ODIN Web: A web-based tool for image annotation, inference, and model evaluation

Tesi di Laurea Magistrale in
Computer Science And Engineering - Ingegneria Informatica

Author: **Simona Malegori**

Student ID: 995216
Advisor: Prof. Piero Fraternali
Academic Year: 2022-2023

Abstract

Computer Vision is a field of study, whose purpose is to replicate, by means of Machine Learning and Deep Learning methods, the ability of humans to extract information and interpret the visual world around them. Computer Vision exploits the vast amount of imagery data available, which has to be annotated with the truth the model should learn from. Being image annotation a laborious task, several tools that provide functionalities to annotate images and create high-quality datasets have been implemented. Thanks to the progress made, models increased in complexity, achieving exceptional results in image interpretation. This, however, added complexity to the model performance evaluation necessary for their understanding and improvement. Studies on model performance analysis yielded the development of several tools that exploit "black-box" analysis techniques to achieve in-depth diagnosis of complex models. These advancements led Computer Vision to acquire an increasing impact in real-world scenarios applications. This combination uncovered the need for tools to allow individuals with no technical knowledge to approach Computer Vision models. In this context, the objective is now the implementation of tools, accessible to non-technical users, that cover the entire pipeline of model development, from image annotation and model analysis to the practical application of Computer Vision models. ODIN Web, is a user-friendly web-based tool for dataset management, image annotation related to image classification, object detection, and instance segmentation tasks, and model performance investigation, that leverages the ODIN "black-box" analysis tool. Starting from this version, this thesis expands ODIN Web by implementing a user system, for collaboration and role-based access control, integrating model inference, and providing geo-visualization functionalities for datasets of geolocalized satellite images. The relevance of the implemented tool was demonstrated by illustrating its usage in some real-application scenarios.

Keywords: computer vision, image annotation tool, model evaluation tool, inference

Abstract in Lingua Italiana

La Computer Vision è un campo di studi il cui obiettivo è di replicare, attraverso metodi di Machine Learning e Deep Learning, la capacità umana di estrarre informazioni e interpretare il mondo visivo attorno a sé. La Computer Vision sfrutta la vasta quantità di dati visivi disponibili, i quali devono essere annotati con la "verità" da cui il modello dovrebbe imparare. Essendo l'annotazione di immagini un compito laborioso, diversi tool che forniscono funzionalità per annotare le immagini e creare dataset di alta qualità sono stati implementati. Grazie ai progressi fatti, i modelli sono aumentati in complessità raggiungendo risultati eccellenti nell'interpretazione di immagini. Tuttavia, ciò ha aggiunto complessità alla valutazione della performance dei modelli necessaria per la loro comprensione e il loro perfezionamento. Studi sull'analisi della performance dei modelli hanno portato allo sviluppo di diversi tool che sfruttano tecniche di analisi "black-box" per ottenere una diagnosi approfondita di modelli complessi. Questi progressi hanno portato la Computer Vision ad acquisire un impatto sempre maggiore in scenari di applicazione reale. Questo ha sollevato la necessità di tool che permettano ad individui che non possiedono conoscenze tecniche, di approcciarsi ai modelli di Computer Vision. In questo contesto, l'obiettivo è quello di implementare tool intuitivi, che coprano l'intero processo di sviluppo di modelli, dall'annotazione di immagini e analisi di modelli all'applicazione pratica dei modelli di Computer Vision. ODIN Web, è un web-based tool intuitivo per la gestione di dataset, per l'annotazione di immagini legata alla classificazione di immagini, alla individuazione e alla segmentazione di oggetti, e per l'investigazione della performance dei modelli, funzione che sfrutta ODIN, un tool di analisi di tipo "black-box". Partendo da questa versione, questa tesi espande ODIN Web implementando un sistema di utenti, per la collaborazione e l'accesso controllato basato sul ruolo, integrando l'inferenza dei modelli, e fornendo delle funzionalità di geo-visualizzazione per dataset formati da immagini satellitari geolocalizzate. L'importanza del tool implementato è stata dimostrata illustrando il suo utilizzo in alcuni scenari di applicazione reale.

Parole chiave: computer vision, strumento per l'annotazione di immagini, strumento per la valutazione di modelli, inferenza

Contents

1 | Introduction

Computer Vision (CV) is a field of study, whose purpose is to replicate, by means of Machine Learning (ML) and Deep Learning (DL) methods, the ability of humans to extract information and interpret the visual world around them. The main Computer Vision problems that are at the basis of image interpretation are those related to the identification of objects and the detection of their position on the image. Indeed, the three most important tasks in CV are *Image Classification*, the problem of determining what object classes are present in an image, *Object Detection*, the task of identifying what objects are inside an image and where they are localized by surrounding them with bounding boxes, and *Instance Segmentation*, which is similar to the previous one but aims at precisely outlining each object instance.

Computer Vision exploits the vast amount of imagery data available, from which models can discover patterns and learn to solve specific problems. Thanks to this, and to the increasing computational power available, in recent years Deep Learning models based on Deep Neural Networks (DNNs), have achieved exceptional results in the above-mentioned tasks. This development has brought Computer Vision into real-world applications, such as environmental protection [25], medical image analysis [18], agriculture [58], transportation [32], and others. Nevertheless, Computer Vision is a complex problem and is still a matter of research.

Given the increasing complexity of DL models, a critical subject is the performance assessment necessary to understand the model behavior and improve its results. For this analysis, both "open-box" and "black-box" techniques can be applied. Although the first ensures a deep layer-by-layer diagnosis, the second allows to obtain similar insights with reduced complexity. The "black-box" analysis computes evaluation performance metrics by comparing the output of a model to the ground truth. Moreover, it makes use of extra properties associated with the data and computes their impact on the performance, to better understand the model behavior and allow for an in-depth diagnosis. Several tools have been developed that address "black-box" analysis of DNNs performance.

The training of ML and DL models requires large amounts of data, which need to be annotated to provide the model with a ground truth to learn from. The annotation of datasets, and specifically of images, is typically manual, thus is a laborious and time-consuming task. Moreover, the quality of the annotations influences the model performance, as from imprecise annotations the model will learn erratic patterns. Therefore, the data annotation process represents a critical step in the development of CV models. Over the years, several tools have been implemented addressing image annotation, by providing sets of features to annotate images and manage datasets to expedite the creation of high-quality ground truths.

As previously mentioned Computer Vision has an increasing impact in real scenarios where it can be used in practical applications. This combination has uncovered the need for tools that aim to allow individuals with no technical knowledge to approach ML and DL models. An example is that of ARPA Lombardia (Environmental Protection Agency of the Region of Lombardy), where the team of experts in environmental monitoring and protection, would benefit from a Computer Vision tool for the analysis of satellite images for illegal landfills and waste detection.

Considering the analyzed context, the objective is the development of tools that cover the entire pipeline of model development, from image annotation and model analysis to the practical application of Computer Vision models. The purpose of this thesis is to extend ODIN Web [52], a web-based tool for dataset management, image annotation related to image classification, object detection, and instance segmentation tasks, and model performance investigation, that leverages the ODIN "black-box" analysis tool [76]. This work expands ODIN Web to deliver a collaborative web tool, with dataset management and image annotation features, that implements model inference and performance analysis and provides geo-visualization functionalities for datasets of geolocalized satellite images. In particular, the contributions of this work, w.r.t. the first ODIN Web version [52], can be summarized as follows:

- Implementation of a user system, with user management and access control.
- Adaptation of the dataset management system to the new user system.
- Implementation of a model inference system with non-blocking execution.
- Implementation of a geo-visualization system for geolocalized satellite images.
- Implementation of integrations between the Annotator and the Map Visualizer.

This thesis is organized as follows:

- Chapter 2 introduces the Computer Vision concept. Then defines metrics and analyses for model performance evaluation and the relative tools available in the literature. Finally discusses data annotation and presents some available image annotation tools.

- Chapter 3 presents the ODIN Web solution and discusses the requirements at the basis of this development work.
- Chapter 4 discusses the implementation of ODIN Web by presenting the architecture, the design, the implementation technologies and the deployment infrastructure.
- Chapter 5 presents the results of the implementation of ODIN Web by illustrating the application's functionalities, and addresses ODIN Web validation.
- Chapter 6 draws the conclusion and discusses future works.

2 | Related Work

This chapter aims to present an overview of Computer Vision and Computer Vision tasks in Section 2.1, Machine Learning model performance evaluation metrics and tools in Section 2.2, and methods and tools for image annotation in Section 2.3.

2.1. Computer Vision

Computer Vision (CV) is a subfield of Artificial Intelligence (AI), whose purpose is to replicate the ability of humans to extract information and interpret the visual world around them. The interest of scientists in the human visual system dates back to the 1960s when researchers first discovered that the brain's image processing starts with simple shapes like oriented edges and lines. Later, with the emergence of the AI field of study and the development of image scanning technologies, the first Optical Character Recognition (OCR) systems were created. Over the years artificial intelligence has evolved, leading to the appearance of object and face recognition applications. Impressive advancements were made in Computer Vision with Deep Neural Networks (DNNs) which outperformed the previous methods. In particular, the Convolutional Neural Network (CNN) AlexNet [44] represented a breakthrough in the field.

2.1.1. Computer Vision Tasks

Computer Vision is based on a comprehensive set of different tasks, that can be employed in image analysis problems relevant to diverse application domains. In the following the core tasks of Computer Vision are reported and described.

Image Classification (CL) is the task of determining what object classes are present in an image. Image classification includes different sub-tasks based on the complexity of the classification task, in particular, one can distinguish between binary and multi-class problems. The *binary* classification represents the task of categorizing the image into one of two defined classes, oftentimes the problem is determining if a target object is present in the image or not. The *multi-class* problems instead apply to the cases in which the set of classes considered for the classification has more than two categories. In particular, multi-class classification can be of two types: *single-label*, the image can be labeled with only one class from the set of known categories, *multi-label*, the image can be classified as depicting one or more objects from the predefined set (Fig 2.2a).

Object Detection (OD) is the task of identifying which objects are present in an image and localizing them on the image. The position of an object is reported through a bounding-box drawn around it (Fig 2.2b).

Image Segmentation is the task of partitioning images into multiple segments belonging to certain classes. There are three formulations of image segmentation, i.e., semantic, instance, and panoptic [55]. *Semantic Segmentation* (SS) is the problem of assigning a semantic label to each pixel in an image. All pixels with the same label are considered together, and for each category, a single mask is created without any distinction between different objects of the same class (see cars or persons in Fig 2.1b). *Instance Segmentation* (IS) is the problem of detecting and outlining each object of interest in the image. The labels are assigned per-object, and a mask is created for each single object instance (2.1c). *Panoptic Segmentation* is the combination of semantic segmentation and instance segmentation. This task requires each pixel of an image to be assigned both a semantic label and an instance identifier. Pixels with the same label and identifier belong to the same object [39].

Fig 2.2 shows a comparison of the expected results, over the same image, for the main Computer Vision tasks: image classification, object detection, semantic segmentation, and instance segmentation.

Further Computer Vision tasks, that find application in many domains, can be mentioned. *Optical Character Recognition (OCR)*, one of the first tackled computer vision tasks, is the problem of recognizing characters in images and converting them into text/string [53]. *Edge Detection* is the task of detecting boundaries of objects in images and is often used as a pre-processing step for other tasks. *Human Pose Estimation* aims to locate the human body parts, estimate their configuration, and build body representation (e.g., body skeleton) starting from images or videos [85]. *Object Tracking* is the problem of identifying and tracking different objects in a video and predicting their trajectory [11]. *Human Action Recognition* is the task of recognizing a human action from a video containing the complete action execution. In case of an incomplete video sequence, the task can be extended to the prediction of the human action [40].

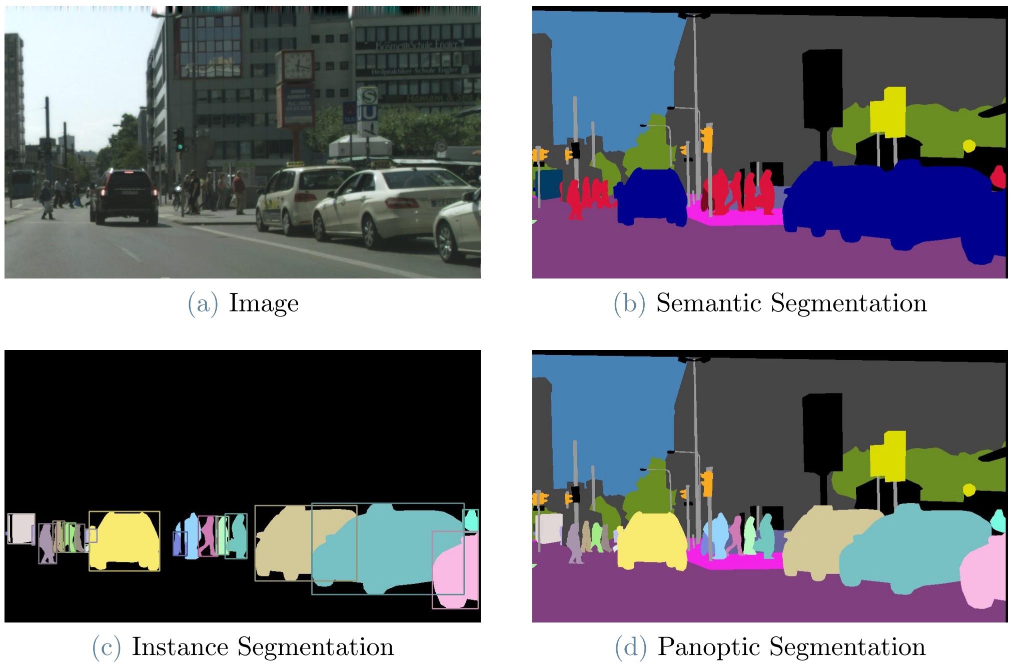
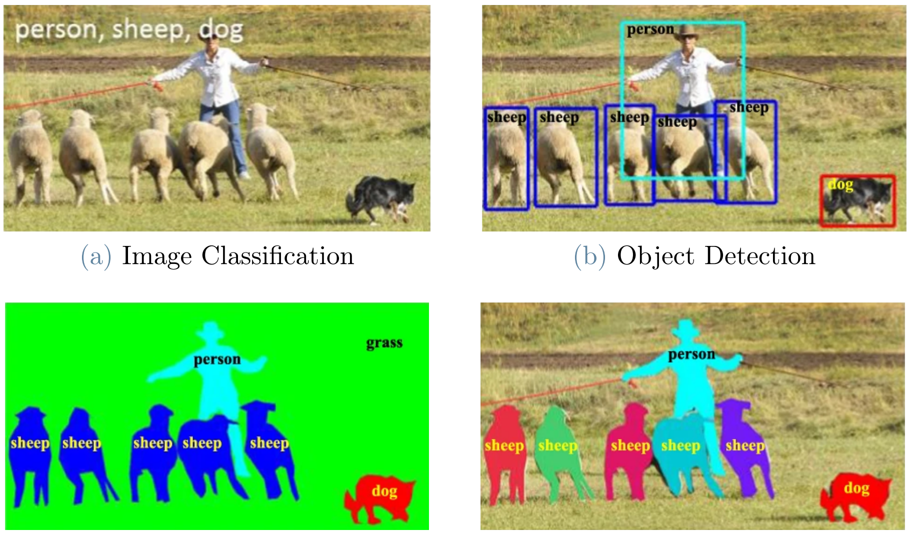

(a) Image (b) Semantic Segmentation (c) Instance Segmentation (d) Panoptic Segmentation

Figure 2.1: For a given image (a), the expected results for the three image segmentation tasks are shown: semantic segmentation (b), instance segmentation (c), and panoptic segmentation (d). Source [39].

2.1.2. Computer Vision and Remote Sensing

Remote Sensing techniques are those that use satellite and airborne sensor technologies to collect geospatial information and data about given areas. The measurements are made by sensors mounted on platforms such as satellites and unmanned aerial vehicles (UAVs), placed from a few hundred meters above the earth's surface (e.g., high-resolution multispectral and hyperspectral imagery, light detection and ranging (LiDAR), and radar systems) to hundreds, or thousands, of kilometers (e.g., orbital satellites) [8]. The use of satellite data enables the coverage of large regions, with reduced costs, and allows repeated acquisitions of the same area. Indeed, Earth Observation (EO) satellites have been deployed for decades and periodically collect earth images. Some common satellite missions are WorldView Series, GeoEye-1, Copernicus Programme, and Pléiades Missions [25].

For the large and assorted amount of geospatial data and images that it generates, Remote Sensing is extremely valuable in the field of environmental monitoring, providing an effective and economic solution for land surface observation, and in particular for solid waste detection. In [25], the authors provide a complete survey of the relevant approaches for the identification and monitoring of solid waste disposal sites and the data sets and



(a) Image Classification (b) Object Detection

(c) Semantic Segmentation (d) Instance Segmentation

Figure 2.2: An example, for a given image, of the expected results for the core computer vision tasks: image classification (a), object detection (b), semantic segmentation (c), and instance segmentation (d). Source [50].

techniques employed for such tasks. Solid waste detection can be approached through different tasks, such as:

- *landfill or sparse waste detection*, that consists in detecting the presence of solid waste in remote sensing data;
- *landfill sites monitoring*, that involves the analysis of landfills changes over time;
- *landfill sites detection and monitoring*, a combination of the previous two, that aims at landfill detection and subsequent monitoring of changes;
- *illegal dumping distribution characterization*, that aims at characterizing the distribution of illegal dumps by integrating remote sensing data with additional information such as road networks and land use maps;
- *mapping of areas with a high risk of illegal waste dumping*, that consists in defining a probabilistic map of the areas where the presence of illegal waste sites is likely;
- *waste heat contamination monitoring*, that concerns the identification of waste in the environment through thermal models of solid waste disposal sites;
- *identification of subsurface fires within landfills*, that aims at monitoring subsurface fires in landfill areas through analysis of Land Surface Temperature;
- *assessment of suitable landfill locations*, that involves analyzing and selecting suit-

able locations for constructing new landfills to minimize the environmental impact.

To address these tasks there are various techniques, that leverage different remote sensing data (e.g., aerial images, thermal parameters, and others), GIS variables, and non-geographical information. Some of the employed methods are:

- *visual interpretation*: image analysis by human experts;
- *descriptive indices extraction and analysis*: extraction of descriptive indices from multispectral images, then used for waste detection;
- *multi-factor analysis*: waste detection algorithms exploiting multi-modal inputs such as GIS data and descriptive indices;
- *features extraction and classification*: waste detection algorithms exploiting image features, such as brightness, spectral, and spatial features;
- *traditional CV techniques*: application of CV techniques to multispectral images, with tasks such as image classification, object detection, and semantic segmentation;
- *Deep Learning CV techniques*: application of CV techniques based on the recent outstanding Deep Learning architectures to optical or multispectral images.

The approaches to adopt vary significantly based on the scale of the solid waste disposal to be detected, as the characteristics that can be exploited are different. Large sites emit large quantities of gas that produce heat and thus thermal parameters and vegetation indices can be employed. Smaller sites do not have a significant impact on those indicators and therefore the analysis of optical and spectral signatures is preferable.

Traditional CV techniques for Remote Sensing Images

Several traditional Computer Vision techniques, both object-based and pixel-based, have been applied to multispectral images to perform solid waste detection. In [5] and [36] **object-based classification** is obtained through *Multi-Resolution Segmentation (MRS)*, i.e., a bottom-up region-merging technique that starts with one-pixel objects and proceeds in subsequent steps by merging smaller image objects into bigger ones. In [78], image objects are defined using *SLIC (Simple Linear Iterative Clustering)*, an unsupervised K-Means-based algorithm, that clusters pixels based on their color similarity and closeness on the image plane. In [21] the authors use a **pixel-based classification** approach on multispectral images, adopting both an unsupervised classification technique (*ISODATA*), based on the spectral response, and a *Maximum Likelihood Parametric (MLP)* supervised classifier. In [69] is presented an approach to detect floating plastic materials in the ocean through pan-sharpened hyperspectral images. It consists of a combination of an unsupervised *K-Means clustering* algorithm and a supervised classification method, the *Light Gradient Boosting Model (LGBM)*, whose final result is a probability map representing the probability that a pixel contains plastic or not.

Deep Learning CV techniques for Remote Sensing Images

Following the development of powerful Deep Learning architectures, the application of such techniques on optical and multispectral aerial images has yielded significant results in the environmental monitoring field. Different architectures have been employed for Computer Vision tasks, such as image classification, object detection, and semantic segmentation, on Remote Sensing images. Some techniques implement **pixel-level**, or patch-level, Neural Network classifiers: in [45], the authors trained a *Convolutional Neural Network (CNN)* on spatial, spectral, and temporal features of multispectral images; in [47], an *Artificial Neural Network (ANN)* is employed together with a *Decision Tree* to perform pixel classification by waste type or land use, based on multispectral channels and vegetation indices. Several architectures have been employed for **semantic segmentation** tasks: in [15], a model based on the *U-Net* [63] architecture is applied to multispectral images to produce binary masks denoting waste sites; similarly, in [60], the authors trained landfill detection models on pansharpened images, which combine multispectral images with the high-resolution details of a panchromatic band. The models are based on *U-Net* and *Fully Convolutional Network (FCN)* architectures, and use VGG [66] and ResNet [31] as feature extractors. In [81], a model based on the *DeepLabv3+* [9] architecture with *Xception* [10] backbone network is used to perform semantic segmentation on high-resolution pansharpened images to identify construction and demolition waste in urban areas. In [71], a **binary classification** model is trained on labeled optical images at different ground resolutions to perform a scene classification task. The binary classifier exploits *ResNet* as the network backbone and augments it with a *Feature Pyramid Network (FPN)* [49] architecture. The model can identify a wide variety of solid waste materials dispersed in a vast region composed of urban and extra-urban scenarios. In [87] and [68], a *CNN* **object detection** model is trained to localize landfills in urban scenarios. Specifically, in [87] a novel *Asymmetric Deep Aggregation (ADA)* block is used as the backbone to extract features of weakly visible waste. Then, an *Efficient Attention Fusion Pyramid Network (EAFPN)* merges multi-scale geospatial information using attention fusion blocks, and the resulting feature maps are fed to a *YOLO* [61] detection head for the final waste localization. In [68] the waste type is also predicted (domestic, construction, covered, and agricultural waste). The proposed architecture is based on a

ResNet backbone to extract high-level features that are connected to *Blocked Channel Attention (BCA)* modules. The network is based on a *Feature Pyramid Network (FPN)* structure to preserve multi-scale features of dumpsites.

2.2. Model Performance Evaluation

Model performance evaluation is a fundamental step in the development of Machine Learning models, and is an essential means both in the training stage and testing stage. Performance evaluation is necessary to understand how a model is performing and the direction to follow in order to optimize it, to measure the effectiveness over unseen data, and to compare different models and determine the optimal one. Model performance is measured through the computation and analysis of the so-called performance metrics defined in 2.2.1. Those metrics are also used as the basis for the implementation of performance analysis tools, some of which are described in 2.2.2.

2.2.1. Analysis and Metrics for Model Performance Evaluation

Literature presents a great number and variety of metrics including standard, commonly used, and more specific ones. Research shows that different metrics evaluate different characteristics of a model [22, 56], which is why several metrics are used for a complete analysis. Moreover, the set of metrics used can differ based on the type of machine learning task that the model is performing, e.g., classification or object detection.
Here some important concepts are introduced, and some relevant metrics are defined and described.

Threshold The threshold is the limit that defines whether a prediction is positive or negative. In particular, the threshold is the confidence value, for classification tasks, or the IOU value, for object detection tasks, above which the prediction is to be considered positive.

