INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

January 23rd 2020

Innovative project

Eric Alata INSA
Thierry Monteil INSA
Ludovic Tancerel Wiifor

Project Report

Autonomous vision for image analysis and decision transmission

EyeMB Project

Arnaud Guiller guiller@etud.insa-toulouse.fr
Fu Yimin y_fu@etud.insa-toulouse.fr
Isabella Mi Hyun Kim mihyunki@etud.insa-toulouse.fr
Noël Taillardat ntaillar@etud.insa-toulouse.fr
Poonkuzhali Pajanissamy pajaniss@etud.insa-toulouse.fr

Institut National des Sciences Appliquées INSA
Département Génie Électrique et Informatique DGEI
Innovative & Smart Systems ISS
Year 2019-2020 19/20

Innovative project

January 23rd 2020

Eric Alata	INSA
Thierry Monteil	INSA
Ludovic Tancerel	Wiifor

Project Report

Autonomous vision for image analysis and decision transmission

EyeMB Project

Arnaud Guiller	guiller@etud.insa-toulouse.fr
Fu Yimin	y_fu@etud.insa-toulouse.fr
Isabella Mi Hyun Kim	mihyunki@etud.insa-toulouse.fr
Noël Taillardat	ntaillar@etud.insa-toulouse.fr
Poonkuzhali Pajanissamy	pajaniss@etud.insa-toulouse.fr

Institut National des Sciences Appliquées	INSA
Département Génie Électrique et Informatique	DGEI
Innovative & Smart Systems	ISS
Year 2019-2020	19/20

Table of content

Abstract

Bringing Artificial Intelligence (AI) at the edge of the Internet of Things (IoT) is part of the challenges that developers are going to face in the next years. In this project, we aimed at developing an IoT device, capable of capturing images, processing them on board, and send the deciphered data on an IoT network. The images to recognize were 7-segments LCD digits. The challenge we faced was to use a Neural Network and embed it on a specific, new microcontroller—the Gap8, from Greenwaves Technologies—designed for enabling AI in low-battery-powered IoT devices. We developed a Convolutional Neural Network (CNN) algorithm based on the well-known LeNet-5 architecture, and trained with TensorFlow a model of our own, based on images taken with a GAPuino board, embedding the Gap8 processor and connected to a low-power QVGA camera. On computer, the accuracy of our solution was proved on all our tests. On GAPuino board, we experimented and run some provided CNN, based on a model trained with the MNIST dataset (handwritten digits), and confirmed its accuracy. We then developed and run on GAPuino our own algorithm, capable of taking pictures, locating digits in the image, resizing them to fit to the trained model size, and run a MNIST-based CNN on each of the digits. The NNtool, a library provided by Greenwaves Technolgies enabling Keras and TensorFlow models to be imported on Gap8, is still in development, so at this point, we are not able to use our own LCD-based model in the embedded CNN. Communicating the results on The Thing Network through a LoRa module, as part of the lowest priorities, is still in progress.

1 Introduction

This project is part of our 5th year Innovative and Smart Systems (ISS) teaching program, as part of the Innovative project module. The idea is to put into practice our technical skills acquired during our time at INSA, as well as an approach to other aspects of our formation, such as communication, teamwork and project management.

Looking at needs in industry, we are now living in an era of transition from the third industrial revolution towards Industry 4.0. This new revolution represents a paradigm shift in the way objects communicate among themselves and how data is acquired and received. The main purpose of Industry 4.0 is to transform data into a digital form in a smart, interconnected environment in order to make data easier and faster to access. It is this context that starts our autonomous vision project.

This project consists in prototyping an autonomous embedded solution, capable of recognizing images and transmitting data. More specifically, it should be able to read machine indicators (digits, gauges), process the computing image on board and only transmit deciphered data, instead of transmitting whole images, since such transmissions does not meet IoT networks capacities, nor devices energy consumption possibilities. Here, we focused on LCD's digits recognition, following the specifications presented later in this report, and basing our solution on Neural Networks to recognize data. The digits read could represent temperature from real LCD screens, as from refrigerators of restaurants and food industries, if we take a use case example (see part below). The motivation behind this project is to propose an innovative and smart way to collect and transmit data from those indicators, to avoid food from being spoiled unnecessarily, in the case of our use case.

This project thus perfectly fits into our formation towards IoT, by proposing a connected device solution to resolve issues from our day to day lives—even if here, we think about Business-to-Business (B2B) applications, and do not refer to Business-to-Consumer (B2C) market, when talking about day to day life issues.

With this autonomous vision project, we had the opportunity to work in an innovative project with an actual client from the industry, Ludovic Tancerel. Client for this project, in the way he gives us the guidelines, project specifications, and will receive the deliverables of the project, M. Tancerel was as well as our tutor, helping us find ways and solutions to our problems.

2 Context: bringing AI at the edge

2.1 IoT Trends: to more and more AI

Today, devices being part of what we call the "Internet of Things" (IoT) are becoming more and more important and spreading each day a bit more. Sensors, actuators, they are usually small components enabling basic operations: getting an environment parameter, locating something… However, the actual trend in this field is to target some more valuable data.

Data is indeed at the heart of IoT objectives: retrieving data and making it valuable. A first challenge might be faced when developing massive devices networks: data is usually uploaded through IoT networks until data centers (we usually refer to "the cloud"), where they are processed. However, huge amount of devices might imply saturated networks, as well as saturated data centers. This explains a first meaning of "bringing AI at the edge": from cloud computing, deporting some of the data processing to gateways (fog computing), or even to the devices them-self (edge computing). Now, in a general way, providing IoT devices that can retrieve more than a movement, a temperature, or light—real *smart* devices—is also a need for more and more companies. Limited batteries, energy, memory and processing remain however a hardware challenge to solve in order to reach this goal—a challenge that some semiconductor companies have already started to solve.

Prototyping an IoT device consuming low energy and using Neural Network is the aim of our project.

2.2 A use case: our solution, a way to connect devices

The technology developed in this project can have many applications. The very aim in indeed to allow our solution to record information displayed by devices that are actually not connected and that cannot be, like old expansive machines that only allow to display their information, or devices based on analogic or physical treatment of data (water meters, gauges…). An example of a simple use case, presented to us by our client Wiifor, addresses the problems of storing food in large, industrial-sized refrigerators. Nowadays, many big food companies end up losing food, and consequently profit, due to a poor monitoring of the temperature of their equipment. Currently, most of these refrigerators simply indicate their temperature with no monitoring. When there is a malfunction during the night, no one is informed nor can prevent food from spoilage.

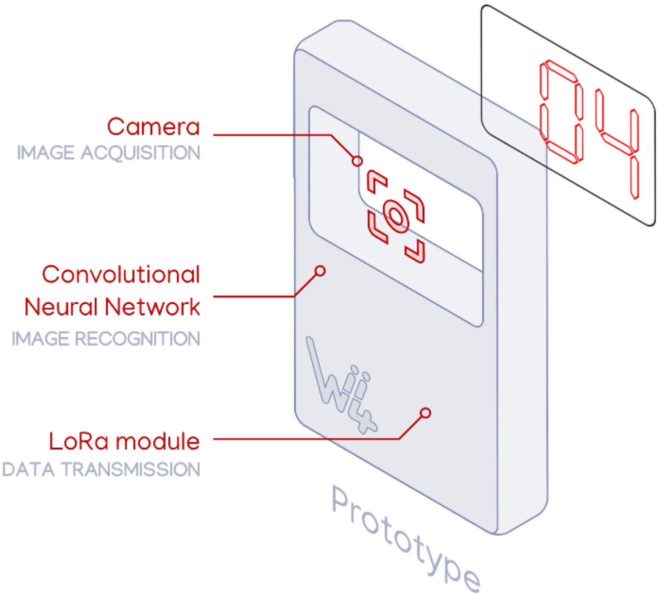
In the future, these refrigerators will most likely be connected, so restaurant managers will be able, first, to know in real time changes in the temperature, but also know if the temperature has risen above an authorized threshold and thus prevent the restaurant from serving contaminated food to their clients. Finally, they would also know if the refrigerator is not functioning correctly, allowing the restaurant to make any necessary decisions to save food, and consequently avoid unnecessary loss of both food and money.

Our solution could provide all the advantages of a connected solution—without requiring the purchase of a new, expensive refrigerator.

3 Objectives

3.1 Overview

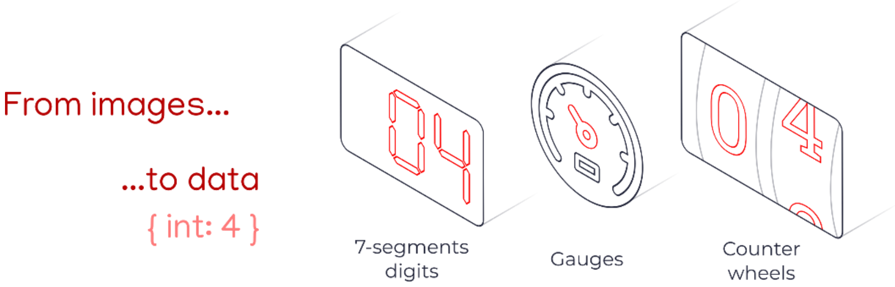
Our objective is to develop and present a functioning prototype that is capable of capturing images, interpreting from the image the correct value of the data displayed inf front of it, based on a Machine Learning approach (in the context of the use case defined above, the data would be the temperature of an industrial refrigerator), and transmitting the processed data in real time. As introduced in the context part, this means bringing computing to the edge of the Internet of Thing, since processing the data (in our case, images) directly on the device.



Overview of the objective: an IoT prototype

3.2 Specifications

The data to read shall be displayed on a 7–segment LCD screen. Possible other targets might be considered, like gauges, or counter wheels displays.



Specification: data to read

Our client has also requested that we implement our solution in such a way that the device is autonomous and energy efficient. That is why, for this project, we worked on a very recent embedded processor called Gap8. It is part of the specifications to explore it, define what kind of smart processing it is possible to achieve with it, and eventually, be able to evaluate the maturity of this recent solution. Based on documentation delivered by the company which designed it, this chip provides an energy performance that is a sixteen-times better than traditional boards on the market. Being pioneers on the use of this new board—launched in 2018—presents many risks since we are part of the first users to test its problems and limitations; at the same time, this became one of the main objectives of this project. We get a Gapuino development board (embedding the Gap8 processor), to complete this work (see Materials).

In addition, one challenge of this project is Machine Learning. One of the specifications of our client is to build our own digit database, to train a Neural Network and then integrate a custom trained algorithm into the embedded device, as a way to prove the possibility of a wide range of future other applications.

To meet those specifications, we had to work on many topics that were very new for us and to changing technology field. This is both a great challenge and a main objective as it allows each one of us to explore and learn more about new domains.

3.3 Deliverables

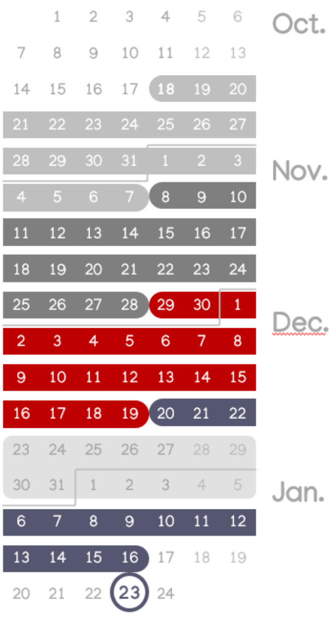
More than just a prototype, at the end of our project we shall be able to deliver:

- A bibliography about Neural Networks
- A database composed of LCD digits with pictures taken from our board
- A CNN model capable of recognizing LCD digits from this database
- A working prototype capable of capturing images, based on Gapuino board
- A guideline for Gapuino development, included in this report (see Material & Development)

5 Management

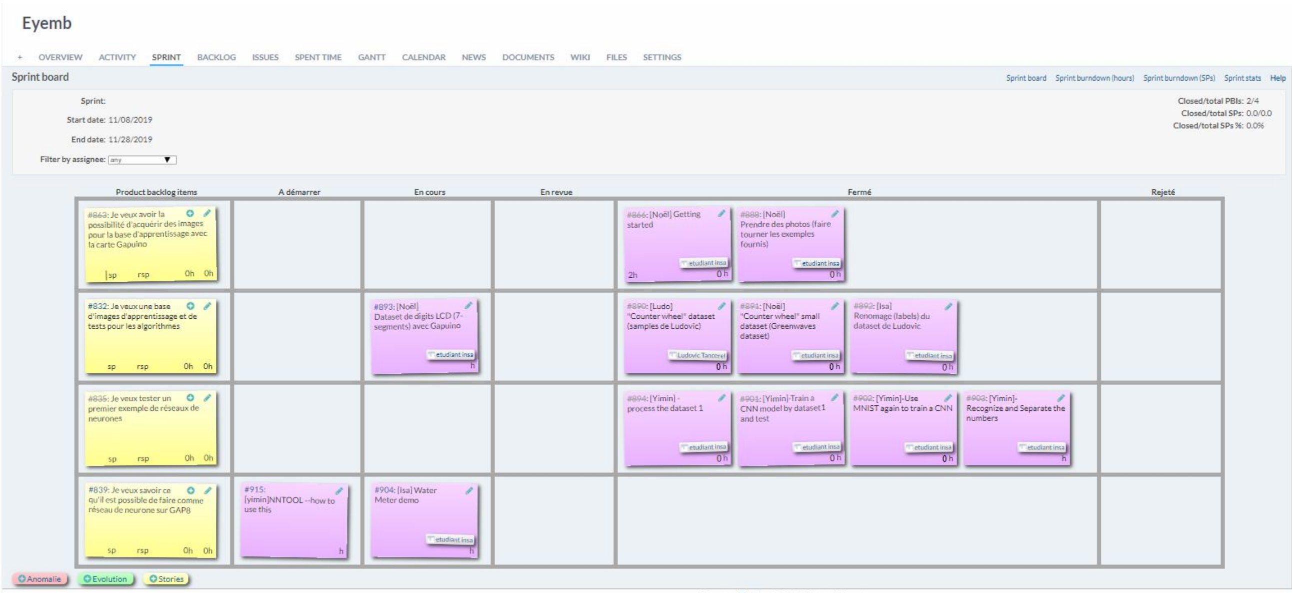
5.1 Agile method

During this project we adopted the Agile method, creating sprints and defining stories using the organization platform Redmine, which was provided us access by Wiifor. The idea of this method is to allow a more flexible development, establishing the next objectives and tasks based on the current advancement of the project, which perfectly suit for development and exploration works, like ours. At the end of each cycle ("sprint"), we should be able to have a small functioning part of what is the whole system that we aim to achieve in the end. The idea also is to keep track of how the work was divided between the group and how much time was spent on each task.



