Semi-automatic Image Tagging for ML Workflows: a GUI for Industrial Applications

Vasco Luís Teixeira Costa

Instituto Superior de Engenharia do Porto

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

2025

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Projeto/Estágio, do 3º ano, da Licenciatura em Engenharia Eletrotécnica e de Computadores

Candidate: Vasco Luís Teixeira Costa, Nº 1211139, 1211139@isep.ipp.pt

Scientific Guidance: Fernando Jorge Soares Carvalho, fjc@isep.ipp.pt

Internship Company: Gislotica Projeto e Fabrico de Sistemas Mecanicos

Intership Supervisor: Lobinho Gomes, lobinho.gomes@gislotica.pt

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

20 de janeiro de 2025

Agradecimentos

Com a conclusão deste projeto e desta etapa significativa, gostaria de expressar minha profunda gratidão ao Engenheiro Lobinho Gomes pela orientação e pela oportunidade de trabalhar em um projeto tão desafiador. Agradeço também pelo apoio constante e por ter sido acolhido em uma empresa de excelência.

Um agradecimento especial ao Engenheiro Fernando Carvalho pela orientação atenta, sempre disponível para me auxiliar em todas as dificuldades encontradas. Sua dedicação e empenho foram fundamentais para o meu sucesso.

Também quero agradecer aos meus colegas que compartilharam comigo os momentos de trabalho na empresa, tornando as pausas nos matraquilhos momentos inesquecíveis.

Por fim, agradeço à minha família e à minha namorada por sempre me incentivarem a ir mais longe, e por todo o apoio ao longo do meu percurso acadêmico.

Resumo

Este projeto tem como objetivo o desenvolvimento de uma interface gráfica que permita a seleção de um conjunto de subimagens a partir de uma imagem original, registrando suas coordenadas, largura e altura. Além disso, possibilita o treinamento dessas imagens com base na utilização do Darknet, que emprega o YOLOv4, permitindo o treinamento de um dataset que, após concluído, pode ser utilizado para testar imagens não etiquetadas.

O projeto é dividido em três fases. A primeira consiste na implementação da parte da interface que não envolve inteligência artificial, como os botões de funcionamento, o display de imagens, entre outros. A segunda fase abrange a introdução da inteligência artificial no código e a configuração da rede neural para o funcionamento do treinamento. A terceira e última fase compreende todo o processo de testes, as previsões realizadas pela inteligência artificial, e a transferência das coordenadas para os locais adequados, permitindo a demonstração das seleções na interface.

O projeto também inclui uma experiência interna para avaliar a eficiência do uso de uma rede convolucional auxiliar na fase de testes, resultando em duas variantes de funcionamento.

Este projeto utiliza a linguagem C++ em conjunto com uma extensão open source do QT no Visual Studio 2022. A escolha do QT open source foi motivada pelo fato de que a versão paga do QT framework não seria viável.

Este documento apresenta, com riqueza de detalhes, a implementação do sistema alinhada aos objetivos propostos, os resultados obtidos e as perspectivas para trabalhos futuros.

Palavras-Chave

C++, QT, GUI, Machine Learning, YOLOv4.

Abstract

This project involves the development of a graphical interface that enables the selection of sub-images from an original image, recording their coordinates, width, and height. The training phase is based using Darknet with YOLOv4, allowing the creation of a trained dataset that can subsequently be applied to unlabeled images for testing.

The development is structured into three phases. The first phase focuses on building the interface components unrelated to artificial intelligence, such as operational buttons and image display. The second phase incorporates artificial intelligence into the code, configuring the neural network for training purposes. The final phase covers the testing process, including predictions made by machine learning and transferring the coordinates to the appropriate location to display the selections on the interface.

The project also includes an internal experiment to evaluate the efficiency of a convolutional assistance network during the testing phase, offering two operational variants. The application was implemented in C++ with the open-source QT extension for Visual Studio 2022, this approach leverages QT due to its cost-free availability compared to the paid QT framework. The document provides a detailed account of the system's implementation, the results achieved, and potential directions for future work.

