



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

ODIN Web: An interactive dashboard for black-box deep learning error diagnosis

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING - INGEGNERIA IN-
FORMATICA

Author: **Alessandro Mastropasqua**

Student ID: 944996

Advisor: Prof. Piero Fraternali

Co-advisors: Federico Milani, Rocio Nahime Torres, Niccolò Zangrando

Academic Year: 2021-22

Abstract

Classification, Object Detection, and Instance Segmentation have become key fields in the computer vision research context. Their aim is to recognize the presence, or presence and location, of a specific class within an image in an automatic way using neural models. Over the years, however, the architectures of these models have become progressively complex due to the constant need to increase their performance. This has made their evaluation more and more arduous, leading to evaluate them as "black-boxes" using standard evaluation metrics necessary to perform fair analysis on models and understand their behavior to be able to optimize it for a given dataset or task.

In this way, several papers describe the development of frameworks aimed at enhancing the diagnosis of deep neural models over the standard evaluation metrics, implementing tools with graphical interfaces that support the user during all analyzes by presenting the results cleanly and legibly. The interfaces that these tools present are often sparse and do not allow the user to vary the types of analysis, nor do they offer sets of metrics to fully understand the behavior of the model.

ODIN Framework aims at generalizing and integrating into a unique solution the main approaches to error diagnosis extending the standard evaluation metrics with custom properties and metrics, a wide range of off-the-shelf metrics, and analysis reports. Being ODIN a tool with great potential but with the limit of being python based and therefore accessible only to users capable of programming, we felt the need to compensate for this weakness by extending it with a dashboard that can quickly display the most common reports without any programming effort, still allowing the more experienced to refine the analysis as they want. The purpose of the presented work is to introduce ODIN Web, a web-based application capable of compensating for the lack of a complete tool, with an intuitive interface, easy to install, and which supports users in analyzing the results of their models without any programming effort. Finally, to demonstrate the utility and effectiveness of the tool, two types of use cases, applied to the ArtDL and PASCAL VOC 2007 datasets, are illustrated.

Keywords: Computer Vision, web-based application, diagnosis tool

Abstract in lingua italiana

Classificazione, rilevamento di oggetti e segmentazione delle istanze sono diventati campi importanti nell'area di ricerca sulla visione artificiale. Il loro scopo è riconoscere la presenza, o la presenza e la posizione, di una classe specifica all'interno di un'immagine in modo automatico utilizzando modelli neurali. Nel corso degli anni, però, le architetture di questi modelli sono diventate progressivamente complesse a causa della costante necessità di aumentarne le prestazioni. Ciò ha reso la loro valutazione sempre più ardua, portando a valutarli come "scatole nere" utilizzando metriche di valutazione standard necessarie per eseguire un'analisi equa sui modelli e comprenderne il comportamento per poterlo ottimizzare per un determinato set di dati o attività. In questo modo, diversi articoli descrivono lo sviluppo di framework volti a migliorare la diagnosi di modelli neurali profondi rispetto alle metriche di valutazione standard, implementando strumenti con interfacce grafiche che supportano l'utente durante tutte le analisi presentando i risultati in modo pulito e leggibile. Le interfacce che questi strumenti presentano sono spesso scarse e non consentono all'utente di variare i tipi di analisi, né offrono set di metriche per comprendere appieno il comportamento del modello. ODIN Framework mira a generalizzare e integrare in un'unica soluzione i principali approcci alla diagnosi degli errori estendendo le metriche di valutazione standard con proprietà e metriche personalizzate, un'ampia gamma di metriche e report di analisi. Essendo ODIN uno strumento dalle grandi potenzialità ma con il limite di essere python based e quindi accessibile solo agli utenti capaci di programmare, abbiamo sentito la necessità di sopperire a questa debolezza estendendolo con una dashboard in grado di visualizzare velocemente i report più comuni senza alcun sforzo di programmazione, consentendo comunque ai più esperti di affinare l'analisi a loro piacimento. Lo scopo di questo lavoro è quello di presentare ODIN Web, un'applicazione web-based in grado di sopperire alla mancanza di uno strumento completo, con un'interfaccia intuitiva, facile da installare, e che supporti gli utenti nell'analisi dei risultati dei propri modelli senza alcun sforzo di programmazione. Infine, per dimostrare l'utilità e l'efficacia dello strumento, vengono illustrate due tipologie di casi d'uso, applicati ai dataset ArtDL e PASCAL VOC 2007.

Parole chiave: Computer Vision, applicazione web-based, strumento di diagnosi

Contents

| | |
|--|------------|
| Abstract | i |
| Abstract in lingua italiana | iii |
| Contents | v |
| | |
| 1 Introduction | 1 |
| | |
| 2 Related Work | 5 |
| 2.1 Computer Vision | 5 |
| 2.2 Black-box error diagnosis | 8 |
| 2.3 Annotation tools for ML datasets | 12 |
| 2.3.1 Data Annotation | 12 |
| 2.3.2 Annotation Tools | 14 |
| 2.4 User interface for dashboards | 19 |
| | |
| 3 An Interactive Dashboard for ODIN | 23 |
| 3.1 Requirements | 23 |
| 3.1.1 Dataset use cases | 24 |
| 3.1.2 Annotator use cases | 26 |
| 3.1.3 Analyzer use cases | 28 |
| 3.2 System Architecture | 29 |
| 3.2.1 Technologies Involved | 32 |
| 3.2.2 Back-End | 35 |
| 3.2.3 Front-End | 42 |
| | |
| 4 ODIN Web In Action | 61 |
| 4.1 ArtDL | 61 |
| 4.2 PASCAL VOC 2007 | 67 |

| | |
|--------------------------------------|-----------|
| 5 Conclusions and Future work | 71 |
| Bibliography | 73 |
| List of Figures | 79 |
| List of Tables | 81 |
| Ringraziamenti | 83 |

1 | Introduction

In Machine Learning, the vast availability of data sets and open challenges have made it possible to create the essential foundations for the progress. In this vast environment it is possible to identify a branch called Deep learning, which is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example.

It allows computational models, that are composed of multiple processing layers, to learn representations of data with multiple levels of abstraction.

These methods have significantly improved the state of the art in many fields of analysis such as medical image analysis[29], acoustic recognition of the road surface[44], finance[24], regarding safety and construction of portfolios, and so on.

It has demonstrated incredible success in a variety of applications domain in the past few years, and with some new applications, it continues to produce new possibilities.

Its main uses are in *Image Classification*, whose goal is to identify the categories of objects present in a given image, *Object Detection*, whose aim is not only to identify the categories of objects inside an image but also their location using bounding boxes and *Instance Segmentation*, that requires to identify the objects and localize them in a finer way providing a pixel-level segmentation mask for each object instance. These tasks are obtained thanks to deep neural models which are constantly optimized based on their performances with a bunch of data. Performance analysis on these architectures can be achieved in two ways:

- **Opening the box** with model interpretation techniques those aim to break down a complex model into smaller areas of interest that are processed sequentially, and in this way try to asses the relationship between input, the inner layers and the output.
- An alternative approach is to treat the model as a **black box** enriching the input samples with labels, or extra annotations, useful for then model response but that they do not impact on its training phase. Such performance-driven meta-annotations enable the computation of tasks and data set specific metrics whereby scientists can identify the features of the input responsible of prediction errors and

focus their model improvement efforts.

Over the years many error diagnosis tools have made their way with the aim of expanding the standard evaluation methods based on metrics by providing visual support through graphs thanks to which it was possible to understand the behavior of the model and exploit this knowledge in order to increase its performance. However, nowadays many tools do not offer a complete sector that supports the analysis, in the first place, but which also offers to the user the possibility of having a complete view of the data used as input in the model and the possibility of interacting with it.

In fact, the current state of the art does not provide a complete tool that allows to create a dataset from scratch and, at the same time, once populated, to be able to analyze the performance of your models for that particular dataset.

The aim of the presented work is to make up for the lack of a complete tool by presenting ODIN Web, a web app that aims to integrate in a single tool all both the management of the data sets and the error diagnosis for neural models.

ODIN Web offers a 360-degree analysis of the latter by exploiting analyzes that vary from the analysis of the simple dataset created, up to that of the predictions of the model by exploiting multiple metrics for analyzes that touch all the crucial points of the model's performance.

This work aims to extend ODIN [47], a complete framework developed entirely in python that addressed Image Classification, Instance Segmentation, Object Detection tasks and the possibility to enrich the training set, and export it into a web based solution. This work stems from the need to make ODIN accessible to everyone, including people who are not particularly familiar with python. This has led us to want to implement an intuitive web-based application that does not require any programming skills but at the same time allows any user to analyze their dataset and the predictions of their models by exploiting all the analyzes that ODIN[47] offers. For this reason we have created a tool with:

- A visual interface that guide the user into the set preparation by labeling and annotating it with categories and meta-annotations constantly added in real time.
- A complete interface that allows the user to inspect and analyze the results of the model in order to improve performance through metrics and graphical results, And in addition the possibility of comparing the model under analysis with the results obtained from other models to which the same inputs were given in analysis.
- The ability to install the program locally through a simple installation guide with the open source code or through the docker by installing its image

In the course of this document we will analyze ODIN Web in detail, from why there was a need to implement a tool with these features, up to a detailed analysis of each of its individual components both as regards the client side and the server side. This document is structured as follows:

- *Chapter 2* – an overview of the computer vision is provided, the tools currently available are subsequently presented both as regards the annotation of the dataset and for the analysis phase and, finally, the guidelines used for the development of the ODIN Web dashboard are introduced.
- *Chapter 3* – the tool is analyzed in its entirety: starting with the description of the technologies used, we then move on to the analysis of all the implementation and structural aspects of the server side, with the description of each module, and of the client side in which the developed interface and offered analyzes.
- *Chapter 4* – ODIN Web use cases are illustrated for the analyzer applied to ArtDL and PASCAL VOC 2007 datasets.
- *Chapter 5* – conclusions are drawn and future works listed.

2 | Related Work

2.1. Computer Vision

Computer vision is a field of artificial intelligence that aims to analyze images and videos with the aim of extracting understandable information. Computer vision is also often associated with human vision as it tries to mimic the sight of humans through machines to interpret spatial data. Obviously it is a complex challenge but over the years researchers have made great strides by improving themselves in various computer vision tasks, namely:

- *Image Classification*: is the task of recognizing classes of objects within images. This task can be divided into sub-tasks that depend on the number of classes and objects within the images. Starting from the binary image classification, it is addressed to all those types of problems that go to have an image as input in which to detect a single class out of a predefined set of two classes (for example whether an image represents a certain class or not) . In multi-class (single-label) problems, on the contrary, the image is associated with a class that belongs to a set of more than two classes. A final task for classifying images is called multi-label, that is, unlike the multi-class single label, the input image can be associated with an indefinite number of classes on a set of classes greater than two. It is often used to recognize various objects in images such as the Person class in the Figure 2.1.
- *Object Detection*: visualized in Figure 2.2 this type of task focuses more on identifying the position of the object within the image rather than just labeling it. This kind of task makes use of bounding boxes to locate and highlight the position of the object.
- *Instance Segmentation*[21]: has come to be one of the relatively important, complex and challenging areas in machine vision research. Aimed at predicting the object class-label and the pixel-specific object instance-mask, as shown in 2.4 it localizes different classes of object instances present in various images. Instance segmentation aims to help largely robotics, autonomous driving, surveillance, etc.

- *Semantic Segmentation*[18]: very close to the concept of instance segmentation, with the simple difference that instead of identifying the object and its area pixel by pixel within the image, it labels each pixel of the image not with the aim of identifying an object in particular but to create a single cluster of the same class. An example is provided in Figure 2.3.
- *Pose Estimation*[37]: Human pose estimation is the process of determining through an image, the position of the various parts of the body of an individual such as head, shoulders, elbows, hands etc. It has many applications in contexts such as sports, character animation, clinical analysis and everything related to the recognition of actions. as we can see in Figure 2.5 each person is identified by highlighting their joints.
- *Object Tracking*[51]: which is similar to object detection with the difference that the goal is to trace the movement of the object inside a sequence of frame (usually a video).
- *Action Detection*[25]: aims to assign a label to the action presented in a video.

Over the years these tasks have been approached through neural models, also known as CNN (Convolutional Neural Network)[50]. A CNN is a particular type of architecture that is based on different building blocks, such as:

- *Convolutional Layer* – Layer in which filters are applied to the original image or feature maps. They are characterized by an activation function that filters the inputs by converting a 3D cube, previously set of feature maps to a 3D cube, with one 2D map per filter.
- *Pooling Layers* – They tend to be exploited to reduce the size of the network. They are similar to convolutional layers, with the difference that they perform specific functions, including max-pooling (maximum value per filter region), or average pooling (average value in the filter region).
- *Fully connected Layers* – Inserted before CNN output to flatten the result before classification

CNN consists of an alternation of stacks of different convolutional layers followed by pooling layers all with different parameters as in Figure 2.6. Over the years, the need to increase the performance of their models has led to a continuous increase in the size of these structures up to the point of making them difficult to analyze and grasp the weak points with certain types of data. Therefore, we reached the point where the analysis where "the box is opened" was no longer sufficient as their demanding interpretation.



Person

Figure 2.1: Image Classification

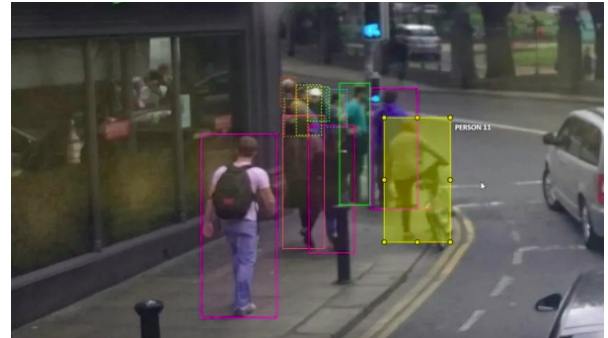


Figure 2.2: Object Detection



Figure 2.3: Semantic Segmentation

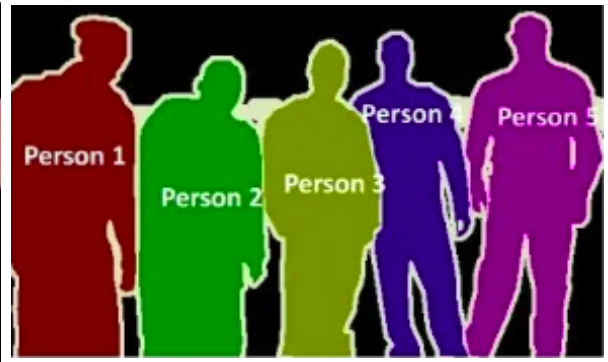


Figure 2.4: Instance Segmentation



Figure 2.5: Pose Estimation, source: [36]

This is the reason why the frequency of use of the tools, which exploit black-box analysis, by directly analyzing the model's performance under certain types of data has increased over the years.

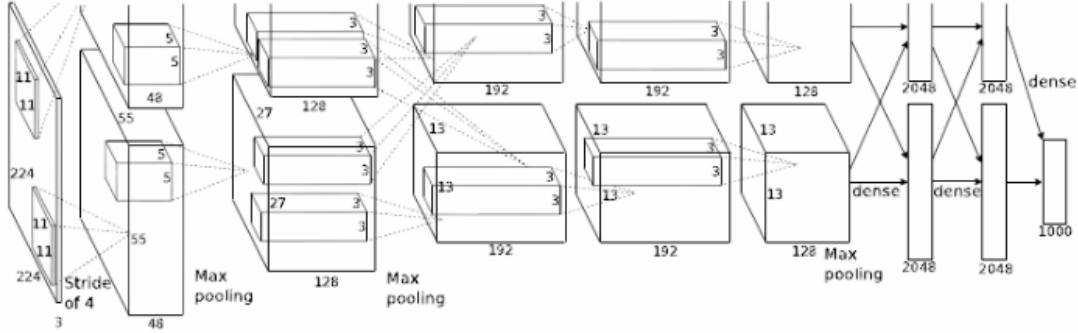


Figure 2.6: Architecture of Krizhevsky et al.'s DCNN [31]

2.2. Black-box error diagnosis

To be able to fulfill the tasks that computer vision offers us, it is necessary to use deep neural models, and, in order to understand the benefits of a given model, it is necessary to analyze and catch the characteristics of its architecture. As mentioned earlier this performance analysis can be performed in two ways: the first approach is to 'open the box' with the aim of finding a link between the input, the various internal layers of the model, its nodes, and the output provided by it. The other, that is what we will discuss in this document and implemented by ODIN, is the one called 'black-box' and consists in associating extra annotations to the input, which are not exploited in the training phase, but capable to make us better understand the model output by analyzing which meta-annotations had the greatest impact on model errors.

Many tools, nowadays, implement neural model analysis with a black-box approach. In this section we will analyze some of the most important tools and, if they are provided, a quick description of the interface they present. They will be described and compared based on the following characteristics:

- *Task*: Type of task supported by the analyzes provided by the tool. The types of tasks reported are: Classification (CL), Object Detection (OD), Instance Segmentation (IS), Semantic Segmentation (SS), Action Detection (AD), and Bias Detection (BD).
- *Media types*: Each tool can support one or more types of media, including: image, videos and Generic which comprises text, graph and etc.
- *Dataset independence*: possibility to process different data sets.
- *Code*: In case the tool is open source, the link to the code will be available.

- *Interface*: if the tool provides a web-based/stand-alone interface or not.

| Reference | Year | Task | Media | Data set independence | Code | Interface |
|---------------------|------|----------------|---------|-----------------------|------|-----------|
| Hoiem et al.[26] | 2012 | OD | Image | Yes | Link | No |
| COCO API[32] | 2014 | OD, IS, PE | Image | Yes | Link | No |
| Alsallakh et al.[3] | 2014 | CL | Generic | Yes | - | Yes |
| ModelTracker[5] | 2015 | CL | Generic | Yes | - | Yes |
| Prospector[30] | 2016 | CL | Generic | Yes | - | Yes |
| Squares[40] | 2016 | CL | Images | Yes | - | Yes |
| Manifold[54] | 2018 | CL | Generic | No | Link | Yes |
| DETAD[4] | 2018 | AD | Video | Yes | Link | No |
| Ye et al.[49] | 2019 | BD | Generic | Yes | - | Yes |
| What If Tool[48] | 2019 | CL, BD | Generic | Yes | Link | Yes |
| TIDE[6] | 2020 | OD, IS | Image | Yes | Link | No |
| TF-GraF[52] | 2020 | OD | Image | Yes | Link | Yes |
| Boxer[16] | 2020 | CL | Generic | Yes | Link | Yes |
| OpenVino[12] | 2020 | CL, OD, SS, IS | Image | Yes | Link | Yes |
| GNNVis[27] | 2020 | CL | Graph | Yes | - | Yes |

Table 2.1: Tools list ordered by ascending publication year. Adapted from [15]

Tools Description

Table 2.1 presents the identified tools. For each tool, the table reports the name, the publication year, the task, the target media, if it can be used with different data set, and the link to the code, if the tool is open-source.

- *Hoiem et al.* [26] is a pioneer work regarding the analysis of ML models with a black-box approach. The author has shown how there is a correlation between the characteristics of the object, such as size, occlusion, aspect ratio, truncation and visibility of parts, and how they affect the performance of the model. The tool is developed entirely in Matlab and does not provide a graphical interface. Before analyzing the dataset and the predictions associated with it, the user needs to manually modify the various parameters within the files. Furthermore, all analyzes are performed through commands executed in Matlab.
- *COCO API*[32] in this framework it is possible to find a step towards using the properties of the object in which the calculation of the mean Average Precision

(mAP) is different based on the size of the object. It includes APIs developed with the following languages: Lua, Python, and Matlab. Specifically, the PythonAPI can be run on any interpreter in order to take advantage of all the functions of COCO. In fact, it allows the user to analyze the dataset and its annotations via a viewer, but also to analyze the performance of the model whose results will be returned in textual format within the Notebook. Coco has no web interface.

- *Alsallakh et Al.* [3] authors propose a new graphical analysis that arranges different classes based on a radial layout and use histograms to show the statistics of the true/false positive/negative associated with each class and their prediction confidence. However, the technique lacks the ability to support effective comparison of multiple models.
- *ModelTracker*[5] provides a stand-alone tool called Ice for analyzing the output of a classification mixing traditional metrics summarization and novel types of interactive visualizations. This tool is aimed at experts, as the interface that describes the overall performance of the model with direct data can be difficult to interpret for a novice.
- *Prospector*[30] a web-based interface used for understanding how a specific feature contributes to the prediction by adjusting the feature value and examine the corresponding change of the predicted result.
- *Squares* [40] proposed by Ren et al., provides a stand-alone tool that offers a performance analysis system based on histograms to expose prediction scores in a multi-class classification task and allows the user to investigate multiple models by comparing multiple graphs. Although innovative, the concept of comparison is based solely on comparing two histograms without creating a real visual indicator of how the two models behave in front of the same instance. The technique lacks the ability to characterize the model diversity at the instance level.
- *Manifold*[54] this tool is developed entirely in the React language and is the first example of a tool in which performances were compared on multiple models. In other words, the outputs of several models were compared trying to understand the strengths and try to lead to a biased cognition based on their relation to the underlying data.
- *DETAD* [4] developed entirely in python, it does not provide a stand-alone interface but relies on a Conda environment to run. The focus of this work is on the localization of temporal actions in videos. The diagnosis method and tool allow

False-Positive and False-Negative analysis and the estimation of the sensitivity of mAP-based metrics to six action characteristics: context size, context distance, agreement, coverage, length, and the number of instances.