	1	2	3	4	5	6	Oct.
7	8	9	10	11	12	13	
14	15	16	17	18	19	20	
21	22	23	24	25	26	27	
28	29	30	31	1	2	3	Nov.
4	5	6	7	8	9	10	
11	12	13	14	15	16	17	
18	19	20	21	22	23	24	
25	26	27	28	29	30	1	Dec.
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	
23	24	25	26	27	28	29	
30	31	1	2	3	4	5	Jan.
6	7	8	9	10	11	12	
13	14	15	16	17	18	19	
20	21	22	23	24			

In the figure below we show an example of some stories defined in the first sprint for the month of November that included the caption of images with the board, the construction of a database, testing a neural network and studying the possibilities of neural networks with the GAP8.



Eyemb

Sprint board

Sprint:
Start date: 11/08/2019
End date: 11/28/2019
Filter by assignee: any ▼

Closed/total PBIs: 2/4
Closed/total SPs: 0.0/0.0
Closed/total SPs %: 0.0%

Product backlog items	A démarrer	En cours	En revue	Fermé	Rejeté
~~#863~~: Je veux avoir la possibilité d'acquérir des images pour la base d'apprentissage avec la carte Gapuino sp rsp 0h 0h				~~#866~~: [Noël] Getting started etudiant insa 2h 0 h ~~#888~~: [Noël] Prendre des photos (faire tourner les exemples fournis) etudiant insa 0 h	
#832: Je veux une base d'images d'apprentissage et de tests pour les algorithmes sp rsp 0h 0h		#893: [Noël] Dataset de digits LCD (7-segments) avec Gapuino etudiant insa h		~~#890~~: [Ludo] "Counter wheel" dataset (samples de Ludovic) Ludovic Tancerel 0 h ~~#891~~: [Noël] "Counter wheel" small dataset (Greenwaves dataset) etudiant insa 0 h ~~#892~~: [Isa] Renomage (labels) du dataset de Ludovic etudiant insa 0 h	
~~#835~~: Je veux tester un premier exemple de réseaux de neurones sp rsp 0h 0h				~~#894~~: [Yimin] - process the dataset 1 etudiant insa 0 h ~~#901~~: [Yimin]-Train a CNN model by dataset1 and test etudiant insa 0 h ~~#902~~: [Yimin]-Use MNIST again to train a CNN etudiant insa 0 h ~~#903~~: [Yimin]-Recognize and Separate the numbers etudiant insa h	
#839: Je veux savoir ce qu'il est possible de faire comme réseau de neurone sur GAP8 sp rsp 0h 0h	#915: [yimin]NNTOOL --how to use this h	#904: [Isa] Water Meter demo etudiant insa h			

Stories of the first sprint done in the Redmine platform

The stories of each sprint were defined in regular meetings we maintained with our client. We were able to meet each 2 to 3 weeks to discuss about our progress, difficulties and define our next steps. The main objectives and an idea of the main stories of each of the four sprints are presented below:

- 1st sprint: Getting started with CNN
  Build a bibliography on CNN; Run a first CNN on computer (given dataset)

- 2nd sprint: Getting started with Gapuino
  Start with GAPuino; Enabling taking pictures; Run CNN with custom datasets;
  Work on STM (see *Risk management* below)

- 3rd sprint: Towards a first CNN on Gapuino
  Build our dataset from GAPuino camera; Run a first CNN on GAPuino;
  Finalize CNN with custom models; Learn how to embed custom models on GAPuino

- 4th sprint: Towards complete, working prototype
  Use NNtool to import our trained model on GAPuino (this part is still in work); test and valid the model on computer; Process image on board (locate, resize);

5.2 Risk and risk management

Our project presented a big risk due to the technology we decided to use for the development. We made the decision to experiment with a new microcontroller technology, the GAP8, from a young French company called Greenwaves Technologies that was founded in 2014.

The reason we decided to take this risk is because it was a suggestion from our client and also because the GAP8 board has some very particular specifications that make it different from traditional microcontrollers (that we already had experience with, such as the STM32). This gave us an opportunity of learning from a new hardware.

The GAPuino is a board specifically constructed to address new technology trends towards edge computing and IoT. It allows its users to implement direct neural networks models (training and evaluation) directly on the board. The extract from Greenwaves Technologies website demonstrates exactly the great potential of this technology and how attractive it is to this new market that is growing:

> "The GAP8 IoT application processor enables the cost-effective development, deployment and autonomous operation of intelligent devices that capture, analyze, classify and act on the fusion of rich data sources such as images, sounds, radar, infra-red or vibrations for years on a battery. GAP8 delivers the computational and power agility necessary to integrate sophisticated signal processing and artificial intelligence into small, battery operated devices living on the very edge of the network."

We are currently working with the third version of the GAP8 which was officially released on 16th October 2019, implying recent hardware and software modifications. This means we are being pioneers in the use of this technology and also that there aren't many forums nor experts we could exchange with in case of problems or questions. Through our project development process, we found out we were the first ones testing this specific release, which still had bugs unknown or not documented by the developers of the product.

We really started to understand this risk and what it could imply in Sprint 2, after days spent trying to only install the SDK and then trying to take a simple picture, with the camera. To address this risk, we assigned there a member of the group to a risk management task: explore other alternatives to make the Neuronal Network computation and the IoT connectivity on board, like the STM32.AiCube. Whatsoever, as GAP8 started to present good results (functioning capture of images and example demos) and due our limited workforce, we decided to come back to focus our efforts totally on GAP8 during Sprint 3.

However, further on we started to encounter some instabilities with the tool. From the beginning of our work in October, the software development kit was updated more than three times by the developers due to bug corrections. An example of a major correction, we can site that while we are writing this report, the whole communication bridge is being transferred from PLPbridge to OpenOCD. That implicates in changes in almost all the basic configuration files.

Besides that, it appears we were the first ones to find a problem of incompatibility between the autotiler (tool responsible for the translation of codes into Gap8 format) and the CNN generators in SDK release 3.0 that made it not possible to run the example neural networks model on the board. This incompatibility found out in beginning January was fixed in the new release of the SDK 3.1. But it still has some bugs that are undergoing corrections.

5.3 Repartition of the tasks in the project

At the beginning of the project every member of the group contributed to constructing a bibliography on Neural Network. After that, the work was split accordingly to what we considered the biggest difficulties and what required more hours of work.

Since the GAP8 presented some big challenges on establishing the development environment and running/debugging programs that were too dependent on operating system (the SDK was made to work a specific Virtual Machine), so two people from the group were allocated to this task completely focusing on the GAP8.

Also, because of we faced the problem that the SDK was too dependent on the operating system, we decided to work in a solution for this by allocating one member of the group to constructing a docker environment.

For the Neural Network part, the member of the group that had already experience in this domain was in charge of this part.

6 **Materials**

or what we used, and **why**

6.1 Hardware

For the development of our project, we used the following hardware components:

- a Gapuino development board, from Greenwaves Technologies
- a QVGA Low Power Camera (part of a developer kit with the Gapuino board)
- an STM32 Discovery IoT Node board
- a RN2483 LoRa™ Technology Module from Microchip, and a LoRa™ Mote demonstration board embedding it

GAPuino development board

GAP8 ultra-low power IoT Application Processor Chip

Low Power QVGA Image Sensor

QVGA resolution

Black and white sensor

STM32 Discovery IoT Node

For IoT development

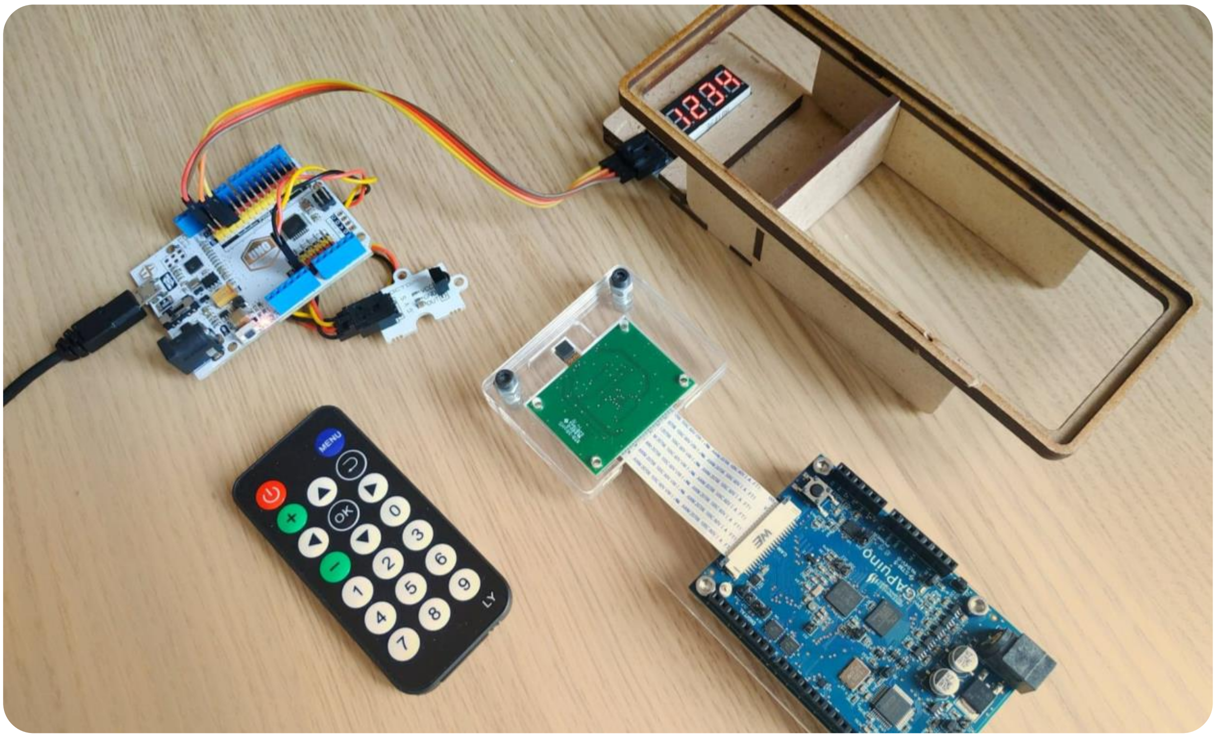
Compatible with STM32Cube.Ai

LoRa mote

Development board

Embedding RN2483 Technology module

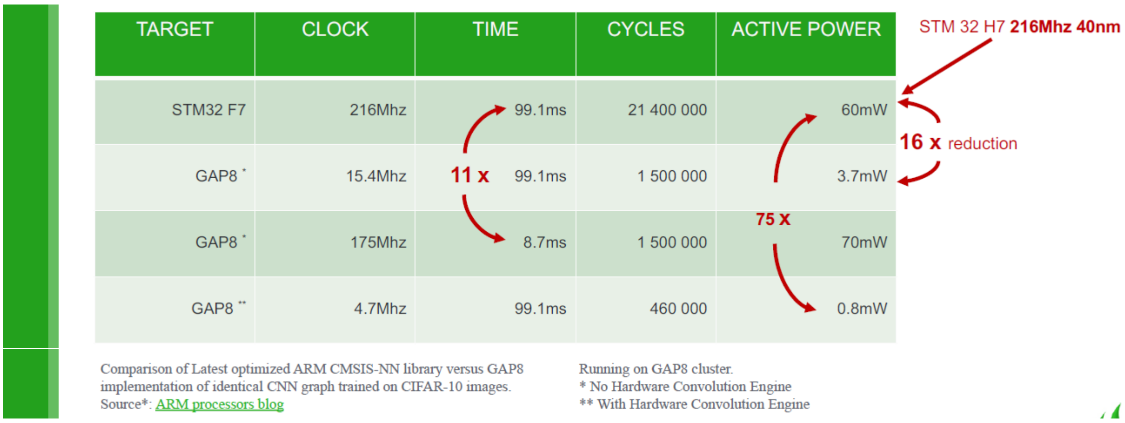
On top of those main components, we used a simple TM1637 LCD display for tests, debugs and demonstration, controlled with an ESP8266 or an Arduino UNO (digits could be controlled by an Infrared Remote and receiver).



