Towards a new benchmark for background subtraction algorithms in computer vision

Auteur : Provoost, Dylan
Promoteur(s) : Van Droogenbroeck, Marc
Faculté : Faculté des Sciences appliquées
Diplôme : Master en sciences informatiques, à finalité spécialisée en "intelligent systems"
Année académique : 2023-2024
URI/URL : https://github.com/pierard/BGS-Recommender-Platform; http://hdl.handle.net/2268.2/20477

LIÈGE université
Sciences Appliquées

University of Liège
Faculty of Applied Sciences

Implementation of a recommendation platform for a computer vision task

A Master Thesis submitted in order to obtain the degree of Master in Computer Sciences

Author
Dylan Provoost

Supervisor
Marc Van Droogenbroeck

Academic Year of 2023-2024
June 10, 2024

Abstract

Background subtraction is the initial step in many computer vision applications. As such, choosing the most appropriate algorithm for a given task is crucial as it can significantly impact the performance of the entire system. However, the selection process is often challenging due to the large amount of available algorithms, and the lack of standardised benchmarks. Traditionally, researchers and industry professionals have relied on paper surveys and tools like ChangeDetection.net (CDNet) to identify potential algorithms. This thesis attempts at addressing these limitations by proposing a novel Background Subtraction Recommender Platform, a scalable, extendable, and modular web-based system that adapts to the user's input to recommend the most suitable algorithms for his needs, abstracting away the complexity of the selection process. The main goal of the application is to provide a contextualised ranking of the algorithms, and to streamline the algorithm development process by allowing the submission and execution of new algorithms while providing insights on their performances. To do so, this solution leverages state-of-the-art algorithm evaluation procedures. Additionally, contextualisation of the recommendation process is guided by semantic segmentation algorithms to put emphasis on the content of the video sequences. The ranking system is evaluated for a set of algorithms obtained from the BGSLibrary on real-world data built upon the well-known CDNet dataset, demonstrating its ability to provide meaningful recommendations to the user.

Acknowledgements

First of all, I would like to thank my supervisor, Professor Marc Van Droogenbroeck for his guidance and advices which allowed me to lead the development in the right direction. I am also grateful for his course on computer vision, which provided a strong foundation for the core concepts of this thesis.

Next, I would like to extend my gratitude to Sébastien Piérard and Anthony Cioppa for the invaluable feedbacks they gave during our meetings that allowed me to improve the overall quality of the project.

I would like to thank again Sébastien Piérard who not only supervised my work thorough the entire project but also for his knowledge about algorithm evaluation methods which helped me a lot during the development of the ranking system.

I am also thankful to Anaïs Halin for her help and advices on semantic segmentation algorithms which lead to significant improvements in the recommendation process.

Last but not least, I would like to thank my family and friends who supported me and helped my reflection during the various stages of this thesis.

Contents

Chapter 1

Introduction

The choice of an algorithm is a crucial step in the development of a computer vision, as it can have a significant impact on the application's performances. However, choosing the algorithm that is best suited for a situation is no easy task. In fact, the performance of an algorithm can greatly vary depending on factors such as data (i.e. the nature of data, size of the dataset, etc.), the task at hand, but also on the evaluation process being used. This process typically involves the computation of numerous scores that will be used to compare a set of algorithms on a dataset. Nevertheless, the choice of scores, and their interpretation will vary depending on the specific applications that are targeted. For example, an algorithm designed for surveillance cameras that detect all moving objects will likely be preferred over one that misses intrusions, even if it means producing false alarms. The nature of the dataset is also a key factor for the recommendation, especially for machine learning based algorithms, as some algorithms perform better in certain scenarios than others.

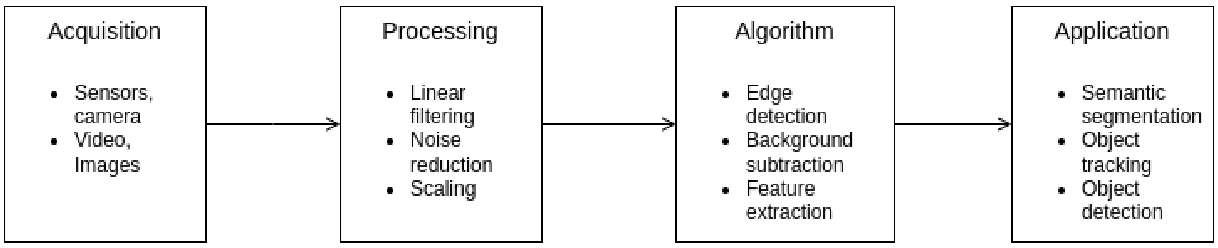


Figure 1.1. The four main areas of computer vision.

Computer vision is a field that is in constant evolution that can be defined by four main interconnected areas (see Figure 1.1):

- **Acquisition**: this area focuses on the way the data is captured (i.e. cameras, sensors, etc.),
- **Processing**: relates to the way the data is processed (i.e. filtering, morphological operators, etc.),
- **Algorithm**: the algorithm that is used to extract information from the data. This can be achieved through classical image processing techniques (edge detection, watershed, etc.), or via data-driven methods (machine learning, deep learning, etc.),

- **Application**: the way the information is used. This corresponds to the specific task at hand (i.e. object detection, tracking, recognition, etc.). It also incorporates the evaluation dimension.

As a result algorithms are regularly being introduced, which makes it difficult for researchers and industry professionals to keep track of the latest advances. Moreover, the evaluation of algorithms is a time-consuming task that requires a good amount of quality data. This is especially true for tasks like semantic segmentation, or background subtraction (BGS) that requires pixel-wise masks in order to be accurate. In that effort, a few video datasets have been introduced in the past years. Notable benchmarks include DAVIS challenges for video object segmentation [1, 2], the MOT challenges for multiple object tracking [3–5], and the ChangeDetection.net (CDNet) benchmark for background subtraction [6, 7].

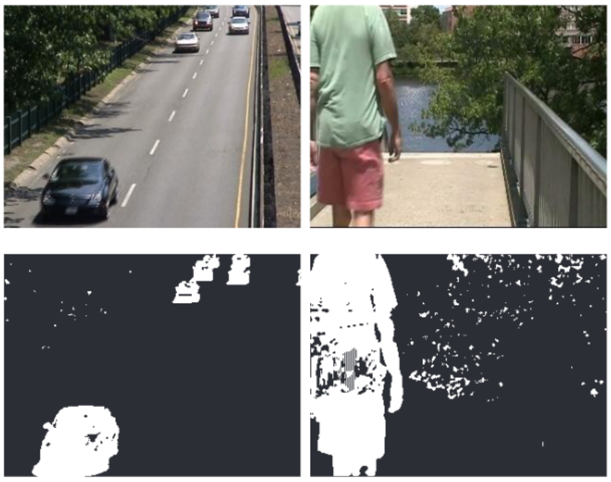


Figure 1.2. Example of a foreground extraction. The top image is the original frame, the bottom image is the foreground mask. The white pixels represent the foreground, and the black pixels represent the background. Source: [6].

Background subtraction (also known as foreground detection) is a fundamental task in the computer vision field. The background subtraction (BGS) is a key step in many applications such as surveillance, traffic monitoring, gesture recognition, and more. It consists in detecting changes in video sequences, specifically separating moving objects (called foreground) from the static background. Although the concept is simple, the definition of what constitutes the foreground can be fuzzy, and can vary depending on the specific application. For instance, waving leaves on a tree might not be desired in the foreground for a surveillance application. The setting of the videos can present challenging situations such as jitter, moving camera, noise, shadows, illumination changes, occlusions, camouflage, etc.

Consequently, it makes it difficult (if not impossible) to design an algorithm that works perfectly in all possible scenarios. Most algorithms are usually strong in some settings but weak in others. Therefore, it is necessary to try different methods and compare them in order to find the best fit for a given situation. Traditionally, this has been made possible through the use of benchmarks. In the case of BGS, CDNet is a good example, it provides a substantial amount of videos captured under various conditions, each of

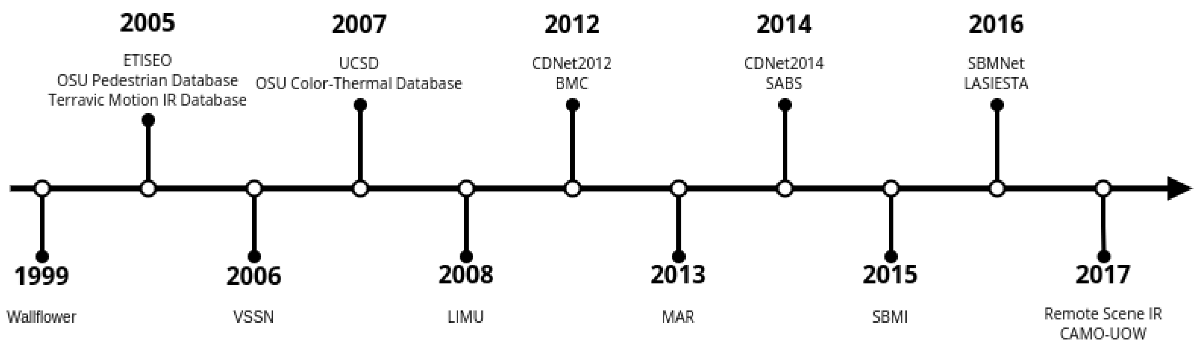


Figure 1.3. Timeline of the different datasets that were introduced in the past years.

them accompanied with ground truth masks that for the evaluation of the performance of algorithms. Moreover, it also provides a ranking system that compares the results of the submitted algorithms across several metrics and categories. This benchmark and its content will be further detailed in chapter 4.

This work is part of the ongoing effort to provide a better way to recommend algorithms for background subtraction. The thesis focuses on the application and algorithms parts of the computer vision process, its focus is to propose a new way to recommend algorithms for background subtraction that is not oriented towards sole performance metrics. Instead, the recommendation is made such that it can be understood by a non-expert user. In summary, the aim is to provide a flexible system that:

1. Provides a recommendation for an algorithm based on several factors (i.e. the content of the video, the challenges that are present, preferred output, hardware specification, . . .),
2. Allows researcher to upload their algorithms to see how they perform on the dataset,
3. Provide insight into how the algorithm would have been recommended to users based on a history of requests,
4. Can be gradually extended by adding new data (videos, categories, algorithms, . . .).

As a first building stone, this master thesis provides a prototype that uses the CDNet dataset[6, 7] and a selection of well known BGS algorithm in an effort to demonstrate the feasibility of such a recommendation system.

The present work is organised as follows. The next chapter describes the different concepts that are key to the understanding of this work. It covers concepts related to the background subtraction, recommendation systems (and more specifically algorithm recommendation), existing benchmarks, and techniques that were used in the implementation of the different modules of the platform. The third chapter presents the methodology that was followed in the realisation of this work, and a detailed description of the objectives of the thesis. The fourth chapter presents the realisation that were made for this master thesis. It contains a description of the dataset, the algorithms that were used for the evaluation, and a complete explanation of the platform that was developed. The fifth chapter describes the results obtained with the platform. Those are specifically geared towards the ranking system, and the use of semantic labels in the recommendation process.

The sixth chapter is a discussion of the results as well as the limitations, and possible improvements. Finally, the last chapter concludes the work and presents some perspectives for future work.

Chapter 2

Background

The present chapter is used in order to clarify a variety of key concepts of this report. This part talks about the concept of background subtraction, its challenges, applications, but also a brief overview of the basic algorithms of this technique.

2.1 Recommendation Systems

Recommendation systems (RS) can be viewed as a particular case of information filtering systems (IFS). An information filtering system is defined as a system that removes information before presenting it to a human user. Hence, preventing the user from being overloaded with irrelevant information. In order to perform this filtering, a set of information about the user is compared to a set of reference features.

The recommendation systems are mostly involved in decision-making processes where the user does not have a clear idea of what he/she wants. They are primarily directed at individuals who lack the experience or competence to make a decision about a potentially large number of items. Therefore, such systems must predict items that are worth recommending to the user.

2.1.1 Types of Recommendation Systems

In order to determine the utility of an item for the user, RS can be regrouped into several categories. Hereafter are listed a few examples of the most common types of recommendation techniques:

- **Collaborative Filtering:** This technique consists in the recommendation of an item to the user based on other recommendations made to user that have similar profiles.
- **Content-Based:** This technique relies solely on the user's profile in order to make a recommendation. The utility is determined by comparing the features of the items to the preferences of the user. This way, if the user has positively rated an item in the past, he'll be most likely to be recommended items of that have similar attributes.
- **Hybrid Systems:** This technique consists as a combination of other technique that aims at using the advantages of each techniques to mitigate their drawbacks.

2.1.2 Algorithm Recommendation

The recommendation of an algorithm can be assimilated to a type of content-based recommendation technique. Indeed, when looking for an algorithm to solve a task, one will look at its characteristics. It can be technical requirements such as memory usage (spatial complexity), speed of the algorithm (temporal complexity). However, the primary characteristic one seeks tends to be the performance it can provide regarding the problem to solve. The determination of those performances strongly depends on the type of problem the algorithm is trying to solve. For instance, for background subtraction the accent is put on the ability for the algorithm to correctly classify each pixel of the image [8].

2.2 Background Subtraction

Employed in the context of the foreground detection task, the term **background subtraction** refers to techniques that are commonly used to detect and isolate moving object from a video sequence. These techniques aim at generating foreground masks for a video. The masks are obtained using algorithms that take as input a background model and the current frame in order to produce a binary image as output. The produced image only contain pixel values of 0 (black) and 255 (white) where black represents the background (static objects), and white is the foreground (objects in motion). This can be assimilated to a classification tasks where each pixel of the image is labelled as either background or foreground. The main challenge of the algorithms is to be able to obtain (and maintain) an accurate background model that can be used to accurately detect foreground objects.

Classification The task of classification is the process of assigning a category or class to a new observation, given a set of training data that contains observations for which the category is known.

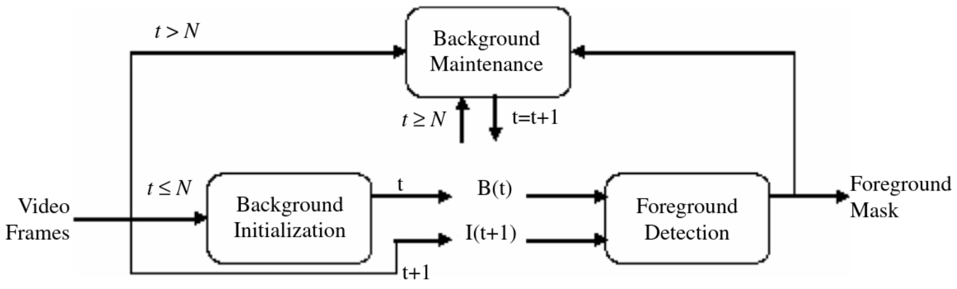


Figure 2.1. Steps of a typical background algorithm. In this diagram, N is the number of frames used in the initialisation of the background model. $B(t)$ and $I(t)$ represent the background and the current frame at time t respectively. Source: [9, Bouwmans]

.

When designing a BGS algorithm, one must consider a few aspects. As shown by Fig. 2.1, a typical BGS method is composed of three main steps: (1) the initialisation of the background model, (2) the detection of the foreground, and finally (3) the maintenance of the background model. The choice of model is therefore crucial in the performance of the algorithm as it will determine the complexity of the representation for the background. A typical choice is to use a Gaussian Mixture Model (GMM) [10] which allows for the representation of a multi-modal background.

Background initialisation The initialisation of the background model refers to the process of building a reference frame (or statistical model) that will be compared to the other frames of the sequence. The main challenge of the initialisation is to obtain a "clean" background model. That is, a model that is free of any object that does not belong to the background. This is often done by considering a few frames at the beginning of the sequence in order to construct the model. However, in real-world scenarios, it is rare to have access to a set of training frames that are completely free of moving objects.

Foreground detection Foreground detection is the process of comparing the current frame to the background model in order to retrieve a mask that represents the changes. As mentioned above, this amounts to a classification task where each pixel is labelled as either background or foreground.

Background maintenance The background model must be updated over time in order to adapt to changes in the scenes. Indeed, as mentioned previously, the video sequence can present changes in the background under some conditions. Making it required for the model to adapt to these variations in order to maintain an accurate representation of the background. Several strategies are used to update the model. The most common is to perform a blind update of the model regardless of the segmentation result. This is done by performing a moving average of the pixel values of the frame as shown in Eq. 2.1.

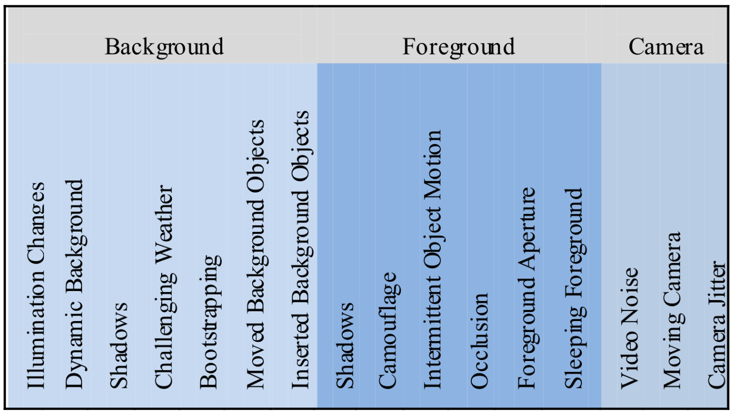
$$B(t) = (1 - \alpha) \cdot B(t-1) + \alpha \cdot I(t) \tag{2.1}$$

where α represents the learning rate. Another strategy is to update the model using the information obtained from the segmentation. This approach uses different learning rates for background and foreground pixels. However, this strategy may lead to inaccuracies in the background model in case of false detections. It is also possible to use a fuzzy update strategy where the update takes into account the confidence of the segmentation result.

As mentioned in the introduction, even though the definition itself is relatively simple, the definition of what constitutes the foreground objects (and the background respectively) tends to be blurry. Videos can present elements that are in motion but should not be considered as foreground (i.e. weaving trees). These scenarios can lead to false detection by the algorithms.

2.2.1 Challenges

According to Bouwmans (2014)[9], the good functioning of a BGS algorithm is dependent on three main factors: the movement of the camera, the illumination of the scene, and the movement of the background. In an ideal scenario the camera would be fixed, the illumination constant, and the background static. However, in reality, these conditions are rarely met. Datasets used to evaluate BGS algorithms are much likely to bring challenging situations that can severely impact the performance of the algorithms. Figure 2.2 shows examples of typical challenging scenarios. These situations can be related to the background, the foreground, or the camera itself.



Background							Foreground						Camera		
Illumination Changes	Dynamic Background	Shadows	Challenging Weather	Bootstrapping	Moved Background Objects	Inserted Background Objects	Shadows	Camouflage	Intermittent Object Motion	Occlusion	Foreground Aperture	Sleeping Foreground	Video Noise	Moving Camera	Camera Jitter

Figure 2.2. Summary of challenges encountered in the context of background subtraction. The challenges are regrouped in three categories: background, foreground, and camera. Source: Kalsotra and Arora (2019) [11]

.

Illumination Changes The illumination of a scene can vary over time. In outdoor scenes, it can be caused by the fast variations of the natural light caused by the sun being occulted by clouds (or the opposite), or simply by the gradual change in lighting condition due to the time of day. In indoor scenes, these illumination change can be caused by the switching on/off lights. These changes affect pixel colour intensities of the frames leading to false detections of foreground objects, and therefore to inaccuracies in the background model.

Shadows Shadows that are cast by objects in the scenes can be a source of confusion for the algorithms. Indeed, they can be misinterpreted as a moving object which causing false detections. This is especially true with hard shadows that have a high contrast with the background (i.e. shadows during in a bright sunny day).

Intermittent Object Motion Objects of the foreground can remain static for a short period of time. During that period, they are slowly integrated in the background model. This leads to ghosting artefacts when the object starts moving again.

Weather Conditions The weather conditions can have a significant impact on the detection of foreground objects. Poor weather conditions such as heavy rain, snow, or fog can lead to misclassifications of the pixels, increasing the number of false positives.

Camera Jitter Camera jitter refers to the small vibrations of the camera that can be caused by wind, or the camera being mounted on a moving object. Those vibrations introduce motion in the sequence which can lead to false detections without a robust maintenance mechanism [9].

Moving Camera The movement of the camera (i.e. sequences captured with embedded cameras, ptz camera, . . .) present a significantly greater challenge than static cameras. This is because the movement of the camera itself introduces motion in the entire background. The BGS algorithm struggles to distinguish this background motion from the actual foreground objects, making it difficult to maintain an accurate background model.

Dynamic Background The background of the scenes itself can be dynamic, containing elements that move and cause variation in pixel values (i.e. weaving trees, rippling water, etc.). These variations can be misinterpreted by the algorithm, causing it to misclassify background elements as foreground.

Camouflage Objects in the scenes can have similar colours to the ones of the background. This can lead to problems in the detection as the algorithms may not be able to distinguish the foreground from the background.

Noise Video sequences may have poor quality images. This can be caused by the camera itself, or by the compression of the video. This can lead to the presence of faulty pixels, introducing artifacts that can be misinterpreted by the algorithms.

Bootstrapping Initialising the background model can be a challenge. In some situations it is impossible to obtain a background image that is representative of the scene. This can be caused by the presence of moving object during the initialisation phase. As a result, the generated background model is not accurate and lead to issues throughout the video sequence.

