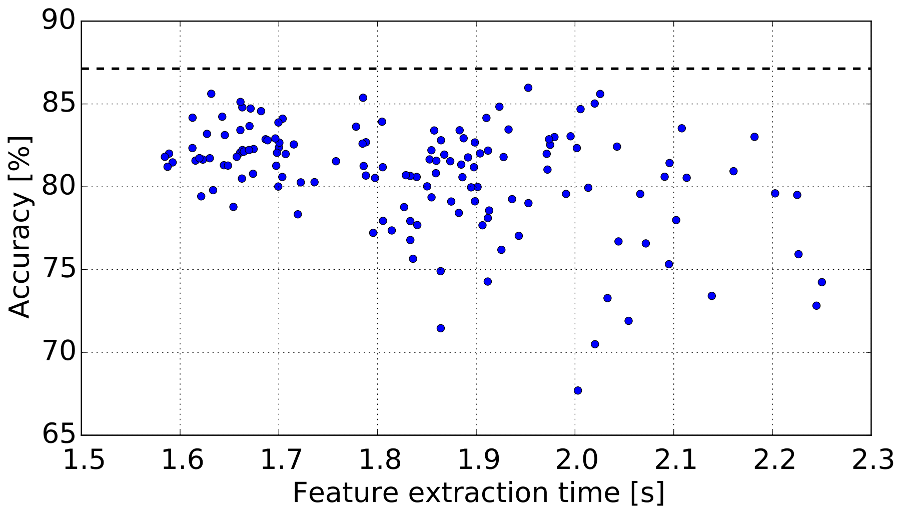


FIGURE 2.4: Fourth model search: results

Comparing these results to the ones of the third search step, the average execution time is around 1.9s due to the 32 filters in the first layer of the network but the maximum accuracy is 85.97%, greater than the maximum accuracy of the third search step.

2.2.5 Model search: Fifth step

The parameters of the fifth step search-base model *fg11-ht-l2-s5* are shown in table 2.10.

		Layer 0	**Layer 1**	**Layer 2**
fbcorr	shape	-	3 x 3	5 x 5
	filt num	-	16	128
threshold	th min	-	0	0
	th max	-	∞	∞
lpool	shape	-	5 x 5	5 x 5
	order	-	1	10
	stride	-	2	2
lnorm	shape	9 x 9	5 x 5	7 x 7
	stretch	10.0	10.0	1.0
	threshold	1.0	1.0	1.0

Table 2.10: *fg11-ht-l2-s5* model: parameters

The fifth model search is performed varying:

- The normalization neighborhood in all layers ($b^l, l \in \{0, 1, 2\}$): $\{3, 5, 7, 9\}$

The resulting 64 models time-accuracy performance are shown in figure 2.5.

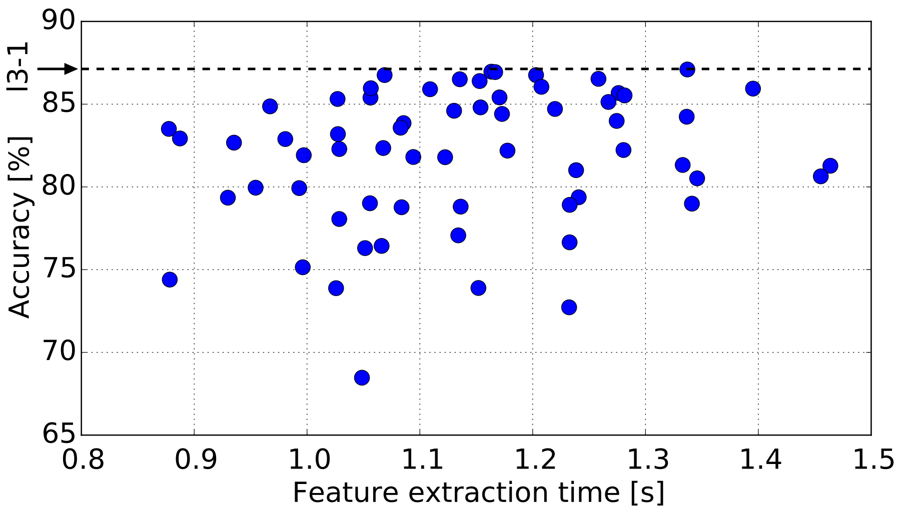


Figure 2.5: Fifth model search: results

All the models evaluated in this step have 16 filters in the first layer and the average execution time is around 1.15s. The maximum accuracy however is really near to the benchmark one (*fg11-ht-l3-1*), with an execution time up to 17 times smaller.