Overview of most of our hardware materials for the project

6.1.1 GAPuino: a Gap8-based development board

The GAP8 is an IoT application processor that enables the deployment of artificial intelligence solutions at the very edge. GAP8 delivers sufficient computational and power agility necessary to capture, analyze and act on rich sources of data such as images, sounds and vibrations with ultra-low power operation cost compared to other similar solutions in the market as shown on the comparison below, extracted from a presentation of its product by Greenwaves Technologies:



TARGET	CLOCK	TIME	CYCLES	ACTIVE POWER
STM32 F7	216Mhz	99.1ms	21 400 000	60mW
GAP8 *	15.4Mhz	99.1ms	1 500 000	3.7mW
GAP8 *	175Mhz	8.7ms	1 500 000	70mW
GAP8 **	4.7Mhz	99.1ms	460 000	0.8mW

STM 32 H7 216Mhz 40nm
16 x reduction
11 x
75 x

Comparison of Latest optimized ARM CMSIS-NN library versus GAP8 implementation of identical CNN graph trained on CIFAR-10 images.
Source*: ARM processors blog

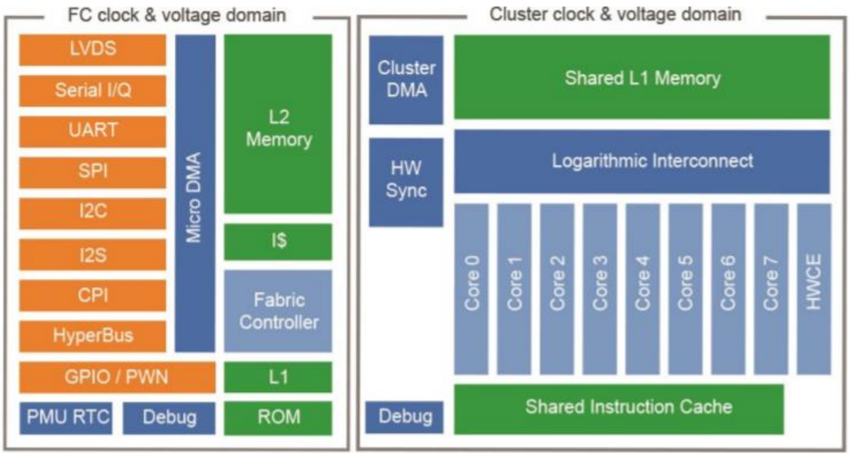
Running on GAP8 cluster.
* No Hardware Convolution Engine
** With Hardware Convolution Engine

An example of GAP8's Energy Efficiency[1]

The main highlights that of the board processor's architecture that enables GAP8 to deliver an ultra-low power and demand-driven operation are:

- A series of highly autonomous smart I/O peripherals for connection to cameras, microphones and other capture and control devices
- A fabric controller core for control, communications and security functions
- A compute cluster of 8 cores
- A Convolutional Neural Network accelerator, Hardware Convolution Engine (HWCE)
- Multiple DMA units, allowing autonomous, fast, low power transfers between memory areas in parallel with computation

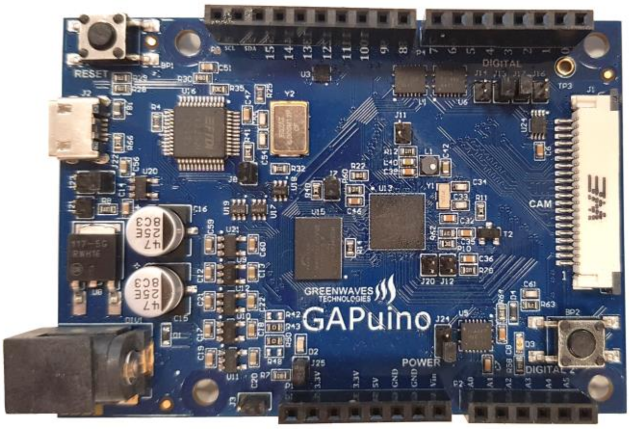
The GAP8 processor architecture is shown in the figure below:



Processor architecture of the GAP8

[1] Image from https://content.riscv.org/wp-content/uploads/2019/10/GreenWaves-RISC-V-Tour-Presentation.pdf
Source of the comparison itself: https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/new-neural-network-kernels-boost-efficiency-in-microcontrollers-by-5x)

The GAP8 processor we used is embedded on the Gapuino board (see figure below) that was launched in January 2019, with the GAP8 SDK V3.0 release that was launched in October 2019. This board costs 100,00€.



Gapuino board

6.1.2 STM32: Discovery IoT Node Board

Another board option to develop our autonomous vision solution was the the STM32 Discovery IoT Node board. It enables a wide diversity of applications by exploiting low-power communication and also supports multiple low power wireless standards such as WiFi and Bluetooth Low Energy (BLE). The STM32 can be used with an expansion software to connect to the Amazon Web Services (AWS) IoT platform, and access tools and services in the Cloud, such as device monitoring and control, data analysis, and machine learning.

Despite STM32 being a more traditional technology in the market compared to Greenwaves Technologies that is quite recent (formed in 2014), we chose to work with GAP8 because it offers a better computation performance for machine learning, a lower energy consumption and also because it was a technology that our client had a lot of interest in trying.

6.1.3 RN2483 LoRa module

The LoRa module we choose is provided by Microchip, and enable a simple exchange of data toward The Thing Network thanks to its UART interface. Ludovic chooses the module for its connection, and provided the LoRa mote, a development kit than is useful for checking LoRa connection, and quickly discover the UART interface. We were however glad to discover he chose the same module than we used in class during the semester—except that, working on a small project with Arduino and its library, we there did not use ourselves the UART interface. An explanation of the interface is provided in the RN2903 LoRa® Technology Module Command Reference User's Guide[2].

[2] http://ww1.microchip.com/downloads/en/DeviceDoc/40001811A.pdf

6.2 Software

In this section we will get into details on the software tools we used to develop our project. In the first part we will explain our choices concerning our Convolutional Neural Networks solution: why we chose these programming languages and libraries to build our Neural Networks model.

In the second part we will explain about the other software part of our project: the GAP8 environment for development on the board.

6.2.1 Convolutional Neural Network tools

Languages and libraries for the CNN

For the software part of our project, we needed to train a model (neural network) to achieve the function of identifying the data on the dashboard. For that we chose **Python** as a programming language, as it's a commonly used for this type of applications and we used it to implement image processing. The software involved part of the project consisted of the construction of our own data set, the neural training and testing of the network and passing the program to our GAP8 development board through the NNtool (Neural Network Tool).

For our Python code part, we used two very important libraries: **TensorFlow** and **OpenCV**.

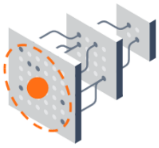
Logos of Pyhton, TensorFlow, and OpenCV

About TensorFlow

TensorFlow is an end-to-end open-source platform for Machine Learning (ML). It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications. It has three main features:

1. **Easy model building**

Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.



2. **Robust ML production anywhere**

Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.

3. **Powerful experimentation for research**

A simple and flexible architect-ture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.

About OpenCV

OpenCV is a cross-platform computer vision library released under the BSD license (open source) and can run on Linux, Windows, Android, and Mac OS operating systems. It is lightweight and efficient-it consists of a series of C functions and a small number of C ++ classes, and provides interfaces to languages such as Python, Ruby, MATLAB, etc., and implements many general algorithms in image processing and computer vision.

In our project, we mainly call the functions to perform a series of image processing operations, such as grayscale, denoising, binarization, etc.

6.2.2 GAP8 Software Development Kit (SDK)

Software Development Kit: how it works, and its problems

The GAP8 Software Development Kit (SDK) contains all the software tools and packages to allow developers to execute and compile applications on the GAP8 IoT Application Processor. Setting up the SDK properly is crucial to configuring the development environment of our project. The steps to install the toolchain can be found at: https://github.com/GreenWaves-Technologies/gap_sdk.

With the installation of the SDK toolchain we arre able to establish a connection between the GAP8 processor and our PC. That should allow us to:

- Program / control GAP8
- Debug your application using GDB
- Program the GAPuino flash memory with applications

Setting up the SDK was the first step in our project. We began working on it right after we received our board (by the end of October). Whatsoever, we already faced some challenges since the version of the board we worked on is quite recent and the SDK was undergoing changes and updates during our project.

The first release of the tool was launched on 25th May 2018 and the second release of the tool in 9th November 2018. We worked with the third version of the GAP8 which was officially released in 16th October 2019 (the same month we started our project). That meant that some instructions of the installation of the software development kit were not up to date and still undergoing changes, so we constantly had to check the changes made on the SDK and keep it up to date. For more information on the SDK releases, consult: https://github.com/GreenWaves-Technologies/gap_sdk/releases.

The SDK going through these constant changes during our project slowed us down on actually starting to discover the board. We were only able to set up the developing environment properly and run programs on GAP8 by the end of November, almost one month after we received the board. However, the help of the developers on the GAP8 developers forum, on SDK's repository Issues, and by e-mail helped us to go through some problems we would not have be able to solve on our own.

Documentation

On top of the SDK: https://github.com/GreenWaves-Technologies/gap_sdk, other documentation can be found :
- Manuals, schematics: https://greenwaves-technologies.com/sdk-manuals/
- Documentation about the SDK: https://greenwaves-technologies.com/manuals/BUILD/HOME/html/index.html

Environment: Virtual Machine

One problem we faced right on the start of our project development was that the software development kit was only adapted to work on a specific operating system. We tried to work on some operating systems (Ubuntu 18.04, or the WSL—Windows Subsytem for Linux) that should have been supported by the toolchain. It appeared that the SDK was not really adapted for these environments.

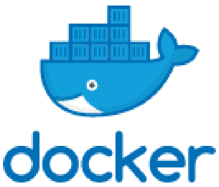
After some unsuccessful attempts, we made the choice to use the very same operating system than the Greenwaves Technologies developers used for Gap8 SDK (Ubuntu 16.04 Xenial), in the same virtualization product, VirtualBox, as they precised in the READM.me of the SDK repository. Our aim was here to avoid most of environment problems, and work in the best conditions. At first, this choice seemed to be a good one, since SDK finally compiled. However, using a virtual machine (VM) has some drawbacks because it makes compiling and running program considerably slower.

The fact that the VM was a very slow environment to work with, and considering the long and sometimes unstable installation of the SDK, make us decide, after talking to our client and with his agreement, to work on a solution for this problem: the docker.

> NB: It is only after having set up our environment in a Virtual Machine (VM) than Greenwaves Technologies developers eventually provide, among the few updates and release happening while we were working on the project, a release for their SDK, in December 2019, supporting Ubunutu 18.04. We did not have enough time to stop our development in our environments to try this new release.

A docker solution

Docker is a technology used for creating, running and deploying applications with the help of containers. Containers allow a developer to rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code. In order to run and compile the GAP8 SDK in all possible environments, and also to stop the slow process of virtual machine, we opted for installing it in docker.

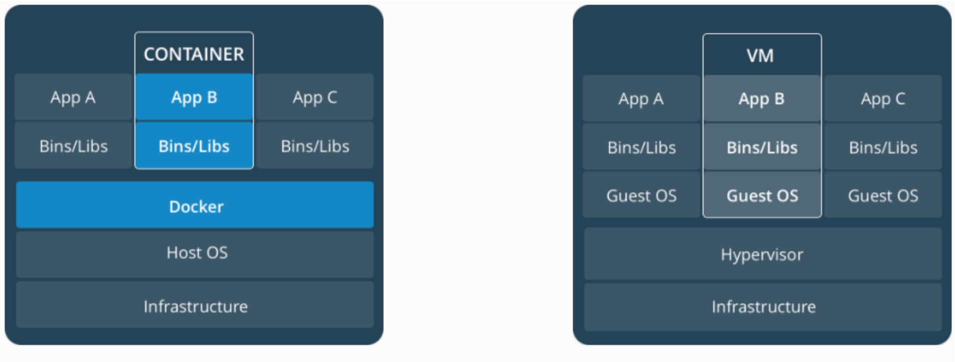


Docker logo

Unlike a virtual machine, a container does not need to boot the operating system kernel, so containers can be created in less than a second. This feature makes the docker to be more advantageous than virtual machine. The difference between the docker container and virtual machine is described as follows:

Container: Requires less space than VMs. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space.

Virtual Machine: Requires more tons of Gbs occupying necessary binaries and libraries. It is also slow to boot which makes our Sdk compilation struck in the middle. The hypervisor allows multiple VMs to run on a single machine.