2.3 Evaluation

According to Sanches *et al.*[12] the evaluation of a BGS algorithm depends on three key factors: the dataset used which contains a collection of videos on which the algorithm will be executed; the metrics (or scores) that will be used to evaluate the performances of the algorithms; and a set of state-of-the-art (SOTA) algorithms that will be used as reference to compare the effectiveness of the new algorithms.

As previously mentioned, a "good" background subtraction algorithm is one that is able to accurately maintain a representation of the background model such that it is able to correctly classify each pixel of the image. The evaluation of the performance of the algorithms is crucial, and can be either subjective or objective.

- **Subjective Evaluation:** This type of evaluation is based on the visual inspection of the results by human viewers, taking the user's experience and perception into account. However, this type of evaluation is time-consuming and provides scores that depends on the setup of the experiment,
- **Objective Evaluation:** This type of evaluation is based on measurements that are made on the results of the algorithms. These measurements involve the computation of several metrics/scores that will then be objectively compared in order to assess the performance of the methods.

BGS can be defined as a binary classification task, where each pixel of the video is classified as either foreground or background. This makes the evaluation of BGS algorithms identical to the one of classifiers. Formally, classification is defined by a ground truth class $y \in \{c^-, c^+\}$ which corresponds to the true category of the objects and a predicted class $\hat{y} \in \{c^-, c^+\}$ which is the prediction made by the algorithm.

From this definition can be derived the confusion matrix, a $n_c \times n_c$ (n_c is the number of categories, in this case 2) contingency table that summarises the performance of an algorithm based on four elements: the True Positives (TP) which are objects correctly predicted as positive, the True Negatives (TN) which are objects correctly predicted as negative, the False Positives (FP) which are objects incorrectly predicted as positive, and False Negatives (FN) which are objects incorrectly predicted as negative.

From the confusion matrix, several scores can be computed in order to evaluate the performance of the algorithms. Each of which provides a different aspect of the performance of the algorithms. The most common scores are the following:

- **Accuracy:** The accuracy is the ratio between the amount of correctly classified objects (TP, TN), and the total number of predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

- **Precision:** The precision, also called Positive Predictive Value (PPV), is the ratio between the number of correctly classified positive objects (TP), and the total number of positive predictions (TP + FP).

$$\text{PPV} = \frac{TP}{TP + FP} \tag{2.3}$$

- **Recall:** The recall, also called True Positive Rate (TPR), is the ratio of the number of correctly classified positive objects (TP), and the total number of positive objects (TP + FN). This score will be high when the algorithm does not incorrectly label objects as negative.

$$\text{TPR} = \frac{TP}{TP + FN} \tag{2.4}$$

- **Specificity:** The specificity, also called True Negative Rate (TNR), is the ratio between the amount of correctly classified negative examples (TN), and the total number of negative examples (TN + FP). This score will be higher when the algorithm does not incorrectly label objects as positive.

$$\text{TNR} = \frac{TN}{TN + FP} \tag{2.5}$$

- **Negative Predictive Value:** The Negative Predictive Value (NPV) is the ratio between the number of correctly classified negatives, and the total amount of negative predictions. It will be higher when the algorithm does not incorrectly label objects as negative.

$$\text{NPV} = \frac{TN}{TN + FN} \tag{2.6}$$

- **F1-Score:** The F1-Score is the harmonic mean of the precision and the recall. It is a balanced score which will be high when both precision and recall are high.

$$\text{F1-Score} = 2 \cdot \frac{\text{PPV} \cdot \text{TPR}}{\text{PPV} + \text{TPR}} \tag{2.7}$$

2.3.1 Summarization Procedure

The evaluation procedure proposed by benchmarks like LASIESTA [13], or CDNet [6, 7] typically involve a computation of the performance indicators (PI) for each videos. For the CDNet benchmarks, the PIs are first averaged within a video category, and then the resulting indicators are averaged between categories. This technique has the drawback of making the resulting indicators not interpretable, and to break the intrinsic relationships between the PIs.

To address this issue, Piérard and Van Droogenbroeck (2020) [14] proposed a summarisation procedure that defines a probabilistic framework which has the advantage of preserving the relationships between the PIs. The summarised indicator is presented as the weighted arithmetic mean of the indicators of each video $\{I_{\mathcal{A}|\mathcal{B}}(v)|v \in \mathbb{V}\}$, where the weights correspond to the normalised product between the importance given to the video v ($P(V = v)$), and the corresponding indicator $\mathrm{I}_{\mathcal{B}}(v)$. Formally, a probabilistic indicator can be defined as:

$$\mathrm{I}_{\mathcal{A}|\mathcal{B}}(v) = P(\Delta \in \mathcal{A}|\Delta \in \mathcal{B}),\ \text{with } \emptyset \subsetneq \mathcal{A} \subsetneq \mathcal{B} \subseteq \{TN, FP, FN, TP\}$$

, which represents the conditional probability of the outcome Δ to belong to the set $\mathcal{A}$ given it belongs to set $\mathcal{B}$ for a specific video v. Confusion matrix quantities can be expressed using unconditional PIs which are the proportion of each quantity in the confusion matrix.

For an algorithm, a score (i.e. Recall) can be obtained by computing the unconditional PIs for the whole dataset. These quantities are then plugged directly into the formulas to obtain the summarized indicator of a score.

2.4 Ranking Tile

The Ranking Tile is a crucial concept for the understanding of this work. Indeed, it is extensively used in the recommendation process of the algorithms. This concept was initially proposed by Piérard *et al.* in a work which is still unpublished at the time of writing this thesis. In this paper, they introduced a probabilistic framework that defines the notion of performance. From this framework, they derived an infinite, and parametric family of ranking scores that can be used to rank algorithms. The ranking tile itself is a graphical representation of the ranking scores. The computation of those scores is based on notions of Satisfaction (S), and Importance (I).

Satisfaction The satisfaction is defined as a random variable $S : \Omega \to \mathbb{R}$ that, in the context of two-class classification, takes its value in $\{0, 1\}$. Specifically, considering Ω as the set of possible outcomes of the classification, $\Omega = \{TP, TN, FP, FN\}$ in the context of binary classification. The satisfaction values are defined as follows:

$$S(o) = \begin{cases} 1, & \text{if } o \in \{TP, TN\} \\ 0, & \text{if } o \in \{FP, FN\} \end{cases} \tag{2.8}$$

, where $o \in \Omega$ is the outcome of the classification.

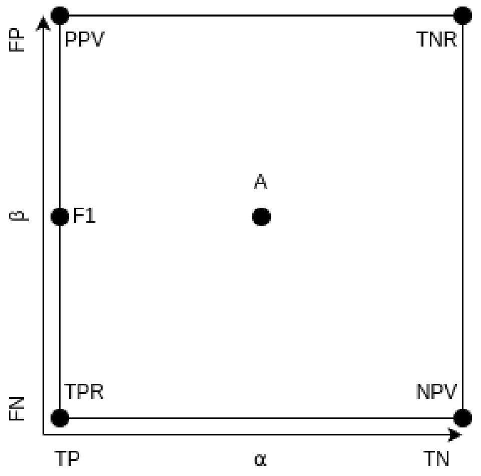


Figure 2.3. Graphical representation of the ranking tile.

Importance The importance value is defined as a non-negative random variable $I \neq 0$ that is used to parametrise the ranking score. Within the framework of BGS, the importance values are determined by two real values $\alpha, \beta \in [0, 1]$ as follows:

$$I_{\alpha,\beta}(o) = \begin{cases} 1-\alpha, & \text{if } o = TP, \\ \alpha, & \text{if } o = TN, \\ 1-\beta, & \text{if } o = FN, \\ \beta, & \text{if } o = FP \end{cases} \tag{2.9}$$

Using Eq. 2.8 and Eq. 2.9, the ranking score can be defined as:

$$R_I = \frac{\mathbb{E}_P[IS]}{\mathbb{E}_P[I]} = \frac{\sum_{o\in\Omega} I(o)S(o)P(\{o\})}{\sum_{o\in\Omega} I(o)P(\{o\})} \tag{2.10}$$

, where P is the probability distribution of the outcomes of the classification.For binary classificaiton, the equation can be simplified to:

$$R_I(P) = \frac{I(TN)P(\{TN\}) + I(TP)P(\{TP\})}{I(TN)P(\{TN\}) + I(TP)P(\{TP\}) + I(FN)P(\{FN\}) + I(FP)P(\{FP\})} \tag{2.11}$$

, which represents the ranking score of the algorithm given the probability distribution of its outcomes. From this formula, it is easy to derive the classical scores such as the accuracy, precision, recall, etc. For instance, the accuracy score can be obtained by setting the values of α and β to 0.5. Figure 2.3 shows the graphical representation of the tile with the "classical" scores.

2.5 Previous Work

The field of background subtraction has been the subject of numerous studies. When it comes to background subtraction algorithm recommendation, the current state-of-the-art lies in surveys that compares the performances of the algorithms on a set of datasets. These surveys are often based on the computation of the scores mentioned above. The works of Bouwmans (2014) [9], and Sanches *et al.* (2020) [12] are examples of such surveys. These works provide a comprehensive overview of the state-of-the-art algorithms and their performances on a few datasets. However, there is currently no platform that allows a user to obtain a recommendation that is not only based on metrics, but also on his needs. Specifically, that allows a fine-tuning of the ranking of the algorithms using parameters such as the context of the video, the hardware used, or the constraints of the application.

Even if no such platform currently exists, the scientific community has been working on the development of benchmarks that allow the comparison of algorithm. Some of them allowing the researcher to upload the results of their algorithm in order to compare them to the SOTA algorithms. These benchmarks provide a set of videos that are often annotated with ground truth masks which is a key element in the evaluation of BGS methods. Here is a non-exhaustive list of the most well-known benchmarks and datasets regarding background subtraction:

- Toyoma *et al.* [15] One of the first dataset that was used to assert the performances of BGS algorithms. It is composed of 7 sequences, each of which representing a different challenge. The dataset was also provided with one ground truth frame per sequence.
- Goyette *et al.* [6] The first version of the Change Detection dataset. It contains 31 video sequences that are spanned across 6 categories. Each video is annotated with ground truth masks. The benchmark also showcases the performances of the BGS methods using a set of metrics. The ranking is made in the general setting (all categories), and in each category separately. It is also possible to download the masks produced by the submitted algorithms.
- Vacant *et al.* [16] The BMC (Background Models Challenges) dataset has the particularity to contain synthetic videos with full ground truth annotations. It also provides videos long real sequences which are only partially labelled.
- Bloisi *et al.* [17] introduced the Maritime Detection, Classification, and Tracking dataset which is designed for multiple tasks including background subtraction. The dataset showcases video sequences that were taken from a maritime environment, some of them provided with a partial ground truth.
- Wang *et al.* [7] The second version of the Change Detection dataset. This version contains 53 video sequences that includes the 31 from the 2012 dataset and introduces 5 new categories while retaining the same set of features.
- Cuevas *et al.* [13] The LASIESTA (Labeled and Annotated Sequences for Integral Evaluation of SegmenTation Algorithms) was proposed in 2016. It contains 48 fully annotated video sequences that are divided into 14 categories. The work associated with the dataset also provides a benchmark of a few BGS algorithms.
- Jodoin *et al.* [18] The SBMNet benchmark is a dataset that was specifically designed for the task of background initialisation. It is composed of a set of videos that were taken from other datasets (CDNet, LASIESTA, LIMU, . . .). It is provided with a website identical to the one of the CDNet benchmark, which also comes with a ranking system.
- Introduced more recently by Zhang *et al.* (2022) [19], the Airport Ground Video

Surveillance (AGVS) dataset contains 25 long videos that were taken from an airport, each of them being annotated with pixel-wise segmentation masks. The dataset is specifically targetted to the detection of aircraft on the ground, and contains challenging situations such as camouflage, illumination changes, shadows, and more.

- Etc.

Chapter 3

Methodology

This chapter will detail the methodology followed in the creation of the ranking platform. In particular, it will detail the objectives of the project, as well as a detail about the milestones of the work.

3.1 Thesis objectives

The main objective of this thesis is to create a platform allowing a user to obtain recommendations for background subtraction algorithms. These recommendations are based on a wide range of videos, which are described using relevant tags (categories) that represent the content or the context of the videos. This web application would set itself apart from existing benchmark by providing a flexible recommendation/ranking system usable for non-expert users. Allowing them to obtain advices about algorithms that are best suited for their needs.

Firstly, the platform must be able to provide a multivariate recommendation for an algorithm that takes into account (1) the importance made on the specific outputs of the algorithms (i.e. the amount of true positives, false negatives, etc.), (2) the context of the videos used to determine the ranking, meaning recommending the algorithms that perform better in certain conditions, and finally the technical aspects of the algorithms, filtering out the ones that do not meet the specific criteria.

To allow researchers to compare their algorithms to the state-of-the-art, the tool must provide a way to submit new algorithms, and execute them on-demand in order to compute their performance against the video dataset, therefore inserting them into the ranking in place.

Building up upon the two previous objectives, the platform also needs to provide a way for the researchers to improve their algorithms. For that purpose, the application aims at providing an insight about past recommendations by keeping an history of the requests, giving the researcher information about how his algorithm would have been recommended for specific requests.

Lastly, the platform needs to be expandable. Along with algorithms, it must allow the augmentation of its pool of videos, and categories such that it stays up to date with the future developments in the field of background subtraction, and more broadly with computer vision.

3.2 Methodology details

The methodology that was followed in the creation of a prototype for the ranking platform can be compared to the agile framework. Indeed, although the project was not explicitly divided into sprints, it involved a series of iterations, each of which was focused on a specific aspect of the project. Moreover, the project was guided by feedbacks from the supervisor, avoiding the waterfall model. This allowed for a more flexible approach, which was particularly useful in refining the platform. The following steps are the broad milestones that guided this master thesis:

- **Bibliography research and survey of the existing benchmarks**: Conducting a literature review to understand the current state of the art in background subtraction and benchmarking.
- **Selection of the dataset to be used**: Choosing a suitable dataset for the implementation, the testing, and evaluation of the ranking platform.
- **Assignation of tags to the videos**: Defining the categories that would be best suited for each video and relevant to the ranking.
- **Automation of the ground truth production**: Implementing a system to automatically create segmentation masks for the videos that lack proper ground truth. Allowing for a broader range of videos to be used in the platform.
- **Context design for testing (split between train/tests/validation)**: Designing a test framework that would allow for the evaluation of the ranking process in a meaningful way.
- **Implementation of background subtraction baseline to be run on videos**: Implementing a basic BGS algorithm for the comparison with the other algorithms.
- **Implementation of a platform for algorithm ranking**: Setting up the platform that would allow the user to obtain a recommendation on the best algorithm for his needs.

These steps were not strictly followed in the order presented above, they were taken as guidelines to ensure the project was moving in the right direction. Some steps, such as the selection of the dataset and the definition of tags, were intertwined to optimise the workflow. The steps will be detailed in upcoming chapters of the thesis.

Chapter 4

Implementation Details

In this chapter are presented the details of the implementation of the Background Subtraction Recommender Platform. It explores the details of the architecture, providing insights about its main components. The chapter also highlights the core features of the prototype. Finally, a discussion about the choices made during the development, offering insights about the technical considerations that shaped the design of the application. The source code of all the implemented features is publicly available on Github: `https://github.com/pierard/BGS-Recommender-Platform`.

4.1 Dataset

In the context of algorithm evaluation, and by extension of algorithm ranking, the selection of an appropriate dataset crucial as it allows to have a common ground for the comparison. Ideally, a "good" dataset should cover the whole distribution of the data that the algorithms are likely to encounter. However, in practice it is impossible to obtain. When deciding on which dataset to use, a few criteria can be considered such as the size of the dataset, the diversity of the data, the presence of ground truth, etc. For this master thesis, the deciding factor has been the presence of labelled frames. Having a pre-existing ground truth alleviated the need for manual production and allowed to focus on the ranking process.

4.1.1 ChangeDetection 2014

The BGS Recommender platform uses the ChangeDetection 2014 dataset [7] as its main (and currently only) source of data. CDNet 2014 is a well known dataset, and a benchmark that is used to evaluate background subtraction algorithms. It is composed of 53 video sequences that are spanned across 11 categories (figure 4.1). Each video sequence is provided with partial, manually annotated ground truth. Notably, all videos within CDNet are camera-captured, and do not contain any synthetically generated data.

Categories

- Baseline: The videos in this category contain a mixture of challenges that are present in the other categories. These sequences are considered as less challenging.

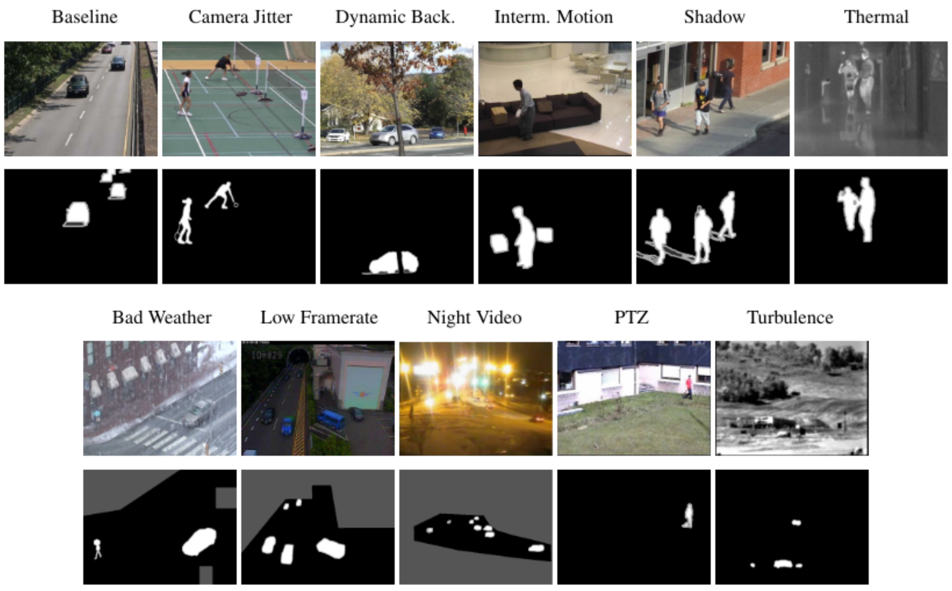


Figure 4.1. Image showing one frame of each video category of the CDNet 2014 dataset along with its corresponding ground truth. Source: *Wang et al.* [7]

- Camera Jitter: This category includes indoor and outdoor videos that have been captured with an unstable camera.
- Challenging Weather: Contains videos that have been filmed in challenging winter conditions such as snow storms, fog, etc.
- Dynamic Background: This category regroups scenes that include heavy parasitic background motion. This can be due to weaving trees, shimmering water, etc.
- Intermittent Object Motion: Sequences in this category contain objects that are moving intermittently. For instance, object that are in motion, stop for a while, and then start moving again.
- Low Framerate: The videos in this category have been captured with a low framerate. This can be challenging for the algorithms as it may cause sudden changes in the background.
- Night Videos: The sequences in this category have been captured during the night and contain primarily traffic scenes.
- PTZ: The videos in this category have been captured by Pan-Tilt-Zoom camera in slow continuous pan mode, intermittent pan mode, and 2-position patrol-mode PTZ, or in/out zoom.
- Shadow: Contains videos that feature strong shadows. Some of them are due to a bright sunny day, casting shadows of pedestrians, trees, etc.
- Thermal: This category contains videos that have been captured by infrared cameras.
- Turbulence: The sequences in this category include outdoor scenes that showcase air turbulence caused by rising heat.

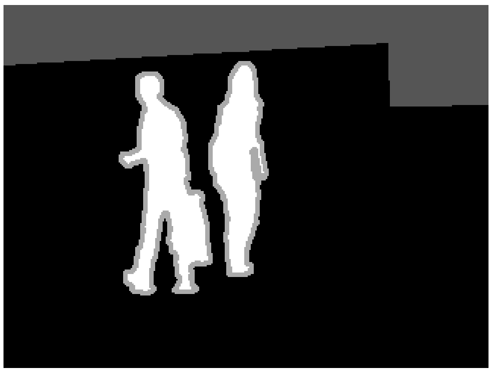


Figure 4.2. Image showing the ground truth of the "backdoor" video sequence of the CDNet 2014 dataset

Ground Truth

As previously mentioned, the CDNet dataset comes with accurate pixel-wise ground truth for each of the video sequences. The ground truth is provided in the form of binary masks and cover a subset of the frames of each video. Regarding the videos that were included by the 2014 dataset, the authors only provide the first half the annotations in order to avoid overtuning of algorithms parameters. Each video is provided with three additional files: ROI.bmp, ROI.jpg, and temporalROI.txt. The first two files are used to define the Region Of Interest of the video (the regions that have been annotated). The temporalROI.txt file is used to define the interval of frames that have been annotated. As for the ground truth itself, it is contains five different labels, which are defined as greyscale values, mapped as follows:

- 0: Static objects
- 50: Hard shadows
- 85: Outside the region of interest. The first few hundred frames of each video are labelled as outside of the ROI to avoid potential errors during the initialisation phase of algorithms.
- 170: Unknown motion (part that cannot be accurately labelled, due to semi-transparency and motion blur)
- 255: Motion

4.2 Technologies and Tools

The next few paragraphs of this section are aimed at describing the tools and technologies that have been used to make the development of the platform possible. Each of the items listed below are accompanied by a brief description that includes the reasons why they have been chosen as well as the role they play in the platform. The list itself does only include elements that are crucial for the functioning of the project. That is, the technologies that are essential in the understanding of the systems that will be explained later in this chapter.