2.3 Dual-Layers CNN optimal model

A collection of 413 network models, shown in figure 2.6, is obtained putting together all the model search steps and removing the duplicates.

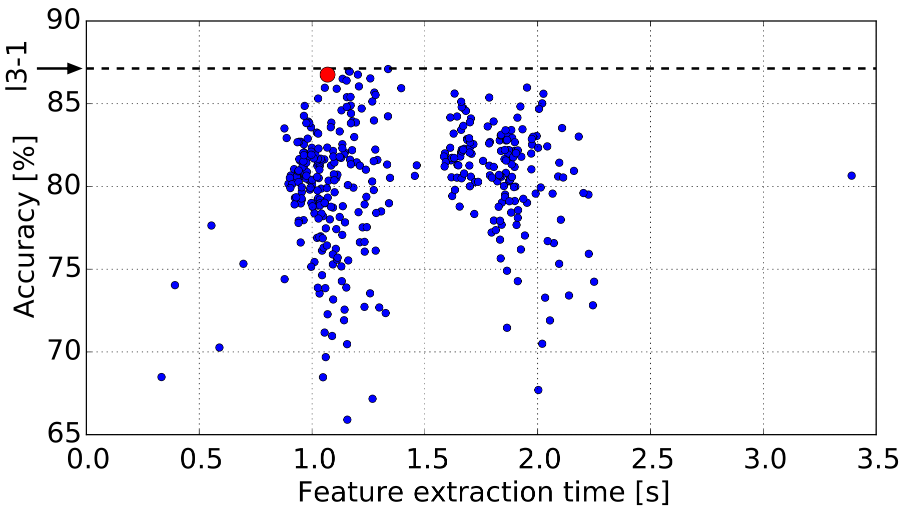


Figure 2.6: Dual layer model search results

It is still possible to separate by hand the models having 16 filters in the first layer (time smaller than 1.5s) from the ones having 32 filters in the first layer (time higher then 1.5s), even if some outliers from the first model search are present.

The red-dot model is selected as pseudo-optimal dual layer CNN model and is named *fg11-ht-l2-opt*. Its parameters are shown in table 2.11 and its accuracy reaches 86.73%.

		Layer 0	**Layer 1**	**Layer 2**
fbcorr	shape	-	3 x 3	5 x 5
	filt num	-	16	128
threshold	th min	-	0	0
	th max	-	∞	∞
lpool	shape	-	5 x 5	5 x 5
	order	-	1	10
	stride	-	2	2
lnorm	shape	9 x 9	5 x 5	3 x 3
	stretch	10	10	1
	threshold	1	1	1

Table 2.11: *fg11-ht-l2-opt* model: parameters

Notice that the stepwise model search described above is not complete in the space of parameters, and thus the pseudo-optimal model could be suboptimal. However the whole process shows how an initially slow yet well performing neural network model (*fg11-ht-l3-1*) could be reduced and optimized for a specific task in a few number of steps.

The time profiling of the *fg11-ht-l2-opt* model in table 2.12 shows that even if the most time demanding operation is again the filter-bank correlation in the second layer, the impact on the total time is only the 53%, with respect to the 71% of the first dual-layer model (*fg11-ht-l2-s1*).

	Time [s]	**Time %**
Layer 0	00.02	01.86
lnorm	00.02	01.85
Layer 1	00.20	18.61
fbcorr	00.05	04.29
lpool	00.13	12.66
lnorm	00.02	01.58
Layer 2	00.84	79.49
fbcorr	**00.56**	**52.96**
lpool	00.27	25.33
lnorm	00.01	01.09
Tot	***01.06***	***100.00***

TABLE 2.12: Dual layer pseudo-optimal model: time profiling

With respect to the original three layer network (*fg11-ht-l3-1*) the loss in accuracy of the pseudo-optimal dual layer network (*fg11-ht-l2-opt*) is only 0.4% with a 94% time reduction.

