QGIS plugin for geospatial data processing in the cloud

Oriol Casas Carrasquer

Resum– QGIS és una eina GIS de programari de codi obert que amb l'ajuda d'un plugin es pot comunicar amb un servei web, els quals han estat desenvolupats en aquest projecte. Un grup d'investigadors han desenvolupat algoritmes d'aprenentatge automàtic que processen imatges geospatials, aquests processos tenen uns costos computacionals considerables per executar-se en una màquina local. El servei web executarà els processos en un servidor i retornarà els resultats a QGIS per analitzar i emmagatzemar informació. Per obtenir els resultats, els processos s'inicien en imatges de Docker montades a partir de Dockerfiles personalitzats per a cada procés.

Paraules clau– GeoAI, GIS, QGIS, Aprenentatge automàtic, processament al núvol, Servei Web, Client-Servidor

Abstract– QGIS is a GIS open source software tool which with the help of a plugin can communicate with a web service, both of them have been developed in this project. A group of researchers has developed Machine Learning algorithms that process geospatial images, these processes have considerable computational costs to run in a local machine. The web service will run the processes in a server and return the results to QGIS in order to analyze and store information. To get the results, the processes are launched in Docker images build by custom Dockerfiles for each process.

Keywords– GeoAI, GIS, QGIS, Machine learning, Cloud processing, Web Service, Client-Server

1 INTRODUCTION

GEOGRAPHICAL INFORMATION SYSTEMS[1] (GIS) are software tools that manipulate data from real-world linked to a spatial reference and can perform a wide range of tasks with them. Among these tasks, the most notorious are: store, manipulate, analyze and present data. In order to do these tasks, the data needs information about which coordinates are represented, which features shall be known about these coordinates and how can different coordinates are related between them. The data is represented by two types of layers that are listed below.

- **Raster layers** are pixel based. Information is organized in a 2D matrix, where each matrix cell (pixel) provides data of a rectangular area.
- **Vector layers** describe real-world locations using points (an exact point), lines (paths) or polygons (regions). These locations are described by features. At the same time, each feature is defined by its attributes, data in the form of text or numerical information.

Figure 1 shows an example of how these layers can represent a real-world location.

The most popular GIS software are: QGIS[3] and ArcGIS[4].

QGIS is an open source program which supports multiple formats of layers. It also allows using data from external web services. QGIS counts with a wide community and it even incorporates third-party GIS packages. Lastly, it allows developing plugins for the program which can be useful to automate tasks and customize some tasks.

ArcGIS has a licensing system, different types of paid licenses are available based on the level of functionalities wanted. It disposes of different desktop applications which can be extended with paid extensions.

The functionalities in both platforms are similar but QGIS does not require to buy every functionality and extra tool that could be needed. This means we have a free system with no costs and extensible using plugins apart from its community with lots of support being open source.

In the last decade, a significant amount of georeferenced information has been gathered by drones in addition to decades of data gathered by satellites. The discipline that studies and analyzes this information is called **Geospatial**

• E-mail de contacte: oriol.casasc@e-campus.uab.cat
• Menció realitzada: Enginyeria del Software
• Treball tutoritzat per: Daniel Ponsa Mussarra (CVC)
• Curs 2019/20

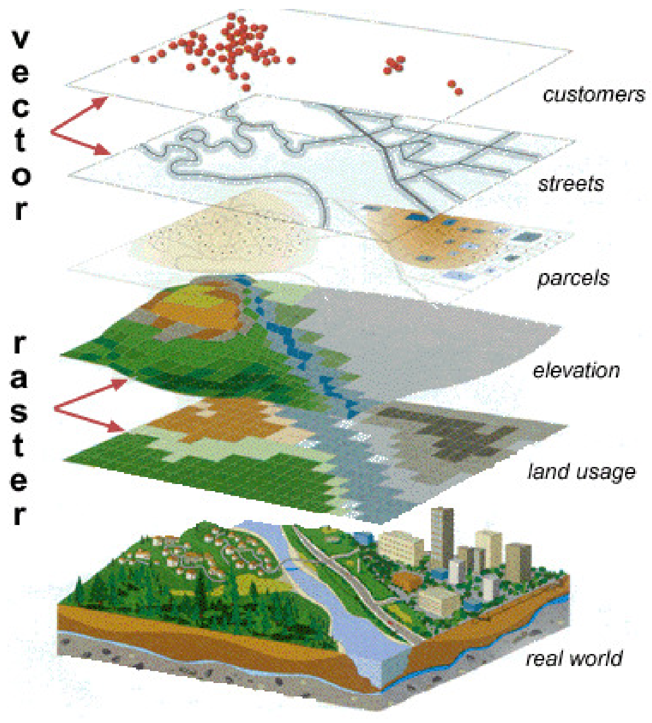


Fig. 1: GIS layers example. Source: [2]

Imaging[5]. GIS tools can analyze this data and combine different types of imagery.

Today's research is very active in proposing new approaches to analyze this data. These new methods are not available in GIS programs. In fact, most times the processing is made in external tools. Due to the large amount of data and the new computationally expensive techniques, the processing is done in calculation servers with parallel computation capacities.

Being able to connect QGIS to this cloud processing in a comfortable way is very interesting, and in fact, this is what motivated this project.

The group MSIAU has developed Machine Learning algorithms to be used in this context. Therefore, a system that combines the power of QGIS and their algorithms is desirable.

The section 2 covers the essential concepts related to spatial data processing and the tools and libraries most commonly used for its analysis. Once the context is explained, the objectives to be achieved in this project are set out in the section 3. To meet these goals, the methodology that has been followed is described in the section 4, where all the details of the steps taken and their purpose are given. The main milestones of the project are highlighted in the section 4.2. The results obtained, together with a brief description of the tasks performed, are set out in the section 5. Some tasks that could not be completed are explained in the subsections of this section. Finally, the conclusions are put together with work to be done as a continuation of the project in the section 7.

2 State of the Art

To clarify the context of this TFG, some concepts are explained below.

The first concept is **GeoAI[6]**, which is the combination of artificial intelligence like Machine Learning with geospatial solutions like QGIS.