- *Ye et al.*[49] a stand-alone tool focused on the analysis of the dataset and the distribution of classes within it. Provides a simple and effective interface supported by a wide range of filters to analyze the components of the dataset on different degrees of granularity. The authors present the results obtained with label noise and a visualization work for data quality management with some similarities to interactive labeling.
- *What If Tool*[48] a web-based tool that leverages a visual interface to help understand class distributions in the dataset and model output. It can also be used on various notebooks. A user has the possibility to manage different characteristics of the input data set in order to analyze how these changes affect the predictions of the model. Furthermore, the tool provides a fairness analysis and different strategies for optimization.
- *TIDE*[6]: a framework developed entirely in python but it does not develop any type of interface. It is used for analyzing the sources of error in object detection and instance segmentation algorithms. Error analysis is performed in such a way that the impact of each error on the overall performance is handled in an isolated way. It also allows the comparison of several models at the same time.
- *TF-GraF* [52] The authors provide a user-friendly graphical framework for object detection built on TensorFlow Object Detection API[2]. It allows everyone, even without any knowledge of DL frameworks, to design, train, and evaluate models without coding efforts.
- *Boxer*[16] web-based application developed entirely in Vue and TypeScript. It provides a customizable interface in order to view only the analyzes of interest. The graphs present in the analyzes are minimal and sometimes without the appropriate filters in order to have a partial view of the analysis.
- *OpenVino*[12] a stand-alone tool that provides 360-degree support for analyzing the results of a model. It has an intuitive but at the same time functional and complete interface for the analysis of predictions. It focuses on model analysis, optimization and deployment. Novel visualizations and evaluations are introduced to support hardware optimization and model calibration. Computational graph visualization allows developers to investigate the runtime representation of a model. Calibration

techniques enable the acceleration of model performance while decreasing memory impact (keeping in consideration accuracy) and deployment to a target system.

- *GNNVis*[27] The authors presented a new approach for the analysis of graphical neural networks (GNNs) providing a performance analysis by analyzing common error patterns in a group of nodes that directly influenced the model and also allowing a comprehensive analysis of the graph topology. It is a web-based tool developed in python and flask, for the back-end part, and with React, TypeScript, and D3 for the front-end application. It features a clean and intuitive interface that guides the user throughout the analysis.

As it could be deduced from the analysis, there are very few tools that allow having an interactive interface and at the same time offer the user the complete customization of the data and the analyzes performed on them. Furthermore, each tool focuses mainly on a certain type of task, completely excluding the other. This is why we felt the need to develop ODIN Web. That is to provide a tool with a user-friendly and attractive interface but at the same time equipped with all the necessary features to be able to better analyze your data.

2.3. Annotation tools for ML datasets

2.3.1. Data Annotation

Training models is one of the key parts when it comes to deep learning. Often the model's performance after being trained largely depends on the quality and breadth of data that has been provided to it. For that reason, we can say that the preparation of the training set is crucial to obtain excellent performance from the model and, therefore, it must be supported by a tool that allows a quick, but at the same time efficient, annotation. Obviously, there are a huge amount of annotators on the market, each of which specializes in annotating certain types of data. These data can range from simple text files, to images up to multimedia files. But, before introducing them, it is important to define what a data annotation is and how it is divided into categories according to the purpose of the data set.

A *Data Annotation* is the process of making text, audio, or images of interest understandable to machines through labels. As already mentioned, the preparation of the data set to be dared to feed the model is an essential step to achieve the goal that has been set for the model as: **the higher the quality of annotated data fed to the training model, the higher the quality of the output**. This is the reason why many developers prefer

human labor for the annotation process as the process could also be automated using a machine but the capacity of recognition and entity-relationship association that belong to the human being are preferred in case of complex or sensitive data. After this brief introduction to data annotation we can define the types of annotation that we will need later to describe the Annotation tools. As mentioned previously, the types of annotation depend primarily on the resource that must be annotated:

- **Text Annotation:** since the capacity for language is a subjective quality of the person, a machine is not able to grasp the emotional characteristics of the person which can be expressed through a totally subjective text with new trends, slang, humor and different types of communication. For these reasons text annotation can be categorized as: *Sentiment Annotation*, depicted in Figure 2.7, which evaluates attitudes and emotions behind a text labeling it as Positive, Negative or Neutral. *Text or Semantic Classification* is the analysis and categorization of a text under a list of predefined labels, often used to group text files under a predetermined heading shown in the 2.8, and finally the *Entity Annotation* which allows you to identify keywords in the text (entities) with a close relationship between them (Figure 2.9).

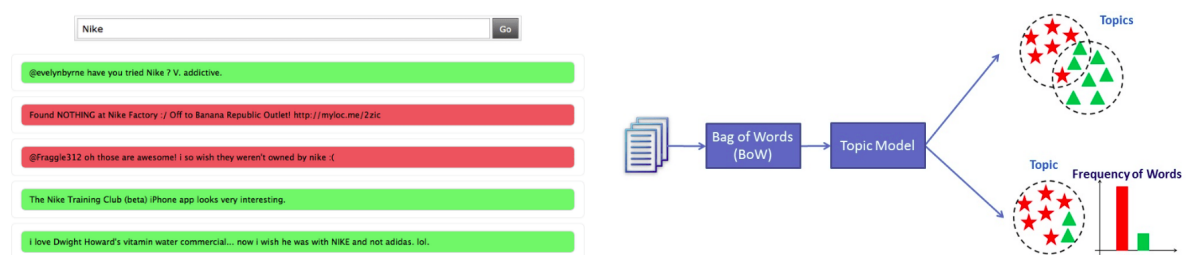


Figure 2.7: Sentiment Annotation. Source [41]

Figure 2.8: Semantic Classification. Source: Link



Figure 2.9: Entity Annotation. Source [39]

- **Image Annotation:** one of the key aspects of computer vision, is the process of classifying the images through labels that your model will subsequently analyze and recognize in the future. In computer vision these labels are also called *classes*, expanding the simple image with extra information about it. It is possible to identify several task of image annotation, the most used are: *Image Classification*, that

requires images to have a *single label* or *multi labels* to identify the entire image, aims to build a model capable of recognizing the presence of one or more elements within the images in the data test. *Object Detection* or *Recognition* adds a further step to the simple concept of classification by giving the developer the possibility to identify which class a certain object belongs to, as happens for classification, but also to delimit its position in the image by drawing a bounding box, allowing to know the exact position and number of occurrences of the class within the image. An evolution that aims to identify the exact position of the element is called *Image segmentation* which divides the image through lines, points, etc. for example, a pixel-level analysis where each pixel is assigned to a certain class. Obviously the programmer's task is not to label every single pixel but to delimit as precisely as possible the element in the image through segments. This type of annotation can in turn be divided into two sub-categories that focus either on identifying the presence of the category by approximating it (*Semantic Segmentation*[17]) or on identifying the exact number of elements by dividing them one by one (*Instance Segmentation*[20]).

- **Video Annotation:** video annotation is defined as the task of labeling and tagging video frames to train the model later. The difference between simple image annotation is that the video annotation process is done frame-by-frame. As with image annotation, video annotation can be used for the following types of tasks: *Object detection* and *Object localization*, as for Image annotation, and in addition a bunch of new tasks which are *Object Tracking*, in which the aim is to annotate a series of frames being careful to ensure consistent identifiers for each unique object in the sequence of images, and *Activity Tracking*, similar to object tracking, consists in navigating through human activity also contributes to a better perception of the environment and helps prevent accidents, even if those are initiated by unpredictable pedestrian behavior.
- **Audio Annotation:** a subset of data annotations that involves the classification of audio components from people, animals, the environment, instruments, and so on.

2.3.2. Annotation Tools

Annotation tools are an important resource for developing custom models for a variety of purposes. The choice of the annotator in the initial stages of building the data set, is therefore, crucial. The researcher must be able to identify the objective of the model (type of annotation and consequently on which task type the model will be based), the

development environment and the effectiveness of the annotator that is going to be used. Many annotation tools, also listed in this work[34], have been developed over the years and in this section, the most used will be cited and analyzed by comparing them based on the following criteria:

- Technical Criteria, with which technical aspects of the software will be treated, are:
 - *Technologies involved* – with which the tool is built.
 - *Installation Type* – this entry will be divided into two categories: web-based (running on a server) and stand-alone(as a program that can works even offline)
 - *Source Code* – with this criterion the possibility of analyzing the source code of the tool is taken into account.
 - *Online Demo* – categorized as *yes* or *no* in case is possible to test the tool with an online demo that briefly sum up the main features
 - *Free* – whether the software is free or paid
- Functional Criteria, with which functionalities of the tool are analyzed, are:
 - *Annotations Format* – in which format annotations are saved: XML, JSON, TXT, other
 - *Type of Annotations* – based on the annotation types mentioned in the previous section they will be labeled as SA (Sentiment Annotation), SC (Semantic Classification), EA (Entity Annotation) for what concern the Text Annotation type, and IC (Image Classification), OD (Object Detection), SS (Semantic Segmentation) and IS (Instance Segmentation) for Image Annotations.
 - *Pre-Annotation support* – if it has the support to already created annotations or it gives the possibility to load it.

Tools description

- *PhotoStuff* [22] is a platform independent, image annotation tool which uses an ontology to provide the expressiveness required to assert the contents of an image, as well information about the image.
- *Light Tag* [38]: Text annotation tool that allows you to take advantage of all the annotation tasks: Semantic Classification, Sentiment Annotation and Entity Annotation with a clean and intuitive interface. It allows you to manage the dataset in a

| Tool | Reference | Year | Doc. |
|--------------|-----------|------|------|
| PhotoStuff | [22] | 2005 | - |
| LightTag | [38] | 2009 | Link |
| Anafora | [11] | 2013 | Link |
| TagTog | [10] | 2014 | Link |
| VIA | [13] | 2016 | Link |
| VoTT | [1] | 2017 | Link |
| ImgLab | [19] | 2018 | Link |
| CVAT | [43] | 2018 | Link |
| Label Studio | [45] | 2019 | Link |
| COCO | [7] | 2019 | Link |

Table 2.2: Annotators list ordered by ascending publication year

complete way from creation to its enrichment with meta-data. It also permits you to have a visual feedback regarding entity annotations through trees.

- *Anafora* [11]: is an annotator used to annotate documents based on previous works, such as Protege¹ and Knowtator [35]. It has been designed to read AnaforaXML only, a format defined by the developers themselves, who provide a script that allows you to convert files to the required format. Anafora allows the user to annotate text files with an entity annotations format which divides annotations into two types: *Entity* and *Relation*. An Entity annotation associates a certain span in the text with a type and list of properties. Relation annotations specify a relationship between multiple Entities.
- *TagTog*[10]: one of the best text annotation tools. Tagtog is a text annotation tool that can be used to annotate text either automatically or manually. It supports native PDF annotation and includes pre-trained NER templates for automatic text annotation. In addition to the Tagtog tool, the company provides an annotation service through a network of skilled workers in the required sector who will annotate the texts under commission.
- *VGG Image Annotator (VIA)* [13] is a stand alone annotation tool for images, audio and video developed to work offline without the need of installation or setup, running on a web browser. The VIA software allows human annotators to define and describe spatial regions in images or video frames, and temporal segments in audio or video. It also allows to export annotations in a plain text data formats, such as JSON and CSV, and, therefore, are amenable to further processing by other software tools.

¹<https://protege.stanford.edu/>

- *VoTT*[1] is one of the most used tools with *ImgLab*, it allows the annotation of images for both object detection and Instance segmentation. The tool allows both to label images with classes preloaded from a .vott file or with new classes added during the annotation phase. VoTT also allows the annotation of video files frame by frame, with an interface full of commands and shortcuts to make the annotation faster and more functional. Finally, export annotations in different formats including: Azure Custom Vision Service, Microsoft, Cognitive Toolkit (CNTK)[42], TensorFlow[2] (Pascal VOC and TFRecords), VoTT (generic JSON schema), Comma Separated Values (CSV) is granted.
- *ImgLab*[19] is the most used image annotator at the moment, based entirely on web but it also offers the possibility to be installed locally. One of its greatest advantages is the support of multiple file formats including: XML, Pascal VOC, COCO. It allows multiple tasks including object detection, Semantic and Instance segmentation with a user friendly interface and rich documentation.
- *Computer Vision Annotation Tool (CVAT)* [43]: is a classic image annotator that allows to performs Object Detection, Instance Segmentation and image classification, using powerful features those including: interpolation of bounding boxes between key frames, automatic annotation using TensorFlow OD API [2] and deep learning models in Intel OpenVINO[12] IR format and shortcuts for most of critical actions. It also offers UX and UI optimized for computer vision annotation tasks.
- *Label Studio*[45]: is a complete 360-degree tool, covering all aspects of data annotation, as it offers services for annotating texts, with classification, entity annotation, question answering and sentiment analysis, images and videos with image classification, object detection and semantic segmentation. The installation of the tool could be performed locally or deploy it in a cloud instance and has a unique configuration setup called Labeling Config where you can design your own customized UI.
- *COCO Annotator* [7] provides many distinct features including the ability to label an image segment (or part of a segment), track object instances, labeling objects with disconnected visible parts, efficiently storing and export annotations in the well-known COCO format. It also exploits advanced tools such as DEXTR and MaskRCNN and Magic Wand to execute automatic annotation.

Tools Comparison

In this sections, a comparison of the Technical and Functional criteria of the tools listed in 2.2 and described in Section 2.3.2 are discussed. In table 2.3 it is possible to note

| Reference | Techs. | Installation | Code | Demo | Free to charge |
|-----------|--|--------------|---------------|------|----------------|
| [22] | Java | Web | Not Available | No | Yes |
| [38] | - | Web | Not Available | No | No |
| [11] | Apache, Django | Web | Available | Yes | Yes |
| [10] | - | Web | Not Available | No | No |
| [13] | - | Stand-Alone | Available | Yes | Yes |
| [1] | React + Redux | Web | Available | No | Yes |
| [19] | jQuery, Bootstrap, Riot.js, SVG.js | Web | Available | Yes | Yes |
| [43] | - | Web | Available | Yes | Yes |
| [45] | Nginx, PostgreSQL | Stand-Alone | Available | Yes | Yes |
| [7] | Flask, MongoDB, Mongo-Engine, Vue, Axios, PaperJS, Bootstrap | Web | Available | Yes | Yes |

Table 2.3: Technical criteria of the annotation tool

that most of the annotation tools are web-based, to avoid, probably, the installation of the software locally and to have the possibility to exploit the computational power of the server to perform more complex tasks needed to prepare the data set. In fact, in table 2.4 it is possible to see how most of the tools based on image annotation prefer an approach based on instance segmentation while not neglecting object detection. In most cases, especially with regard to recently developed tools, the annotation process is not completely left in the hands of the user, but instead the researcher is the person who completes the annotations of images not recognized by the annotation system. An example of a tool that takes advantage of this type of approach is CVAT [43]. In fact, it has a tool called automation instruments which copy and propagate objects such as bounding boxes, segments, etc. so as not to leave the user with the task of having to

| Reference | Annotation Type | Annotation Format | Pre-Annotation |
|-----------|------------------------|-------------------|----------------|
| [22] | OD | RDF/XML | Yes |
| [38] | SA, SC, EA | CSV, JSON | Yes |
| [11] | EA | AnaforaXML | No |
| [10] | SC, EA | JSON | Yes |
| [13] | OD, IS, SS | CSV, JSON | Yes |
| [1] | OD, IS | JSON | Yes |
| [19] | OD, SS, IS | XML,JSON | Yes |
| [43] | OD, IS, SS | CSV, JSON, other | Yes |
| [45] | SA, SC, EA, OD, IS, SS | CSV, JSON, other | Yes |
| [7] | OD, IS, SS | JSON | Yes |

Table 2.4: Functional criteria of the annotation tool

select all the objects labeled by the element but to execute it automatically.

Over the years, standard formats have also been set for saving on file (also called ground truth) of annotations. In fact, as we can see from the table, most of the files, both input and output, are generated as JSON (Javascript Object Notation) files with MS COCO as standard format. As for the annotations saved in XML it is assumed a standard format dictated by Pascal VOC. Finally, Table 2.4 shows that the most developed task type is certainly that of Entity Annotation (as regards the tools considered).

2.4. User interface for dashboards

The continuous increase in data and consequently the analyzes carried out on them has led over the years to the increasingly looming need for the creation of increasingly effective, simple and accessible dashboards. In this section we will discuss all the best practices²³ that should be respected during the implementation and that have been useful for the implementation of ODIN Web. To develop the ODIN Web dashboard it was necessary to compare different tools, both for the annotator and for the analyzer, to find the strengths and shortcomings that made the tool itself and the data easy to analyze and usable.

- *Audience*: the priority aspect that must be addressed when developing a dashboard is: the user target. ODIN Web was born with the aim of making ODIN[47] known to users who were not familiar with the python programming language. Furthermore, the idea was to create a user friendly interface capable of being professional but at

²<https://www.sisense.com/blog/4-design-principles-creating-better-dashboards/>

³<https://www.toptal.com/designers/data-visualization/dashboard-design-best-practices>

the same time understandable even by a novice user

- *Color Design*: One of the fundamental principles for dashboard design. Colors are a great way to grab the user's attention. As many application design guides recommend, it is best to choose a pair, maximum of three, minimal colors for the application design and a couple of colors to help describe tasks such as adding (blue) or removing (red) data. Furthermore, the inclusion of an intuitive color scale that defines a range of data can give the user a clear and quick idea of what the data is saying.
- *Minimalist*: Another good practice for developing the design of a dashboard is to keep on the screen only and only the information that is useful at the moment. From a design point of view "anything that doesn't convey useful information should be removed". In ODIN Web, the creation of the graphs has been structured in such a way that every single element within them is useful for the purposes of interpreting the data.
- *Consistency*: one of the most important features a dashboard must have is consistency. Having a consistent layout impresses the user with an idea of cleanliness and order. The information and layouts of the various components, including the page structure, must be somewhat similar and in the same positions. This concept applies to the dashboard itself but also to the graphs contained in it. An example could be the labels, the legend, the axes, the chart style and the tools must always appear in the same position and with the same layout.
- *Context*: especially in a dashboard where there are only graphs, the user must have a clear idea of what data he is going to analyze and what their meanings are. In ODIN Web we have solved this problem by inserting, first of all, self-explanatory titles for each card containing a chart and in addition a tooltip, thus maintaining the concept of minimality mentioned above, where a description of the chart and a link to the documentation are provided. Furthermore, thanks to the library used to draw the graphs, labels are generated every time a data point is hovered, adding an additional layer of information.
- *5-seconds rule*: this is a classic rule that can be found on the net and consists in the concept of speed in the rediscovery of information. A user should be able to find any information they need in less than 5 seconds. This problem can be solved in two ways: the first is to remove unnecessary graphs while the second, exploited in ODIN Web, is to organize the dashboard into sections of different graphs that can be easily localized through keywords.

- *Performance*: This is the last, but perhaps most important, rule for implementing a good dashboard. A dashboard may also have satisfied all the previous principles but if the performance in obtaining, processing and displaying data is bad the user will suffer. For this reason we have tested ODIN Web both on small amounts of data but also and above all on large datasets trying to optimize data processing by transferring only and only the required data, thus not weighing down its performance.

In order to implement the ODIN Web interface, many tools that offer dashboards have been taken into consideration both for what concerns the layout and for the functionalities present, or in some cases missing. The main tools are COCO and Label Studio, for what concern the annotator component, and What If Tool and OpenVino for the analyzer part. Obviously, also with regard to the organization of the code, some best practices have been followed which will be covered more in Chapter 3.

3 | An Interactive Dashboard for ODIN

3.1. Requirements

For what concern this section, the non-functional and functional requirements are taken in consideration. This work aims to develop a web interface capable of exploiting all the features of the previous work [46], [47] and subsequently [53], implementing an application capable of relieving the user of any programming effort with the aim of:

- providing the user with a simple and intuitive tool in use. For this reason a README guides the user in the installation process: from cloning the repository, to installing the required packages, and starting the dashboard. To relieve the user to those steps, we also provide ODIN Web as a Docker¹ container in order to benefit from the features, listed in the following sections, that dockerizing the app offers us. The decision to offer ODIN Web also in dockerized format stems from the aim of making a tool that is simple to use but also to be installed, ensure the possibility to make it accessible in different ways.
- support the user throughout the management of the data set
- support the user in the comprehension of the results provided by the model using properties, i.e., annotations that do not contribute to model training but can be exploited for understanding performance

ODIN Web supports multiple computer vision tasks which, depending on the task, apply slight modifications to the observations. Tasks are divided into two macro categories: classification and localization.