Chapter 3

Rate-accuracy tradeoff

The pseudo-optimal dual layer network model (*fg11-ht-l2-opt*) generates 14 x 14 x 128 float32 features as output, for a total of 800kBit/image. This value is more than 10 times the size of a 100 x 100 grayscale JPEG image (roughly 62kBit/image) and thus is not optimal for an efficient wireless network transmission.

The goal of this Chapter is to understand the relationship between features size (rate) and accuracy when features are lossy coded. The development of an ad-hoc coding architecture is made in two phases: a Principal Component Analysis based dimensionality reduction (sec. 3.1) followed by features quantization and coding (sec. 3.2).

The output features are reshaped in order to have an N x P matrix, where N is the number of images and P is the number of features (25088 for the *fg11-ht-l2-opt* CNN model).

3.1 PCA based dimensionality reduction

3.1.1 PCA dimensionality reduction through SVD

The training set of each split is used to determine the bias for each feature. The bias is removed from both the training and the test set. Singular Value Decomposition

(SVD) of the unbiased training set (M_{train}) is calculated ($U\Sigma V^T = M_{train}$) and the first K columns of U are multiplied by the first K columns of Σ to get the transformed training set $M_{train}^{PCA(K)}$. The unbiased test set of the same split (M_{test}) is transformed multiplying it with the first K rows of the rotation matrix V, previously calculated with SVD: $M_{test}^{PCA(K)} = M_{test} \cdot V_K^T$. The newly obtained $M_{train}^{PCA(K)} \in \mathbb{R}^{N_{train} \cdot K}$ and $M_{test}^{PCA(K)} \in \mathbb{R}^{N_{test} \cdot K}$ are used to train the SVMs and evaluate the accuracy of the split according to the *PubFig83* protocol.

3.1.2 Dimensionality reduction

Figure 3.1 shows the impact of PCA with $K = \{200, 400, 800, 1600, 3200, 6400\}$ in the rate-accuracy plane. The PCA is trained in each split independently as described above. The rate is calculated as $K \cdot 32$ bit/image.

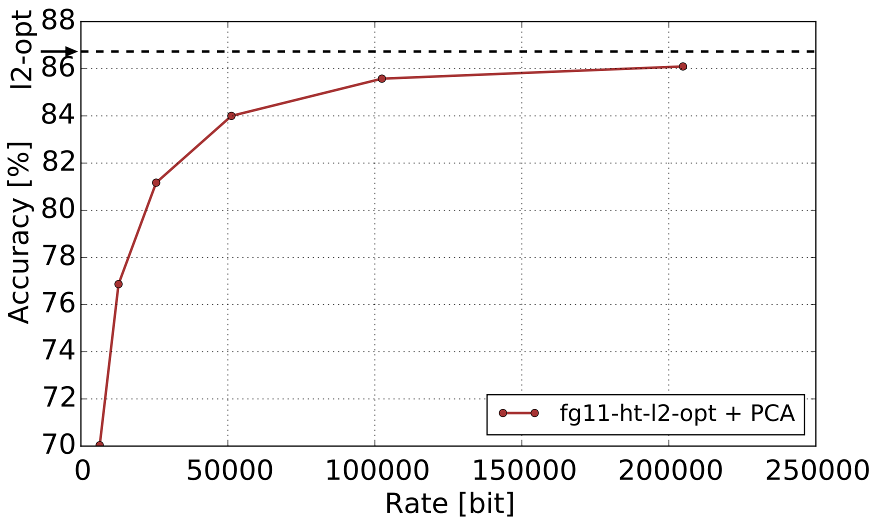


FIGURE 3.1: PCA based dimensionality reduction: impact on the rate-accuracy plane

Table 3.1 shows the comparison, in terms of rate reduction and accuracy loss between, the PCA based dimensionality reduction and the *fg11-ht-l2-opt* features extraction without dimensionality reduction (rate = 802816 bit, accuracy = 86.73%).