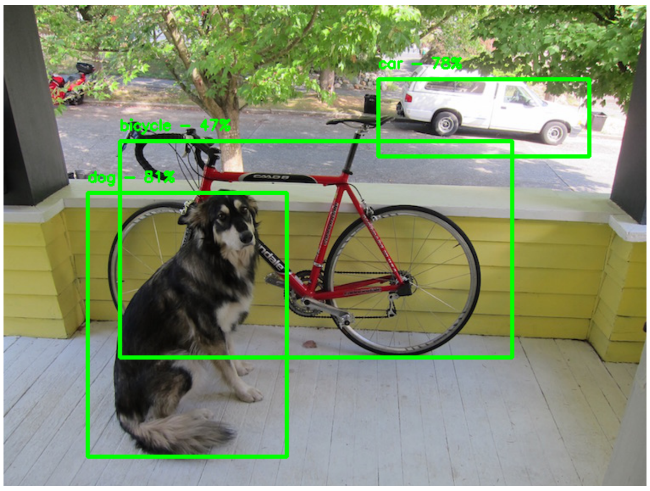


Fig. 2: Object detection example

Another basic concept is **Machine Learning[7]**, the ability of artificial intelligence to improve from experience and learn automatically. To do so, a training process is required. The algorithms build models from samples which will define their behavior and decision making.

The main types of machine learning are listed below:

- Supervised machine learning: the system needs an existing set of inputs and their desired outputs to do the training. This way, the set of data which is already tagged, the training set, will be used as the model to classify future incoming data.
- Unsupervised machine learning: in this case, there is no labeled data to do the training. The training set consists of inputs and the system has to explore the existing data, for example by grouping and clustering every feature available, to find the structure and define the models by itself.

The following list details some of the most important use cases of Machine Learning that are more commonly applied in GIS[8]:

- **Object Detection:** task that identifies objects of a certain type and its location in the image. In a society where there are systems that constantly take images, it can be really useful in fields like video surveillance or social studies. See Figure 2.
- **Semantic Segmentation:** task that classifies every pixel of an object in the image in a particular class. This way, all the objects in an image can be classified by its type and can be differentiated from the others. See Figure 3.
- **Instance Segmentation:** task that mixes the previous two. It identifies each object in the image in a pixel level. The boundaries of each object are clear and the object is properly classified by its type. See Figure 4.

This use cases can be helpful but they need a mechanism to take part in a GIS software workflow.

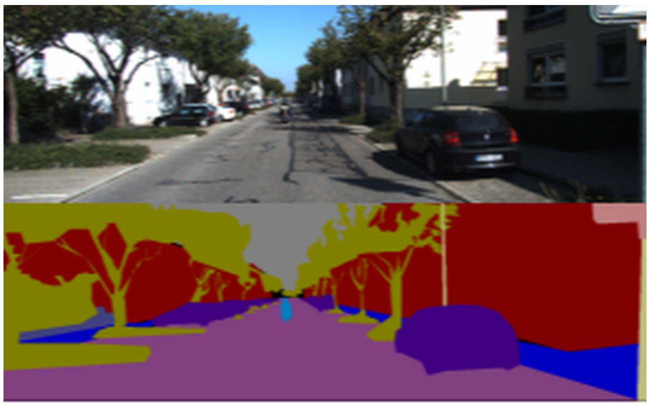


Fig. 3: Semantic segmentation example

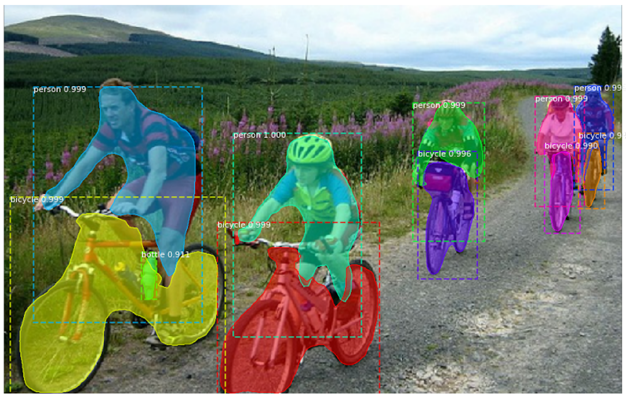


Fig. 4: Instance segmentation example

QGIS, the GIS software used in this project, has lots of plugins and a great community. Plugins can be created using Python and Qt. The pros of this program are that it is free and open plus a plugin is needed in order to use the already developed processes. Some frameworks let the user apply some Machine Learning.

Orfeo ToolBox[9] is an open source library build over ITK, a popular C++ library, that offers the user a wide range of AI applications and can be used in most Operating Systems. Between its applications, we find supervised and unsupervised machine learning. The processes that were interesting for this project. Some of the tools it offers once it is added to QGIS have been quoted as important for this project before. It can classify and segment objects in images. Despite these facts, using this library would require the adaptation of the developed algorithms to fit them in the library.

Raster Vision[10]Raster Vision is another framework. In this case, it has been developed in Python. It has its own workflow with built-in support for chip classification, object detection, and semantic segmentation using Tensorflow. Again, the developed algorithms should be adapted to this framework.

Both of them have builtin functions to apply Machine Learning but the code should be rewritten which is unwanted.

In the case of ArcGIS, it does not need any third-party solution as it already provides the necessary tools, ArcGIS API for Python, to help every step of the Machine Learning workflow so it would be easier to insert these processes but once again, some code should be rewritten and also the

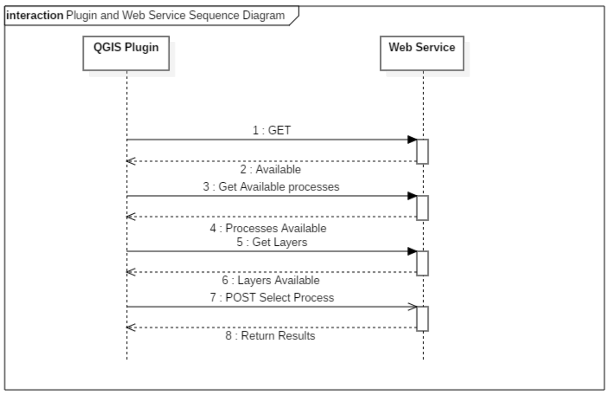


Fig. 5: Sequence diagram of the connection between QGIS plugin and web service

licenses should be avoided.

