CNN Inference Acceleration using low-power devices for Human Monitoring and Security Scenarios

Juan Mas[a], Teodoro Panadero[b], Guillermo Botella[a,c], Alberto A. Del Barrio[a], Carlos García[a,c,*]

[a] *Fac. Informática, Universidad Complutense Madrid (Spain)*
[b] *ING Direct, Las Rozas, Madrid (Spain)*
[c] *Instituto de Tecnología del Conocimiento-Universidad Complutense de Madrid (Spain)*

Abstract

Security is currently one of the top concerns in our society. From governmental installations to private companies and medical institutions, they all have to address directly with security issues as: access to restricted information quarantine control, or criminal tracking. As an example, identifying patients is critical in hospitals or geriatrics in order to isolate infected people, which has proven to be a non- trivial issue with the COVID-19 pandemic that is currently affecting all countries, or to locate fled patients. Face recognition is then a non-intrusive alternative for performing these tasks. Although FaceNet from Google has proved to be almost perfect, in a multi-face scenario its performance decays rapidly. In order to mitigate this loss of performance, in this paper a cluster based on the Neural Computer Stick version 2 and OpenVINO by Intel is proposed. A detailed power and runtime study is shown for two programming models, namely: multithreading and multiprocessing. Furthermore, 3 different hosts have been considered. In the most efficient configuration, an average of 6 frames per second has been achieved using the Raspberry Pi 4 as host and with a power consumption of just 11.2W, increasing by a factor of 3.3X the energy efficiency with respect to a PC-based solution in a multi-face scenario.

Keywords: Convolutional Neural Network, OpenVINO, Neural Compute

*Corresponding author
Email address: garsanca@ucm.es (Carlos García)

1. Introduction

Year 2012 was a turning point in the classification of images. Since the appearance of AlexNet [1] by that year, the utilization of Deep Neural Networks (DNNs) for classifying million of images has proved to be even more effective than humans [2]. While the application of DNNs mostly focused on cloud-based systems, nowadays the trend is changing, as there is an increasing demand for running these algorithms on embedded systems at the edge [3].

In this scenario, face recognition has become a major goal of machine learning and deep learning development. This is a critical step for security cameras or in general any system tracking people [4]. Being able to identify a concrete person among the crowd is essential for security purposes. In first place, face recognition was limited to high restricted, security areas where it was used to verify the identity of the people who tried to access. However, it has recently widened its scope to other security issues like criminal tracking in public areas, which has drifted into many technological issues related to the increase in the number of faces to infer.

As mentioned in [5], face recognition became popular using holistic methods as the well-known Eigenfaces [6] or local-based features algorithms [7] . Unfortunately, these methods failed to produce the expected results. In recent years, the explosion of Big Data and Deep Learning has allowed some implementations to get noticeably high levels of accuracy (up to 99.97%), as it is the case of Google FaceNet [8]. These methods are based on Convolutional Neural Networks (CNN) [9] and, according to the results presented by the Labeled Faces in the Wild (LFW) [10], they are almost as accurate as humans. Nevertheless, in spite of these high accuracy levels, the performance of the system depends on the number of faces to be recognised in each frame. Thus, in order to comply with these performance requirements, it is necessary to increase the computation capability of the system. An example of this statement is shown in Table

Table 1: Performance of FaceNet on an i7-4702MQ@2.2GHz. The Frames Per Second are measured for 299x299 pixel videos. "R" means random number of faces across the video.

Number of faces	Min FPS	Avg FPS	Max FPS
1	15	21.84	26
5	1	6.03	8
15	1	1.78	7
R	1	9.77	14

1. As observed, the number of frames per second [11, 12] rapidly decreases as the number of faces appearing on a video increases, which motivates the usage of an accelerator to keep a higher performance.

Currently the efforts have moved from improving the accuracy to lighten the neural networks [13] and accelerate inference times [14, 9], which leads to further degrading the performance of the system when many faces need to be analyzed. Moreover, the previously mentioned decrease in performance is associated with an increase in the power consumption, which is very relevant when talking about monitoring applications. This paper will focus on the employment of energy efficient embedded systems to perform the inference stage. Concretely, in this work a cluster of 3 Intel Neural Compute Stick 2 [15] is proposed to mitigate the aforementioned losses of performance with a low-cost and low-power hardware. Together with the OpenVINO toolkit [16], these are the Intel basic technologies for accelerating inference. This toolkit allows to optimize the neural networks before the deployment and tweak them in order to improve their performance in tightly constrained platforms.

Leveraging these technologies, then in this paper we propose a real-time face recognition application based on FaceNet [8]. The inference stage will be accelerated through the NCS2 cluster. Results show that it is possible to increase the performance taking advantage of the horizontal scalability of these kind

of accelerating hardware devices drifting into interesting low-cost alternatives for the deployment of the aforementioned applications. Therefore, our major contributions can be listed as follows:

- The construction of a low-power and low-cost NCS2 cluster to run a face recognition application.
- The mitigation of the performance decay in a multi-face scenario, while providing an energy efficient solution.
- The evaluation of two programming models to run the face recognition applications, namely: one based on multithreading and other based on multiprocessing, the latter being the best match for the proposed cluster.
- A scalability study based on three different hosts: a laptop and two embedded devices as Raspberry Pi 3B+ and 4.
- The proposed cluster with 3 NCS2 sticks has achieved an average of 6 FPS (10 FPS peak) in multi-face scenarios, increasing the energy efficiency by a factor of 3.3X with respect to a CPU-based solution.

The rest of the paper is organized as follows: Section 2 describes some preliminary concepts about Facenet; Section 3 describes the state of the art regarding the different acceleration technologies available and the prior attempts of face recognition over the NCS we are using for the research; Section 4 will describe the hardware architecture mounted to make the experiments; Section 5 refers to the software application developed and some of its main features; Section 6 explains the carried out experiment and presents the results obtained and finally Section 7 gives our remarks on the work.

2. Facenet

FaceNet was presented in 2015 by F. Schroff, D. Kalenichenko, and J. Philbin [8]. It is a model that allows representing facial features into a unified format, named *embedding*, composed of 128 values. As can be seen in Figure 2, an

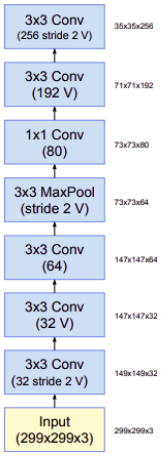


Figure 1: General stem of an Inception ResNet-v1 CNN architecture.

embedding is a vector-like representation that digests the discriminant features of a certain human face. This representation has been proven to be very useful in literature, as it is not necessary to apply Support Vector Machine (SVM) or Principal Component Analysis (PCA) classification algorithms after performing the inference stage, which is a very common practice in image recognition. In fact, in order to predict whether two faces are the same or not, it is only necessary to obtain the euclidean distance between two different embeddings and check if this distance is lower than a certain threshold fixed beforehand.

This model has an accuracy of 99.96% according to the LFW benchmarks [10] and is widely used by the community. It is usually implemented using Tensorflow [17] and many Inception [18] variants. An Inception ResNet-v1[19] has been used in this research. This architecture provides high levels of accuracy while managing to mantain a reduced computational cost. An overview of the Inception Resnet-v1 CNN is depicted in Figure 1. As seen in the picture, the input size is 299x299 pixels. Thus, prior to the inference phase, images must be resized to this resolution in order to be digested by the CNN. This fixed input makes the inference agnostic to the original resolution of the video, so increasing this parameter would only impact the preprocessing phase.