PCA based dimensionality reduction can be used to effectively reduce the storage space or the amount of data sent through the network while keeping a high level of accuracy.

K	Rate [bit]	Rate reduction [%]	Accuracy [%]	Accuracy loss [%]
200	6400	99.20	70.04	16.69
400	12800	98.41	76.87	9.86
800	25600	96.81	81.17	5.56
1600	51200	93.62	84.00	2.73
3200	102400	87.24	85.58	1.15
6400	204800	74.49	86.10	0.63

Table 3.1: Dual layer pseudo-optimal model: the effect of PCA based dimensionality reduction on rate and accuracy

3.2 Features quantization and coding

After PCA based dimensionality reduction, a fixed step (Q) mid-thread uniform quantizer is used to quantize the features and to reduce their entropy. The number of levels (L) of the quantizer is determined in each split by the range of the training set features. The quantizer is unique for all the features. An optimized Python implementation of the quantizer is shown in the listing below.

```python
def midtread_quantizer(train_set, test_set, bin_width):

    import numpy as np
    import numexpr as ne

    bin_width = np.float32(bin_width)
    # determine thresholds and centroids ---
    abs_max = np.max([-train_set.min(), train_set.max()])

    th = np.arange(bin_width/2, abs_max, bin_width, dtype=np.float32)
    centr = np.arange(bin_width/2+bin_width/2, abs_max+bin_width/2, bin_width
    , dtype=np.float32)

    th = np.concatenate((-th[::-1], th))
    centr = np.concatenate((-centr[::-1], np.array([0]), centr))

    assert th.size == centr.size - 1

    symbols = np.arange(centr.size, dtype=np.int32)

    # quantize with symbols ---
    train_set_q_s = np.zeros(train_set.shape,dtype=np.int32)
    test_set_q_s = np.zeros(test_set.shape,dtype=np.int32)

    for idx in range(1,th.size):
        th_min = th[idx-1]
        th_max = th[idx]
        sym = symbols[idx]
        train_set_q_s = ne.evaluate('where((train_set>=th_min)&(train_set<th_max),sym,train_set_q_s)')
        test_set_q_s = ne.evaluate('where((test_set>=th_min)&(test_set<th_max),sym,test_set_q_s)')

    # last level ---
    th_min = th[-1]
    sym = symbols[-1]
    train_set_q_s = ne.evaluate('where((train_set>=th_min),sym,train_set_q_s)')
    test_set_q_s = ne.evaluate('where((test_set>=th_min),sym,test_set_q_s)')

    # symbols to centroids ---
    bias = np.int32((symbols.size-1)/2)
    train_set_q_c = ne.evaluate('(train_set_q_s-bias)*bin_width')
    test_set_q_c = ne.evaluate('(test_set_q_s-bias)*bin_width')

    return train_set_q_s, test_set_q_s,train_set_q_c, test_set_q_c, symbols
```

The *numexpr* package has been extensively used to reduce the quantization time since it allows the fast evaluation of *where* expressions on arrays and vector by scalar multiplications. *Numexpr* parses the expression into its own operation-code, that is used by the integrated computing virtual machine. The array operands are split in small chunks, to fit the cache of the CPU, and then passed to the virtual machine. All the temporaries and constants in the expression are kept in the same small chunk as the operand, avoiding additional memory space and bandwidth waste.

The rate required to sent the quantized features is estimated with an arithmetic coder, working on each feature independently. For each split the number of occurrences of symbol i in feature j is determined on the training set ($n_{i,j}^{train}$). If symbol i never occurs in feature j then $n_{i,j}^{train}$ is set to 1. The a priori probability $p_{i,j}^{train}$ of symbol i in feature j is estimated as

$$p_{i,j}^{train} = \frac{n_{i,j}^{train}}{\sum_{l=0}^{L-1} n_{i,l}^{train}}$$

The number of occurrences of symbol i in feature j on the test set ($n_{i,j}^{test}$) is determined and the a posteriori probability of symbol i in feature j is estimated on the test set as

$$p_{i,j}^{test} = \frac{n_{i,j}^{test}}{\sum_{l=0}^{L-1} n_{i,l}^{test}}$$

The rate is finally estimated as

$$r = -\sum_{i=0}^{K-1} \sum_{j=0}^{L-1} p_{i,j}^{test} \cdot \log_2 \left(p_{i,j}^{train}\right)$$