Python Python is a high-level programming language that is widely used in the field of machine learning and computer vision. As such, it contains a vast number of libraries that are used in the field (i.e. PyTorch, Numpy, Scikit Learn, etc.). The language is at the core of the project as it is used in every backend component of the platform, except for the database (DB) and the Redis server. The choice of Python as the main language has been driven by API library used in the project (FastAPI). The main reason for this is that the platform will be maintained by a team of researchers that already have a good knowledge of the language, and will therefore be able to easily understand the code and improve the project. Another reason for choosing this language lies in a core feature of the platform which is the ability to execute algorithms that are likely to be written in Python.

Figure 4.3. FastAPI logo. Source: https://fastapi.tiangolo.com/

FastAPI FastAPI[1] is a modern Python web framework used to build RESTful APIs. It is based on standard Python type hints, and is built upon the Starlette[2] framework. FastAPI makes use of Pydantic in order to validate the data that is sent to the API. As mentioned above, the choice of FastAPI as the backend framework has mainly been main for maintenance purposes. Another guideline that has been involved in the choice of the framework is that it is lightweight and doesn't ship with many dependencies out of the box allowing for a more controlled environment.

Figure 4.4. Logo of the Celery library. Source: https://en.wikipedia.org/wiki/

Celery Celery[3] is a Python library that is used to manage distributed tasks. As defined by the documentation, Celery is a task queue that focuses on real-time processing. It is used in combination with a backend (in this case Redis) which serves as storage for the tasks that are to be executed. It also allows to store their results. The choice of Celery has been made relatively early in the development of the platform, and especially of the API. One of the main reason being that the standard background tasks are limited when

[1]FastAPI website: https://fastapi.tiangolo.com/
[2]Starlette website: https://www.starlette.io/
[3]Celery documentation: https://docs.celeryq.dev/en/stable/index.html

it comes to complex queuing systems in FastAPI. As for the choice of the library itself, it was one of the recommended choices from the documentation of FastAPI. Furthermore, having a distributed queue system is in line with the scalability of the project. Indeed, it would be possible to have celery apps deployed on different servers which could still communicate via Celery and Redis. The library was used both in the API and worker (generic and segmentation) modules.

Redis Redis[4] is a system that makes use of in-memory databases and key-value stores. Similar to other technologies such as RabbitMQ, Redis is used as a message broker by Celery. As such it constitutes a central piece in the communication with the workers. For this work, the choice of Redis over RabbitMQ has been made for its simplicity. Specifically because the tasks themselves are simple and thus do not require the advanced features of RabbitMQ. Furthermore, Redis is known for its speed due to the in-memory storage. In the application, it is used both as message broker, and result backend for the Celery workers.

PostgreSQL As an object-relational database management system, PostgreSQL[5] is used to store the meta-data of the platform, providing an efficient way to store and retrieve data. The choice of Postgres, and more broadly of a database system has been made after a few iterations of the project. The reason behind it being mostly because, as the complexity of the data model grew, it felt necessary to opt for a solution that allows to take advantage of the relations between entities. While using a simple file-based storage system (which was the originally used to manage data in the API), or even a No-SQL database like MongoDB would have been possible, it would have required adaptation to handle interconnections between objects. Regarding Postgres specifically, it provides a reliable production-ready, and scalable solution which supports a wide range of data types like arrays, JSON, etc. This is particularly useful for some models of the platform that require the storage of array-like data.

SQLAlchemy Used in the API module, SQLAlchemy[6] is an ORM[7] (Object-Relational Mapping) library that is used to interact with the database. The library itself is used to programmatically manage the database, it allows to manage the DB tables and their content without requiring explicit SQL queries. SQLAlchemy was the recommended choice for FastAPI as it is easily integrated with other elements like Pydantic. The main reason for which one would want to use an ORM is to abstract the SQL query logic, and to allow for a more maintainable codebase.

Alembic Less involved in features per se, Alembic[8] library is worth mentioning. Used in combination with SQLAlchemy, it provides a handy tool to handle the migrations of the database. Specifically, it allows to change the database schema easily using Python. It was straightforwardly chosen as a complement of SQLAlchemy and is used to define, and update the DB entities that are required by data models of the platform.

[4]Redis website: `https://redis.io`

[5]PostgreSQL website: `https://www.postgresql.org/`

[6]SQLAlchemy documentation`https://www.sqlalchemy.org/`

[7]ORM: mapping between classes in a programming language and database entities

[8]Alembic documentation: `https://alembic.sqlalchemy.org/en/latest/`

Algorithm	**Author(s)**
AdaptiveBackgroundLearning	-
AdaptiveSelectiveBackgroundLearning	-
CodeBook	*Kim et al.* [20]
FrameDifference	-
KNN	*Zivokic and Van der Heijden* [21]
LOBSTER	*St-Charles et al.* [22]
MixtureOfGaussianV2	*Zivokic* [23]
PAWCS	*St-Charles et al.* [24]
PixelBasedAdaptiveSegmenter	-
SigmaDelta	*Manzanera and Richefeu* [25]
StaticFrameDifference	-
SuBSENSE	*St-Charles et al.* [26]
TwoPoints	-
ViBe	*Barnich and Van Droogenbroeck* [27]
WeightedMovingMean	-
WeightedMovingVariance	-

Table 4.1. List of the algorithms of the BGSLibrary that are used in the BGS Recommender platform

BGSLibrary A module that implements a wide range of background subtraction algorithms. The BGSLibrary [28] is a C++ library that allows to easily execute BGS method by providing a common interface for all the algorithms. The library makes it easy to tweak the parameters of each algorithm by modifying their configuration. Although the BGSLibrary is written in another language, it is provided with a Python wrapper that can be installed using the pip package manager, which allowed for a seamlessly integration within the workers of the platform. Table 4.1 shows a summary of the algorithms that have been used in the platform to provide recommendations.

MMSegmentation MMSegmentation [29] is a framework that allows the implementation and evaluation of semantic segmentation methods. It contains a selection of popular algorithms that have been pre-trained on various datasets (i.e. ADE20K, COCO Stuff, PASCALVOC, etc.). The library is written in Python and based on PyTorch, a famous deep learning library. Within the platform, it is used by the segmentation workers to create semantic masks in order to provide a visual cue that can help selection tags of interest for the recommendation process. Its usage was decided based on the paper written by Piérard *et al.*, which introduced the importance tile by comparing the results of segmentation methods.

Figure 4.5. Docker logo. Source: `https://en.wikipedia.org/wiki/`

Docker Docker[9] is the cornerstone of the whole project. It is a platform that allows to build, share, and run application in containerised environments. Having the platform modules running in isolated containers was a key requirement of this work. Firstly, it provides a secure environment for the execution of the algorithms, mitigating risks associated with potentially malicious code. Secondly, it allows to have a fine control over the dependencies of the modules that is agnostic of the host system. Finally, it is also a way to ensure scalability of the application. For all those reasons, Docker has been an obvious choice for this work. In addition to the basic use of Docker, the Docker Compose tool was used to simplify the deployment and management of the containers. This tool is used to define the services with a YAML configuration file and to start/stop/build the module with a single command.

NGINX NGINX[10] is employed both as a web server that serves the frontend service, and as a reverse proxy that gives access to the API and the file server. Still in the perspective of scaling the application, NGINX was chosen for its performance and its ability to handle a large amount of concurrent requests. Moreover, it allows to hide the internal structure of the network by acting as a gateway that redirects the requests to the appropriate services.

Svelte Lastly, Svelte[11] is a JavaScript library used in the frontend module. It is a reactive framework that allows to build web applications. For the purpose of the prototype, Svelte was an ideal choice as it is lightweight, easy to understand, and facilitate the creation of dynamic user interfaces. Moreover, Svelte components are regrouped under single files that contain the HTML, CSS, and JavaScript code, allowing developers to quickly understand a component's functionality. This focus on maintainability is in line with the project to produce a future-proof application.Also, the framework ships with a few features that eliminates the need for external libraries. For example, it has a built-in store that allows to manage the state of the application.

4.3 Use Cases and Features

From figure 4.6 we can see the use cases of the platform. It regroups the main features that have been identified for the platform at the beginning of the project. At first glance, it can be noticed that these use cases are split across three main actors: the industrial user, the researcher, and the administrator, each inheriting the use cases of the previous actor. This means that the administrator is able to perform all the actions of the researcher and the industrial user. Similarly, the researchers can perform all actions available to industrial users, while also having additional capabilities. Each of the use case is accompanied with a symbol that represents its current implementation status ($\checkmark$: the feature is fully implemented, $\sim$: the feature is partially implemented, and $\bullet$ the feature is not implemented).

[9] Docker website: https://www.docker.com/
[10] NGINX website: https://www.nginx.com/
[11] Svelte website: https://svelte.dev/

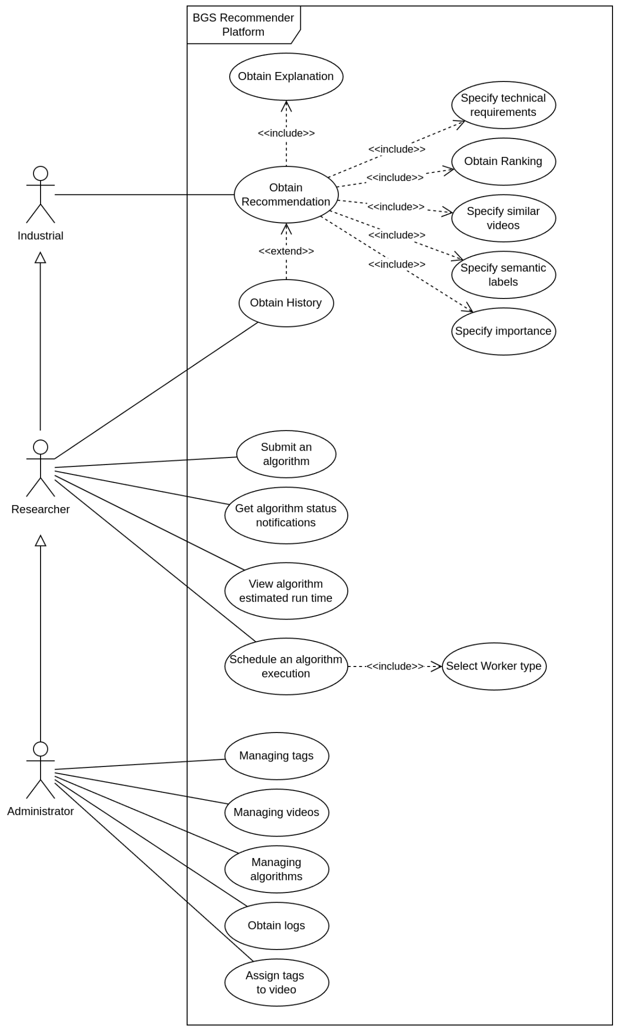


Figure 4.6. Diagram of the use cases of the platform.

4.3.1 Industrial User

The industrial user (IU) could be compared to the anonymous user. It is the typical user that has access to features (e.g. the recommendation) that do not require any authentication. The IU is typically an anonymous user looking for a recommendation for a BGS algorithm. However, he may not have the technical knowledge about classification algorithms, and more specifically about background subtraction. For that reason, the recommendation process is tailored to a contextualisation of the user's needs.

Obtain Recommendation ✓ As mentioned above, the industrial user can obtain a background subtraction algorithm recommendation. This is done by specifying context information such as the type of videos he is working with. As shown on figure 4.6, the recommendation process itself is composed of several other features.

Specify technical requirements • One of the core features of the recommendation process is the ability to specify the technical requirements of the user. Indeed, some context might require specific algorithm features. For example, the user might want algorithms that are can work on embedded systems which have limited resources. In such cases, the best algorithm would be the one that can run on the user's system, which may not be the state-of-the-art.

Specify similar videos $\sim$ Another feature of the recommendation process is the ability for the user to define videos that are similar to the ones he is working with. In this context, he would be presented with examples of videos/frames from the dataset of the platform. From these examples, the user could select the ones that are most interesting to him, allowing the recommendation to focus on the algorithms that work best on these types of videos.

Specify semantic labels ✓ Similar to the previous feature. This time, the user is presented with examples of images from the dataset that have been annotated with semantic labels. The user can then select the labels that are most relevant to his context. This feature is used in parallel with the previous one, and allow the IU to have a finer control over the included context.

Specify importance ✓ This feature relates to the ranking tile defined in the background chapter. Using the tile, the user is able to define the importance to be given to the components of the confusion matrix. For example, he could specify that he wants to minimise the amount of false negatives by setting the β coefficient to 1 (and 0 for false positives). This would ensure that the resulting ranking would place those algorithms that produce the least amount of false negatives at the top.

Obtain ranking ✓ The last feature of the recommendation process is obviously the ranking. Quite straightforwardly, after specifying all the required information, the industrial user is presented with a list of the algorithms that might fit his context. The resulting list is sorted in decreasing order of the ranking scores (the scores obtained using the ranking coefficients).

Obtain explanation ✓ Linked to the ranking, the user can also obtain a human-readable explanation of the recommendation he has obtained. The purpose of this explanation is to summarize all the information that has been given and provide insight about the effects of the query parameters on resulting recommendation.

4.3.2 Researcher

The researcher is a user who has access to a few more features than the industrial user. The main difference is that the researcher has needs for features that require authentication to the platform. His main characteristic is the ability to upload and execute his algorithms. This allows him to obtain insights about their behaviour against the dataset, and also to compare them with the other algorithms.

Submit an algorithm ✓ The feature that characterises the researcher actor is his ability to upload his own algorithms to the platform. It is achieved by submitting an archive file containing the source code and optional configuration files. The platform then takes care of the execution of the algorithm on the data.

Obtain insights • During the execution of the algorithm, the researcher is able to obtain insights about the present run. This includes metrics such as the estimated runtime, the memory usage, the number of pixels processed per second, a preliminary ranking of the algorithm, etc. Also, this feature allows the researcher to be informed in case of an error, such that he can correct it and resubmit the algorithm.

Get status notification • A feature that is linked to the insights is the status notification, it allows the researcher to be notified when the algorithm has finished its execution but also in case of errors.

Select a worker • When submitting his algorithm, the researcher is able to select the configuration of the worker that will execute it. Its main purpose is to be able to adapt to the requirements of the algorithms such as the language used, the libraries required, etc.

Obtain History $\sim$ This use case is related to the recommendation process. It corresponds to a history of the requests made by industrial users to obtain recommendations. When a submitted algorithm has been run on the dataset, the researcher is able to see how it would have been ranked in past requests.

4.3.3 Administrator

The administrator is the user that has access to all the features of the platform. He is able to manage the users, but also every aspect of the recommendation system (i.e. the videos, the algorithms submitted, the tags, etc.). The "management" feature relates to Create, Read, Update, and Delete (CRUD) operations on the platform's data

Manage users • The administrator is able to manage the users of the platform. Basically, he is able to accept or reject the registration of a user. He is also able to revoke a user the access to the platform if needs be.

Manage videos ✓ The administrator is able to manage the videos stored on the platform. He can upload new videos, delete existing ones, update their metadata (or replace the files), and view the videos that are stored.

Manage algorithms ✓ Similarly to the videos, the administrator is able to manage the algorithms. He can perform all CRUD operations even on the algorithms that have been submitted by the researchers.

Manage tags ✓ The administrator is able to manage the tags of the platform (CRUD operations). The tags are used to characterise the videos of the dataset. Furthermore, they are used in the recommendation process to filter out irrelevant algorithms.

Assign tags to videos ✓ This feature is related to the previous one. It allows the administrator to categorise a video by assigning tags to it.

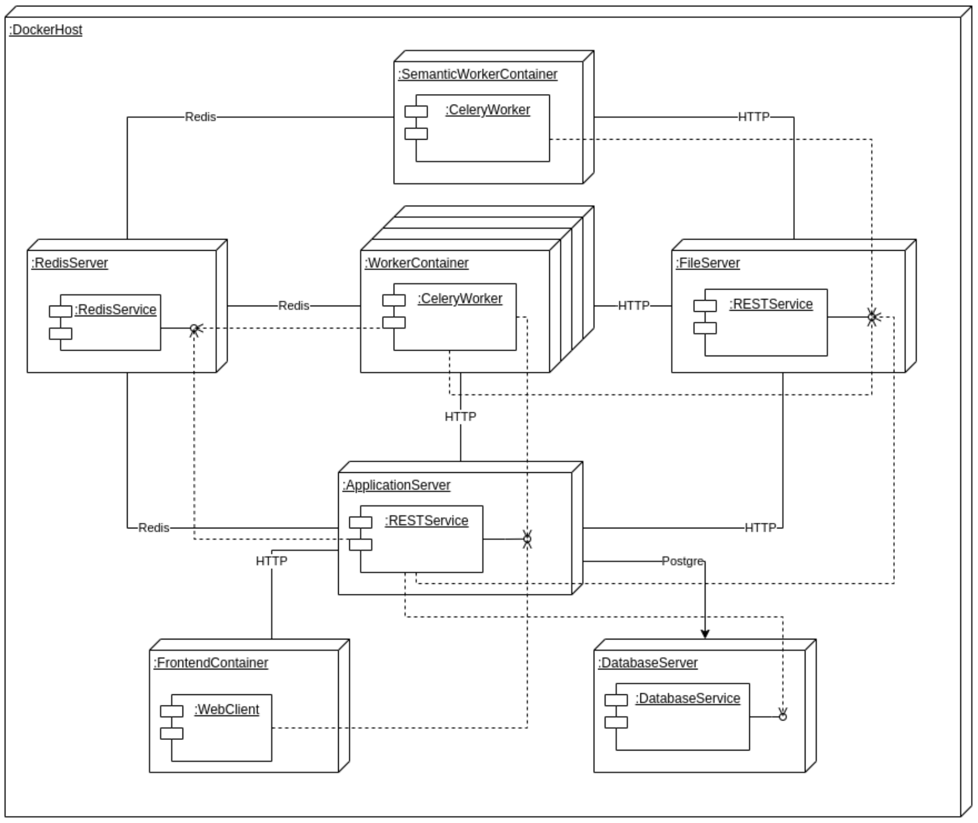


Figure 4.7. Architecture diagram of the platform.

4.4 Architecture

The architecture of the platform is depicted in figure 4.7, it is composed of several components that interact. The main components are the frontend, the backend, the database, the storage, and the workers. This section will describe in detail the role of each part, the protocols that are used, as well as the interactions between them. Firstly, it is important to note that all the services are containerised using Docker. It should also be noted that although the modules are designed to be deployable on different hosts, the platform can currently only be hosted on a single server (as depicted by the *:DockerHost* module).

4.4.1 Frontend

The frontend is the component that is responsible for the user interface (UI) of the platform. It is the part that the user interacts with in order to use the features of the application. Its interactions are restricted to the REST service of the *:ApplicationServer* (API) module via the HTTP protocol.

4.4.2 Database

The *:DatabaseServer* module is the element that has the responsibility to store all the "meta-data" of the platform. The meta-data is the information that allows to describe the various elements such as the videos, the algorithms, the tags, etc. The database is using a PostGreSQL data management system and only communicates with the *:ApplicationServer* module via the PostGreSQL protocol.

4.4.3 File Server

The *:FileServer* is the platform's file storage service, it is responsible for storing (and retrieving) files such as videos and the algorithms on the server using the HTTP protocol. Similarly to the database, the file server is never accessed directly from services other than the *:ApplicationServer*. The module can be accessed using regular endpoints using HTTP.

4.4.4 Redis Server

The *:RedisServer* is used as a message broker in the communications between the API and the workers. The API sends tasks to the Redis database through the Redis protocol, the workers are then able to retrieve, and treat them from the service. Additionally, the Redis server is also used as cache for storing the results of the segmentation tasks, such that they can be accessed directly upon completion.

4.4.5 Workers

"Classic" Workers The classic workers, represented by the *:WorkerContainer* modules, are the services responsible for the execution of background subtraction algorithms. For that purpose, they communicate with the *:RedisServer* (via Redis protocol) to retrieve the tasks that have been sent by the *:ApplicationServer*. The workers then execute the algorithms on the videos after retrieving relevant information from the *:ApplicationServer*, and the files from the *:FileServer*. Finally, the results are sent back to the API which handles the communication with the database and the file server.

Semantic Workers The segmentation workers are similar to the classic workers. However, their sole purpose is the execution of semantic segmentation algorithms. They are used to annotate frames of videos with semantic labels. Their behaviour is similar to their generic counterpart: they communicate with the *:RedisServer* to retrieve the tasks and to store the results which can then be accessed directly by the API. The masks produced are then stored on the file server for later access by the UI.

4.4.6 Application Server

The *:ApplicationServer* or "API" is the core of the platform, it contains most of the business logic of the application. The responsibilities of the API are to link the "outside world" with the internal components of the platform. This central role is reflected by the ability for the *:ApplicationServer* to communicate with all of the other services, using mostly HTTP (as illustrated by Figure 4.7). It is important to note that the API does not communicate directly with the worker but with the Redis database. The workers, however, know the API and use it to retrieve the data they need to execute the algorithms.

4.5 Backend Services

As it was previewed in the architecture section, the application is divided in several services. Each of which has a specific role in the platform. This part of the report delves into the details of each service, their implementations, and the interactions between their components. In this section is also described the main guidelines that were followed in the design of the modules.

