# WhoAreYou (WAY): A Mobile CUDA Powered Picture ID Card Recognition System

Raffaele Montella, Alfredo Petrosino(✉), and Vincenzo Santopietro

Department of Science and Technology, University of Naples Parthenope, Naples, Italy
petrosino@uniparthenope.it

**Abstract.** The paper reports a novel cloud based approach for image matching between high-resolution images of faces and low resolution images of ID Cards. We design our application matching the mobile cloud computing design guidelines with the use of CUDA kernel invocation from regular mobile devices (devices that naively don't support CUDA GPGPUs) as a novel contribution. Face matching is performed by the OpenFace deep neural network, which evaluates pre-processed images in cloud, whilst pre-processing is done on mobile device. To test our system, we built an image dataset of 30 subject caputeres in 10 different poses, denoised to reduce any traces of stamps or watermark on the ID cards, mixed to the well known ORL and LFW datasets.

**Keywords:** Cloud computing · Neural network · Deep learning
Face matching

## 1 Introduction

This paper proposes a cloud based solution to automatic face matching between low resolution photos on ID Cards and high-resolution face photos. This solution could be applied to every possible scenario where an identification is required. The are several aspects that may affect performances, that may be due to the person that is going to be identified, due to the documents or to the acquisition system, like hair-style changes, since the photo on ID Cards is shot way before the identification step (*aging*), stamps or watermarks on the ID Card and other aspects introduced by the acquisition system like the resolution of the device used to scan the ID Card. As instance, in Fig. 1, we can clearly see some of the aspects we discussed about, for example the poor conservation of the ID Card on the right that led to the erosion of the photo.

The contribution of the present paper is thus twofold. We design our application matching the mobile cloud computing design guidelines with the use of CUDA kernel invocation from regular mobile devices (devices that naively don't support CUDA GPGPUs) to allow fase detection be a service to provide to user who usually adopts mobile devices. Also, we describe a novel benchmarking framework that we set up in order to evaluate and compare face detection based


S. Battiato et al. (Eds.): ICIAP 2017 International Workshops, LNCS 10590, pp. 375–382, 2017.
https://doi.org/10.1007/978-3-319-70742-6_35


**Fig. 1.** Two ID Cards obtained in different cities.

on matching for recognizing person identity by his/her face compared with ID Card. The results are devoted to assess to what extent typical aspects in face matching challenges pose troubles to common, like EigenFaces and LBPH, and novel, like deep neural OpenFace, methods.

The paper is organized in four sections. Section 2 describes the system, while Sect. 3 describes the algorithms adopted for benchmarking. Section 4 reports results and discusses some critical issues. Sections 5 draws some conclusions and future developments.

## 2 Design and Architecture

The capabilities of mobile devices have been improving very quickly in terms of computing power, storage, feature support, and developed applications [4]. We design our application matching the mobile cloud computing design guidelines with the use of CUDA kernel invocation from regular mobile devices (devices that naively don't support CUDA GPGPUs) as a novel contribution. To make it possible we use GVirtuS4j [9] a pure Java framework enabling Android devices to CUDA remote invocation calls using the GVirtuS Linux back-end [8]. The developer prototypes the kernels on a CUDA enabled regular x86_64 machine. Then the compiled kernels are embedded in the application project and invoked using GVirtuS4j. The actual GPU used by the mobile phone could be hosted on cloud or on premises: it works in both cases. Nevertheless for a real working ID Card face recognition application the scalability and the availability is one of the main design requirements. We used the Amazon Web Services because offering an advanced Infrastructure as a Service deployment, affordable GPGPU equipped virtual machine instances, a complete API for programmatical interaction with the cloud and, last but not the least, a good level of service uptime.

The Figure 2 represents system big picture.