The effect on the rate-accuracy plane of quantization and coding after PCA dimensionality reduction is shown in figure 3.2. Each line is obtained preserving a different number of PCA components $K = \{800, 1600, 3200, 6400\}$. The points in each line are obtained evaluating rate and accuracy at different quantization steps $Q = \{0.05, 0.03, 0.01, 0.005\}$.

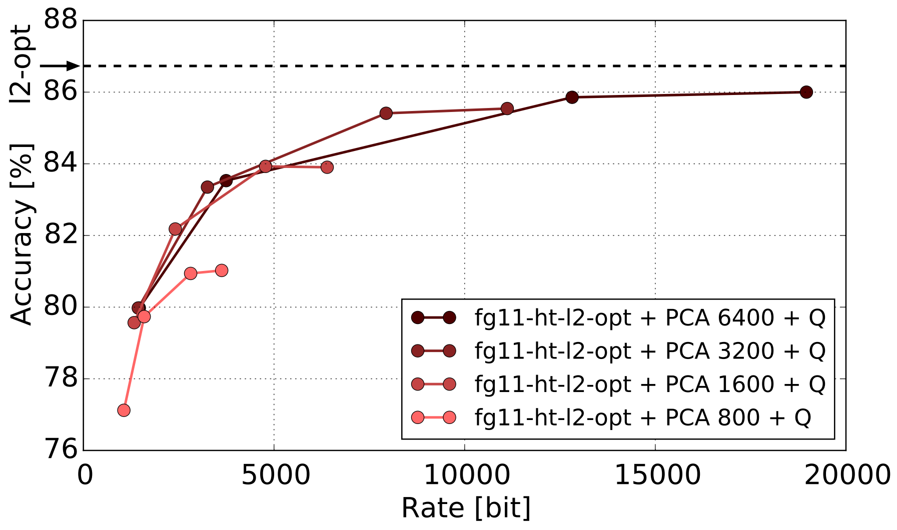


Figure 3.2: PCA dimensionality reduction, quantization and entropy coding: impact on the rate-accuracy plane

With respect to PCA dimensionality reduction without quantization, the reduction in terms of rate is over 90%, while the reduction of accuracy is less then 1%. Some numeric examples of are reported in table 3.2.

K	Q	**PCA only**		**PCA + Q + coding**	
		Rate [bit]	**Accuracy [%]**	**Rate [bit]**	**Accuracy [%]**
6400	0.005	204800	86.10	18963	86.00
	0.01			12818	85.86
3200	0.01	102400	85.56	7939	85.41
	0.03			3253	83.35
1600	0.03	51200	84.00	2410	82.18

Table 3.2: Dual layer pseudo-optimal model: the effect of quantization and entropy coding after PCA based dimensionality reduction on rate and accuracy

The combination of PCA based dimensionality reduction, quantization and entropy coding allows to reach a great level of compression while keeping an accuracy comparable to the one of the complete *fg11-ht-l2-opt* CNN model.

Chapter 4

Analyze-Then-Compress vs. Compress-Then-Analyze

In an Analyze-Then-Compress enabled Visual Sensor Network a sensor node acquires an image, extracts features from the image, encodes the features and sends the encoded features to a sink node. This kind of approach requires low complexity algorithms due to the limited amount of processing power available on sensor nodes.

In a Compress-Then-Analyze Visual Sensor Network a sensor node acquires an image, preforms lossy compression of the image and sends the compressed image to a sink node, where features extraction takes place.

Both the ATC and CTA paradigms need to face issues related to image or features transmission to the sink node. Noisy or band-limited transmission channels can significantly reduce the available bandwidth. Moreover, sensor nodes are battery powered and this requires special attention to the amount of data sent over the network due to the energy cost of network transmissions.