Keywords

C++, QT, GUI, Machine Learning, YOLOv4.

Table of Contents

Table of Figures

Acronyms

AI – Artificial Inteligence

API – Application Programming Interface

BoF – Bag of Freebies

CUDA – Compute Unified Device Architecture

CIoU – Complete Intersection over Union

CSP – Cross-Stage Partial

CPU – Communications Processor Unit

IDE – Integrated Development Environment

SPP – Spatial Pyramid Pooling

PANet – Path Aggregation Network

YOLO – You Only Look Once

1 Introduction

This document details the development and implementation of a solution to the selection of huge amounts of objects in images using artificial intelligence recognition of objects. This project was undertaken during an internship at Gislotica Mechanical Solutions, as part of the Project/Internship curricular unit in the 3rd year of the Electrical and Computer Engineering degree. This chapter outlines the context and objectives of the project, along with the methodology employed to achieve the final goal and provides an overview of the report organization.

1.1 Background

Gislotica is a company that designs and develops industrial machinery using standard components to make its solutions flexible.

Artificial intelligence (AI) is a sub-discipline in computer science devoted to the research and development of algorithms and systems able to do work typically demanding human intelligence, including learning from data, recognizing patterns, making decisions, understanding natural language, and perceiving environments.

AI is a technology that gets enabled and, in return, gets embedded within health, finance, automotive, and industrial automation fields.

AI systems can process and churn out huge volumes of data, make sense of patterns, and perform tasks with accuracy and efficiency exceeding that of humans by using advanced algorithms and computational power. The mainstay of AI technology comprises machine learning-the ability of the system to get better in the course of time through data exposure-and deep learning, which hosts multi-layered neural networks to model complex relationships.

AI is a very important tool for innovation and efficiency in modern industry and research; its ability to adapt, learn, and process optimize processes drives the development of intelligent industrial

machines and solutions. The need for quality control in the pattern recognition process was one of the reasons this project was initiated because it can become prone to errors over time.
The use of AI techniques in Gislotica industrial machinery ensures it will be exceptionally accurate and reliable, maintaining high quality and equal performance in pattern recognition.

1.2 Objectives

The final goal of this project is to create an interface (Figure 1) that enables the creation of a group of selections, which can be used by the AI as a basis for its assumptions.