When the application is deployed in the wild, the user scan the ID-Card picture using the WAY app. The app performs all the preparation steps using the CUDA kernels invoked remotely in order to extract the face features. Then a

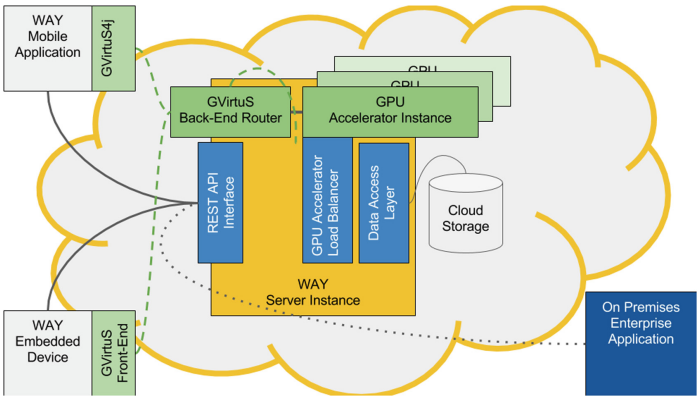


**Fig. 2.** The architectural schema of the Amazon Web Service deployment. Dotted lines represent GVirtuS CUDA remote invocations performed by mobile or embedded devices. Filled lines are regular REST requests/responses performed by client applications or remote enterprise infrastructures.

REST service is invoked for the main recognition step. The result is returned by the service. The both stages could be computationally intensive, battery eager, and privacy demanding. With the proposed system we overcome the three pits:

- Compute-intensive operations: the GVirtuS load balancer distributes CUDA remote kernel invocations on different GPGPU enabled instances in respect of computation load and costs;
- Battery-consuming task: the computation is offload tightly for CUDA invocation and loseley for feature recognition with the main target of energy usage drainage mitigation.
- Privacy-enforcing: there is no ID-Card picture related image transfer among the WAY app and the recognition engine: the app invokes the CUDA kernel remotely in order to extract the pattern's features. The app leverages on a remote REST web service to get data about the recognized picture.

GVirtuS (back-end) and GVirtuS4j (front-end) implement a split driver based virtualization and remoting. While other similar software components for GPGPU vitalization are strongly CUDA oriented, GVirtuS is completely plug-in based: both front-end/back-end and the communicator could be enhanced implementing new capabilities.

The GPU virtualization architecture is based on a split-driver model [3] (also known as driver paravirtualization), involves sharing a physical GPU. Hardware management is left to a privileged domain. A front-end driver runs in the unprivileged VM and forwards calls to the back-end driver in the privileged domain [5]. The back-end driver then takes care of sharing resources among virtual machines. This approach requires special drivers for the guest VM. The split driver model is currently the only GPU virtualization technique that effectively allows sharing

the same GPU hardware between several VMs simultaneously [7]. This framework offers virtualization for generic GPU libraries on traditional x86 computers. At the current state, GVirtuS supports leading GPGPU programming models such as CUDA and OpenCL. It also enables platform independence from all the underlying involved technologies (i.e. hypervisor, communicator, and target of virtualization).

The GPU Accelerator Load Balancer component is responsible for GPGPU enabled instances metrics measurement and related policy enforcement in order to honour the expected performances.

The GVirtuS Back-End Router component ensure the coherence between GPGPU enabled clients and the cloud hosted back-end instances.

### 2.1 The Toolkit Libraries

In order to extend the set of CUDA functions supported by GvirtuS, it necessary to define three main components for each library that are responsible for the communication between the guest and host machine:

- Front-end Layer
- Back-end Layer
- Function Handler

The first one contains the definitions of the wrapper functions called by the client, with the same signature as the library ones, where the name of the requested routine and the addresses of the input parameters, variables and host/device pointers, are encapsulated in a buffer that is sent to the back-end through a communicator. For each CUDA toolkit library the back-end layer stores a function handler that declares, for each function of the toolkit library, a handler function used to execute the requested routine properly. Once the buffer is received by the back-end, the handler retrieves the input parameters, executes the requested routine and if needed sends a buffer containing the output variables and host/device pointers back to the client. With the described technique we implemented cuFFT, cuBLAS, cuRAND e cuDNN libraries we used for the algorithm implementation.

## 3 Face Matching

In this section we are going to describe the algorithms used and their performances, focusing on the approach based on deep learning.