The classification task refer to the problems where the model needs to identify one or more categories present in the observations. The classification tasks supported for this

¹<https://www.docker.com/>

macro-category are:

- *Single-Label Classification*: input is classified in only one class, but with a class set that contains more than two categories.
- *Multi-Label Classification*: multiple categories are associated to the observations those can represent different classes. In this case the model can output more than one category
- *Binary Classification*: refers to all the problems that has the need to recognize only one class inside the input image considering tow categories only.

The localization task, instead, is supported through:

- *Object Detection*: refers to the task of recognizing an object and find its location by surrounding it with a bounding box.
- *Instance Segmentation*: refers to the task of recognizing an object and find its location by providing a pixel-level segmentation mask.

Both classification and localization are supported by the ground truth file and predictions which are expressed in MS COCO format. For what concern the functional requirements that the application must meet, a bunch of user cases from the creation of the data set, to the creation of the observations, up to the analysis of the results are provided. All use cases will be organized as follows:

- *Use Case*: the goal that the user wants to achieve through the application.
- *Preconditions*: statements, or truths, about what must take place before and after the use case.
- *Basic Flow*: the series of steps required to achieve the goal.
- *Alternative Flow*: is a variation of the basic flow scenario
- *Conclusion*: the final output of the application.

In all the cases the only actor of the system is the User which interact with ODIN Web in order to accomplish 3 macro categories of tasks:

3.1.1. Dataset use cases

Create a dataset

An ODIN Web user wants to create a new dataset with which to train, and subsequently analyze, their model

- **Precondition:** user must be logged in to ODIN Web.
- **Basic Flow:** The user is on the dataset selection page. Go and press the button to add a new dataset. The user must enter the name of the new dataset, a list of classes and specify the task type. In addition, ODIN Web offers the possibility to insert the properties with the relative values and in case already exists, the path that directs to the folder of the images to be noted. In the event that one of the mandatory fields has not been satisfied, the user will be notified via toast, otherwise he will be redirected to the main page of the annotator where he can start annotating the images.

Import a dataset

An ODIN Web user wants to import one of him/her dataset in order to perform some analysis on the results of its model

- **Precondition:** user must be logged in to ODIN Web.
- **Basic Flow:** The user is on the dataset selection page. Press the button to add a new dataset when you are on the creation page press the link "Already have a dataset? Import". The user enters the name with which the dataset will be displayed in ODIN Web and specifies all required fields such as: path to the ground truth file and the task type. If he wants to carry out the analysis of the model results, in addition to those of the dataset, he can enter the path to the predictions also assigning an identification name for that model. In the event that one of the mandatory fields has not been satisfied, the user will be notified via toast, otherwise he will be redirected to the landing page in which can access to the annotator page or into the analyzer.
- **Alternative Flow:** The user is on the dataset selection page. Press the button to add a new dataset when you are on the creation page press the link "Already have a dataset? Import". The user wants to import a specific dataset already present in ODIN Web but assigning them different predictions. In this case the user chooses the dataset to import in the list of datasets present in ODIN Web on the left, in this way all the fields will be filled in automatically. The only thing the user has to enter is the name of the new dataset and the new path to the predictions folder. If the name does not already exist, the user is redirected to the landing page where he can annotate the imported dataset or analyze it together with the new predictions.

Edit settings of the created dataset

An ODIN Web user wants to take advantage of the function offered by the analyzer which allows to compare several models at the same time.

- **Precondition:** user must be logged in to ODIN Web and must have already entered the dataset to which he wants to add the model.
- **Basic Flow:** The user is on the dataset selection page. Choose the dataset to add the model to and press the edit pop-up button on the dataset card. It is redirected to the page where all the settings concerning the dataset are listed, presses the button to add a new model, assigns the name and path to the predictions. If the data entered is in the right format, the user is directed to the landing page where he can proceed with the analysis. Otherwise a toast with the error to be corrected will be displayed.

3.1.2. Annotator use cases

Add images

An ODIN Web user wants to add multiple images to the dataset in order to be able to annotate them.

- **Precondition:** the user must be logged in to ODIN Web and must have already entered the dataset in which he wants to add the images.
- **Basic Flow:** The user is on the dataset selection page. He chooses the dataset to annotate and, when he is on the landing page, he starts the annotator. The user opens the folder in which the current images are located and drags the new ones into it. Finally press the *Refresh* button which will load the new images and display them in the annotator.
- **Alternative Flow:** The user is on the dataset selection page. He chooses the dataset to annotate and, when he is on the landing page, he starts the annotator. The user goes into the settings and changes the path of the images to that of the folder with the updated images. When saved, it will be returned to the main page of the annotator.

Create an annotation

An ODIN Web user wants to annotate some images to be able to train his model later.

- **Precondition:** the user must be logged in to ODIN Web.

- **Basic Flow:** The user is on the dataset selection page. It follows the same steps described in the use case *Create Dataset*[3.1.1]. Once they have landed on the main page of the annotator, choose the image they want to annotate and, by clicking on it, the editor, in which is possible to annotate the image, will be open. In case of classification task the user will have the possibility to select the category/categories and the properties that belong to the image. In case of localization task, it will be able to create the bounding box / segmentation mask and apply the respective category and associated properties to it. Once the last annotation has been completed, the user can return to the home by pressing the save button which will update the ground truth file with the latest annotations.

Add a Category

The user has realized that many of his images contain a category not yet present in the ground truth file and wants to add it to update the annotations.

- **Precondition:** the user must be logged in to ODIN Web and must have already entered the dataset in which he wants to add the category.
- **Basic Flow:** the user is on the data set selection page. Navigate through the datasets and select the one you want to add the category to, after which start the annotator. By going to the annotator settings, the user will be able to enter the name of the category in the appropriate box. Once all the missing categories have been entered, without any replication, he can save and be redirected to the main page of the annotation. Here the user can choose any image to annotate and in the list of available categories there will also be the new ones.

Export the ground truth file

The user wants to use the gt file externally from ODIN Web, and they want to download it.

- **Precondition:** the user must be logged in to ODIN Web and must have already entered the dataset.
- **Basic Flow:** the user is on the data set selection page. Navigate through the datasets and select the one you want to add the category to, after which start the annotator. By going to the annotator settings, the user will be able to download the gt file by clicking on the button *Export GT*. Once done, the file will be automatically downloaded by the browser.

3.1.3. Analyzer use cases

Dataset analyses

The user wants to analyze if the dataset he has created is well balanced as regards the number of classes and if they are well distributed for each property.

- **Precondition:** the user must be logged in to ODIN Web and must have already entered the dataset.
- **Basic Flow:** the user is on the data set selection page. Browse the datasets and select the one you want to analyze. Once directed to the landing page, start the analyzer. After setting the required parameters, he is directed to the analysis page. In the Dataset tab, select the categories tab to understand if in the annotation phase it has distributed the number of categories well or if one has more occurrences than the others. After which he moves to the Distribution of Property tab in which, navigating between the properties and their relative values, he tries to understand if the property values are well distributed among all categories.

Model analyses

The user has just tested his model and wants to find out what the weaknesses of his model are.

- **Precondition:** the user must be logged in to ODIN Web and must have already entered the dataset with the predictions.
- **Basic Flow:** the user is on the data set selection page. Browse the datasets and select the one you want to analyze. Once directed to the landing page, start the analyzer. After setting the required parameters, he is directed to the analysis page. Clicking on the *prediction* tab and selecting *properties*, it analyzes each score per-property. The user notes that, for a given property value, the model has a low score. This highlighted a weakness under that particular condition. The user therefore tries to clarify whether the problem is due to a low number of observations with that property value or if the model actually has a weakness by displaying both the distributions of the property on the categories, and the false positives and negatives distributions.

Compare multiple models

The user has built two models with different structures but with the same purpose. Now he wants to understand where one model is more robust than the other and how to exploit these differences in order to build a complete one.

- **Precondition:** the user must be logged in to ODIN Web and must have already entered the dataset with the two predictions.
- **Basic Flow:** the user is on the data set selection page. Browse the datasets and select the one you want to analyze. Once directed to the landing page, start the analyzer. After setting the required parameters, he is directed to the analysis page. Browsing through the various analyzes he decides to start trying to grasp what are the weaknesses in the recognition of the various categories for each model. Going into the error analysis select both models to carry out the analysis. Browsing through the various categories, they notice a huge difference in terms of false positive errors for category x . In this case, model one is weaker in terms of its identification caused by similarity errors. Similarly, the second model has a lower error impact but for generic errors. This means that model one was able to recognize the discriminating feature of category x , unlike the second, but made more mistakes as another category is similar to x .

3.2. System Architecture

In this section, the proposed architecture for ODIN Web will be dealt with in detail, analyzing the different components on both client and server sides and the various technologies used for its implementation.

ODIN Web focuses on the concept of dataset, which is a folder created and stored on the server side that contains the ground truth, predictions, images, and, optionally, the `properties_file`.

For what concern the ground truth file, it contains all the information relating to the dataset, jointly with the observations and related annotations that will be compared with the model's predictions. Entering into the specifics of the file composition, the following fields are stored:

- *categories* – an updated list of all the categories present in the data set with their relative ids.
- *meta-annotations* – a list of all the properties associated to the images with their related values.
- *task_type* – is the task related to this data set.
- *images* – an updated list of all the images present in the data set with their relative ids and a flag used to know if this image was discarded in annotation phase.

- *observations/annotations* – a list of all the observations/annotations for all the images that contain, not only the category and, in case of localization, the relative position (as bounding box or segmentation), but also the properties that denote domain specific characteristic of the object.

Moreover, in this version, to give the user the possibility to exploit the same data set for multiple tasks (in case it has been imported and therefore the annotation structure is different) the concept of configuration has been introduced. Each data set has a *.json* file containing the configuration to be analyzed, with the addition of alternative paths for predictions and a series of customized properties such as the name of the model in case the user wants to take advantage of the option of comparator, the properties file path, in case there is one already existing in another folder, and the *match_on_filename* function. In addition to the ground truth file, the folder also has (user-loaded) model predictions that make it possible for ODIN Web to elaborate the data and provides, as output, different analyses as shown in Figure 3.1. In order to create or modify the ground truth

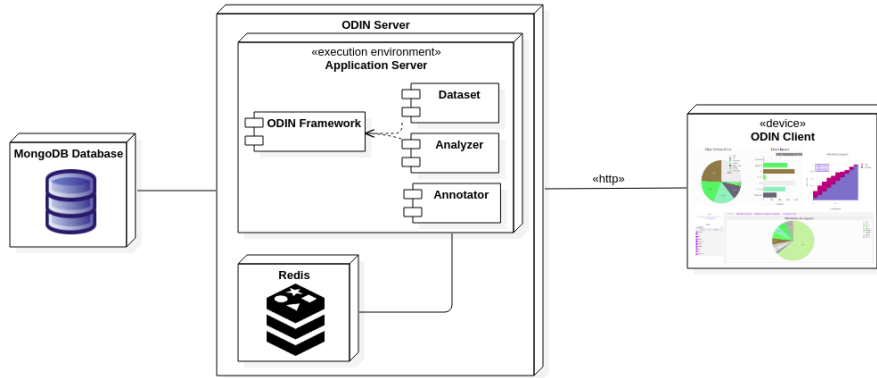


Figure 3.1: ODIN Web architecture

file, the user has the option to add annotations and properties, or modify them, using the Annotator component. For more information about the Annotator see Section 3.2.3.

The application flow will be divided into two main components:

- *Annotator*: allows the user to access, or if necessary create, the data set to provide to the model. Once created, the user can populate the data set and specify, for each observation, the category, or categories in case of multi-label classification, it represents and the properties that characterize it. Once all the samples are annotated, they will be saved in the right format on the ground truth file.
- *Analyzer*: it can be defined as the main feature of ODIN Web. It allows you to analyze the predictions of one or more models easily and without programming

effort. By simply selecting the dataset, the predictions associated with it, and the settings with which you want to perform the analyzes, you can access all the functions that ODIN Framework already offers, provided with a simple interface and equipped with all the filters necessary for inspection and comprehension of the analyzes at different levels of granularity.

3.2.1. Technologies Involved

ODIN Web was developed using various technologies to create both the Client and the Server part of the application. In this section we will analyze in detail the technologies used both in the front-end and back-end and those used in the communication between them, going to support the choice.

Starting from the client side, for the construction of the application user interface it was chosen to take advantage of Vue.js thanks to its ability to break down the application into modules making the code cleaner and tidier. The processing part of the client is managed through the Javascript scripting language. As for the Server part, it is developed entirely in python, following the line of the previous work [47][53]. As for the communication and reception of requests by the server, it is handled entirely by Flask. In the next part we will analyze these technologies specifically.

Docker

Docker is an open-source project that automates the process of deploying applications within software containers, providing an additional abstraction thanks to virtualization at the Linux operating system level. Thanks to docker are ensured:

- *Portability*: docker containers offer a lot of portability. With containers, it is possible to build an application once, place it inside a container image and run it on any host environment that supports Docker and runs on the same family of operating system.
- *Code Isolation*: docker allows to run many containers simultaneously on a given host. Containers are lightweight and include everything needed to run the application, so it is not necessary to rely on what is currently installed on the host.
- *Dependencies packaging*: every library or dependency that the container needs to work is already installed inside it so as to encapsulate everything inside it without the need to be supported by external libraries
- *Fast Installation Procedure*

In order to take advantage of the features offered by Docker, We have decided to divide the architecture as follows:

- a Docker container in which MongoDB resides and communicates with the ODIN container only to exchange information within the same network and avoiding to expose a port outside the network.

- A Docker container in which ODIN and the Redis database reside.

We, therefore, decided to expose a single port outside the network and it will be the fundamental one for the communication between the Client (Vue and Axios) and the Flask server, which is mapped on port 6006.

Vue.js

Vue.js², or Vue, is a javascript framework that allows you to create user interfaces through a declarative approach, abstracting the process of rendering data and updating the DOM. It is based on the HTML, CSS and javascript standards, and offers a minimal and organized component declaration. Furthermore, the two main features that make Vue one of the most used frameworks for developing Web interfaces are:

- *Declarative Rendering*: Vue extends standard HTML with a template syntax that allows us to declaratively describe HTML output based on JavaScript state.
- *Reactivity*: Vue automatically tracks JavaScript state changes and efficiently updates the DOM when changes happen.

Its choice was guided by the fact that it allows to develop a software with a reactive MVVM(Model-View-ViewModel) architecture. In fact, the user interacts with the HTML based web page and normally the whole page has to refresh even if just one object changes. Vue uses a virtual copy of the original DOM that figures out what elements require updating, without re-rendering the entire DOM, greatly improving app performance and speed.

```
import { createApp } from 'vue'

createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount('#app')
```

```
<div id="app">
  <button @click="count++">
    Count is: {{ count }}
  </button>
</div>
```

The above image is an example of the features offered by Vue. In this case it is possible to see how the javascript code and the component template are completely separated from each other. The @click event allows javascript to keep track of state changes, in this case pressing the button to increase the counter.

²<https://vuejs.org>

Axios

As for asynchronous calls we have decided to rely on Axios³. Defined by Vue as the unofficial library for handling ajax calls, it is a derivative of the \$http Service module used by Angular and made standalone. Axios is a Javascript library used to make HTTP requests from node.js or XMLHttpRequests from the browser and it supports the Promise API that is native to JS ES6. It was chosen for its ease of use but at the same time its ability to handle promise-based calls, which give the possibility to take advantage of Javascript async and wait for a more readable and effective asynchronous code. Finally, as regards security, It can be used to intercept HTTP requests and responses and enables client-side protection against XSRF. It also has the ability to cancel requests.

```
import axios from "axios";
axios.get('/users')
  .then(res => {
    console.log(res.data);
  });
```

In this figure an example of a call made by the client is shown. In this case a Promise is generated by Axios, when the asynchronous call is successful, the result is printed in console.

Flask

Flask⁴ is a web framework. This means flask provides tools, libraries, and technologies that allow you to build a web application.

Flask is part of the categories of the micro-framework. Micro-framework are normally framework with little to no dependencies to external libraries. This has pros and cons. Pros would be that the framework is light, there are little dependency to update and watch for security bugs, cons is that some time you will have to do more work by yourself or increase yourself the list of dependencies by adding plugins. For what concern Flask, its dependencies are:

- Werkzeug a WSGI utility library
- jinja2 which is its template engine

In ODIN Web we used Flask mainly for its *App Routing* functionality, which is used to map specific URLs with associated functions that are built to perform a task. In addition

³<https://axios-http.com/docs/intro>

⁴<https://flask.palletsprojects.com/en/2.0.x/>

to the URL as a route parameter, the `@app.route()` decorator also accepts a second argument: the list of HTTP methods that are supported by that URL. Furthermore, it is possible to define a route with carrot brackets `<>` inside that indicate a variable, in order to enable routes to be dynamically generated.

Redis

For this work Redis⁵ was used as a memory to store the data structure converted into Bitmap of the instances of the objects associated with the user as regards Dataset, Analyzer and Comparator. It was chosen, first of all for its performance, as it offers response times of less than a millisecond, allowing millions of requests per second. Also, since Redis data resides in memory, it enables data access with low latency and high throughput. The main advantage is that, unlike traditional databases, data in memory does not require going to disk, minimizing engine latency to microsecond levels.

MongoDB

MongoDB⁶ is a NoSQL database used for managing large volumes of data to be stored. ODIN Web has adopted this type of database as regards the storage of observations / annotations of a dataset which can vary from hundreds to thousands. MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB.

We opted to use MongoDB as it was developed specifically for the management of un-structure data, allowing them to be stored in json format. In this way it is possible to save automatically and without manipulating the annotations provided by the client, in addition it allows to update the schemes quickly. In our case the structure of the collections is very simple as the only data to manage are the observations and annotations, two collections have been created to manage these data in an orderly and separate way.

3.2.2. Back-End

In this section we will break down and analyze in detail the computational part of ODIN Web. It has been built in modules in order to have a clear distinction of what are the features offered and to facilitate subsequent extensions of the app. In this case, each module covers a specific function, from the management of annotations, to the creation of the data set, up to the management of the analyzes. As we can see in Figure 3.2 every

⁵<https://redis.io/>

⁶<https://www.mongodb.com/>

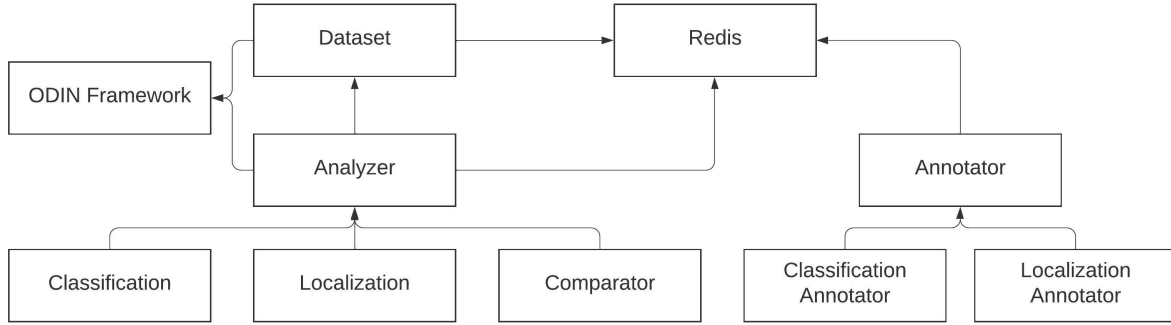


Figure 3.2: UML Class Diagram representing the main components

module is split in order to manage in a separated way the two type of analyzes performed by ODIN Framework and each of their functions are invoked through Flask thanks to the concept of routing.

Dataset

The dataset component deals with the management of the dataset in its entirety. To create it, on the client side, the following parameters must be entered:

- *dataset_name*: the name with which the folder in which the data set will reside will be generated.
- *categories*: list of classes belonging to the data set.
- *meta_annotations*: a dictionary that has properties as keys and a list of property values as values. (*'property'*: [*'property_value1'*, *'property_value2'*, ...])
- *taskType*: type of the task that the user wants to address.
- *images_path*: the relative path in which the dataset can find the images. If the user does not specify any folder, a specific folder will be created for that dataset that the user will have to populate manually.

In addition, it is possible to import an existing data set. In this case the parameters to specify are:

- *dataset_name*: the name with which the folder in which the data set will reside will be generated.
- *ground_truth_file*: path to the ground truth file.

- *prediction_path*: Path of the proposals of a single model or list of couples. Each couple contains the model name and the corresponding proposals path.
- *images_path*: the relative path in which the dataset can found the images.
- *properties_file*: in an optional path, and it refers to the properties_file created by ODIN for that data set.
- *taskType*: task for which the ground truth file was created.
- *match_on_filename*: Indicates whether the predictions refer to the ground truth by file_name or by id.

In both cases, the folder is generated with the name chosen by the user and a configuration file in which the main information will be saved such as: path for the ground truth, predictions, images and the properties file, the match on file name and the type of task requested. In the event that the data set has been imported and the ground truth file does not have the meta-annotations field, it will be generated automatically by extracting all the properties and their values from the annotations field inside the file.

The component also contains all the methods necessary for the management of the configuration file and the analyzes that can be carried out on the data set such as:

- *get_categories_distributions*: it returns the distribution values of the categories in the dataset
- *get_properties_distributions*: provides the distribution of all the properties loaded in the dataset instance, and for each of their value, its distribution among all the categories.
- *get_co_occurrence_matrix*: provides the matrix of the occurrences between categories.