ArcGIS API for Python is a built-in API for ArcGIS. It comes with lots of tools that eases tasks as downloading data or importing layers. This tasks can be accomplished by a plugin in QGIS.

3 Objectives

The goal of this project is to build a plugin for QGIS that lets users connect with a web service with built-in functions that apply Machine Learning on the layers selected by the users. We also want to avoid rewriting the existing code and make a good use of QGIS and its power alongside these services.

The QGIS plugin must establish the connection with the web service and shall send the necessary information so that the service can perform the desired process with its params. Once the response arrives, the plugin has to create a new layer where the results will be shown.

On the server side, a web service is needed so it can launch the processes as the plugin requires. It also has to answer requests with information about available processes and layers.

Figure 5 shows the sequence diagram of the connection between QGIS plugin and web service. The messages 5 and 6 are optional as they are only called when the user wants to use a layer from the server, not from QGIS.

In order to achieve this, some specific objectives must be reached and are listed below:

- Implement a web service to provide access to image processing algorithms.
- Establish a protocol to connect the QGIS plugin with the server.
- Refer the results pixels, points or areas with the proper geographical coordinates.
- Retrieve all the services available in server.
- Send data between QGIS plugin and web service and vice versa.
- Retrieve the results and show them to the user.

4 METHODOLOGY

The product was not clear from the beginning but the contact with the client and final user was established. An iterative and incremental flow was followed so the product could be defined and the stakeholders could check that it was appropriate as the project moved forward.

In order to accomplish the objectives, the project has been split into three phases.

The first phase was focused on gathering information and requirements from the final users. At the same time, some investigation about QGIS and its frameworks, the way they work and how to develop a plugin was done. Also, the Minimum Viable Product was defined here. The planning had to be done at this stage although revisions were made because of the iterative flow.

The 2nd and 3rd phases are the ones where the plugin and the web service have been implemented. The second being more focused on the MVP development. And the third one was thought as a phase to end the MVP and complete the product.

The QGIS Plugin and the web service had their own repository created in Github, a control version tool. Every time a milestone was completed or a feature was developed, the changes were pushed. This way, if the development of the next feature broke the software the changes could be reversed easily.

Other tools used for the development have been Qt Creator to implement the Graphical User Interface of the plugin. This program was chosen because there is an existing plugin that creates the bases of new plugins and uses ".ui" files.

In order to implement the Python code, Visual Studio Code was used. It is a powerful IDE that can be used for multiple programming languages and formats in a single tool. This way, apart from developing in Python, the Dockerfiles and JSON files could be edited in the same application.

Lastly, the tools used for writing the reports and drawing diagrams were Overleaf, which is a LaTex editor. Grammarly, a tool that corrects spelling and grammar. And finally, StarUML was used for the Use Case and Sequence diagrams, although some figures were drawn in Draw.io and the user story map in Miro app.

4.1 Minimum Viable Product (MVP)

The **Minimum Viable Product** is the product that has the minimum features implemented to consider it is valuable for the client. The MVP is used as a reference that proves the product can be developed and sets the base for its development.

For this project, the MVP has been divided into two parts as it is a two-sided system. On one side, there is the server which will launch the process and return the results. On the other side, a plugin for QGIS is needed in order to establish the connection and communicate with the server. It must do the right requests at the right moment.

4.1.1 MVP - Web Service

The service should be up and running, accepting connections and responding to them. This service must be able to manage JSON files as the petitions and process information are using this format. The processes and layers are not being stored in a database but in the server's folder.

Apart from returning its availability, the server must respond to requests asking for which processes and layers are available. For the MVP, only two processes are available: save layer and load layer. On the server side they work as an example for core functions as saving files sent from the plugin and sending server's files to the plugin.

Finally, the web service must be able to build and launch Docker containers from Dockerfiles stored in local folders.

4.1.2 MVP - QGIS Plugin

The plugin should be able to connect to the server and check if it is available. It is responsible to establish the connection and show all the information the user needs.

This information includes the available processes from the web service. The plugin guides the user making the drop-down menus available every time a new option, the process selection or the layers available, is selected. It also should give the user the possibility to abort the current process and show in which state it is.

Once there is an output or result, the plugin must create a new layer where the results are visualized and it should link the pixels to their coordinates properly.

4.2 Planning

In this section the main milestones will be explained in Table 1. The planning was based in the three phases of the project, being the second one the most important.

The planning has been modified multiple times due to blocks in the development. These blocks have ended in major delays that have affected the final result of this project.

The second phase had to be extended a couple times causing the delay of the third one which could not be started at the end.

Also personal timetables changes have had a great impact in the development of this project.

4.3 Phases

4.3.1 Requirements and design iteration (1st Phase)

In this first iteration, the purpose was to define the product, understand which were the purposes it should have and design a product that fits the needs of the stakeholders.

The first steps were to design the interface, the first sketches of the plugin were made and with the help of the stakeholders, these sketches better defined. Also, the diagrams and specs were drawn and written at this phase.

The stakeholders were interviewed twice to plan how the plugin would be more user friendly, which features are useful and which ones are not. Firstly, a user story map was done and explained to the stakeholders in the first interview. This way, we put together the idea we had of the plugin and some ideas were exchanged. After this first interview, the sketches were done and discussed in a second interview which was useful to set the design. The information needed to be processed and how to display it was defined. This interview also was useful to see if the design of the plugin was clear for them or should be sketched another way.