Difference between Docker container and Virtual machine

1) We studied the introduction about the docker using : https://docs.docker.com/get-started/ and installed the docker using: https://docs.docker.com/install/

2) Once in the terminal, we retrieved the basic image from ubuntu: 16.04

3) Later we pulled the image of the following docker file: https://github.com/tianon/docker-brew-ubuntu-core/blob/d4313e13366d24a97bd178db4450f63e221803f1/xenial/Dockerfile, but there were some errors.

4) Currently we are working in the retrieval of image from the following docker file:

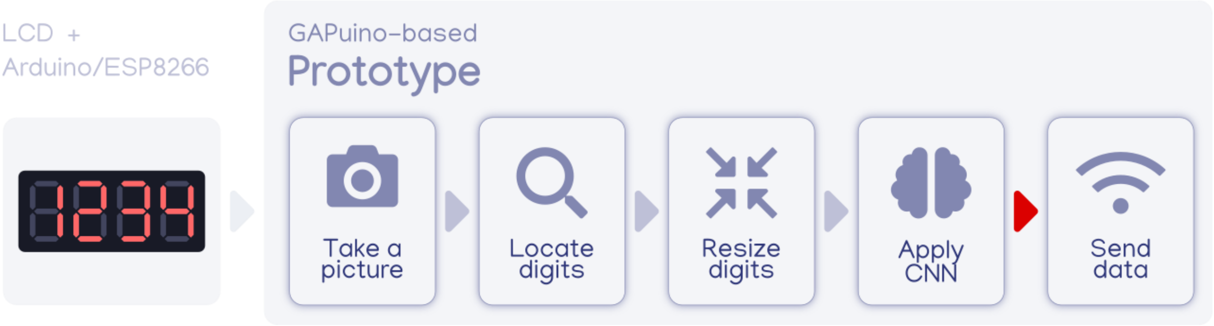
```
FROM ubuntu:16.04

RUN apt-get -q update \
&& apt-get -qy --no-install-recommends --no-install-suggests install -y \
build-essential git libftdi-dev libftdi1 \
doxygen python3-pip libsdl2-dev curl cmake \
libusb-1.0-0-dev scons gtkwave libsndfile1-dev \
rsync autoconf automake texinfo libtool pkg-config \
&& apt-get clean \
&& rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# Install git lfs
RUN curl -s https://packagecloud.io/install/repositories/github/git-lfs/script.deb.sh | sudo bash
\
&& sudo apt-get install git-lfs
&& git lfs install

# Install GAP RISCV Toolchain
ENV GAP_RISC5_TOOLCHAIN /opt/dedale/gap_sdk
RUN mkdir -p $GAP_RISC5_TOOLCHAIN \
&& git lfs clone https://github.com/GreenWaves-Technologies/gap_riscv_toolchain_ubuntu_18.git \
&& cd $GAP_RISC5_TOOLCHAIN/gap_riscv_toolchain_ubuntu_18 \
&& ./install.sh
```

7 Development

or what we did, and **how**

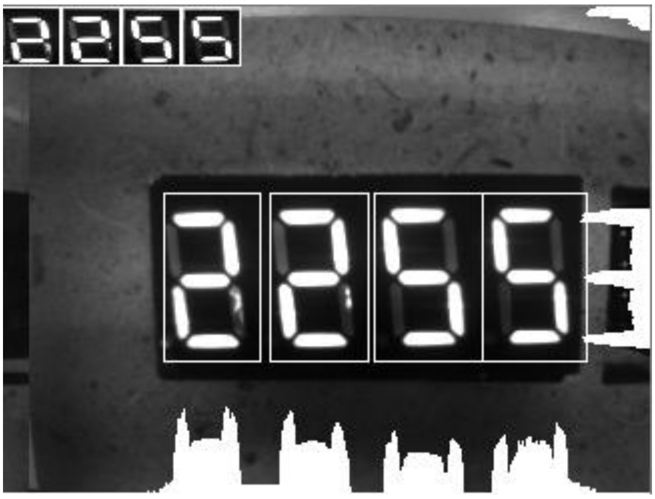
7.1 Overview of the project

Here we present an overview of our project. Our Gapuino based prototype is capable of taking pictures using a QVGA camera that captures image saving them in ppm format directly in our PC. From the taken picture, we developed an algorithm running on board capable of locating the position of the digits in the image. With the resized digits we can apply our created CNN model to recognize what digit it is.



Block diagram of the prototype

The result can be verified as an output of our code, with an image exported to the computer enabling a visualization of how the algorithm extracted data from the picture. For debug and demonstration, we draw below rectangles around the located digits, along with some "plot" of the brightness density of rows and columns, used to locate the digits.

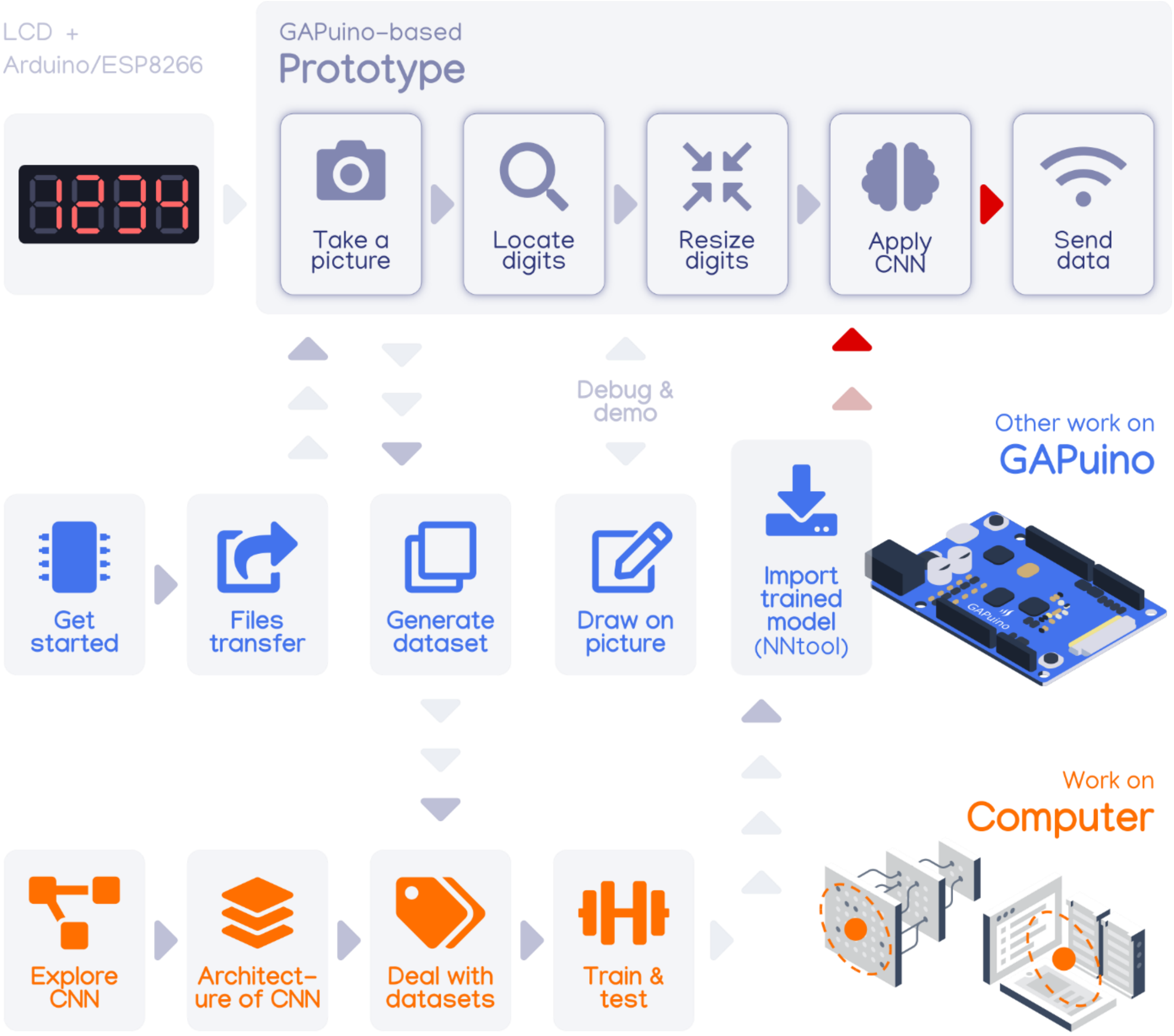


Example of an output of the program

Eventually, the trained CNN, developed with TensorFlow, is able to recognize digits located and resized by the Gapuino. On Gapuino, we also manage to run some pre-trained CNN successfully. Still, the actual SDK does not enable now a real to implement of model in Gap8, while it should soon be, as the NNtool required is currently being debugged by its developers.

However, before and to do this prototype, a lot of other work was required, work that did not eventually be technically part of the prototype. Indeed, as shown in the figure below our project can be divided into two major parts: work done on the Gapuino and work done on the computer (with Python, TensorFlow and OpenCV). These two parts are obviously integrated and interact constantly with each other.

The following sections focus on each of the “tile” presented below, to explain how work (or how to implement) each of them.



An overview of the project

7.2 Work on computer

The work on computer consists on defining our CNN model, which we will explain on the following section. We will explain how Neural Networks can be applied to solving our problem in next part of this report. Then doing a pre-treatment of images of the dataset to train our model. With the trained and tested model, we should be able to export it to the board using the NNtool.

7.2.1 Convolutional Neural Networks (CNN)

Explore CNN

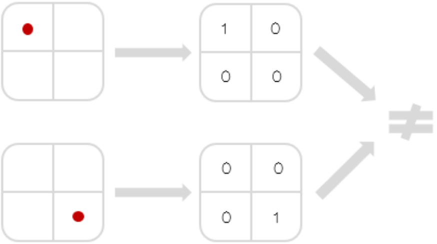
What problem can CNN solve

CNN can solve the problem of too much data to process. In our case the data is image. An image is made up of pixels, and each pixel is made up of color that is represented by a number.

Before the emergence of CNN, images were a difficult problem for artificial intelligence for two reasons: the amount of data to be processed by the image is too large, resulting in high cost and low efficiency and the difficulty in retaining the original features in the process of digitalizing the image, resulting in low accuracy of image processing.

The first problem that CNN solves is "simplifying complex problems", reducing a large number of parameters into a small number of parameters, and then processing. One important note: in most scenarios, dimensionality reduction does not affect the results. For example, many machine learning algorithms have proven that a 1000 pixels picture reduced to 200 pixels does not affect whether the picture is a cat or a dog.

The second problem that CNN addresses is the preservation of image features. To explain this we will make a simple explication of how the traditional way of digitalizing pictures works. As it’s possible to see from the picture below: if a circle means “1” and no circle means “0”, then the position of the circle will produce a completely different data expression. But from a visual point of view, the content (essence) of the image remains unchanged, only the position has changed.



Example of how digitalization

So what we can see is that when we move the objects in the image, the parameters obtained are very different and the traditional way of digitalizing images doesn’t meet what we want to achieve with image processing, which is maintaining the “essence” of the image even when features are reduced. CNN comes to solve this problem. It retains the characteristics of the image in a similar visual way. When the image is flipped, rotated or transformed, it can also effectively identify similar images.

Human visual principles

Before we understand the principles of CNN, let's take a look at what human vision is. The principle of human vision is as follows: starting from the original signal input (the pupils take in pixels), then do preliminary processing (the cerebral cortex finds edges and directions), and then comprehension of the image (the brain determines what the shape of the object in front of the eye is).

Huma vision also recognizes images layer by layer: starting by simple elements to then construct a complex image and comprehend what that image is. We can see that the features at the lowest level are simpler and more similar, the more upwards, the more features and complexity and the image starts gaining meaning. Until finally the features are combined into corresponding images so that humans can accurately distinguish different objects.

This is what CNN tries to do: emulate this feature of the human brain by constructing a multilayer neural network. Firstly, identifying image features at the lower level, then combining several lower-level features forming higher-level features. And finally making the classification at the top level.

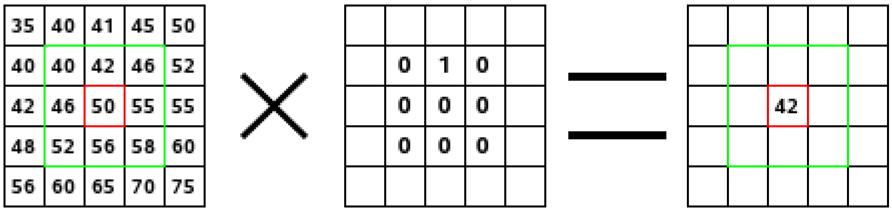
How does a convolutional neural network work: basic principles of a CNN

A typical CNN consists of three parts:

1- Convolutional layer
2- Pooling layer
3- Fully connected layer

The convolutional layer is responsible for extracting local features in the image. The pooling layer is used to greatly reduce the parameter magnitude (dimensionality reduction). And the fully connected layer is used to output the desired result. Now we will explain in more details how each layer works.

Convolution - Extracting features from the image. The operation process of the convolutional layer is shown in the following figure. On the left is the image matrix: each pixel is marked with its value. The initial pixel has a red border. The kernel action area has a green border. In the middle is the kernel and, on the right is the convolution result.