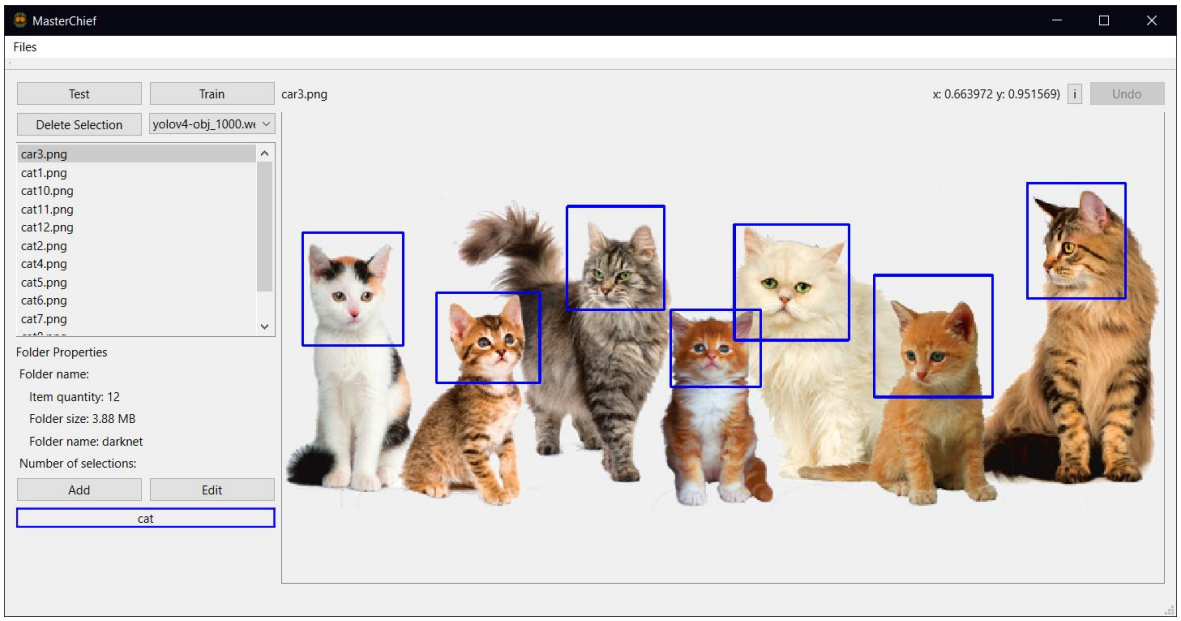

Figure 1 Aplication interface

The specific objectives consist of:

- Formulation of an ergonomic design
- Creation of the back-end code
- Implementation of the Artificial Intelligence
- Configuration of the neuronal network

1.3 Timeframe

The internship began on the day 12 of April of 2024, having been the first week used to do the onboarding. On the same day was given the information about the use of the software necessary and the objectives of the overall project. A secondary file named MasterChief, which was named after one of my favorite game trilogy, was used to do experiments and some of my first learnings using C++ and Qt, some of which are in the finished project. the development of the project began, which was divided into several stages. The first was the study of several possible approaches to the problem, then the study of these approaches and the choice of the most adequate one. The software development was made in rather decent pace regarding the darknet implementation phase where took a little bit longer.

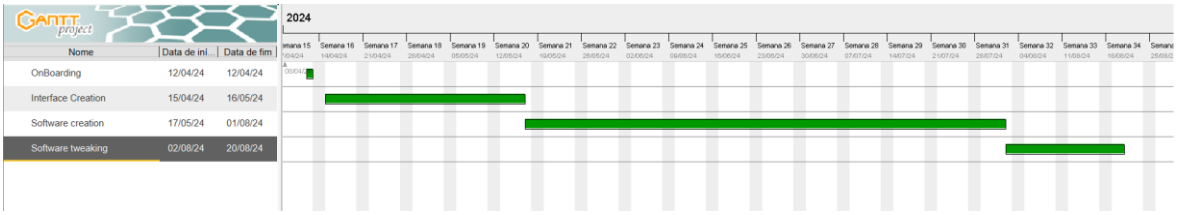

Figure 2 Project's Gantt diagram

1.4 Organization of the Report

This document is structured into five chapters, each detailing a specific aspect of the project. The current chapter introduces the project, providing an overview of its background, objectives, and timeline.

Chapter 2 delves into the technologies utilized, explaining how each one aligns with the project's requirements. It also includes a comparison and analysis of two similar projects.

Chapter 3 explores the backbone of the project as well as the front-end, explaining their functions through flowcharts. It outlines what each component does and why they are essential to the overall system.

Chapter 4 focuses on the results as well as providing an explanation of those results.

Finally, Chapter 5 offers a retrospective on the project, drawing conclusions and suggesting potential future enhancements.

2 Tools and Tecnologies

This chapter covers all tools and technologies used in the making of this project, both hardware and software, and how the two parts are connected.

2.1 Tools

2.1.1 QT: Code less

Qt (Figure 2) provides a full range of tools needed to develop software applications or embedded devices, from planning and designing the product to developing, testing, and preparing the products for the future. With more than 1+ billion devices and applications powered and assured by Qt, there are thousands of leading companies like Ducati, LG, ABB, and Peugeot that are actively using Qt for their digital solutions [1].

Qt ensures that there is always clear communication between designers and developers, whereby one common framework unifies their language. This allows them to work concurrently and thus cut down the time wasted in feedback loops, which would slow down the overall iteration process. Therefore, it will provide them with aesthetically pleasing user interfaces that are functional. Qt makes development easy, thus boosting productivity. This goes a long way in making the life of developers relatively easy, hence driving efficiency and cutting down time to market for applications and embedded devices [2].

Such powerful quality assurance tools from Qt automate testing processes to ensure that a product meets the very highest standards in performance, design, reliability, and maintainability. These solid tools help ensure continued product excellence by maintaining product quality in improvements.