In Figure 2, a real example consisting of 3 different image embeddings is shown, with their euclidean distance calculated in order to check if they are the same face or not. Face A is different from B and C which correspond to the same person. The threshold (1.20), is first set following the suggestions written in the official documentation of the pre-trained model. In any case, this has been later verified as a suitable value after performing numerous tests.

$$A = \begin{bmatrix} -0.0457619, 0.1293915, -0.0316607, \\ -0.0340939, -0.0560946, -0.0381384, \\ ... \\ 0.1105223, 0.0640519, -0.0385667 \end{bmatrix}$$

$$B = \begin{bmatrix} -0.0239239, -0.0061291, -0.0948852, \\ -0.0508952, -0.0372205, -0.0342746, \\ ... \\ -0.0131129, 0.1026618, 0.1148589 \end{bmatrix}$$

$$C = \begin{bmatrix} 0.0581797, 0.1263990, -0.1173248, \\ -0.1074057, -0.0122532, -0.1002343, \\ ... \\ 0.0469320, 0.0187737, 0.0711396 \end{bmatrix}$$

$$d(A, B) = 1.475706$$

$$d(B, C) = 0.6011642$$

Figure 2: FaceNet embeddings of faces A, B and C

3. Related Work

According to W. Shi [20], Edge Computing is defined as "the enabling technologies allowing computation to be performed at the edge of the network, on

downstream data on behalf of cloud services and upstream data on behalf of IoT services". As it has been mentioned in Section 1, Big Data approaches boosted the performance and utilization of CNNs for implementing face recognition. However, if the inference stage is conducted at the edge, the bandwidth can be saved for other purposes and the latency of the application is reduced. For this reason, during the last years many companies like Nvidia or Intel have invested important amounts of money in hardware specifically designed for performing the inference stage of neural networks on the edge.

Intel has designed the Intel Movidius Neural Compute Stick (NCS) [21, 22] and its revision, the Neural Compute Stick 2 (NCS2). The NCS and NCS2 are small USB 3.0-based sticks which contain Myriad 2 and Myriad X Vision Processing Units (VPUs), respectively, and they can be plugged-in any device based on Windows, Linux, MacOS or Raspbian. Myriad 2 [22] includes 4Gbits of LPDDR3 DRAM, imaging and vision accelerators, and an array of 12 VLIW (Very Long Instruction Word) vector processors called *shaved* processors. These processors are used to accelerate neural networks by running parts of the neural networks in parallel. Among many improvements, Myriad X mounts 16 shaved processors and is able to employ the half-precision floating point format, i.e. 16-bit numbers. These sticks can be used along the Intel OpenVINO toolkit [16]. This toolkit allows to optimize CNNs and deploy the neural networks on one or various Intel devices (CPU, GPU, NCS, FPGA, etc). Other examples of edge devices are the Jetson boards by Nvidia [23], or the Coral ones by Google [24], both possessing software support from TensorRT and TensorFlow, respectively.Hardware comparisons have already been made between the NCS2 and the Coral USB [25], however, we highlight the main differences in Table 2.

Although these are pretty recent technologies, there is an increasing interest on them. As an evidence we can find some works as the one by Kristiani et al. [26] for image classification using NCS2, or the work by N. Adnan et al. [27], which leverages the use of a Raspberry Pi in combination with the NCS to perform object detection. In other plane, the usage of CNNs for medical research using OpenVINO in combination with the new Intel Xeon CPUs [28],

the efforts in deploying CNNs on FPGAs [29] or the work of Lin et al. [30], studying OpenVINO and TensorFlow for deploying a traffic sign classification and detection system on an FPGA, are examples that prove the variety of scenarios in which this framework could be further used, besides embedded devices and IoT use cases.

Table 2: Comparison among state-of-the-art embedded devices.

Board	Price($)	Consumption (Watt)	Specifications
Nvidia Jetson TK1	NA	10.86[1]	NVIDIA Kepler Quad-Core ARM Cortex-A15 2 GB memory, 16 GB eMMC
Nvidia Jetson TX2	479.00[3]	15[2]	NVIDIA Pascal GPU 256 cores Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-Core ARM Cortex-A57 MPCore 8GB 128-bit LPDDR4 Memory 32GB eMMC 5.1
Raspberry Pi 3B+	48.42[3]	5[2]	Broadcom BCM2837B0 Cortex-A53 SoC@1.4GHz 1GB LPDDR2 SDRAM
Raspberry Pi 4	66.04[3]	9[2]	Broadcom BCM2711 Cortex-A72 1,5GHz Quad-Core Broadcom VideoCore VI 500MHz 4GB LPDDR4-2400 SDRAM
Neural Compute Stick 2	79.54[3]	1.2	MYRIAD X VPU USB 3.1 Type-A (data/power) 16 Vector VLIW "SHAVE" Processors
Google Coral USB	69.9[3]	2	Google Edge TPU coprocessor USB 3.0 Type-C* (data/power)
Google Coral Dev Board	153.29[2]	2	NXP i.MX 8M SoC (quad Cortex-A53, Cortex-M4F) Google Edge TPU coprocessor Integrated GC7000 Lite Graphics 1 GB LPDDR4 8 GB eMMC

1. Total Power at DC input of the board. See [31].
2. Maximum Power Consumption.
3. Prices from official distribution stores in Spain.

Regarding FaceNet, there has been several attempts trying to implement face recognition applications on embedded systems. For instance, OpenFace [32], which is a face recognition library specially designed for mobile platforms. In 2019, E. Jose et al. [33] implemented FaceNet and Multitask Cascaded

Convolutional Networks (MTCNN) [34], which is an algorithm composed of 3 CNNs for detecting and aligning faces within an image, on a Jetson TX2 in order to implement a multicamera surveillance system. Also in 2019, Salhi et al. [35] developed an evolutionary face recognition algorithm and deployed it on a Jetson TK1 with some real time oriented improvements. Our study differs from them in the cost and power of the employed technologies, which are lower than those of the Jetson TX2, with a cost of $399 in the offical Nvidia store [36], while the NCS2 costs $78,35. It must be noted that the TK1 model is currently discontinued. A further comparison is shown in Table 2.

It is also possible to find works in literature combining face recognition and NCS. For instance, the work by Wang et al. [37] or [38], by Moka and Morris, which utilize a NCS and Intel Movidius Neural Compute SDK (NCSDK) [39] to accelerate face recognition on a Raspberry Pi. In this paper we will focus on its successor, the NCS2 and OpenVINO, as NCSDK does not support NCS2. Regarding this, there are already some studies using the NCS2 with face recognition too. That is the case of the work by Yuan Xie et al. [40], where they analyze the overall speedup of Sphereface using Mobilenet on a hardware configuration consisting of a Raspberry Pi 3B+ with an NCS2, obtaining performance of 7.031 FPS. In this paper, the focus is not set in analysing the performance of the NCS2 with face recognition algorithms but in mitigating the loss of performance in multiface scenarios with a cluster hardware configuration of NCS2.

None of the aforementioned works studies the possibility of horizontally scaling this technology, taking into account not only the performance in different situations but also the cost of the technology and the power consumption. Therefore, in this paper, a cluster of 3 NCS2 is proposed to mitigate the performance decay when employing FaceNet on multiface-images. Furthermore, the power of the system will be studied under different scenarios and configurations.