35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

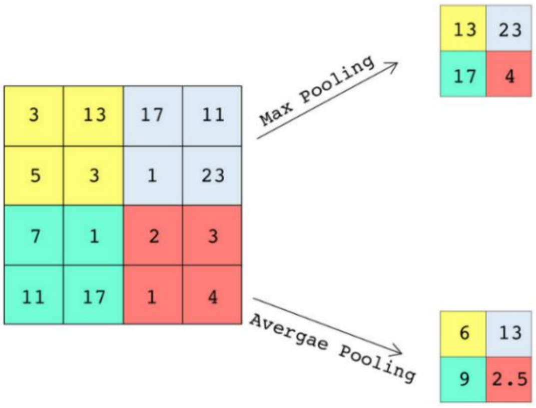
	0	1	0	
	0	0	0	
	0	0	0	

		42		

Example of convolution

Here is what happened: the filter read successively, from left to right and from top to bottom, all the pixels of the kernel action area. It multiplied the value of each of them by the kernel corresponding value and added results. The initial pixel has become 42: (40*0)+ (42*1)+(46*0) + (46*0)+(50*0)+(55*0) + (52*0)+(56*0)+(58*0) = 42. (the filter doesn't work on the image but on a copy). As a graphical result, the initial pixel moved a pixel downwards.

Pooling layer (down sampling) - data dimensionality reduction to avoid overfitting



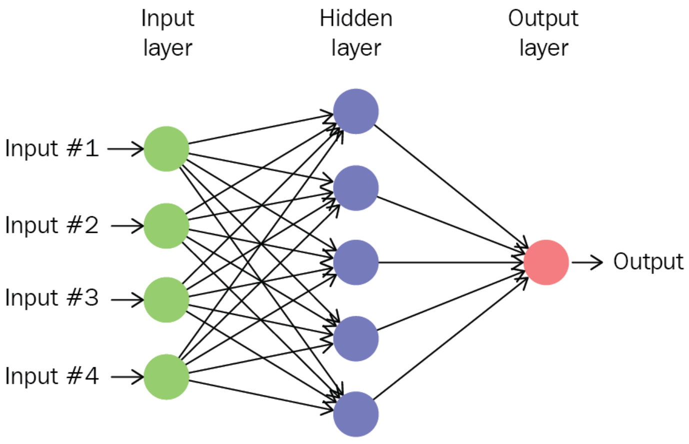
Example of pooling

The pooling layer is simply down sampling, which can greatly reduce the dimensionality of the data. The process can be explained through the example of picture above. In this example a 4x4 image is down sampled to a 2x2 by taking the maximum value or the average value of each sub-region.

The pooling layer can reduce the data dimension more effectively than the convolution layer. Not only can this greatly reduce the amount of computation, but it can also effectively avoid overfitting.

Fully connected layer - output results

This part is the last step. The data processed by the convolution layer and the pooling layer is input to the fully connected layer to get the final desired result. Below is a three-layer structure example; a real CNN however comes with a multilayer structure.

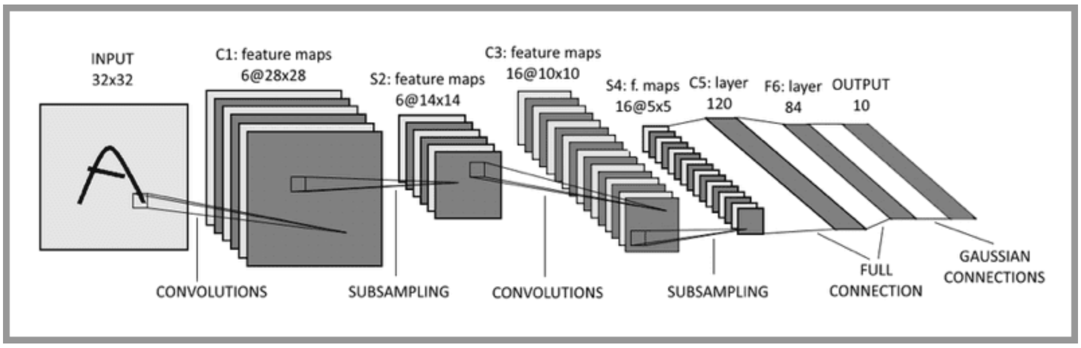


Example of fully connected layer

7.2.2 The architecture of our Neural Network: LeNet-5

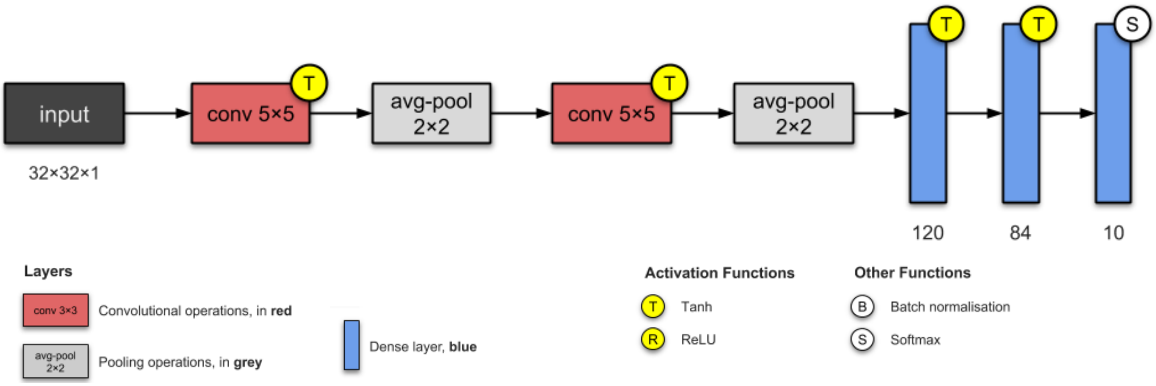
Architecture of CNN

In the figure below, the structure of a CNN **LeNet-5** is shown in the following figure. That's a well-known architecture and it's that's the structure we deployed for our CNN.



LeNet-5 CNN architecture

Below, we present a less traditional yet very clear presentation of the architecture, and what steps are required to implement it, since each block can be used as a function, including in Gapuino board also.



LeNet-5 CNN block diagram architecture

Most CNN examples start with dealing with the MNIST dataset (see following section), to train and test a model capable of recognizing handwritten digits. Here, it is the same architecture which is applied. The only difference is that MNIST dataset pictures are not 32x32px. It is also the architecture used in the example of CNN running on board provided by Greenwaves Technologies, since they use the MNIST: this choice of architecture fits our requirement considering the very nature of the image (digits), the order of classification required (0-9), and will allow us to it on the Gapuino board more easily.

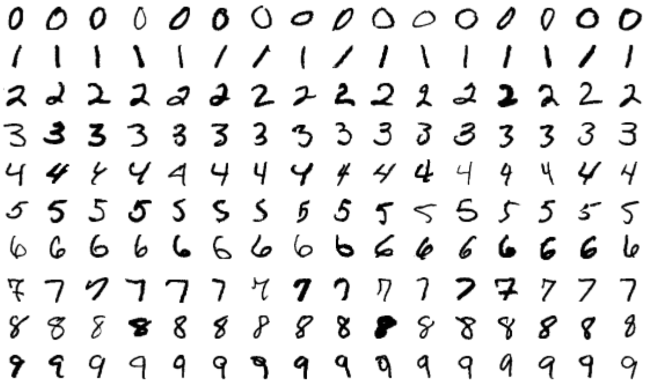
Finally, here is an example of an interactive online application of how **LeNet-5** recognize the handwriting numbers: https://cs.ryerson.ca/~aharley/vis/conv/. With this link we are able to visualize all the principles we just explained and play with layers and different architectures.

7.2.3 Dealing with datasets—and other techniques to recognize numbers

Deal with datasets

In our project, we tried three techniques to recognize numbers. First, we used the MNIST handwritten digital dataset. Then, for LCD numbers, we defined the data set by defining the highlights and shadows corresponding to each number. Finally, our Client want us to build our own dataset using the screen photos they gave us, and we successfully completed this task.

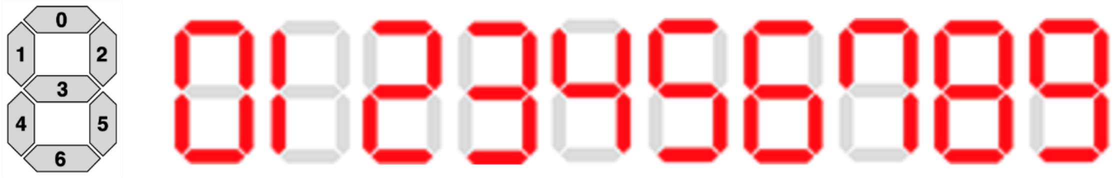
MNIST



The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. Through our first trained model approach we could verify that with the handwritten digits dataset (MNIST) it isn't possible to recognize 7-segment digits. So we had to start working on the construction of our own database.

LCD

The figure below shows that a seven-segment LED, our common LCD screen is actually a tube through a combination of seven-segment LED, which is essential to understand its composition. It can be observed by this figure to contain 7-segment LED, and marked with 0-6, respectively, there is also a decimal point 8, the following segments have corresponding led lights, when we need to show different figures only when need to light up different segments of LED lights can be, this principle is very simple! By combining different segments, we can obtain 128 kinds of combinations (i.e., 7 th power of 2), the specific combination of the results shown below for the 128 combination, we only 0-9 wherein the digital comparator 10 interest, other combinations are not the target of our attention.



Through the above picture we can observe that when we light up the LED of a specific segment, the LCD can display a specific number, then we can judge the current number by judging the characteristics of different segments.

Our dataset

We used our Gap8 development board to captured and saved more than 250 pictures of a 7-segments LCD screen, presenting 4 digits at once. As it will we explain in following sections (Build a dataset), each picture has been taken with the same position, to reduce the time we would have on pre-process the pictures. Still, some processing had to be done, to get a clean, usable dataset. First we needed to convert the captured picture format (from .ppm to .jpg or .png). Then, based on the fixed, predefined position of each digit, we were able to easily separate the numbers by positioning the coordinate range of each number inside the program.

> To go further, when the number of digits changes or we cannot predict how many digits will appear on the screen, we have designed an image processing method to separate each character, set the threshold value by binarizing the image, determine whether the binarized image is black text on white background or white text on black background, and then determine whether there is a digital part on this column according to the proportion of font color pixels in each column.
>
> We used size position for building our dataset, in order to have a systematic well defined position, however we implemented such a locating method in Gapuino board, for the final prototype.

Then because of the camera, the numbers were mirrored, so we had to turn them. The size of each number that we captured is close to 50x80, so in order to save training time and computation resources (planning to run it on GAPuino), and to use conveniently the same Neural Network Architecture than used on the previous examples based on MNIST dataset, we resized the digits to 28x28 pixels. Finally, we classified them from 0 to 9 numbers, since labelling pictures is the indispensable element of the training process.

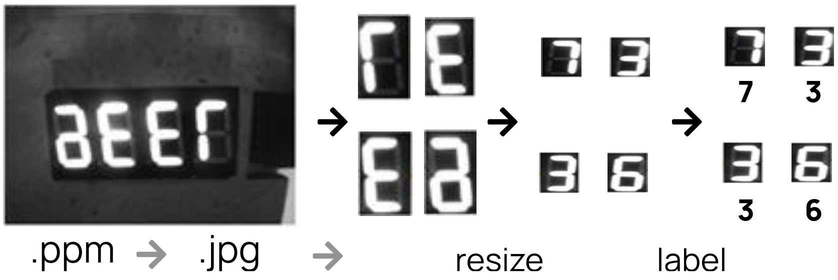


Image pre-processing

7.2.3 Training and testing

Train & test

We trained three models, the first one was trained on the MNIST handwritten dataset, the second one was actually to let the computer know the structure of each number by defining the highlight and shadow areas of the LCD. It is a digital recognition (image classification) model that we realize by building CNNs from the pictures we actually get.

Here we mainly show how we train the third model: first, after we have completed a series of pre-processing operations such as taking digital pictures from the original screen and completing orientation adjustment, we put the 0-9 numbers into the corresponding folders, so that the pictures in the same folder are given the same label during training.

Our python program is mainly divided into four parts, training, testing, preprocessing and model.

In the **Model.py**, we built our neural network structure:

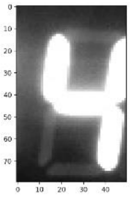
- A simple convolutional neural network, two (convolution + pooling layers), two fully connected layers, and the last softmax layer for classification.
- 3x3 convolution kernel (3 channels), maximum pooling, activation function selection relu

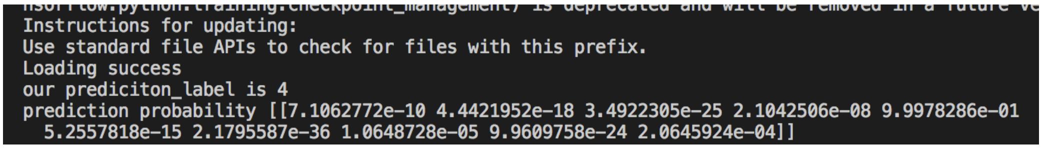
At the same time, we define the calculation and optimization methods of the network output loss, and the method of evaluating / calculating the accuracy rate.