Analyzer

The modules shown in the figure 3.3 deal with the reception and processing of data concerning the model analysis phase. As we can see, the components are divided by type of task:

- *analyzer_classification*: deals with the reception and routing of all analysis requests concerning a dataset with task type *single*, *multi label* and *binary classification*.
- *analyzer_localization*: deals with the reception and routing of all analysis requests concerning a dataset with task type *object detection* or *instance segmentation*.

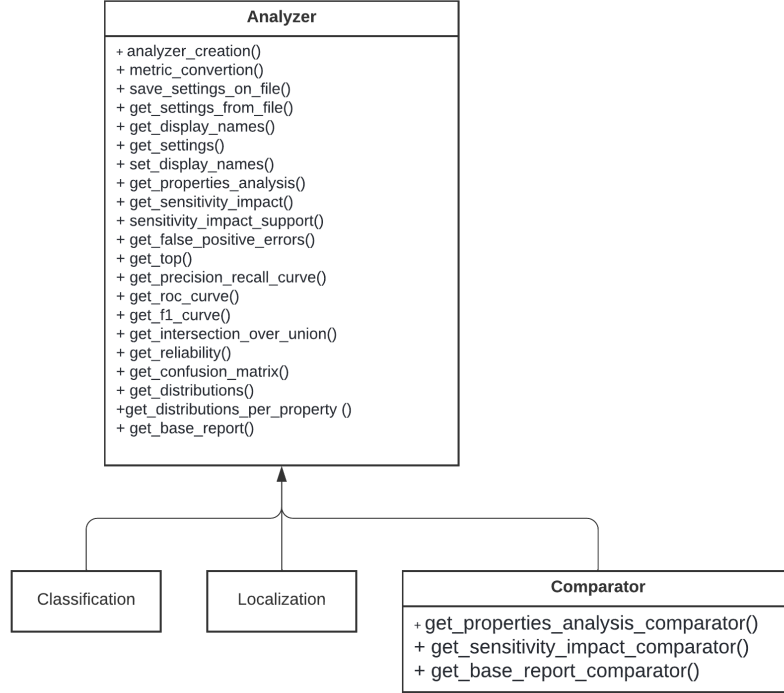


Figure 3.3: UML of the Analyzer Component

- *comparator*: used when it is necessary to carry out analyzes with multiple models.

The *analyzer* is the main component of ODIN Web that takes care of instantiating the analyzer class, saving it and subsequently calling all the functions from the ODIN Framework and generating the ideal data format to be sent later to the client. To create the analyzer the only required parameters are:

- *properties*: a list of all the properties that the user wants to load and view during the analysis.
- *already_loaded*: a boolean indicating whether the dataset has loaded a *properties_file* or created a new one

Furthermore, the first time a dataset analysis with predictions is performed, a series of parameters can be set:

- *similar_classes*: list of groups of categories inside the dataset which are similar to each other
- *Threshold*: the confidence threshold applied to each model loaded
- *iou-threshold*: used only for localization tasks, the intersection-over-union threshold is divided in strong-threshold, used in order to consider all the predictions with a

IoU value less than it as False Positive, and weak-threshold for the identification of the localization errors

- *categories-factor-check*: a boolean to know if the user wants to apply the normalization
- *categories-factor*: is the normalization factor applied by the analyzer for each category
- *properties-factor-check*: a boolean to know if the user wants to apply the normalization also on the properties
- *properties-factor*: list of all the normalization factors for each property

Those parameters are stored in the *config.json* file. In this way, the next time the user chooses to analyze the model, it will be loaded automatically with the parameters selected the previous time, or modified within the analyzer settings. When the analyzer is created, ODIN Web instantiates two different objects based on the number of models loaded. In this way the user has the possibility, in the event that more than one model has been loaded, to take advantage of one of the main features of ODIN Framework, that is the ability to compare multiple models at the same time in the analyzes that support this mode through the *Comparator*. All the analyses are listed in the comparator object in Figure 3.3.

Annotator

The ODIN Web annotator has been divided into modules, as shown in Figure 3.4, that independently manage all the operations concerning the required task. Based on it, the right module is taken into account in order to perform a Classification task with *classification_annotator* or a Localization task with *localization_annotator*, for what concern the management of the annotations made by the user. While, for everything related to the management of the ground truth, the *annotator* module is designed.

Start the annotator

By accessing the main page of the annotator the *start_annotator()* function is invoked. It performs all the operations necessary for the validation and loading of the data residing in the ground truth file in order to provide the user with a complete view of the data set and annotations applied to it so far. In the event that the user has inserted, or removed, images the function will automatically update the ground truth file, without deleting the

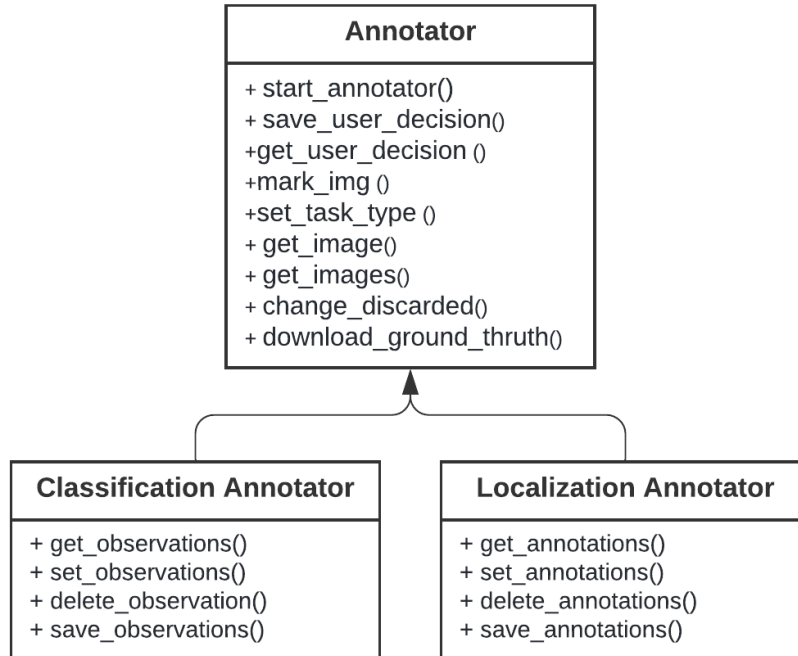


Figure 3.4: UML of the Annotator Component

annotations that could be useful at a later time.

As for saving annotations, it was decided to take advantage of MongoDB. In fact, in anticipation of a large number of annotations, it was decided to adopt a non-relational DBMS in order to be able to perform queries, such as get annotations referring to a specific image, instantly without compromising the user experience and limit the number of accesses to the ground truth file. In this way, two collections, in order to store annotations separately for each task, are made.

In Figure 3.5 the composition of the Observation collection within MongoDB is figured. In this case, it is managed in such a way as to support both annotations for the multi-label task, and for single and binary. The collection contains the following fields:

- *_id*: Primary key used by MongoDB to identify a document in a particular collection. An ObjectID is a 12-byte Field Of BSON type where: the first 4 bytes representing the Unix Timestamp of the document, next 3 bytes are the machine Id on which the MongoDB server is running, next 2 bytes are of process id and the last Field is 3 bytes used for increment the objectid.

- *file_name*: name of the image the observation is associated with.
- *category*: field used by *binary* and *single-label* classification tasks, reports the class identifier in the categories field of the ground truth file.
- *categories*: field used by *multi-label* classification tasks, reports a list of class identifiers which refers to the ids of the categories field of the ground truth file.
- *uid*: identifier of the observation within the ground truth file. Used in such a way that each annotation will always and only have an identifier. (Different from *_id* in that two different datasets can have annotations with the same id).
- *datasetName*: Name of the dataset to which the observation refers
- *meta_annotation_values*: generic field that indicates one / more fields referring to properties. (If a dataset has "Color?" with "rgb" or "bw" values, there will be a "Color?" with value "yes / no" if it is inserted ad null if it is not specified).

As for the Annotation collection, the structure is almost the same, except for some small differences due to the type of annotation that the task requires. Indeed Figure 3.6 exhibit:

- *category_id*: field used to report the class identifier in the categories field of the ground truth file to which the annotation is referred.
- *segmentation*: list of points with the following pattern () needed to reconstruct the pixel mask of the segment. This field is used for the *instance segmentation* task.
- *bbox*: contains four values. The first two refer to the coordinates of the top left point of the bounding box, while the other two values refer to the width and height of the bounding box. This field is used for the *object detection* task.
- *image_id*: field to associate the annotation to the image at which are referred.

Redis

This component manages everything related to saving and extracting instances of objects used by the analyzer and to maintain the user's session in case of reloading the page. It consists of the following functions:

- *redis_get/store*: function used to save dataset and analyzer instances. Being Redis a Remote Dictionary Service, it is managed with a concept of key: value. In this case the instances are saved with the key *my_dataset* and *my_analyzer* and in addition the user uuid in order to manage the analyzes of several users at the same time without generating conflicts.

| observation | |
|------------------------|-----------------|
| _id | ObjectId |
| file_name | varchar |
| category | int |
| categories | array[int] |
| uid | int |
| datasetName | varchar |
| meta_annotation_values | varchar |

Figure 3.5: MongoDB Observation Structure

| annotation | |
|------------------------|-----------------|
| _id | ObjectId |
| category_id | int |
| segmentation | array[int] |
| bbox | array[int] |
| image_id | int |
| uid | int |
| datasetName | varchar |
| meta_annotation_values | varchar |

Figure 3.6: MongoDB Annotation Structure

- *get/store_model_name*: Handles the case in which the analyzer has been created to use the comparator functionality. In this way, it is always possible to know which model is currently being analyzed.
- *change_analyzer*: since the model in position 0 in the list is the one analyzed at the moment, this function allows you to change the model under analysis with the one chosen by the user.
- *redis_annotator_get/store*: function used to maintain the user's session in the annotator without having to log in again every time the page is reloaded.

This module is essential to guarantee speed of response to the client when a request is made regarding an analysis previously made. In fact, thanks to the caching performed by the analyzer, it maintains its instance without going to recreate it at each request. In this way it is possible to provide the results of the analyzes in case the user wants to compare a model already analyzed with a new one. In this case, only the analysis of the new model will be calculated.

3.2.3. Front-End

In this section we will introduce all the functions offered by Odin Web and, for each of them, a brief description of the layout.

In order to take advantage of both the annotator and the analyzer the user needs to create or import a dataset to which the observations will initially be associated and later, with the addition of the model predictions, the analyzes will be carried out.

In order to create a dataset from scratch it will be necessary to enter in the interface,

The screenshot shows the 'Create new Dataset' form in the ODIN Web interface. The form is titled 'Create new Dataset' and is located below a header bar that contains 'ODIN WEB' and a 'Logout' link. The form fields are as follows:

- Dataset Name*:** A text input field with the placeholder 'Enter name'.
- Classes*:** A text input field with the placeholder 'Insert class name'. Below it, there are two blue pill-shaped buttons labeled 'class1' and 'class2', each with a small 'x' icon to its right.
- Properties:** A text input field with the placeholder 'No properties loaded'. To its right is a blue '+' icon.
- Task Type*:** A dropdown menu with the placeholder 'Choose a task type'.
- Images Path:** A text input field with the placeholder 'Path to the images folder'.

Below the form fields is a green button labeled 'Create Dataset'. At the bottom of the form, there is a link that says 'Already have a dataset? Import'.

Figure 3.7: ODIN Web Interface: create new dataset

shown in Figure 3.7, the following parameters:

- *Dataset Name* – mandatory – name of the dataset to create
- *Classes* – mandatory – categories of the dataset, it will be necessary to enter at least two categories in order to continue.
- *Properties* – optional – A list of properties with their relative values
- *Task type* – mandatory – task necessary to then create the annotations in the annotator
- *Images path* – optional – Path to the folder containing the images to be annotated. In the event that the user does not specify any path, a folder will be generated within that of the dataset in which the images can be subsequently inserted

If the user wants to import parts of the dataset already present in the folder or not created by ODIN Web, it will be possible to do so through the interface shown in Figure 3.8 and which is accessed through the link 'Already have a dataset? Import'. On the left side you can see the complete list of all the datasets present in the datasets folder that resides in the server. In this way the user can decide to import an entire dataset and, by clicking on it, the form fields will be filled in automatically. The fields present are:

- *Dataset Name* – mandatory – name of the dataset to create

ODIN WEB Logout

←

Select a dataset:

Search: ar

- artdl-od
- artdl-od-configuration
- artdl-ml

Import Dataset

Dataset Name*: New_Dataset

Dataset GT path*: datasets/artdl-od/gt.json

Prediction Path:

| | |
|---------|--------------------------------|
| Model_0 | datasets/artdl-od/dets-faster_ |
| Model_1 | datasets/artdl-od/dets_predici |

Properties path: datasets/artdl-od/Artdl_properties.json

Images path: datasets/artdl-od/images

Task type*: Object Detection

Match on Filename: ☒

Import Dataset

Figure 3.8: ODIN Web Interface: import dataset

- *Dataset GT path* – mandatory – path of the folder containing the gt file they want to use
- *Prediction Path* – optional – Name to be displayed during analysis and path for model predictions. it is possible to insert more than one model to take advantage of the comparator option
- *Properties path* – optional – Path to the properties file previously created for another dataset from ODIN. If it remains empty, it will be updated with a new file called `<dataset_name>_properties` at the first instantiation of the dataset.
- *Images path* – optional – Path to the folder containing the images to be annotated. In the event that the user does not specify any path, a folder will be generated within that of the dataset in which the images can be subsequently inserted
- *Task type* – mandatory – task necessary to then create the annotations in the annotator
- *Match on Filename* – optional – take advantage of the match on file name. Default is false.

After the creation of the dataset it will be possible to take advantage of all the following features offered by ODIN Web which will be extensively discussed in the following sections.

Annotator

As already mentioned, ODIN Framework gives the possibility to create and populate the ground truth file of the data set through the annotator. In this way, it offers a user friendly interface that allows you to have an overview of the entire dataset and various functions able to break it down according to criteria such as type of category, properties or absence of annotations. It also offers an intuitive editor that supports the user in manually annotating images quickly and with every tool at hand. In fact, the user can add observations/annotations, depending on the required task, with the categories and properties previously inserted in the creation of the dataset or added in the course of work.

As already said, in order to work it is necessary that the user has created/imported a dataset with ODIN Web by referencing the directory in which to find the images and providing/creating the ground truth file containing the list of categories and, optionally, of the properties. The first time the annotator is accessed, it will scan the ground truth file and, in case it has been imported, will automatically extract the properties present in the observations field.

At startup the user lands on the annotator main page, figured in 3.9, in which images

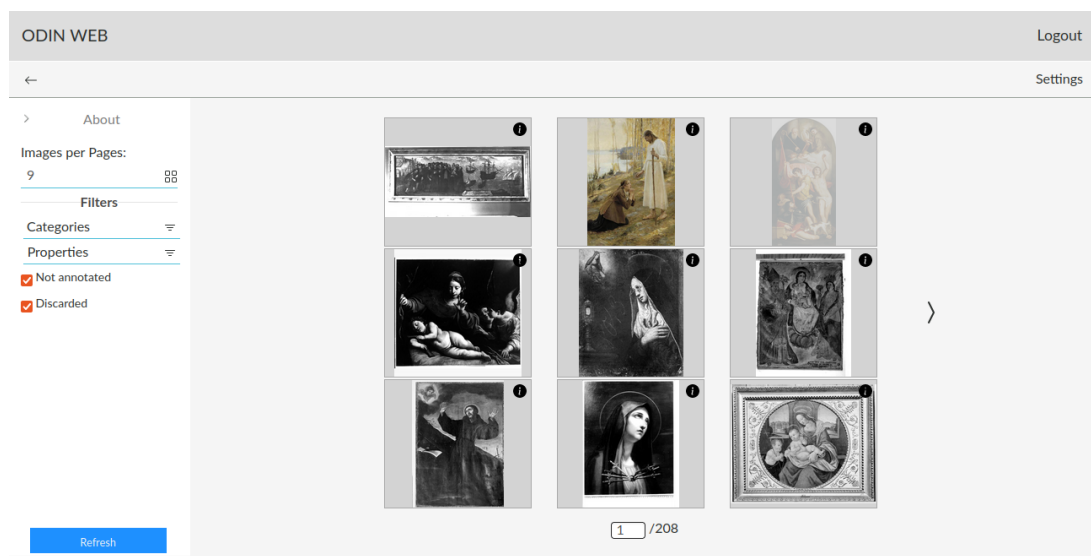


Figure 3.9: Landing page of the annotator

are displayed, with their related information, in a carousel form. The number of images displayed can be modified with the appropriate filter. It also provides a series of filters useful for the user to select only the images of interest: such as those that have yet to be annotated or labeled as belonging to a specific class / property or a set of them. In case

the user wants to add more images when he is already inside, he will only have to insert the new images in the folder of that dataset designated during creation, or created by ODIN Web, and by clicking the *Refresh* button, it analyzes if there are new images and, if so, will automatically update the ground truth file and the related annotator view.

In order to start the annotation task is necessary to click on the image the user wants to annotate, after that they are displayed two different screens based on the required task:

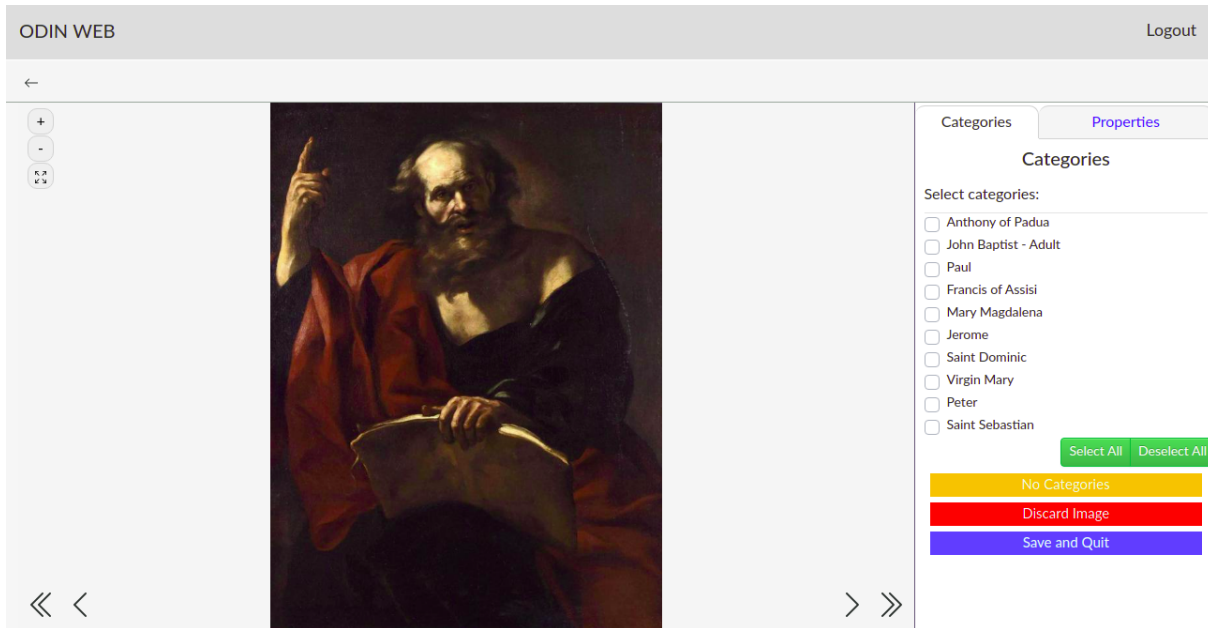


Figure 3.10: Annotator page for Classification tasks

- *Classification*: if the task is classification, the user is directed to the page shown in Figure 3.10 which is divided into two macro columns. In the left section everything concerning the visualization and management of the image is managed. It includes the buttons that allow you to zoom in and out the image, the button to center it (in case the user moves it) and the buttons that allow you to navigate between the images in the lower part of the screen.

In fact, they are created in order to browse one image at a time or go to the first non-annotated image in that direction (if they are all annotated, it remains on the current image). In this way the annotation task is optimized and there is no need to search for the single image to annotate.

In the right column, on the other hand, the categories and the properties previously entered by the user, or automatically loaded by ODIN Web if the dataset has been imported, are shown. In case the task types are single label or binary classification ODIN Web offers the possibility to select only one category through radio buttons, in case of multi-label classification a list of checking box categories are displayed. If

the image does not contains classes it can be labeled as 'no categories' so that it is not taken into account when the user wants to move to the nearest non-annotated image. This feature is only available if there are not selected categories yet.

The *Discard Image* button instead allows the user to completely discard the image from those displayed, in this way inside the carousel of the images on the main page the image will be displayed as darkened and right clicking it offers the possibility to re-integrate it in the images to be displayed.

- *Localization*: as far as localization tasks are concerned, the interface is slightly different. As we can see in Figure 3.11 the view is always divided into two components.