4. Architecture of the System

4.1. Hardware

The hardware architecture of the system is depicted by Figure 3. As can be observed, the accelerating NCS2 cluster appears on the rightmost part of the figure, being connected to the host through an active hub, which is powered by an external power supply. The active hub is required to provide the necessary voltage and current to the NCS2 sticks. Otherwise, specially if the host is an embedded platform as Raspberry, the cluster is not properly powered and cannot achieve a good performance (instability of the usb connection causing losses of information and a downgrade in overall performance). An INA260 module is also attached to the connection between the hub and the supply in order to measure the power consumption through the capture of both voltage and current. The voltage and current values are then read through the INA260 I2C interface using an independently powered Raspberry PI 3B. Additionally, the host power supply, when the host is a Raspberry Pi 3B+, has also been monitored through the INA260. In the following subsections, there is a description of every module taking part in the proposed architecture.

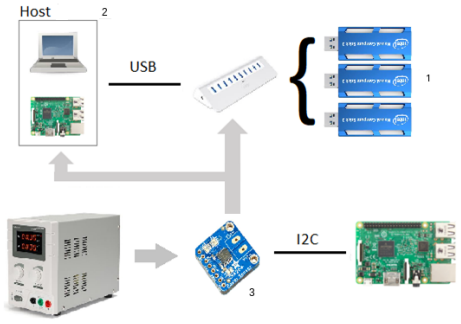


Figure 3: Hardware architecture of the system. 1.- NCS2 Cluster. 2.- Host Device. 3.- INA 260

4.1.1. NCS2

The neural sticks by Intel are the key elements composing the cluster. In this work, we have mounted from 1 to 3 in order to evaluate different performance-

power trade-offs. Each NCS2 can execute parallel inference requests through an Asynchronous API. According to the official documentation [41], the optimal number of concurrent inferences is 4 for each device, so this value has been set in our framework.

4.1.2. Host

In the proposed architecture, two different types of hosts have been tested. On the one hand, a laptop possessing an Intel Core i7-4702MQ@2.20GHz, 8GB RAM DDR3, 500GB SSD SATA3 running an Ubuntu 16.04.06-LTS. On the other hand, two embedded devices have been evaluated: a Raspberry Pi 3B+, which possesses a SoC Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit@1.4GHz, 1GB LPDDR2 SDRAM and a Micro-SD of 32GB; and the lastest generation Raspberry Pi 4, which contains a SoC Broadcom BCM2711 Quad Core Cortex A-72@1,5 GHz, 4GB SDRAM LPDDR4-2400 and a 32GB Micro-SD. In these cases the operating systems are Raspbian Stretch and Raspbian Buster, respectively.

The NCS2 sticks are connected to the host using an USB3.0 active hub. By employing this hub, we assure that no power shortage will affect the NCS2 performance.

4.1.3. INA260

The INA260 is a precise digital current and voltage monitor, so with the product of both magnitudes it is easy to obtain the consumed power. It is compatible with 3V and 5V logic and can measure up to +36VDC. Thanks to the integrated stunt resistor, the chip allows measuring up to +15A in either high or low side. It is compatible with Arduino and Raspberry Pi through its I2C interface.

4.2. Software

The OpenVINO toolkit allows developers to optimize and deploy CNNs for accelerating inferences at the edge. In our application, the Model Optimizer

and the Inference Engine components are being employed. These will be explained in detail in the following subsections. The overall software architecture is represented in figure 4.

4.2.1. Model Optimizer

Tradicionally, when talking about implementations of AI applications, there is a division between training/development and deployment stages. The Model Optimizer (MO) is a command line tool that can be used for adjusting and fine-tuning neural network models in order to accelerate the inferences in later stages. The MO is fed with the definition files of the model (.pb file in case it is a Tensorflow model) and provides two Intermediate Representation (IR) files. These files consist of a .xml file defining the layers, sizes and connections and a .bin file determining the weights of each parameter. When executing the optimization, the user can introduce some other configuration parameters such as floating point precision [42] (FP32, FP16, INT8), channel inversion, layer fusing, etc. However, the method used by the MO to optimize the model is not explained in detail, so developers lose some control over the fine-tuning of the model.

In this paper, an optimized CNN for face detection (face-detection-retail-004) provided by Intel as well as David Sandberg's [43] implementation of Facenet have been used. The area under the recall curve of the face detection network (also known as the Average Precision) is 83%, tested on the WIDER [44] dataset, which possesses faces larger than 60x60 pixels. During the testing phase of the neural network, an unstable behaviour has been observed when reducing the resolution of a single face below 60x60 pixels. In this scenario, we have detected cases in which the faces were not properly detected (wrong bounding limits) and cases in which the face was not detected at all. Finally, both models have been optimized using FP16 format. This format has been chosen because of the computing constraints of the MYRIAD VPU, which does not allow other formats as FP32. Nonetheless, it is worthy to note that the OpenVINO framework can of course optimize models with FP32 format when

targeting CPU and GPU units as inference devices.

4.2.2. Inference Engine

The MO helps developers to make the transition from the training phase to the deployment phase, but the tool that is really in charge of executing the inference is the Inference Engine (IE), which can be installed independently of the rest of components of OpenVINO, as it must be installed in the inference device. That is the reason why, in devices such as Raspberry Pi, only the IE can be installed. This modular structure helps developers to truly separate both environments (training and deployment) only needing two light-weighted, optimized files.

The IE is in charge of executing the inferences as transparent as possible from the developer. It loads the IR files and creates an executable network object, which is then loaded into the respective plugin, depending on the target device. The IE manages the plugin by itself, executing the network and balancing the inference requests among all the available devices. It always allocates the inference request in the least used device from the available ones. Additionally, it is possible to extract performance data through counters, like the inference time per layer among other metrics. Finally, the IE will produce an embedding per input image, which will serve to determine which person is. While the modular structure of the IE allows developers to deploy the model in many environments, the execution phase is carried out by the plugins of each family of devices. It is the plugin itself who decides how the inference requests are managed and distributed among the different cores and threads of each physical device. For example, the MYRIAD X has an optimal number of inference requests of 4. The developer is not the responsible to plan the distribution of the requests but the plugin itself. Although it is possible to configure OpenVINO to use more or less than 4 inference pointers, the developer loses control on the schedule of the inference work.

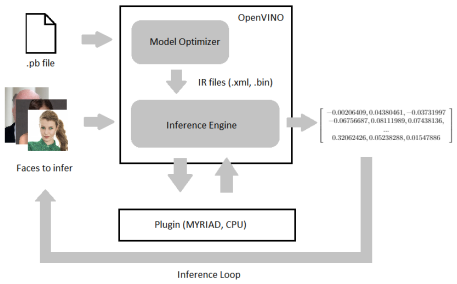


Figure 4: Software architecture of the system.