From the early drafts of the project, a particular attention was put in the structure of the services. The main goal was to have a clear separation of concerns between the different modules. In order to achieve that goal, each module is broken down into "feature" submodules. Moreover, the services were designed as to respect the SOLID principles as much as possible. Those principles are a set of rules that are used to produce maintainable and scalable software. They are the following:

- **Single Responsibility Principle**: A class should have only one reason to change. This means that it should have only one well-defined responsibility. Thus avoiding "swiss-army knife" classes that are hard to maintain.
- **Open/Closed Principle**: A class should be open for extension and closed for modification. This means that the classes should not be modified to add new features. Rather, they should be extended by sub-classes.
- **Liskov Substitution Principle**: Objects of a sub-class should be able to replace objects of the parent class without breaking the program.
- **Interface Segregation Principle**: A class should not be forced to implement interfaces that it doesn't use, hinting at the fact that the interfaces should be as small as possible allowing for more modular code.
- **Dependency Inversion Principle**: High-level modules should not depend on low-level modules. Both should depend on abstractions. The idea behind this principle is to avoid tight coupling between modules by making them depend on interfaces and abstract classes rather than concrete implementations.

4.5.1 API

The API serves as the central coordinator of the entire system, integrating and managing all other services. It is the most important part of the platform. The API is the only service that has a direct access to both the file server and the database containing the metadata. Consequently, its main responsibilities are to provide an interface that allows to manage the resources, and to ensure the communication between services. Ultimately, its role is to act as a façade for the outside world to interact with the application.

Architecture Design

The service is structured into modules that each correspond to a specific set of features related to a resource (e.g. algorithms, videos, performances, etc.). All of these modules are following a similar structure and implement the repository pattern. Within this context, three main components are defined: the routers (or controllers), services, and repositories.

- **Routers**: the routers are the entry points of the API. They define the endpoints[12] of the service. Each of these routers is bound to a resource, and contain methods that interact with services to process the requests. These methods ensure that requests

[12]An endpoint is a reachable URL bound to a HTTP verb

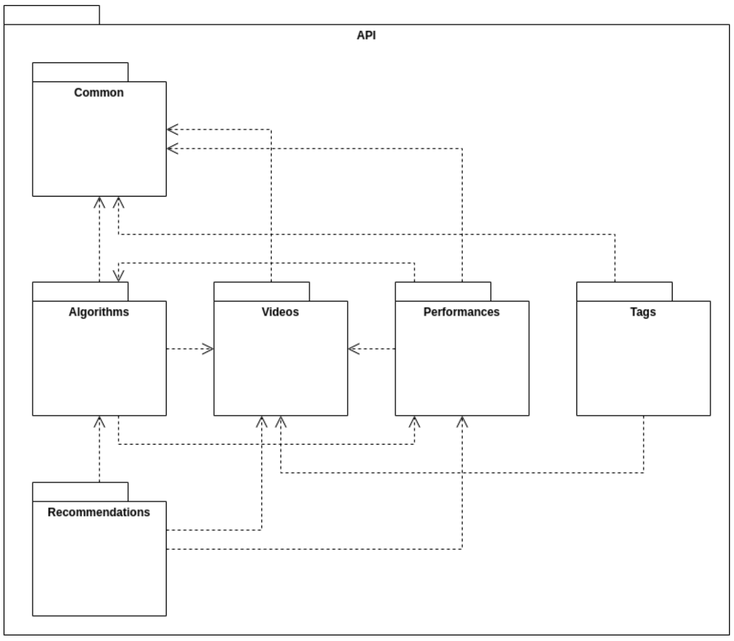


Figure 4.8. Diagram of the packages of the API service.

are appropriately routed to the services responsible for handling the business logic, which may involve contacting multiple services in the process.

- **Services**: the services represent the part of the API which handles the business logic. They process the information provided by the routers to perform potentially complex operation involving the data. Their main role is to ensure the communication with the data access layer by adapting the input information in a comprehensible format.
- **Repositories**: the repositories represent the data access layer, acting as the last element of the chain. Their responsibility is to persist, and retrieve the application's data. They are in charge of performing Create, Read, Update, and Delete operations on the data. The repositories are an abstraction layer between the business logic, and the storage medium.

Figure 4.8 presents an overview the structure of the API. It consists in all the main modules of the resources, and the *common* module which contains shared components related to the configuration of the api service (i.e. database parameters, settings, . . .). The diagram illustrates the interactions between the modules, and it can be noticed that almost all packages rely on others. This is a direct consequence of the separation of concerns, with resource management scattered across all modules. As a result, when a feature involves multiple resources, it uses the appropriate modules. For instance, the *algorithm* module relies on the *video* module in order to schedule executions of algorithms.

The API makes heavy use of dependencies injections in order to decouple classes as much as possible. To make it possible, the service relies on the special function `Depends()` provided by FastAPI. This function is a way to provide shared logic (such as a class or a function) to a module without the need for manual instantiation or code repetition. When combined with Python abstract classes from the abc module, this approach allows for the easy implementation of the Dependency Inversion Principle. The abstract classes are modules that can define methods that can later be implemented, hence defining a common interface that the children must follow. Concrete implementations of those classes can then be injected into the modules that requiring them. This has the major advantage of enabling easy implementation switches without having to heavily modify the dependent modules. This technique is typically used in the routers methods to obtain the necessary services.

Information Flow

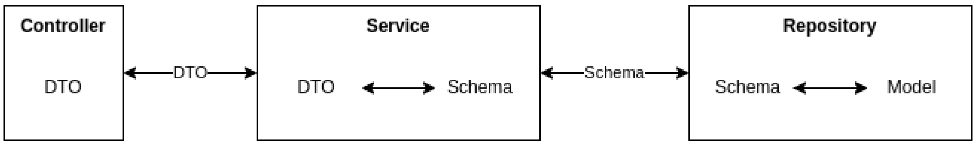


Figure 4.9. Diagram showing the flow of information in the API.

When performing a request to the API, the information flows through the different components of the service. It typically goes from the controller to the database and back. During this process, the data is transformed several times in order to be in the format required by the different modules. There are three types of data class that are used in the API. Figure 4.9 show the information flow of the API, it describes the exchanges between the components, as well as the conversion performed. In the API, three main data classes are used. They are the following:

- **Data-Transfer Object (DTO)**: The DTO is a type of class that is used to communicate with the controllers, and the services. This object typically defines the content of the request made to the router, it contains all the information needed for the request to be treated. When used in responses to the client it can be a modification of the original data schema that can provide more information or hide sensitive data. For instance, a data class like UserDto might only include the username and the email of a user, hiding the password. In the API, the data classes are defined using Pydantic which enables validation of the data when received by the controller. The DTOs are sent from the controllers to the services, where they are converted into the actual schema classes,
- **Schema**: The schema is a class that is used to define the structure of the data, facilitating its use by the services. It defines a direct mapping of the model used by the ORM to communicate with the database. The schema classes are typically employed by the services in their operations. They are used in the communication between the services and the repositories. Those classes are converted into DTOs when they are sent back to the controllers, and in Models when they are sent to the repositories,
- **Model**: The last type of data class is the Model, which defines an object-oriented representation of database entities, specifying the structure of the data, the relationships between the database tables, and the constraints applied to the data. The models provide a convenient way to perform database operations, abstracting away the underlying SQL queries. They are used in the repositories to manipulate the database tables using the SQLAlchemy ORM.

Communication

HTTP The main way of communication with the API is through the Hyper Text Transfer Protocol (HTTP). As aforementioned, each router exposes endpoints, which are typically reached through HTTP requests. A typical request is defined by three elements: the request line, the header, and the body. The request line contains the method, the URI (endpoint), and the version of the protocol. Depending on the method, the request line can also contain data. The headers are meta informations about the request such as cookies, authorisation tokens, the type of data accepted/expected, etc. Lastly, the body is the part that contains the payload that will be received by the client/server.

Although not enforced by the protocol, the method passed with each request has a semantic meaning. Depending on the method, the request will vary in the way information is carried. In the API, each router makes use of the semantic meaning as follows:

- GET: this method defines a simple retrieval request, which will typically expect a resource to be returned in the response body. For these types of request, data can be passed as a *query string* in the uri (i.e. /users?name=john).
- POST: used when creating a resource on the server, or when the data that needs to be passed is too complex for the simple GET endpoints. These types of request receives their data in the body, and can also receive data in the response body.
- PUT/PATCH: similar to the POST requests, these methods are used to update an existing resource on the server. As such, they also ship with the identifier of the resource to modify. The key difference between the two methods is that PUT means that the resource will be completely replaced, while the PATCH is only a partial update. The API module mostly use PATCH methods instead of PUT as it performs partial updates.
- DELETE: the last method type is the DELETE. It is used when deleting a resource on the platform, and only needs the id of the resource which is passed in the URI.

In order to handle client-server requests/responses, the API server will rely on three data types: JavaScript Object Notation (JSON), multi-part form data, and event-streams. The JSON format is used in most requests as it is easier to work with both in the API, and in the client side which natively supports it. The multi-part form data is used to handle specific requests in the video, and algorithms routers. It has the particularity of enabling multiple types of data to be shipped with the request. For example, combining meta-data of the resource with a file. This is particularly useful in the POST/PATCH requests that require the upload of data and file for the algorithms and videos, removing the need of multiple requests. The last type which is used is the event-stream. Its sole purpose is to send server events to the client. This data type is typically used in notification processes in a one-way communication. The REST service uses these event to signify to the frontend that the segmentation workers have finished the computation of masks.

Redis The second communication protocol used by the API is the Redis protocol that is used by the Celery library to create and publish tasks in the in-memory database managed by Redis for later retrieval by the workers. This behaviour is used by services the API to schedule computational heavy tasks such as algorithm execution, therefore avoiding to freeze the service for a potentially long period of time.

Features

The present section describes the features provided by the API service. Each of the subsequent sections are grouped by module, and give an insight about the main classes of each module with detailed explanation about the implementation of the functionalities.

Algorithms

The algorithm module has two main responsibilities that includes the management of the metadata of the algorithms through CRUD operations, and the scheduling of their execution. Figure 4.10 displays the main classes of the module and their dependencies. This diagram clearly showcases the DI principle as the `AlgorithmController` (the router), and the `AlgorithmsService` only depend on abstract classes, ensuring they are not tightly coupled to concrete implementations. Moreover, the figure also shows dependencies of the controller to services of other modules. Namely the `VideosService` and `PerformancesService`, which are used by the controller to retrieve videos, and performances respectively.

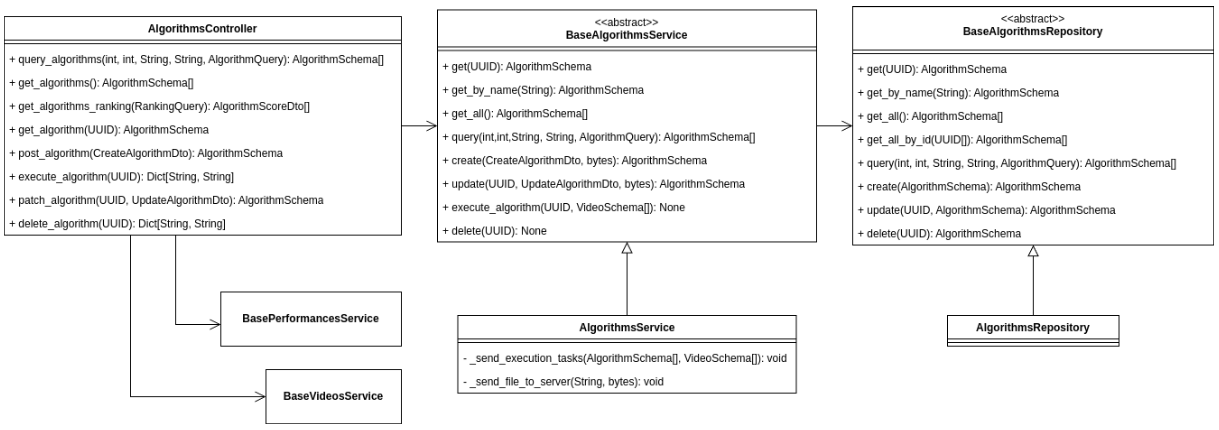

AlgorithmsController
+ query_algorithms(int, int, String, String, AlgorithmQuery): AlgorithmSchema[]
+ get_algorithms(): AlgorithmSchema[]
+ get_algorithms_ranking(RankingQuery): AlgorithmScoreDto[]
+ get_algorithm(UUID): AlgorithmSchema
+ post_algorithm(CreateAlgorithmDto): AlgorithmSchema
+ execute_algorithm(UUID): Dict[String, String]
+ patch_algorithm(UUID, UpdateAlgorithmDto): AlgorithmSchema
+ delete_algorithm(UUID): Dict[String, String]
<<abstract>>
BaseAlgorithmsService
+ get(UUID): AlgorithmSchema
+ get_by_name(String): AlgorithmSchema
+ get_all(): AlgorithmSchema[]
+ query(int,int,String, String, AlgorithmQuery): AlgorithmSchema[]
+ create(CreateAlgorithmDto, bytes): AlgorithmSchema
+ update(UUID, UpdateAlgorithmDto, bytes): AlgorithmSchema
+ execute_algorithm(UUID, VideoSchema[]): None
+ delete(UUID): None
<<abstract>>
BaseAlgorithmsRepository
+ get(UUID): AlgorithmSchema
+ get_by_name(String): AlgorithmSchema
+ get_all(): AlgorithmSchema[]
+ get_all_by_id(UUID[]): AlgorithmSchema[]
+ query(int, int, String, String, AlgorithmQuery): AlgorithmSchema[]
+ create(AlgorithmSchema): AlgorithmSchema
+ update(UUID, AlgorithmSchema): AlgorithmSchema
+ delete(UUID): AlgorithmSchema
BasePerformancesService
BaseVideosService
AlgorithmsService
- _send_execution_tasks(AlgorithmSchema[], VideoSchema[]): void
- _send_file_to_server(String, bytes): void
AlgorithmsRepository

Figure 4.10. Diagram showcasing the structure of the controller-service-repository classes of the algorithms package.

The `BaseAlgorithmsService` is the interface that contains the definition of the business logic of the algorithms. It exposes the resource management features through the `get`, `create`, `update`, and `delete` methods. These methods have a direct mapping in the `BaseAlgorithmsRepository`. The service is also responsible for the execution of algorithms, which is done through the `executeAlgorithm()` method. Its concrete implementation `AlgorithmsService` contains two additional "private" methods `_send_execution_tasks()`, and `_send_file_to_server()`. These methods are used to schedule the actual Celery tasks, and to send algorithms files to the File Server respectively.

Execution Scheduling The router will schedule the execution of algorithms upon receiving a request through its `execute_algorithm(UUID)` endpoint, or when a new algorithm is created with the execution flag set to True. In order to perform the scheduling operation, the controller retrieves all the videos via the `VideosService`, it then proceeds to hand the information to the execution method of the `AlgorithmsService`. The service will then check if the algorithm exists before creating the tasks. If it exists, it creates the tasks on the Redis service using Celery. Otherwise, a `AlgorithmNotFoundException` is raised to notify the user that the algorithm he is trying to run doesn't exist.

In order to handle the communication with other services and to define the content of the requests/responses to the endpoints of the algorithm router, the module exposes the following DTO classes:

- `AlgorithmPerformances`: This class is a simple combination of the algorithm's attributes with a list of performances.
- `AlgorithmQuery`: DTO used when a more complex query is needed rather than the simple retrieval. It contains only optional fields (id, and name), which are used to filter algorithms with matching fields.
- `AlgorithmScore`: This DTO is used by the ranking feature. It contains most of the AlgorithmSchema fields (except the path which is useless in this case), the importance score of the algorithm, and a few other classical scores (i.e. F1 score, precision, recall, ...) that are used for comparison.
- `CreateAlgorithmDto`: This class contains information related to the creation of algorithms. It exposes a name field, a file, and a boolean flag `execute` which is used to decide the execution of the algorithm after creation.
- `UpdateAlgorithmDto`: Similar to the previous DTO except that it contains the id of the algorithm to update, and doesn't expose the execution flag.
- `RankingQuery`: This DTO contains the required information to obtain a ranking. Specifically, the ranking tile weights, the ids of the algorithms of interest, and query objects for the videos and tags.

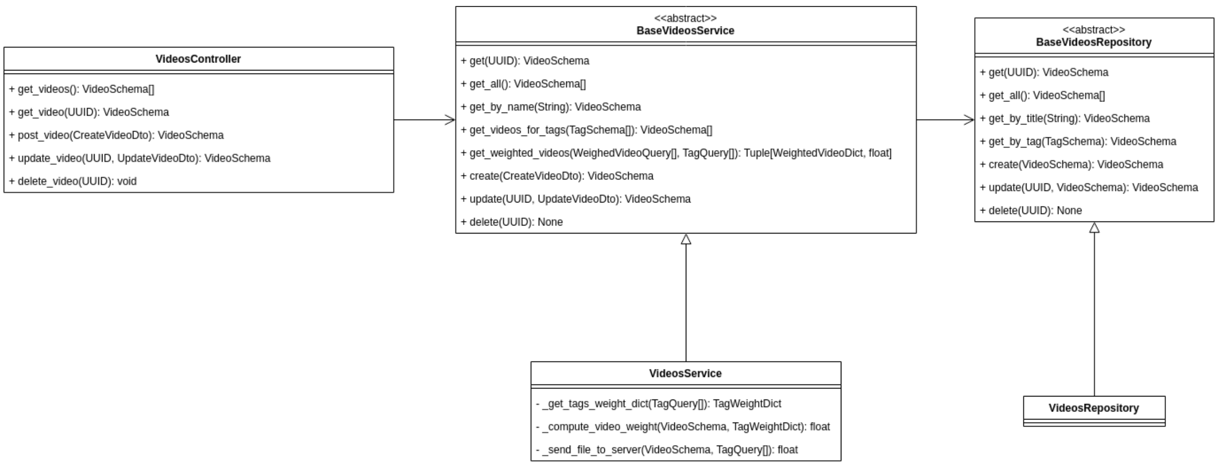

VideosController
+ get_videos(): VideoSchema[]
+ get_video(UUID): VideoSchema
+ post_video(CreateVideoDto): VideoSchema
+ update_video(UUID, UpdateVideoDto): VideoSchema
+ delete_video(UUID): void
<<abstract>>
BaseVideosService
+ get(UUID): VideoSchema
+ get_all(): VideoSchema[]
+ get_by_name(String): VideoSchema
+ get_videos_for_tags(TagSchema[]): VideoSchema[]
+ get_weighted_videos(WeighedVideoQuery[], TagQuery[]): Tuple[WeightedVideoDict, float]
+ create(CreateVideoDto): VideoSchema
+ update(UUID, UpdateVideoDto): VideoSchema
+ delete(UUID): None
<<abstract>>
BaseVideosRepository
+ get(UUID): VideoSchema
+ get_all(): VideoSchema[]
+ get_by_title(String): VideoSchema
+ get_by_tag(TagSchema): VideoSchema
+ create(VideoSchema): VideoSchema
+ update(UUID, VideoSchema): VideoSchema
+ delete(UUID): None
VideosService
- _get_tags_weight_dict(TagQuery[]): TagWeightDict
- _compute_video_weight(VideoSchema, TagWeightDict): float
- _send_file_to_server(VideoSchema, TagQuery[]): float
VideosRepository

Figure 4.11. Diagram showcasing the structure of the videos package.

Videos

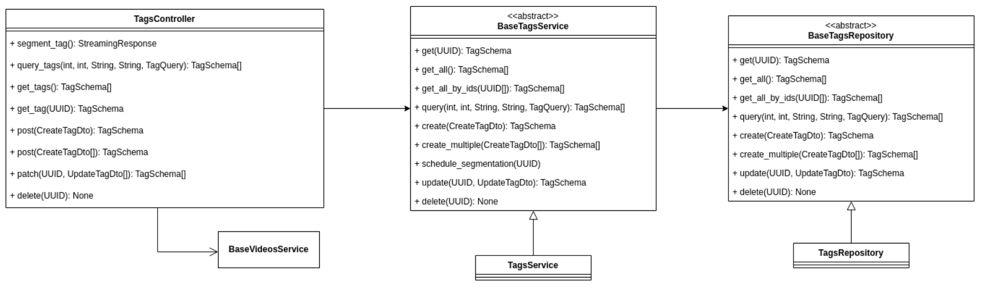
The video module contains the logic used to manage the videos of the platform. This module is rather simple as its sole responsibility is to manage the videos. Its operations therefore do not require the use of other services, hence the absence of dependencies on the diagram (fig. 4.11). Being very lightweight, it only defines a few Data Transfer Objects for its operations:

- `CreateVideoDto`: This DTO is a class that groups the fields that are required to create a new video. It contains two required fields: the title of the video, and the files that are linked to it. These files may include the frames (or video file), the ground truth, a mask for the ROI, and a temporalROI.
- `UpdateVideoDto`: Very similar to the previous DTO, it contains the same fields to which are added the id of the video that is to be updated, a boolean that indicates if the video is in the video is stored as a sequence of images or as a video file, and a list of tags that are assigned to the video.
- `WeightedVideoQuery`: The class defines relevant fields for querying videos in the context of the ranking. It contains the id of the video and a weight. This is used to give more importance to the video when obtaining the ranking score.