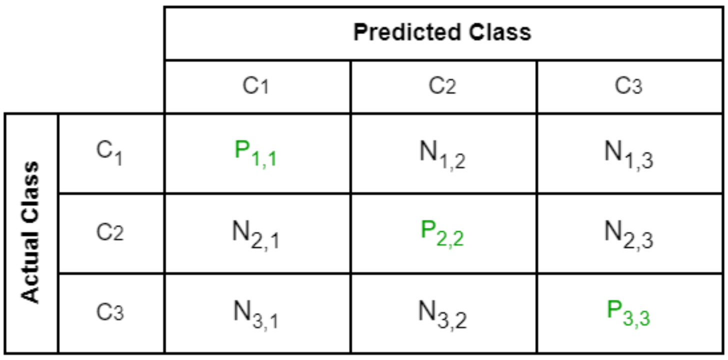
IOU (Intersection Over Union) The IOU measures the similarity between ground-truth and predicted bounding-boxes, and is obtained by dividing their overlapping area by their union area [57]:

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{B_{gt} \cap B_p}{B_{gt} \cup B_p} \tag{2.1}$$

where B_{gt} is the ground-truth bounding-box, and B_p is the predicted bounding-box.

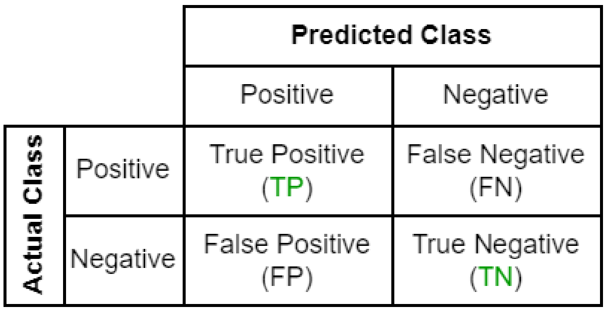
Standard Metrics

Confusion Matrix The Confusion Matrix is a common method to provide insight into the performance of a predictive classifier and in particular into which classes are predicted correctly, and which incorrectly. The matrix (Figure 2.3) has dimension n x n (with n number of classes), with the rows representing the actual classes, and columns the predicted ones. The value of cell (i, j) corresponds to the number of instances of the *i-th* class, that were classified as elements of the *j-th* class. The resulting matrix will show on the diagonal all the correct predictions of the model and in the other positions the wrongful ones.



		Predicted Class		
		C1	C2	C3
Actual Class	C_1	$P_{1,1}$	$N_{1,2}$	$N_{1,3}$
	C2	$N_{2,1}$	$P_{2,2}$	$N_{2,3}$
	C3	$N_{3,1}$	$N_{3,2}$	$P_{3,3}$

Figure 2.3: Confusion Matrix for multi-class.



		Predicted Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Figure 2.4: Confusion Matrix for binary classification.

Considering the Confusion Matrix of the binary case (Figure 2.4), with only a positive and a negative class, the following important definitions can be extracted:

- **TP (True Positive).** The number of positive instances that were correctly classified.
- **TN (True Negative).** The number of negative instances that were correctly classified.

- **FP (False Positive).** The number of positive instances that were wrongfully classified.
- **FN (False Negative).** The number of negative instances that were wrongfully classified.

For object-detection models, a prediction is correct if the predicted bounding-box sufficiently overlaps the ground-truth bounding box. As such, in a similar way TP, FP, FN can be computed, except for TN which is not applicable, given that there is an infinite number of negative bounding-boxes in each image.

Accuracy The accuracy represents the percentage of correct predictions over the total number of instances evaluated.

$$Accuracy = \frac{\text{number of correctly classified instances}}{\text{total number of instances}} \tag{2.2}$$

In the case of binary classification, the accuracy can be computed as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.3}$$

Error Rate The error rate represents the complement of the accuracy, i.e., the number of incorrect predictions over the total number of instances evaluated.

$$ErrorRate = \frac{\text{number of wrongfully classified instances}}{\text{total number of instances}} \tag{2.4}$$

In the case of binary classification, the error rate can be computed as:

$$ErrorRate = \frac{FP + FN}{TP + FP + TN + FN} \tag{2.5}$$

Precision The precision is the fraction of positively predicted instances that were accurate and represents the ability of the classifier to identify only relevant instances.

$$P = \frac{TP}{TP + FP} \tag{2.6}$$

In the case of multiple classes, the previous metrics are computed for each single class, and then micro- or macro-averaged to obtain an overall precision score. The two averages are different as the macro-average does not take class imbalance into account.

$$P_{micro} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + FP_i} \tag{2.7}$$

$$P_{macro} = \frac{1}{n}\sum_{i=1}^{n} P_i \tag{2.8}$$

Recall The recall, also called sensitivity, represents the proportion of positive instances that were correctly identified and represents the ability of the classifier to find all ground-truth instances.

$$R = \frac{TP}{TP + FN} \tag{2.9}$$

In the case of multiple classes, the previous metrics are computed for each single class, and then micro- or macro-averaged to obtain an overall recall score.

$$R_{micro} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + FN_i} \tag{2.10}$$

$$R_{macro} = \frac{1}{n}\sum_{i=1}^{n} R_i \tag{2.11}$$

F1 Score The F1 measure combines precision and recall into a single score, and is computed as the harmonic mean between precision and recall.

$$F1 = \frac{2 * P * R}{P + R} \tag{2.12}$$

In the case of multiple classes, the overall F1 score is computed by micro- or macro-averaging.

$$F1_{micro} = \frac{2 * P_{micro} * R_{micro}}{P_{micro} + R_{micro}} \tag{2.13}$$

$$F1_{macro} = \frac{1}{n}\sum_{i=1}^{n} F1_i \tag{2.14}$$

Curve Analysis

Other than by looking at standard metrics, the evaluation of a model can be done through the analysis of some curves which give insight into the relation between metrics and are especially suitable for model comparison. Among those, common curves are PR Curve, ROC Curve, and F1 Curve. Besides the visual representation, each of these curves can be summarized into a single score, the Area Under the Curve (AUC), which has been proven to be better than accuracy for evaluating and comparing classifiers [34, 35].

PR Curve The PR Curve represents the trade-off between precision and recall and is a plot of recall (x-axis) and precision (y-axis) for different probability thresholds. The reason for the PR Curve is that a good model is one able to find all ground-truth instances (no false negatives) while identifying only the relevant ones (no false positives), i.e., one with high values for both precision and recall. Thus the model can be considered good if the area under the PR curve is high. Given the trend of the PR curve, the computation of the AUC is quite challenging and requires approximations to ease the computation. Several research proposed various interpolation types to simplify the curve, e.g., two common approaches are 11-point interpolation and all-point interpolation. For the PR Curve, its AUC also corresponds to the Average Precision (AP). In the following, AP formulation with all-point interpolation is presented [57]:

$$AP_{all} = \sum_{n} (R_{n+1} - R_n) P_{interp}(R_{n+1}) \tag{2.15}$$

$$P_{interp}(R_{n+1}) = \max_{\{\tilde{R}:\tilde{R} \geq R_{n+1}\}} P(\tilde{R}) \tag{2.16}$$

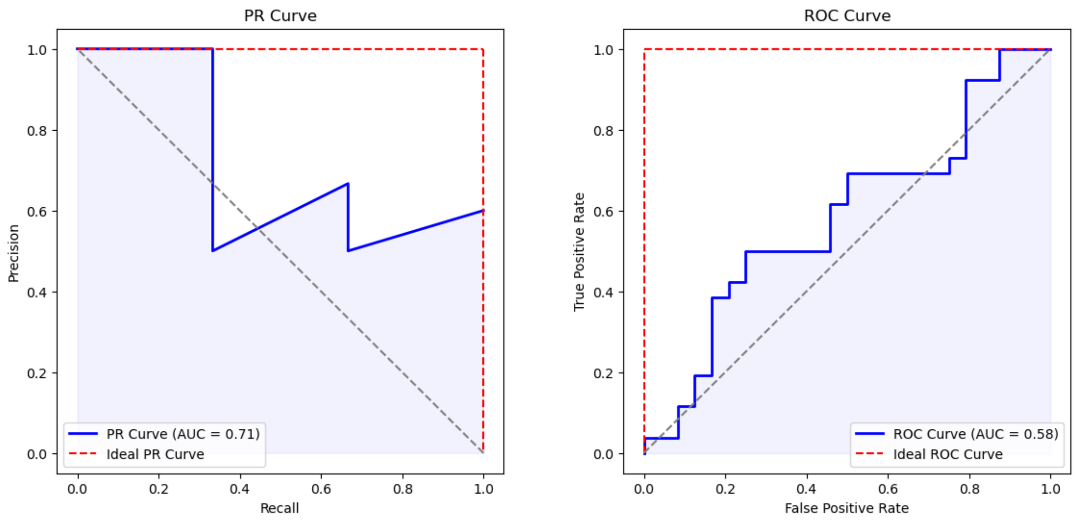
ROC Curve The ROC (Receiver Operator Characteristic) Curve plots the False Positive Rate (FPR) on the x-axis and the True Positive Rate (TPR), i.e., the recall [Eq. 2.9], on the y-axis. This curve shows how the number of correctly classified examples varies with the number of incorrectly classified negative examples [13].

$$FPR = \frac{FP}{FP + TN} \tag{2.17}$$

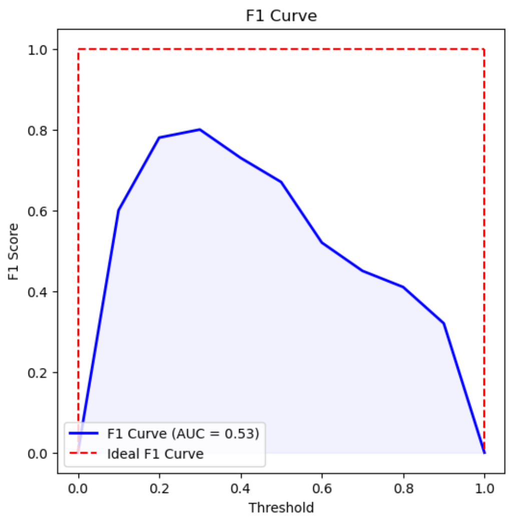
ROC Curve can be represented through a single score, its AUC.

F1 Curve The F1 Curve is represented by considering the F1 Score, on the y-axis, for each possible threshold value, on the x-axis. As for the others, it can be summarized into one value by computing its AUC.

Figure 2.5 shows an example of the PR Curve, ROC Curve, and F1 Curve: in blue the sample curve, and in light blue its AUC, in grey the curve corresponding to AUC=0.5, and in red the perfect curve. The closer a curve is to its perfect form, the better it is. The perfect PR Curve is the one maximizing both precision and recall, i.e., the curve passing through (1, 1). The perfect ROC Curve is the one maximizing the True Positive Rate while minimizing the False Positive Rate, i.e., the curve passing through (0, 1). The perfect F1 Curve is the one maximizing the F1 Score both for threshold equal to 0 and threshold equal to 1, i.e., the curve passing through (0, 1) and (1, 1).



(a) PR Curve (b) ROC Curve



(c) F1 Curve

Figure 2.5: Curve Analysis: examples of PR Curve (a), ROC Curve (b), F1 Curve (c)

Reliability Analysis

When returning a prediction, a model also reports a score defining how confident it is about the prediction, which is expressed as a probability, the likelihood, of the prediction being correct. The reliability analysis focuses on understanding whether the predicted class probabilities reflect the true likelihood, also called confidence calibration [27].

Diagrams For the reliability analysis, two types of diagrams can be employed: confidence histograms, and reliability diagrams.

Confidence histograms plot the distribution of prediction confidence, the average confidence, and the accuracy. This diagram can be used to determine if a model is either over-confident or under-confident; e.g. in Figure 2.6: on the right, the average confidence is quite greater than accuracy, the model is over-confident, on the left instead the average confidence matches closely the accuracy.

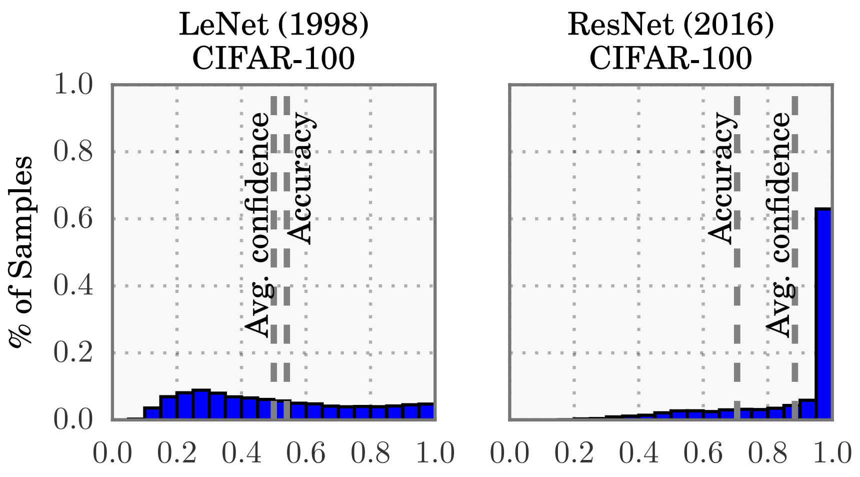


Figure 2.6: Confidence histograms of two different neural networks (LeNet, ResNet) on CIFAR-100 dataset. Source: [27]

Reliability diagrams plot the expected sample accuracy as a function of confidence. A model is well-calibrated when the confidence matches approximately the accuracy, i.e., when the bars are roughly aligned on the diagonal. Figure 2.7 shows the reliability diagrams of two models, well-calibrated (left), and miscalibrated (right).
To find the expected accuracy, the predictions are grouped into M interval bins (of size $1/M$, and the accuracy of each bin is computed.

$$acc(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} 1(\hat{y}_i = y_i) \tag{2.18}$$

where B_m is the set of indices of samples with prediction confidence into the interval $I_m = (\frac{m-1}{M}, \frac{m}{M}]$, $\hat{y}_i$ is the predicted class for sample i, and y_i is the true class. The confidence is defined as follows:

$$conf(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i \tag{2.19}$$

where $\hat{p}_i$ is the confidence for sample i.

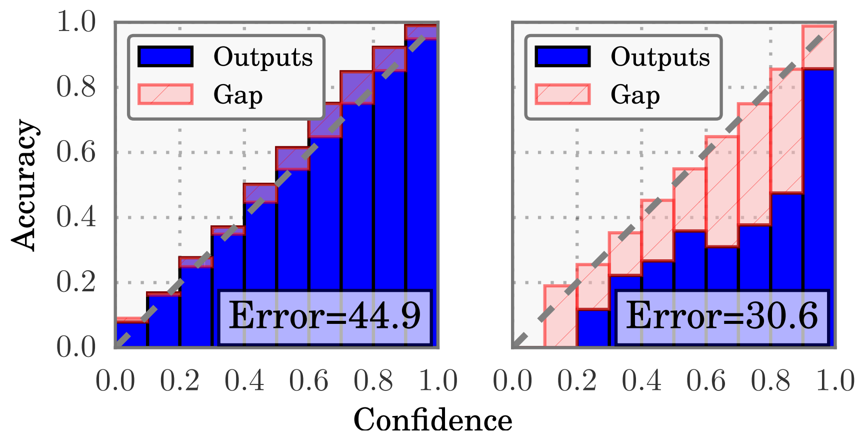


Figure 2.7: Reliability diagrams of two different neural networks (LeNet, ResNet) on CIFAR-100 dataset. Source: [27]

Summary Scores In addition to the graphical representations of calibration, scalar summary values can be convenient. Two measures of miscalibration can be considered: ECE (Expected Calibration Error) and MCE (Maximum Calibration Error), whose value for perfectly calibrated classifiers is 0.

Expected Calibration Error represents the expected difference between confidence and accuracy, which, by considering the M interval bins and n the number of samples, is defined as:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)| \tag{2.20}$$

The difference between $acc(B_m)$ and $conf(B_m)$ is the calibration gap (Figure 2.7).
Maximum Calibration Error represents the maximum difference between confidence and accuracy, useful when reliable confidence measures are required, which is defined as:

$$MCE = \max_{m \in \{1,\ldots,M\}} |acc(B_m) - conf(B_m)| \tag{2.21}$$

Other Metrics and Analysis

Normalization In the case of unbalanced classes, given that the number of True Positives is highly affected by the size of the classes, metrics depending on TP can be defined by considering a normalization of the TP instead [33].

$$TP_{norm_i} = TP_i * \frac{N}{n_i} \tag{2.22}$$

where n_i is the number of observations for class i, and N is the size of each class if the classes were balanced:

$$N = \frac{n_{tot\ observations}}{n_{classes}} \tag{2.23}$$

For example, considering 3 classes with these distributions:

$$n_1 = 100,\ TP_1 = 10 \qquad n_2 = 10,\ TP_2 = 1 \qquad n_3 = 22,\ TP_3 = 6$$

The size of the classes in a balanced scenario would be $N = \frac{132}{3} = 44$, and the normalized true positives would be:

$$TP_{norm_1} = 10 * \frac{44}{100} = 4.4 \qquad TP_{norm_2} = 1 * \frac{44}{10} = 4.4 \qquad TP_{norm_3} = 6 * \frac{44}{22} = 12$$

As expected classes 1 and 2 have the same TP_{norm} as, for both, the true positives were $\frac{10}{100} = \frac{1}{10} = 0.1\%$ of the observations; class 3 instead has a greater TP_{norm} as the ratio of observations that were TP is also greater ($\frac{6}{22} = 0.273\%$).

False Positive Analysis False positives, i.e., wrongful positive detections of classes, are one major type of error, and an analysis of this type of error is important to understand how to improve the model. There are different types of false positives, in particular, for object detection and segmentation [33]:

- *localization error*: the class is correctly predicted but the IoU is lower than the threshold (typically 0.5);
- *confusion with background or unlabeled objects*: an object is detected but is not present in the ground-truth (IoU lower than 0.1);
- *confusion with similar objects*: the object is localized but the predicted class is confused with a similar one;
- *confusion with other objects*: the object is localized but the predicted class is confused with another, not similar, one;

For classification, the false positive errors are:

- *background error*: the detection of a class for an instance whose ground-truth has no classes associated;
- *similarity error*: the detection of an incorrect class but belonging to the same group of similar classes of the ground-truth one;

- *classification error*: the detection of an incorrect class that does not fall in either of the previous cases.

FPs can be analyzed through the impact each false positive type has on the model performance, i.e., by computing for each evaluation metric the improvement that would be obtained by removing a certain error type.

Additional Properties Analysis To better understand the behavior of a model, the analysis of performance may rely also on the so-called meta-annotations or properties. The properties are supplementary annotations, not used for training, that add information to the dataset instances.

- *Per-Property Evaluation*: a deeper analysis can be achieved by computing the previously described metrics, other than on the overall dataset, also by considering specific subsets of instances having the same value for a certain property (per-property analysis).
- *Sensitivity and Impact of Properties*: further insights can be gained by analyzing the impact a property has on a certain metric. Given a metric, its score can be computed for each value of each property. The sensitivity, of the metric w.r.t. a certain property, is defined by the difference between the maximum and the minimum value obtained for the property. The impact, that a property has on a metric, is denoted by the difference between the maximum value and the overall value of the metric.

CAMs Analysis Other insights into the behavior of the model can be deduced by the visual inspection of the results (qualitative analysis). In particular, a valuable analysis is the one that considers the Class Activation Maps (CAMs) [86]. Given an image, the CAM is the matrix, of the same dimensions, that contains for each cell a value denoting the relevance of the corresponding pixel with respect to the classification target class. The CAM represented as a heatmap over the original image, helps in understanding which pixels influenced the most the model prediction.

2.2.2. Tools for Model Performance Evaluation

As previously mentioned, an essential step in Computer Vision model development is the analysis of the prediction performance, whose results are useful for model debugging and improvement. With the advances in Machine Learning, that yielded more powerful and complex architectures based on Deep Learning, this analysis and the understanding of the models' behavior have become even more essential. Two different methodologies can be adopted for performance analysis. One is the “open-box” technique, which evaluates the relationship between the input, the inner layers, and the output of the model. The other is the "black-box" method, which aims to compare the output of the model with the ground-truth computing performance metrics and to understand the model characteristics by analyzing the impact that some properties (meta-annotations) of the input, not used for training, have on the prediction performance. Several tools for the evaluation of Deep Neural Networks implement black-box analysis [24]. In the following, some of the most relevant tools are presented in chronological order and described.

- **Hoiem et al.** [33]: a pioneer work in the black-box error analysis in OD tasks which showed the utility of adding extra annotations to the input besides the training labels. In particular, it provides an analysis of the impact that a set of pre-defined metadata properties, such as size, parts visibility, aspect ratio, shape, and occlusion, have on the performance.
- **COCO API** [48]: is a framework developed together with the MS COCO dataset. It introduces the use of custom input properties as an aid for error diagnosis and differentiates the computation of mean Average Precision based on the object size (mAP_{small}, mAP_{medium} and mAP_{big}). The API also allows to load any data set, in MS COCO format, and to visualize both the images and the annotations.
- **ModelTracker** [4]: provides performance analysis, for classification tasks, through metrics summaries, visual representations, and interactive review of results. It implements analysis aids such as color-coded predictions, classification score ordering, and custom properties tagging of the input.
- **Prospector** [42]: is a web-based tool that implements a dependence technique for determining the impact of each input feature on the model results (i.e., counterfactual analysis). By applying changes to the input data, a measure of the impact on the output is provided. The tool also suggests the change in the value of each input feature that would lead to the greatest performance improvement.
- **Explanation Explorer** [43]: is a tool that provides visual analytics for the diagnosis of errors and the understanding of the behavior of binary classifiers. The analysis is in three steps, declined over three main linked interfaces. The first offers an overview of the overall performance. The second shows the results of an explanation algorithm, which represents the main model decisions and the associated statistics by groups of features (feature-level). The third allows for specific analysis at the instance level.

- **Squares** [62]: is a tool for the interactive performance analysis of multi-class single-label classifiers. It provides the visualization of prediction scores through instance-level histograms, with color codes for distinct classes, and comparison among predictions of different models.

- **DETAD** [3]: is a tool for the identification of temporal actions in videos. The tool enables the False-Positive and False-Negative analysis and the estimation of the sensitivity, of metrics based on mean Average Precision, to six action characteristics: length, context distance, agreement, coverage, context size, and number of instances.

- **Manifold** [84]: is a framework that uses visual techniques to support the evaluation and debugging of ML models. The tool provides an agreement analysis function, that allows the comparison of model pairs by highlighting the similarities and differences of their predictions, and a feature distribution function for feature-wise comparison between subsets of samples.

- **What-If Tool** [80]: the tool allows to analyze the performances of ML systems in hypothetical situations by visualizing the effect of several features on different models and different subsets of data. The tool also provides fairness analysis to detect bias in the input data set (Bias Detection) and other features such as counterfactual analysis, performance measures, and data set fairness optimization.

- **TIDE** [6]: is a tool that supports error diagnosis in object detection and instance segmentation. The tool is applicable to multiple data sets and can be applied directly to output prediction files. It provides the analysis of six types of errors, each one isolated from the others to understand its specific impact on the overall performance. TIDE also includes the comparison of the results of different models.

- **TF-GraF** [82]: is a user-friendly Tensorflow object detection graphical framework. The tool supports pre-processing, training, and evaluation with the MS COCO metrics. It allows the selection of the best-known object detection and instance segmentation architectures (Faster RCNN, SSD, Mask RCNN) through a GUI environment, letting non-experts design, train, and assess models without coding.

- **Boxer** [26]: is a tool for comparing the performances of different classifiers on the same data set. The system provides a customizable interface that allows the visualization of different metrics and graphs and the assessment of the performance of models over different data subsets.

- **OpenVINO** [14]: is an environment for analysis, optimization, evaluation, and deployment of DL models. It includes black-box analysis through metrics for classification, object detection, instance segmentation, and semantic segmentation tasks. It also supports open-box analysis, through the model execution graph (Runtime Graph), and some calibration techniques.