Milestone	Explanation	Date
MVP Definition	This is the first milestone, it defines the basics that should be accomplished in this project.	28/10/2019
MVP implementation	The plugin should have the MVP implemented.	28/11/2019
Web Service running	The server must be running properly locally and the basic functionalities should be available and working.	27/12/2019
Plugin finished	Now the plugin should be finished.	03/01/2020
MSIAU's server running	The web service should be running properly in the group's server.	10/01/2020
Final report	The final report must be ready.	26/01/2020
Presentation	The presentation must be ready before the date to defend the project.	09/02/2020

TABLE 1: TABLE THAT SHOWS THE MAIN MILESTONES

The **MVP** was defined in this phase as it is the prime objective of the project. When the user story mapping was done, its purpose was to have a better understanding of the system, which features should be included in the MVP and which don't. The user story mapping (Appendix A.2) was done using post-its and an online app to have a better looking and clearer image.

The requirements and restrictions of the system were not only taken from this user story mapping. The interview was helpful for the design of the plugin and which features were most useful. Finally, an use case diagram (see Appendix A.3) was also done in order to complete the whole concept of the project and the establishment of the MVP.

In this project no formal specifications documents have been generated as only one person has developed the system. The requirements are defined by the user story mapping, the use-case diagram and plugin sketches (see Annex A.4) that gave the general idea and the goals of the project as they varied as the project moved forward.

Once these documents were settled, the tasks and features that should be included in the MVP, the ones that have more value for the users. A viability analysis had to be done to check that all the required functionalities could be implemented.

Apart from the plugin, designing a proper strategy to communicate with the web service was basic. Different strategies were evaluated.

Firstly, an SSH communication solution was evaluated, it would mean just sending files between plugin and server, but it was dismissed as it required that the plugin should do a great part of the process management, which functions should be called and when.

So the alternative was to mount a web server. This way, the plugin should only do petitions and manage the responses. The server is responsible to supervise the process in the server part, the whole workflow done in this side.

To implement the web based solution, we used **Flask**, a framework that uses Python like the rest of the plugin which makes the language ideal and avoids the need to use different languages for a single project.

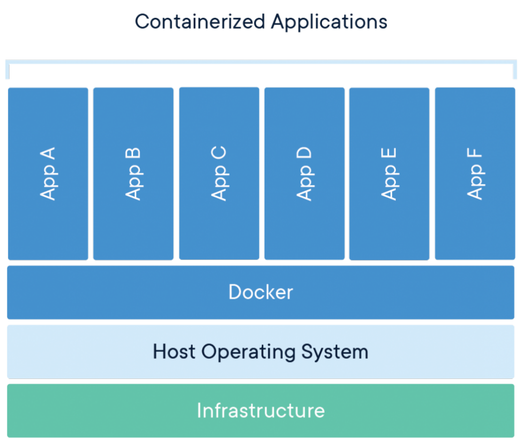


Fig. 6: Docker containers

Docker[11] is a powerful tool that creates containers on a virtualized machine. The containers run on this machine in separate environments where applications are run without sharing any information with other containers. This way, multiple petitions can be done and they will not overlap other processes' data.

It is used to isolate different processes and applications. Containers and applications do not depend on the environment this way and they do not share information across them. Taking the example of the Figure6, the operating system is not specified as it can run over multiple OSs. Over the OS, Docker mounts its containers. These containers do not share any information between them. For example, App A does not have any information nor access to App B. These facts make Docker an ideal service to deploy a web service

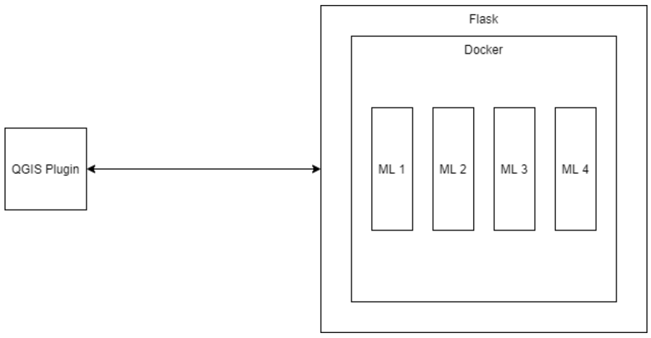


Fig. 7: System Overview

that can be easily scalated.

For this project, Dockerfiles for every process were done. These Dockerfiles contain the steps Docker should take to build the images and which commands should be executed once the container is running.

So the concept of the system is a QGIS plugin that communicates with a web service as seen in Figure 7. Flask is the one responsible to build the Docker images which will be mounted on Docker containers. These containers are launched in parallel and they are represented by ML1, ML2, and so on.

4.3.2 MVP implementation (2nd Phase)

This phase is focused on building the bases and core of the project, making the new plugin and implementing the first functionalities of the web service.

To build a QGIS plugin we need QT for the graphical interface and Python to develop and implement the functionalities.

The design of the plugin was done in QT following the sketches done in the first phase. Some buttons and menus were not implemented in this phase as they do not belong to MVP and could be misleading. These functionalities were related to giving extra information about the processes and layers which are not essential.

Also, the first implementation of the web service was done. Flask was run locally. For the MVP basic functionalities as saving or loading layers were used. This way, we had the basics of the communications between the plugin and the service.

All functionalities should had been tested by the end of the phase and the possible bugs solved.

4.3.3 Finishing the product (3rd Phase)

Once the MVP was finished, it was time to implement the extra features. The ones that better describe more the processes and their needs, as well as having the web service implemented and running in MSIAU's server.

By the end of this iteration, the plugin should have been finished and running without any issue alongside the server.

Before the end of January, the iteration had to be closed and the documentation ready.

5 RESULTS

This section reviews the system components that have been successfully implemented and tested.

The MVP's functionalities were the first elements to be completed. MVP has functionalities to be implemented on both sides of the architecture.

On the server side, the web service should be able to retrieve any request and process it. In order to test the connectivity, the first function to be developed was related to request its availability. Not only it returns a 200 OK HTTP response but its content is the text ”Available”. This response was made this way so any request to this function could know that the server is reachable, the 200 response, and that the service was called properly thereby the ”Available” response.

The next step to do in the server was to respond which processes are available. To manage these processes, a JSON file was created with every process available and information about it as well as an id for each one. For each process, a folder was created and the Dockerfiles to build their images were scripted. In addition to the Dockerfiles, a script in python was created with the flow to be followed by each process. This script is launched once the container is running.