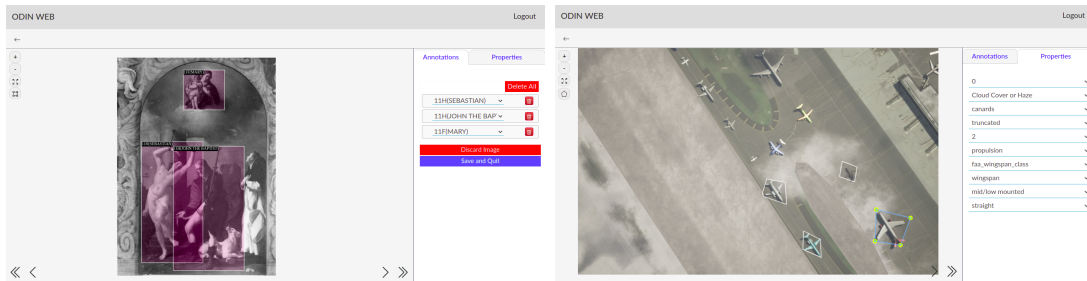


Figure 3.11: Annotator page for Object Detection(left) and Instance Segmentation(right)

In addition to the buttons for managing the image, a button used to create a bounding box, in case the task type is Object detection, or assign the drawing tool to the mouse pointer that allows the user to identify the points of the segmentation mask for that class in the case of instance segmentation, is added. Finally, the right column is the one that concerns the management of annotations. In fact, it lists the annotations created for the current image and the class associated with it. By clicking on an element it is possible to highlight the bounding box / segment associated with it and assign the meta annotations displayed in the Instance Segmentation case in Figure 3.11. Even in this case there is the possibility to label the image as no classes in order to skip in case of search to a non-annotated image, or to completely discard it from the list of images.

Analyzer

The analyzer is the main component of ODIN Framework that allows users to perform different analyses on the model in order to understand the performances of it with different levels of abstraction. In Figure 3.13 it is possible to visualize the flow of the application through all the components and the interfaces of the analyzer. To access it, the analyzer needs to have, on the server side, an instantiated data set with ground truth and, if the user wants to access all the analyzes, the prediction of the models. Once the dataset has

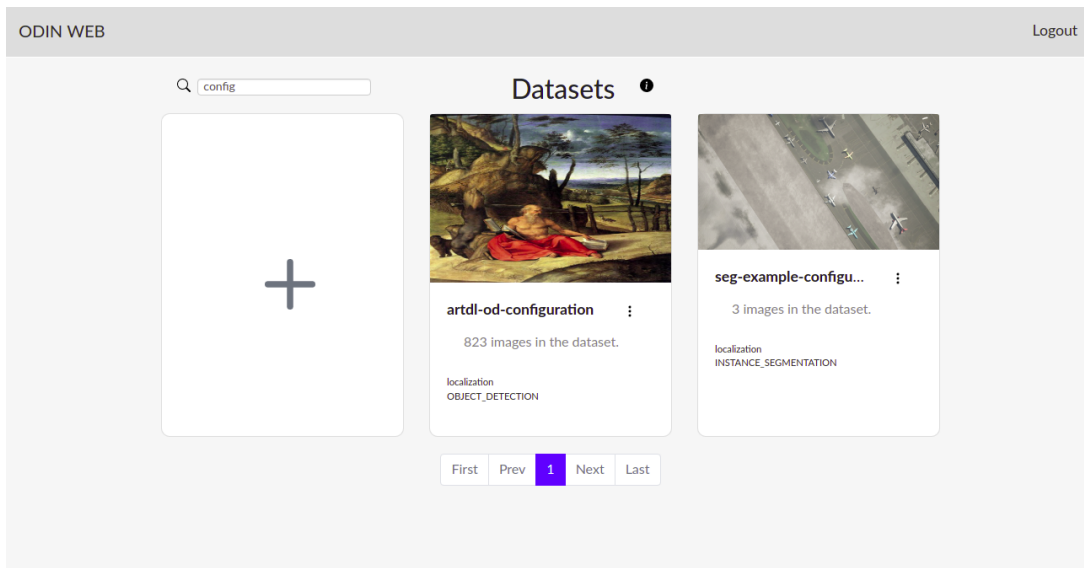


Figure 3.12: Dataset Selection page

been added, it will be displayed in the list in Figure 3.12, which presents all the datasets loaded on ODIN Web. Once the dataset of interest has been selected and the analyzer started, the user has the possibility to configure the various settings that can subsequently be modified, with which the analyzes will be carried out and that they will be saved for subsequent times. After that it will be possible to access all the analyzes made available by ODIN.

With the aim of providing a simple, clean but at the same time functional interface, the entire structure of the analyzer has been divided, and depicted in Figure 3.14, as follows:

- *Analyzes list*(A): each type of analysis has its own tab, in this way the user has the list of all possible analyzes on the top of the screen.
- *Filters column*(B): in the left part of the screen, it is possible to view different components depending on the type of analysis being carried out, among these there are:

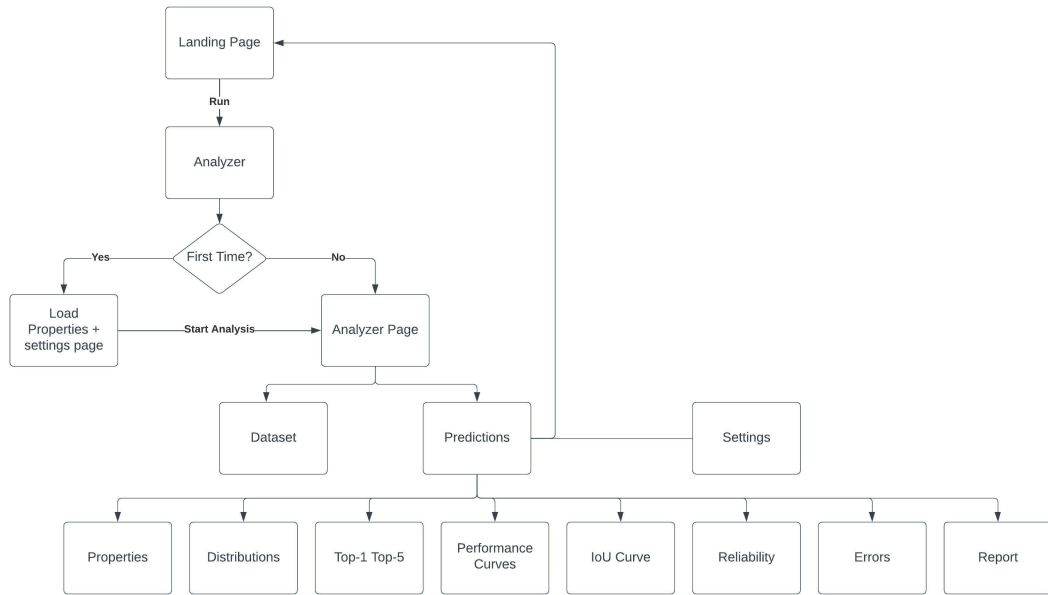


Figure 3.13: Analyzer Flow

- *About*: A resealable box that summarizes in a few lines how the graph must be interpreted.
- *Selectors*: A series of selectors that allow you to change the entity analyzed by the graph.
- *Filters*: filters that allow you to narrow or broaden the data basin for the creation of the graph.
- *Plot tabs(C)*: A series of cards that contain a certain plot that falls under that category of analysis.

The graphs and the availability of data are managed independently by the tab in which they are contained. The sequence diagram shown in Figure 3.15 describes a classic request for data by the tab under analysis, if it is not updated with the latest changes made by the user as regards the number of models or the parameters set in the settings. As we can see, the graphs inside the tab are not updated to the last parameters set by the user. In this case it performs an asynchronous GET request for the data necessary for each analysis. When the data is received, they are supplied to the various sub-components with the *generatePlot* function that store the actual data in the component and create the plot that deal with the generation of the chart. After that, each change performed on filters is managed through the *updateChart* function. If it does not have the necessary data to display the graph, the tab will make a second call.

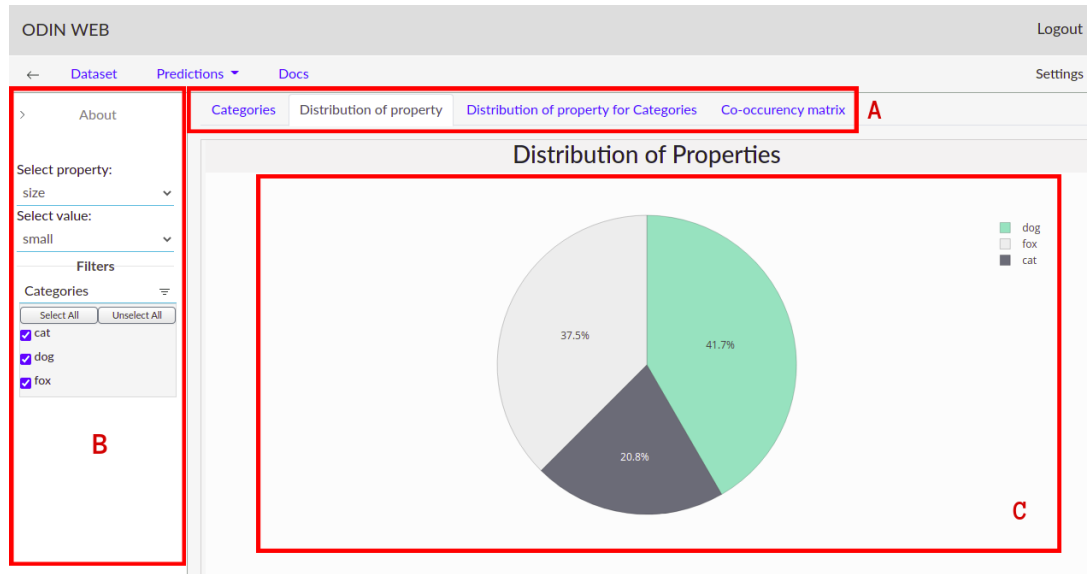


Figure 3.14: Basic structure of each Analyzer Tab

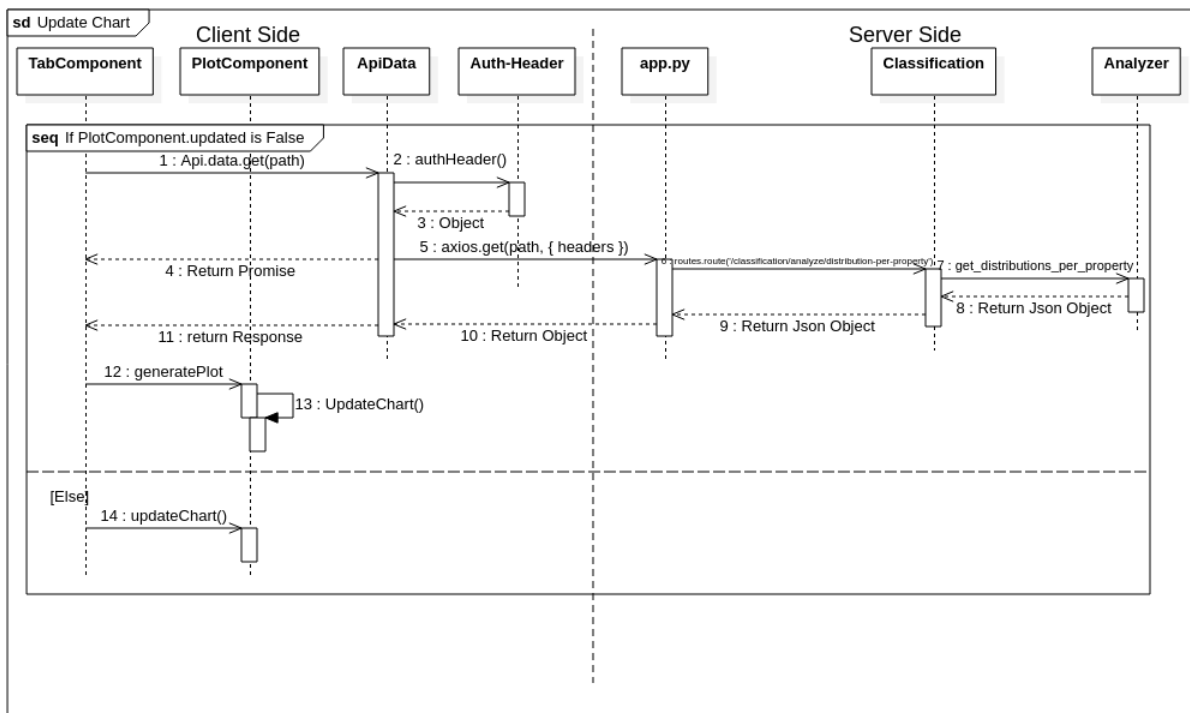


Figure 3.15: Sequence Diagram of the plot creation

The structure of the various tabs and the analyzes are discussed and reported in detail in the following sections.

Dataset

On this page are reported all the analyses related to the data set. In fact, it offers 3

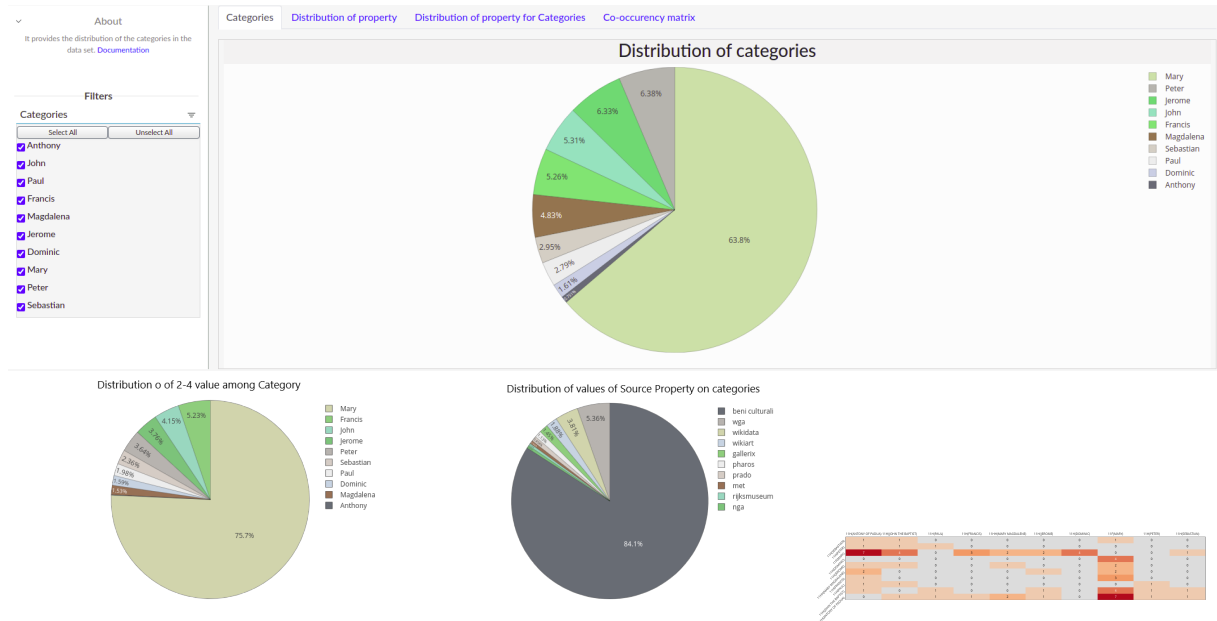


Figure 3.16: Tool Interface for Dataset analyzes (Top), Distribution of Property, Distribution of Property for Categories and Categories Co-occurrence Matrix

functions that allow you to analyze the distributions of categories and properties within the data set. These features are essential in case you want to find a bias in the data set and therefore whether or not to apply normalization during the performance evaluation.

- *Categories* – as we can see in Figure 3.16, it provides a visual representation of the distribution of the selected categories in the dataset. Thanks to the provided filter it is possible to exclude some categories by deselecting them for a partial view.
- *Distribution of Property* – offers a visual representation of the distribution of all the properties selected during the analyzer creation phase, or added later in the settings, and for each of them the distribution of each value on all categories.
- *Distribution of property for Categories* – It allows to view the distribution of the property values selected by the filter on the categories chosen by the user through the category filter.
- *Categories co-occurrence matrix* – offers a graphical representation of the occurrence of a certain category in relation to the others. That is, it highlights a relationship between them.

Analyzer: Properties

In this section all the analyzes concerning properties and their values have been grouped. Two analyzes can be identified in this interface:

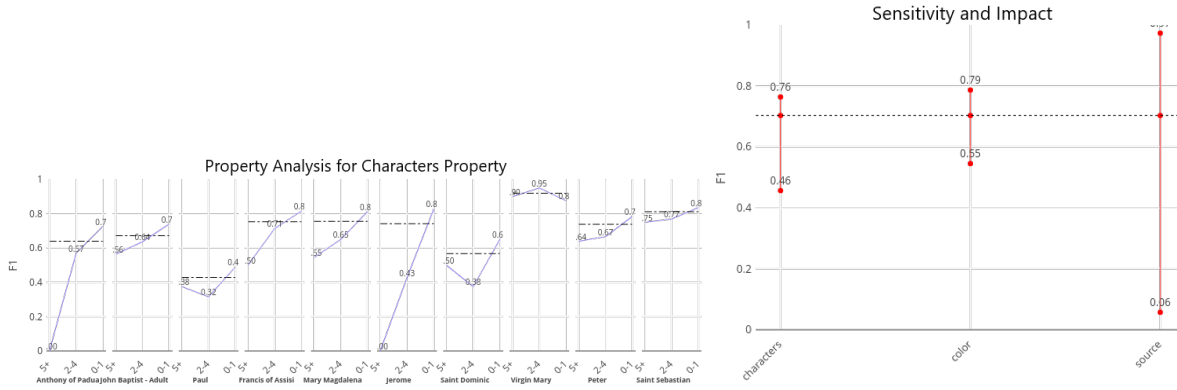


Figure 3.17: Per property (on left) and Sensitivity and Impact (on right) analyzes

- *Properties Analysis* – This analysis shows all the results obtained from the various property values, sorted by overall performance, for each category. The user has the option to divide the values by:
 - *Properties Values*: in this case, the user will have a broader view of the categories and the scores they have obtained for the various values.
 - *Categories*: split used in case they want to analyze a possible relationship between property values.

If they want to exclude certain categories or values, it will be possible to do so through the relative filters.

In the case of a comparison between multiple models, a chart will be generated for each property value, in this way the user can analyze which model has made the highest score for that value in a given category

- *Sensitivity and Impact of Properties* – Two parameters are taken into consideration in this analysis: the first is all the *properties* loaded by the analyzer at that moment, the other is the evaluation *metric* used for the analysis. As we can see in Figure 3.17 for each loaded property two values are defined, the minimum and the maximum evaluation score reached by the model. The values are calculated by averaging all the evaluation scores (minimum / maximum) achieved for each category by the property. Sensitivity is defined as the difference between the minimum and maximum of each property. Finally, the overall performance value of the model defined by the horizontal dashed line is also displayed.

Also this analysis support multi-model comparison, in which is simply displayed a chart per property that plots the values for each model.

Analyzer: Distributions

This part is entirely dedicated to the results given by the model in terms of:

- *True Positive*: a result is defined as true positive when the prediction performed by the model matches the observation regarding the presence of the class in the image.
- *True Negative*: is an outcome where the model correctly predicts the negative class.
- *False Positive*: it is a result given by the model, in which it incorrectly predicted the presence of a class
- *False Negative*: it is a result given by the model, in which it incorrectly predicted the negative class.

Three types of analysis are offered here, namely:

- *Distribution* – a pie chart that offers the possibility of viewing the distribution of categories for each type of result (True-Positive, True-Negative, False-Positive and False-Negative) selectable through the relative filter. On the right of the graph, a table is also provided which lists, for each category, the number of predictions that fall into that distribution. In this case the comparator allows to have a visual representation of the result of each model, per category, through a bar chart.
- *Distribution Per-Property* – provides a higher level of abstraction than the previous analysis, displaying the distribution of the values of the property selected by the user in the appropriate filter, for each category. Like the previous analysis it is possible to browse the distributions through the provided filter.
- *Confusion Matrix* – Provides a global view of distributions for each category, allowing the user to filter by property and value of it as shown in Figure 3.18. In this way the user can analyze in detail what are the strong points of the model, and where it needs to be strengthened.

Analyzer: Top-1 Top-5

This analysis reports the Top-1 and Top-5 scores in terms of accuracy (Figure 3.19) or error rate for each property value.

In particular, we talk about:

- *Top-1 score*: Only the highest probability prediction is taken into account
- *Top-5 score*: checks if the target label is one of the top 5 predictions which are the 5 ones with the highest probabilities.