- **AIDeveloper** [41]: is an open-source software supporting the entire development process of an image classification model: dataset definition and visualization, model training, with various neural network architectures, model optimization, and performance evaluation. The tool presents an easy-to-use GUI that allows DL model training and analysis without coding.

Tool	Ref.	Year	Task	Media	Interface
Hoiem et al.	[33]	2012	OD	Image	No
COCO API	[48]	2014	OD, IS, PE	Image	No
ModelTracker	[4]	2015	CL	Generic	Yes
Prospector	[42]	2016	CL	Generic	Yes
Explanation Explorer	[43]	2017	CL	Generic	Yes
Squares	[62]	2017	CL	Generic	Yes
DETAD	[3]	2018	AD	Video	No
Manifold	[84]	2018	CL	Generic	Yes
What-If Tool	[80]	2019	CL, BD	Generic	Yes
TIDE	[6]	2020	OD, IS	Image	No
TF-GraF	[82]	2020	OD	Image	Yes
Boxer	[26]	2020	CL	Generic	Yes
OpenVINO	[14]	2020	CL, OD, SS, IS	Images	Yes
AIDeveloper	[41]	2021	CL	Images	Yes

Table 2.1: Black-box model performance evaluation tools

Table 2.1 presents a summary of the described tools together with their publication year. For each tool is also indicated: the supported types of media, image, video, and generic which refers to arbitrary record types, whether the tool provides a user interface or not, and the addressed task types: classification (CL), object detection (OD), instance segmentation (IS), semantic segmentation (SS), pose estimation (PE), action detection (AD), and bias detection (BD). All the tools included here are dataset-independent, i.e., allow the processing of different data sets.

2.3. Dataset Annotation

With regards to machine learning and deep learning, the datasets used for training and testing form the foundation of the model development process. Two important aspects of these datasets, that impact the model performance, are the quantity of data available, desirably large, and the quality of that data, poor quality data entry produces erratic data output ("garbage-in, garbage-out" concept). For a supervised model, the input information is the data (images, text, etc.) together with information that represents the reality the model is trained to recognize, i.e., the ground-truth. The quality of the ground-truth given in input in the training phase is critical, therefore data must be prepared and annotated as accurately as possible to provide a correct ground-truth. In [2], the authors present an experiment, to examine how the quality of annotations impacts the model performance, highlighting that an increase of wrongful annotations leads to worse model performance. Therefore, data annotation assumes a substantial role in the preparation of training and testing datasets.

2.3.1. Data Annotation

Data annotation is the process of assigning labels or adding relevant information to the data that supervised machine learning models can use in the learning phase. The target data can be of different types such as text, voice, image, and video.

Text Annotation

When considering machine learning tasks involving text, different annotation types can be applied based on the task to perform.

- *Sentiment Annotation* is the association of sentences with the sentiment/emotion corresponding to the text, it is useful for preparing data for sentiment analysis models whose task is to define whether a text is, for example, positive, negative, neutral, happy, angry, etc.
- *Intent Annotation* is the annotation of text with the intent the sentence conveys, for example, greeting, thanking, accepting, denying, and such. This data is specifically useful for developing virtual assistants and chatbots.
- *Entity Annotation* is the annotation of named entities, phrases, or parts of speech of a sentence (e.g., "NewYork" is a city, "hurricane" is an event). This is useful for developing models that extract different kinds of entities from a given text.

- *Text Classification* associates a sentence, a group of sentences, or a document with a category/topic (e.g., politics, sport, etc.,). It can be used for the machine learning classification models that classify text into categories.
- *Linguistic Annotation* refers to the annotation of the language-related elements of the text such as semantics, and phonetics. This is useful in multiple tasks as it helps understand the phonetics, the intention, and the content of the text.

Image Annotation

When building computer vision models, the data to be annotated are images. Different types of information can be used to annotate images [30]:

- *content-independent*, are information related to the image but not directly to its content (e.g., author, location, date, source, ...), also called meta-annotations/properties. Meta-annotations are not employed in the training process but are useful for the analysis of models and to give insights into the datasets of images.
- *content-dependent*, are low/intermediate level features of the image such as color, texture, etc.
- *content-descriptive*, are information related to the image's semantic content, and to the correspondence between entities in the image and real-world entities. This type of information is the most challenging one to automate and is the focus of research.

Focusing on content-descriptive annotations, those can be free-text, a textual description without a pre-defined structure, and keywords, arbitrarily chosen or pre-defined labels associated with the image or with a region of the image. For different Computer Vision tasks, content-descriptive annotations are required in distinct types/forms:

- Image Classification: requires images annotated with one or more labels or tags, typically chosen from a fixed set of predefined labels, that assess the elements present in the image, or its categorization.
- Object Detection (or Recognition): requires annotations in the form of bounding-boxes, defined by their extreme coordinates, and a label associated with each box.
- Image Segmentation: requires annotations in the form of segmentation masks, or binary masks, that delineate the exact boundaries of the objects, associated with their corresponding labels.

Video Annotation

Video Annotation has some extent of similarity with image annotation, as some machine-learning tasks are quite similar. A video can be annotated frame-by-frame, returning to the annotation of images, or, in some cases, annotations can be applied on the video itself. In any case, the annotation types are similar to the ones used for images, with labels, bounding-boxes, polygons, etc. The ML tasks usually performed on videos are:

- Detection: i.e., detecting objects in the video;
- Localization: i.e., providing the coordinates of found objects;
- Tracking: i.e., tracking an object and predicting its next location. For this task, an object must be annotated with the same identifier so that the model can track it throughout the different frames.

2.3.2. Tools for Image Annotation

The starting point to train and test a supervised computer vision model is a dataset of images, together with a ground-truth, i.e., a set of annotations over the images the model relies on for learning to perform a certain task. Over the years many datasets have been built and publicly shared, with the purpose of being employed for the construction of computer vision models (e.g., ImageNet [64], MS COCO [48], Pascal VOC [19, 20], MillionAID [51], LabelMe [65], AerialWaste [73]). Nevertheless, developers typically work on training models for specific fields or applications and must build training/testing datasets from the ground up or complement pre-existing ones with additional data. To the objective of creating a high-quality and sufficiently large dataset, different strategies can be adopted for the annotation task [51]. The annotation process can be *manual*, a strategy that assures highly accurate annotations but is labor-intensive and time-consuming, *automatic*, which is faster but reveals quality issues, and *interactive*, or semi-automatic, which benefits from both the other methods, reduces time costs by leveraging automatically produced annotations, whose are reviewed by annotators to ensure higher quality.
In this context, the development of annotation tools that seek to ease and expedite the annotation task becomes crucial. Several tools for image annotation, with various characteristics, have been published. Among other distinctive aspects, annotation tools may:

- be based on different *platforms*, for example, software-based or web-based, locally installed or online;
- cover different *annotation types*, such as image classification, object detection, and image segmentation;

- provide different *annotation instruments*, such as bounding-boxes, polygons, polylines, cuboids, etc.;
- implement various *annotation aids*, for example, automatic shape detection;
- offer *import/export* functionalities from/to different *formats*.

In the following, a number of relevant image annotation tools are presented.

- **PhotoStuff** [29]: an image annotation tool that allows to annotate images from the Web or local disk, and integrates the use of ontologies and instance knowledge bases (KBs) in the process. It allows users to describe regions of an image according to concepts in an ontology (RDFS or OWL).
- **LabelMe** [65]: a web-based tool for image and video annotation that provides an easy-to-use drawing interface. Images can be annotated by drawing polygons and labeling the objects, moreover, the tool provides functionalities to query and browse the dataset. This tool was created by the MIT Computer Science and Artificial Intelligence Laboratory to allow users to contribute to the LabelMe dataset, but a version of the tool has been released for whomever to run locally and annotate personal sets of images.
- **VoTT** [54]: is the Visual Object Tagging Tool developed by Microsoft, an annotation tool for image and video assets. It allows the labeling of images and video frames and supports importing and exporting data from local or cloud storage providers, such as Azure Blob Storage. Annotations can be exported in different formats like CNTK, PascalVOC, YOLO, JSON, and CSV. The web application is no longer available but the tool can be installed and run locally.
- **ImageTagger** [23]: an image labeling online collaborative tool. Allows the annotation of images, by drawing bounding-boxes and assigning them labels, the upload and verification of existing labels, potentially generated offline through deep learning methods, the export to user-defined formats, and the collaboration within teams.
- **ByLabel** [59]: a boundary-based semi-automatic annotation tool. Given an image, it automatically detects edge fragments that the user can group to obtain an accurate region mask (composed of one or multiple boundaries, e.g., in the case of holes), to which to associate a label.
- **CVAT** [12]: is the Computer Vision Annotation Tool. It is a web-based and collaborative image and video annotation tool for labeling data for computer vision tasks. It supports many types of annotation: bounding-boxes, polygons, polylines,

points, etc., and offers additional annotation forms such as 3D cuboids, skeletons, and point clouds. Moreover, it offers features such as interpolation of shapes between keyframes, in video annotation, and automation instruments to ease and speed up the annotation process, such as automatic detection of object outlines. It also supports models for semi-automatic annotation, offering some pre-installed models: Attributed face detection, RetinaNet R101, Text detection, YOLO v3, and YOLO v7.

- **V7** [79]: is a powerful and complete AI training data platform. It allows the annotation and processing of images, videos, documents, medical imaging files, and 3D volumetric data. The tool can perform various types of annotations for tasks, including object detection, semantic segmentation, and image classification. It offers multiple annotation instruments such as bounding-boxes, polygons, lines, keypoint, ellipses, skeletons, cuboids, and additional annotation aids like shapes auto-selection. V7 supports the training of Label-Assistant Models and their execution, the execution of a set of public models of different tasks, and the user's models through REST API. This allows for auto-annotation features, that make V7 a valuable tool to quickly produce high-quality annotated data. In the end, it includes more functionalities such as collaboration features, annotators' performance tracking, and annotation workflow management. V7 is freely accessible with reduced functionalities and requires payment for full access.
- **Labelbox** [46]: is a data-centric AI platform comparable to V7. It addresses both computer vision and LLM (Large Language Models) systems. It allows annotation and data curation of images, videos, audio, documents, text, medical images (DICOM), and geospatial tiles (properly prepared according to Labelbox documentation). As V7 it offers several annotation instruments, and automatic aids to speed up the dataset annotation. Similarly, it integrates the execution of models, evaluation of models through some performance indices, and workflow management together with collaboration functionalities. Labelbox provides a free plan with limitations, and payment plans to fully exploit its features.
- **VIA** [16]: is the VGG Image Annotator from the Visual Geometry Group. It allows to define and describe regions in images or video frames, and temporal segments in audio or video. In particular for image annotation, the tool lets to manually define objects, through rectangles, circles, ellipses, polygons, points, and polylines, and attach a textual description or a label, through configurable input types (e.g., text input, checkbox, radio, dropdown, etc). Moreover, it provides the functionality of uploading annotations (e.g., automatically generated offline with a model), that can

be then filtered, selected, and updated.

- **Make Sense** [67]: is an online tool for image labeling. It supports both classification, through labels, and detection, through polygons, bounding-boxes, lines, and points. It supports the export of annotations to different formats, depending on the type, and the import of rectangular and polygonal annotations. Furthermore, the tool integrates three ML models, that run locally thanks to TensorFlow.js: YOLOv5 and COCO for object detection, PoseNet for pose estimation with points.
- **LabelStudio** [70]: is an open-source data labeling tool, for audio, text, images, videos, and time-series. In the context of computer vision, it offers annotation for image classification, object detection, and semantic segmentation tasks. For each annotation task, a labeling setup is provided, but the user can create customized ones using a specifically designed configuration language. The tool allows the import of annotations or predictions, export to multiple formats, both from/to files or cloud storage, and integration with machine learning models, through the specific Label Studio ML backend SDK. The tool must be installed locally or on a server and accessed through a web application. It supports also collaborative annotation, provided that the users have access to the same Label Studio instance.
- **TORAS** [38]: is the Toronto Annotation Suite, originated as an advancement of the previous Polygon-RNN++ tool [1]. It offers interactive segmentation tools that integrate segmenting models, able to automatically detect an object outline, with some precise polygon editing methods. It allows the creation of training datasets, also with collaboration options, and the export of annotations to TORAS, COCO, and mask formats.

In Table 2.2 a summary of the previously described tools is presented, together with their publishing year, the available instruments for the annotation and annotation exporting formats, and whether they provide collaborative functionalities. There are other tools available that can also be mentioned. Among open-source tools there are LabelImg [77], ImgLab [28], and COCO Annotator [7] that provide a lightweight solution with a simple interface for image annotation, offering some annotation instruments for image classification, object detection and segmentation. Tools such as RoboFlow [17], Superannotate, and Kili offer more complex platforms, and are proprietary, with full access based on fees. They provide management of datasets, multiple data types such as images, text, and video, extended labeling features, enhanced by automatic tools like shape-detection and others, collaboration features, and some integrations with models.

Tool	Ref.	Year	Image Annotation	Format	Collab.
PhotoStuff	[29]	2005	bounding-box	RDF/XML	No
LabelMe	[65]	2008	bounding-box, polygon, mask	XML	only for LabelMe dataset
VoTT	[54]	2017	bounding-box, polygon	YOLO, CNTK, Pascal VOC, JSON, CSV	No
ImageTagger	[23]	2018	bounding-box, line, polygon, point	user definable	Yes
ByLabel	[59]	2018	mask	TXT	No
CVAT	[12]	2018	bounding-box, point polygon, polyline mask, skeleton 3D cuboid, point cloud	YOLO, COCO, Pascal VOC, CVAT, others	Yes
V7	[79]	2018	bounding-box, lines polygon, ellipse keypoint, cuboid skeletons, mask	COCO, CVAT DARWIN (XML) YOLO, PNG PASCAL VOC	Yes
Labelbox	[46]	2018	bounding-box, mask, cuboid, polygon, polyline, point	NDJSON	Yes
VIA	[16]	2019	bounding-box, point polygon, polyline, ellipse, circle	JSON, CSV	Yes
Make Sense	[67]	2019	bounding-box polygon point, line	YOLO, CSV, Pascal VOC XML, JSON	No
Label Studio	[70]	2020	polygon, mask, bounding-box, keypoint	COCO, CSV, JSON, YOLO, Pascal VOC XML	Yes
TORAS	[38]	2021	bounding-box, polygon, mask	TORAS, COCO, PNG masks	Yes

Table 2.2: Image Annotation tools

3 | Proposed Solution

3.1. ODIN

ODIN is an evaluation framework developed in [75, 76, 83]. The framework provides functionalities for black-box diagnosis for image classification, object detection, and instance segmentation. It allows the investigation of errors and model performance through a wide range of metrics and diagnostic reports, including the analysis of subsets of the input by exploiting meta-annotations. It also provides a Graphical User Interface (GUI) for annotating the dataset and adding custom extra properties to the input. The tool is implemented in Python and the GUI is realized as a Jupyter Notebook.

3.2. ODIN Web

ODIN Web is a web application, developed in [52], whose aim is to provide a web interface, accessible by non-technical users, that exploits the above-mentioned ODIN framework. The application allows to create and manage datasets, annotate images according to the different computer vision tasks, such as image classification, object detection, and instance segmentation, and investigate model performance. The purpose of this work is to extend ODIN Web with further functionalities to address the requirements presented in this section. The aim is to realize a web tool covering the most steps in the model development pipeline, from data annotation to the application of computer vision models. In particular, the tool should be collaborative and offer dataset management features, annotation for image classification, detection, and segmentation tasks, model performance analysis, inference, and geo-visualization of predictions for geo-localized stellite images.

ODIN Web supports the annotation of five computer vision tasks divided into classification and localization tasks. Classification tasks are those concerning the identification of one or more categories in the image. ODIN Web supports the following classification tasks:

- *Binary Classification*: the task of categorizing an image into one of two defined categories;

- *Multi-Class Single-Label Classification*: the task of classifying an image with only one class from a defined set of more than two categories;
- *Multi-Class Multi-Label Classification*: the task of assigning to an image one or more classes from a defined set of more than two categories.

Localization tasks are those that refer to the problem of identifying which objects are present in an image and localizing them on the image. ODIN Web supports the following localization tasks:

- *Object Detection*: the task of recognizing the objects in the image and defining their location enclosing each one in a bounding box.
- *Instance Segmentation*: the task of recognizing the objects in the image and defining their location by outlining them.

3.2.1. Requirements

The requirements identified for the design of ODIN Web are represented through the use cases diagram in Figure 3.1. Then, some relevant use cases are described, in particular, those related to the user management functionalities, the map and predictions visualization features, and the models' inference features.
For each use case are defined: the goal, the actors involved in the use case, the entry conditions that must be true before the use case starts, the flow of the events, and optionally the exceptions that describe anything leading to not achieving the use case's goal and how is handled.

The actors of the presented use cases are the *Organization Admins*, who are registered by the ODIN Web team upon request, and the *Users*, who, instead, are registered and managed by the admins themselves.

User Management Use Cases

Create a User
An ODIN Web organization admin wants to add a new user to his organization.

- **Actors**: Organization Admin.
- **Entry Conditions**: The admin must be logged in to ODIN Web.
- **Event Flow**: The admin is on the homepage, which corresponds to the *Datasets* page. The admin presses the *Users* button in the navigation bar entering the user

management page. The admin fills out a form, inserting a username and a password for the user he wants to create. Alternatively, the password can be automatically generated by pressing the relative button. Clicking on the *Create* button the user will be registered by the application and a success alert will appear with a recap of the inserted information. The username is inserted into the list of supervised users visible on the same page.

- **Exceptions**: The application recognizes that the username has already been used. A warning is displayed asking the admin to try with another username.

Authorize Users
An ODIN Web organization admin wants to authorize one or more users to a dataset.

- **Actors**: Organization Admin.
- **Entry Conditions**: The admin must be logged in to ODIN Web.
- **Event Flow**: The admin is on the homepage, which corresponds to the *Datasets* page. The admin presses the *Users* button in the navigation bar entering the user management page. The admin selects the target dataset from the dropdown menu, and the application displays two lists of users, respectively the not-authorized ones and the already-authorized ones. The admin selects one or more users from the list of *Not-Authorized* and presses the button to give them permission. A dialog of confirmation is displayed. If the admin confirms the action, the users are enabled to access the target dataset and moved to the *Authorized* list.

Revoke Authorization from Users
An ODIN Web organization admin wants to revoke the authorization to a dataset from one or more users.

- **Actors**: Organization Admin.
- **Entry Conditions**: The admin must be logged in to ODIN Web.
- **Event Flow**: The admin is on the homepage, which corresponds to the *Datasets* page. The admin presses the *Users* button in the navigation bar entering the user management page. The admin selects the target dataset from the dropdown menu, and the application displays two lists of users, respectively the not-authorized ones and the already-authorized ones. The admin selects one or more users from the list of *Authorized* and presses the button to revoke their permission. A dialog of confirmation is displayed. If the admin confirms the action, the users are revoked

their permission to access the target dataset and moved to the *Not-Authorized* list.

Map Visualization Use Cases

Visualize Predictions on the Map

An ODIN Web user wants to visualize the predictions over a geolocalized target dataset onto the map.

- **Actors**: Organization Admin, User.
- **Entry Conditions**: The user must be logged in to ODIN Web. The user must have uploaded the required prediction files in GeoJSON format.
- **Event Flow**: The user is on the *Datasets* page and selects the target dataset entering its page. The user enters the *Map Visualizer* page by pressing the corresponding button. After loading, the map is shown, with Google Maps as default. Onto the map are drawn the perimeter of the area of interest, the predictions in the form of color-coded bounding-boxes, and the activation maps as the contours of the areas the model focused on. The user can manage the visibility of the layers and use an interactive legend to filter the visible predictions.

Open Images from Map Visualizer

An ODIN Web user wants to open a geolocalized satellite image from its corresponding bounding-box on the map.

- **Actors**: Organization Admin, User.
- **Entry Conditions**: The user must be logged in to ODIN Web. The user must have uploaded the required files in GeoJSON format.
- **Event Flow**: The user is on the *Map Visualizer* page of the target dataset. The map is shown together with the perimeter of the area of interest, the color-coded bounding-boxes, and the activation map shapes. By hovering over each bounding-box, a tooltip is opened displaying the name of the corresponding image in the dataset, the coordinates of the center of the square, and the information relative to the annotations associated with the image (categories and properties). With a right-click on the target square, a context menu is opened offering the possibility to open the image in the *Annotator*. Once the user selects the option, a new tab is opened showing the *Annotator* page of the image. The annotations for categories and properties are displayed. The user enables the visibility of the CAMs through the relative layer control menu, and the shapes representing the areas the model

focused on are drawn over the image.

Prediction Visualization Use Cases

Visualize Predictions in the Annotator

An ODIN Web user wants to visualize the predictions over a dataset onto each single image.

- **Actors**: Organization Admin, User.
- **Entry Conditions**: The user must be logged in to ODIN Web. The user must have run the inference of a model on the dataset or uploaded the required prediction files.
- **Event Flow**: The user is on the *Datasets* page and selects the target dataset entering its page. The user enters the *Annotator* page by pressing the corresponding button. A grid of thumb images is displayed, one for each image in the dataset, spread over multiple pages. The user navigates through the grid and presses on a target image entering its annotation page. On this page, the image is visualized and the annotation section for categories and properties is displayed. The user enables the toggle switch *Show Predictions*, and the predictions over the image are shown in the appropriate section. A color-coded border appears around the image as a visual indication of the prediction. If CAMs are available, a layer control menu also appears. The user selects the *CAMs* option from the menu and the activation areas are shown over the image.

Open Map Visualizer from Annotator

An ODIN Web user wants to visualize a specific geolocalized image on the map.

- **Actors**: Organization Admin, User.
- **Entry Conditions**: The user must be logged in to ODIN Web. The user must have uploaded the required files in GeoJSON format.
- **Event Flow**: The user is on the *Annotator* page, where the grid of thumb images is displayed. The user navigates through the grid and presses on a target image entering its annotation page. On this page, the image is visualized and the annotation section for categories and properties is displayed, if available the user can visualize the predictions and CAMs as described in the previous use case. The user presses on the *Open in Map Visualizer* button entering the *Map Visualizer* page. The map is loaded, zoomed, and focused automatically on the position on the map corresponding to the geolocalized image. The user interacts with the map, and the

information represented on it, as described in the Map Visualization Use Cases.

Model Inference Use Cases

Model Selection

An ODIN Web organization admin wants to select or add a model for a dataset.

- **Actors**: Organization Admin.
- **Entry Conditions**: The admin must be logged in to ODIN Web.
- **Event Flow**: The admin is on the *Datasets* page and selects the target dataset entering its page. The admin presses the *Settings* button and accesses the dataset *Settings* page. The admin is presented with a list of the editable settings of the dataset, among which there is a *Models* section. The admin selects the desired model, through its corresponding checkbox, among the ones made available by ODIN Web. If the admin wants to insert a model of his own, he inserts the model name into the dedicated input field, which checks the name availability, and presses the corresponding button to add it. The system will create directories, one for each model, in which the predictions are stored (by inference with the available models) or uploaded (for the additional models).

Model Inference on a Dataset

An ODIN Web user wants to run the inference over a dataset.

- **Actors**: Organization Admin, User.
- **Entry Conditions**: The user must be logged in to ODIN Web.
- **Event Flow**: The user is on the page of the target dataset and enters the *Predictor* page by pressing the corresponding button. The user navigates to the *Models* section and selects the model to run from the dedicated dropdown menu. The user runs the model through the *Run* button. A progress bar appears, with the percentage of completion and an estimate of the remaining time, while the system elaborates the images. During the inference, the user can navigate through the rest of the application. Once the inference is completed the system saves the predictions in TXT format in the directories mentioned in the previous use case. The computed predictions are used in the *Analyzer*, together with the annotated ground-truth, for the analysis of model performance, and in the *Annotator* as described in the Prediction Visualization Use Cases.