In the **PreWork.py**, we get all picture path names in the path, and save it to the corresponding list, at the same time, attach a label and store it in the label list. At the same time, some data preprocessing, image decoding, size adjustment, and standardization processing were performed.

Then we come to **Train.py**, We define the training learning rate, the size to which the predetermined picture needs to be adjusted (here, 28x28), the largest step to train, then create a session in TensorFlow, call the method we wrote before, and continuously optimize the parameters of the neural network And output the current accuracy rate and loss every 100 steps, and finally save our training model as checkpoint (or .h5 for NNTOOL transmission).

Finally, if we want to test our model on computer, we use **test.py**, The method of running the test is very simple. We only need to define the location of our test data set, define the method to extract images from it, and load the model we have saved (here, checkpoint). We can get the images and observe whether the output results are correct.





```
nsorflow.python.training.checkpoint_management) is deprecated and will be removed in a future ve
Instructions for updating:
Use standard file APIs to check for files with this prefix.
Loading success
our prediciton_label is 4
prediction probability [[7.1062772e-10 4.4421952e-18 3.4922305e-25 2.1042506e-08 9.9978286e-01
  5.2557818e-15 2.1795587e-36 1.0648728e-05 9.9609758e-24 2.0645924e-04]]
```

We see that we can get the predicted numbers and the probability that our model predicts each number from 0-9.

7.2.3 Validating

Implementing the model on the Gapuino is not yet possible technically while we are writing this report. However, to prove the accuracy of are the solution, we run the program on Gapuino and, based on the image it has processed, we gave the image to the CNN we just presented. Knowing that the same architecture is already in place on Gapuino board, and the only thing missing is the actual model, this test allowed us to validate our program: if the test was not exhaustive, it was indeed successful, as the CNN did indeed recognize the numbers below very accurately.

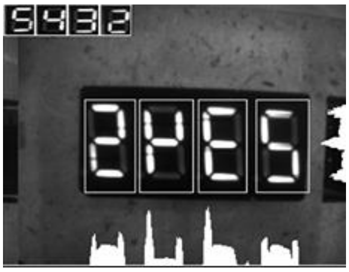


Image extracted from Gapuino, on which we finally validate our model

7.3 Work on Gapuino

To allow our hardware platform Gapuino to compile and run programs, we first had to install all the software developed tools required by the board (by installing the SDK, already explained in the "Software" section) and so prepare the environment for development. Source code itself of our programs are provided to the client.

7.3.1 Getting started with Gapuino

Get started

The environment has been presented above (Materials: Software), regarding the set up, and installation. Here are quick links to main resources provided by Greenwaves Technologies:

Greenwaves documents (we used GAPuino Release B)
SDK repository (Github)
Detailed online documentation

We could precise that the set up takes time, and requires a good connection (better than INSA's connection) to prevent failure when downloading, are even *making* the SDK (often, we had to make all the SDK, and it could easily failed). We used the `gapuino_v2.sh` configuration file targeting the Gapuino boards sold after October 2019, since the board was indeed bought during this first month of release.

> NB: With the last updates of the SDK, the targetted OS presented in the READ.me is eventually Ubuntu 18.04 Bionic Beaver. Other bugs might appear, as well as some issues faced during our installations could have been fixed.

When all the get started of the SDK is done, a `hello_word` example might be lunched: https://github.com/GreenWaves-Technologies/gap_sdk/tree/master/examples/native/pulpos/helloworld

For a first compilation, if some error appears like the following one:

```
Unable to connect to localhost port 9999 (Connection refused)
```

please check the next parameter: the bridge needs to wait before being lunched.

> NB: Gapuino is not the only development board existing. Hovewever, in order to use the GAPPoc (another developement board embedding Gap8 processor and camera, Poc standing Proof-Of-Concept) that Ludovic proposed to us, an OLIMEX or compatble external probe is requied (such as their ARM-USB-OCH-H model), plus an adapter JTAG-20-pin/2.54mm pitch to 10-pin/1.27mm pitch (ARM-JTAG-20-10).

7.3.2 Conveying data from and to computer

Files transfer

For loading or storing files (such as images) from a computer, a bridge tool is required, and Greenwaves provides is pulp-rt-bridge. We used it to test algorithm on board based on images we had on computer, but also to export images from the Gapuino board, taken with the camera.

Some of our problems were due to this bridge, which is *slow*, as explained by the developers, and it was one of the reason why we were not able to compile code at first. To indeed trigger this function in bridge, we need to add a flag in the Makefile, usually already present in samples:

```
PLPBRIDGE_FLAGS += -fileIO <option>
```

`<option>` represents how long the bridge should wait before to start, and the predefined delay (2s) was not long enough: probably because our computer were laptops, running a VM, we needed to increase this parameter in order to be able to open the bridge (up to 10 or 15s, depending on our laptops). Also, to improve the performance of the bridge, we had to change the VM parameter to USB 3.0, and connect the board through a compatible port.

Concerning the use of the bridge in the program, the code starts by opening it, and ends by closing it (see hello world example).

> A new bridge, OpenCD, appeared on mid-December. However, by the end of January, developers are still working on it. Eventually, it should replace the pulp_rt_bridge.

7.3.3 Taking picture

Take a picture

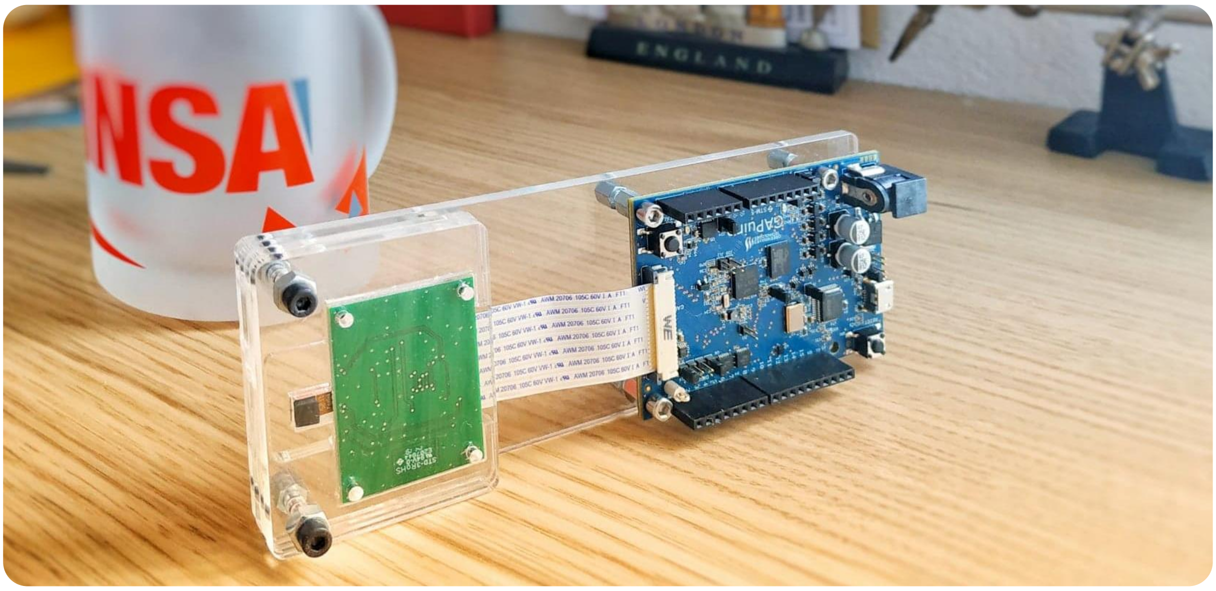
The simplest way to take picture is to follow the example provided in the SDK. For the QVGA low-power camera provided in the developer kit, the Gapuino User's Manual provides an old link, `gap_sdk/examples/pulp-examples/periph/camera/camera_himax` which is not accurate anymore (last version: 7th October; when used at the end of October, the architecture of the SDK has changed). A working sample can be found here: https://github.com/GreenWaves-Technologies/gap_sdk/tree/master/examples/native/pulpos/periph/camera/camera_HIMAX_IO

7.3.4 Generating a dataset

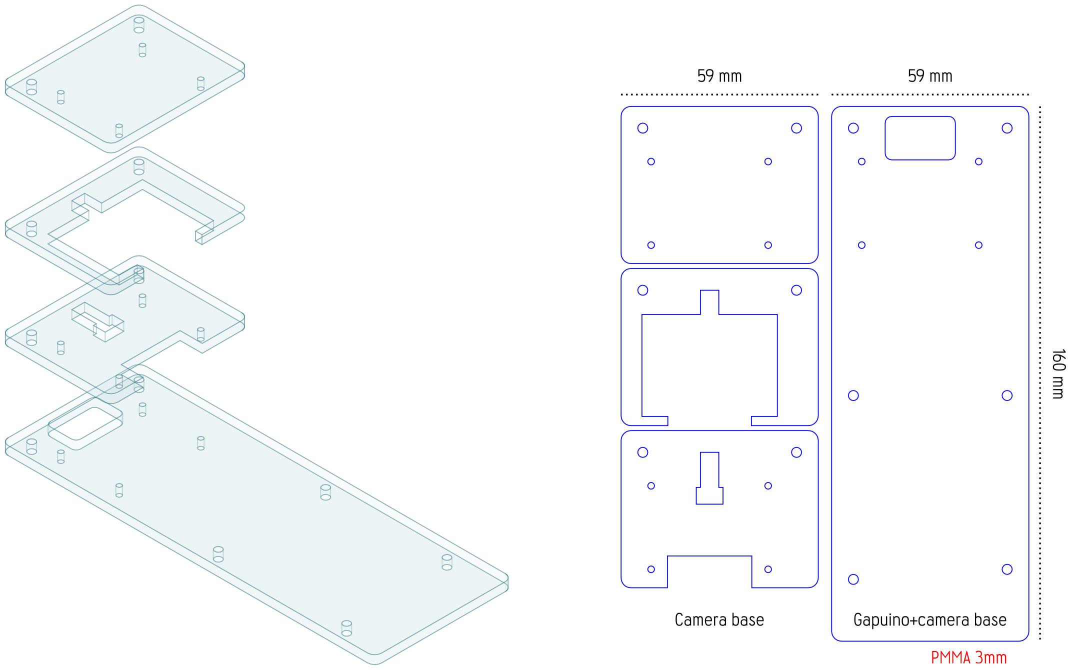
Generate dataset

To produce the data set, we need two things: a reproducible method for taking pictures (to train a CNN with a clean dataset) and a subject (to take pictures of).

A reproducible method: the camera in connected to the board through a flat cable. To get a fixed camera, we built a base at Fabric'INSA. The base was designed on the open software Inkscape, and it was cut in PMMA (plexiglass), with a Laser Cutting Machine. Below are the model and pictures of the base. Plans are provided to the client.