Regarding the metrics used:

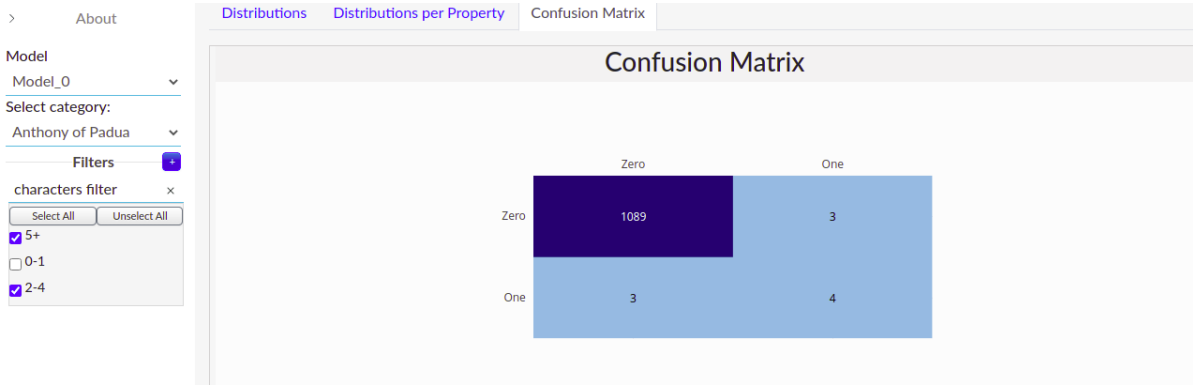


Figure 3.18: Confusion Matrix Interface with filter on Character Property

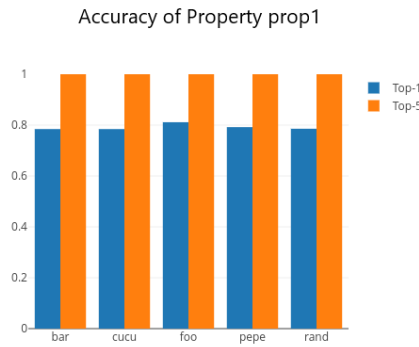


Figure 3.19: Accuracy of Top-1 and Top-5 for Property prop1

- *Accuracy*: is the number of correctly predicted data points out of all the data points. In other words, it is defined as:

$$Accuracy = \frac{TruePositive + TrueNegative}{Totalpredictions}$$

- *Error Rate*: is the number of wrongly predicted data points out of all the data points. In other words, it is defined as:

$$ErrorRate = \frac{FalsePositive + FalseNegative}{Totalpredictions} = (1 - Accuracy)$$

Analyzer: Performance Curves

This section is devoted entirely to analyzing the model's performance using evaluation curves. The curves offer us the possibility to compare overall and categories performances using:

- *Precision-Recall Curve* – For each threshold value, the precision and recall are calculated and plotted. The more the curve is flattened at the top, the higher the performance of the model is.
- *Receiver operating Characteristic Curve* – The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.
- *F1 Curve* – F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It is based on the following formula:

$$F_{beta} = (1 + \beta^2) \frac{precision * recall}{\beta^2 * precision + recall}$$

Thanks to the filters on the left side of the screen, it is possible to change the number of categories displayed, in case the check categories is active, otherwise the type of averaging method.

Analyzer: IoU Curve

Here the graph of the IoU metric is displayed, i.e. Intersection Over Union, which consists

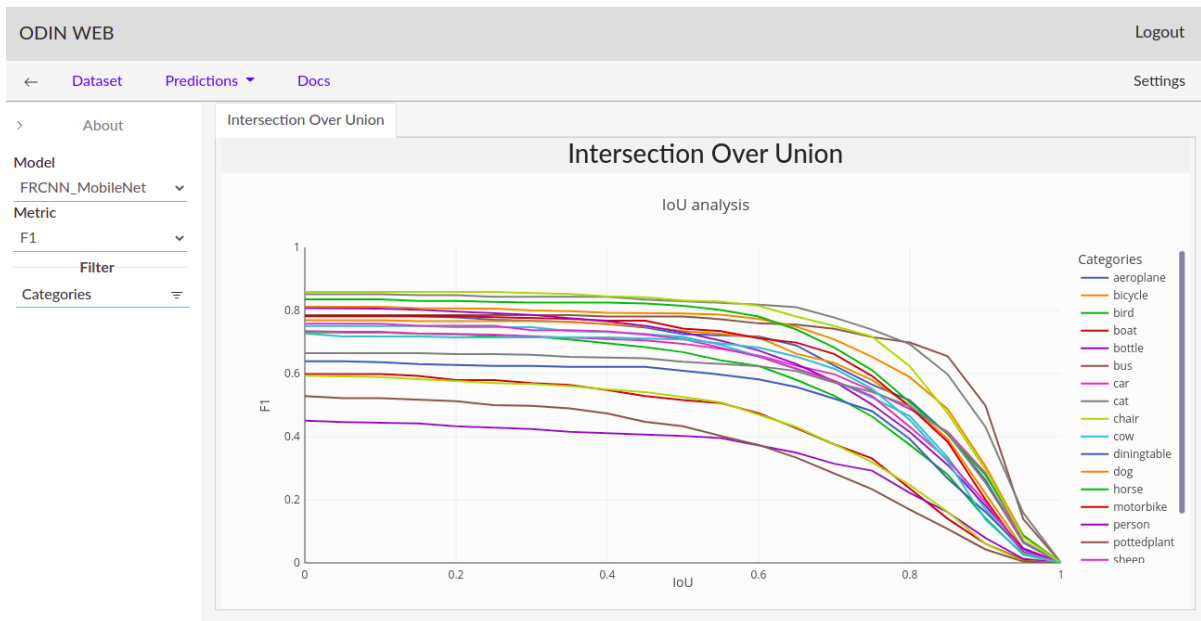


Figure 3.20: Intersection Over Union Curve

of the following formula:

$$IoU = \frac{AreaofOverlap}{AreaofUnion}$$

in this way the user can analyze how precise the model is in the creation of bounding boxes / segments to locate the class within the image. Here too it is possible to analyze the IoU threshold using all the metrics offered by ODIN and selectable by the filter.

Analyzer:Reliability

For each prediction, the classification models assign a value between 0 and 1 called the

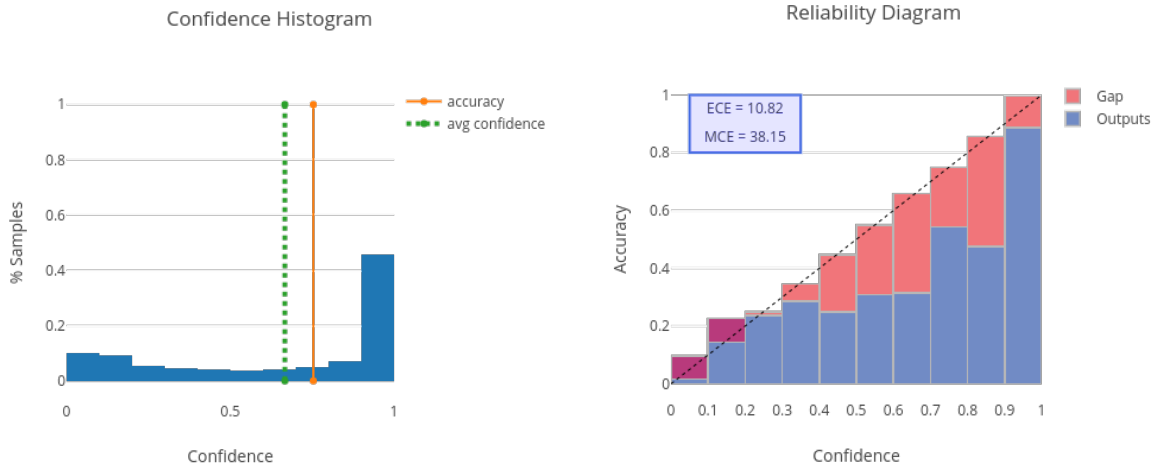


Figure 3.21: Confidence Histogram and Reliability Diagram

confidence score. The plot on the left in the 3.21 image represents the distribution of the confidence scores generated by the model. On the right side, instead, Reliability diagram, which is a visual representation of the model calibration that plot the Expected sample accuracy as a function of the confidence, is visualized. In the second one are also reported:

- *Expected Calibration Error (ECE)*: defined as a weighted average of the difference between the accuracy and confidence difference.
- *Maximum Calibration Error (MCE)*, which indicates the worst-case deviation between confidence and accuracy.

Thanks to the related filter is possible to increment or reduce the *number of bins* which is the number of intervals the confidence scores are grouped into.

Analyzer:Errors

On this page all the analyzes on the prediction errors that the model has committed have

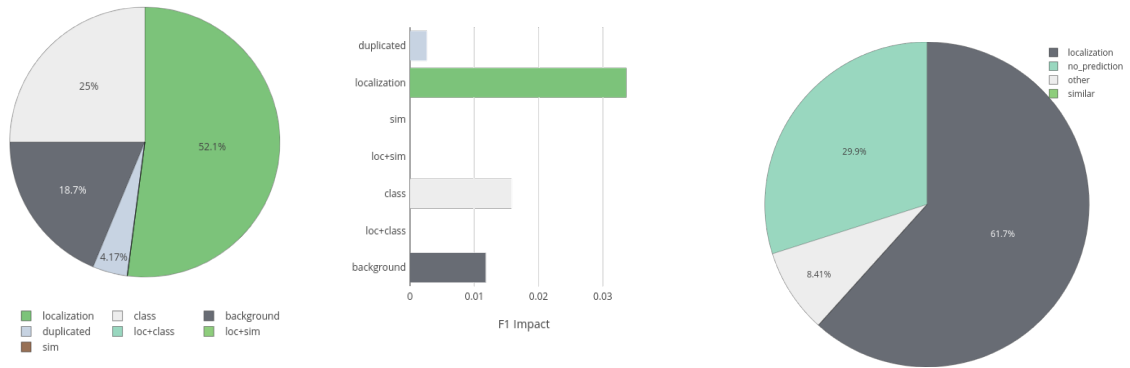


Figure 3.22: Example of False Positive Errors Distribution with their Impact and the False Negative Distribution of Errors

been grouped. These errors are divided into:

- *False-Positive*: it is a result given by the model, in which it incorrectly predicted the presence of a class.
- *False-Negative*: it is a result given by the model, in which it incorrectly predicted the negative class.

as we can see from the Figure 3.22.

The analysis of these errors needs two input parameters that can be chosen through the available filters: the category and the metric used for the analysis. The analyzes are:

- *False-Positive Errors* – this analysis is divided into two plots. The first create the error distribution for that category, instead the error's impact allows the user to understand the weight of the error and how much the model would gain in performance if it were able to completely eliminate that type of error for that class. As regards the errors of the classification tasks, they are divided into:

- *class*: classification error. It is done when the model predicts the wrong class for an observation
- *sim*: similarity error. It is performed when the model assigns an incorrect class but belonging to the group of similar classes in which the correct class is also present.
- *background*: error made when the model assigns a class to an image that does not contain classes.

While the errors for the localization tasks are:

- *duplicated*: caused by the presence of two predictions for the same element in

the image.

- *localization*: the model has identified the correct class but the intersection over union of the bounding box/segmentation mask is less than the IoU-Threshold Strong and greater than the IoU Threshold Weak.
- *sim*: the model correctly generated the bounding box / segmentation mask but confused the class with a similar one.
- *loc + sim*: the model generated the bounding box / segmentation mask with an IoU between the Weak and Strong thresholds and, in addition, confused the class with a similar one.
- *class*: classification error. It is done when the model predicts the wrong class for an annotation
- *loc + class*: the model generated the bounding box / segmentation mask with an IoU between the Weak and Strong thresholds and, in addition, it predicts the wrong class.
- *background*: error for which the model generated a bounding box / segmentation mask with an IoU Threshold lower than n Weak. Often associated with predictions about objects present in the image but not part of one of the classes.
- *False-Negative Errors* – as for false positives, this analysis displays the distribution of false-negative errors committed by the model. These errors are divided into:
 - *Similar*: class associated with an image containing the similar class.
 - *Other*: classification and background errors.

for what concern the classification tasks. In addition to these, for localization tasks are defined also:

- *Localization*: there are no prediction such that the IoU is greater than the IoU Strong threshold.
- *No_prediction*: there are no prediction such that the IoU is at least equal or greater than the IoU Weak threshold.

Analyzer:Report

This page summarizes all the performance scores of the model at each level of granularity. The performances can be summarized as follow:

- *Overall performance*: set of model performances by evaluating the entire data set and providing two different calculation methods, the micro and macro average.
- *Per category performance*: the evaluation metrics are calculated for each category within the data set.
- *Per property performance*: the evaluation metrics are calculated for each property value of those previously loaded with the format *property_value*.

The filters allow the user to view only the essential data, excluding the scores of the categories / properties and the irrelevant metrics at the moment. Furthermore, once all the filters have been set, the user can download the table into a file with the *.xls* extension in order to export it to a spreadsheet application or in Latex⁷ format.

Model Comparator

As already mentioned ODIN allows you to load multiple model predictions at the same time to be able to compare them. The analyzes that allow the comparison between multiple models are:

- *Per-Property Analysis*: generate a graph for each property value in which the scores obtained by the models for that value for a given property are compared.
- *Sensitivity and Impact Analysis*: the comparison of the models is based on the different error types and their impact on the evaluation metric selected.
- *Performance curves*: The selected curve is generated for each model in order to have a clearer view of the performance obtained by them, both overall and by category
- *False-Positive and False Negative Errors*: The distribution of errors associated with a category is plotted in the form of a bar graph in order to compare multiple models at the same time
- *Distributions*: all model distributions are plotted and categorized in order to compare the values obtained by the two models for those distributions.
- *Report*: all the scores for the different metrics obtained by each model for each category and property are reported at the same time

⁷<https://it.overleaf.com/learn/latex/Tables>

4 | ODIN Web In Action

In this chapter we will exploit the functionality of ODIN Web in real cases by analyzing two datasets as regards the analyzer. The first will be ArtDL[33] with a multi-label classification problem, the second one it will be a localization problem on the PASCAL VOC 2007 [14] dataset applied on multiple models, in order to introduce the comparator's functionalities.

4.1. ArtDL

ArtDL dataset is a set of images of paintings by iconography classification, composed of 42 thousand annotated images belonging to a set of 10 classes. This use case exploits a subset of the dataset containing 1864 annotated images belonging to the following classes: *Anthony of Padua*, *John Baptist - Adult*, *Paul*, *Francis of Assisi*, *Mary Magdalena*, *Jerome*, *Saint Dominic*, *Virgin Mary*, *Peter*, and *Saint Sebastian*. For this dataset, the ground truth file was imported. It contains the observations, associated to each image, that includes the categories depicted in the image and, for each observation, 3 meta-annotations:

- *Source* – Name of the museum from which the image was collected. In the case under analysis, the property values are: *beni culturali*, *wga*, *wikidata*, *wikiart*, *gallerix*, *pharos*, *prado*, *met*, *nga*, *rijksmuseum*. This property is already inside in the ArtDL dataset.
- *Characters* – Number of characters present in the painting. They have been categorized as: *0-1*, *2-4*, *5+*. The information related to this property are retrieved using *openpose*[9], a real-time multi-person human pose detection library used to count the number of faces, with a cap of 5 for this dataset, inside the images.
- *Color* – identifies if an image is greyscaled or colored with *bw* and *rgb*. This information is extracted in an automatic way from the image.

When the analyzer is started, the first analysis to be performed is made directly on the dataset used. Figure 4.1 shows the distribution of the various classes in the dataset, showing a presence of the Virgin Mary class with a percentage of 63.8%. This graph

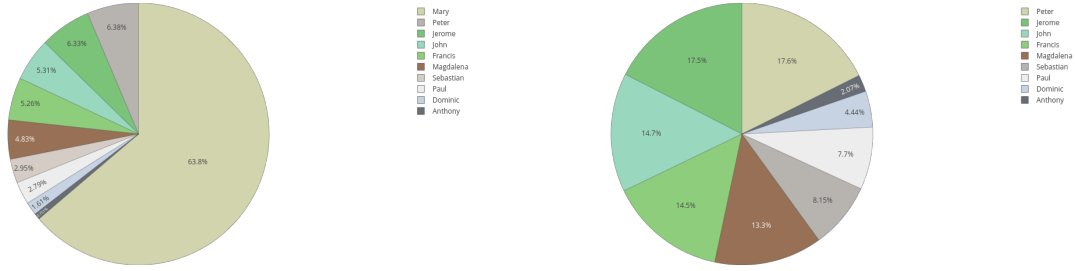


Figure 4.1: Categories distribution of ArtDL dataset

Figure 4.2: Categories distribution of ArtDL dataset without Virgin Mary

denotes a large imbalance in the dataset which makes the performance of the ruling class prevail, in this case Virgin Mary, over those of the lower classes, making the comparison difficult. For this reason, a suitable solution could be to apply some balancing technique, such as weighted loss, undersampling of the majority classes, and oversampling of the minority classes, in order to achieve a fair distribution among categories. Going to filter the prevailing class, as Figure 4.2 shown, it is possible to notice how the remaining classes are equally distributed, except for Anthony Of Padua and Saint Dominic.

In this way, the model of which we are going to analyze the predictions, it is based on a ResNet50 [23] architecture and, in order to overcome the problems of the majority class, an oversampling of the minority classes to the majority class Virgin Mary, has been applied before training.

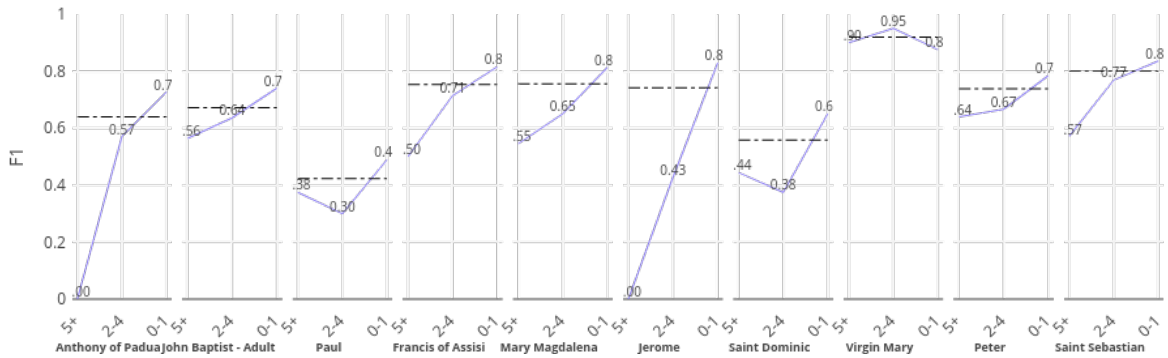


Figure 4.3: Property Analysis of the model

Figure 4.3 shows the performance evaluation of the character property on the model. By analyzing the model performances, it can be seen that has achieved better performance in cases where the painting has a number of characters less than 5. This behavior can be justified by the fact that the presence of more characters within the image leads to have images in which the only feature that can be seen is the face, losing those details which

often helps the model to recognize the class. On the other hand, the presence of few characters inside the image (0-1 or 2-4 cases) allows the image to better represents the saint and, therefore, most of its iconographic symbols are visible. Figure 4.4 presents the

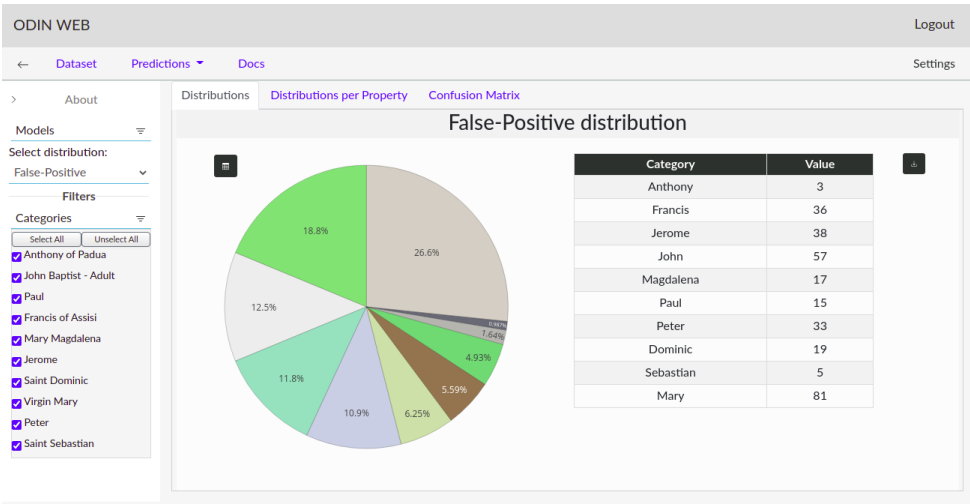


Figure 4.4: False-Positive distribution for the model on ArtDL dataset

False-Positive distribution of the model. Analyzing it, it is clear that the class that scored the highest number of classification errors, in addition to Virgin Mary, is the one associated with John Baptist -Adult with a percentage of 18.8 % on total errors. This advises us that this class could be a potential weakness of our model. In this way, inspecting the

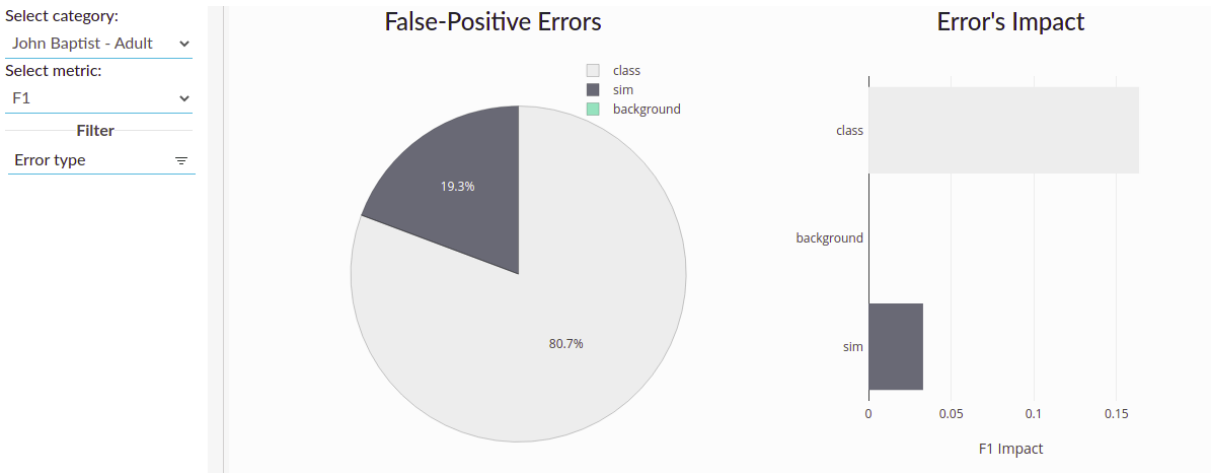


Figure 4.5: False-Positive Errors And Error's Impact of class John Baptist - Adult

False-Positive's errors distribution shown in Figure4.5, it is clear that the John Baptist - Adult class heavily affects the performance of the model. As most of the errors are of *classification* nature, this means that the model failed to highlight any unique characteristics of the class making it a weakness of the model and suggesting to focus the next

efforts on the improvement of this class first. A solution could be to enrich the dataset with more representations of the class in order to help the model grasp the discriminative features of it.