### 3.1 Eigenfaces

Given an image of a face that has to be recognized, the algorithm [12] compares the input image with other images of known subjects. The most meaninfgul features are called eigenfaces, since they represent the eigenvectors of the features-space. We can recognize a face by comparing the sum of the eigenvectors of an unknown face and the sum of the eigevectors of a known face. This approach may fail often if the background is rich of details or the size of the face changes.

### 3.2 LBPH

Local Binary Patterns (LBP) [1] is a highly used operator in computer vision. This approach is robust when there are light changes, since it uses local features. LBP is based on a non-parametric operator defined as a binary string (descriptor of $p$), which given a pixel $p$ in position $(x, y)$, estimates the structure of the neighborhood of $p$. Let's consider a $3 \times 3$ window, as shown in Fig. 3, each bit of the string is 1 if the value of the pixel of the corresponding pixel in neighborhood is less than the central pixel. Furthermore, the LBP operator has been extended (Extended Local Binary Patterns) in order to handle windows of arbitrary size.

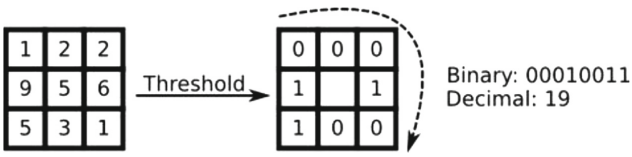


**Fig. 3.** LBP operator example

### 3.3 OpenFace

OpenFace [2] is a Python and Torch implementation of face recognition with deep neural networks [11]. OpenFace is based on the logic flow shown in Fig. 4 to compute low-dimensional representations for the faces in the image. The face detection procedure returns a list of bounding boxes, but there's the problem that a face could not be looking right into the camera. OpenFace handles this situations by applying a 2D affine transformation, which also crops the face so the image is $96 \times 96$ pixels. This image in then processed by the neural network estimates the normalized probability for each user of our database.

## 4 Evaluation

We have conducted four tests in order to evaluate the performances of the approaches mentioned before for face recognition between the photo extracted from the ID Card and a high-resolution photo. We used:

1. ID Card photos and high-resolution photos of 30 subjects (FRs)
2. ORL Database
3. LFW database

In order to create the dataset the first dataset used for performance evaluation, we used a 13 megapixels camera to take high-resolution images of faces and a Brother MFC 1910W to scan the ID Cards. High-resolution face photos have been taken at a distance of 150 cm in a closed environment with white background. We collected data from 30 subjects, whose age goes from 25 up to 45. These data can be arranged in three subsets:

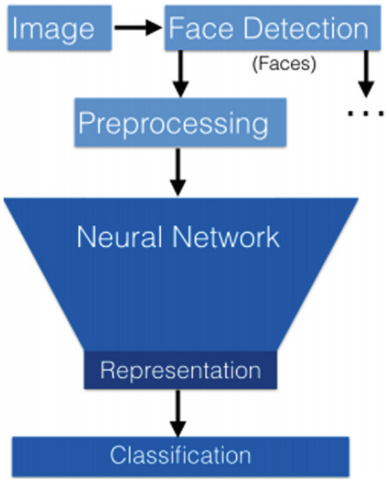


**Fig. 4.** Face recognition with Neural Networks, as shown in [2]

- **Face subset** (Fs), which contains high-resolution face images
- **ID Card Face subset** (IDCFs), which contains the photos on the scanned ID Cards
- **Face Recognition subset** (FRs), which contains the pre-processed images from the Fs and IDCFs.

The ORL [10] dataset of faces, now claimed AT&T "The Database of Faces", includes ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open/closed eyes, smiling/not smiling) and facial details (glasses/no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).

Labeled Faces in the Wild [6] is a database of face photographs designed for studying the problem of unconstrained face recognition. The data set contains more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set.

For each test we performed the training of the faces included in the FRs, a substet of identities from ORL and FLW databases. Then the face recognition procedure is executed over the face extracted from an ID Card. For each test we changed the number of identities that composes the training set, setting it to 10,25,50,100. Since we only have 30 identities that are linked to an ID Card, we performed 30 test of identification. We performed tests on a remote machine with an Intel Core i5-3320m CPU and a Nvidia Geforce 920m. Figure 5 show that the OpenFace solution is of course the most accurate.