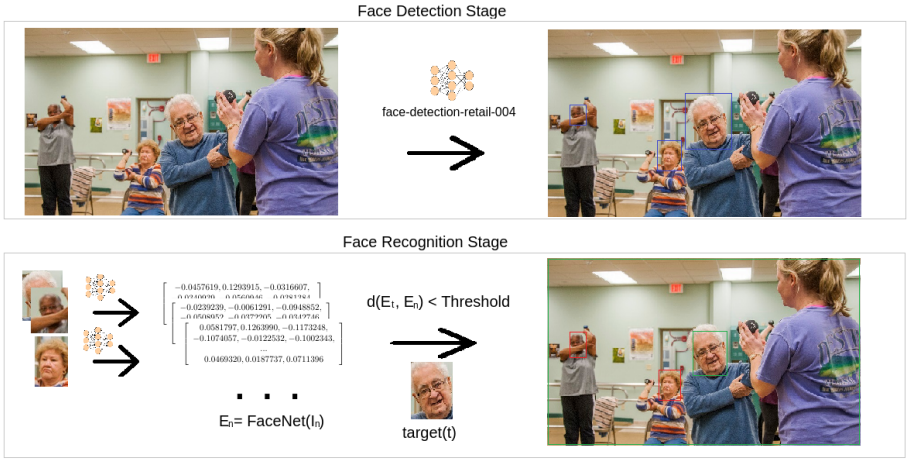


Figure 5: Example diagram of an average execution over a certain frame.

5. Application

Taking into account the architecture described in Section 4, an end-to-end application has been implemented. Given a video taken as a sequence of frames, this application performs face recognition. As depicted in figure 5, in first place the faces belonging to a frame are detected and second, every face is driven to FaceNet to identify to which person corresponds.

Algorithm 1 shows the pseudocode of the inference stage. The work in mainly divided into four stages: getting frames from input, detecting the faces within every frame, applying FaceNet to every detected face and finally showing the results. The final outcome of the application is the same input frame, but with

Algorithm 1 Pseudocode of the inference pipeline

```
frame = getFrameFromVideo(input)
frame = preprocessFrameForDetection(frame)
faces = detectFaces(cnn, frame)
if len(faces) > 0 then
    for all face in faces do
        faceImage = cropFace(face, frame)
        faceImage = preprocessFaceForRecognition(faceImage)
        embedding = recognizeFace(facenet, faceImage)
        if euclidDistance(embedding, targetEmbedding) < THRESHOLD
        then
            markFaceRecognized(face, frame, true)
        else
            markFaceRecognized(face, frame, false)
        end if
    end for
end if
showResults(frame)
```

the faces surrounded by bounding boxes marking their position and whether they are the target face or not.

It must be noted that the asynchronous API of the IE has been applied for performing the face recognition. FaceNet takes as input only one face per inference, that is, for an image with n faces we must dispatch n inferences through FaceNet. The asynchronous API allows to dispatch inferences without waiting for them to finish and enables the retrieval of results when they are necessary. Thus, it is possible to perform several face inferences in parallel on the NCS2 sticks.

The application has been developed using Python3.6. Two parallel execution methods have been implemented for inferences: multithreading and multiprocessing. They will be explained in detail in the following subsections.

5.0.1. Multithreading

In this method, a process is created for the inferences. Each inference task is then executed by a single thread. These processes only manage the inference loop, receiving an image, detecting the faces and applying face recognition on the faces previously found. Finally, the processes put the resulting image into a shared queue used by another process. In this model, no NCS2 is specifically allocated, so the workload of a single image can be shared across different sticks. Under the host point of view, this method is more lightweighted and should be the ideal one for deploying on devices with low computing power. However, it may be restricted due to the so-called Global Interpreter Locker (GIL) [45]. The pseudocode corresponding with the multithreading approach is shown in Algorithm 2.

The GIL was created in order to ensure the sequential execution of threads in the Python interpreter. This lock is needed as Python considers its own interpreter as a resource that must be sequentially accessed. Thus, only one thread per Python process is able to acquire the lock. This behaviour allows Python to improve the efficiency of programs running on single-core environments, because the GIL is unlocked whenever a thread waits for an I/O operation and

Algorithm 2 Pseudocode of the multithreading approach

```
createInputReaderProcess()
numDevices = getNumDevices()
for i in range(0, numDevices) do
    thread = createInferenceThread(i)
    launchThread(thread)
end for
createProcessingResultsProcess()
```

Algorithm 3 Pseudocode of the multiprocessing approach

```
createInputReaderProcess()
devices = getInferenceDevices()
for all device in devices do
    createInferenceProcess(device)
end for
createProcessingResultsProcess()
```

can be optimized through time slicing techniques. Therefore, considering the interpreter as the bottleneck in multi-core devices we propose a multiprocessing approach employing the same strategy. This model will create various processes, each with its own Python interpreter instance, allowing to bypass the effects of the GIL despite of the increase of resources usage.

5.0.2. Multiprocessing Method

This method implies that we are creating various inference processes, one per NCS2 employed to execute the inference. Ultimately, these processes do the same work as the threads above but, in this case, the NCS2 devices cannot be shared among the processes. This method is supposed to be more resource-hungry for the host, but a higher performance is expected. Each process is allocated within each NCS2 individually, so the sticks are not able to share the workload of the same image. The pseudocode corresponding with this approach is shown in Algorithm 3.

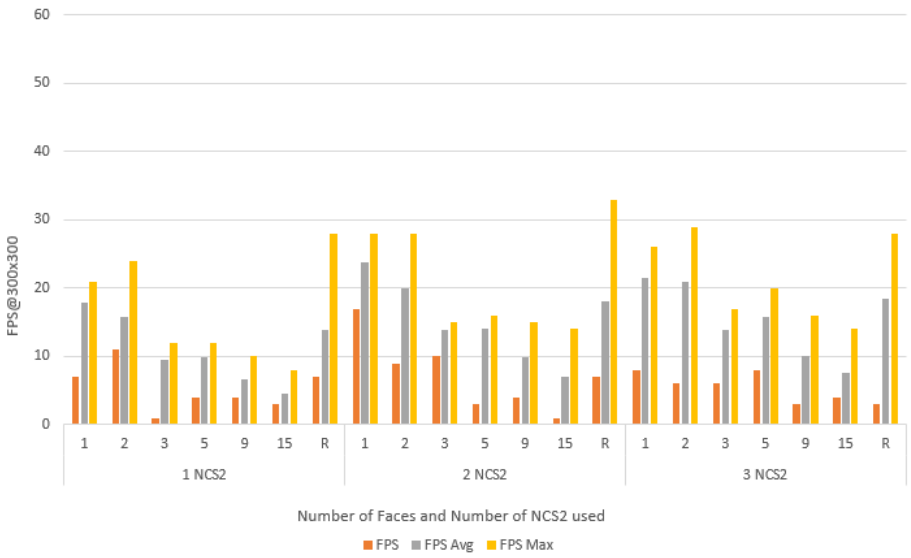


Figure 6: Performance with the Multithreading model varying the number of mounted NCS2 sticks (general purpose host).

A more detailed graphic comparison is shown in figure 7. In order to assure real-time constraints, we take a sample from the video only when a process or a thread have finished inferring the previous image and can take that sample immediately.

6. Experiments

In this section we present our experiments employing the system described in Section 4. For the sake of fairness, the CPU frequency was set to the maximum value to achieve the highest performance.

In order to evaluate the performance, several 299x299 pixel videos accesible using this URL from Google Drive [46] have been processed. The following metrics have been calculated, namely: the minimum, maximum and average frames per second (FPS); the minimum, maximum and average inference time; the average number of faces in relation to the FPS; and the total execution times and number of calculated inferences. Moreover, power figures will also

be shown. These consumption metrics have been acquired using the INA260 as explained in Section 4.