4.1 CTA approach

For the CTA approach JPEG compression is used as benchmark to evaluate:

- The JPEG compression time needed by a RaspberryPi for a greyscale 100 x 100 face image.
- The size (rate) of a 100 x 100 greyscale face image compressed at different quality factors.
- The accuracy on PubFig83 dataset when images are first compressed at different quality factors and then features are extracted with both *fg11-ht-l3-1* and *fg11-ht-l2-opt* models.

A greyscale 100 x 100 image requires on average only 11ms to be compressed on RaspberryPi, 1/100th the time needed for facial feature extraction using the *fg11-ht-l2-opt* CNN model on the same device.

The size of a JPEG compressed 100 x 100 greyscale face image at different quality factors is ranges from 7 to 62 kBit, as detailed in table 4.1.

Quality factor	**Size [kBit]**
10	6.90
30	11.03
50	13.94
70	17.64
90	28.83
100	62.09

Table 4.1: JPEG 100 x 100 greyscale face image size at different quality factors

4.2 ATC vs. CTA

The impact on face recognition accuracy is evaluated extracting features with the *fg11-ht-l2-opt* and *fg11-ht-l3-1* CNN models after JPEG compression at different quality factors. Figure 4.1 shows the comparison on the rate-accuracy plane between:

- ATC approach
  - Sensor node: image acquisition, features extraction with *fg11-ht-l2-opt* CNN, PCA based dimensionality reduction with $K = 1600, 3200, 6400$, quantization with $Q = 0.05, 0.03, 0.01, 0.005$, arithmetic entropy coding

  - Sink node: accuracy estimation

- CTA approach
  - Sensor node: image acquisition, JPEG compression at $QF = 10, 30, 50, 70, 90$
  - Sink node: features extraction - from the compressed image - with *fg11-ht-l2-opt* and *fg11-ht-l3-1* CNN models, accuracy estimation

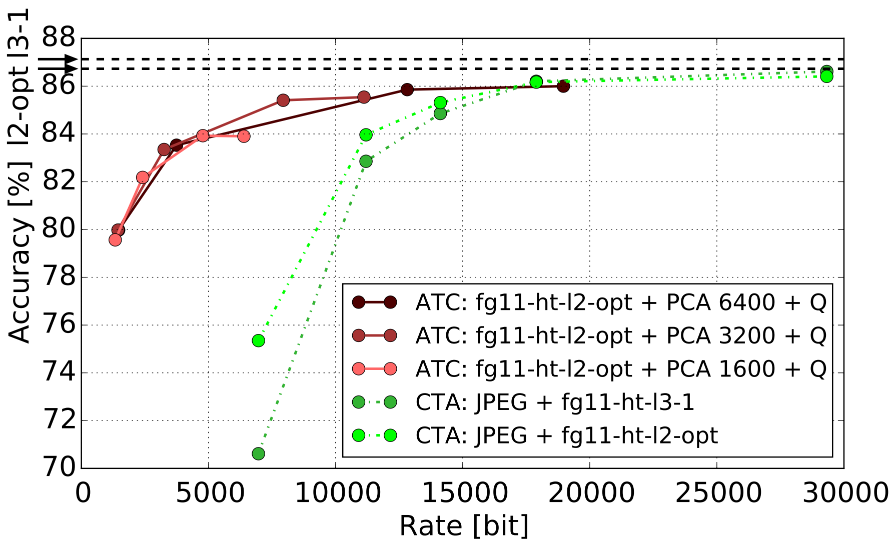


Figure 4.1: Analyze-Then-Compress vs. Compress-Then-Analyze

In a rate constrained environment, below 15kBit/image, the ATC paradigm is the only way to get high accuracy, even if this requires more processing time on sensor nodes to perform feature extraction, reduction, quantization and coding.

For example, with a maximum rate of 11kBit/image, CTA with JPEG compression at $QF = 30$ followed by *fg11-ht-l2-opt* model CNN reaches 83.96% accuracy, while ATC with nearly the same rate reaches 85.54% accuracy with $K = 3200$ and $Q = 0.005$. When the available rate falls below 10kBit/image the CTA accuracy falls significantly below 80% while the ATC approach can guarantee an accuracy level of 82.18% with $K = 1600$ and $Q = 0.03$ with a rate of only 2.4kBit/image.