Model Inference on Images

An ODIN Web user wants to run the inference over some new images distinct from the datasets ones.

- **Actors**: Organization Admin, User.
- **Entry Conditions**: The user must be logged in to ODIN Web.
- **Event Flow**: The user is on the homepage, i.e., the *Datasets* page. The user navigates to the *Models* section, in which he selects one of the available models. The user uploads some images, that will remain separated from the other image collections. The user runs the selected model on the added images. Similarly to the previous case, during system elaboration, a progress bar is displayed. Once the inference is completed, the images appear on a preview grid. The user selects an image entering the page for the visualization of the prediction, including the complete Class Activation Map (CAM) on the image.

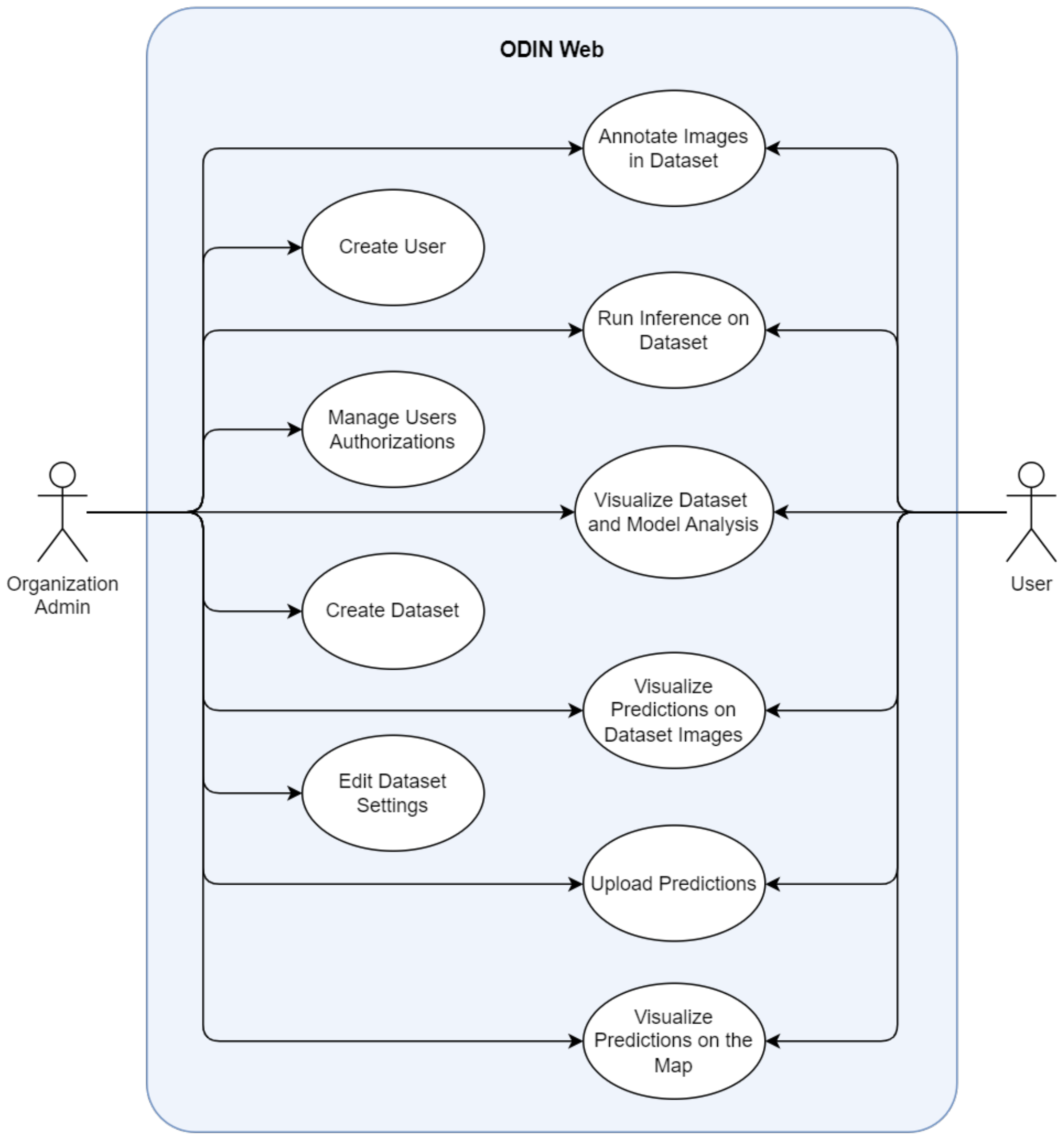

ODIN Web
Organization Admin
User
Annotate Images in Dataset
Create User
Run Inference on Dataset
Manage Users Authorizations
Visualize Dataset and Model Analysis
Create Dataset
Visualize Predictions on Dataset Images
Edit Dataset Settings
Upload Predictions
Visualize Predictions on the Map

Figure 3.1: Use Cases Diagram

4 | Implementation

This chapter aims to describe the design and implementation of ODIN Web. An overview of the employed architecture is described in Section 4.1, the design is discussed in Section 4.2, a description of the implementation technologies is presented in Section 4.3, lastly the application deployment is addressed in Section 4.4.

4.1. Architecture Overview

The architecture adopted for the implementation of ODIN Web is that of a three-tier client-server architecture, that consists of a presentation, an application, and a data tier. The presentation tier is the interface through which the user interacts with the application. The application tier is the component handling the logic of the application. The data tier is the place that stores the information processed by the application. In Figure 4.1, a high-level overview of ODIN Web architecture is represented. ODIN Web is accessible for the users from the client side through a *Web Browser* of choice which communicates with the *Application Server* through HTTP protocol. On the *Application Server* resides the ODIN Web application logic which interacts with the ODIN framework to exploit the services that it offers. The *Application Server* communicates with the *Database Server* which runs an instance of the DBMS, MongoDB, used by the ODIN Web application to store, access, and update data.

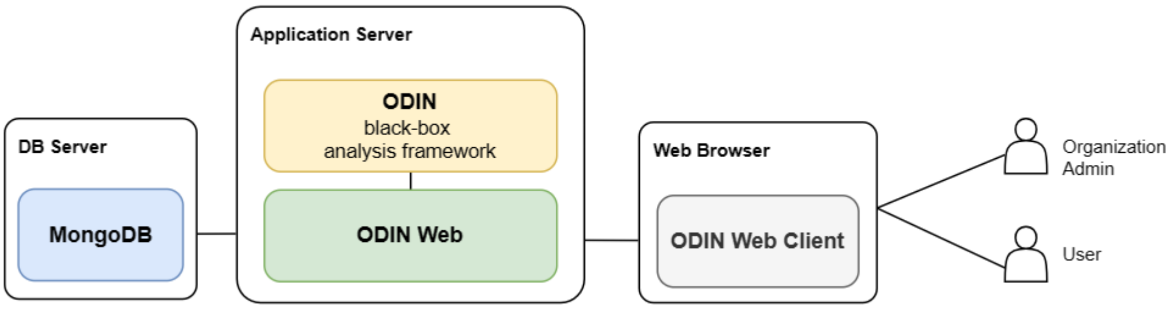


Figure 4.1: ODIN Web Architecture Overview

4.2. Design

In this section are presented and described the design choices at the basis of ODIN Web development. In particular, in the following subsections, the Domain Model and the ODIN Web Backend and Frontend designs are introduced.

4.2.1. Domain Model

Considering the purposes of ODIN Web and the requirements described in Chapter 3 the conceptual design of the domain has been defined. The resulting domain model, reported in Figure 4.2, illustrates the concepts, and their relationships, involved in the design of the ODIN Web application, representing them as classes in the diagram. The diagram is built following the Unified Modeling Language[1] (UML) conventions.

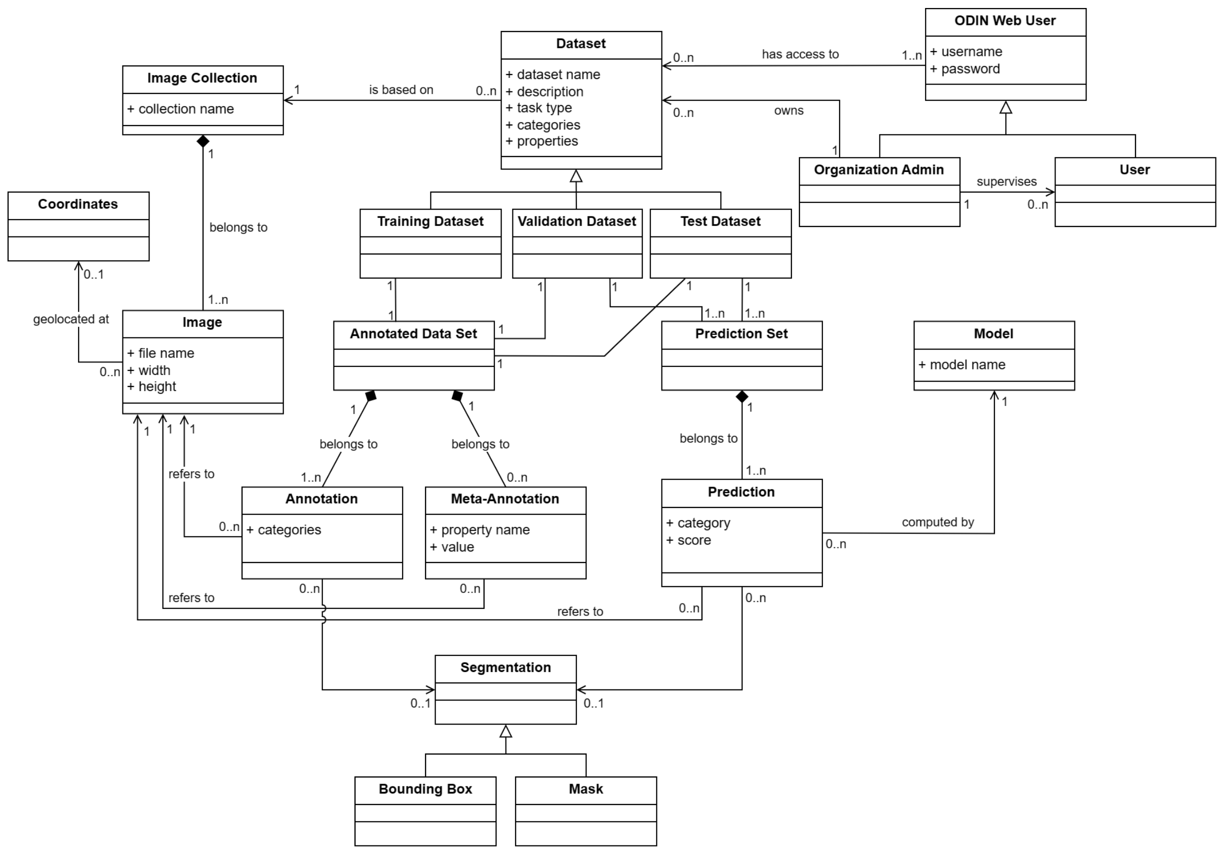


Figure 4.2: ODIN Web Domain Model

The concepts and relationships represented in the domain model diagram of ODIN Web are described in the following.

[1]`https://www.uml.org/`

Odin Web User The *Odin Web User* class represents any end user of the application. A user can be of one of two types:

- **Organization Admin**, who has full rights to create and delete datasets, edit dataset settings, register users, and manage the authorizations for dataset access.
- **User**, who has limited rights, in particular, can visualize and access only the datasets he has been authorized to by the *Organization Admin* and does not have access to any user-management functionalities.

Image The *Image* class represents an image with its *file name*, *width*, and *height*. If the image is geo-localized, it will also have a set of *Coordinates*.

Image Collection The *Image Collection* class represents a named set of *Images*.

Dataset The *Dataset* class represents an object that is associated with an *Image Collection*, the set of images to be considered for the dataset annotation, analysis, and inference, and that is characterized by a name and a description and specific values for the user-defined configuration parameters: *task type*, *categories*, and *properties*. A *Dataset* can also contain the *Annotation Data Set* derived from the annotation process and some *Prediction Sets* derived from the inference on the dataset images. The *Dataset* class generalizes the three types of dataset *Training Dataset*, *Validation Dataset*, and *Testing Dataset*.

Annotation The *Annotation* class represents an annotation made over an *Image*. As defined in 2.3, an annotation is the information associated with the image to be used in model training as the truth. The *Annotation* concept includes the different forms in which an annotation can be represented based on the computer vision task to address and may include a *Segmentation* for the localization ones.

Segmentation The *Segmentation* class represents a set of image-related coordinates combined or not with image dimensions that encode a shape drawn to delimit a certain area of an image. In particular, this class generalizes two distinct types of segmentations that are *Bounding Boxes* and *Masks*.

Meta-Annotation The *Meta-Annotation* class represents a property value associated with an *Image*. As defined in 2.3, a meta-annotation is the information associated with the image that is not used for training, but that can be functional in the dataset or model performance analysis.

Annotation Data Set The *Annotation Data Set* class represents the set of *Annotations* and *Meta-Annotations* associated with the set of images of the relative *Dataset*. Of the *Annotation Data Set* only the annotations are to be considered in the training of a model. The meta-annotations instead can be used for a deeper and more informative model performance analysis.

Model The *Model* class represents a Machine Learning/Deep Learning model.

Prediction The *Prediction* class represents a prediction obtained from *Model* inference over an image. A *Prediction* has a score, which represents the prediction confidence over a category, and in case of detection or segmentation tasks, also a *Segmentation* that is either a *Bounding Box* or a *Mask*.

Prediction Set The *Prediction Set* class represents a set of predictions computed by a model over the set of images of the relative *Dataset*. It can be used to show the predictions over the images and to perform model performance analysis and model comparison.

4.2.2. Backend

This section describes the design of the ODIN Web backend, that handles the logic of the application. ODIN Web backend is built into modules to separate the various functionalities of ODIN Web, such as user management, dataset management, annotation, analysis, model execution, and map visualization. Figure 4.3 illustrates the class diagram that represents the main components of ODIN Web backend extending the model in [52].

User

The User component concerns users and user management features. There are two types of users: the admin and the simple user. Admins are registered upon request by the ODIN Web team, while users are created by the admins, who register them by inserting:

- *username*;
- *password*.

As shown in Figure 4.4, the User component offers a series of methods to deal with user management. This component provides login methods, both through *username* and *password* or through JSON Web Tokens (JWT) [37]. A JWT temporary token is generated upon login with username and password and saved at the frontend in the Cookies. In

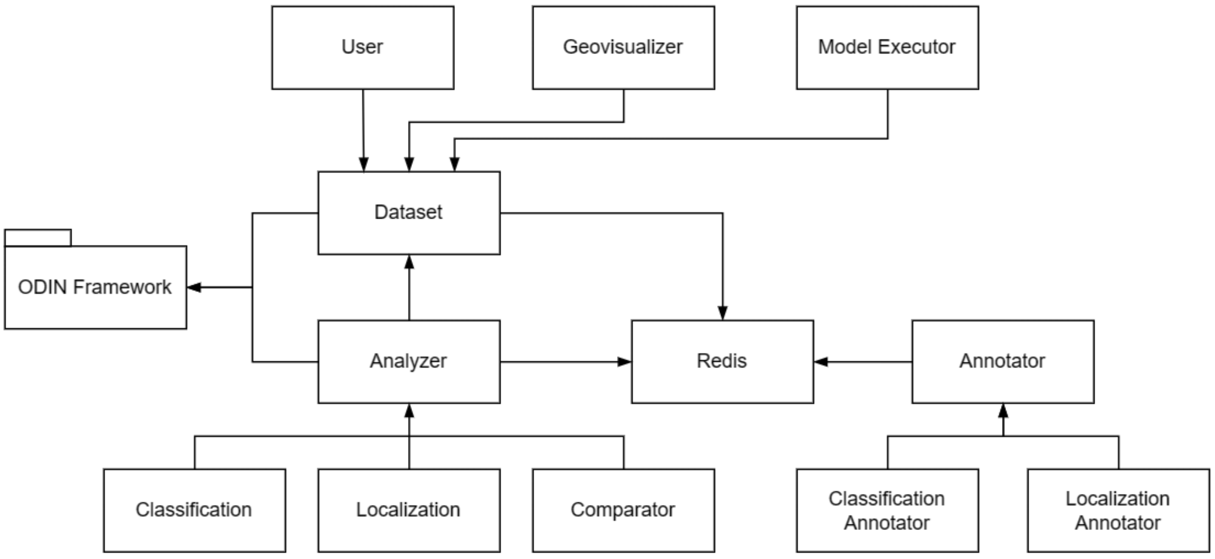


Figure 4.3: UML Class Diagram of the main components of ODIN Web Backend.

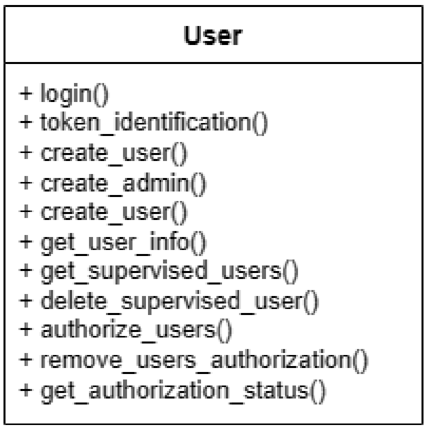
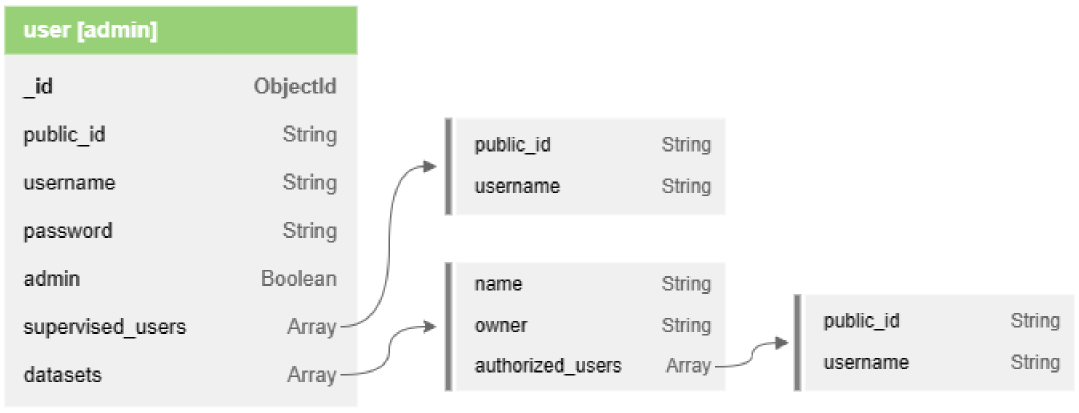


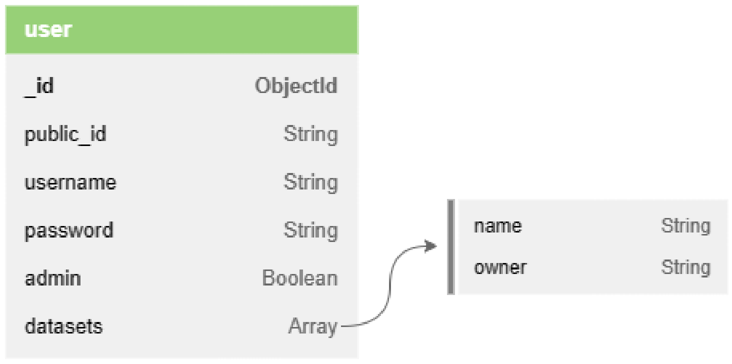
Figure 4.4: UML Class Diagram of User Component

this way, it can be later employed for automatic authentication on ODIN Web, if it has not expired. For authentication and authorization purposes the functions exposed by ODIN Web are decorated with `@token_required`, which checks whether the user is authenticated, or in specific cases with `@admin_only` if the function is reserved to the admins. Other methods provided by this component are for user creation and management by admins, and for granting or revoking authorizations.
Figure 4.5 illustrates how the user instances are structured inside the `users` collection in the chosen database MongoDB (4.3). For clarity, the admin and the user structures are represented separately in the figure. Users information is organized in the fields:

- `_id`: the primary key generated by MongoDB;
- `public_id`: a unique identifier generated at creation;



(a) Admin



(b) User

Figure 4.5: MongoDB structure of user: user with admin privileges (a), simple user (b), the two cases are represented separately for clarity.

- `username`: the username associated with the user;
- `password`: the password stored in hashed form;
- `admin`: a flag that indicates whether the user is an admin or not;
- `datasets`: an array of datasets. Each dataset is a document with:
  - `name`: name of the dataset;
  - `owner`: the dataset owner `public_id`.

  Only if the user is an admin, the dataset document contains also the field:
  - `authorized_users`: an array containing the users authorized by the admin to access the dataset, each with its `public_id` and `username`.

Only in the case, that the user is an admin, its document contains also the field:

- `supervised_users`: an array containing the users created by the admin, each with its `public_id` and `username`.

Dataset

The Dataset component concerns the management of datasets. As defined in Section 4.2.1 a dataset is a composite object that contains multiple pieces of information. A dataset is created by an admin that specifies:

- *dataset name*: the name of the dataset. Different datasets cannot be named equally.
- *dataset description*: a text that describes the dataset.
- *task type*: the type of computer vision task to address.
- *categories*: the list of classes to be considered for annotating the dataset images. In case the task is *Binary Classification*, then the classes are two and only two.
- *properties*: the meta-annotation properties that can be used during annotation to add extra information. ODIN Web allows to define properties in three formats:
  - *unique*: the property has a fixed set of user-defined values;
  - *range*: the property can assume integer values in a user-defined range;
  - *textual*: the property value is an arbitrary text.
- *images path*: the path to the collection of images the dataset is based on;
- *models*: the models selected or associated with the dataset from the dataset settings.

On the backend each admin has a dedicated directory for storing the content of the datasets. Each dataset has its directory in which is stored: the *configuration* file, which contains all the dataset settings parameters, the *ground truth* file, which can be uploaded or downloaded, the *properties* file, the directories to store model *predictions*, and those to store the *geolocalized prediction* files.

Annotator

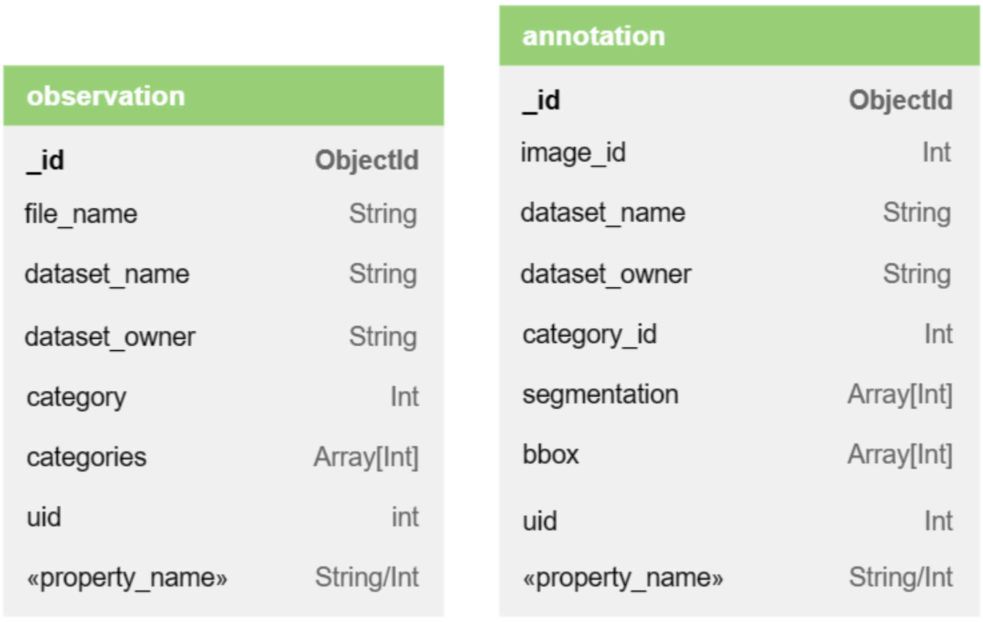
The Annotator component is the one used to manage everything related to the annotation process. The component is divided into three modules:

- *Classification Annotator*: handles annotations specifically for the classification tasks, i.e., binary, multi-class single-label, and multi-class multi-label classification.
- *Localization Annotator*: handles annotations specifically for the localization tasks, i.e., object detection and instance segmentation.