6.1. Results on a general purpose station host

In this first experiment, several configurations of the cluster have been tested depending on the number of mounted NCS2 sticks: 1, 2 and 3. Moreover, the number of faces appearing on videos have been considered in this study, employing videos that contain: 1, 2, 5, 9, 15 and a random number of faces (labeled as R). It must be noted that all these cases the host is the i7-based laptop.

Figure 6 contains the FPS data when employing the Multithreading model. As expected, FPS slow down when the number of faces increases inside the videos. It is noticeable that number of NCS2 sticks hardly report speedup. In order to analyze this issue deeply, the inference times have been measured as well, as shown in Table 3. According to the results, we observe that more NCS2 sticks hardly varies the computation times related with each inference in the accelerators, so the loss of performance is due to the aforementioned GIL or the internal scheduler of OpenVINO. OpenVINO scheduling policy consists in allocating the inference request to the available device possessing the lowest workload. However, the OpenVINO scheduling bottleneck has to be discarded, as with the Multiprocessing model the results are much better (see in Figure 8).

The Multiprocessing results are summarized in Figure 8. As this figure depicts, the application scales reasonably well when increasing the number of NCS2 sticks. For example, when considering the case of just one face, the maximum performance achieved is 20 FPS@299x299, 40 and close to 58, with 1, 2 and 3 NCS2 sticks, respectively. This effect is also observed considering the average number of FPS (FPS Avg in the figure), so the system built is able to provide a higher performance in the multi-face scenario.

Table 4 confirms this last aspect, showing the speedup results obtained for the Multiprocessing model. The speedups are close to their ideal value for 2 and 3 NCS2 sticks in most of cases. It must be noted that the best accelerations are

Table 3: Inference times in the random case execution, using the Multithreading model and with 1, 2 and 3 NCS2 sticks.

1 NCS2	Total number of inferences: 2279 Min Inference Time: 14.17 ms Avg Inference Time: 20.16 ms Max Inference Time: 25.41 ms
2 NCS2	Total number of inferences: 2279 Min Inference Time: 14.17 ms Avg Inference Time: 19.91 ms Max Inference Time: 24.78 ms
3 NCS2	Total number of inferences: 2279 Min Inference Time: 14.22 ms Avg Inference Time: 19.84 ms Max Inference Time: 25.90 ms

achieved when the number of faces increases, which means that more inferences are processed. The proposed implementation not only presents good scalability rates in a multi-stick system, but also behaves especially well in high demand environments.

Figure 9 shows the average temporal evolution of FPS and faces for the random case on Multiprocessing model. As can be seen, the loss of performance due to the extra computation with more faces in the scene is compensated by far with the use of the multi-stick solution. We highlight the worst case seen using 3-NCS2 is always above 25 FPS@299x299. In order words, the use of several

accelerators not only translates into higher peak performance, but also allows to support loads enhancing system resilience.

6.2. Results on an embedded platform

In this subsection, a similar study is carried out, but considering a Raspberry Pi 3B+ and Raspberry Pi 4 as the hosts of our system.

Unlike the previous section we have only considered the experimentation for the Multiprocessing model. Figure 10 shows the FPS for 1, 2 and 3 NCS2 sticks in combination with the Raspberry Pi 3B+ and considering a range of faces in the scene. As can be seen, although there exists some speedups for several sticks, scalability rates differ from observed in the i7 host.

The performance boost is not so pronounced as in the prior case. In fact, there is not a big improvement when increasing from 2 to 3 the number of NCS2 sticks. Nevertheless, Raspberry Pi 3B+ is low-power platform. Even though the FPS@299x299 performance is lower than the frame rates achieved in the previous experiments as seen in Figure 10, the power consumption of the Raspberry Pi 3B+-based architecture have been considerably reduced.

In order to obtain the power consumption, the INA-based system was utilized, as described in Section 4. The measured power consumption is shown in Figure 11 for the three configurations of the cluster. Table 5 contains the minimum, maximum and the Root Mean Square (P_{RMS}) values of the power figures obtained through the measurement system. As observed, there is an approximate increase of 1.2 Watt per stick. Table 6 shows the power consumption of the whole system then. As can be seen, the upper bound is around 10W when employing the 3-NCS2 sticks configuration.

Taking into account the results extracted from the Raspberry Pi 3B+, another embedded board has been tested as well. Unlike its predecessor, the Raspberry Pi 4 provides USB3.0 connections, while it is a bit more power hungry. The performance results are shown in Figure 12. Using version 4 Raspberry has almost doubled the average FPS in multi-face environments. As in the previous board, there is no significant speedup when scaling from 2 to 3 NCS2 sticks.

However, it is important to note that Raspberry 4 with 2 NCS2 have reached almost the same performance rate that the isolated PC based i7 with much less power requirement and and with at least a third of its overall cost. Despite still far from the video real-time conditions, processing on average 6 FPS (around 1 frame per 166 ms) in multi-face scenarios is enough for the targeted environment, as an hospital or governmental installations.

Finally, the power consumption has been measured as well as in the case of the Raspberry Pi 3B+-based system, and it never surpasses 11.2W, which is also a very low power consumption.

6.3. Efficiency comparison

Comparing general purpose hosts with embedded platforms is a difficult task, as they differ not only in features, but in power and cost. This section addresses this problem by comparing the efficiency of the previously studied systems in terms of performance/power ratio. These results are shown in Table 7. It must be noted that in the case of the PC, the power consumption of the i7 is 37W, but the maximum consumption of the device is 57.9W according to the 3DMark06 benchmark[47]. These results have been extracted from this full benchmark of the device [48].

As can be observed, the Raspberry Pi 3B+-based cluster and the PC-based cluster are close in terms of FPS/Watt. It is noteworthy to see how the former is better in mono-face scenarios, while the latter gets better efficiency figures in multi-face environments. In any case, the Raspberry Pi 4 surpasses both of them in most of the cases. Finally, it is noteworthy to mention that the cluster with three NCS2 sticks and the Raspberry Pi 4 host achieves a 3.3X improvement in terms of FPS/Watt with respect to the PC-based solution in multi-faced scenarios.

7. Conclusion

In this paper, an NCS2 cluster has been proposed to accelerate the face recognition task to address security issues in non-restrained scenarios.

As has been shown, this cluster allows mitigating the loss of performance in multi-face scenarios, which is critical in situations as the COVID-19 pandemic. The multiprocessing model has suited better than the multithreading one to the features of this cluster. All in all, the cluster provides a low-power and low-cost solution to being deployed in the aforementioned scenarios, specially when combined with the Raspberry Pi 4 host, achieving an average performance of 6 FPS and improving more than three times the energy efficiency of the baseline solution.

As future work, it would be interesting to explore different implementations to overcome the limitations imposed by the Python GIL.

Acknowledgment

This paper has been supported by the EU (FEDER) and the Spanish MINECO and CM under grants S2018/TCS-4423, RTI2018-093684-B-I00 and TIN 2015-65277-R, as well as the UCM-Banco Santander Grant PR26- 16/20B-1. Besides, UCM - Innova docentia competitive Projects: 2018-2019 (Project 315, acronym Reconnet) and 2019-2020 (Project 205, acronym Transform) have supported this paper as well.

References

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, 2012, pp. 1097–1105.

[2] Y. LeCun, Y. Bengio, G. E. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444. doi:10.1038/nature14539.
URL https://doi.org/10.1038/nature14539

[3] Y. Deng, Deep learning on mobile devices: a review, in: Mobile Multimedia/Image Processing, Security, and Applications 2019, Vol. 10993,

International Society for Optics and Photonics, SPIE, 2019, pp. 52 – 66. doi:10.1117/12.2518469.