If the selected process is ”load a layer”, the service shall return the available layers. Another JSON was created with the layers and their information. Both JSONs must be edited manually, but in the case of layers, there is a use case where the file is automatically edited. When a layer is saved, the process is responsible for updating the content.

Finally, the most obvious functionality was to run a process. The web service processes the request and gets all the information needed, which process should be launched and all the necessary parameters to run. It is also responsible for saving and managing the layers involved in the process. In the same function, the docker images are built and the containers launched. A binding between the web service and the docker container is made in order to communicate both of them. This way, the process in the container has all the parameters needed to run and can inform the service about the gotten results.

Before saving the files, the web service checks if the file is retrieved and if the extension is an expected one. The system has been designed to be only capable of retrieving zip or geotiff files in the server side.

An extra functionality was created. It launches a simple ubuntu image in docker which will print ”Hello World” on screen. It was useful when Docker was incorporated in the development to test if it was running as expected.

In order for the plugin to have the functionalities implemented, in Qt the first dialog was designed to link the functions to be developed to the corresponding elements. In Figure 8 the main window of the plugin is displayed. The drop-down menus and a text field for the server address are available. Also, a label is displayed to indicate that the plugin is connected to the server or not.

Once the plugin verifies if the communication is established, it request which are the available processes and shows them to the user using a drop-down menu.

This drop-down menu enables the layers menu. The plugin knows when to show QGIS' project layers or the layers from the server.

The label changes its color to green when the connection is established. Also the first option of each drop-down is automatically selected if they are enabled. See Figure 9 as

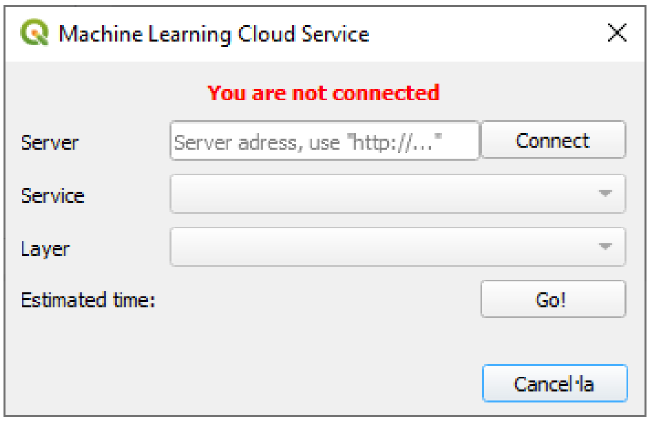


Fig. 8: Plugin's main dialog

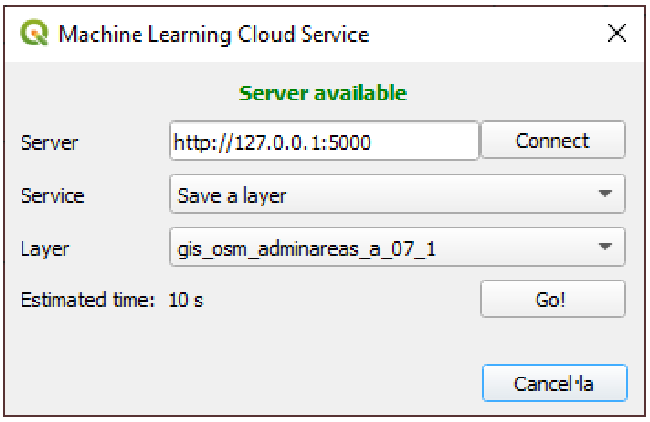


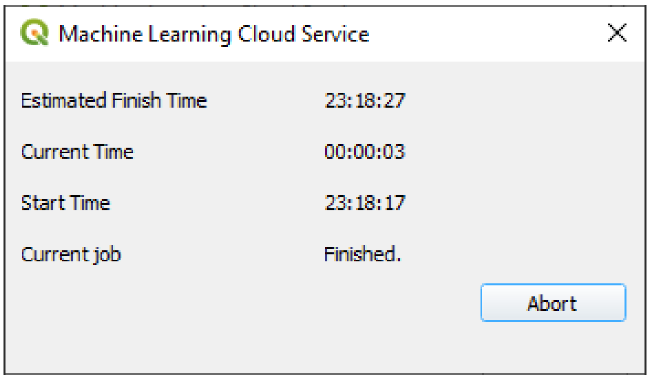
Fig. 9: Plugin's main dialog when filled

an example of this state.

In order to show the progress, another dialog is launched, see Figure 10. This one, indicates timing information as well as the current task that is being done. A button to abort the process is available too.'

Once the layer from the project is selected, depending on if it is a raster layer or a vector one, the plugin writes it to a temporary directory. Note that if it is a shapefile, more than one file is created. So, in order to send them, they are compressed in a zip file.

Lastly, the plugin receives the response and creates the required layer to show the results.


Machine Learning Cloud Service

Estimated Finish Time 23:18:27
Current Time 00:00:03
Start Time 23:18:17
Current job Finished.

Fig. 10: Process dialog

5.1 MVP analysis

This subsection will analyze which features have been accomplished and which ones have not.

Firstly, the web service side is almost complete. The MVP has been accomplished on this side. Although the docker containers are not fully independent as they share files and folders directly from the server there is not any feature missed. It is not easy to prevent the container and the web service from sharing directories as the containers are isolated and sending and receiving information from them is not a very desirable use case when you want them to be independent.

It also has security mechanisms and functions related to treating the files received in order to avoid risky extensions to be opened or executed in the server. The service is missing an admin so nobody can manage the processes JSON and the folders created to store their information.

Lastly, the web service has been developed in the local machine where the system has been developed. This way, the logs, outputs, and inputs could be easily accessed and eased the development. There has not been time to implement the service in the server as it was desired.

On the other side, the plugin's MVP was developed at 90%. It establishes the connection with the server, the dialogs are fully functional, it can export the layers to send them and shows the necessary information to the user.

Although it creates new layers and can import the shapefiles from the server. Geotiff files have been problematic and the function to import them is not functional. This causes the pixel binding to real-world coordinates cannot be checked.