Even when the available bandwidth is not so strictly limited, the ATC loss in accuracy with respect to CTA is only 1.13% considering in the case of JPEG with $QF = 100$ and ATC with $K = 6400$ and $Q = 0.03$.

Chapter 5

Conclusions and future work

The method proposed in this work to adapt an existing Convolutional Neural Network for Face Recognition, originally developed for a powerful computer, to run on a low power device shows that with a guided CNN model simplification the recognition accuracy can be kept at a high level while greatly reducing the energy impact, thanks to execution time and transmission size reduction. An interesting continuation on this line-up is the application of the same typology of CNN network simplification to other computer vision algorithms, ranging from object recognition to people tracking.

The ATC approach has shown to be really competitive with respect to CTA, not only when the available network bandwidth in a VSN is limited, but also in optimal networking conditions. The benefit of this paradigm change are extended also to privacy related issues arising when images are sent through a communication infrastructure. Certainly what ATC is missing is the availability of a faster CNN extractor implementation, able to run in a total time comparable to the one of JPEG compression.

From a VSN point of view the implementation of the proposed energy efficient CNN model in a real sensor network would require some additional efforts. Even if the actual Python code is highly optimized, and many of the most computationally intensive operations are executed with Fortran based libraries, a low level language porting (e.g. C++) of the whole CNN implementation would benefit from removing Python overhead on the operations. The development in a low level language could explore the time reduction due to the use of vectorized floating point NEON instructions (available on ARMv7 architectures with VFPv3 unit), the possibility to empower the GPU or the multimedia co-processor of parts of the calculations and even the possibility to run the algorithm on tiny, application specific lower power platforms, such as DSP.

Finally from an energetic perspective, the RaspberryPi power consumption specifications are not available, thus no estimation on the real power consumption are possible. The building of an accurate power consumption measurement setup, both of the networking module and of the processing module, could allow a more precise energy guided optimization.

Bibliography

[1] Daniel G. Costa, Ivanovitch Silva, Luiz Affonso Guedes, Francisco Vasques, and Paulo Portugal. Availability issues in wireless visual sensor networks. *Sensors*, 14(2):2795–2821, 2014. ISSN 1424-8220. doi: 10.3390/s140202795. URL `http://www.mdpi.com/1424-8220/14/2/2795`.

[2] W. W. Bledsoe. The model method in facial recognition. *Panoramic Research Inc.*, August 1966.

[3] Takeo Kanade. Picture processing system by computer complex and recognition of human faces. In *Doctoral dissertation, Kyoto University.* November 1973.

[4] L. Sirovich and M. Kirby. Low-Dimensional Procedure for the Characterization of Human Faces. *Journal of the Optical Society of America A*, 4(3):519–524, 1987.

[5] M.A. Turk and A.P. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pages 586–591, Jun 1991. doi: 10.1109/CVPR.1991.139758.

[6] P.J. Phillips, J.R. Beveridge, B.A. Draper, G. Givens, A.J. O'Toole, D.S. Bolme, J. Dunlop, Yui Man Lui, H. Sahibzada, and S. Weimer. An introduction to the good, the bad, amp; the ugly face recognition challenge problem. In *2011 IEEE Intl. Conference on Automatic Face Gesture Recognition (FG)*, pages 346–353, March 2011. doi: 10.1109/FG.2011.5771424.

[7] M. Gunther, A. Costa-Pazo, C. Ding, E. Boutellaa, G. Chiachia, H. Zhang, M. de Assis Angeloni, V. Struc, E. Khoury, E. Vazquez-Fernandez, D. Tao, M. Bengherabi, D. Cox, S. Kiranyaz, T. de Freitas Pereira, J. Zganec-Gros,

E. Argones-Rua, N. Pinto, M. Gabbouj, F. Simoes, S. Dobrisek, D. Gonzalez-Jimenez, A. Rocha, M.U. Neto, N. Pavesic, A. Falcao, R. Violato, and S. Marcel. The 2013 face recognition evaluation in mobile environment. In *Intl. Conference on Biometrics (ICB)*, pages 1–7, June 2013. doi: 10.1109/ICB.2013.6613024.