- *Annotator*: handles the management of the ground truth for both classification and localization tasks.

The annotations are saved on the MongoDB database. Figure 4.6 illustrates the structure of *observations*, organized to store annotations for classification tasks and part of the `observations` collection, and the structure of *annotations*, organized to store annotations for localization tasks and part of the `annotations` collection. With respect to the previous structure described in [52], the new schema, includes one additional field for both observations and annotations:

- *dataset_owner*: is the `public_id` of the owner of the dataset associated with the annotation or observation, which is needed so that annotations and observations can be univocally identified.



observation	
_id	**ObjectId**
file_name	String
dataset_name	String
dataset_owner	String
category	Int
categories	Array[Int]
uid	int
«property_name»	String/Int

annotation	
_id	**ObjectId**
image_id	Int
dataset_name	String
dataset_owner	String
category_id	Int
segmentation	Array[Int]
bbox	Array[Int]
uid	Int
«property_name»	String/Int

(a) Observation (b) Annotation

Figure 4.6: MongoDB structure of annotations: observation, represents a classification annotation (a), annotation, represents a localization annotation (b). Updated [52].

Analyzer

The Analyzer component is the one used to manage everything related to the analysis process. The component is divided into four modules:

- *Classification* Analyzer: handles the analysis requests specifically for the classification tasks, i.e., binary, multi-class single-label, and multi-class multi-label classification.

- *Localization* Analyzer: handles the analysis requests specifically for the localization tasks, i.e., object detection and instance segmentation.
- *Comparator*: handles the analysis requests concerning the comparison between multiple models.
- *Analyzer*: is the main analyzer module that instantiates and saves the analyzer class, which is then used to exploit the services offered by the ODIN Framework. The analyzer object has the following settings:
  - *properties*: the set of properties among the ones defined for the dataset to be considered in the analysis;
  - *similar-classes*: groups of categories to be considered similar;
  - *threshold*: the confidence threshold to apply, for each model;
  - *iou-threshold*: the IOU threshold for localization tasks, for each model;
  - *categories-factor-check* and *categories-factor*: whether category normalization is to be applied, and the relative factor, for each model;
  - *properties-factor-check* and *properties-factor*: whether properties normalization is to be applied, and the relative factor, for each model;

Model Executor

The Model Executor component is the one used to manage model inference. It exposes two methods:

- `run_model()`: is the function that processes the HTTP request coming from the frontend and extracts the model inference parameters: the *name* and *owner* of the dataset, the *model* selected for the inference, and the *type of task*.
- `task_progress()`: is the function called to retrieve the progress of the inference.

When the frontend calls `run_model()`, this function executes extracting the data, calling the `execute_model()` task, and returning its `id`. The `execute_model()` is a `@shared_task` that runs in the background on the `celery worker`, performing the inference through calls to the prediction method of the target model. While this task is executed the rest of the application remains available. The frontend periodically retrieves the status of an inference task by calling `task_progress()`, providing as argument the task `id`.

Geovisualizer

The Geovisualizer component handles the information regarding the Map Visualization functionalities. In particular, it exposes methods to:

- retrieve the information needed to populate the Map Visualizer;
- upload the geolocalization information, such as the area `shape` or `kml` files, or the `geojson` prediction files.

Redis

The Redis component is used to manage the data stored on the Redis server (4.3). The component provides the other modules functions to *store/get/change* dataset and analyzer instances, keep track of the model under analysis when using the comparator, and maintain the user's session in the annotator. Data is stored in a key-value structure, where the key includes the user *uuid*, i.e., a unique temporary id that changes over different sessions.

4.2.3. Frontend

This section introduces the design of the ODIN Web frontend. The user application interface comprises different pages, that are built following a component-based approach. Indeed, components allow to split the user-interface into independent and reusable pieces, which can then be assembled and nested to realize more complex components. Figure 4.7 illustrates the tree of nested components of the ODIN Web frontend. All components are nested under the root instance, i.e., the App component. Directly inside the root, there are the components representing the different views of ODIN Web, that themselves encapsulate other components, until the basic ones.

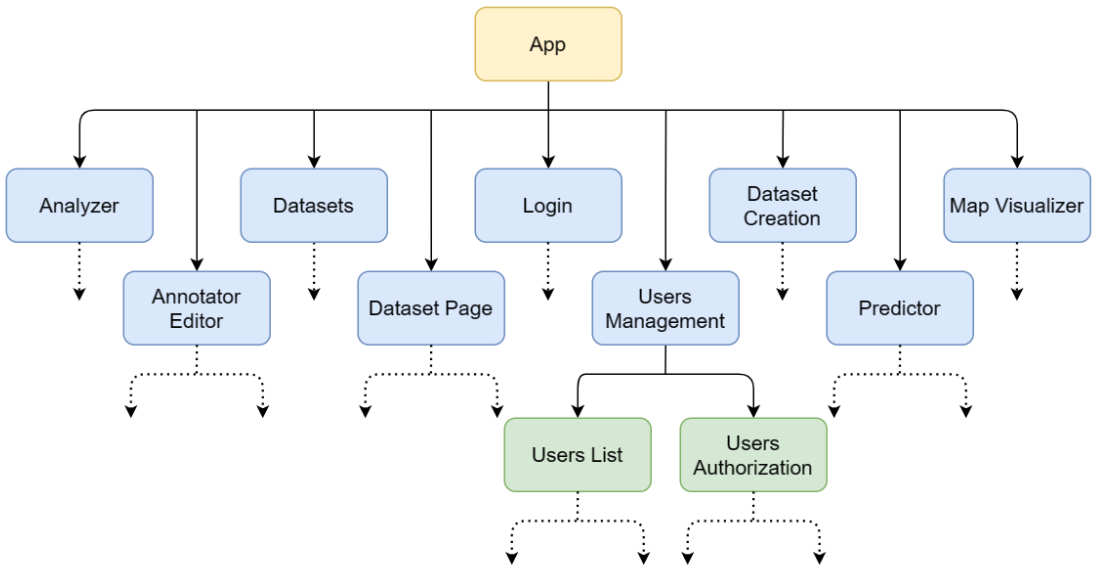


Figure 4.7: ODIN Web Frontend Components

The entire ODIN Web user-interface is illustrated and described in Chapter 5.

4.3. Implementation Technologies

In this section, the technologies adopted for the ODIN Web implementation are discussed. In particular, are presented the ones employed for backend, frontend, database, communication, and deployment. The described technologies follow the ones used in [52] for the first implementation of ODIN Web, and are extended with others necessary to implement the new version presented in this work. As well as ODIN [75, 76, 83], the backend of ODIN Web is written in Python, while the frontend uses Javascript, HTML, and CSS.

Docker

Docker[2] is an open-source platform for developing, deploying, and running applications. It provides the ability to package applications in isolated environments, enabling the separation of the applications from the infrastructure. Specifically, docker provides the following useful features:

- *isolation*: docker allows to package an application and its dependencies into a standardized unit called a container. The container is lightweight and self-contained, and as such does not need to rely on what is installed on the host. Many containers can be run simultaneously on the same host.
- *portability*: the docker containers include everything needed to run the target application and can run consistently on multiple different host environments.
- *fast and consistent delivery*: given the characteristics of containers those can be tested locally, shared, and delivered consistently regardless of the underlying host.

For a description of how dockers are employed for the ODIN Web deployment refer to Section 4.4.

Flask

Flask[3] is a micro web framework implemented in Python and is one of the most popular Python web application frameworks. Flask is lightweight and flexible as it does not depend on particular external libraries, except for Werkzeug, a Python utility library for Web Server Gateway Interface (WSGI) applications, and Jinja, a Python template engine. Flask can be integrated through the many available extension libraries that allow to easily add needed functionalities.

[2]https://www.docker.com/

[3]https://flask.palletsprojects.com/en/3.0.x/

Flask allows the mapping of specific URLs to the associated functions through the `@app.route()` decorator, which requires, as arguments, the URL string and the list of HTTP methods supported by that URL: `@app.route('/url/of/function', methods=['GET'])`. In ODIN Web, Flask is used to build the web application and handle the HTTP communication with the frontend on the backend side.

Celery

Celery[4] is a simple, flexible, and reliable distributed system to process messages, written in Python. It is a powerful task queue for real-time processing and task scheduling. Celery is suitable for simple background tasks, as well as complex multi-stage programs and schedules. It can be used independently, but can also be integrated and configured with Flask to let tasks access the Flask application. Through the `@shared_task()` decorator, a function can be defined as a task to be executed in the background, running separately on a celery worker instead of on the main Flask application. In ODIN Web, Celery is employed to run the inference of models in the background while the rest of the application remains available.

MongoDB

MongoDB[5] is a NoSQL document-based database suitable for scalable applications. It is schemaless and stores data in collections and JSON-like documents consisting of key-value pairs, where the value may be a native data type, an array, or other documents.
ODIN Web adopts MongoDB as the database, given its flexibility in the structure of data, and the documental format for the storage. Indeed, ODIN Web mostly works with data in JSON format, e.g. annotations in MS COCO format, that can be easily mapped to the MongoDB database. The structure is not fixed and, as such, is suitable to store instances of the same concept that contain different fields, e.g. different types of users (Figure 4.5). Moreover, MongoDB is scalable and can deal with large volumes of data, which is an important aspect considering the amount of expected annotations. The MongoDB database of ODIN Web consists of four collections: *observations*, for storing image classification annotations, *annotations*, for storing detection and segmentation annotations, *images*, for storing image information, and *users*, for storing users with their relationship and datasets access authorizations.

Redis

Redis[6] is an open-source, in-memory data structure store that can be used as a database,

[4]https://docs.celeryq.dev/en/latest/index.html
[5]https://www.mongodb.com/
[6]https://redis.io/

cache, message broker, and streaming engine. In ODIN Web Redis is used as a caching memory to store instances of Dataset and Analyzer associated with a user in a session. The advantage is that Redis data resides in memory, instead of on disk, achieving top performance with low latency and high throughput.

Vue.js

Vue.js[7] is a Javascript framework for building web user interfaces, that covers most frontend development features. It is based on standard HTML, CSS, and Javascript, and provides a declarative and component-based programming model. The Vue.js framework has two major features:

- *Declarative Rendering*: it extends the standard HTML with a template syntax that allows to describe the HTML output based on the Javascript state. In this way, the manual manipulation of the DOM is reduced to a minimum, and Vue takes care of updating the view when the state changes.
- *Rectivity*: it automatically monitors changes in the Javascript state and updates the DOM to reflect those changes. The update is efficient as Vue maintains a virtual DOM (v-DOM) to compute the difference between the old and new DOM so that only the necessary parts are re-rendered.

In ODIN Web, Vue.js is used to create the whole web application interface. The components are built using the Single-File format feature of Vue.js, which encapsulates the component's logic (Javascript), the template (HTML), and the style (CSS) in a single file for each component. Components are implemented singularly and then nested and assembled to compose web pages.

Axios

Axios[8] is a promise-based HTTP Client for Node.js and the browser. Axios can be used to make XMLHttpRequests from the browser or HTTP requests from node.js. It supports the Promise API, for handling asynchronous calls, the automatic request body serialization to JSON, Multipart or FormData, and URL-encoded form, and the automatic handling of JSON data in response. For these characteristics, Axios has been used for handling HTTP requests and responses in the Vue.js ODIN Web frontend.

[7]https://vuejs.org/
[8]https://axios-http.com/

4.4. Deployment

ODIN Web is deployed, as mentioned in Section 4.1, on a three-tier architecture. Specifically, ODIN Web is accessible online from the user's device through a Web Browser of choice. The *ODIN Web Client* interacts with the server through the HTTP protocol. On the server side, ODIN Web is deployed on the *ODIN Server* with three dockers that contain respectively: the *Application Server* and the *Database Server*, which are networked and communicate with each other, and the *Ortophoto Server*, which exposes its services through HTTP. The deployment infrastructure is illustrated in Figure 4.8 and the content

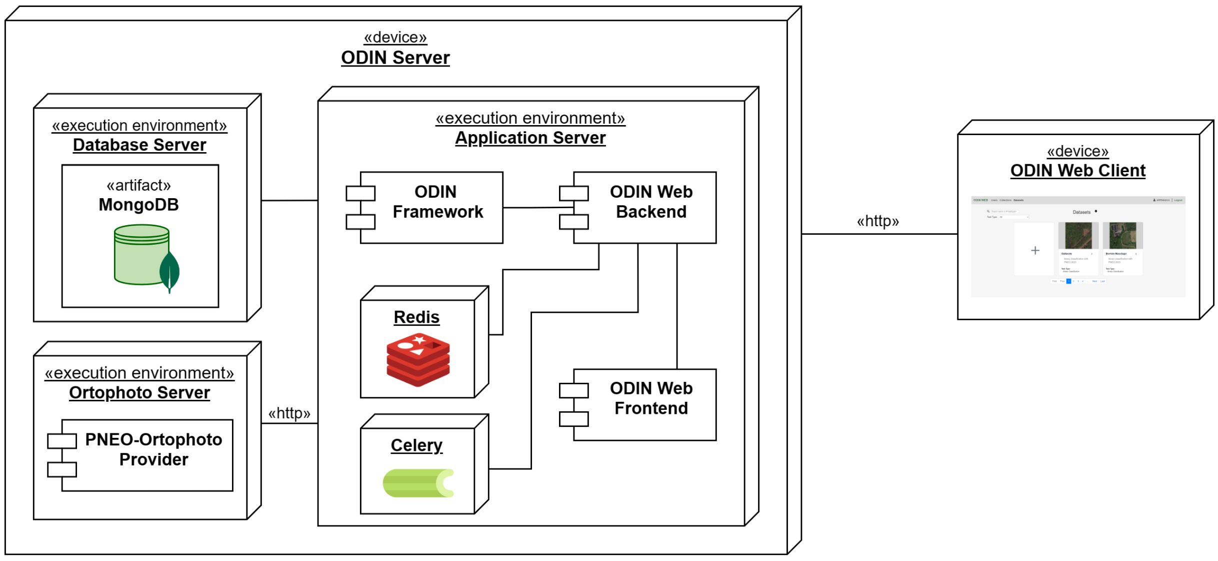


Figure 4.8: ODIN Web Deployment Diagram

of the three dockers is described in the following.

Application Server The first docker, based on Python 3.10.8, represents the *Application Server* that includes all the components handling the application logic.

- *ODIN Framework*: the ODIN framework is installed as a `pip` package and therefore accessible as a Python library.
- *Celery*: runs the `celery worker` that offers parallel computation.
- *Redis*: runs the `redis server` used for caching and session maintaining.
- *ODIN Web Backend*: is the ODIN Web backend module running with the Flask framework. It constitutes the core of the ODIN Web application.
- *ODIN Web Frontend*: is the ODIN Web frontend module constituting the web application user interface.

Database Server The second docker represents the *Database Server* on which the MongoDB database resides. This docker is connected to the first one to allow interaction. Specifically, the ODIN Web Backend accesses the MongoDB through its IP address.

Ortophoto Server The third docker represents the *Ortophoto Server* on which the *Ortophoto Provider* runs providing access to the PNEO and Ortophoto tiles. The ODIN Web Frontend accesses this service through HTTP Requests.

5 | Results and Validation

This chapter presents the implementation results of ODIN Web, by illustrating the application's functionalities, and addresses ODIN Web validation.

5.1. Features

Following the requirements in Chapter 3, ODIN Web was implemented as described in Chapter 4 to realize a collaborative web-based tool offering dataset management features, annotation for classification, detection, and segmentation computer vision tasks, model performance analysis, inference, and geo-visualization of predictions. In this section, ODIN Web is demonstrated by illustrating the areas dedicated to those functionalities:

- User Management;
- Dataset;
- Annotator;
- Analyzer;
- Map Visualizer;
- Predictor.

5.1.1. ODIN Web Access

ODIN Web is available online. The access browser must support Javascript execution. An admin account can be obtained upon request to the ODIN Web team, while users are registered by their admin. After reaching the website the user must log in by inserting his *username* and *password*. ODIN Web provides *automatic authentication* if the user has previously accessed and the authentication token has not expired.
After logging in, ODIN Web displays the homepage, which corresponds to the *Datasets* page. To navigate the sections of ODIN Web the user can use the navigation bar on top (Figure 5.1). After entering a dataset, an additional breadcrumb navigation bar, positioned right under the main one, helps the user understand his location on the website and quickly move back to a preferred page.

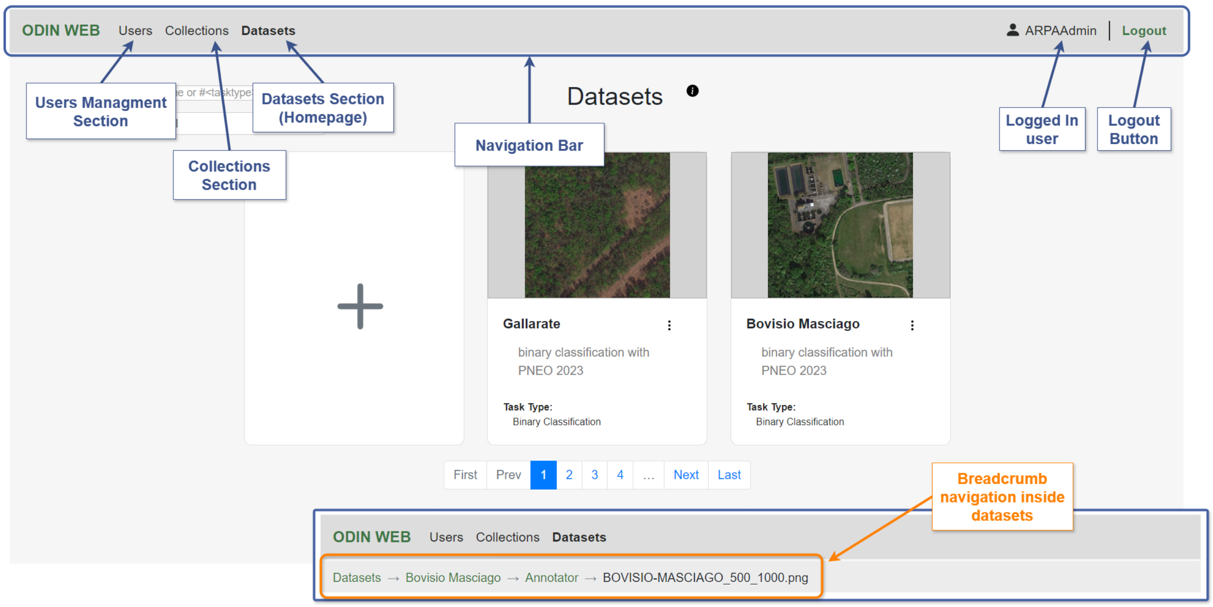


Figure 5.1: Overview of ODIN Web Interface

5.1.2. User Management

The user management feature is restricted to the admins. By entering the *Users* page, the admin is presented with a bipartite interface. On the upper half (Figure 5.2), the admin can **create a user**, by filling out the form on the left, and visualize the list of supervised users on the right. The admin can use the password generation button to generate a random password, which is placed into the relative fields of the form.

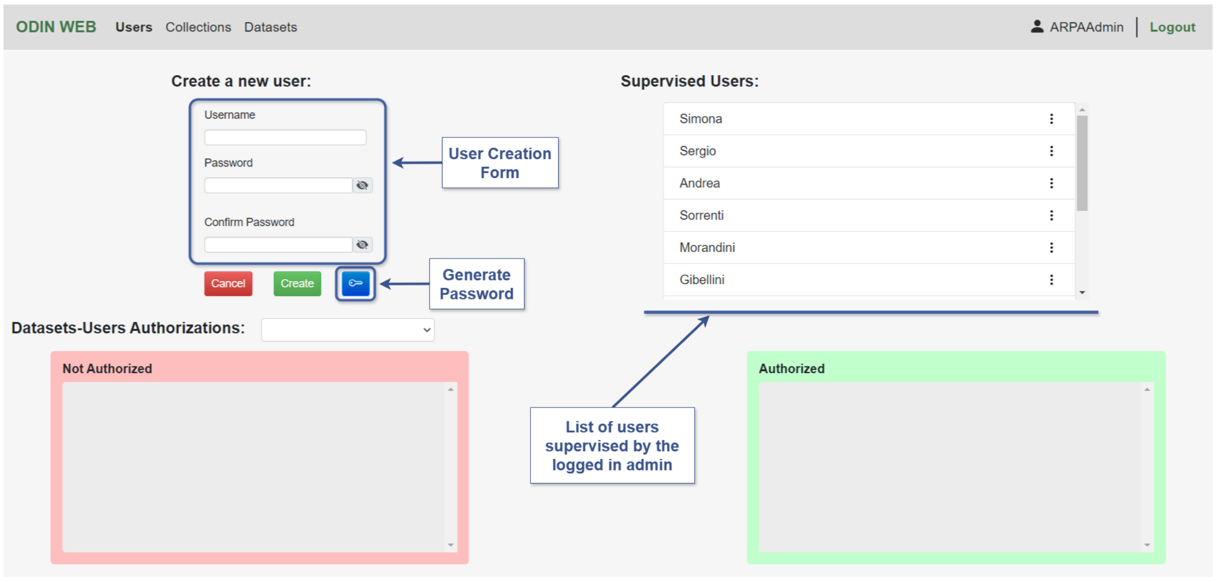


Figure 5.2: User Management: users creation

On the lower half (Figure 5.3), the admin can select the target dataset from the dropdown menu (1). The users authorized and not authorized to that dataset appear in the corresponding lists. The admin can **authorize** some users by selecting them (2a) and then clicking on the button (3a) to move them to the authorized. The admin can **revoke the authorization** from some users by selecting them (2b) and then clicking on the button (3b) to move them back to the not authorized.

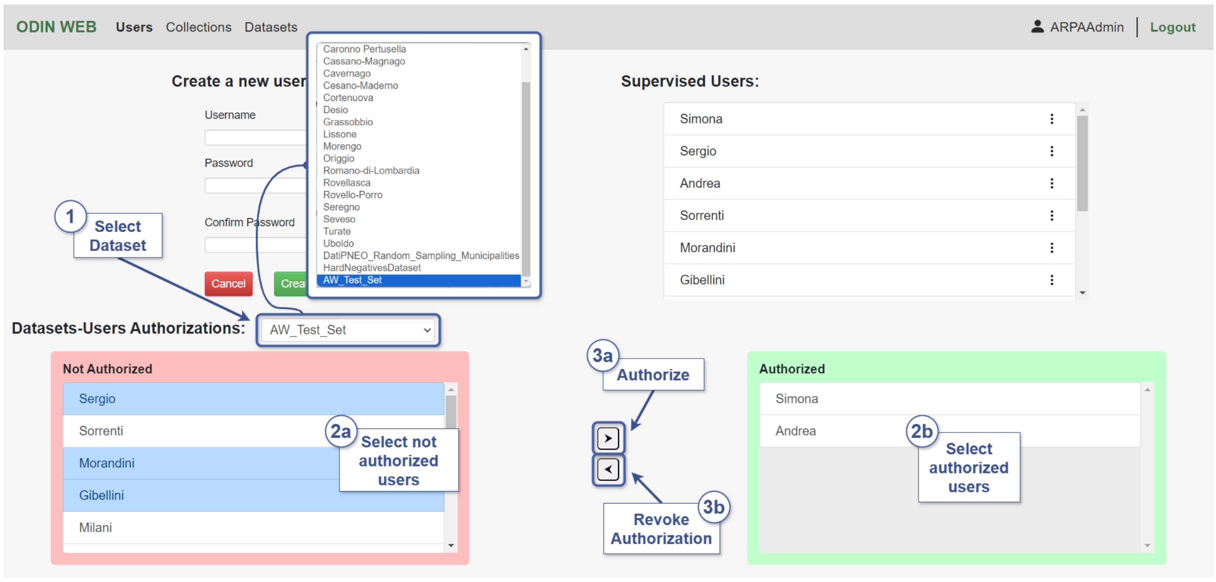


Figure 5.3: User Management: authorize/remove authorization

5.1.3. Dataset

The *Datasets* page, which is also the homepage, presents the users with a horizontal paginated list of datasets. Datasets can be searched by name, or filtered by task type through a dropdown menu. Each dataset is represented through a card reporting the dataset name, description, and task type, and can be accessed by clicking on it. This page offers different functionalities according to the user's role.