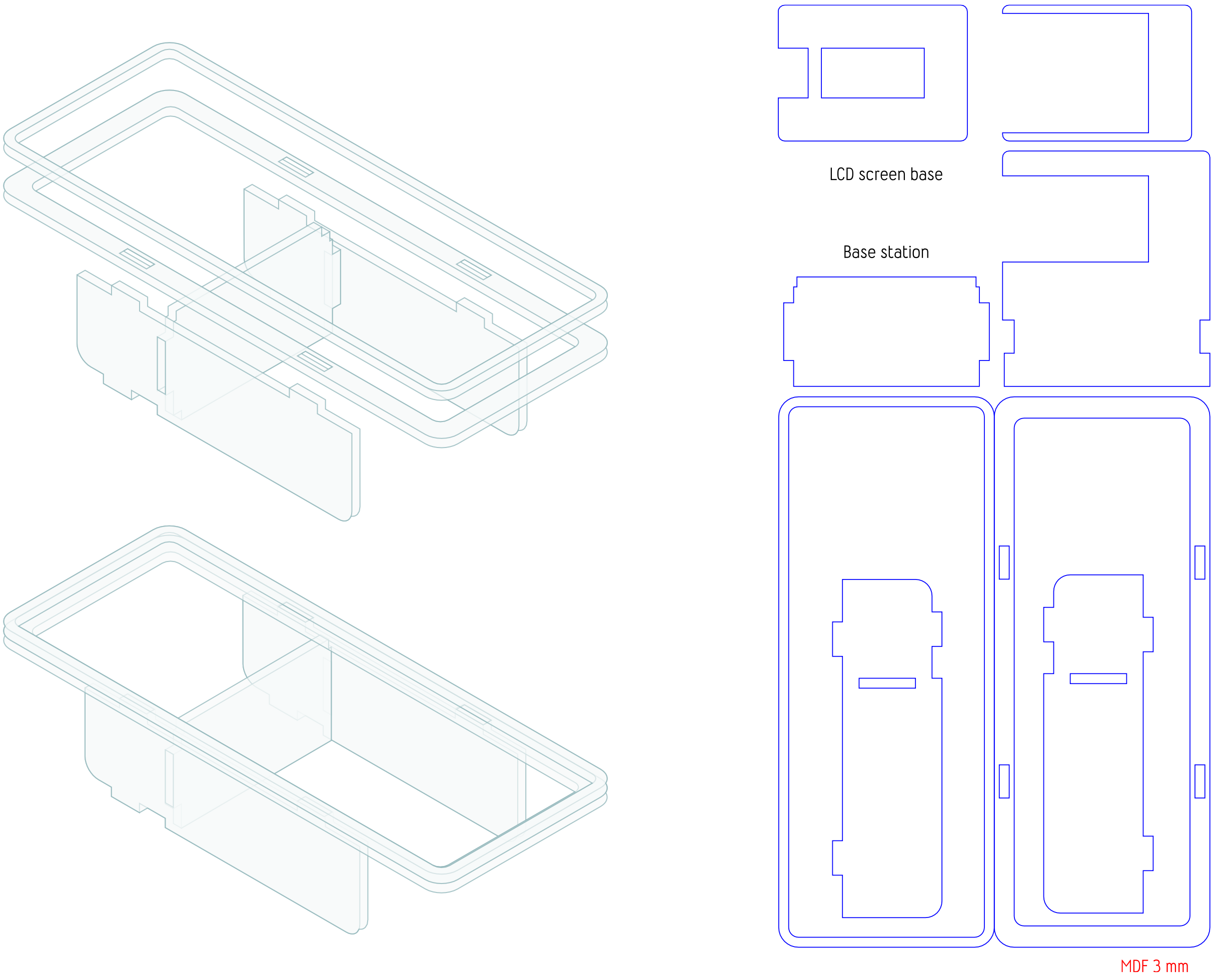


Gapuino base

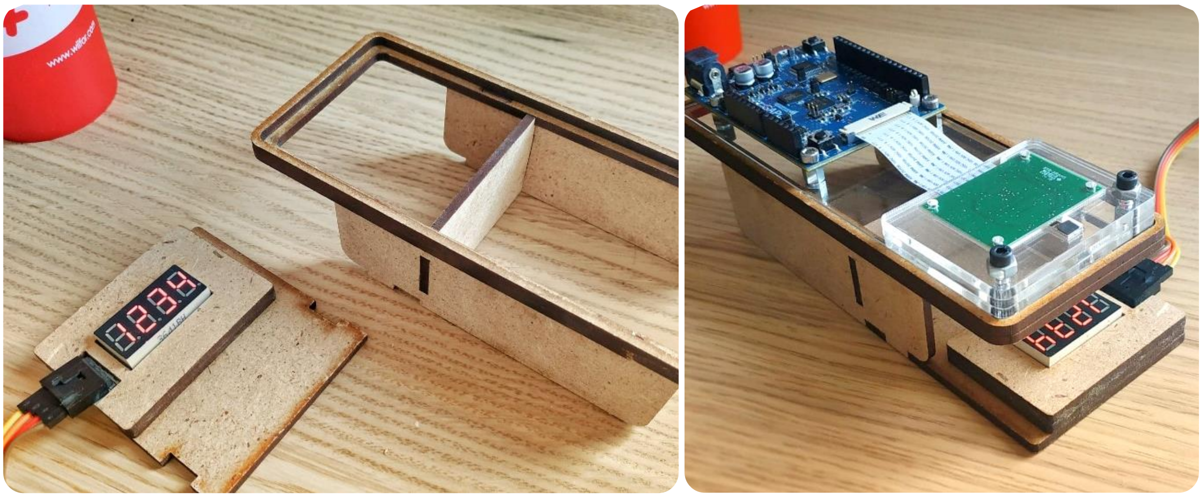


Gapuino base: illustration & plan

A subject: we first tried to take picture of a phone displaying digits, but the results were neither well looking, convenient, nor realistic. We thus build a small station (Laser Cutting, MDF material) hosting a 4 digits TM1637 LCD screen, connected to either an ESP-8266 or Arduino, with short program in Arduino (using the corresponding `TM1637.h` library for the screen, and later `IRremote.h` library to control the display using an Infrared remote).



Base station: illustration & plan



Base station & complete set up for generating dataset

7.3.5 Locating digits

Locate digits

Regarding the localization of the digits, we could have use a CNN to recognize the whole scene. It would have required a much more sophisticated architecture, a huge dataset, different kinds of screens, and to perform a regression on the overall (here, we do a classification). We decided to start our thinking from the idea that digits were inline. There could be one, to, or more of them, in different size, different places, and we need to recognize and find them.

By drawing a "spectrum" are the density of brightness of each row, and each column, we can highlight places where digits are. To increase the accuracy of this method, we applied a threshold first (making every pixel binary, to only consider brightest pixels. The threshold was taken at mid-range of the color brightness scale, by only considering MSB (most significant bit) of each pixel.

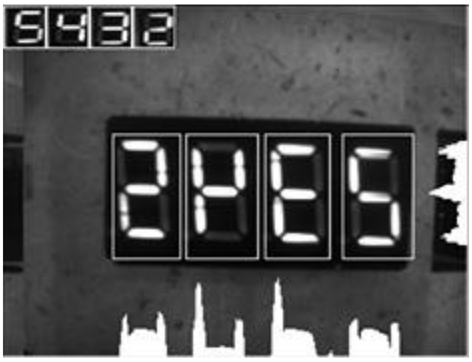


Illustration of a brightness density plot on Gapuino (using "Draw on picutre"), then demonstration of the locating algotihm working (rectangles) and drawing of digits once resized (using "Resize picture")

Drawbacks (issues for future work)
We must change the algorithm for black on white LCD screens.
If there is some white background somewhere around the black screen, it does not work.

Advantages
Simple algorithm, requiring few resources (perfect for embedded systems).
High accuracy for lighting digits, and nearly 100% for digits on a whole black background.
Allow to find digits in different positions, at different sizes.

7.3.6 Drawing on picture

Draw on picture

To debug those algorithms, it was helpful to directly draw on the picture taken from the camera and transfer it to the computer (it required between 30 and 60 seconds each time, depending on our laptops, mainly due to the bridge).

Images are stored in a one-dimension array. To access a `(x,y)` pixel, we have to reach `img[y*IMG_W + x]`, where `IMG_W` stands for "image width" (here 324, the camera working in a 244x324px resolution instead of the classical QVGA resolution of 240x320px). Moreover, when accessing the array representing the future PPM file to be transmitted to or from the computer, an area shall be kept for the PPM header, hence the line `imgBuff0[PPM_HEADER + y*IMG_W + x]` often used in the code.

To modify pixels values, we can directly modify the array. In some applications provided in the SDK, we found a short library (`ImageDraw.h`) doing simple tasks as plotting rectangles. We only need to copy .c and .h files in the project and modify imports and Makefile to use it.

7.3.7 Resizing image

Resize digits

There are different kinds of algorithms to scale pictures (downscale as upscale) without loosing any features. Considering our target (IoT device), our aim (28x28px pictures, same as the model trained in TensorFlorw), and the picture itself, it appeared that the simplest algorithm worked perfectly well (as shown on the picture above). It is a nearest-neighbor algorithm, where we simply sample the source image to the required resolution. Each pixel of the new picture becomes a copy of a pixel from the original, found at the a `(x,y)` pixel of the new pixel, multiplied by the scale factor (the other pixels around this one are ignored).

7.3.8 Applying a CNN

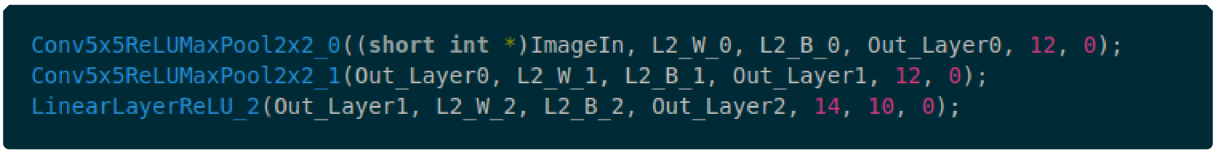
Apply CNN

To try a CNN working example, we considered the Water Meter application example:

https://github.com/GreenWaves-Technologies/gap_sdk/tree/master/applications/WaterMeter

At first, it seemed to be close to our own specification, however: images where here imported from the computer (not camera), digits positions were pre-defined in the code (no localization), digits sized were already fitting 28x28px (no resize), and the CNN applied relied on a MNIST trained-model (not our LCD one). However, it is a perfect example to show a real, actual CNN running on Gapuino.

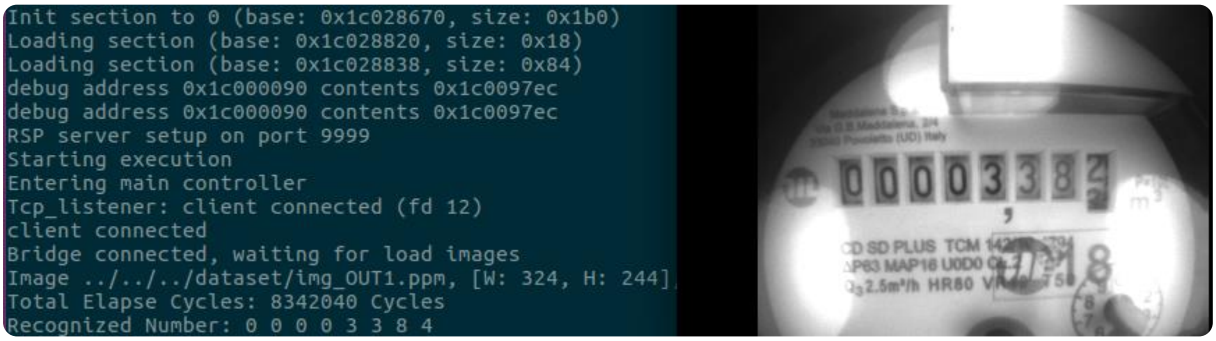
It applies the architecture previously explained, as highlighted with the code below:



```
Conv5x5ReLUMaxPool2x2_0((short int *)ImageIn, L2_W_0, L2_B_0, Out_Layer0, 12, 0);
Conv5x5ReLUMaxPool2x2_1(Out_Layer0, L2_W_1, L2_B_1, Out_Layer1, 12, 0);
LinearLayerReLU_2(Out_Layer1, L2_W_2, L2_B_2, Out_Layer2, 14, 10, 0);
```

We developed our final application starting from this one and implemented the required features instead of the basic import from computer. Since we cannot yet import our own model, it still runs the CNN based on the MNIST dataset: the "only" missing part would be to import it, in place of the MNIST one.

Here some output of the application:



```
Init section to 0 (base: 0x1c028670, size: 0x1b0)
Loading section (base: 0x1c028820, size: 0x18)
Loading section (base: 0x1c028838, size: 0x84)
debug address 0x1c000090 contents 0x1c0097ec
debug address 0x1c000090 contents 0x1c0097ec
RSP server setup on port 9999
Starting execution
Entering main controller
Tcp_listener: client connected (fd 12)
client connected
Bridge connected, waiting for load images
Image ../../../dataset/img_OUT1.ppm, [W: 324, H: 244]
Total Elapse Cycles: 8342040 Cycles
Recognized Number: 0 0 0 0 3 3 8 4
```

Exemple of CNN running on Gapuion: Water meter application

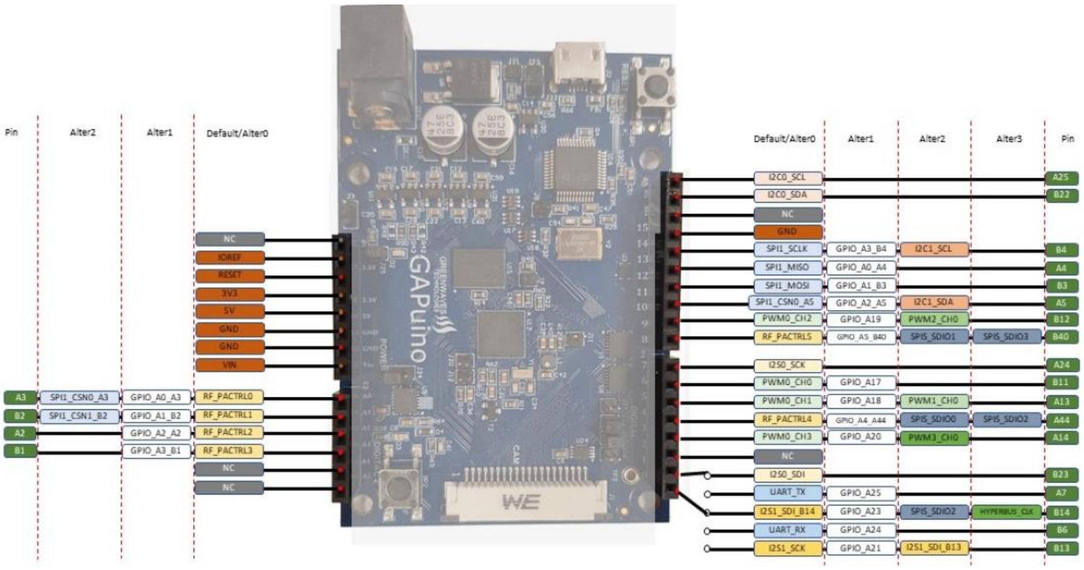
Because of a simple training and a fixed position, the CNN does not recognize the last rotating character.