The plugin establishes the connection to the server properly, although it could have issues if the server needs the user to log in.

5.2 Problems faced during the development

The very first problem was the fact that the GIS software was unknown and I have never worked with them. I had to learn how they worked, understand them and create a plugin for one of these systems. Some terms were unheard-of before this project.

At the same time, I had to understand the purposes of the project, what should be done: it was not easy as I hardly had some interaction with the software nor the university group at this point. Thanks to the first interviews with the tutor and the group the understanding and purposes were clearer and the project was being better detailed.

These first points led to insecurities and not knowing which steps to take. Which features should be developed or how the plugin and web service would communicate. But as more and more information was given to me in addition to my research, these insecurities disappeared.

Once the product was settled, the plugin had to be implemented. The hardest features have been related to files management and exporting different layers is also a hard task if the necessary knowledge about QGIS Python library's classes and methods. First of all, the shapefiles had to be packed in a zip in order to be sent to the web service. The zip file and the layers were written in a temporary folder. The big issue here was to find out why the zipping caused problems and blocked QGIS. Finally, it turned out to be the fact that the zip file was being created inside itself and had to skip zipping zip files.

On the server side, Docker was the one causing major blocks. It started well, the test function that was developed was working as expected. The first built images did not print any output nor did they show a sign of being working. After hours looking into it, it turned out to be Dockerfiles' problems. The way they work and their commands were reviewed and the issue was found. Every command was launched at build time and the containers were doing nothing. Using Python was useful as Python has a library to integrate Docker in the system easily. It is one of the only three languages supported officially by Docker. A reason to use this framework before another one like Java Spring, a powerful framework that is widely used to deploy Java applications in a server but it is supported unofficially.

Python is a very useful language, it can be used both in the plugin and the web service which allows the developer to use a single language and be able to test the whole workflow easily. It was a little unexpected that the server side was so comfortable to develop in Flask and Python.

6 Future work

On the server side, some mechanism should be developed in order to avoid multiple processes overlap one another. As it is at the end of the development, if two users call the same service, the JSON would be overwritten and end in a wrong result. The issue here is that there is not an easy solution to get and set data between the docker container and the service.

Additionally, the server should be managed by an admin. The JSONs and images have to be modified in the server and there is not an existing function to manage it. This feature should be implemented only granting access to the admin. The admin also could create some cron jobs to cleanup the files that are not needed anymore in the service as the processes leave files in the server that are not useful ones the process finishes and the results are sent.

And last but not least, the plugin needs some fixes too. There are visual issues and not all the objectives have been accomplished.

The requests should be launched in a background asynchronous thread in the plugin. This way the process dialog would show the details while the process is running. Also it would make the "abort" option available to the user in the same dialog.

Another feature that came up in an initial final product idea was implementing an option that would show the user a table of the processes and their description.

This initial final product idea also included a feature to partition the layer to process. In some cases, the layer needs more precise dimensions or resolution for the developed processes and it would be helpful a tool to partition it automatically.

Lastly, the plugins should relate a pixel from a raster layer to its real world coordinates. The lack of the functionality that adds a raster layer aggravates the test and implementation of this feature.

7 Conclusions

In order to meet the goals, the first step was to collect the specifications of the system the final users require. This way, a product that fits the user needs could be proposed.

To do this, a client-server architecture was designed to communicate QGIS with an external web service that launches the different processes developed independently of the operating system.

In order to develop the MVP on the plugin side, we have been able to export and import different types of layers. It is also capable of sending and receiving these layers to the server. There was not enough time to implement all the functions to make the plugin create all types of layers as it can not import raster layers.

On the other hand, on the server side, the MVP has been fully developed. It can send and receive files from the client and process them properly. In addition, it is also able to identify what type of file it receives and whether it should process it or not.

Acknowledgement

I want to thank Daniel Ponsa for his help as tutor. He has aided and guided me every time I needed guidance and made all the possible in order to keep the project going. I also want to thank MSIAU's researchers for their advice and ideas for the plugin.

This work has been done with the support of the BOSSS project (TIN2017-89723-P).

References

[1] GIS Cooperative and Fort Collins. "The unique qualities of a geographic information system: a commentary". In: *Photogrammetric Engineering and Remote Sensing* 54.11 (1988), pp. 1547–1549.

[2] TERC Inc. *Eyes in the Sky II*. visited on 2020-02-06. Nov. 2016. URL: https://serc.carleton.edu/eyesinthesky2/week5/intro_gis.html.

[3] *QGIS*. visited on 2020-01-29. 2019. URL: https://qgis.org/en/site/.

[4] *ArcGIS*. visited on 2020-01-29. URL: https://www.esri.com/en-us/arcgis/about-arcgis/overview.

[5] J.D. Bossler et al. *Manual of Geospatial Science and Technology*. CRC Press, 2010. ISBN: 9781420087345.

[6] K. Jansen and R. Parsons. *GeoAI: Feature detection and classification*. visited on 2019-11-05. URL: https://orbica.world/uploads/files/GEOAI-A5.pdf.

[7] Unknown. *What is Machine Learning? A definition*. visited on 2019-11-05. Expert System, Mar. 2017. URL: https://expertsystem.com/machine-learning-definition/.

[8] R. Singh. *Integrating Deep Learning with GIS*. visited on 2019-10-05. Medium, Feb. 2019. URL: `https://medium.com/geoai/integrating-deep-learning-with-gis-70e7c5aa9dfe`.

[9] *Orfeo ToolBox*. visited on 2019-12-11. 2019. URL: `https://www.orfeo-toolbox.org/`.

[10] *Raster Vision*. visited on 2019-11-25. 2018. URL: `https://rastervision.io/`.

[11] *Docker*. visited on 2020-01-31. 2019. URL: `https://www.docker.com/`.

[12] O. Casas. *Story Map*. visited on 2020-12-06. Oct. 2019. URL: `https://miro.com/welcomeonboard/B8ITP1fPr1km3mP20pC5Er62xvO2qSCENtjOTqdTqflDuBkWsJqsC7a1t4zJHqb3`.