The *Datasets* page for admins (Figure 5.4) allows them to create datasets through the *Dataset Creation* page accessible by clicking on the relative button. An admin has access to all the datasets he created.

The *Datasets* page for users allows them to visualize and open the datasets they have been authorized to by their administrator (Figure 5.5).

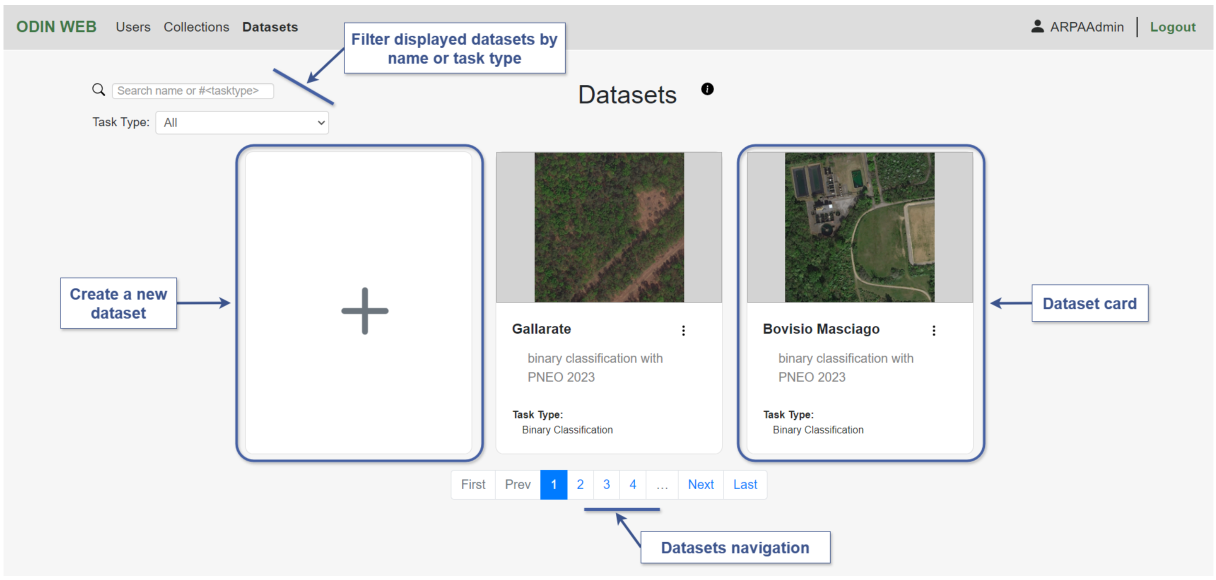


Figure 5.4: Datasets: datasets page for admins

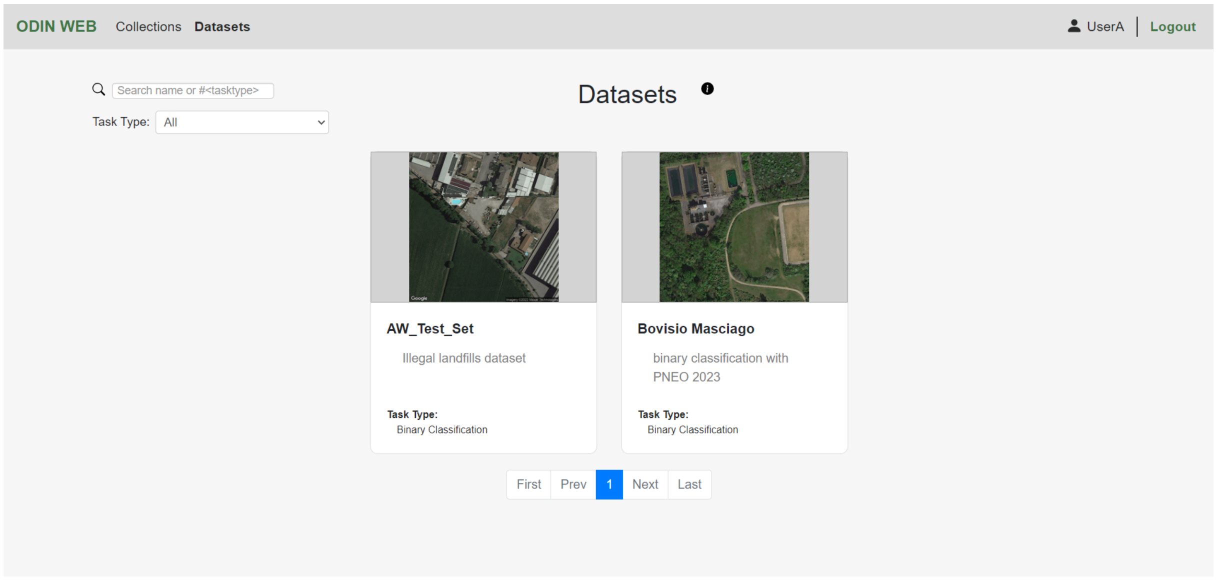
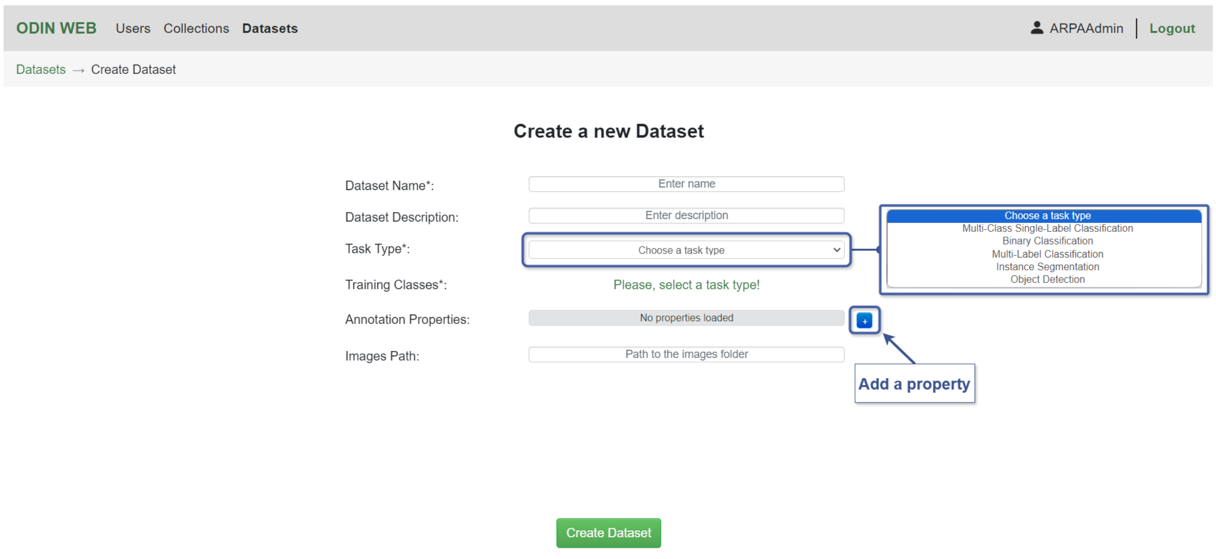


Figure 5.5: Datasets: datasets page for simple users

Dataset Creation

The *Dataset Creation* page allows admins to create a dataset (Figure 5.6). The admin should fill out the dedicated form by inserting the dataset name and an optional description, selecting the task type among the 5 supported ones, defining the training classes, also called categories, and the annotation properties, and inserting the path to the target collection of images. The admin is presented with two different training class definition forms, according to the selected task (Figure 5.7). The properties can be defined in 3 different formats (Figure 5.8): unique, the property has a fixed set of user-defined values, range, the property can assume integer values in a user-defined range, and textual, the property value is an arbitrary text.

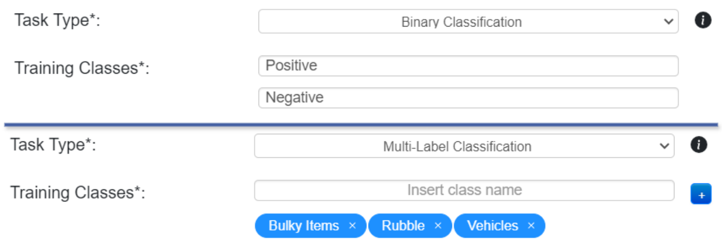

ODIN WEB Users Collections **Datasets** ARPAAdmin | Logout

Datasets → Create Dataset

Create a new Dataset

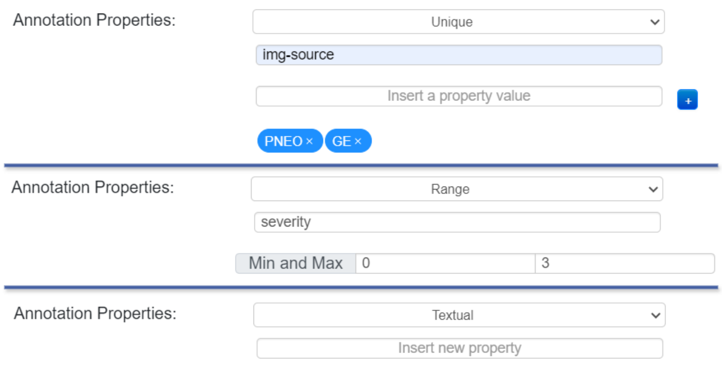
Dataset Name*:	Enter name
Dataset Description:	Enter description
Task Type*:	Choose a task type
Training Classes*:	Please, select a task type!
Annotation Properties:	No properties loaded
Images Path:	Path to the images folder

Figure 5.6: Dataset Creation page


Task Type*:	Binary Classification
Training Classes*:	Positive
	Negative

Task Type*:	Multi-Label Classification
Training Classes*:	Insert class name
	Bulky Items × Rubble × Vehicles ×

Figure 5.7: Dataset Creation: on the top, training classes definition for binary classification, on the bottom, training classes definition for the other four supported task types.


Annotation Properties:	Unique
	img-source
	Insert a property value
	PNEO × GE ×

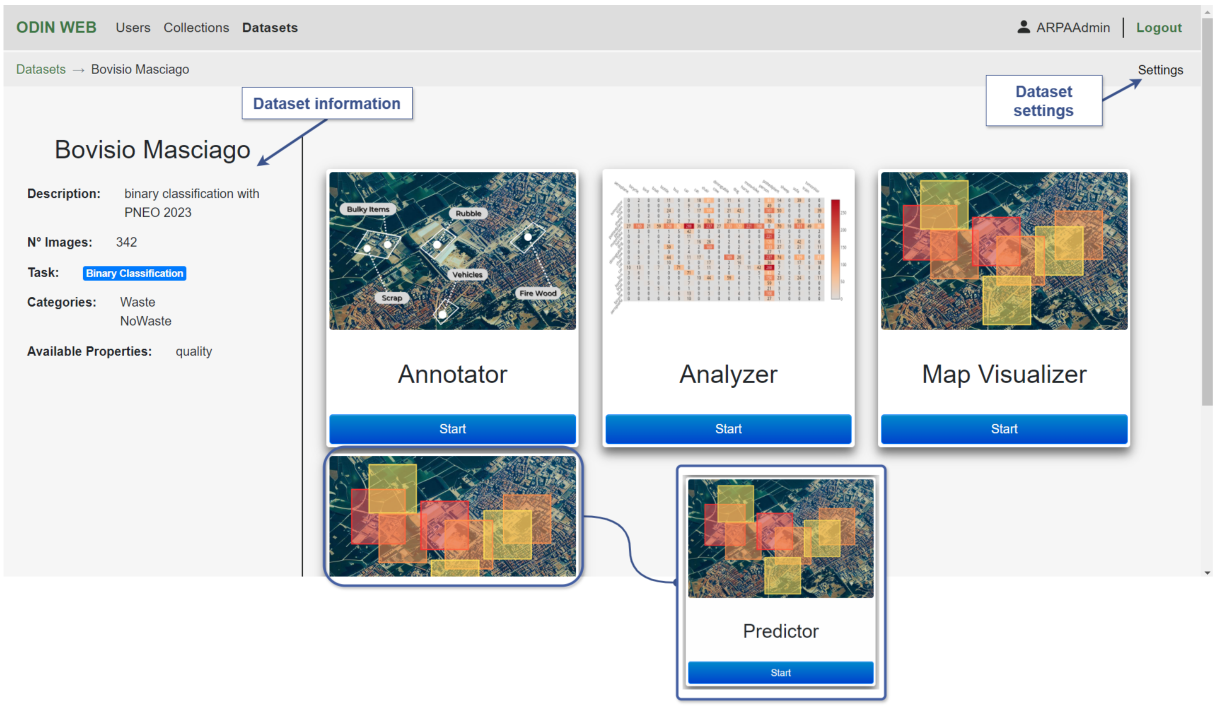
Annotation Properties:	Range
	severity
	Min and Max 0 3

Annotation Properties:	Textual
	Insert new property

Figure 5.8: Dataset Creation: properties definition for unique, range, and textual format.

Dataset Page

The *Dataset Page* is accessed by clicking on the dataset card in the *Datasets* page. This page (Figure 5.9), provides, on the left, a set of information relative to the dataset, such as its name, description, number of images, addressed task, etc. On the right, are displayed the buttons to the 4 functionality sections of the dataset. On the top-right, the *Dataset Settings* can be accessed, only by admins, through the relative button.



Description: binary classification with PNEO 2023

N° Images: 342

Task: Binary Classification

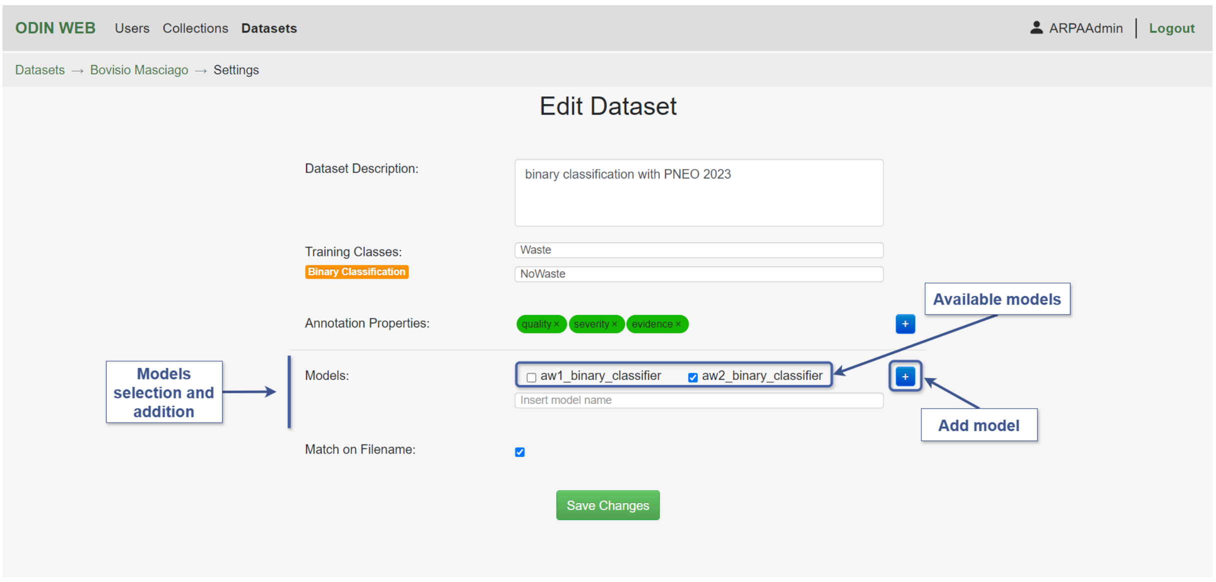
Categories: Waste NoWaste

Available Properties: quality

Figure 5.9: Dataset Page

Dataset Settings

The *Dataset Settings* page allows the admin to edit the dataset settings, select models from the available ones, or add other models (Figure 5.10). The models available at the moment on ODIN Web are the solid waste classification models: *aerialwaste binary classifier (v1)* [73], *aerialwaste binary classifier (v2)*, and *aerialwaste multi-class classifier*. The addition of custom models results in the creation of directories, in which the user can upload the predictions he obtained by running the model on the dataset.



Edit Dataset

Dataset Description:	binary classification with PNEO 2023
Training Classes: Binary Classification	Waste
	NoWaste
Annotation Properties:	quality × severity × evidence ×
Models:	☐ aw1_binary_classifier ☑ aw2_binary_classifier
	Insert model name
Match on Filename:	☑

Figure 5.10: Dataset Settings

5.1.4. Annotator

The *Annotator* is accessible by clicking on the relative button on the *Dataset Page* of a target dataset. This page (Figure 5.11), presents a paginated grid of the dataset images. On the left, a menu allows the user to decide the number of images displayed on one page, and apply filters to visualize only a set of the images. Buttons for the upload/download of ground-truth are available under the filters. The user can navigate through the grid and click on an image to open it in the annotation interface.

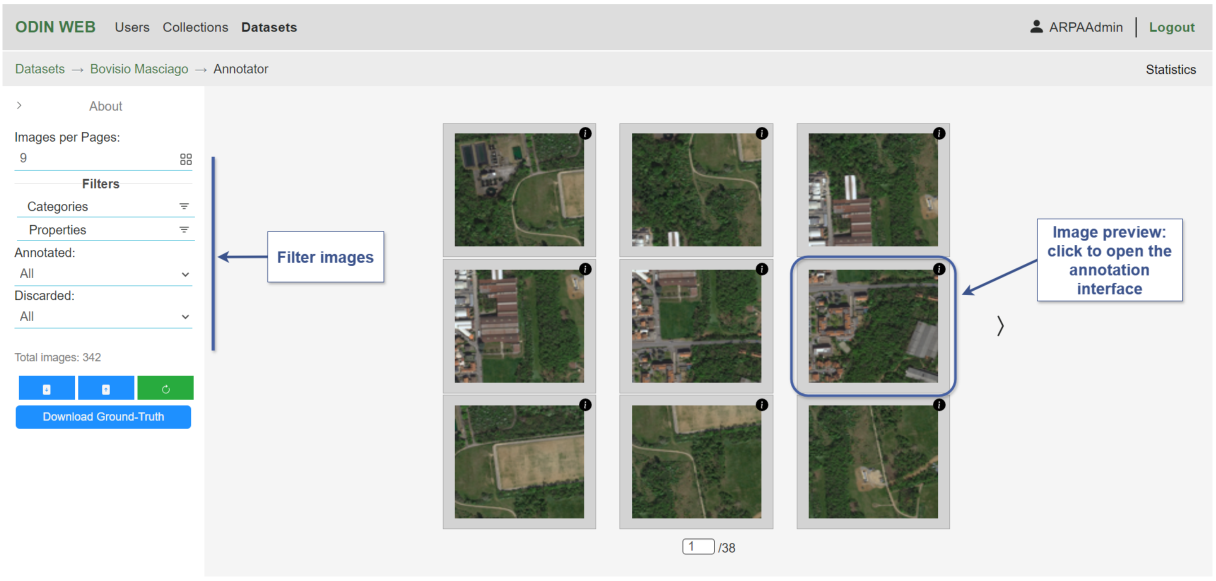
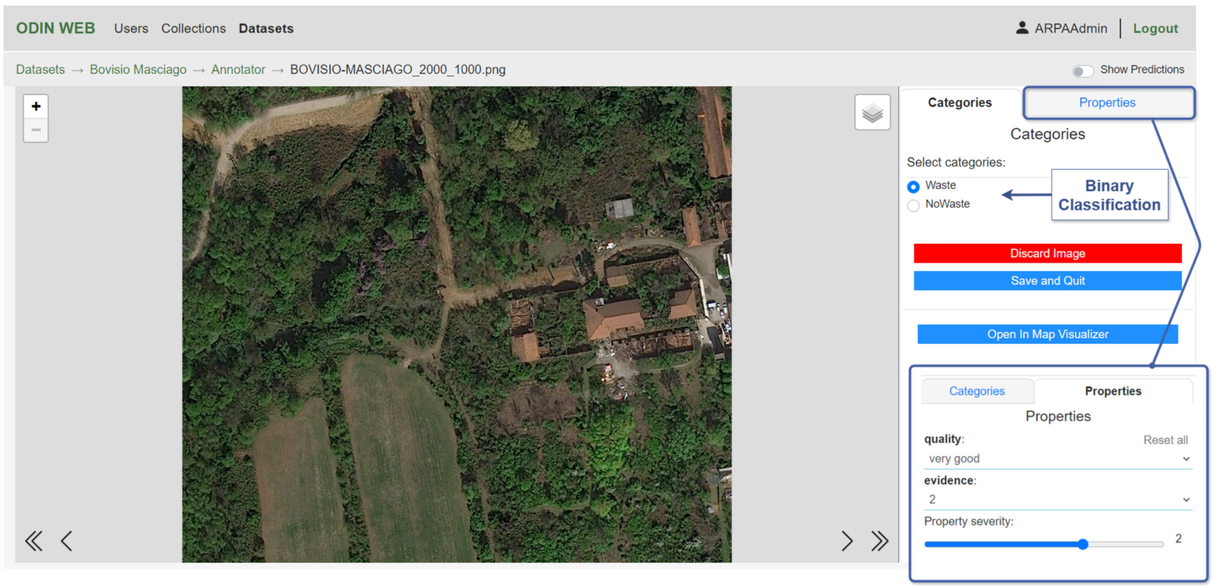


Figure 5.11: Annotator

The *Annotation Interface* (Figure 5.12) displays the image in the center and a section on the right that comprises a *Categories* tab, for the class assignment, and a *Properties* tab, for the meta-properties annotation. The annotation tools change according to the computer vision task of the dataset as illustrated in Figure 5.12 and 5.13.


Datasets → Bovisio Masciago → Annotator → BOVISIO-MASCIAGO_2000_1000.png

Select categories: [✓ Waste] NoWaste

quality: very good

evidence: 2

Property severity: 2

Figure 5.12: Annotator: an overview of the annotation interface (binary classification).