Similarly to the algorithm module, the video module contains an abstract class called `BaseVideosService` that contains mainly CRUD operations. The interface also provides two notable methods: `get_videos_for_tags()` which is used by the tag router to retrieve a set of unique videos representing all the tags provided as argument, and `get_weighted_videos()` which returns a list of videos that have been weighted according to tags of interest.

Video Weighting The video weighting process constitutes an integral part of the recommendation/ranking procedures. Indeed, each of them will require a weighted version of the videos in order to obtain the summarised ranking score. For that purpose, the `VideosService` uses the `get_weighted_videos()` method to obtain those weights. The weights are obtained by multiplying the initial weight of each video by the weighted average of its tags. Formally, this quantity is defined by eq. 4.1 where $w_t, w_v \in [0,1]$ represent the initial weights of the tags and videos respectively, $\mathbb{I}_v : \mathbb{R} \to \{0,1\}$ is a function that outputs 1 when the tag belongs to the video, and 0 otherwise.

$$W_v = w_v \frac{\sum_{t \in T} w_t \mathbb{I}_v}{\sum_{t \in T} w_t} \tag{4.1}$$

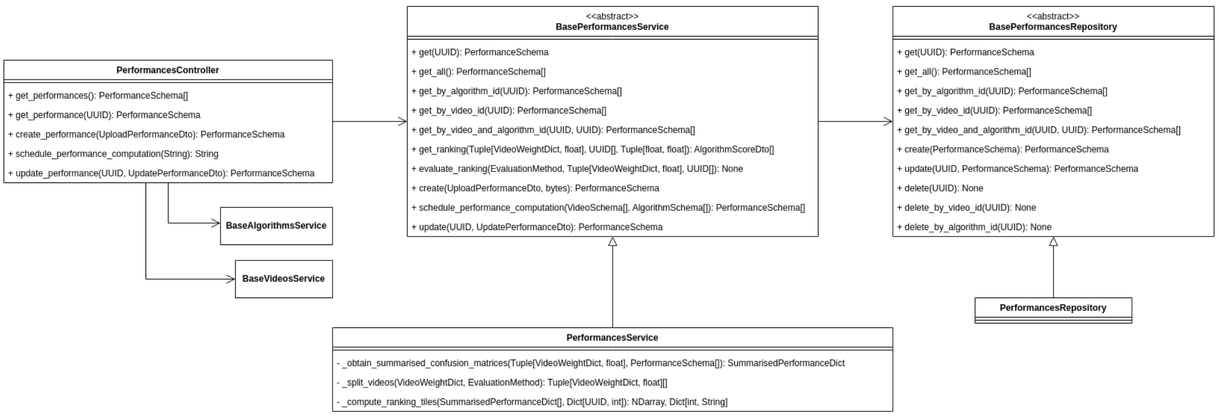

TagsController
+ segment_tag(): StreamingResponse
+ query_tags(int, int, String, String, TagQuery): TagSchema[]
+ get_tags(): TagSchema[]
+ get_tag(UUID): TagSchema
+ post(CreateTagDto): TagSchema
+ post(CreateTagDto[]): TagSchema
+ patch(UUID, UpdateTagDto[]): TagSchema[]
+ delete(UUID): None
BaseVideosService
<<abstract>>
BaseTagsService
+ get(UUID): TagSchema
+ get_all(): TagSchema[]
+ get_all_by_ids(UUID[]): TagSchema[]
+ query(int, int, String, String, TagQuery): TagSchema[]
+ create(CreateTagDto): TagSchema
+ create_multiple(CreateTagDto[]): TagSchema[]
+ schedule_segmentation(UUID)
+ update(UUID, UpdateTagDto): TagSchema
+ delete(UUID): None
TagsService
<<abstract>>
BaseTagsRepository
+ get(UUID): TagSchema
+ get_all(): TagSchema[]
+ get_all_by_ids(UUID[]): TagSchema[]
+ query(int, int, String, String, TagQuery): TagSchema[]
+ create(CreateTagDto): TagSchema
+ create_multiple(CreateTagDto[]): TagSchema[]
+ update(UUID, UpdateTagDto): TagSchema
+ delete(UUID): None
TagsRepository
Figure 4.12. Diagram showcasing the structure of the tags package.

Tags

The tags module is also relatively simple. It has the management responsibilities as well, and is capable of scheduling semantic segmentation tasks for the segmentation worker. The router has the particularity to expose a SSE[13] endpoint with the `segment_tag()` method. This route returns *text/event-stream* instead of JSON, this allows the frontend to receive periodic updates about the status of the segmentation tasks.

Segmentation Scheduling When receiving a request on the `/segmentation` endpoint, the router will call the `VideosService` in order to retrieve a set of videos that represent all the tags stored in database. The `schedule_segmentation(UUID)` of the `TagsService` is then called for each video. This method will create Celery tasks for the segmentation worker.

Regarding the DTOs, the tags package contains three classes. The first one is the `CreateTagDto` which is used to create a new tag. It only contains the label of the tag. The second one is the `TagQuery` class which is used in the recommendation process to put the emphasis on specific categories of videos. Its attributes are identical to the original `TagSchema` class except that it also contains a weight field that is used to define the importance of the tag in the recommendation process. Finally, there is the `UpdateTagDto` which is similar to the create DTO except it contains the id of the tag to update.



PerformancesController
+ get_performances(): PerformanceSchema[]
+ get_performance(UUID): PerformanceSchema
+ create_performance(UploadPerformanceDto): PerformanceSchema
+ schedule_performance_computation(String): String
+ update_performance(UUID, UpdatePerformanceDto): PerformanceSchema
BaseAlgorithmsService
BaseVideosService
<<abstract>>
BasePerformancesService
+ get(UUID): PerformanceSchema
+ get_all(): PerformanceSchema[]
+ get_by_algorithm_id(UUID): PerformanceSchema[]
+ get_by_video_id(UUID): PerformanceSchema[]
+ get_by_video_and_algorithm_id(UUID, UUID): PerformanceSchema[]
+ get_ranking(Tuple[VideoWeightDict, float], UUID[], Tuple[float, float]): AlgorithmScoreDto[]
+ evaluate_ranking(EvaluationMethod, Tuple[VideoWeightDict, float], UUID[]): None
+ create(UploadPerformanceDto, bytes): PerformanceSchema
+ schedule_performance_computation(VideoSchema[], AlgorithmSchema[]): PerformanceSchema[]
+ update(UUID, UpdatePerformanceDto): PerformanceSchema
PerformancesService
- _obtain_summarised_confusion_matrices(Tuple[VideoWeightDict, float], PerformanceSchema[]): SummarisedPerformanceDict
- _split_videos(VideoWeightDict, EvaluationMethod): Tuple[VideoWeightDict, float][]
- _compute_ranking_tiles(SummarisedPerformanceDict[], Dict[UUID, int]): NDarray, Dict[int, String]
<<abstract>>
BasePerformancesRepository
+ get(UUID): PerformanceSchema
+ get_all(): PerformanceSchema[]
+ get_by_algorithm_id(UUID): PerformanceSchema[]
+ get_by_video_id(UUID): PerformanceSchema[]
+ get_by_video_and_algorithm_id(UUID, UUID): PerformanceSchema[]
+ create(PerformanceSchema): PerformanceSchema
+ update(UUID, PerformanceSchema): PerformanceSchema
+ delete(UUID): None
+ delete_by_video_id(UUID): None
+ delete_by_algorithm_id(UUID): None
PerformancesRepository

Figure 4.13. Diagram showcasing the structure of the performances package.

Performances

The performance module is the most important of the API, it contains the logic that produces the ranking (and therefore recommendation) of the algorithms. Its main features are (1) the management of the performance data, (2) the scheduling of the performance computation, and (3) the computation of the ranking. The performance package only contain on two DTOs:

- `UploadPerformanceDto`: This DTO is used to upload a new performance, it contains the id of the algorithm, the id of the video, and the file containing the output produced by the algorithm,

[13]SSE: Server-Sent Events are events messages that the server sends that can be listened to

- `UpdatePerformanceDto`: This class, unlike the previous one, does not contain the files. It is used to update the confusion matrix of a performance when the workers have completed its computation. It contains all the fields of the database model.

The `BasePerformanceService` exposes the methods used to handle the CRUD operation. It also contains the `get_ranking()` method that computes the algorithm ranking, and the `schedule_performance_computation()` which uses Celery to create the task for the computation of the CMs of the algorithms by the workers. The service is implemented by the concrete `PerformancesService` class which also has a few private utility methods that are used in the ranking and evaluation processes.

Ranking Computation The computation of the ranking is enabled by the algorithm router's `get_ranking()` method which provides the `PerformancesService` with the already weighted videos, a list containing the ids of the algorithms, and a tuple containing the ranking tile coefficients. Using these elements, and the list of all the performances, the service will compute the ranking as follows:

1. The service will first obtain a summarised version of the CMs of the algorithms by performing a weighted average of the CMs using the weights of the videos,
2. The summarised CMs are then weighted using the importance coefficients α and β in order to compute their ranking score,
3. The resulting algorithm list is then sorted by decreasing order of their score and returned back to the router.

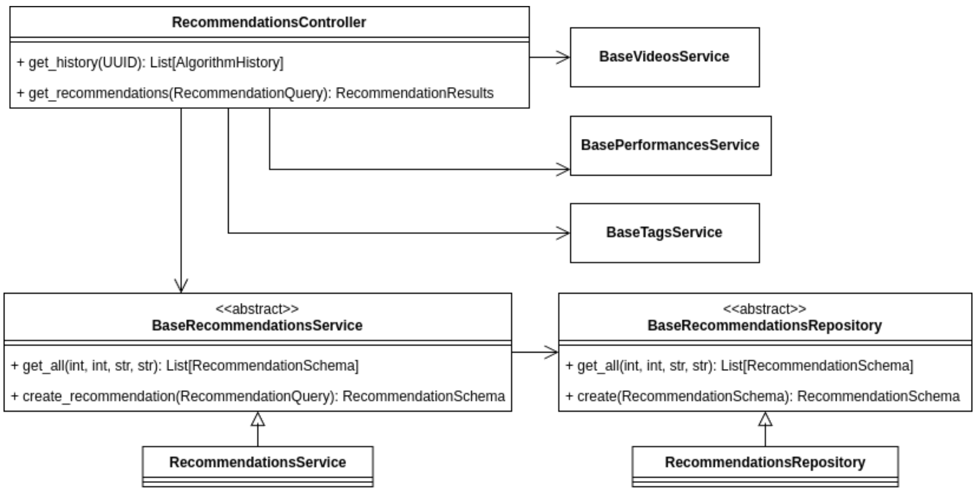

RecommendationsController
+ get_history(UUID): List[AlgorithmHistory]
+ get_recommendations(RecommendationQuery): RecommendationResults
BaseVideosService
BasePerformancesService
BaseTagsService
<<abstract>>
BaseRecommendationsService
+ get_all(int, int, str, str): List[RecommendationSchema]
+ create_recommendation(RecommendationQuery): RecommendationSchema
<<abstract>>
BaseRecommendationsRepository
+ get_all(int, int, str, str): List[RecommendationSchema]
+ create(RecommendationSchema): RecommendationSchema
RecommendationsService
RecommendationsRepository

Figure 4.14. Diagram showcasing the structure of the recommendations package.

Recommendations

The recommendation package is a rather lightweight module, its main classes only contain a few methods. This module has two main roles: the first one is to process the user query defined in the `RecommendationQuery` (and to return an appropriate response as a `RecommendationResult`), the second one is to provide a list of the rankings obtained by an algorithm based on the history of queries. The implementation of these features require the router to depend on the modules of the videos, tags, and performances services.

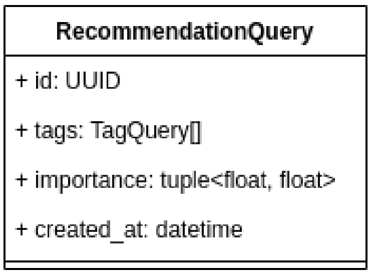


Figure 4.15. Diagram of the RecommendationQuery class.

The `RecommendationQuery` class is a Pydantic model that is used to define the a request for recommendation formulated by the user. As shown by figure 4.15, this query defines four fields: its id, a list of `TagQuery` objects, the ranking coefficients, and a creation date. Among these fields, only the tags, and importance are effectively used in the recommendation process.

Obtaining a recommendation Upon receiving a request on its `get_recommendation()` endpoint, the recommendation controller first proceeds to retrieve the weighted videos from the `VideoService` using the tag queries of the request. It then proceeds to call the `RecommendationsService` to store the query. When the storage is complete, retrieves the ranking of the algorithms by sending the weighted videos, and the ranking coefficients to the `PerformanceService`. Finally, it returns a `RecommendationResults` object that contains the ranking, and the content of the initial query.

Obtaining the history of rankings The router also expose a `/history` endpoint which take the id of an algorithm in the query string. When receiving a request, the router will first retrieve the history of the queries sorted in descending order of their creation date (newer first) by calling the `get_all()` method of the `AlgorithmService`. For each queries, the router will:

1. Obtain the ranking from the `PerformancesService`,
2. Find the rank of the algorithm with the id specified in the query string,
3. Create a `AlgorithmHistory` object that contains the rank of the algorithm, its name, the ranking coefficients (α and β), and a list of `TagQuery` that were associated with the recommendation.

Error Management

When working with data, errors are bound to happen, especially when it involves a user in the process. In that regards the BGS platform makes no exception. In order to handle these errors, the API implements a few mechanisms that help the user correcting his inputs. These mechanisms take advantage of the HTTP protocol once again by sending appropriate status code, and message when an error occurs. To do so, it makes use of the `HTTPException` class of FastAPI. When thrown, if not caught, this exception class has the particularity to return a HTTP error. The services of the application take advantage of this behaviour be inheriting the base `HTTPException` to return custom errors. In the API, three types of errors can be identified, each of which corresponding to a HTTP status code:

- *The resource was not found*: the typical 404 error, this error happens when the user tries to access a resource or an endpoint that does not exist. The API module send this messages whenever it has to search for data in its operations.
- *The resource already exists*: this error happens when adding a new resource (e.g. a video, and algorithm, etc.) which already exists. It is linked to the 409 (Conflict) status code.
- *The request body couldn't be processed*: attached to the 422 code (UnprocessableEntity), this message is received when the user sends data which is wrongly formatted. This type of error is mostly sent by Pydantic models when the body of the request doesn't conform to the declared Schema (e.g. when required fields are missing, or have incorrect types).
- *An error occurred when treating the request*: the most generic type of the three, this corresponds to the status 400 (BadRequest). This particular error is sent when no more precise types can be sent.

4.5.2 File Server

The File Server(FS) is a simple service, it is composed of a single module responsible for the management of the files of the platform. Similarly to the API, the FS uses the FastAPI library which constitutes its only dependency. The library is used to run the server and exposes the endpoints that allows the storage and deletion of files, it also statically serves files for easy retrieval. Currently, the FS only manages files associated with three types of resources: algorithm, video, and performance.

Endpoints

Being such a simple service, the FS doesn't ship with a complex structure. In fact, almost all of its logic is defined in the main file which contains 7 endpoints, including the serving of the `/data` directory. Among the endpoints, three of them are dedicated to the storage of resources and are associated with HTTP PUT methods. The other half is dedicated to the deletion of resources and are therefore associated whith the DELETE method. Finally, the FS gives direct access to it data folder which contains the files that are stored by the PUT requests.

4.5.3 Workers

Worker services are responsible for the execution of potentially computation heavy tasks. Their primary objective is to be able to execute Python algorithms regardless of their dependencies. These modules make heavy use of the Celery Python library to retrieve and execute tasks. They are declined in two flavours:

- the "generic" workers (GW) which are taking care of two type of tasks: the execution of the background subtraction algorithms, and the computation of confusion matrices.
- the segmentation workers (SW) which are only assigned tasks linked to the execution of semantic segmentation algorithms.

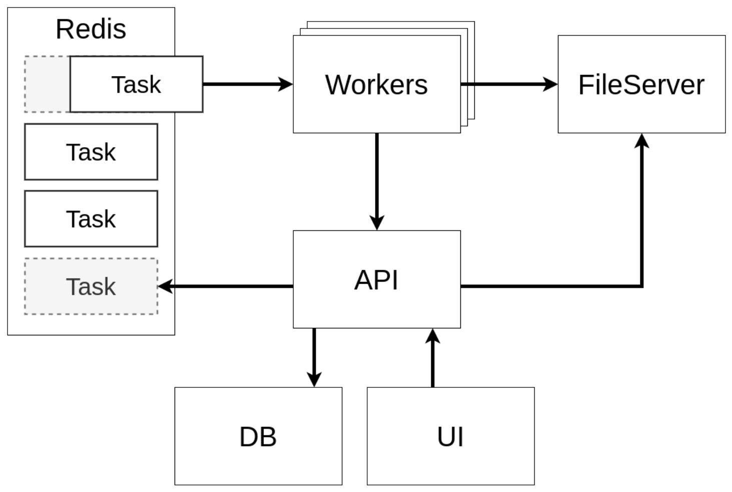


Figure 4.16. Diagram of the flow of the tasks in the application.

The key difference between these two types is that the generic version is highly versatile and can essentially be used to execute any Python code. While segmentation version is completely geared towards semantic segmentation tasks.

Tasks Flow

Figure 4.16 shows the flow of the tasks in the application. This flow typically starts with a request from the UI (or any other HTTP client), the API then publishes the task on the queue in the Redis database. Using Celery, workers are then able to pop the task from the queue to execute it. When done, if needs be, the result is put back on the Redis storage for the API to retrieve it.

Generic Workers

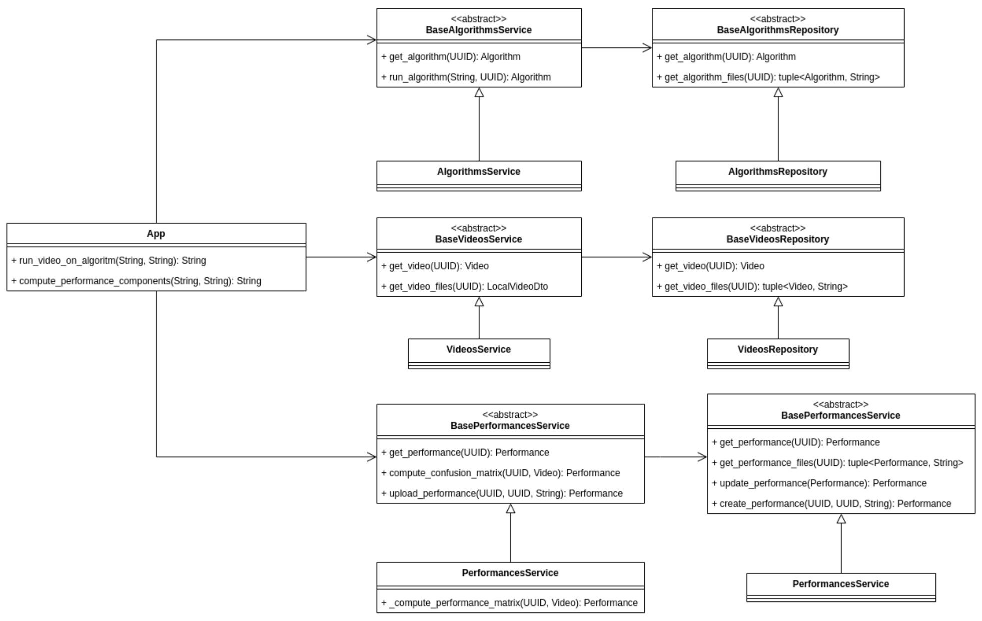

<<abstract>>
BaseAlgorithmsService
+ get_algorithm(UUID): Algorithm
+ run_algorithm(String, UUID): Algorithm
<<abstract>>
BaseAlgorithmsRepository
+ get_algorithm(UUID): Algorithm
+ get_algorithm_files(UUID): tuple<Algorithm, String>
AlgorithmsService
AlgorithmsRepository
App
+ run_video_on_algoritm(String, String): String
+ compute_performance_components(String, String): String
<<abstract>>
BaseVideosService
+ get_video(UUID): Video
+ get_video_files(UUID): LocalVideoDto
<<abstract>>
BaseVideosRepository
+ get_video(UUID): Video
+ get_video_files(UUID): tuple<Video, String>
VideosService
VideosRepository
<<abstract>>
BasePerformancesService
+ get_performance(UUID): Performance
+ compute_confusion_matrix(UUID, Video): Performance
+ upload_performance(UUID, UUID, String): Performance
<<abstract>>
BasePerformancesService
+ get_performance(UUID): Performance
+ get_performance_files(UUID): tuple<Performance, String>
+ update_performance(Performance): Performance
+ create_performance(UUID, UUID, String): Performance
PerformancesService
+ _compute_performance_matrix(UUID, Video): Performance
PerformancesService

Figure 4.17. Diagram of the main classes of the worker stack.

To a certain extent, the generic workers follow a similar structure to the one of the API. Specifically, they are subdivided into three main modules: algorithms, videos, and performance. These modules themselves contain a service, a repository, and a few other classes that are used to handle data. In this context, the services are responsible for the execution of the tasks, and the repositories still are used to interact with the other service, therefore acting as the data access layer. Furthermore, the workers contain definitions for the tasks they can handle. These tasks can be viewed as their endpoints as they act as the entry point in which they receive their work. Upon receiving a task, the worker will contact the appropriate services to complete the task. The generic workers are defined to handle two types of tasks: the execution of an algorithm on a video, and the computation of the confusion matrix of an algorithm for a video.