[4] K. Heath, L. Guibas, Facenet: Tracking people and acquiring canonical face images in a wireless camera sensor network, in: 2007 First ACM/IEEE International Conference on Distributed Smart Cameras, 2007, pp. 117–124. doi:10.1109/ICDSC.2007.4357514.

[5] M. Wang, W. Deng, Deep face recognition: A survey, ArXiv abs/1804.06655.

[6] M. Turk, A. Pentland, Eigenfaces for recognition, Journal of Cognitive Neuroscience 3 (1) (1991) 71–86, pMID: 23964806. arXiv:https://doi.org/10.1162/jocn.1991.3.1.71, doi:10.1162/jocn.1991.3.1.71.
URL https://doi.org/10.1162/jocn.1991.3.1.71

[7] A. Corpas, L. Costero, G. Botella, F. D. Igual, C. Garca, M. Rodrguez, Acceleration and energy consumption optimization in cascading classifiers for face detection on low-cost arm big. little asymmetric architectures, International Journal of Circuit Theory and Applications 46 (9) (2018) 1756–1776.

[8] F. Schroff, D. Kalenichenko, J. Philbin, Facenet: A unified embedding for face recognition and clustering, CoRR abs/1503.03832.

[9] M. S. Kim, A. A. D. Barrio, L. T. Oliveira, R. Hermida, N. Bagherzadeh, Efficient mitchell's approximate log multipliers for convolutional neural networks, IEEE Trans. Computers 68 (5) (2019) 660–675.

[10] G. B. Huang, M. Ramesh, T. Berg, E. Learned-Miller, Labeled faces in the wild: A database for studying face recognition in unconstrained environments, Tech. Rep. 07-49, University of Massachusetts, Amherst (October 2007).

[11] D. G. Fernández, A. A. D. Barrio, G. B. Juan, C. García, M. Prieto, R. Hermida, Complexity reduction in the HEVC/H265 standard based on smooth region classification, Digital Signal Processing 73 (2018) 24–39.

[12] D. G. Fernández, G. Botella, A. A. Del Barrio, C. García, M. Prieto-Matías, C. Grecos, Hevc optimization based on human perception for real-time environments, Multimedia Tools and Applications doi:10.1007/s11042-018-7033-y.

[13] R. Ding, Z. Liu, R. D. S. Blanton, D. Marculescu, Lightening the load with highly accurate storage- and energy-efficient lightnns, ACM Trans. Reconfigurable Technol. Syst. 11 (3) (2018) 17:1–17:24. doi:10.1145/3270689.
URL http://doi.acm.org/10.1145/3270689

[14] M. S. Kim, A. A. D. Barrio, R. Hermida, N. Bagherzadeh, Low-power implementation of mitchell's approximate logarithmic multiplication for convolutional neural networks, in: 23rd Asia and South Pacific Design Automation Conference, ASP-DAC 2018, Jeju, Korea (South), January 22-25, 2018, 2018, pp. 617–622.

[15] Intel® neural compute stick 2, https://software.intel.com/en-us/neural-compute-stick, accessed: 29-08-2019.

[16] Intel® distribution of openvino™toolkit, https://software.intel.com/en-us/openvino-toolkit, accessed: 19-09-2019.

[17] G. LLC, Tensorflow: An end-to-end open source machine learning platform, https://www.tensorflow.org/, accessed: 19-09-2019.

[18] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9. doi:10.1109/CVPR.2015.7298594.

[19] C. Szegedy, S. Ioffe, V. Vanhoucke, Inception-v4, inception-resnet and the impact of residual connections on learning, CoRR abs/1602.07261. arXiv:1602.07261.
URL http://arxiv.org/abs/1602.07261

[20] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet of Things Journal 3 (5) (2016) 637–646. doi:10.1109/JIOT.2016.2579198.

[21] Intel® movidius™neural compute stick, https://movidius.github.io/ncsdk/ncs.html, accessed: 29-08-2019.

[22] I. Corporation, Intel® Movidius™Neural Compute Stick, https://software.intel.com/en-us/neural-compute-stick, [Online; accessed 25-October-2019] (2019).

[23] Nvidia, Embedded Systems for Next-Generation Autonomous Machines, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/, [Online; accessed 25-October-2019] (2019).

[24] G. LLC, Coral, https://coral.withgoogle.com/, [Online; accessed 25-October-2019] (2019).

[25] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris, N. D. Lane, Embench: Quantifying performance variations of deep neural networks across modern commodity devices, in: The 3rd International Workshop on Deep Learning for Mobile Systems and Applications, 2019, pp. 1–6.

[26] E. Kristiani, C.-T. Yang, C.-Y. Huang, isec: An optimized deep learning model for image classification on edge computing, IEEE Access 8 (2020) 27267–27276.

[27] N. A. OTHMAN, I. AYDIN, A new deep learning application based on movidius ncs for embedded object detection and recognition, in: 2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), 2018, pp. 1–5. doi:10.1109/ISMSIT.2018.8567306.

[28] G. Mathew, S. Sindhu Ramachandran, V. S. Suchithra, Lung nodule detection from low dose ct scan using optimization on intel xeon and core processors with intel distribution of openvino toolkit, in: TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), 2019, pp. 1783–1788.

[29] G. Dinelli, G. Meoni, E. Rapuano, G. Benelli, L. Fanucci, An fpga-based hardware accelerator for cnns using on-chip memories only: Design and benchmarking with intel movidius neural compute stick, International Journal of Reconfigurable Computing 2019.

[30] Z. Lin, M. Yih, J. M. Ota, J. D. Owens, P. Muyan-zelik, Benchmarking deep learning frameworks and investigating fpga deployment for traffic sign classification and detection, IEEE Transactions on Intelligent Vehicles 4 (3) (2019) 385–395. doi:10.1109/TIV.2019.2919458.

[31] N. Corporation, Technical Brief. NVIDIA Jetson TK1 Development Kit, Tech. rep., NVidia Corporation (2014).

[32] B. Amos, B. Ludwiczuk, M. Satyanarayanan, Openface: A general-purpose face recognition library with mobile applications, Tech. rep., CMU-CS-16-118, CMU School of Computer Science (2016).

[33] E. Jose, G. M., M. H. T. P., S. M. H., Face recognition based surveillance system using facenet and mtcnn on jetson tx2, in: 2019 5th International Conference on Advanced Computing Communication Systems (ICACCS), 2019, pp. 608–613. doi:10.1109/ICACCS.2019.8728466.

[34] K. Zhang, Z. Zhang, Z. Li, Y. Qiao, Joint face detection and alignment using multitask cascaded convolutional networks, IEEE Signal Processing Letters 23 (10) (2016) 1499–1503. doi:10.1109/LSP.2016.2603342.

[35] H. A. Mohamed Salah Salhi, Atef Alaaeddine Sarraj, Implementation of an evolutionary facial recognition algorithm on jetson tk 1, Journal of Multimedia Processing and Technologies 10 (1). doi:10.6025/jmpt/2019/10/1/1-17.

[36] Jetson store, https://www.nvidia.com/en-us/autonomous-machines/jetson-store/, accessed: 15-10-2019.

[37] Z. Wang, Z. Cheng, H. Huang, X. Zhou, Y. Liu, Design and implementation of vehicle unlocking system based on face recognition, in: 2019 34rd Youth

Academic Annual Conference of Chinese Association of Automation (YAC), 2019, pp. 121–126. doi:10.1109/YAC.2019.8787608.