APPENDIX

A.1 Mean-Ends tree

The tree of the Figure 11 shows the mean-ends tree of the project. This tree was generated at the beginning of the elicitation to identify project purposes based on the needs to be met.

In the central box, we find the main objective. The upper boxes represent the needs to be met and the lower ones represent the goals to meet these needs.

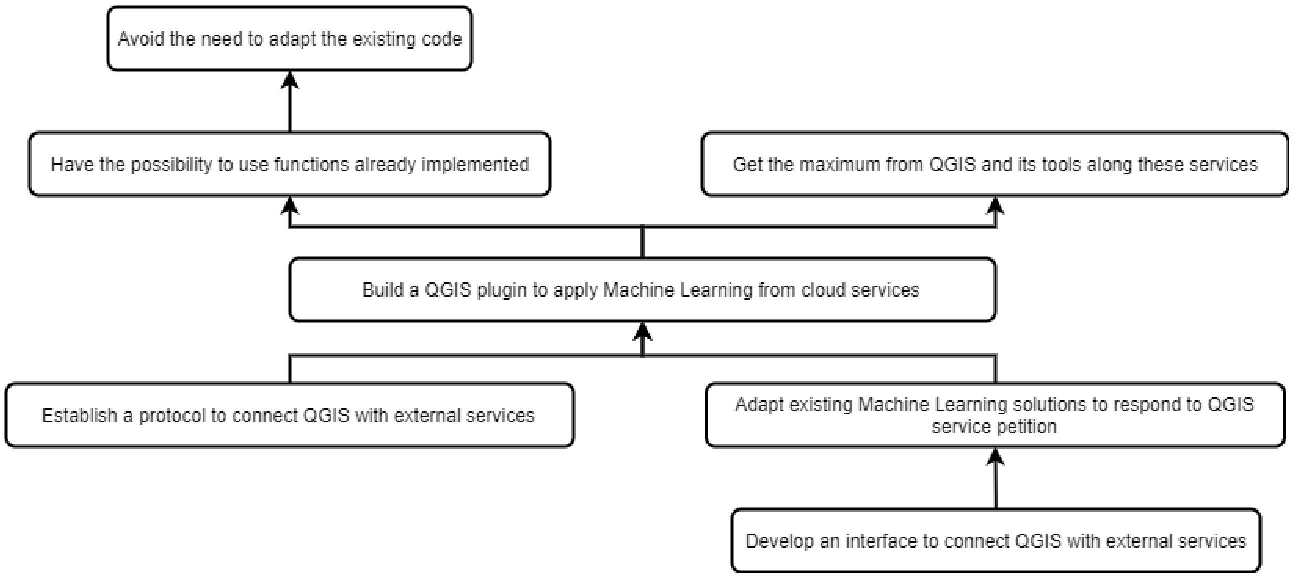

Avoid the need to adapt the existing code
Have the possibility to use functions already implemented
Get the maximum from QGIS and its tools along these services
Build a QGIS plugin to apply Machine Learning from cloud services
Establish a protocol to connect QGIS with external services
Adapt existing Machine Learning solutions to respond to QGIS service petition
Develop an interface to connect QGIS with external services

Fig. 11: Mean-Ends tree

A.2 User story mapping

The user story mapping was done in Miro app and can be found on the following link[12]. The first sketch was done with post its and can be seen in the Figure 12.

This user story represents the workflow of the system. Each task is represented in order and contains a list of what functionalities each task must have. These functionalities are grouped as indispensable or not and if they add value to the product to be developed. In other words, they are classified as to whether they should be part of the MVP or not.

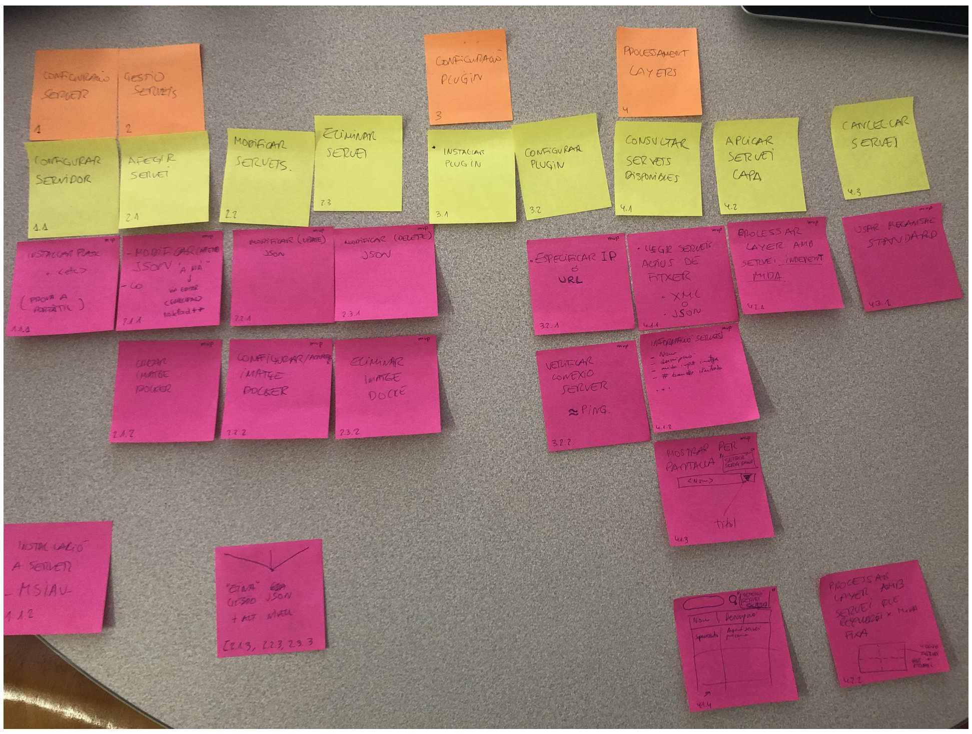


Fig. 12: User Story Mapping

A.3 Use case diagram

The Figure 13 shows the use cases of the project. It is really useful to define the MVP alongside the user story mapping.