The internal structure of the service is shown in figure 4.17. From the diagram, it is immediately visible that the type of structure is identical to the one of the API except for the routers. This structure is used to ensure the modularity of the workers such that they can easily be augmented with new tasks. Each feature module implements the communication with the other platform components, and their services also define the logic that is used to accomplish the task. An exception is made for the `VideosService` which is not directly linked to a task dedicated to the videos. Similar to the API, each class inherits from an abstract class that defines the interface. However, since the workers do not use FastAPI, the dependencies injection is done manually in the main module.

Tasks

Algorithm execution This task is responsible for the execution of a BGS algorithm on a particular video. To do so, the worker will use the ids of the video, and of the algorithm it receives in argument to retrieve the files using information about their storage location obtained from the API. After downloading the files, the algorithm is run on the video, the result stored, and then sent back to the API. As for the task, it is defined as `run_algorithm_on_video(video_id, algorithm_id)` and is composed of the following steps:

1. First, the worker will create a virtual environment (using pip) in which the algorithm will run.
2. It will then install some generic dependencies in the environment (i.e. numpy, opencv, ...), and the specific dependencies of the algorithm using the requirements.txt file if provided,
3. When the dependencies are installed, it executes the algorithm in a subprocess using the command line interface (CLI) of the algorithm. To the CLI, the worker inputs the path of the video, and the path where the output should be stored,
4. Upon completion of the algorithm execution, the worker proceeds delete the environment, and to send the output to the API. The output is sent as a multi-part form data containing a zip file of the masks produced by the background subtraction, and some meta-data.

Confusion matrix computation The second task of the worker is the computation of confusion matrices. Similarly to the previous task, the worker receives the ids of the video, and of the performance object. It then retrieves the files, and computes the confusion matrix using the output of the algorithm, and the ground truth of the video.

Segmentation Workers

The segmentation workers are very similar to their generic counterpart. Their main difference is in the structure of the module. Indeed, being simpler, they only contain a few files. Their only responsibility is to execute semantic segmentation algorithms. To do so they make use of the MMSegementation library which contains a set of pre-trained models. The main reason why the segmentation workers are separated from the generic ones is because they have a different purpose. While the GW are focused towards the execution of BGS algorithms, and computation of performances, the SWs are used to compute segmentation masks that will directly visible by the user during the recommendation process. Furthermore, the set of dependencies they require to run is different, it also allows to scale them separately. However, having the workers separated means that they cannot share the same database bucket to store their tasks and results. This is a direct consequence of the separation of the tasks, the GWs aren't aware of the tasks intended for segmentation workers, and vice-versa.

Segmentation Model The segmentation workers are using a pre-trained neural network called Mask2Former[30] to segment the video. This model was chosen for two reason, the first one being that it was trained on the ADE20K dataset which contains images that are similar to the videos present in CDNet. The second reason was because the results obtained by Piérard *et al.* in their work on the ranking tile showed that it was ranked first for a significant portion of the tile.

Frame segmentation The only task of the SWs is the segmentation of a video frame using a pre-trained model. The task is defined as `compute_semantic_masks(video_id)` and only requires the id of the video to segment. The segmentation process is the following:

1. When the workers receives a task, it will first contact the API to obtain information about the video it is asked to segment.
2. After retrieving the metadata of the video, it uses the path attribute to retrieve the files of the video, and then proceeds to extract the archive.
3. The worker then selects one of the frame of the video at random, and executes the model in inference mode on it.
4. The model then produces an output defined as a colour mask that contains the identified semantic labels.
5. The worker then sends the output mask to the file server for temporary storage, and returns the path of the mask along with its labels. The returned data is stored in the Redis database for the API to retrieve.

4.6 User Interface

Along with the backend services, the user interface (UI) is a key component of the platform, providing the users with visual feedback on their actions. The UI is a web interface that is built using the Svelte framework and is therefore accessible in any browser. This application is primarily structured as a dashboard, a layout chosen to meet the requirements. Since the platform is mostly used for data management operations, having a global view of the data is essential. Dashboard layouts are particularly well suited for information presentation, making them a natural choice for the platform.

The frontend (FE) application was built using as to rely on a minimal set of dependencies. This decision was motivated by several reasons, the main one being the ease of maintenance. Fewer dependencies make the codebase easier to understand for future maintainers of the platform. Additionally, this approach also bring more stability to the program, making it less prone to breaking upon dependencies updates. The following non-exhaustive list contains the core dependencies of the frontend service:

- Svelte: The JavaScript framework used to build the UI,
- svelte-routing: A routing library that is used to manage the different pages of the application,
- axios: A library that is used to simplify the use of HTTP requests by providing an easy configuration of the requests,
- typescript: The language in which the components code are written. It provides type-checking, and better code completion than plain JavaScript,
- vite: The packager used to bundle the application.

4.6.1 Structure

The frontend service is designed in a feature-oriented way, grouping the components linked to a specific feature together. Still in a future-proofing perspective, the components are separated in such a way that it is easy to implement new functionality and to build upon existing ones. The FE service is divided into three main parts:

- **Common Components**: The common components are the building blocks of the frontend service. They are the smallest unit of the UI, and are combined to create more and more complex components. These components are agnostic of any features, meaning that they can be reused in any part of the application. Therefore avoiding code repetition. Furthermore, they could even be reused in other applications if needed. Typical components include buttons, inputs, navigation bars, ...
- **Feature Components**: These type of components are tightly linked to the application they are used in and/or to a particular state management system (i.e. React Redux). They are usually formed by an aggregation of generic components, and augmented with data related to the feature. For instance, the recommendation form is a feature component, it is composed of buttons, inputs, and other components that are combined to create a form. It is also responsible for the requests that are sent to the API to obtain the recommendation.

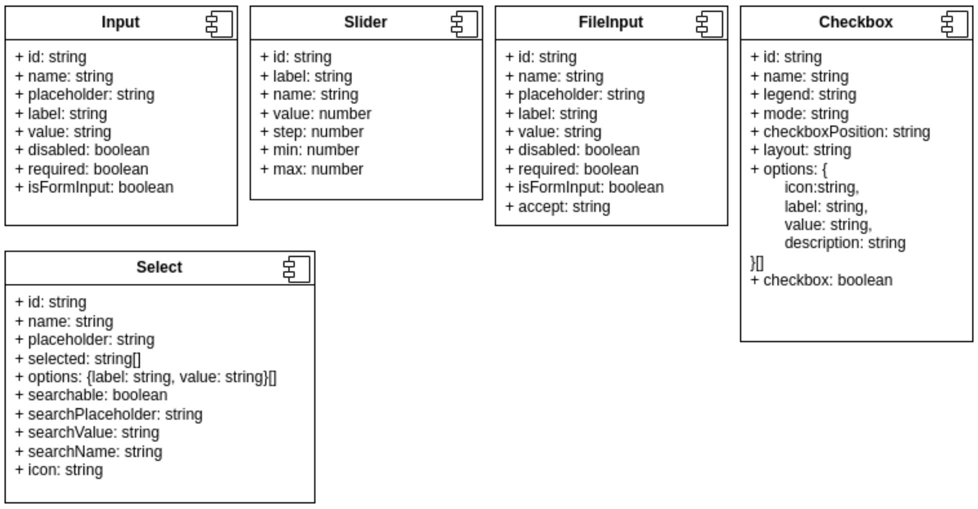


Figure 4.18. Diagram of the main form components of the UI.

- **Pages**: Pages are the highest level of components, they define the different views of the UI, which also make them tied to the features of the application. The pages define the layout of the platform, and are a combination of feature and common components.

4.6.2 Design

Although no particular design guidelines were imposed, the UI was designed to be as user-friendly as possible. Its look is heavily inspired by the Carbon Design System[14]. The reason for this choice is that the look and feel of the design system is in line with the desired look of the platform. However, the FE application isn't using pre-made components. Instead, its styles were built from the ground up, only using the design system as guideline.

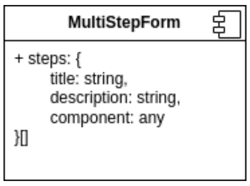
Responsiveness The UI is designed to be as responsive as possible. This means that it can be used on any devices, and will adapt to the screen size. To make it possible, the layouts were carefully designed through extensive use of media queries and Flexbox to ensure that the components are displayed correctly on smaller screens.

4.6.3 Components

In order to build the FE, a set of components had to be created, most of which can be categorised as "form" components due to their link to data input. Indeed, a significant part of the platform's features is to be able to manage data (i.e. algorithm upload, video upload, etc.) as such the frontend must provide components that allow to input, and display data. Also, the components are designed to be as reusable as possible. This is achieved by the use of attributes (called props) that allow to pass data to the component. They are also self-contained, meaning that all the logic that is required to make them work (the TypeScript code, the HTML, and the style) is comprised in the same file, making it easier to maintain them.

[14]Carbon Design System: https://www.carbondesignsystem.com/

Form components These components are used to provide data from the user to the application. They are composed of a set of inputs that can hold different types of data, and/or present the data in the most convenient way depending on its type (i.e. use a slider for numbers, radio button for choices, . . .). For the needs of this work a few form components were created. Figure 4.18 shows the main components along with their respective props. The first thing that can be noticed from the diagram is that they all share a few properties: id, name, label, value, and disabled. Those properties are indeed standard HTML input properties which needs to be present on all form inputs. However, the components themselves also have distinct attributes which are related to the type of content they can hold. For example, the FileInput has a `accept` property that defines the type of files that can be uploaded. The Select is the only component that makes use of another form component (the Input) to define a search field that allows to filter the options of the select.

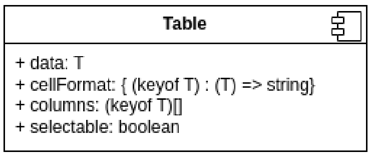


```
+ steps: {
      title: string,
      description: string,
      component: any
}[]
```

Figure 4.19. Diagram of the properties of the multistep form component.

MultiStepForm Used by the recommendation process, the multi-step form (see figure 4.19) is a component that allows to combine the outputs of multiple form. This has the advantage of splitting the whole data collecting process into smaller, more manageable parts. The component has a single property called "steps". A step is itself an object that is defined by a title, a description, and a component. The component is the actual form that is displayed to the user. The title and description props are used to give the context of each step to the user. The multi-step form is also responsible for the navigation between the steps. It works by using the `submit` event sent by the forms to switch to the next step, and combine the data. An example of its usage can be seen in the features' section under the recommendation form.

Tile The tile component, usually used in combination with sliders, is used to display the importance values in a graphical way. In its simplest form, the tile is simply a 2D square that contains a dot representing the coordinate of the sliders. It is used in two area of the platform: the recommendation form, and the filters of the ranking page. This component has a limited set of properties which are the (X, Y) coordinates of the point. The tile itself is fully interactive, it can be clicked on to change the coordinates of the point.



```
+ data: T
+ cellFormat: { (keyof T) : (T) => string}
+ columns: (keyof T)[]
+ selectable: boolean
```

Figure 4.20. Properties of the table component

Table The table (figure 4.20) component is used to display data in a tabular form. It is used by the ranking, the videos, and the algorithms pages. The component makes use of

Typescript generics to define the type of data that is displayed. This allows to have a more flexible table which can adapt to any type of data. As such, it contains four properties: the data which is a generic type, the cellFormat which is an object used to formats the data, the columns that should be displayed, and a boolean value that indicates if the table elements can be selected.

4.6.4 Features

The purpose of the UI is to provide an easy way to interact with the platform, allowing the user to obtain recommendations, upload algorithms, view the dataset, etc. It can be divided into three main parts: the recommendation form, the ranking page, and the resources management part. From the initial use cases of the platform, a few of them couldn't be implemented in the time frame of this work as efforts were put on the creation of a robust functional platform. Still, the core features that were identified at the beginning of the project are implemented. For this prototype, no user management was implemented as it wasn't a priority for the project.

Recommendation

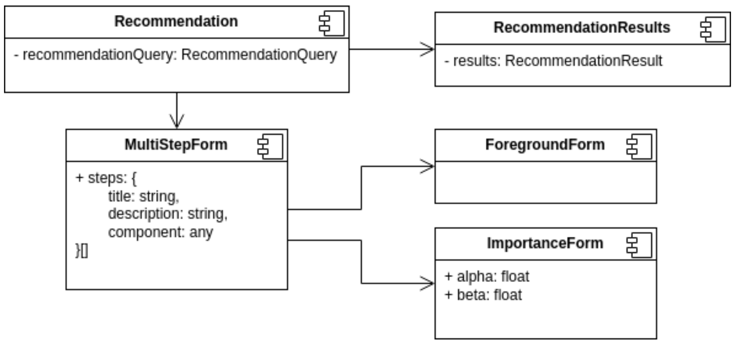
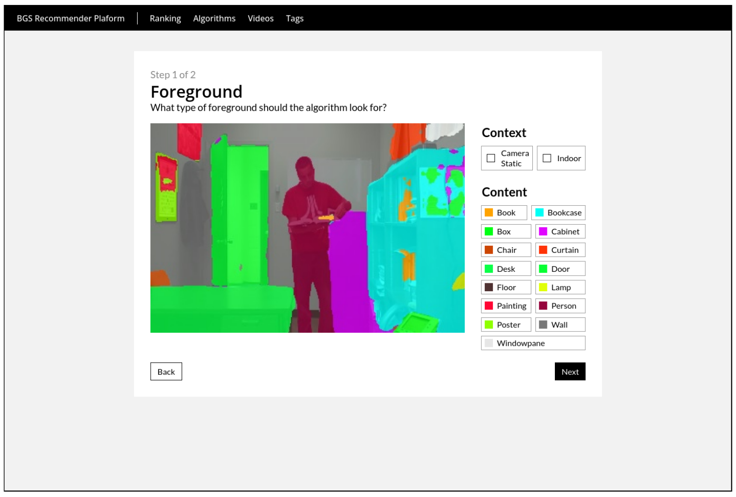


Figure 4.21. Diagram showing the components that are used in the recommendation pages

The recommendation form is the part of the UI that corresponds to the industrial user use case. The whole recommendation feature is implemented by the combination of three components: the `MultiStepForm` which groupes the `ForegroundForm` and `ImportanceForm` components together in a single form, and the `RecommendationResults`.

The `ForegroundForm` component is used to define the tags that are important to the user. These include information about the content, and/or the context of the video sequences that will be used in the ranking. As shown by figure 4.22, the user is presented with a video frame that has been segmented using a semantic segmentation algorithm. The segmented image is presented along with a list of tags present in the video associated with the proper colours.

The `ImporanceForm` is used to obtain the α and β coefficients used to weigh the components of the confusion matrix. In this part of the form, the user is presented with the ranking tile, accompanied by two sliders that allow to have a finer control over the values. The main components used in this section are the `Tile`, and the `Slider` components.



Step 1 of 2

Foreground

What type of foreground should the algorithm look for?

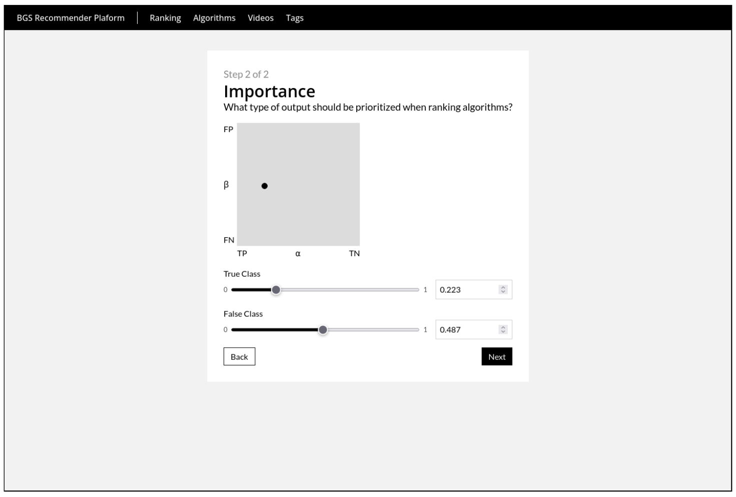
Context

☐ Camera Static ☐ Indoor

Content

🟧 Book	🟦 Bookcase
🟩 Box	🟪 Cabinet
🟫 Chair	🟥 Curtain
🟩 Desk	🟩 Door
🟫 Floor	🟨 Lamp
🟥 Painting	🟥 Person
🟩 Poster	⬛ Wall
⬜ Windowpane	

Figure 4.22. Screenshot of the first step (the context step) of the recommendation form.



Step 2 of 2

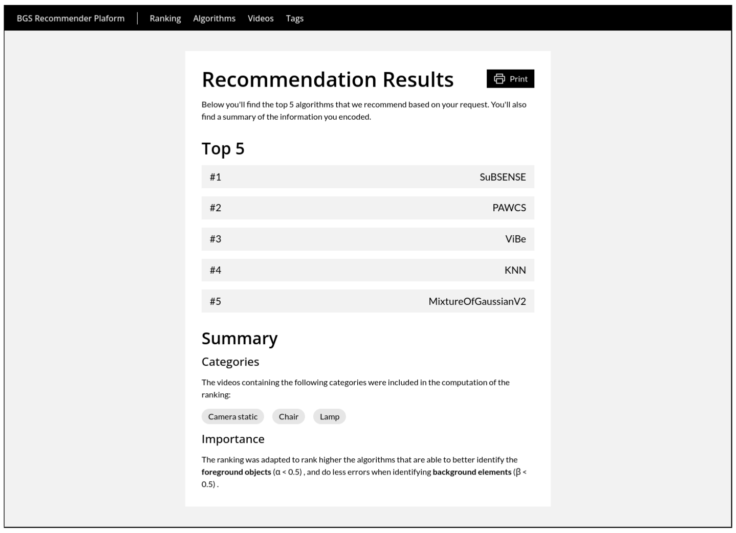
Importance

What type of output should be prioritized when ranking algorithms?

True Class 0 — 1: 0.223

False Class 0 — 1: 0.487

Figure 4.23. Screenshot of the second step (the importance step) of the recommendation form.



Recommendation Results

Below you'll find the top 5 algorithms that we recommend based on your request. You'll also find a summary of the information you encoded.

Top 5

#1	SuBSENSE
#2	PAWCS
#3	ViBe
#4	KNN
#5	MixtureOfGaussianV2

Summary

Categories

The videos containing the following categories were included in the computation of the ranking:

Camera static · Chair · Lamp

Importance

The ranking was adapted to rank higher the algorithms that are able to better identify the **foreground objects** ($\alpha < 0.5$) , and do less errors when identifying **background elements** ($\beta < 0.5$) .

Figure 4.24. Image showing the page obtained when completing the recommendation forms

Lastly, when the user has encoded his preference, he is presented with the last part of the recommendation process which is the `RecommendationResults` page. This page displays a simplified version of the ranking that obtained by the query. It also provides the user with a summary of the query that was just performed, along with human-readable explanation of the recommendation. The results page is designed to be printable, removing irrelevant elements such as the navigation bar to improve the clarity of the print.