[38] A. Boka, B. Morris, Person recognition for access logging, in: 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), 2019, pp. 0933–0936. doi:10.1109/CCWC.2019.8666483.

[39] Intel® movidius™neural compute sdk, https://movidius.github.io/ncsdk/, accessed: 15-10-2019.

[40] Y. Xie, L. Ding, A. Zhou, G. Chen, An optimized face recognition for edge computing, in: 2019 IEEE 13th International Conference on ASIC (ASICON), 2019, pp. 1–4.

[41] N. Smith, Transitioning from intel® movidius™neural compute sdk to intel© distribution of openvino™toolkit, https://software.intel.com/en-us/articles/transitioning-from-intel-movidius-neural-compute-sdk-to-openvino-toolkit, accessed: 24-09-2019 (2019).

[42] A. A. D. Barrio, N. Bagherzadeh, R. Hermida, Ultra-low-power adder stage design for exascale floating point units, ACM Trans. Embedded Comput. Syst. 13 (3s) (2014) 105:1–105:24.

[43] D. Sandberg, facenet, https://github.com/davidsandberg/facenet, accessed: 14-11-2019 (2018).

[44] S. Yang, P. Luo, C. C. Loy, X. Tang, Wider face: A face detection benchmark, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[45] Thread state and the global interpreter lock, https://docs.python.org/3/c-api/init.html#thread-state-and-the-global-interpreter-lock, accessed: 15-10-2019.

[46] Video repository, https://drive.google.com/open?id=1cWwSCtgch9g1m5C7IJ6Jv5PASoOp1fDq, accessed: 15-11-2019.

[47] https://benchmarks.ul.com/legacy-benchmarks, https://benchmarks.ul.com/legacy-benchmarks, accessed: 14-11-2019 (2019).

[48] Review lenovo ideapad z510 notebook, https://www.notebookcheck.net/Review-Lenovo-IdeaPad-Z510-Notebook.105627.0.html, accessed: 14-11-2019 (2013).

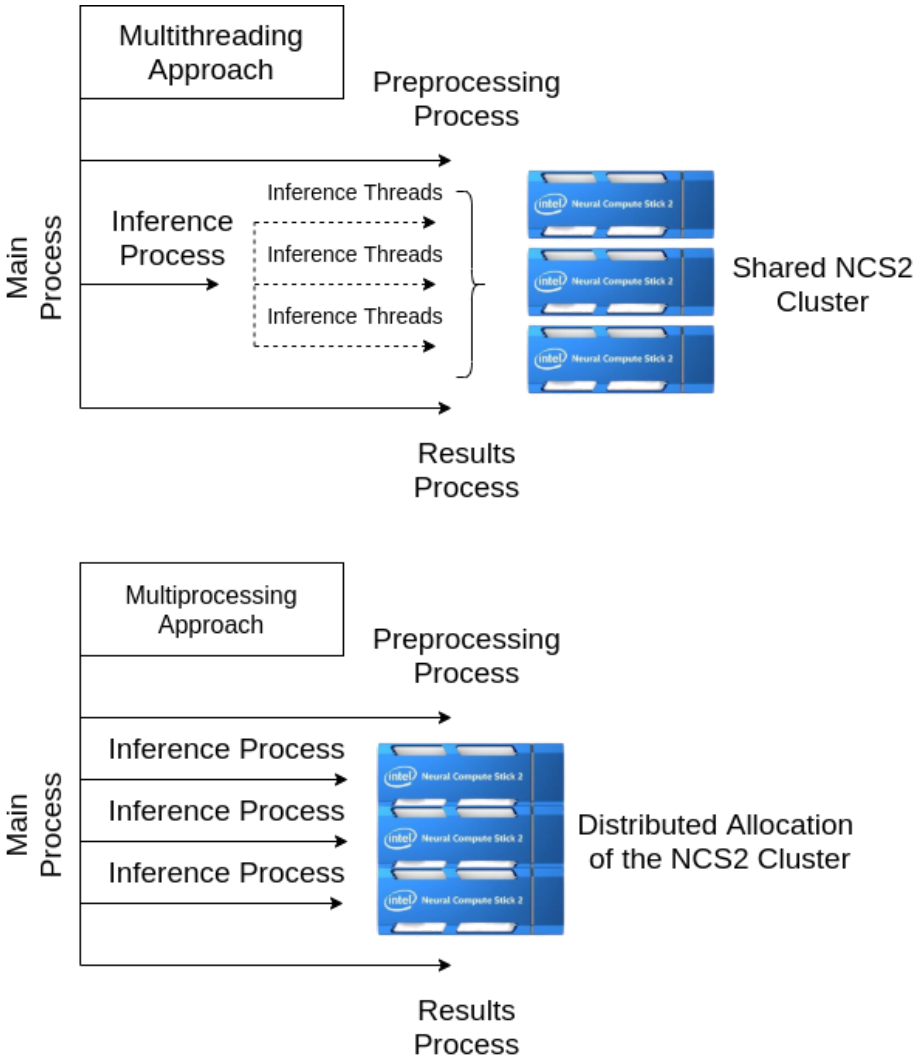


Figure 7: Graphical comparison between the parallel models used in the research.

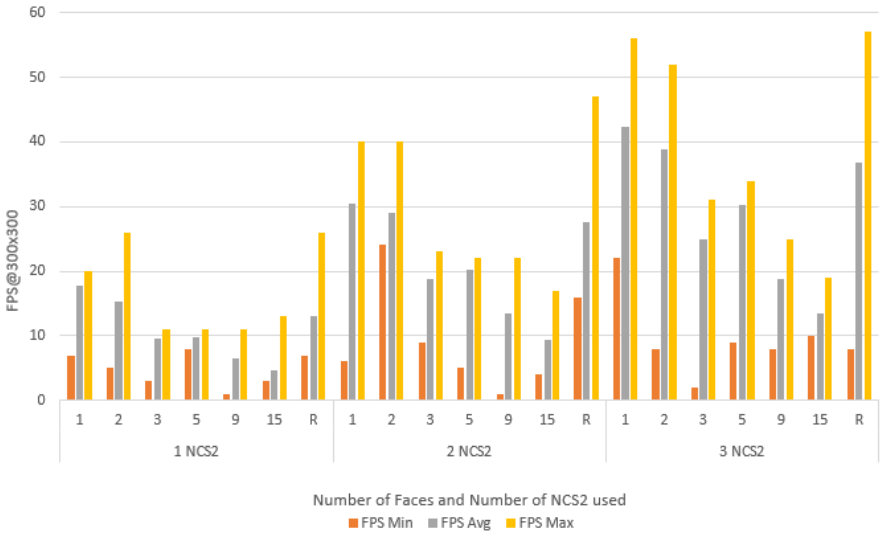
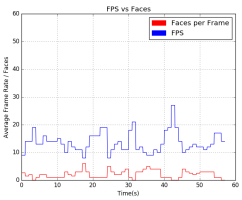
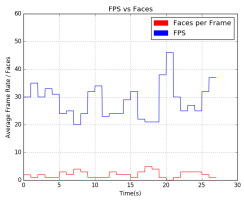


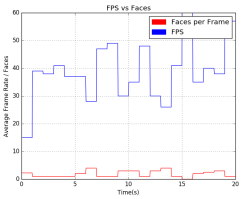
Figure 8: Performance with the Multiprocessing model varying the number of mounted NCS2 sticks (general purpose host).