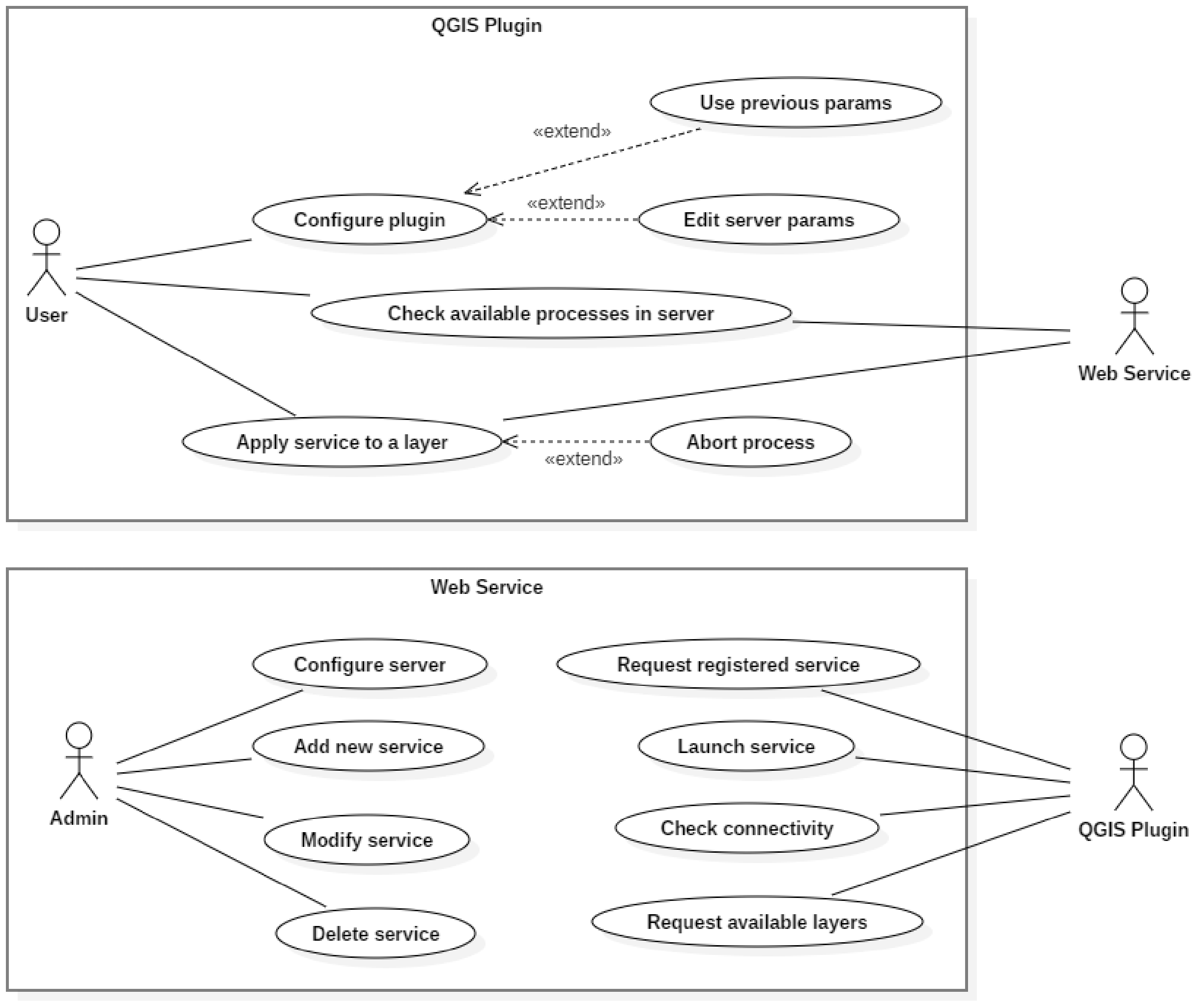


Fig. 13: Use cases Diagram

A.4 Sketches

The Figure 14 shows the first sketches done. They represent the first idea of how the GUI of the plugin should be. They were shown to final users, and with the help of these, the interface was redesigned.

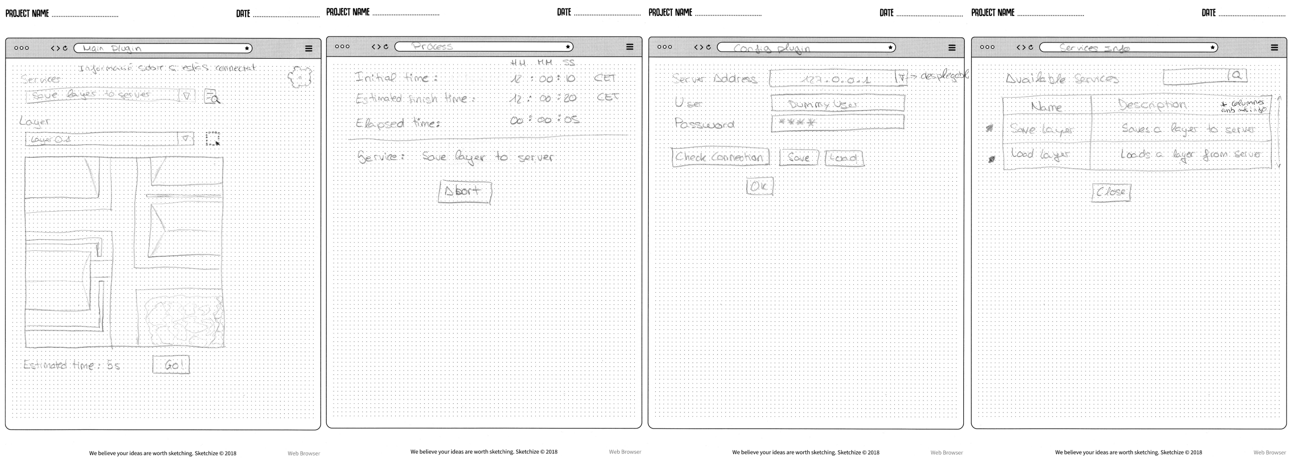


Fig. 14: First sketches done