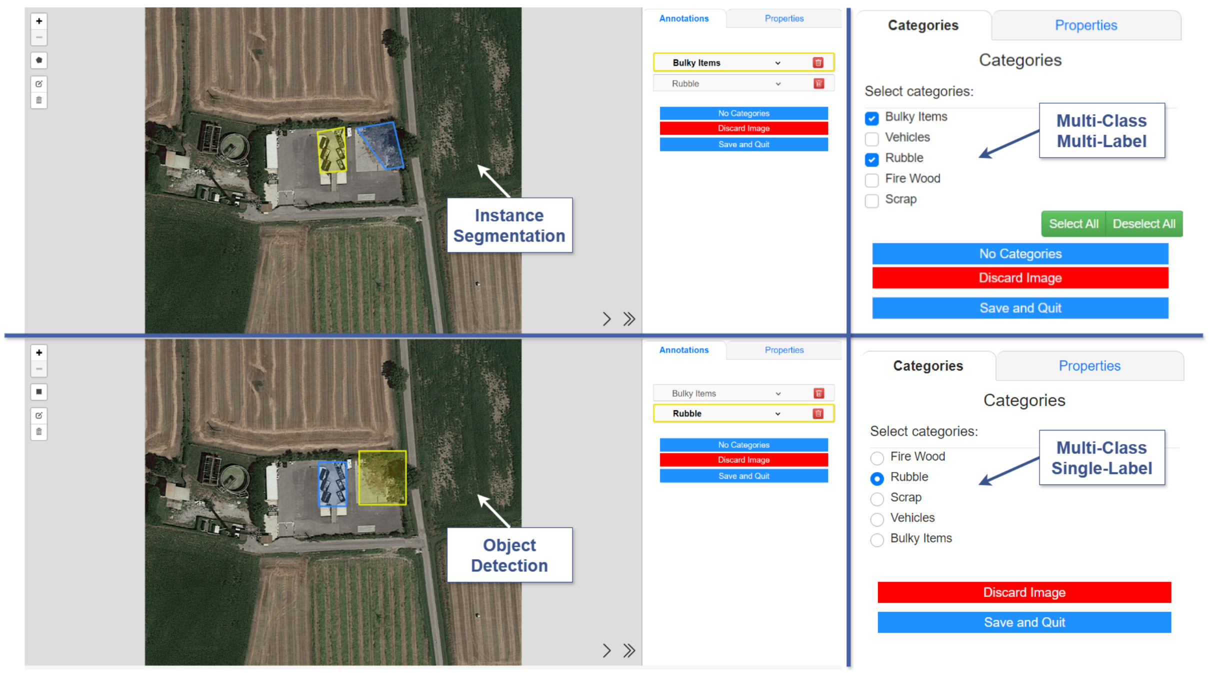


Figure 5.13: Annotator: annotation interface for instance segmentation, object detection, multi-class multi-label classification, and multi-class single-label classification.

When predictions are available for the current dataset, a switch toggle appears on the top-right of the annotation interface, allowing the user to decide whether he wants to visualize the prediction scores in the section on the right or not. Moreover, if the Class Activation Map (CAM) is available, it can be displayed over the image, by enabling it from the layer controls. Finally, a button is available, for the geolocalized datasets that have the GeoJSON prediction files, to open the image in the *Map Visualizer*.

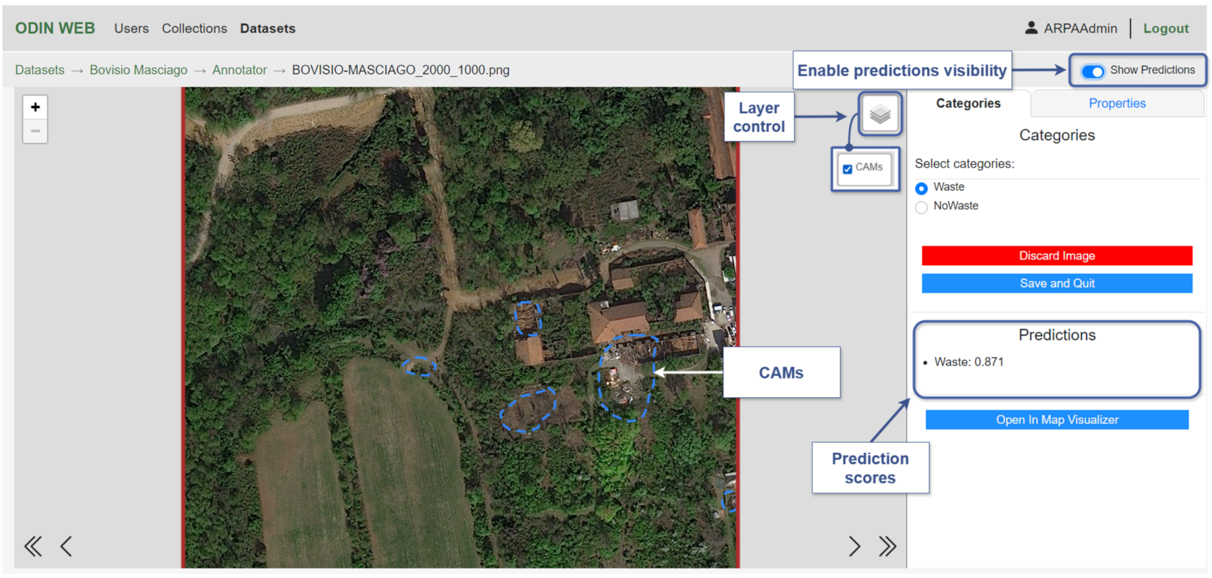


Figure 5.14: Annotator: visualization of predictions

5.1.5. Analyzer

The *Analyzer* is accessible by clicking on the relative button on the *Dataset Page* of a target dataset. It is divided into two main areas respectively: the *Dataset Analysis*, the analysis of categories and properties distribution within the dataset, and the *Predictions Analysis*, enabled if predictions are available, for the model performance analysis. The *Predictions Analysis* is itself divided into multiple sections, accessible from the dropdown menu, depending on the type of analysis they implement.

Each section of the analyzer presents on the left a column menu with filters and selectors, on the top a navbar to navigate among different declinations of the same analysis type, and on the remaining part the specific analysis diagrams and tables.

In Figure 5.15 is shown the *Dataset Analysis* section. In Figure 5.16 and 5.17 are shown two different *Predictions Analysis*, respectively the *TP, TN, FP, FN Distributions* and the *Performance Curves* (PR, ROC, F1).

The *Analyzer* provides all the metrics and analyses discussed in Chapter 2.2, and the comparison between multiple models. It allows both for per-category diagnosis, and per-property diagnosis, which leverages the meta-annotations to perform an analysis on subsets of the dataset, according to some property value, and compute the impact of the properties. The analysis settings discussed in the Paragraph 4.2.2 are editable in the analyzer *Settings* section, accessible from the relative button on the top-right.

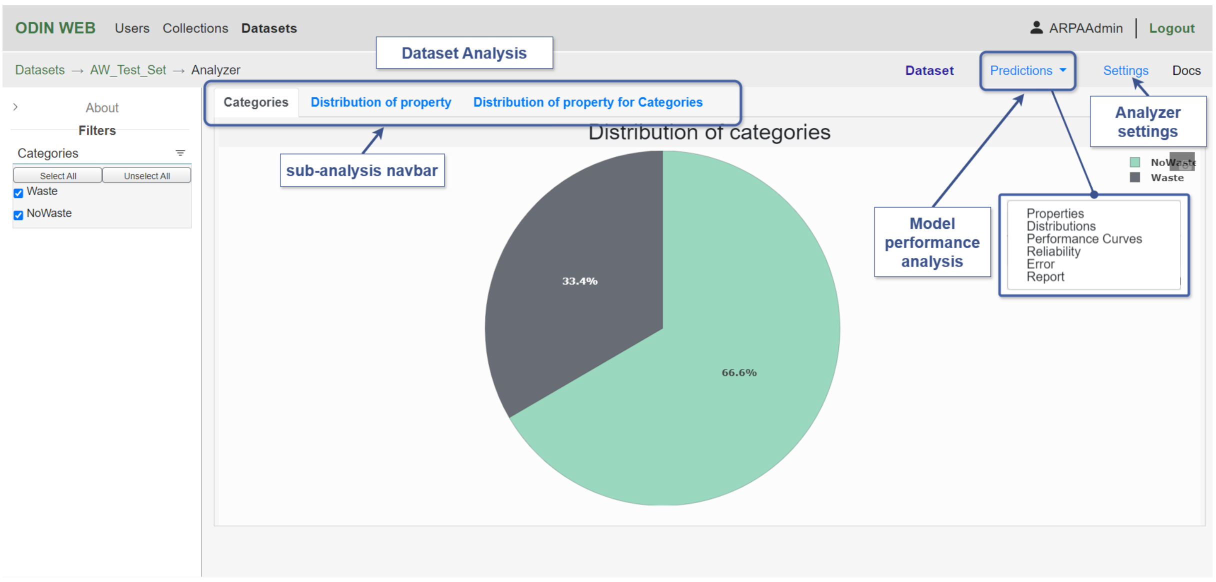


Figure 5.15: Analyzer: Dataset Analysis section

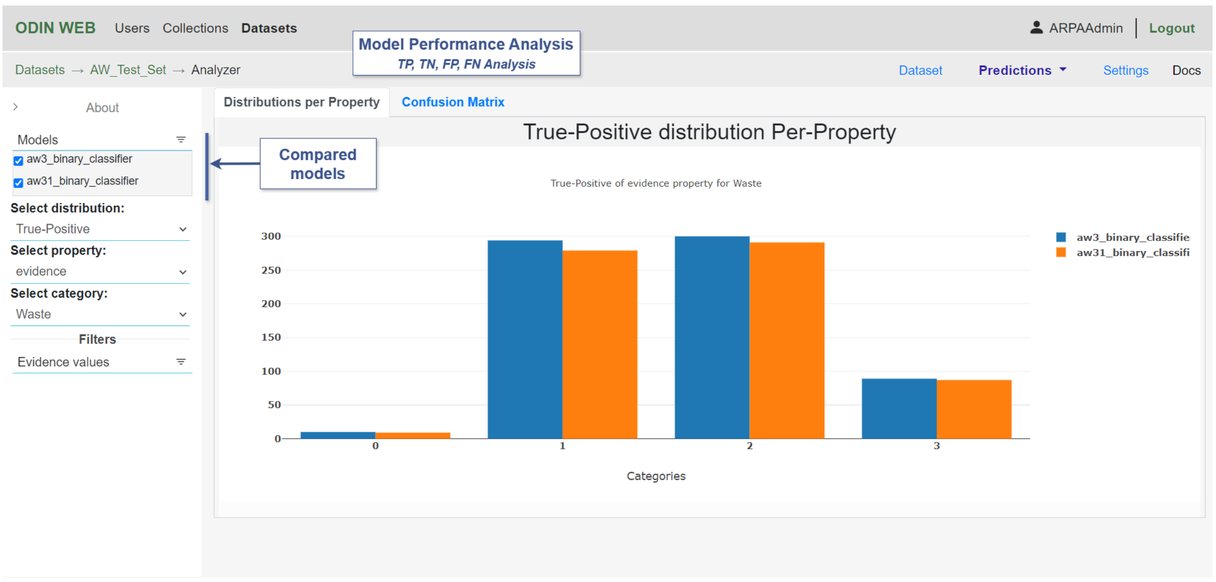


Figure 5.16: Analyzer: the comparative analysis of the *TP, TN, FP, FN Distributions* of two models. Accessible by selecting *Distributions* from the *Predictions* dropdown menu.

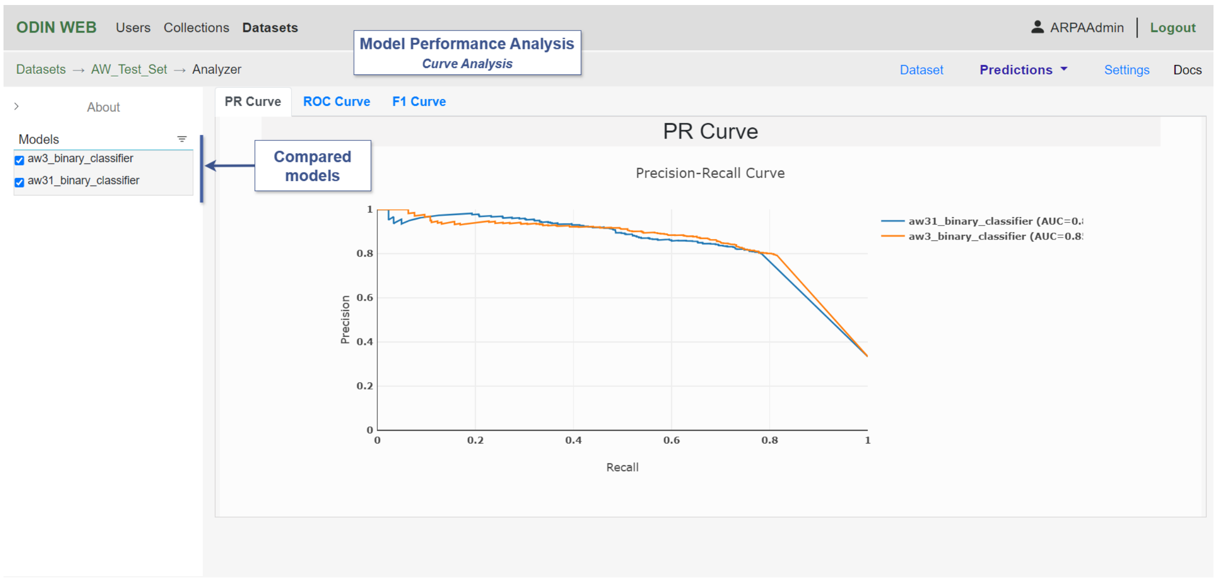


Figure 5.17: Analyzer: the comparative analysis of the *PR Curves* of two models. Accessible by selecting *Performance Curves* from the *Predictions* dropdown menu.

5.1.6. Map Visualizer

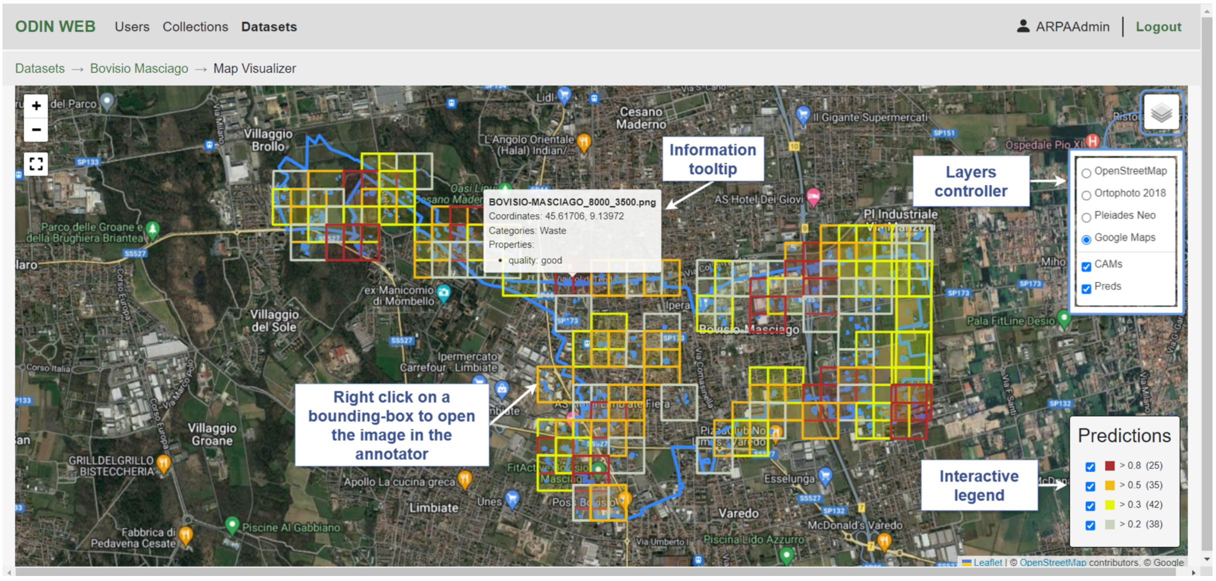
When the dataset is geolocalized and predictions in GeoJSON format have been uploaded, the *Map Visualizer* button on the *Dataset Page* is enabled. Once entered the *Map Visualizer*, the map is displayed, which by default is the Google Maps one. Over the map are rendered (Figure 5.18):

- the perimeter of the area of interest;
- the predictions in the form of color-coded bounding-boxes;
- the activation maps (CAMs), i.e., the contours of the areas the model focused on.

From the *layers controller* in the top-right corner, the user can select a different map source, and enable or disable the visibility of CAMs and predictions. The map sources available are Google Maps, OpenStreetMap, Ortophoto2018, and Pleaiades Neo. The last two are partial maps, that cover only some specific areas in Lombardy.

From the *interactive legend* in the bottom-right corner, the user can enable or disable the bounding-boxes and CAMs with the associated prediction belonging to a certain confidence interval. The legend shows, for each confidence interval, the number of the corresponding bounding-boxes.

The user can move around the map through *Click and Drag*, and zoom in and out the view. By hovering over each *bounding-box*, a tooltip is opened displaying the name of the corresponding image in the dataset, the *coordinates* of the center of the square, and the annotations associated with the image for both categories and properties. With a right-click on the target square, a context menu is opened from which the user can open the image in the *Annotator*.

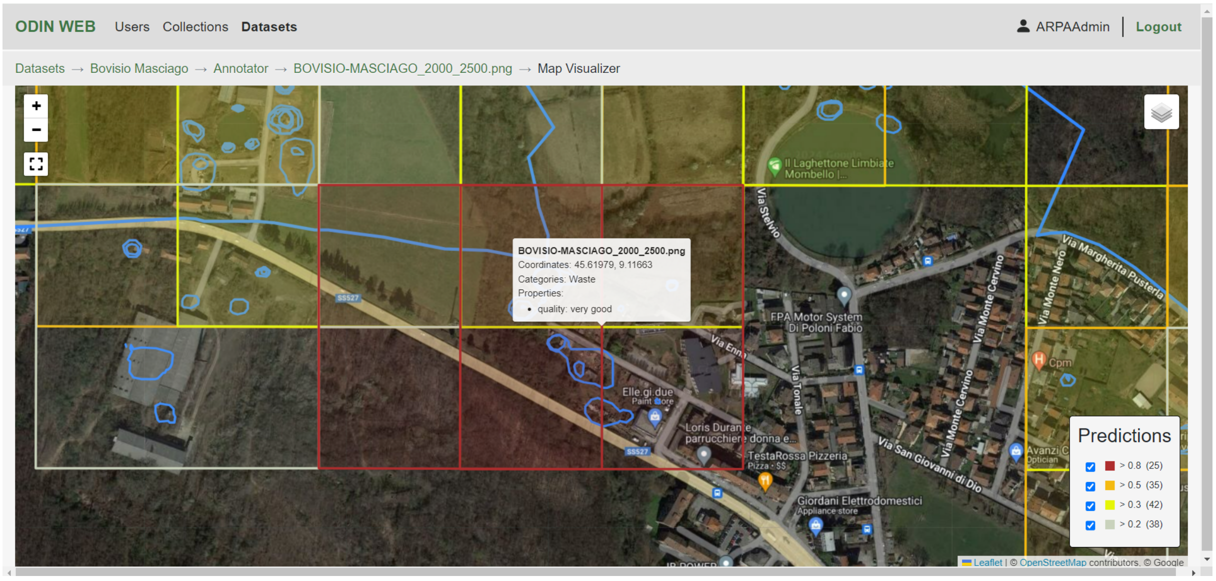


BOVISIO-MASCIAGO_8000_3500.png
Coordinates: 45.61706, 9.13972
Categories: Waste
Properties:
- quality: good

Predictions
- 🟥 > 0.8 (25)
- 🟧 > 0.5 (35)
- 🟨 > 0.3 (42)
- ⬜ > 0.2 (38)

Figure 5.18: Map Visualizer

When the user clicks on the *Open in Map Visualizer* button in the *Annotator Interface* the *Map Visualizer* is opened. The map is focused and zoomed automatically on the position on the map corresponding to the geolocalized satellite image (Figure 5.19).


ODIN WEB
Users
Collections
Datasets
ARPAAdmin
Logout
Datasets → Bovisio Masciago → Annotator → BOVISIO-MASCIAGO_2000_2500.png → Map Visualizer

BOVISIO-MASCIAGO_2000_2500.png
Coordinates: 45.61979, 9.11663
Categories: Waste
Properties:
- quality: very good

Predictions
- 🟥 > 0.8 (25)
- 🟧 > 0.5 (35)
- 🟨 > 0.3 (42)
- ⬜ > 0.2 (38)

Figure 5.19: Map Visualizer: map zoomed on the coordinates of the opened image.

5.1.7. Predictor

The *Predictor* is accessible by clicking on the relative button on the *Dataset Page* of a target dataset. From this page, the user can select a model from the dropdown menu and run the inference on the dataset (Figure 5.20). The models the user can choose from, are those previously selected in the *Dataset Settings* 5.1.3.

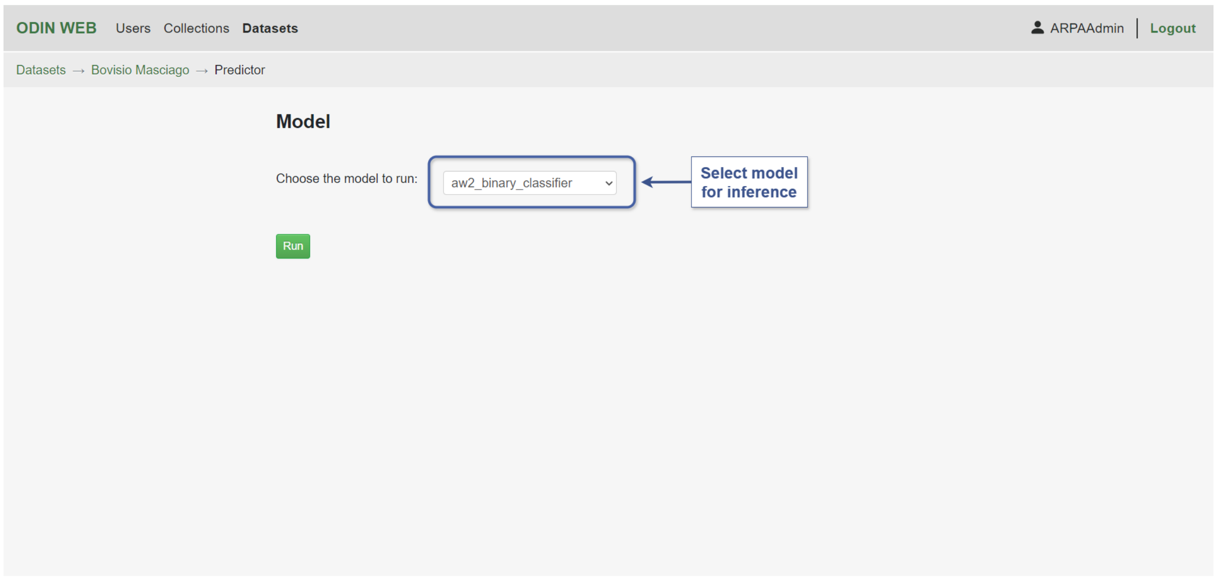


Figure 5.20: Predictor

While the model is running a progress bar is displayed. Under the bar, on the left is reported the name of the image that is currently being processed and the estimated remaining time to inference completion. On the right is displayed the percentage of completion. (Figure 5.21). While the model is running the user can freely navigate throughout the rest of the website.

The results of the prediction are stored on the server in the dataset directory inside the appropriate folder associated with the executed model. Predictions are stored in TXT files, following the format required for the analysis by the ODIN framework. Multiple models can be run on the same dataset, to obtain the previously illustrated comparative analysis in the *Analyzer*(Section 5.1.5).