(a) 1 NCS2



(b) 2 NCS2



(c) 3 NCS2

Figure 9: Faces and FPS@299x299 in the random case with different number of NCS2 sticks.

Table 4: Average FPS@299x299 and speedup employing the Multiprocessing model with respect to the number of faces per video and number of mounted NCS2 sticks.

N. NCS2	Faces	FPS	Speedup
2 NCS2	1	30.42	1.71
	2	29.16	1.89
	3	18.87	1.98
	5	20.16	2.00
	9	13.53	2.00
	15	9.38	1.99
	R	27.60	2.00
3 NCS2	1	42.40	2.38
	2	38.76	2.51
	3	25.00	2.63
	5	30.16	3.07
	9	18.87	2.91
	15	13.50	2.87
	R	36.80	2.80

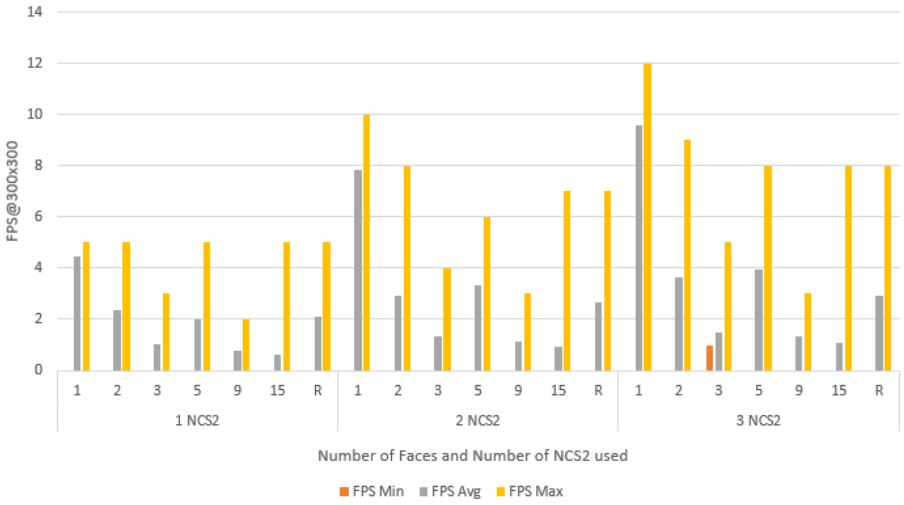


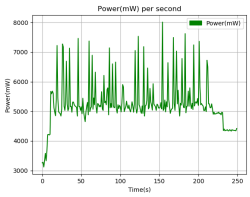
Figure 10: Performance with the Multiprocessing model varying the number of mounted NCS2 sticks and using the Raspberry Pi 3B+ as host.

Table 5: Power consumption values of the NCS2

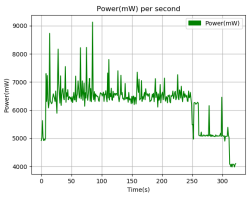
1 NCS2	Maximum: 2530 mW Minimum: 1100 mW P_{RMS}: 1712.16 mW
2 NCS2	Maximum: 4440 mW Minimum: 1560 mW P_{RMS}: 2975.24 mW
3 NCS2	Maximum: 5080 mW Minimum: 2410 mW P_{RMS}: 4215.40 mW

Table 6: Power consumption values of the NCS2 + Raspberry Pi 3B+

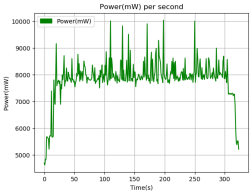
1 NCS2	Maximum: 8010 mW Minimum: 3120 mW P_{RMS}: 5302.10 mW
2 NCS2	Maximum: 9130 mW Minimum: 3980 mW P_{RMS}: 6255.43 mW
3 NCS2	Maximum: 10050 mW Minimum: 4630 mW P_{RMS}: 7883.98 mW



(a) 1 NCS2



(b) 2 NCS2



(c) 3 NCS2

Figure 11: Power consumption of the proposed system with 1, 2 and 3 NCS2 plus the Raspberry Pi 3B+.

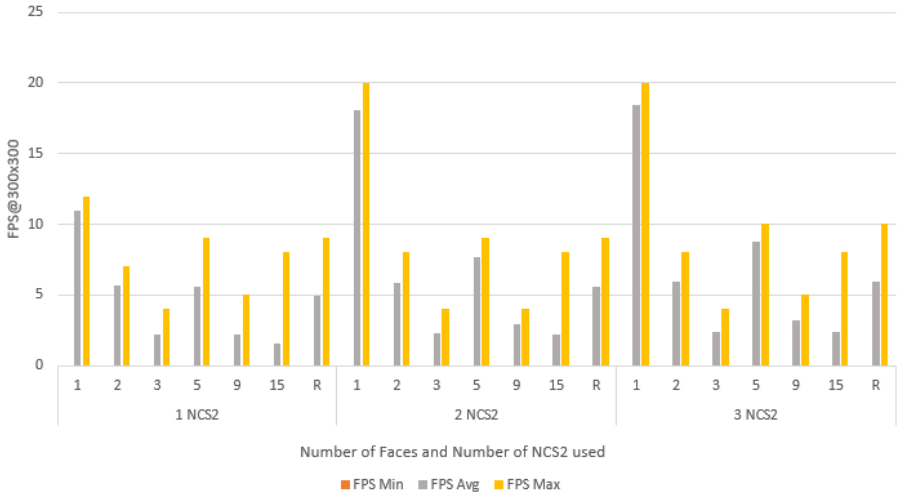


Figure 12: Performance with the Multiprocessing model varying the number of mounted NCS2 sticks and using the Raspberry Pi 4 as host.

Table 7: Ratio Performance/Power of the proposed system with different hosts and with respect to the number of NCS2 and faces per video.

Num. NCS2	Host	Max.Power(Watt)	Num. Faces	FPS@299x299	FPS/Watt
None	PC	57.9	1	21.84	0.37
			5	6.03	0.10
			15	1.78	0.030
			R	9.77	0.16
1	PC	59.1	1	17.83	0.30
			5	9.83	0.16
			15	4.71	0.07
			R	13.13	0.22
	Raspberry Pi 3B+	8.10	1	4.46	0.55
			5	2	0.24
			15	0.64	0.079
			R	2.11	0.26
	Raspberry Pi 4	8.8	1	11	1.25
			5	5.53	0.62
			15	1.51	0.18
			R	4.95	0.56
2	PC	60.3	1	30.42	0.50
			5	20.16	0.33
			15	9.38	0.15
			R	27.6	0.45
	Raspberry Pi 3B+	9.13	1	7.84	0.86
			5	3.97	0.43
			15	0.94	0.10
			R	2.64	0.28
	Raspberry Pi 4	10	1	18.06	1.80
			5	7.71	0.77
			15	2.16	0.21
			R	5.54	0.55
3	PC	61.5	1	42.4	0.68
			5	30.16	0.49
			15	13.5	0.21
			R	36.8	0.59
	Raspberry Pi 3B+	10.05	1	9.59	0.95
			5	3.97	0.39
			15	1.07	0.10
			R	2.92	0.29
	Raspberry Pi 4	11.2	1	18.4	1.64
			5	8.75	0.78
			15	2.39	0.41
			R	5.97	0.53