Ranking

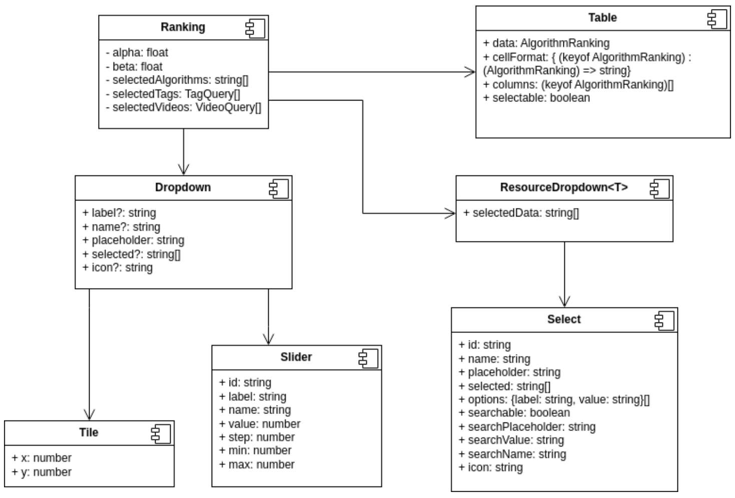
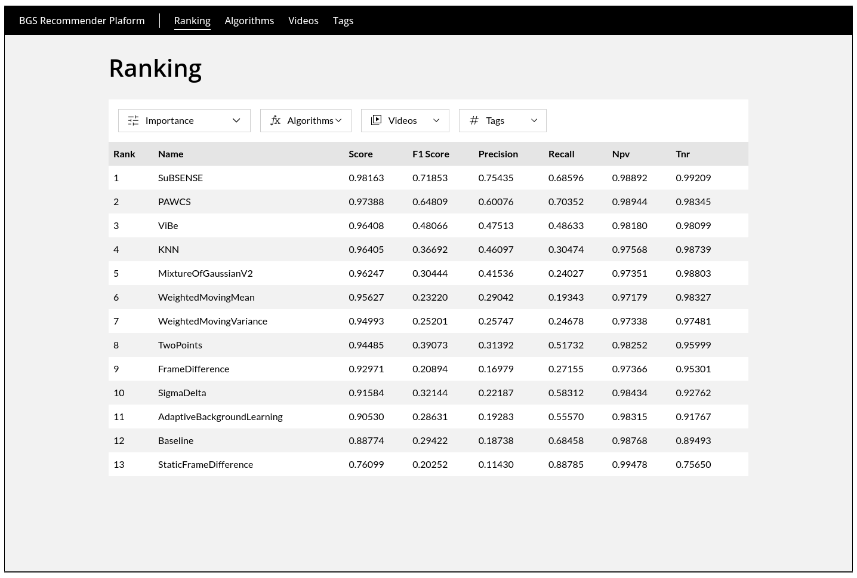


Figure 4.25. Diagram showing the components used in the ranking page

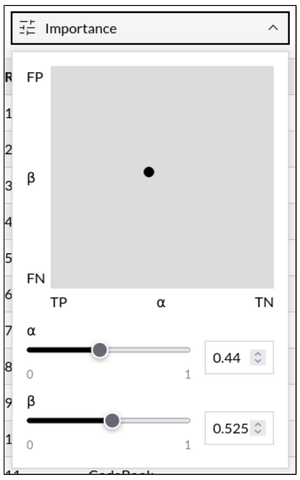


Ranking

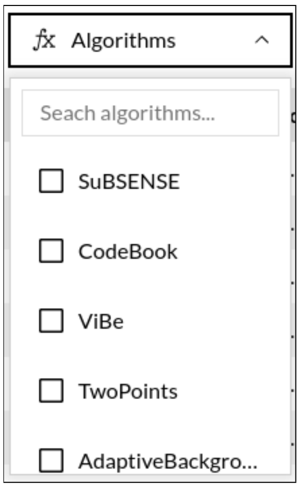
Rank	Name	Score	F1 Score	Precision	Recall	Npv	Tnr
1	SuBSENSE	0.98163	0.71853	0.75435	0.68596	0.98892	0.99209
2	PAWCS	0.97388	0.64809	0.60076	0.70352	0.98944	0.98345
3	ViBe	0.96408	0.48066	0.47513	0.48633	0.98180	0.98099
4	KNN	0.96405	0.36692	0.46097	0.30474	0.97568	0.98739
5	MixtureOfGaussianV2	0.96247	0.30444	0.41536	0.24027	0.97351	0.98803
6	WeightedMovingMean	0.95627	0.23220	0.29042	0.19343	0.97179	0.98327
7	WeightedMovingVariance	0.94993	0.25201	0.25747	0.24678	0.97338	0.97481
8	TwoPoints	0.94485	0.39073	0.31392	0.51732	0.98252	0.95999
9	FrameDifference	0.92971	0.20894	0.16979	0.27155	0.97366	0.95301
10	SigmaDelta	0.91584	0.32144	0.22187	0.58312	0.98434	0.92762
11	AdaptiveBackgroundLearning	0.90530	0.28631	0.19283	0.55570	0.98315	0.91767
12	Baseline	0.88774	0.29422	0.18738	0.68458	0.98768	0.89493
13	StaticFrameDifference	0.76099	0.20252	0.11430	0.88785	0.99478	0.75650

Figure 4.26. Image showing the ranking page of the platform.

The ranking page displays the algorithms in descending order of their ranking score, using the `Table` component. The table entries are defined by the `AlgorithmScoreDto` object which contains the algorithm's name along with its ranking score, and other classical metrics (e.g. F1 score) used for comparison. The table is accompanied by four dropdown menus used to refine the ranking. These menus, in order, are dedicated to the ranking coefficients, the filtering of algorithms, the weighting of videos, and weighting of tags. The



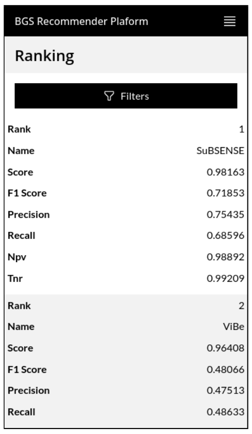
(a) Image showing the dropdown containing the ranking tile.



(b) Image showing the dropdown used to filter algorithms in the ranking

Figure 4.27. Example of filters used in the ranking page.

filters will affect the order of the ranking by changing the weights used in the ranking computation, except for the algorithm filter which simply selects the algorithms to display in the ranking.


BGS Recommender Plaform

Ranking

Filters

Rank	1
Name	SuBSENSE
Score	0.98163
F1 Score	0.71853
Precision	0.75435
Recall	0.68596
Npv	0.98892
Tnr	0.99209
Rank	2
Name	ViBe
Score	0.96408
F1 Score	0.48066
Precision	0.47513
Recall	0.48633

Figure 4.28. Image showing the mobile version of the ranking page.

The mobile version of the ranking page (figure 4.28) is very similar to the desktop version. The main difference is that the filters are collapsed in a button which opens a modal[15] containing the filters. The table is also displayed in a list form. The dropdown containing the importance tile is displayed along with the other filters in the modal.

Resources Management

The last feature implemented on the platform is the resource management. This regroups the management of algorithms, videos, and tags of the platform. The pages related to resources are built using the same set of components: a `Table` that displays the information about the resource, a set of `Modals` that are triggered by clicking on the action buttons

[15]Modal: a modal is a dialog view that overlays the content of a screen until the action it requires is completed or aborted

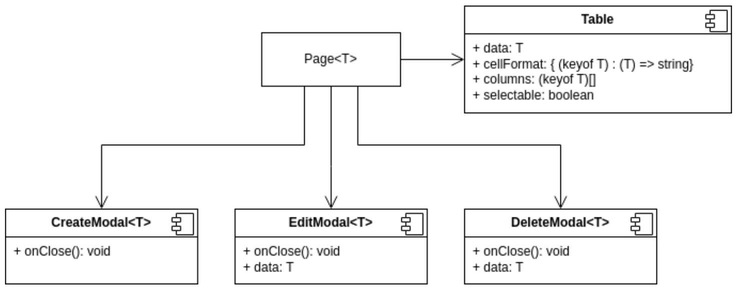
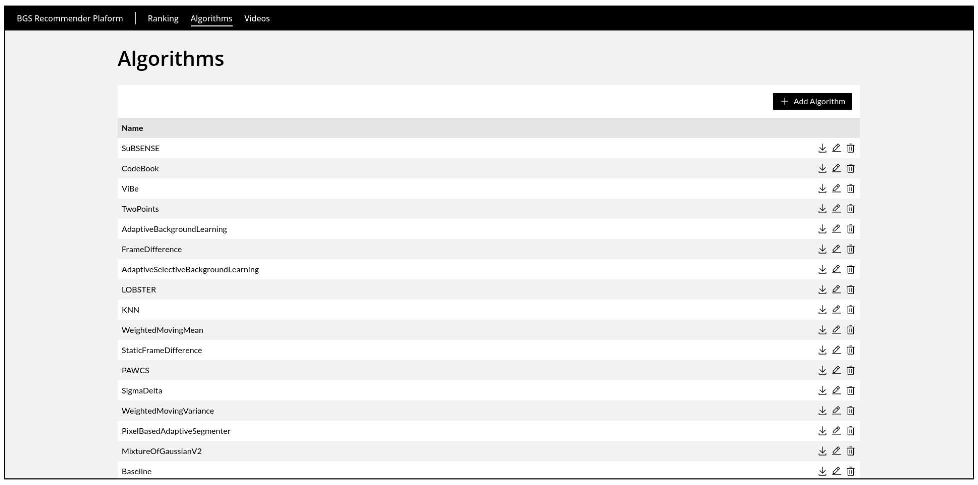


Figure 4.29. Diagram showing the dependencies of a page dedicated to resource management. The T type represent a generic type which is to be replaced with the type of resource being handled (i.e. video).

present in each entry of the table, and an add button that is present on top of the table and allow to create a new resource.

Resource deletion When performing delete operations, the user is always prompted with a confirmation dialog (see fig. 4.32c and 4.31d). This is made to avoid accidental deletion of important resources, as recovery of removed data isn't possible.



Algorithms

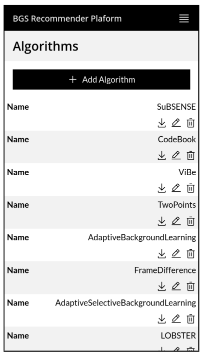
Name
SuBSENSE
CodeBook
ViBe
TwoPoints
AdaptiveBackgroundLearning
FrameDifference
AdaptiveSelectiveBackgroundLearning
LOBSTER
KNN
WeightedMovingMean
StaticFrameDifference
PAWCS
SigmaDelta
WeightedMovingVariance
PixelBasedAdaptiveSegmenter
MixtureOfGaussianV2
Baseline

Figure 4.30. Image showing the desktop version of the algorithms page of the platform.

Algorithms

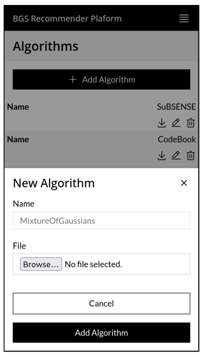
All logic related to the management of algorithms can be accessed from the algorithms page (figure 4.30). This page contains a very simple table view that shows all the algorithms that have been added to the platform. The table lines contain the name of the algorithm, its status, and action buttons. The actions that can be performed on an algorithms are the execution of the algorithm (on all videos), the edition of its information, its deletion, and the downloading of its files. When creating or editing an algorithm, the user has to fill in its name, and provide a file for the algorithm. The creation form also requires the user to decide whether or not the algorithm should be scheduled immediately upon creation.

Figure 4.31 shows four screenshots of the mobile version of the algorithm view. On these images a few differences can be noticed. Firstly, the navigation has been collapsed



Name	SuBSENSE
Name	CodeBook
Name	ViBe
Name	TwoPoints
Name	AdaptiveBackgroundLearning
Name	FrameDifference
Name	AdaptiveSelectiveBackgroundLearning
Name	LOBSTER

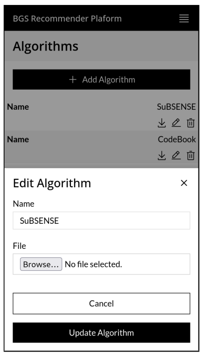
(a) Image showing the mobile version of the table contatinting the algorithms.



MixtureOfGaussians

Browse... No file selected.

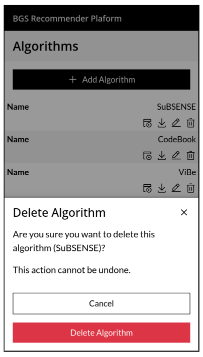
(b) Image showing the mobile version of the modal used to create a new algorithm.



SuBSENSE

Browse... No file selected.

(c) Image showing the mobile version of the modal used to edit an algorithm.



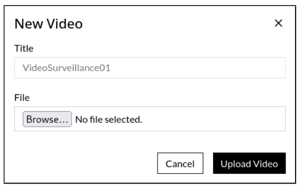
Are you sure you want to delete this algorithm (SuBSENSE)?

This action cannot be undone.

(d) Image showing the mobile version of the confirmation shown when deleting an algorithm

Figure 4.31. Mobile views of the algorithms page and modals.

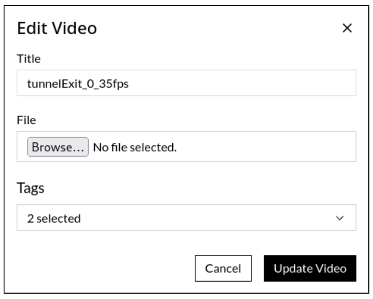
in a hamburger menu, which is a common pattern used to avoid clutter in the navigation header. Secondly, the modals are anchored to the bottom of the view, and take the full width of the screen. Beyond these adaptations, the mobile interface incorporates font sizes modification, making the text more readable, and adapts the size of the buttons for an enhanced usability on touchscreens.



VideoSurveillance01

Browse... No file selected.

(a) Image showing the modal used to upload a new video.

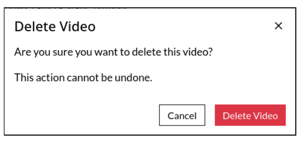


tunnelExit_0_35fps

Browse... No file selected.

2 selected

(b) Image showing the modal used to edit a video.



Are you sure you want to delete this video?

This action cannot be undone.

(c) Image showing the confirmation modal that appears when deleting a video.

Figure 4.32. Images showing the modals used to create, and edit a video.

Video

Similarly to the algorithms, the video table allows the user to access the creation, edition, and deletion modals (figure 4.32). The creation form includes a field to input the title, and a file input to upload the video. On the editing form, the user can also change the title of the video, and he is also able to assign tags to it. These modals are toggled using the "Add Video" button, and the edit button of the videos table respectively.

Tags

The tag management part is exactly similar to the algorithm and video parts. The only difference lies in the actions that can be performed on a tag. Namely, a tag can only be added, edited, or deleted.

4.7 Data Management

From the beginning of the project, the data management was a key aspect of the platform. The data is the cornerstone of the application, making the choice of storage system crucial. Indeed, having a poorly chosen data storage medium can have a significant impact on all the services of the application. At first, the choice of the storage type was to use a file-based system in the form of a csv[16] file storage. However, as the system grew in complexity, it became clear that a relational database would be a better fit for the use case of the platform. A few reasons motivated this choice:

- Firstly, the use of a database alleviates the need for manual management of concurrent data access. This responsibility is delegated to the DataBase Management System (DBMS). In the context of the BGS Recommender Platform, the data is very likely to be accessed by multiple services simultaneously (notably by the workers). Using a database avoids the implementation a manual locking mechanism to ensure data consistency, due to the ACID[17] properties of databases,
- Secondly, a relational database allows to benefit from the relations between entities. For the platform, this means that the data models can easily be linked together. Once again, this is a feature that prevents from having to implement manually a relationship system for a potentially complex data model.

4.7.1 Data Model

The data model of the platform is relatively simple. It is composed of four main entities: the algorithms, the videos, the performances, and the tags (figure 4.33). The first element that can be noticed from this diagram is that it is relatively simple, and has "only" a few entities. Notably, it contains four schemas that are linked together as follows:

- `PerformanceSchema`: A performance is linked to a single video, and a single algorithm. It is used to represent the result of the execution of an algorithm on a video. This result is stored as a mask (represented by the "path" field). It also contains a confusion matrix that is computed from the mask,
- `VideoSchema`: A video is linked to multiple performances as it will be used to store the results of the execution of multiple algorithms. This link is however implicit as the video has no information about the performances it is linked to. The video also contains a list of tags that are associated with it,
- `AlgorithmSchema`: Similarly to the video, an algorithm is implicitly linked to multiple performance. The algorithm is used to represent the BGS algorithm that is used to segment the video. It only contains an id, name, and a path to the file that contains the algorithm,
- `TagSchema`: A tag is implicitly linked to multiple videos. As already mentioned, it is used to define the context, and content of the video. It contains an id, and a label,
- `RecommendationSchema`: The recommendation made to the user is linked to several tags that were specified in the query. In the diagram they are defined as `TagQuery`

[16]CSV: Comma Separated Value

[17]ACID: Atomicity, Consistency, Isolation, and Durability. Properties of DBMS transaction that ensure the reliability, and accuracy of operations.

objects which are a modified version of the `TagSchema` that is associated with a weight.

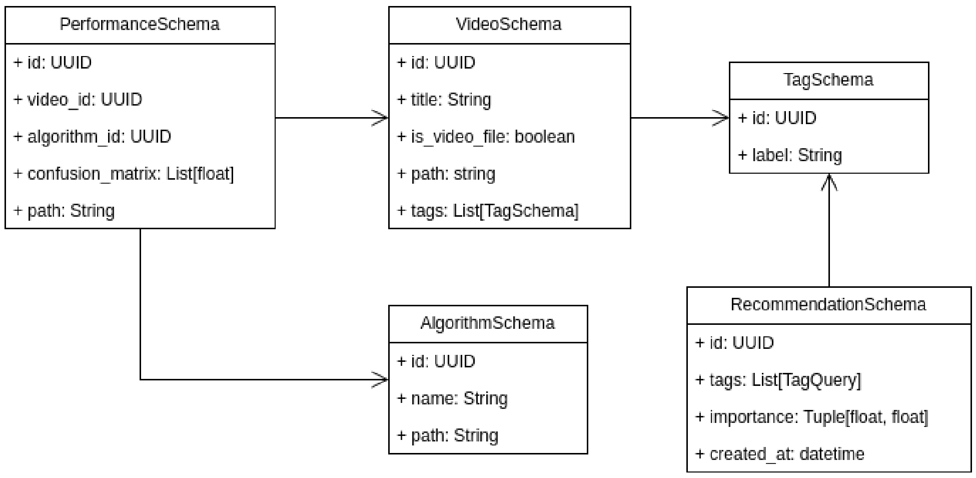


Figure 4.33. Diagram of the data model of the platform.

The schemas all share common properties. They all include a universally unique identifier (UUID) used to identify the object. A few of them also contain a "path" field which is defined as a string, and contains the path to a file which is bound to the resource (i.e. the frames of the videos, the code of the algorithm, etc.). The file itself is defined by a relative path and is potentially not stored locally on the file server. Using relative paths offer a few advantages compared to absolute paths: it facilitates seamless file relocation, and ensures that the services are using these files remain agnostic of their actual location. In the case of the platform, it is the file server who has the responsibility to store, and resolve the paths of the files. This allows for a greater flexibility in the storage.

4.8 Algorithms

To ensure the good functioning of the ranking/recommendation process, and of the algorithm execution, several algorithms needed to be added to the platform. For this purpose, the bgslibrary package was used. This package contains a plethora of BGS algorithms that can be used out-of-the-box, and can easily be parametrised. Specifically, 16 of its algorithms have been adapted and integrated into the platform's data. Additionally, a baseline was implemented for comparison purposes.

BGSLibrary For the library to be usable by the workers, its algorithms have been divided into atomic chunks, each of which containing its own algorithm. A chunk is a Python module that contains the minimal set of files required to run the algorithm. These files are the configuration file required by the bgslibrary to set the algorithms' parameters, the actual Python program that is executed, and a file containing the dependencies to be installed before running the algorithm.

4.8.1 Baseline

Having a baseline is a requirement for any algorithm benchmark as it allows for new algorithms to be compared against, and assert their performance. The baseline constitutes a minimum standard for the algorithms to be considered usable. As such, it needs to be a simple algorithm. For this thesis, the algorithm that was implemented is the Static Frame Differencing. Its principle is very simple, the algorithm keeps a reference frame from the video sequence, and performs a subtraction with the current frame. Pixels are then classified based on a threshold that determines the minimal change required for a pixel to be considered in motion. Formally, this algorithm is expressed as follows:

$$|I_t - B| > Threshold \tag{4.2}$$

, where I_t is the value for a pixel at time t, and B the value of the pixel from the reference frame. The implementation made for this work uses an arbitrary threshold value of 35, and uses the first frame of each video as reference for the background.

Chapter 5

Results

In order to assess the quality of the recommendation and ranking system, an evaluation protocol had to be implemented. This protocol was integrated into the API along with the other features. This section describes the evaluation method that was set up for the project. It also discusses the results obtained from the experiments.

For these experiment, the entire CDNet2014[7] was used. The dataset was curated, and the categories were replaced by the newly built tag system. Each of its videos were reassigned categories which are a combination of contextual categories (BGS challenges, sequence setup, etc.), and semantic labels extracted from the ADE20K dataset that relate to the content of the videos. The following tags were defined and used in the experiments:

- Context: dynamic_camera, static_camera, indoor, outdoor, and shadows.
- Content: car, person, road, boat, tree, and water.

Regarding the algorithms, the experiments are based on a set of 17 methods, 16 of which were obtained from the bgslibrary Python package, the remaining one being the baseline described in chapter 4. The number of algorithms from the library was deliberately restricted due to technical limitations which prevented the use of the complete pool provided by the BGSLibrary, reducing it to the following list:

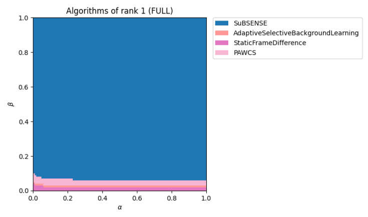
- Adaptive Background Learning
- Adaptive Selective Background Learning
- Baseline
- CodeBook
- Frame Difference
- KNN
- LOBSTER
- Mixture Of Gaussians V2
- PAWCS

- Pixel Based Adaptive Segmenter
- Sigma Delta
- Static Frame Difference
- SuBSENSE
- Two Points
- ViBe
- Weighted Moving Mean
- Weighted Moving Variance

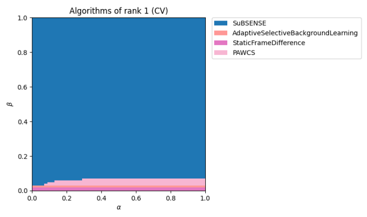
The experiment was conducted across a discretized version of the ranking tile. Specifically, the α and β coefficient were considered on the interval $[0, 1]$ with a step size of 0.01, computing the ranking for each point.

5.1 Ranking Stability

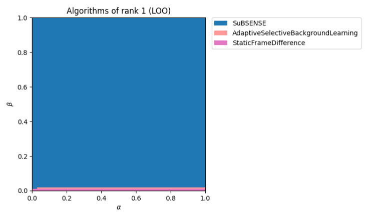
The objective of this experiment was to show the stability of the ranking process over the whole dataset (and potentially over unseen data). This was achieved by comparing the ranking obtained from the whole dataset to a ranking obtained by cross validating the scores achieved on subsets of the entire data (folds). In that effort, three types of rankings were computed: complete, cross-validation (CV) with ten folds, and leave-one-out-cross-validation (LOO) where each fold corresponds to one video. All of the rankings were evaluated for the entire tile.