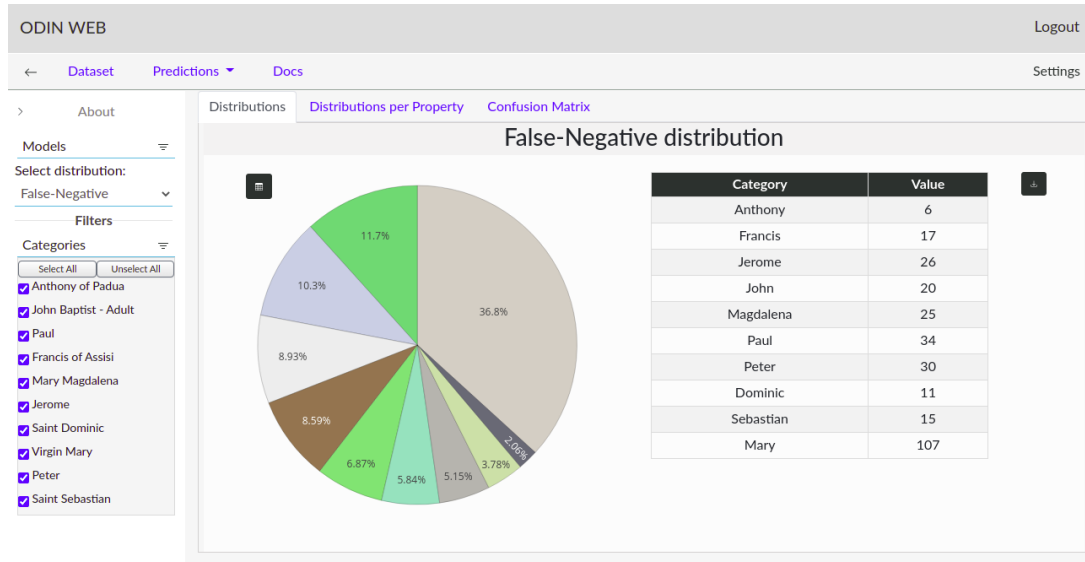


Figure 4.6: False-Negative distribution for the model on ArtDL dataset

Inspecting the distribution of False-Negatives, Figure 4.6 shows Virgin Mary has, also, the higher number of False-Negative errors, which could be caused by other women present in the image and classified as Mary, or by images that have incorrect annotations. But, the most interesting data concerns the Paul class, which scores with a percentage of 11.7 % on total errors despite being one of the least present classes within the dataset as seen in Figure 4.1 (about 2.8%). The reason for this high number of errors is due to the fact that, in this dataset, there are similar classes and, thanks to the ODIN Web settings, it is possible to specify them in order to have a clearer overview of the error types.

Similar classes present in this datasets are setted in setting as it shows Figure 4.7. Figure 4.8 represents the distribution of the error types over the two classes taken into account. It is possible to see how making similar classes explicit revealed that almost 20% of the model errors attributed to the Peter class are caused by similarity with the Jerome and Paul classes. In the same way, the Maria Magdalena class has a high number of False-negatives, mainly caused by its resemblance to the Virgin Mary class. The analysis suggests to apply some Fine-Grained Visual Categorization(FGVC) Techniques to help the model focus on the subtle iconographic symbols that are unique for each class.

Table 4.1 is the summary of evaluation score obtained by the model. As we can see, the

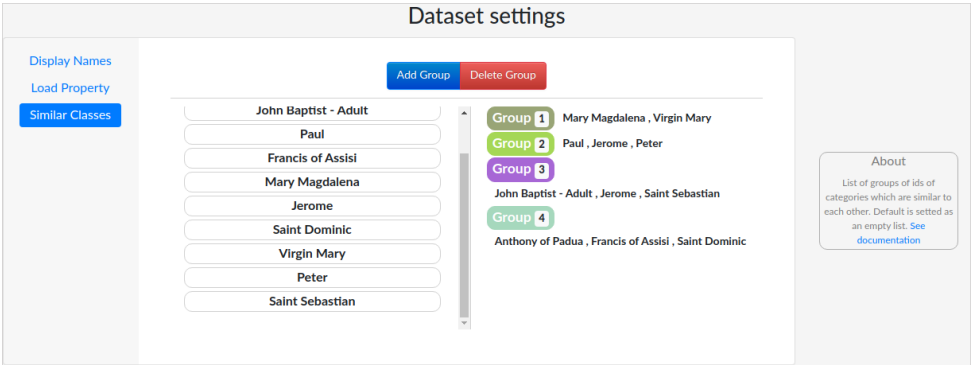


Figure 4.7: Similar Classes Loaded in the tool

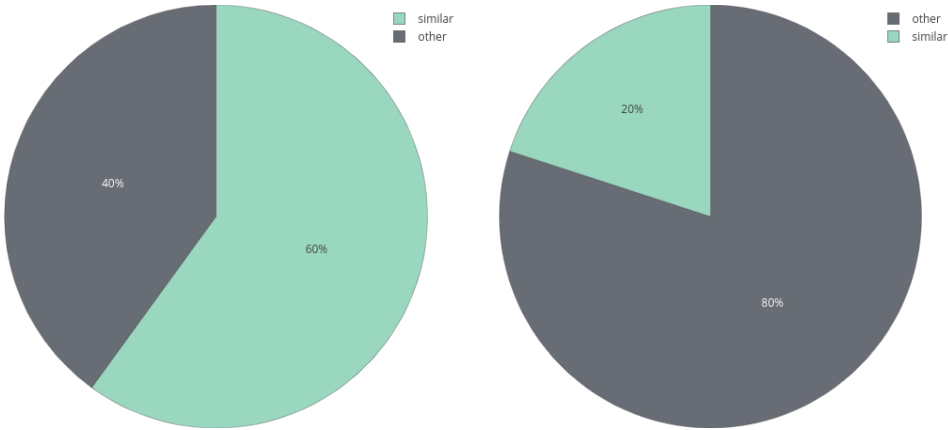


Figure 4.8: Mary Magdalena and Peter False Negative errors with similar classes

overall performances of the model are good. However, analyzing the individual scores obtained from the categories and properties, some of the considerations made previously are confirmed. In fact, as regards the character properties, it is clearly visible that as the number of characters increases, the performance decreases considerably.

| type | label | Precision | Recall | F1 | Average_Precision | ece | mce |
|----------|------------------------|-----------------|------------------|------------------|-------------------|---------------------|---------------------|
| Total | avg macro avg micro | 0.7095 0.838 | 0.7062 0.8443 | 0.7005 0.8412 | 0.7087 0.8692 | not supp. 0.1088 | not supp. 0.3815 |
| Category | Anthony of Padua | 0.7273 | 0.5714 | 0.64 | 0.648 | 0.1679 | 0.7407 |
| | John Baptist - Adult | 0.5809 | 0.798 | 0.6723 | 0.7492 | 0.1632 | 0.5562 |
| | Paul | 0.5455 | 0.3462 | 0.4235 | 0.3684 | 0.1251 | 0.896 |
| | Francis of Assisi | 0.6923 | 0.8265 | 0.7535 | 0.7482 | 0.1485 | 0.6354 |
| | Mary Magdalena | 0.7927 | 0.7222 | 0.7558 | 0.7903 | 0.1343 | 0.6875 |
| | Jerome | 0.7077 | 0.7797 | 0.7419 | 0.7727 | 0.1652 | 0.6612 |
| | Saint Dominic | 0.5 | 0.6333 | 0.5588 | 0.509 | 0.1224 | 0.755 |
| | Virgin Mary | 0.9304 | 0.91 | 0.9201 | 0.9647 | 0.0807 | 0.3528 |
| | Peter | 0.7295 | 0.7479 | 0.7386 | 0.757 | 0.1484 | 0.8573 |
| | Saint Sebastian | 0.8889 | 0.7273 | 0.8 | 0.7799 | 0.1148 | 0.5461 |
| | Property | | | | | | |
| | source_wikidata | 0.6737 | 0.665 | 0.668 | 0.7129 | not supp. | not supp. |
| | source_wga | 0.5732 | 0.4603 | 0.4945 | 0.6252 | not supp. | not supp. |
| | source_gallerix | 0.6467 | 0.6376 | 0.633 | 0.6696 | not supp. | not supp. |
| | source_beni_culturali | 0.546 | 0.595 | 0.5631 | 0.5698 | not supp. | not supp. |
| | source_rijksmuseum | 0.3 | 0.3 | 0.3 | 0.5125 | not supp. | not supp. |
| | source_prado | 0.5417 | 0.575 | 0.5514 | 0.5396 | not supp. | not supp. |
| | source_wikiart | 0.5667 | 0.4362 | 0.4845 | 0.6786 | not supp. | not supp. |
| | source_net | 0.55 | 0.55 | 0.5333 | 0.51 | not supp. | not supp. |
| | source_nga | 0.5 | 0.5 | 0.5 | 0.5 | not supp. | not supp. |
| | source_pharos | 0.335 | 0.34 | 0.2993 | 0.4551 | not supp. | not supp. |
| | color_bw | 0.5621 | 0.6038 | 0.5689 | 0.5961 | not supp. | not supp. |
| | color_rgb | 0.7737 | 0.7458 | 0.7565 | 0.7671 | not supp. | not supp. |
| | characters_0-1 | 0.8041 | 0.7326 | 0.7564 | 0.7783 | not supp. | not supp. |
| | characters_2-4 | 0.6133 | 0.6694 | 0.6063 | 0.65 | not supp. | not supp. |
| | characters_5+ | 0.4152 | 0.5486 | 0.4541 | 0.4741 | not supp. | not supp. |

Table 4.1: Base Report Analysis for ArtDL Dataset

4.2. PASCAL VOC 2007

PASCAL VOC 2007 is a dataset for image recognition consisting of 9963 images containing 24640 annotated objects. This dataset is time-honored to evaluate performance in object category detection. For this analysis the test set of PASCAL VOC 2007, which comprises of 4952 images, is treated. The classes taken into account for this analysis and belonging to the dataset are:

- Person: person
- Animal: bird, cat, cow, dog, horse, sheep
- Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

For each annotation, a set of properties are placed side by side, namely:

- AreaSize: is the dimension of its bounding box. Values are: XS (extra-small), S (small), M (medium), L (large), and XL (extra-large).
- AspectRatio: is defined as object width divided by object height, computed from the PASCAL VOC 2007 bounding box annotation. Similarly to object size, objects are categorized into extra-tall (XT), tall (T), medium (M), Wide (W), and extra-wide (XW), using the same percentiles.

In this analysis, we compare two models: Faster-RCNN (with MobileNet backbone) and the same with the addition of the Feature Pyramid Network (FPN). The weights of the models are obtained from public repositories trained with COCO dataset (since this dataset also contains the PASCAL VOC). The models are compared in ODIN Web to catch the strengths and weaknesses of each.

Figure 4.9 reports the Per-Property analysis made over the Area Size property for the Faster-RCNN model. It is possible to notice how the performances decrease as the size of the object decreases. One possible solution is to adopt architectural improvement techniques to also allow small object detection, such as the one proposed in [8]. Another hint, in case the researcher wants to trade some performance aspects with the recognition of the presence of particularly small objects, it could be adopted a different type of model, for example, a Feature Pyramid Network (FPN).

The performances of the FPN model are depicted in Figure 4.10 and, as expected, it presents higher performance for annotations associated with properties values, such as XS

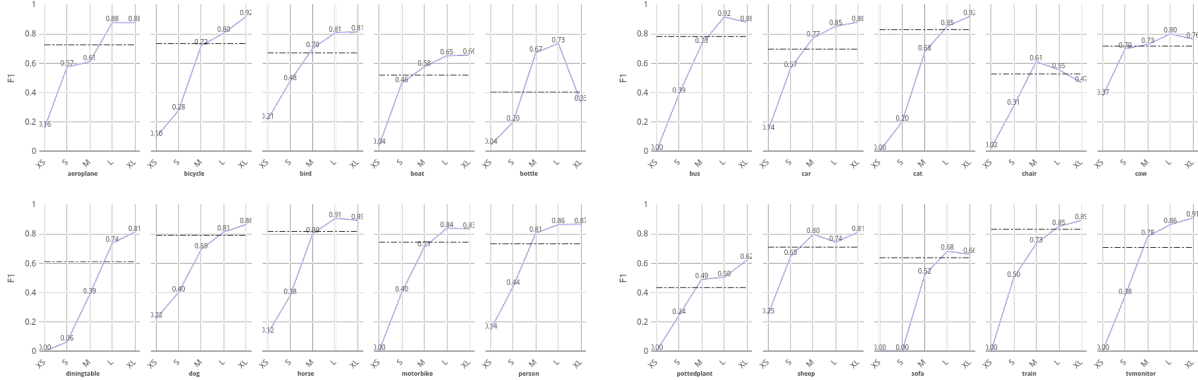


Figure 4.9: Per-Property Analysis on Faster-RCNN model

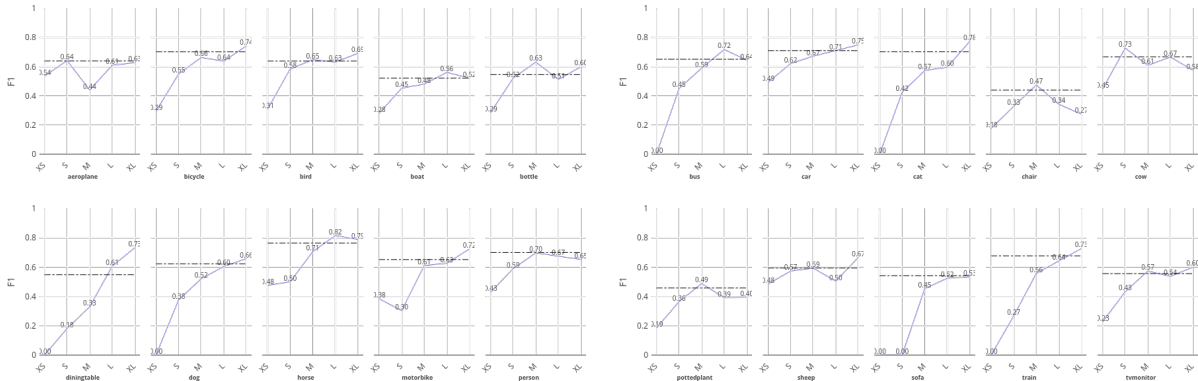


Figure 4.10: Per-Property Analysis on FPN model

and S, but significantly decreasing performance in cases where the size is larger proving that, for this particular model, performances are less affected by the size of the bounding boxes. Figure 4.11 represents the sensitivity and impact graph that compares the two models simultaneously. As already said, it is possible to see how FPN is much less sensitive and with a lower impact than Faster-RCNN, while as regards aspect ratio property the two models have similar behaviors. From this analysis, we can conclude that the FPN model would be a valid substitute for the Faster-RCNN if the researcher is willing to give up to 6.0% of the overall performances (from 68 to 62) while having a model that recognizes all types of bounding boxes.

By analyzing Figure 4.12, it reports the reliability diagram applied to the two models separately, we can see how, unlike the FPN model, the Faster-RCNN model is not well calibrated. In this case, the model is underestimating its predictions, presenting an expected calibration value of 11.37. This condition can be improved by applying some calibration technique such as Bayesian Binning into Quantiles, Platt scaling, or its simplest version, temperature scaling.

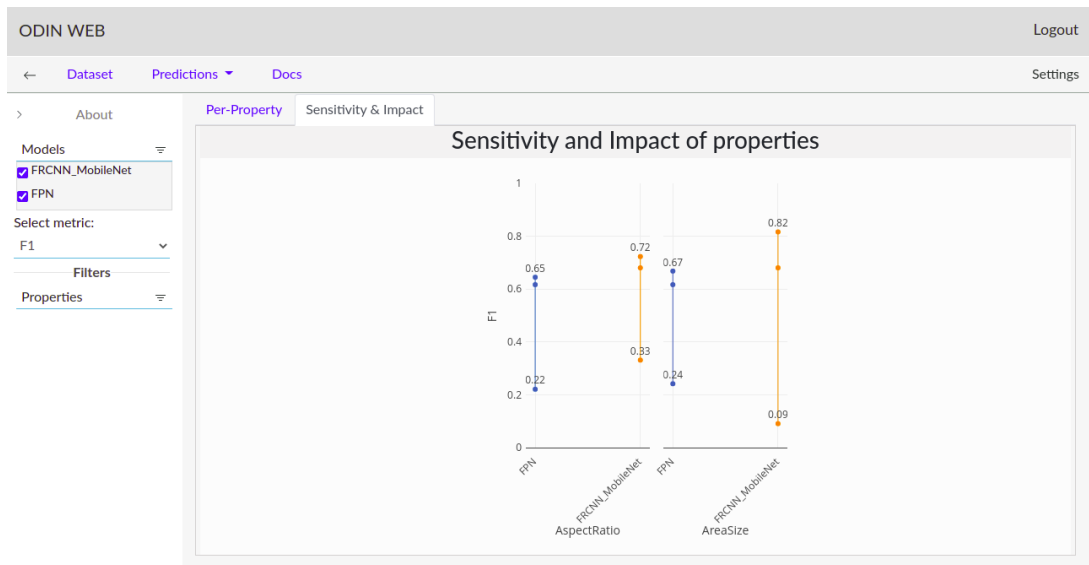


Figure 4.11: Sensitivity and Impact of the FPN and Faster-RCNN compared

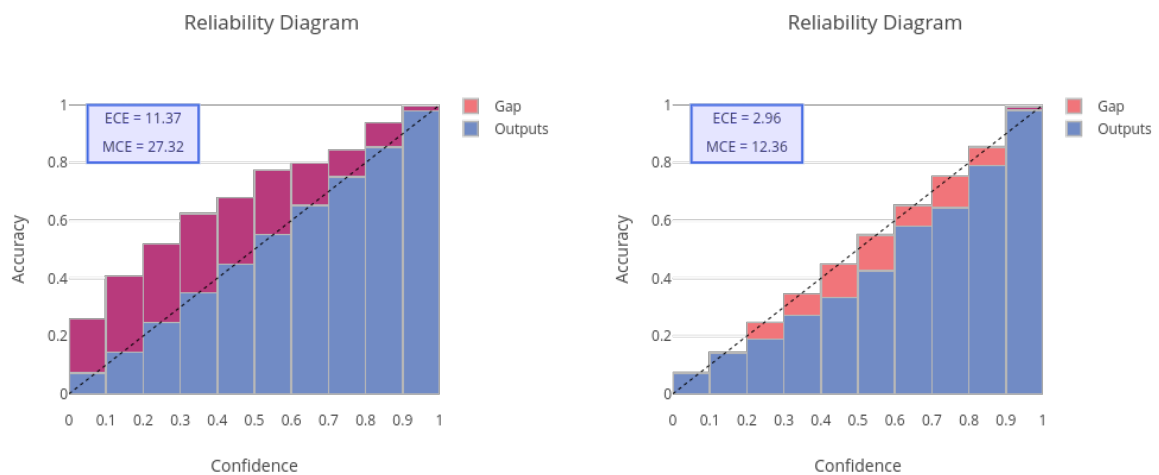


Figure 4.12: Reliability Diagram of the Faster-RCNN(left) and FPN(right)

Focusing on the FPN model, we can notice that it is well calibrated with an expected calibration value of 2.96 while overestimating its predictions. This behavior is also noted by analyzing the distribution of False-Positives, in 4.13, between the two models, in which FPN commits almost twice as many errors compared to Faster-RCNN. Therefore, It may be useful to understand what are the types of false-positive errors committed by FPN to find optimizations to further improve its performance.