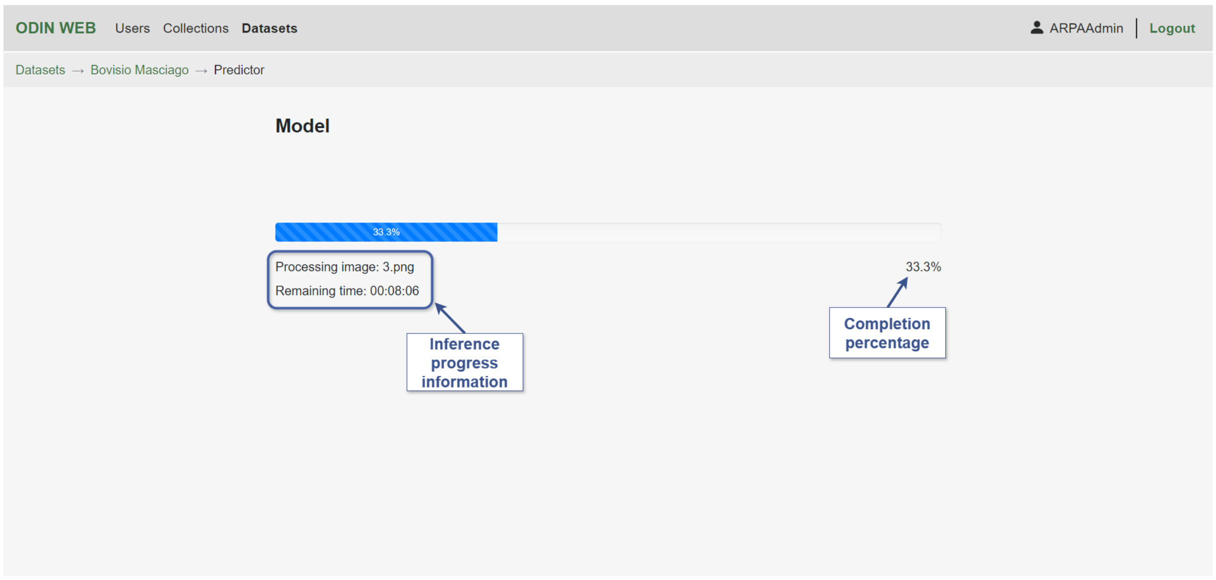


Figure 5.21: Predictor: progress of the running inference

5.2. Validation

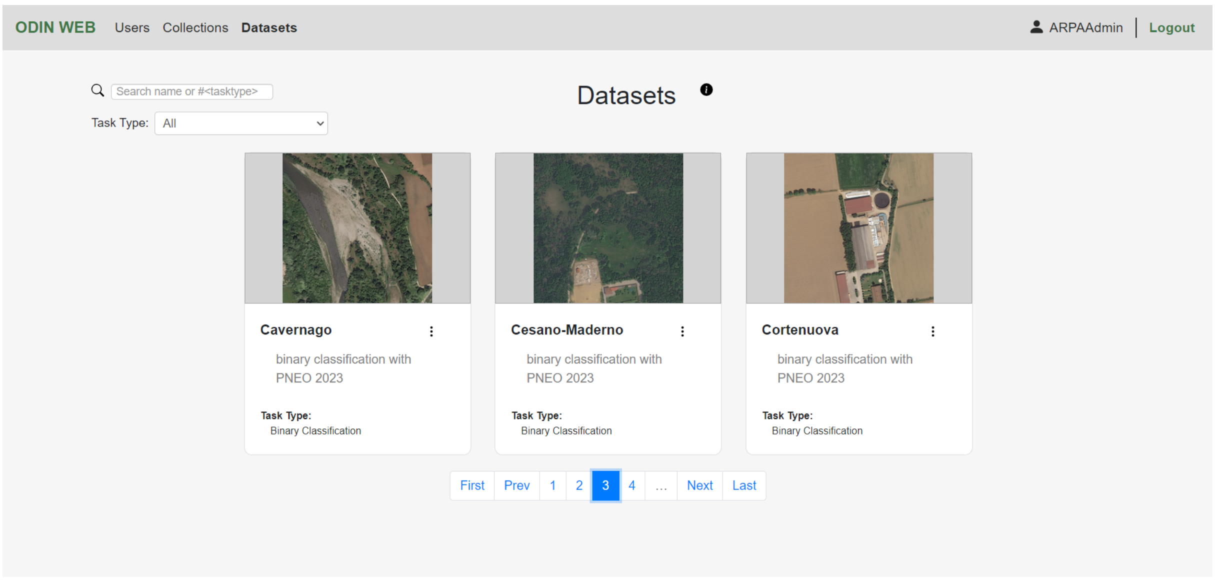
ODIN Web was validated internally, through a set of different application use cases, and externally by a team of photo interpreters from ARPA Lombardia (Environmental Protection Agency of the Region of Lombardy) to verify its effectiveness and compliance with the requirements.

Predictions Analysis on PNEO Municipalities

For this use case, given the PNEO satellite images of 20 municipalities located in Lombardy, a dataset was created for each municipality. All datasets were defined for the binary classification task, with categories *Waste* and *NoWaste*, and three properties: *quality*, to annotate the prediction quality as "bad", "good", or "very good", *evidence*, and *severity*.

The predictions in TXT format for performance analysis and in GeoJSON format for map visualization were uploaded to the server. For two of the municipalities, instead, the predictions in TXT format were generated by running the inference with the *aerialwaste binary classifier (v2)* model directly from ODIN Web to test the functionality.

The datasets were assigned to the members of the internal team through the user management system. Each member was asked to analyze 2 or 3 datasets. The task on each dataset was to verify the quality of the predictions with a confidence level greater than 0.8. This was done by opening the map visualization, filtering out, through the legend, the bounding boxes with lower confidence, and opening each image in the annotator vi-



Cavernago
binary classification with PNEO 2023
Task Type: Binary Classification

Cesano-Maderno
binary classification with PNEO 2023
Task Type: Binary Classification

Cortenuova
binary classification with PNEO 2023
Task Type: Binary Classification

Figure 5.22: Datasets for the predictions analysis on PNEO municipalities use case

sualizing the CAM on the image and choosing a value between "bad", "good", and "very good" for the quality property.

Model Performance Analysis on AerialWaste Test Set

For this use case, a dataset for binary classification has been created employing one of the test sets from the AerialWaste2.0 dataset [74]. The dataset comprehends 2605 satellite images. The categories are *Waste* and *NoWaste*, while the properties are *evidence*, *is candidate location*, *site type*, *severity*, *valid fine grain*, and *image source*. The annotations for those classes and categories, available at [74], were uploaded on the server. The models to be analyzed were not available on ODIN Web, as such inference was run offline and the predictions were uploaded in TXT format. Then the performances of the two models were analyzed and compared in the Analyzer.

Model Performance Analysis and Comparison on AREU dataset

For this use case, a dataset, with 2625 aerial images captured with UAVs, has been created in the context of the AREU (Regional Emergency Ugency Agency) project, which aims at the identification of targets in aerial images. The task for the dataset is binary classification with classes *Target*, and *Background*, the property for meta-annotation is the *occlusion*, which can be "none", "low", "medium", or "strong". The predictions over this dataset have been computed offline with three models, *LeNetV3*, *MobileNetV3Small*, and *EfficientNetB0*, and then uploaded for exploring the model performance analysis and the comparison between the three binary classifiers in the Analyzer.

Annotation Analysis of Multi-Class Single-Label on FORESTAMI dataset

For this use case, a dataset with 714 satellite images has been created in the context of the FORESTAMI project. The dataset was uploaded together with the annotations. The objective was to verify the correctness of the multi-class single-label annotations through the Annotator of ODIN Web. The categories for classification are *areas for storing building materials or other aggregates and/or with unauthorized activities*, *uncultivated and/or abandoned areas*, *informal urban gardens*, *waste disposal areas and any informal artifacts*, and others. The properties associated are *waste*, if present or not, and *vegetation* type.

Re-Annotation of AerialWaste 1.0 dataset by ARPA

For the re-annotation of the AerialWaste 1.0 dataset [72], the ARPA team of expert photo interpreters is involved. For this purpose, an admin account was created for the team administrator. The team is composed of five more experts, thus the ARPA admin registered five users.

The AerialWaste dataset, composed of 3478 satellite images, was split into eight subsets. For each of those subsets, a dataset for multi-class multi-label classification was created in ODIN Web. The admin authorized each user to the datasets he was assigned to work on. The datasets were built as follows:

- Out of the total number of images, 2763 are assumed to contain waste. This set of images was split into five datasets. The objective of the annotation is to define whether the assumption is correct, and in that case, annotate the categories present in the image.
- For the remaining 715 images, the annotations were already available and they were uploaded to ODIN Web. This set of images was split into three datasets. The objective is to verify the previous annotations and correct them when necessary.

The application use cases above described allowed the validation of ODIN Web on the following functionalities: user creation and management, dataset creation and annotation, model inference, model performance analysis and model comparison, and predictions visualization on the map.

6 | Conclusions and Future Work

In this thesis, the ODIN Web application has been described. ODIN Web has been developed to address the necessity for tools that can expedite the image annotation process, and for methods that allow the evaluation of the performance of complex models. Additionally, the objective is to offer user-friendly instruments that can be accessed by individuals with no technical knowledge to operate with Computer Vision models.

ODIN Web [52] is a tool that integrates, in a web-based application: dataset management, image annotation for tasks such as image classification, object detection, and instance segmentation, and model performance investigation and comparison through the ODIN framework. ODIN [76] is a Python tool that offers extensive "black-box" analysis instruments. In this work, ODIN Web has been extended with further functionalities with the aim of realizing a web-based tool that covers the most steps in the model development pipeline, from data annotation to the application of Computer Vision models. The web application was enriched with a user system involving admins and simple users, that allows for collaboration and role-based access control of datasets. The dataset management system was adapted to fit the new user system. Finally, model inference execution was integrated, and a geo-visualization interface for geolocalized satellite images and predictions was implemented.

In conclusion, an analysis of the available tools for performance analysis and for image annotation highlighted that tools tend to specialize in either one of the two problems. There are a few integrated comparable tools that, although, lack an exhaustive model performance analysis and some functionalities available on ODIN Web. As such, ODIN Web can be considered an innovative and valuable tool.

Future work on ODIN Web will concentrate on:

- implementing the Collections section to allow agile upload and management of collections of images;
- implementing a Models section for more sophisticated management of models, for both the ones available on the platform and the ones added by the users;

- enhancing the analysis of models even more, by including functionalities such as model performance comparison on samples;
- integrating the inference of users' custom models through REST API;
- implementing automatic retrieval of image tiles starting from uploaded shape files.

Bibliography

[1] D. Acuna, H. Ling, A. Kar, and S. Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 859–868, 2018.

[2] K. Alhazmi, W. Alsumari, I. Seppo, L. Podkuiko, and M. Simon. Effects of annotation quality on model performance. In *2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 063–067. IEEE, 2021.

[3] H. Alwassel, F. C. Heilbron, V. Escorcia, and B. Ghanem. Diagnosing error in temporal action detectors. In *Proceedings of the European conference on computer vision (ECCV)*, pages 256–272, 2018.

[4] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 337–346, 2015.

[5] G. Bilotta, V. Barrile, and G. M. Meduri. Recognition and classification of illegal dumps with object based image analysis of satellite data. *A-+ A*, 1:4, 2012.

[6] D. Bolya, S. Foley, J. Hays, and J. Hoffman. Tide: A general toolbox for identifying object detection errors. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 558–573. Springer, 2020.

[7] J. Brooks. COCO Annotator. `https://github.com/jsbroks/coco-annotator/`, 2019.

[8] H. I. Chaminé, A. J. Pereira, A. C. Teodoro, and J. Teixeira. Remote sensing and gis applications in earth and environmental systems sciences, 2021.

[9] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-

decoder with atrous separable convolution for semantic image segmentation, 2018. arXiv:1802.02611.

[10] F. Chollet. Xception: Deep learning with depthwise separable convolutions, 2017. `arXiv:1610.02357`.

[11] G. Ciaparrone, F. L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera. Deep learning in video multi-object tracking: A survey. *Neurocomputing*, 381:61–88, 2020.

[12] CVAT.ai-Corporation. Computer Vision Annotation Tool (CVAT), Feb. 2024. URL `https://doi.org/10.5281/zenodo.10696672`. Available: `https://www.cvat.ai/`.

[13] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006.

[14] A. Demidovskij, A. Tugaryov, A. Kashchikhin, A. Suvorov, Y. Tarkan, F. Mikhail, and G. Yury. Openvino deep learning workbench: towards analytical platform for neural networks inference optimization. In *Journal of physics: Conference series*, volume 1828, page 012012. IOP Publishing, 2021.

[15] M. R. Devesa and A. V. Brust. Mapping illegal waste dumping sites with neural-network classification of satellite imagery. *arXiv preprint arXiv:2110.08599*, 2021.

[16] A. Dutta and A. Zisserman. The via annotation software for images, audio and video. In *Proceedings of the 27th ACM international conference on multimedia*, pages 2276–2279, 2019.

[17] B. Dwyer, J. Nelson, J. Solawetz, and et. al. Roboflow (Version 1.0) [Software]., 2022. Available from: `https://roboflow.com`.

[18] E. Elyan, P. Vuttipittayamongkol, P. Johnston, K. Martin, K. McPherson, C. Jayne, M. K. Sarker, et al. Computer vision and machine learning for medical image analysis: recent advances, challenges, and way forward. *Artificial Intelligence Surgery*, 2, 2022.

[19] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

[20] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111:98–136, 2015.

[21] F. Faizi, K. Mahmood, M. Chaudhry, and A. Rana. Satellite remote sensing and image processing techniques for monitoring msw dumps. In *Proccedings of 5th EurAsia Waste Management Symposium*, pages 26–28, 2020.

[22] C. Ferri, J. Hernández-Orallo, and R. Modroiu. An experimental comparison of performance measures for classification. *Pattern recognition letters*, 30(1):27–38, 2009.

[23] N. Fiedler, M. Bestmann, and N. Hendrich. Imagetagger: An open source online platform for collaborative image labeling. In *RoboCup 2018: Robot World Cup XXII 22*, pages 162–169. Springer, 2019.

[24] P. Fraternali, F. Milani, R. N. Torres, and N. Zangrando. Black-box error diagnosis in deep neural networks for computer vision: a survey of tools. *Neural Computing and Applications*, 35(4):3041–3062, 2023.

[25] P. Fraternali, L. Morandini, and S. L. H. González. Solid waste detection in remote sensing images: A survey. *arXiv preprint arXiv:2402.09066*, 2024.

[26] M. Gleicher, A. Barve, X. Yu, and F. Heimerl. Boxer: Interactive comparison of classifier results. In *Computer Graphics Forum*, volume 39, pages 181–193. Wiley Online Library, 2020.

[27] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[28] A. K. Gupta. ImgLab. `https://github.com/NaturalIntelligence/imglab`, 2018. Available: `https://solothought.com/imglab/`.

[29] W. Halaschek, J. Golbeck, A. Schain, M. Grove, B. Parsia, and J. Hendler. Photostuff: An image annotation tool for the semantic web. In *Proceedings of the Poster Track, 4th International Semantic Web Conference*, pages 2–4, 2005.

[30] A. Hanbury. A survey of methods for image annotation. *Journal of Visual Languages & Computing*, 19(5):617–627, 2008.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. arXiv:1512.03385.

[32] G. T. S. Ho, Y. P. Tsang, C. H. Wu, W. H. Wong, and K. L. Choy. A computer vision-based roadside occupation surveillance system for intelligent transport in smart cities. *Sensors*, 19(8):1796, 2019.

[33] D. Hoiem, Y. Chodpathumwan, and Q. Dai. Diagnosing error in object detectors. In *European conference on computer vision*, pages 340–353. Springer, 2012.

[34] M. Hossin and M. N. Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.

[35] J. Huang and C. X. Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering*, 17(3):299–310, 2005.

[36] M. Đidelija, N. Kulo, A. Mulahusić, N. Tuno, and J. Topoljak. Segmentation scale parameter influence on the accuracy of detecting illegal landfills on satellite imagery. a case study for novo sarajevo. *Ecological Informatics*, 70:101755, 2022.

[37] M. B. Jones, J. Bradley, and N. Sakimura. JSON Web Token (JWT). RFC 7519, May 2015. URL `https://www.rfc-editor.org/info/rfc7519`.

[38] A. Kar, S. W. Kim, M. Boben, J. Gao, T. Li, H. Ling, Z. Wang, and S. Fidler. Toronto annotation suite. `https://aidemos.cs.toronto.edu/toras`, 2021.

[39] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollar. Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[40] Y. Kong and Y. Fu. Human action recognition and prediction: A survey. *International Journal of Computer Vision*, 130(5):1366–1401, 2022.

[41] M. Kräter, S. Abuhattum, D. Soteriou, A. Jacobi, T. Krüger, J. Guck, and M. Herbig. Aideveloper: deep learning image classification in life science and beyond. *Advanced science*, 8(11):2003743, 2021.

[42] J. Krause, A. Perer, and K. Ng. Interacting with predictions: Visual inspection of black-box machine learning models. In *Proceedings of the 2016 CHI conference on human factors in computing systems*, pages 5686–5697, 2016.

[43] J. Krause, A. Dasgupta, J. Swartz, Y. Aphinyanaphongs, and E. Bertini. A workflow for visual diagnostics of binary classifiers using instance-level explanations. In *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 162–172. IEEE, 2017.

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[45] C. Kruse, E. Boyda, S. Chen, K. Karra, T. Bou-Nahra, D. Hammer, J. Mathis, T. Maddalene, J. Jambeck, and F. Laurier. Satellite monitoring of terrestrial plastic waste. *PloS one*, 18(1):e0278997, 2023.

[46] Labelbox. "Labelbox," Online, 2024. [Online]. Available: `https://labelbox.com`.

[47] S. Lavender. Detection of waste plastics in the environment: Application of copernicus earth observation data. *Remote Sensing*, 14(19):4772, 2022.

[48] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[49] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017. doi: 10.1109/CVPR.2017.106.

[50] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen. Deep learning for generic object detection: A survey. *International journal of computer vision*, 128:261–318, 2020.

[51] Y. Long, G.-S. Xia, S. Li, W. Yang, M. Y. Yang, X. X. Zhu, L. Zhang, and D. Li. On creating benchmark dataset for aerial image interpretation: Reviews, guidances, and million-aid. *IEEE Journal of selected topics in applied earth observations and remote sensing*, 14:4205–4230, 2021.

[52] A. Mastropasqua. ODIN Web: an interactive dashboard for black-box deep learning error diagnosis. Master's thesis, Politecnico di Milano, ING - Scuola di Ingegneria Industriale e dell'Informazione, 2022. URL `https://hdl.handle.net/10589/188236`.

[53] J. Memon, M. Sami, R. A. Khan, and M. Uddin. Handwritten optical character recognition (ocr): A comprehensive systematic literature review (slr). *IEEE Access*, 8:142642–142668, 2020.

[54] Microsoft. Visual Object Tagging Tool (VoTT), 2017. URL `https://github.com/microsoft/VoTT`.

[55] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3523–3542, 2021.

[56] J. D. Novaković, A. Veljović, S. S. Ilić, Ž. Papić, and M. Tomović. Evaluation of classification models in machine learning. *Theory and Applications of Mathematics & Computer Science*, 7(1):39, 2017.

[57] R. Padilla, S. L. Netto, and E. A. Da Silva. A survey on performance metrics for object-detection algorithms. In *2020 international conference on systems, signals and image processing (IWSSIP)*, pages 237–242. IEEE, 2020.

[58] D. I. Patrício and R. Rieder. Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review. *Computers and electronics in agriculture*, 153:69–81, 2018.

[59] X. Qin, S. He, Z. Zhang, M. Dehghan, and M. Jagersand. Bylabel: A boundary based semi-automatic image annotation tool. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1804–1813. IEEE, 2018.

[60] A. Rajkumar, C. A. Kft, T. Sziranyi, and A. Majdik. Detecting landfills using multispectral satellite images and deep learning methods, 2022.

[61] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. doi: 10.1109/CVPR.2016.91.

[62] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE transactions on visualization and computer graphics*, 23(1):61–70, 2016.

[63] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[64] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[65] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77:157–173, 2008.

[66] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. arXiv:1409.1556.

[67] P. Skalski. Make Sense. `https://github.com/SkalskiP/make-sense/`, 2019. Available: `https://www.makesense.ai/`.

[68] X. Sun, D. Yin, F. Qin, H. Yu, W. Lu, F. Yao, Q. He, X. Huang, Z. Yan, P. Wang, et al. Revealing influencing factors on global waste distribution via deep-learning based dumpsite detection from satellite imagery. *Nature Communications*, 14(1): 1444, 2023.

[69] N. Taggio, A. Aiello, G. Ceriola, M. Kremezi, V. Kristollari, P. Kolokoussis, V. Karathanassi, and E. Barbone. A combination of machine learning algorithms for marine plastic litter detection exploiting hyperspectral prisma data. *Remote Sensing*, 14(15):3606, 2022.

[70] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov. Label Studio: Data labeling software, 2020-2022. URL `https://github.com/heartexlabs/label-studio`. Open source software available from https://github.com/heartexlabs/label-studio.

[71] R. N. Torres and P. Fraternali. Learning to identify illegal landfills through scene classification in aerial images. *Remote Sensing*, 13(22):4520, 2021.

[72] R. N. Torres and P. Fraternali. Aerialwaste, Aug. 2022. URL `https://doi.org/10.5281/zenodo.7034382`.

[73] R. N. Torres and P. Fraternali. Aerialwaste: A dataset for illegal landfill discovery in aerial images, 2022.

[74] R. N. Torres and P. Fraternali. Aerialwaste, May 2023. URL `https://doi.org/10.5281/zenodo.7991872`.

[75] R. N. Torres, P. Fraternali, and J. Romero. Odin: An object detection and instance segmentation diagnosis framework. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*, pages 19–31. Springer, 2020.

[76] R. N. Torres, F. Milani, and P. Fraternali. Odin: Pluggable meta-annotations and metrics for the diagnosis of classification and localization. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 383–398. Springer, 2021.

[77] Tzutalin. LabelImg. git code (2015), 2015. URL `https://github.com/tzutalin/labelImg`.

[78] Y. Z. Ulloa-Torrealba, A. Schmitt, M. Wurm, and H. Taubenböck. Litter on the streets-solid waste detection using vhr images. *European Journal of Remote Sensing*, 56(1):2176006, 2023.

[79] V7. V7: AI Data Engine for Computer Vision and Generative AI, 2018. URL `https://www.v7labs.com/`.

[80] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viégas, and J. Wilson. The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics*, 26(1):56–65, 2019.

[81] K. Yang, C. Zhang, T. Luo, and L. Hu. Automatic identification method of construction and demolition waste based on deep learning and gaofen-2 data. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 43:1293–1299, 2022.

[82] H. Yoon, S.-H. Lee, and M. Park. Tensorflow with user friendly graphical framework for object detection api. *arXiv preprint arXiv:2006.06385*, 2020.

[83] N. Zangrando. The ODIN framework, a tool for image classification diagnosis. Master's thesis, Politecnico di Milano, ING - Scuola di Ingegneria Industriale e dell'Informazione, 2021. URL `https://hdl.handle.net/10589/177405`.

[84] J. Zhang, Y. Wang, P. Molino, L. Li, and D. S. Ebert. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE transactions on visualization and computer graphics*, 25(1):364–373, 2018.

[85] C. Zheng, W. Wu, C. Chen, T. Yang, S. Zhu, J. Shen, N. Kehtarnavaz, and M. Shah. Deep learning-based human pose estimation: A survey. *ACM Computing Surveys*, 56(1):1–37, 2023.

[86] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

[87] L. Zhou, X. Rao, Y. Li, X. Zuo, Y. Liu, Y. Lin, and Y. Yang. Swdet: Anchor-based object detector for solid waste detection in aerial images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 16:306–320, 2022.

List of Figures

List of Tables

Acknowledgements

Ringrazio il professor Piero Fraternali per l'opportunità che mi ha dato di svolgere questa tesi, per la sua passione, la sua disponibiltà e il suo supporto durante questo percorso.

Ringrazio Sergio Herrera per l'aiuto e il sostegno con cui mi ha affiancato durante tutti questi mesi. Allo stesso modo, ringrazio anche l'*AerialWaste Task Force*.

Ringrazio il mio fidanzato Francesco per aver sempre creduto in me ed esserci stato sempre per me, per avermi sempre supportata ... e sopportata.

Ringrazio mio fratello Gabriele e i miei genitori per avermi sempre sostenuta e incoraggiata in tutti questi anni.

Ringrazio la RdG Monza per l'affetto dimostratomi.

Ringrazio i miei amici e colleghi politecnici per i bei momenti passati insieme.