(a) Ranking tile for algorithms of rank 1 obtained from the full dataset



(b) Ranking tile of algorithms of rank 1 obtained from 4 fold cross-validation

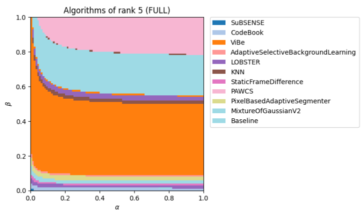


(c) Ranking tile of algorithms of rank 1 obtained from LOO cross-validation

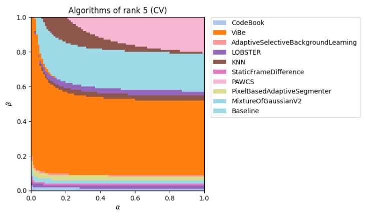
Figure 5.1. Ranking tiles of algorithms of rank ones for full, CV, and LOO

Figure 5.1 shows the ranking tile of the top-ranked algorithms for various values of α and β. The plot displays consistent rankings across full dataset, cross-validated, and LOO approaches. Exception made for PAWCS algorithm which is only absent from the LOO tile. Overall, confirming the stability of the ranking for the first rank. A similar comparison can also be observed for the last ranks.

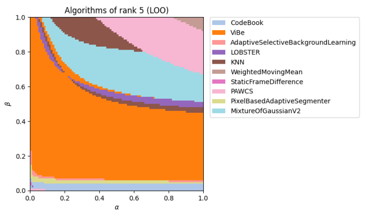
Although some small instabilities can be observed, the ranking of the intermediate ranks remain relatively stable. This is particularly visible from fig. 5.2 where KNN is a lot more present in the CV and LOO tiles. Those instabilities are most likely due to



(a) Ranking tile for algorithms of rank 5 obtained from the full dataset



(b) Ranking tile of algorithms of rank 5 obtained from 10 fold cross-validation



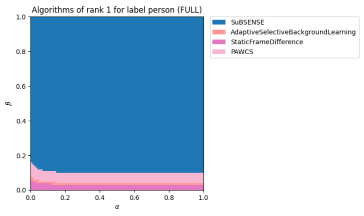
(c) Ranking tile of algorithms of rank 5 obtained from LOO cross-validation

Figure 5.2. Ranking tiles of algorithms of rank 5 for full, CV, and LOO

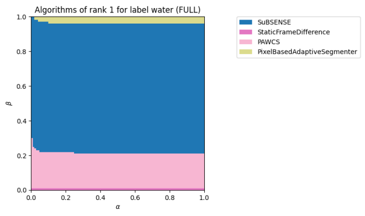
the cross-validation process in which the videos are grouped into folds. Depending on the ordering split, some categories might be more prominent in some folds, ultimately introducing a bias towards algorithms that perform well in those categories that would affect the final ranking.

5.2 Semantic Classes Utility

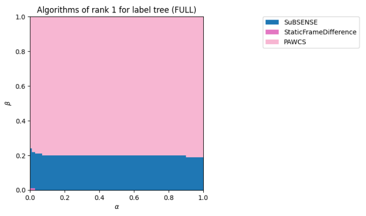
The second experiment that was conducted had the goal to prove the utility of the use of semantic labels in the recommendation process. For that purpose, semantic classes were added to the tags of the platform (see experimental setup at the beginning of this section). This experiment was achieved by comparing the rankings obtained without specifying any class, to rankings obtained by specifying the labels. All of the labels were weighted equally with a value of 1.



(a) Ranking tile of the algorithm of rank 1 for the person category



(b) Ranking tile of the algorithm of rank 1 for the water category

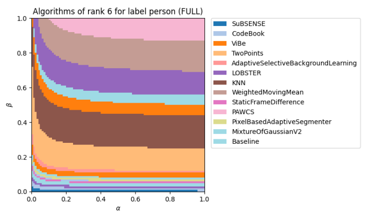


(c) Ranking tile of the algorithm of rank 1 for the tree category

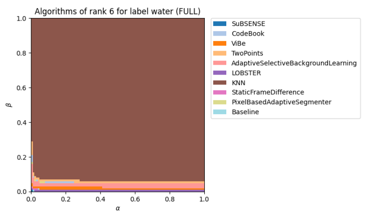
Figure 5.3. Ranking tiles of the algorithms of rank 1 for semantic labels person, tree, and water.

As shown by figure 5.3, the first rank obtained for each semantic label roughly stays similar to the ranking of the full dataset, with some algorithms such PAWCS taking more space on the tile in most categories, especially for the "tree" label where it covers the majority of the tile (as opposed to SuBSENSE for the other cases). Some categories displays others algorithms being ranked first as well for specific categories (i.e. PixelBasedAdaptiveSegmenter for water label). This behaviour hints at the utility of using semantic labels to further fine-tune the recommendation process.

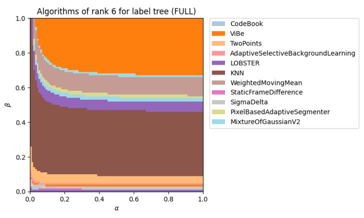
Similar to the stability experiment, the intermediate ranks for the semantic classes are shown to significantly differ from the ranks of the full dataset. These differences ranges



(a) Ranking tile of the algorithm of rank 5 for the person category



(b) Ranking tile of the algorithm of rank 5 for the water category



(c) Ranking tile of the algorithm of rank 5 for the tree category

Figure 5.4. Ranking tiles of the algorithms of rank 5 for semantic labels person, tree, and water.

from simple area modification, to complete order change. However, those results once again have to be carefully considered as the experiments only featured a restricted amount of semantic classes, which lead to significant reduction of the amount of videos considered in some cases.

Chapter 6

Discussion

This part of the report is dedicated to performing a retrospective analysis of the project. In particular, it will provide insights into the project strengths, and potential pitfalls of the implementation. The chapter will also include a brief discussion of improvements that could be made to the current platform.

6.1 Retrospective Analysis

The project was initiated with the goal of providing a novel way to recommend background subtraction algorithms. The idea was to provide a recommendation tool that is capable of using user input to fine-tune the ranking of algorithms, guiding him through the selection process. Additionally, the platform was also designed to be extendble by allowing the addition and execution of new algorithms while providing insight on their performance against past requests. It also allows the dataset to be extended by the addition of new videos, and categories. Feature-wise, the implemented platform fulfils all of the initial requirements with some caveats.

Firstly, it is able to provide a recommendation that is based on a score that is function of the user input. Specifically, the user is able to mark his preference for the output of the algorithms (that is the components of the confusion matrix) by using the ranking tile. Moreover, he also has the possibility to specify the contextual, and semantic labels that are of interest to him. However, the recommendation process still has some flaws. For instance, from the UI, it is not possible to provide specific weights for the categories. Currently, the recommendation process doesn't provide a way for the user to specify technical constraints on the algorithms (i.e. memory usage, pixels treated per second, etc.).

Secondly, the platform can be augmented with new data. It allows the addition, modification, and deletion of videos, algorithms, and categories. But, no user management system was implemented in the context of the work, meaning that the platform doesn't provide a permission system that would prevent unauthorized users from modifying the data. From the definition of the use cases in the implementation details, it is clear that, apart from the administrator, the users should only have control over the resources they have created. Since this work's objectives were to prove the feasibility of the recommendation system, the user management system was left out of the scope of the project. Nevertheless, this feature would be crucial to achieve a production-ready system.

Thirdly, the platform allows the autonomous execution of algorithms. It is made possible by the use of a complex task management system that works in conjunction with Docker to provide a secure, and isolated environment for the execution. The whole platform relies on Docker Compose to manage the services. This has the advantage of being simple to deploy, and to scale, but this also enlightens some limitations of the application. Namely, the services responsible for the processing of BGS algorithms tasks must be running at all time. Meaning that it is currently impossible to start workers on the fly. Moreover, workers all need to share the same Docker configuration. In the present implementation, the workers are all built upon a specific version of Python. This could be limiting for algorithms using libraries that are not compatible with the version of Python used in the worker.

Finally, the application is able to provide insights on the potential ranking of an algorithm based on the request history. The user requests are stored and reused to obtain the history of the ranking of an algorithm based on those queries. However, this feature is only implemented on the API side and does not yet have a visual representation.

On a separate note, although promising results were obtained regarding the ranking stability and the utility of semantic labels, it's important to consider limitations. The experiments were conducted on the CDNet dataset which exhibits a plethora of categories. However, the dataset remain limited by its size and doesn't cover all possible types of real-world videos. Overall, extensive experimentations should be conducted to obtain complete and more comprehensive results.

6.2 Improvements

Building upon the shortcomings identified by the previous section, the platform could be improved by addressing the aforementioned issues. Following are some enhancements that could be made. These improvements cover the current infrastructure, as well as potential new features that could complement the current work.

- **Cluster management system**: The use of a cluster management system such as Docker Swarm, Kubernetes, or Openshift would provide a more robust, and scalable architecture. This would make the platform more resilient to failures, and allow the workers to be started on the fly, preventing the waste of resources when the platform is not in use. This solution would also be suitable for further customisation of the workers' environment.
- **Pseudo ground truth production**: In its current state, the platform relies on manually labelled data to provide a recommendation. As previously mentioned, obtaining ground truth for video sequences is a time-consuming task. A potential improvement would be to use a semi-supervised learning approach to generate an empirical ground truth for the data. Allowing the use of non BGS specific datasets. Similarly, the task of assigning tags to the videos could be helped by using the semantic worker.

- **Permission system**: The platform should be equipped with a robust permission system that would prevent the modification of the data by unauthorised users, and would allow users to contribute to the platform while keeping control over their resources.
- **Logging**: The platform should feature a centralised logging system that would allow the administrators to monitor the activity of the services. Easing the detection of potential issues, and providing insights on the usage of the platform.
- **Natural Language Processing**: The recommendation system could be further enhanced by the use of Large Language Models (LLM). These models could be used to provide a text-based way of specifying the user preferences, rather than using a predefined form. Moreover, this process could also be used in the explanation given to the user at the end of the recommendation process.
- **Generic recommendation system**: The current recommendation process could be extended other domains. For instance, the system could be used to recommend algorithms related to semantic segmentation, or object detection. This would require slight changes in the handling of tasks, and possibly the definition of new types of workers.
- **Proxy file server**: File storage is a crucial part of the project. Currently, it is completely local, and handled by a FastAPI service. This could be further improved by allowing multiple storage locations (e.g. S3 bucket, Azure Blob Storage, etc.). Using the file server as a proxy in charge of retrieving the files from the storage location to make it available to the services.
- **Advanced Network Management**: The workers have one significant limitation which lies in the current architecture that doesn't prevent one to make HTTP requests to the outside of the platform. This means that a user could upload an algorithm that could retrieve the masks from a distant server instead of executing an actual BGS method. Consequently, leading to potentially skewed ranking for such algorithms. To address that issue, the idea of using a local Docker network was considered. However, since the workers should be able to install packages on the fly, simply blocking all outgoing requests wasn't a viable option. A more suitable solution could be to set up a firewall that would only allow specific requests to go through, therefore ensuring that the results would be accurate.
- ...

Chapter 7

Conclusion

This thesis has presented a novel approach to the problem of background subtraction algorithm recommendation. Building upon the work of Piérard *et al.* on the ranking tile, and taking inspiration on well known benchmarks such as ChangeDetection, this work proposes the Background Subtraction Recommender Platform. This scalable and modular platform serves as an initial step towards a comprehensive user-focused background subtraction benchmarking system.

The key contributions of this work include the development of flexible recommendation tool that can adapt to various user inputs, the creation of an extendable platform capable of integrating new algorithms, videos, and categories, and the implementation of a web-based interface that allows to interact with the system in a user-friendly manner. By addressing the needs of non-expert users, the platform aims to simplify the BGS algorithms selection process, and therefore to foster the development of more advanced computer vision applications.

The implications of this thesis are significant, and do not limit to background subtraction. The BGS Recommender Platform offers a versatile tool that is not only focused towards research, but also towards the industry. For researchers, the system provides a versatile tool that facilitates the testing and comparison of algorithms by researchers. For industry professionals, it offers a solution for the selection of the most appropriate BGS algorithm for their specific application.

In conclusion, while many areas are still to explore, this work lays foundations for future developments in the field of background subtraction, and will hopefully contribute to the development of more robust and reliable algorithms. This thesis aspires at fostering future research and innovation in the field ultimately leading to significant advancements in the computer vision.

Bibliography

[1] Jordi Pont-Tuset et al. “The 2017 DAVIS Challenge on Video Object Segmentation”. Mar. 1, 2018. DOI: 10.48550/arXiv.1704.00675. arXiv: 1704.00675[cs]. URL: http://arxiv.org/abs/1704.00675 (visited on 11/03/2023) (page 2).

[2] Sergi Caelles et al. “The 2019 DAVIS Challenge on VOS: Unsupervised Multi-Object Segmentation”. In: *arXiv (Cornell University)* (Jan. 1, 2019). DOI: 10.48550/arxiv.1905.00737. URL: https://arxiv.org/abs/1905.00737 (page 2).

[3] Laura Leal-Taixé et al. “MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking”. In: *arXiv (Cornell University)* (Jan. 1, 2015). DOI: 10.48550/arxiv.1504.01942. URL: https://arxiv.org/abs/1504.01942 (page 2).

[4] Anton Milan et al. “MOT16: A Benchmark for Multi-Object Tracking”. In: *arXiv (Cornell University)* (Jan. 1, 2016). DOI: 10.48550/arxiv.1603.00831. URL: https://arxiv.org/abs/1603.00831 (page 2).

[5] M. Fabbri et al. “MOTSynth: How Can Synthetic Data Help Pedestrian Detection and Tracking?” In: *arXiv (Cornell University)* (Jan. 1, 2021). DOI: 10.48550/arxiv.2108.09518. URL: https://arxiv.org/abs/2108.09518 (page 2).

[6] Nil Goyette et al. “Changedetection.net: A new change detection benchmark dataset”. In: *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 2012, pp. 1–8. DOI: 10.1109/CVPRW.2012.6238919 (pages 2, 3, 11, 13).

[7] Yi Wang et al. “CDnet 2014: An Expanded Change Detection Benchmark Dataset”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014, pp. 393–400. DOI: 10.1109/CVPRW.2014.126 (pages 2, 3, 11, 13, 17, 18, 55).

[8] Francesco Ricci, Lior Rokach, and Bracha Shapira. “Recommender Systems: Techniques, Applications, and Challenges”. Nov. 2021, pp. 1–35. DOI: 10.1007/978-1-0716-2197-4\{_\}1. URL: https://doi.org/10.1007/978-1-0716-2197-4_1 (page 6).

[9] Thierry Bouwmans. “Traditional and recent approaches in background modeling for foreground detection: An overview”. In: *Computer science review* 11-12 (May 1, 2014), pp. 31–66. DOI: 10.1016/j.cosrev.2014.04.001. URL: https://doi.org/10.1016/j.cosrev.2014.04.001 (pages 6–8, 13).

[10] Chris Stauffer and W. Eric L. Grimson. “Adaptive background mixture models for real-time tracking”. In: *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)* (1999). DOI: 10.1109/cvpr.1999.784637. URL: https://doi.org/10.1109/cvpr.1999.784637 (page 6).

[11] Rudrika Kalsotra and Sakshi Arora. "A Comprehensive Survey of Video Datasets for Background Subtraction". In: *IEEE access* 7 (Jan. 1, 2019), pp. 59143–59171. DOI: 10.1109/access.2019.2914961. URL: https://doi.org/10.1109/access.2019.2914961 (page 8).

[12] Sílvio Ricardo Rodrigues Sanches et al. "Recommendations for evaluating the performance of background subtraction algorithms for surveillance systems". In: *Multimedia tools and applications* 80.3 (Sept. 29, 2020), pp. 4421–4454. DOI: 10.1007/s11042-020-09838-x. URL: https://doi.org/10.1007/s11042-020-09838-x (pages 9, 13).

[13] Carlos Cuevas, Eva María Yáñez, and Narciso García. "Labeled dataset for integral evaluation of moving object detection algorithms: LASIESTA". In: *Computer Vision and Image Understanding* 152 (2016), pp. 103–117. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2016.08.005. URL: https://www.sciencedirect.com/science/article/pii/S1077314216301138 (pages 11, 13).

[14] S. Pierard and M. Van Droogenbroeck. "Summarizing The Performances Of A Background Subtraction Algorithm Measured On Several Videos". In: *2020 IEEE International Conference on Image Processing (ICIP)* (Oct. 2020). DOI: 10.1109/icip40778.2020.9190865. URL: https://doi.org/10.1109/icip40778.2020.9190865 (page 11).

[15] Kentaro Toyama et al. "Wallflower: principles and practice of background maintenance". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision* (Jan. 1999). DOI: 10.1109/iccv.1999.791228. URL: https://doi.org/10.1109/iccv.1999.791228 (page 13).

[16] Antoine Vacavant et al. "A Benchmark Dataset for Outdoor Foreground/Background Extraction". Jan. 2013, pp. 291–300. DOI: 10.1007/978-3-642-37410-4\{_}25. URL: https://doi.org/10.1007/978-3-642-37410-4_25 (page 13).

[17] Domenico Daniele Bloisi, Andrea Pennisi, and Luca Iocchi. "Background modeling in the maritime domain". In: *Machine vision and applications* 25.5 (Dec. 2013), pp. 1257–1269. DOI: 10.1007/s00138-013-0554-5. URL: https://doi.org/10.1007/s00138-013-0554-5 (page 13).

[18] Pierre-Marc Jodoin et al. "Extensive Benchmark and Survey of Modeling Methods for Scene Background Initialization". In: *IEEE transactions on image processing* 26.11 (Nov. 2017), pp. 5244–5256. DOI: 10.1109/tip.2017.2728181. URL: https://doi.org/10.1109/tip.2017.2728181 (page 13).

[19] Xiang Zhang et al. "AGVS: A New Change Detection Dataset for Airport Ground Video Surveillance". In: *IEEE transactions on intelligent transportation systems* 23.11 (Nov. 1, 2022), pp. 20588–20600. DOI: 10.1109/tits.2022.3184978. URL: https://doi.org/10.1109/tits.2022.3184978 (page 13).

[20] Kyungnam Kim et al. "Real-time foreground–background segmentation using codebook model". In: *Real-time imaging* 11.3 (June 1, 2005), pp. 172–185. DOI: 10.1016/j.rti.2004.12.004. URL: https://doi.org/10.1016/j.rti.2004.12.004 (page 22).

[21] Z. Zivkovic and Ferdinand Van Der Heijden. "Efficient adaptive density estimation per image pixel for the task of background subtraction". In: *Pattern recognition*

letters 27.7 (May 1, 2006), pp. 773–780. DOI: 10.1016/j.patrec.2005.11.005. URL: https://doi.org/10.1016/j.patrec.2005.11.005 (page 22).

[22] Pierre-Luc St-Charles and Guillaume-Alexandre Bilodeau. "Improving background subtraction using Local Binary Similarity Patterns". In: *IEEE Winter Conference on Applications of Computer Vision* (Mar. 1, 2014). DOI: 10.1109/wacv.2014.6836059. URL: https://doi.org/10.1109/wacv.2014.6836059 (page 22).

[23] Z. Zivkovic. "Improved adaptive Gaussian mixture model for background subtraction". In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* (Jan. 1, 2004). DOI: 10.1109/icpr.2004.1333992. URL: https://doi.org/10.1109/icpr.2004.1333992 (page 22).

[24] Pierre-Luc St-Charles, Guillaume-Alexandre Bilodeau, and Robert Bergevin. "Universal background subtraction using word consensus models". In: *IEEE transactions on image processing* 25.10 (Oct. 1, 2016), pp. 4768–4781. DOI: 10.1109/tip.2016.2598691. URL: https://doi.org/10.1109/tip.2016.2598691 (page 22).

[25] Antoine Manzanera and Julien C. Richefeu. "A new motion detection algorithm based on Σ-Δ background estimation". In: *Pattern recognition letters* 28.3 (Feb. 1, 2007), pp. 320–328. DOI: 10.1016/j.patrec.2006.04.007. URL: https://doi.org/10.1016/j.patrec.2006.04.007 (page 22).

[26] Pierre-Luc St-Charles, Guillaume-Alexandre Bilodeau, and Robert Bergevin. "Flexible Background Subtraction with Self-Balanced Local Sensitivity". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops* (June 1, 2014). DOI: 10.1109/cvprw.2014.67. URL: https://doi.org/10.1109/cvprw.2014.67 (page 22).

[27] Olivier Barnich and Marc Van Droogenbroeck. "ViBe: A Universal Background Subtraction Algorithm for Video Sequences". In: *IEEE transactions on image processing* 20.6 (June 1, 2011), pp. 1709–1724. DOI: 10.1109/tip.2010.2101613. URL: https://doi.org/10.1109/tip.2010.2101613 (page 22).

[28] Andrews Sobral. "BGSLibrary: An OpenCV C++ Background Subtraction Library". In: *IX Workshop de Visão Computacional (WVC'2013)*. Rio de Janeiro, Brazil, June 2013. URL: https://github.com/andrewssobral/bgslibrary (page 22).

[29] MMSegmentation Contributors. "MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark". https://github.com/open-mmlab/mmsegmentation. 2020 (page 22).

[30] Bowen Cheng et al. "Masked-attention Mask Transformer for Universal Image Segmentation". In: *arXiv (Cornell University)* (Jan. 1, 2021). DOI: 10.48550/arxiv.2112.01527. URL: https://arxiv.org/abs/2112.01527 (page 42).