7.3.9 Connecting LoRa module through UART

Send data

We missed some time to develop this feature. How LoRa's module works is however acquired, and the best document to consider in order to *talk* to it remains the RN2903 LoRa® Technology Module Command Reference User's Guide we presented in the Material section. We can already send data from computer to The Thing Network through a UART console. For development needs, the activation of the module on The Thing Network is done through an ABP (Activation by Personalization) rather than using an OTAA (Over The Air Activation), that should be used for deployment. Data sent can then be seen on The Thing Network console.

Gapuino only provides one UART peripheral, already used by the FTDI USB bridge (to flash and print in the console at runtime). Based on Gapuino User Manual and Schematics, we know that we need to shorten Jumper J13 instead of J14 and J17 instead of J16 to connect the UART peripheral to the pins P4.8 and P4.7 (respectively TX and RX). By default, those pins are connected to I2S0 and I2S1. Then, we need to shorten J7 and J8 to enable the use of UART one those pins and UART_TX and UART_RX to the FTDI USB bridge. However, we missed some time to implement, test and debug this feature.



Pin	Alter2	Alter1	Default/Alter0
			NC
			IOREF
			RESET
			3V3
			5V
			GND
			GND
			VIN
A3	SPI1_CSN0_A3	GPIO_A0_A3	RF_PACTRL0
B2	SPI1_CSN1_B2	GPIO_A1_B2	RF_PACTRL1
A2		GPIO_A2_A2	RF_PACTRL2
B1		GPIO_A3_B1	RF_PACTRL3
			NC
			NC

Default/Alter0	Alter1	Alter2	Alter3	Pin
I2C0_SCL				A25
I2C0_SDA				B22
NC				
GND				
SPI1_SCLK	GPIO_A3_B4	I2C1_SCL		B4
SPI1_MISO	GPIO_A0_A4			A4
SPI1_MOSI	GPIO_A1_B3			B3
SPI1_CSN0_A5	GPIO_A2_A5	I2C1_SDA		A5
PWM0_CH2	GPIO_A19	PWM2_CH0		B12
RF_PACTRL5	GPIO_A5_B40	SPIS_SDIO1	SPIS_SDIO3	B40
I2S0_SCK				A24
PWM0_CH0	GPIO_A17			B11
PWM0_CH1	GPIO_A18	PWM1_CH0		A13
RF_PACTRL4	GPIO_A4_A44	SPIS_SDIO0	SPIS_SDIO2	A44
PWM0_CH3	GPIO_A20	PWM3_CH0		A14
NC				
I2S0_SDI				B23
UART_TX	GPIO_A25			A7
I2S1_SDI_B14	GPIO_A23	SPIS_SDIO2	HYPERBUS_CLK	B14
UART_RX	GPIO_A24			B6
I2S1_SCK	GPIO_A21	I2S1_SDI_B13		B13

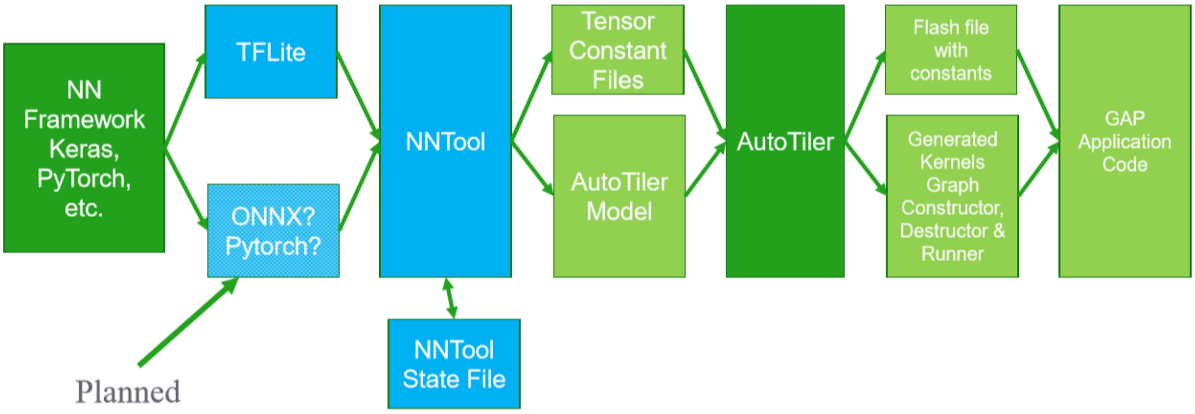
GAPuino connectors (from User Manual)

7.3.10 Importing a trained model: the NNtool

Import trained model (NNtool)

To run neural networks applications on the board, the installation of a specific tool (the NNtool) developed by Greenwaves Technologies is required. This tool allows us to transform our trained neural network model into a GAP8 code that can be computed directly on the board. This is a new feature that came with the release 3 of the GAP8 SDK, launched in October 2019.

Basically, what this tool does is: it maps high-level graphs (such as Keras, TensorFlow that we used in the development of our CNN) directly into GAP8 with automatic quantization. That is, with an automatic conversion of the CNN model that reduces the model's size and improves the use of the CPU and the computation time. This quantization consists on reducing the number of bits that represent a number with the aim to reduce bandwidth and storage, lower power consumption, speedup computations. That's why this solution is so interesting. In the figure below, we show how NNtool works.



Block diagram of the NNtool

So what NNtool does is:

1 – After training the CNN model (by running the *train.py* file), the model is saved in a h5 file format
2 – Then, it converts the H5 file to a TFLITE file using TensorFlow's TFLITE converter
3 – It then generates an NNtool state file by running an MMtool script with commands to adjust the tensor and activation order, fuse certain operations together and automatically quantify the graph
4 – It then uses this state file to generate an AutoTiler graph model. The autotiler translates code into GAP8 application code

From that we can run the saved Neural Network model on the board and use to it embedded on the system.

Unfortunately, until now there's no documentation available on how to use this NNtool to deploy our own TensorFlow model on the GAP8. We entered in contact with the developer's of the board and them too we're having troubles with the NNtool because of incompatibility between the autotiler the generators. They said this problem is due to a migration of the SDK from PLPBRIDGE, a tool from the PULP project which allows communication between our PC and a GAP8 processor, to OpenOCD which is faster.

Normally, with the NNtool working, we should be able to run our model directly on the board by just using our *train.py* file and the NNtool. For now this is the only missing link to our project and this could be easily implemented in the future to have a complete functioning prototype.

Conclusion

Eventually, we were able to deliver a neural networks model trained with a database constructed by us and fully adapted to our use case scenario. We were also able to deliver a functioning prototype using a new embedded technology, the GAP8 microcontroller, capable of taking pictures and automatically locating and resizing digits in the image so our neural networks could evaluate and predict the LCD display.

Since we worked with a new technology still undergoing some development updates, we could not compile and run directly our neural networks model directly on board. The missing link between our neural network model provided and the image (made through the NNtool) could possibly be a topic for a future work at Wiifor, for example. In this report, we gave information on how this could be done, once the new bridge's problems will be resolved by the Greenwaves Technologies developers—and it should be soon.

This project made us put in practice some of the skills learned during our teaching program at INSA from a technical side, but also make us learn on our own a lot of other knowledge and skills from other technical fields, since the projects brought together Neural Networks, IoT, understanding microcontrollers architecture, embedded coding, virtual machines and container environments. Moreover, it gave us a practical and continuous work on communication, project management and teamwork.

Through this innovation project, we did exercise many different skills from different fields and detect difficulties each one of us had to improve. It was also a self-discovery exercise and we are glad to have learned those knowledges, discovered these skills, and produced those results.

Acknowledgements

We would like to thank our English professor, Joseph Shea, for his continuous help during the semester, the time he gave us to work on our project, and his effort on correcting our mistakes and helping us improve our presentations skills—this is also why we would like to apologize, for not having completed his report soon enough to give him the chance to correct us until the end. But again, for giving us one of the most precious gifts we could have this semester: time. To work on this project.

We would also like to thank our Innovation and Acceptability professor, Eric Jolivet, for giving us guidance and encouragement to improve our work in an innovative way. Also for making us reflect on project management and the impact our work could have on society.

Also, a special thanks to professor Eric Alata for his support as being our INSA tutor during this whole process.

Finally, we are delighted to thank our client Mr. Ludovic Tancerel for giving us the opportunity to work on this project in the first place, but also for being so involved in the whole process. More than just a client for this, M. Tancerel was indeed a real tutor to us, making us comfortable in all the works we did, being friendly and always offering his time and help when he could.

Bibliography

[1] **A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning**, Dinpanjan Sarkar, 14 Nov 2018. Link to the article

Really practical. Explains the theories of CNN applying to a code. The part about the regularization of the images is really informative.

[2] **Snagging Parking Spaces with Mask R-CNN and Python, Adam Geitgey**, 21 Jan 2019. Link to the article

Explains how region CNN works and has an example of code application. This can be used in the future in the case the caption of image isn't a fixed distance from the object.

[3] **ImageNet Classification with Deep Convolutional Neural Networks**, Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. Link to the article

Explain how we use R-CNN to do the classification.

[4] **Fast R-CNN**,Ross Girshick, 27 SEP 2015. Link to the article

Propose how fast R-CNN works to classify objects and its advantages compare to traditional R-CNN.

[5] **Rich feature hierarchies for accurate object detection and semantic segmentatio**, Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik, 2014 CVPR. Link to the article

Why is R-CNN for: to separate images in small regions (or region with CNN's features) and how we do object detection with R-CNN.

[6] **Conceptual Understanding of Convolutional Neural Network - A Deep Learning Approach**, Sakshi Indolia, Anil Kumar Goswami, S. P. Mishra, Pooja Asopa, 2018. https://www.sciencedirect.com/science/article/pii/S1877050918308019

Provides the conceptual understanding of the CNN with its three most common architectures and learning algorithms.

[7] **Continual lifelong learning with neural networks: A review**, Parisi , Ronald Kemker , Jose L. , Christopher Kanan , Stefan Wermter, 2019. https://www.sciencedirect.com/science/article/pii/S0893608019300231

Summarizes the main challenges linked to lifelong learning for artificial learning systems and compare existing neural network approaches that alleviate, to different extents, catastrophic forgetting.

[8] **From Cloud Down to Things: An Overview of Machine Learning in Internet of Things**, Farzad Samie , Lars Bauer, and Jörg Henkel, Fellow, IEEE, 2019. Link to the article

A state of the art of Machine Learning (ML) by usage (Classification, Regression, Clustering, Reinforcement decision making: good to read to discover ML), ML in IoT (applications, uses)

New trends:
- Machine Learning at IoT Edge (limited energy budget, computation capability, and memory : it requires 1) more efficient hardware platforms to carry on ML algorithms and 2) optimizing ML techniques to reduce their power consumption, memory requirement, and computation intensity.
- Approximate Computing in ML (for some techniques, approximation can lead to save energy and computing)

- Distribute Computation Among Layers (Offloading).

[9] **An overview of next-generation architectures for machine learning: roadmap, opportunities and challenges in the IoT era**, Shafique, M., Theocharides, T., Bouganis, C-S., Hanif, MA., Khalid, F., Hafiz, R. Rehman, S., 2018. Link to the article

Contains some performance comparison between hardware platforms, algorithms, etc. for machine learning in the IoT.

[10] **The Research of Intelligent Image Recognition Technology Based on Neural Network**, LI Huanliang, 2015 https://download.atlantis-press.com/article/18539.pdf

[11] **R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms, Understanding object detection algorithms**, Rohith Gandhi, 2018. Link to the article

Special CNN detecting object position (see title) > most are memory and computing consuming. Faster R-CNN and YOLO might be intersting.

[12] **Machine Learning for OpenCV**, Michael Beyeler, 2017. Link to the book

In the descitpion provided (*About This Book*):
- Load, store, edit, and visualize data using OpenCV and Python
- Grasp the fundamental concepts of classification, regression, and clustering
- Understand, perform, and experiment with machine learning techniques using this easy-to-follow guide
- Evaluate, compare, and choose the right algorithm for any task

Links: Table of contents | What this book covers

Chapters to read for basic knowledge: Getting started with machine learning | Problems that machine learning can solve | Understanding the machine learning workflow | Understanding supervised learning

[13] **A Ultra-Low-Energy Convolution Engine for Fast Brain-Inspired Vision in Multicore Clusters**, Conti, Francesco & Benini, Luca. (2015). https://ieeexplore.ieee.org/document/7092475

Emphasizes the performance of architectures based on a cluster of power-optimized RISC processors with Hardware Convolutional Engine (HWCE, like the architecture of the Gap8)

[14] **Computer vision & deep learning: Resource Guide,** Adrian Rosebrock, 2019. https://www.pyimagesearch.com/static/cv_dl_resource_guide.pdf

List of resources to start discovering, learning, doing computer vision & deep learning

- List & quick presentation of python libraries & packages (overview) : OpenCV, dlib, schikit-image, Keras, imutils, schikit-learn, TensorFlow, mxnet
- List of tutorials, guides, & blogs posts on those subjects : OpenCV and image processing, deep learning, facial applications, OpenCV and video, Optical Character Recognition (OCR), Raspberry Pi, etc.
- Books & courses
- Conferences

[15] **Illustrated: 10 CNN Architectures,** Raimi Karim, 2019.

https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

A visual explanation of ten CNN architectures, with explanation of each layer meaning, in a general overview.