What emerges from the analysis, and depicted in Figure 4.14, is that many classes within the dataset are confused by the model due to their similarity, impacting on performances.

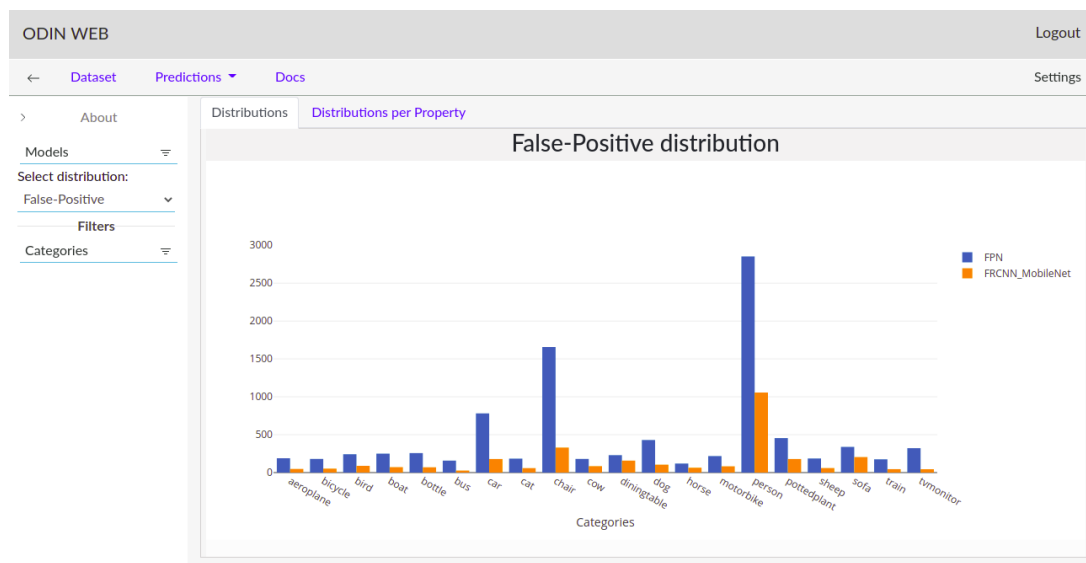


Figure 4.13: False-Positive distribution of the Faster-RCNN and FPN compared

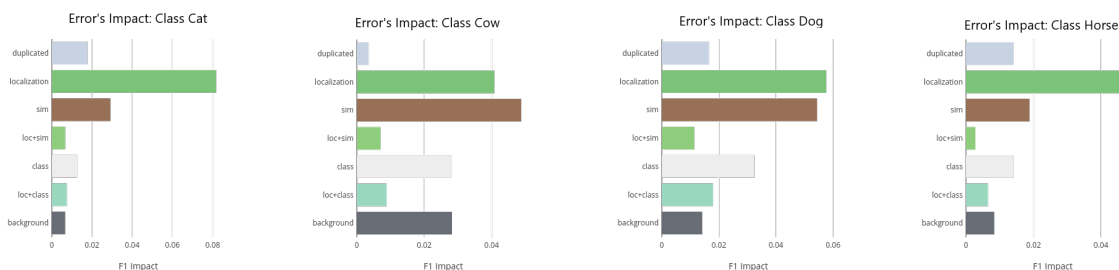


Figure 4.14: False-Positive Error's Impact of cat, cow, dog, and horse classes in FPN model

In this example we have taken only one case of similar classes referring to classes: cat, cow, dog, and horse. Therefore, it would be useful to apply fine-grain classification techniques or Loss functions in order to better discriminate against similar classes. For example, by applying entropy into loss term, it is possible to achieve better performance for multi similar object pairs as well as single similar object pairs as demonstrated in [28].

5 | Conclusions and Future work

In this work, ODIN Web, a web app that aims to extend the already existing ODIN framework, is presented. ODIN Web offers the possibility to exploit all the functionalities provided by ODIN simply and without any programming effort. During this project two of the main features of ODIN were provided with a graphical interface: the Annotator can guide the user in a simple but effective way during the entire phase of the dataset population, offering a clean and user-friendly interface. In this way, they can create or add new annotations, for each created/imported dataset facilitating them in the management of the ground truth file that will be automatically generated in the correct format (based on the MS COCO format). The annotation phase supports both classification, through labels, and localization tasks, through bounding boxes and segmentation masks, with an interactive editor in which is possible to modify the annotations related to the dataset. As for the analyzer, it provides all the analyzes implemented by ODIN Framework, allowing the user to view the results in graphical format and providing all the filters necessary to view only the data of interest. It also allows you to compare multiple models at the same time and update the analysis parameters in real-time. As for future works, we expect to expand ODIN Web with features already present in the framework, such as:

- implement the Visualizer component to give the user the ability to fully explore the ground truth and predictions of his model for both computer vision tasks.
- allow the annotator to support textual observations in the annotation phase
- expansion of the types of meta-annotations offered by the annotator: range of values and textual parameters.
- add support analysis for Class Activation Maps (CAMs), both from a quantitative and qualitative point of view.
- implement automatic extraction of certain types of properties such as: object-count and colors.

In conclusion, it has been demonstrated that the current state of the art in the field of black-box analysis frameworks there are very few tools that allow having an interactive

interface and at the same time offer the user the complete customization of the data and the analyzes performed on them. Therefore, we think that ODIN Web could be a considerable step forward that can allow anyone to interface with the world of computer vision in a simple way but, at the same time, can also support researchers, and not, to extract multiple and useful information for the refinement of their models.

Bibliography

- [1] Vott (visual object tagging tool), 2017.
- [2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.
- [3] B. Alsallakh, A. Hanbury, H. Hauser, S. Miksch, and A. Rauber. Visual methods for analyzing probabilistic classification data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1703–1712, 2014. doi: 10.1109/TVCG.2014.2346660.
- [4] H. Alwassel, F. C. Heilbron, V. Escorcia, and B. Ghanem. Diagnosing error in temporal action detectors. *ArXiv*, abs/1807.10706, 2018.
- [5] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh. Model-tracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI ’15, page 337–346, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450331456. doi: 10.1145/2702123.2702509. URL <https://doi.org/10.1145/2702123.2702509>.
- [6] D. Bolya, S. Foley, J. Hays, and J. Hoffman. Tide: A general toolbox for identifying object detection errors. In *ECCV*, 2020.
- [7] J. Brooks. COCO Annotator. <https://github.com/jsbroks/coco-annotator/>, 2019.
- [8] C. Cao, B. Wang, W. Zhang, X. Zeng, X. Yan, Z. Feng, Y. Liu, and Z. Wu. An improved faster r-cnn for small object detection. *IEEE Access*, 7:1–1, 08 2019. doi: 10.1109/ACCESS.2019.2932731.
- [9] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

- [10] J. M. Cejuela, P. McQuilton, L. Ponting, S. J. Marygold, R. Stefancsik, G. H. Millburn, B. Rost, and the FlyBase Consortium. tagtog: interactive and text-mining-assisted annotation of gene mentions in PLOS full-text articles. *Database*, 2014, 04 2014. ISSN 1758-0463. doi: 10.1093/database/bau033. URL <https://doi.org/10.1093/database/bau033>. bau033.
- [11] W. Chen, Wei-Teand Styler. Anafora: A web-based general purpose annotation tool. In *Proceedings of the 2013 NAACL HLT Demonstration Session*, pages 14–19. Association for Computational Linguistics, 2013. URL <http://www.aclweb.org/anthology/N13-3004>.
- [12] A. Demidovskij, Y. Gorbachev, M. Fedorov, I. Slavutin, A. Tugarev, M. Fatekhov, and Y. Tarkan. Openvino deep learning workbench: Comprehensive analysis and tuning of neural networks inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 783–787, 2019. doi: 10.1109/ICCVW.2019.00104.
- [13] A. Dutta and A. Zisserman. The VIA annotation software for images, audio and video. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM ’19, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6889-6/19/10. doi: 10.1145/3343031.3350535. URL <https://doi.org/10.1145/3343031.3350535>.
- [14] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. <http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html>.
- [15] P. Fraternali, F. Milani, R. N. Torres, and N. Zangrando. Black-box error diagnosis in deep neural networks, 2021. URL <https://arxiv.org/pdf/2201.06444.pdf>.
- [16] M. Gleicher, A. Barve, X. Yu, and F. Heimerl. Boxer: Interactive comparison of classifier results. *Computer Graphics Forum*, 39, 2020.
- [17] Y. Guo, Y. Liu, T. Georgiou, and et al. A review of semantic segmentation using deep neural networks. In *International Journal of Multimedia Information Retrieval* 7, pages 87–93, 2018. URL <https://doi.org/10.1007/s13735-017-0141-z>.
- [18] Y. Guo, Y. Liu, T. Georgiou, and M. S. Lew. A review of semantic segmentation using deep neural networks. In *International Journal of Multimedia Information Retrieval*, pages 87–93, 2018. ISBN 2192-662X. doi: 10.1007/s13735-017-0141-z. URL <https://doi.org/10.1007/s13735-017-0141-z>.
- [19] A. K. Gupta and et. al. Imglab, 2018.

- [20] A. Hafiz and G. Bhat. A survey on instance segmentation: state of the art. In *International Journal of Multimedia Information Retrieval* 9, pages 171–189, 2020. URL <https://doi.org/10.1007/s13735-020-00195-x>.
- [21] A. M. Hafiz and G. M. Bhat. A survey on instance segmentation: state of the art. In *International Journal of Multimedia Information Retrieval*, pages 171–189, 2020. ISBN 2192-662X. doi: 10.1007/s13735-020-00195-x. URL <https://doi.org/10.1007/s13735-020-00195-x>.
- [22] C. Halaschek-wiener, J. Golbeck, A. Schain, M. Grove, B. Parsia, and J. Hendler. Photostuff-an image annotation tool for the semantic web. 01 2005.
- [23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.
- [24] J. Heaton, N.G.Polson, and J.H.White. Deep learning in finance, 2016. URL <https://arxiv.org/pdf/1602.06561.pdf>.
- [25] S. Herath, M. Harandi, and F. Porikli. Going deeper into action recognition: A survey. *Image and Vision Computing*, 60:4–21, 2017. ISSN 0262-8856. doi: <https://doi.org/10.1016/j.imavis.2017.01.010>. URL <https://www.sciencedirect.com/science/article/pii/S0262885617300343>. Regularization Techniques for High-Dimensional Data Analysis.
- [26] D. Hoiem, Y. Chodpathumwan, and Q. Dai. Diagnosing error in object detectors. *Computer Vision–ECCV 2012*, pages 340–353, 2012.
- [27] Y. Huang, J. Zhang, Y. Yang, Z. Gong, and Z. Hao. Gnnvis: Visualize large-scale data by learning a graph neural network representation. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management, CIKM ’20*, page 545–554, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368599. doi: 10.1145/3340531.3411987. URL <https://doi.org/10.1145/3340531.3411987>.
- [28] M. Ju, S. Moon, and C. D. Yoo. Object detection for similar appearance objects based on entropy. In *2019 7th International Conference on Robot Intelligence Technology and Applications (RiTA)*, pages 192–198, 2019. doi: 10.1109/RITAPP.2019.8932791.
- [29] J. Ker, L. Wang, J. Rao, and T. Lim. Deep learning applications in medical image analysis, 2017. URL <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8241753>.

- [30] 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*, CHI '16, pages 5686–5697, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3362-7. doi: 10.1145/2858036.2858529. URL <http://doi.acm.org/10.1145/2858036.2858529>.
- [31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.
- [32] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Zitnick. Microsoft coco: Common objects in context. volume 8693, 04 2014. ISBN 978-3-319-10601-4. doi: 10.1007/978-3-319-10602-1_48.
- [33] F. Milani and P. Fraternali. A dataset and a convolutional model for iconography classification in paintings. *Journal on Computing and Cultural Heritage (JOCH)*, 14(4):1–18, 2021.
- [34] M. neves and J. Seva. Annotationsaurus: A searchable directory of annotation tools, 10 2020.
- [35] P. V. Ogren. Knowtator: a protégé plug-in for annotated corpus construction. In *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 273–275, Morristown, NJ, USA, 2006. Association for Computational Linguistics. doi: <http://dx.doi.org/10.3115/1225785.1225791>.
- [36] Y. Okugawa, M. Kubo, H. Sato, and V. Bui. Evaluation for the synchronization of the parade with openpose. *Proceedings of International Conference on Artificial Life and Robotics*, 24:443–446, 01 2019. doi: 10.5954/ICAROB.2019.OS17-2.
- [37] W. Ouyang, X. Chu, and X. Wang. Multi-source deep learning for human pose estimation. pages 2337–2344, 06 2014. doi: 10.1109/CVPR.2014.299.
- [38] T. Perry. LightTag: Text annotation platform. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 20–27, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. URL <https://aclanthology.org/2021.emnlp-demo.3>.

- [39] S. Pyysalo, T. Ohta, R. Rak, D. Sullivan, C. Mao, C. Wang, B. Sobral, J. Tsujii, and S. Ananiadou. Overview of the id, epi and rel tasks of bionlp shared task 2011. *BMC bioinformatics*, 13 Suppl 11:S2, 06 2012. doi: 10.1186/1471-2105-13-S11-S2.
- [40] 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, 2017. doi: 10.1109/TVCG.2016.2598828.
- [41] H. Saif and H. Alani. Alleviating data sparsity for twitter sentiment analysis. *CEUR Workshop Proceedings*, 838, 01 2012.
- [42] F. Seide. Keynote: The computer science behind the microsoft cognitive toolkit: An open source large-scale deep learning toolkit for windows and linux. In *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages xi–xi, 2017. doi: 10.1109/CGO.2017.7863722.
- [43] B. Sekachev, N. Manovich, M. Zhiltsov, A. Zhavoronkov, D. Kalinin, B. Hoff, T. Osmannov, D. Kruchinin, A. Zankevich, DmitriySidnev, M. Markelov, Johannes222, M. Chenuet, a andre, telenachos, A. Melnikov, J. Kim, L. Ilouz, N. Glazov, Priya4607, R. Tehrani, S. Jeong, V. Skubriev, S. Yonekura, vugia truong, zliang7, lizhming, and T. Truong. opencv/cvat: v1.1.0, Aug. 2020. URL <https://doi.org/10.5281/zenodo.4009388>.
- [44] S. Squartini and L. G. et al. Riconoscimento acustico del fondo stradale attraverso algoritmi di deep learning. URL <https://web.uniroma1.it/et2018/sites/default/files/memorie/Squartini.pdf>.
- [45] M. Tkachenko, M. Mikhail, S. Nikita, H. Andrey, and L. Nikolai. Label studio: Data labeling software, 2020-2021. URL <https://github.com/heartexlabs/label-studio>. Open source software available from <https://github.com/heartexlabs/label-studio>.
- [46] R. N. Torres, P. Fraternali, and J. Romero. Odin: An object detection and instance segmentation diagnosis framework. In A. Bartoli and A. Fusiello, editors, *Computer Vision – ECCV 2020 Workshops*, pages 19–31, Cham, 2020. Springer International Publishing. ISBN 978-3-030-65414-6.
- [47] R. N. Torres, F. Milani, and P. Fraternali. Odin: Pluggable meta-annotations and metrics for the diagnosis of classification and localization. In G. Nicosia, V. Ojha, E. La Malfa, G. La Malfa, G. Jansen, P. M. Pardalos, G. Giuffrida, and R. Umeton,

- editors, *Machine Learning, Optimization, and Data Science*, pages 383–398, Cham, 2022. Springer International Publishing. ISBN 978-3-030-95467-3.
- [48] 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, 2020. doi: 10.1109/TVCG.2019.2934619.
 - [49] S. Xiang, X. Ye, J. Xia, J. Wu, Y. Chen, and S. Liu. Interactive correction of mislabeled training data. In *2019 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 57–68, 2019. doi: 10.1109/VAST47406.2019.8986943.
 - [50] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi. Convolutional neural networks: an overview and application in radiology. In *Insights into Imaging*, pages 611–629, 2018. URL <https://doi.org/10.1007/s13244-018-0639-9>.
 - [51] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4):13–es, dec 2006. ISSN 0360-0300. doi: 10.1145/1177352.1177355. URL <https://doi.org/10.1145/1177352.1177355>.
 - [52] H. Yoon, S.-H. Lee, and M. Park. Tensorflow with user friendly graphical framework for object detection api. 06 2020.
 - [53] 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 <http://hdl.handle.net/10589/177405>.
 - [54] 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, 2019. doi: 10.1109/TVCG.2018.2864499.

List of Figures

| | | |
|------|--|----|
| 2.1 | Image Classification | 7 |
| 2.2 | Object Detection | 7 |
| 2.3 | Semantic Segmentation | 7 |
| 2.4 | Instance Segmentation | 7 |
| 2.5 | Pose Estimation, source: [36] | 7 |
| 2.6 | Architecture of Krizhevsky et al.'s DCNN [31] | 8 |
| 2.7 | Sentiment Annotation. Source [41] | 13 |
| 2.8 | Semantic Classification. Source: Link | 13 |
| 2.9 | Entity Annotation. Source [39] | 13 |
| 3.1 | ODIN Web architecture | 30 |
| 3.2 | UML Class Diagram representing the main components | 36 |
| 3.3 | UML of the Analyzer Component | 38 |
| 3.4 | UML of the Annotator Component | 40 |
| 3.5 | MongoDB Observation Structure | 42 |
| 3.6 | MongoDB Annotation Structure | 42 |
| 3.7 | ODIN Web Interface: create new dataset | 43 |
| 3.8 | ODIN Web Interface: import dataset | 44 |
| 3.9 | Landing page of the annotator | 45 |
| 3.10 | Annotator page for Classification tasks | 46 |
| 3.11 | Annotator page for Object Detection(left) and Instance Segmentation(right) | 47 |
| 3.12 | Dataset Selection page | 48 |
| 3.13 | Analyzer Flow | 49 |
| 3.14 | Basic structure of each Analyzer Tab | 50 |
| 3.15 | Sequence Diagram of the plot creation | 50 |
| 3.16 | Tool Interface for Dataset analyzes (Top), Distribution of Property, Distribution of Property for Categories and Categories Co-occurrence Matrix | 51 |
| 3.17 | Per property (on left) and Sensitivity and Impact (on right) analyzes | 52 |
| 3.18 | Confusion Matrix Interface with filter on Character Property | 54 |
| 3.19 | Accuracy of Top-1 and Top-5 for Property prop1 | 54 |

| | | |
|------|--|----|
| 3.20 | Intersection Over Union Curve | 55 |
| 3.21 | Confidence Histogram and Reliability Diagram | 56 |
| 3.22 | Example of False Positive Errors Distribution with their Impact and the False Negative Distribution of Errors | 57 |
| 4.1 | Categories distribution of ArtDL dataset | 62 |
| 4.2 | Categories distribution of ArtDL dataset without Virgin Mary | 62 |
| 4.3 | Property Analysis of the model | 62 |
| 4.4 | False-Positive distribution for the model on ArtDL dataset | 63 |
| 4.5 | False-Positive Errors And Error's Impact of class John Baptist - Adult . . | 63 |
| 4.6 | False-Negative distribution for the model on ArtDL dataset | 64 |
| 4.7 | Similar Classes Loaded in the tool | 65 |
| 4.8 | Mary Magdalena and Peter False Negative errors with similar classes . . . | 65 |
| 4.9 | Per-Property Analysis on Faster-RCNN model | 68 |
| 4.10 | Per-Property Analysis on FPN model | 68 |
| 4.11 | Sensitivity and Impact of the FPN and Faster-RCNN compared | 69 |
| 4.12 | Reliability Diagram of the Faster-RCNN(left) and FPN(right) | 69 |
| 4.13 | False-Positive distribution of the Faster-RCNN and FPN compared | 70 |
| 4.14 | False-Positive Error's Impact of cat, cow, dog, and horse classes in FPN model | 70 |

List of Tables

| | | |
|-----|---|----|
| 2.1 | Tools list ordered by ascending publication year. Adapted from [15] | 9 |
| 2.2 | Annotators list ordered by ascending publication year | 16 |
| 2.3 | Technical criteria of the annotation tool | 18 |
| 2.4 | Functional criteria of the annotation tool | 19 |
| 4.1 | Base Report Analysis for ArtDL Dataset | 66 |

Ringraziamenti

Questo messaggio è rivolto alle persone più importanti della mia vita: la mia famiglia.

Voglio ringraziare i miei genitori e i miei nonni che mi hanno dato l'opportunità di proseguire gli studi e di arrivare a questo punto.

Vi ringrazio per essermi stati vicini e avermi sempre supportato anche nei momenti in cui pensavo di non potercela fare.

Grazie per avermi spronato e reso la persona che sono oggi.

Grazie Giulia, per essermi stata vicino nel periodo più importante della mia vita e avermi supportato nei momenti più bui di questi ultimi anni, sei stata il regalo più grande.

Vi ringrazio, questo traguardo è dedicato a tutti voi.