[8] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[9] Oren Barkan, Jonathan Weill, Lior Wolf, and Hagai Aronowitz. Fast high dimensional vector multiplication face recognition. In *The IEEE Intl. Conference on Computer Vision (ICCV)*, December 2013.

[10] Dong Chen, Xudong Cao, Fang Wen, and Jian Sun. Blessing of dimensionality: High-dimensional feature and its efficient compression for face verification. In *CVPR*, pages 3025–3032. IEEE, 2013.

[11] G.B. Huang, Honglak Lee, and E. Learned-Miller. Learning hierarchical representations for face verification with convolutional deep belief networks. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2518–2525, June 2012. doi: 10.1109/CVPR.2012.6247968.

[12] N. Pinto, Z. Stone, T. Zickler, and D. Cox. Scaling up biologically-inspired computer vision: A case study in unconstrained face recognition on facebook. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 35–42, June 2011. doi: 10.1109/CVPRW.2011.5981788.

[13] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[14] A. Marcus and O. Marques. An eye on visual sensor networks. *Potentials, IEEE*, 31(2):38–43, March 2012. ISSN 0278-6648. doi: 10.1109/MPOT.2011.2178279.

[15] A. Redondi, L. Baroffio, M. Cesana, and M. Tagliasacchi. Compress-then-analyze vs. analyze-then-compress: Two paradigms for image analysis in visual sensor networks. In *IEEE Intl. Workshop on Multimedia Signal Processing (MMSP)*, pages 278–282, Sept 2013. doi: 10.1109/MMSP.2013.6659301.

[16] L. Baroffio, M. Cesana, A. Redondi, M. Tagliasacchi, and S. Tubaro. Coding visual features extracted from video sequences. *IEEE Transactions on Image Processing (T.IP)*, 23(5):2262–2276, May 2014. ISSN 1057-7149. doi: 10.1109/TIP.2014.2312617.

[17] Mayssaa Al Najjar, Milad Ghantous, and Magdy Bayoumi. *Video Surveillance for Sensor Platforms - Algorithms and Architectures*, volume 114 of *Lecture Notes in Electrical Engineering*. Springer, 2014. ISBN 978-1-4614-1856-6. URL `http://dx.doi.org/10.1007/978-1-4614-1857-3`.

[18] Mohammad Rahimi, Rick Baer, Obimdinachi I. Iroezi, Juan C. Garcia, Jay Warrior, Deborah Estrin, and Mani Srivastava. Cyclops: In situ image sensing and interpretation in wireless sensor networks. In *In SenSys*, pages 192–204. ACM Press, 2005.

[19] Meiyan Zhang and Wenyu Cai. Vision mesh: A novel video sensor networks platform for water conservancy engineering. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 4, pages 106–109, July 2010. doi: 10.1109/ICCSIT.2010.5565158.

[20] Wu chi Feng, Wu chang Feng, and Mickael Le Baillif. Panoptes: Scalable low-power video sensor networking technologies. In *In MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 562–571. ACM Press, 2003.

[21] RaspberryPi Fundation. Raspberrypi soc: Bcm2835. `http://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/README.md`.

[22] ARM Ltd. Arm1176jzf-s technical reference manual. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301h/DDI0301H_arm1176jzfs_r0p7_trm.pdf`.

[23] D. Cox and N. Pinto. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. In *Automatic Face Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on*, pages 8–15, March 2011. doi: 10.1109/FG.2011.5771385.

[24] N. Pinto, N. Poilvert, and G. Chiachia. Convolutional neural networks using random filter weights. `https://github.com/giovanichiachia/convnet-rfw`.

[25] N. Kumar, A.C. Berg, P.N. Belhumeur, and S.K. Nayar. Attribute and simile classifiers for face verification. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 365–372, Sept 2009. doi: 10.1109/ICCV.2009.5459250.