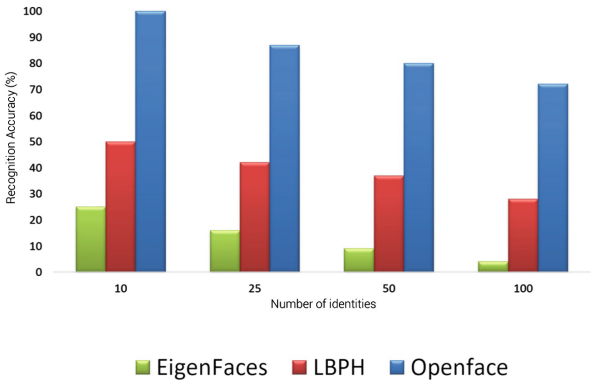


**Fig. 5.** Algorithms' accuracy

## 5 Conclusions and Future Works

The exponential growth of computer hardware performance is leading to faster matching. A system like WAY uses a video stream which is already being captured in most of the places in which security is important, like airports, banks etc. As future work we could use the proposed system with company badges too and not just with ID Cards, capturing employees entrance end exit times. This kind of application is not affected by aging and light changes, since the training photos of the employees can be updated regularly and the camera is set in the always in the same place.

**Acknowledgement.** This research has been funded by the European Commission under the Horizon 2020 program through Grant Agreement No 644312, corresponding to the "Heterogeneous Secure Multi-level Remote Acceleration Service for Low-Power Integrated Systems and Devices" (RAPID) project.

## References

1. Ahonen, T., Hadid, A., Pietikainen, M.: Face description with local binary patterns: Application to face recognition. IEEE Trans. Pattern Anal. Mach. Intell. **28**(12), 2037–2041 (2006)
2. Amos, B., Ludwiczuk, B., Satyanarayanan, M.: Open-face: a general-purpose face recognition library with mobile applications. Technical report CMU-CS-16-118, CMU School of Computer Science (2016)
3. Armand, F., Gien, M.: A practical look at micro-kernels and virtual machine monitors. In: Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference, CCNC 2009, pp. 395–401. IEEE Press, Las Vegas (2009). ISBN:978-1-4244-2308-8, http://dl.acm.org/citation.cfm?id=1700527.1700644
4. Bahl, P., et al.: Advancing the state of mobile cloud computing. In: Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services, pp. 21–28. ACM (2012)

5. Dunlap, G.W., et al.: Execution replay for multiprocessor virtual machines. In: VEE 2008 - Proceedings of the 4th International Conference on Virtual Execution Environments, pp. 121–130 (2008). ISBN: 9781595937964. https://doi.org/10.1145/1346256.1346273
6. Learned-Miller, E., et al.: Labeled faces in the wild: a survey. In: Kawulok, M., Emre Celebi, M., Smolka, B. (eds.) Advances in Face Detection and Facial Image Analysis, pp. 189–248. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-25958-1_8
7. Li, W.: GPU-based computation of Voxelized Minkowski Sums with Applications. PhD thesis. University of California (2011)
8. Montella, R., Giunta, G., Laccetti, G.: Virtualizing high-end GPGPUs on ARM clusters for the next generation of high performance cloud computing. Cluster Comput. **17**(1), 139–152 (2014)
9. Montella, R., Giunta, G., Laccetti, G., Lapegna, M., Palmieri, C., Ferraro, C., Pelliccia, V.: Virtualizing CUDA enabled GPGPUs on ARM clusters. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (eds.) PPAM 2015. LNCS, vol. 9574, pp. 3–14. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32152-3_1
10. Samaria, F.S., Harter, A.C.: Parameterisation of a stochastic model for human face identication. In: Proceedings of 1994 IEEE Workshop on Applications of Computer Vision, pp. 138–142 (1994). https://doi.org/10.1109/ACV.1994.341300
11. Schro, F., Kalenichenko, D., Philbin, J.: FaceNet: a unied embedding for face recognition and clustering. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
12. Turk, M., Pentland, A.: Eigenfaces for Recognition. J. Cognit. Neurosci. **3**(1), 71–86 (1991). ISSN: 0898–929X