With Qt, you can act on customer insights by analyzing usage intelligence that lets you continuously improve your applications and embedded devices. This way, you guarantee that your digital products remain relevant and efficient over time. All those advanced features come with a cost, and by no means is Qt free software. However, it was used in this project to make use of the open-source extension available to Visual Studio 2022 [3].

Through this extension, the developers can enjoy some of the great functionalities possessed by Qt, but without the added costs accompanied by such functionalities. A caveat, however, is that the open-source Qt functionalities will have certain limitations compared to the paid ones.

Advantages of Qt:

Qt has many advantages, mostly in providing cross-platform consistency. It's developed in a way to make performance consistent and provide a native look and feel for each popular operating system be it Windows, macOS, Linux, iOS, or Android. This allows the developer to write code just once and deploy it on multiple platforms with minor adjustments, which streamlines the development process and reduces the time to market of applications [4][5].

Moreover, Qt has a library of rich tools that significantly improve the development process in relation to graphics, animation, and user interface design. The framework consists of strong tools to aid developers in building cool-looking responsive applications, making it a favorite for projects needing high-quality graphics and complex UI elements [5]. Another strong advantage of using Qt is that it has a mature ecosystem. Over the years of its existence, Qt has accumulated extensive documentation, strong community support, and substantial third-party integration. This allows developers to have all the resources within the Qt ecosystem to overcome challenges, learn best practices, and integrate with other technologies in a seamless manner [6].

Disadvantages of Qt:

There are also several noted drawbacks despite its many advantages attached to Qt. A steep learning curve is one of the main drawbacks usually associated with Qt. Compared to more straightforward frameworks, particularly for developers new to programming or those who are not C++ developers, the learning curve for Qt can be quite steep. This challenge slows down the onboarding process for new developers and increases the time it takes for them to become proficient with the framework [7].

Another major downside to Qt is its licensing costs. While it offers open-source licenses, the commercial license is often so expensive that small enterprises and individual developers may struggle to afford it. This cost barrier can be significant for startups and independent developers who lack the budget to invest in expensive development tools [8].

Additionally, the abstraction layers used by Qt for cross-platform compatibility, although helpful, can introduce overhead. This overhead can be a disadvantage for applications that do not require the heavyweight graphical capabilities that Qt excels at. In such cases, lightweight frameworks may outperform Qt due to reduced overhead, making them more efficient for certain types of projects [9].

Competitors [3]:

In comparison, the evaluation for Qt should bear in mind the strengths and weaknesses of its competitors. Another such contender is React[10], with its reputation in the market based mainly on the fact that not only are web-based applications developed easily, but it has a huge community and high flexibility, allowing it to integrate with many technologies for the backend. However, this focus on web applications as a major concern makes it unsuitable for a fairly complex desktop application, hence restricting its use.

Another strong competitor is Telerik and Kendo UI, which are UI component frameworks with many capabilities for fast development, but even more in the .NET world. They will work well for developers who have a desire to make very fast, efficient web apps. This being the case, Telerik and Kendo UI are similar to React-incredibly web-focused and likely to get expensive quite quickly, hence setting a certain threshold on smaller projects. The official site for Telerik is [11], and for Kendo UI, you can visit [12].

Another alternative to look at is Syncfusion Essential Studio. It is known for the large component library, strong enterprise support, and high performance. It is the very thing that quite suits applications on an enterprise level that are extremely demanding in robust and scalable solutions. Though not very appealing for small projects with small-scale requirements, one of its cons would be price for small developers. The official site for Syncfusion Essential Studio can be found at [13].

In such a way, Qt provides robust cross-platform capabilities. At the same time, with such a gain come trade-offs in terms of steep learning curves, high costs, and performance overheads. Qt competes not just with React, but also with other libraries like Telerik, Kendo UI, Syncfusion, etc.; hence, while choosing one would greatly depend on specific needs and constraints, as associated with the project in question. Each of them bears different sets of advantages and drawbacks, making them suitable for different kinds of projects and development environments.

2.1.2 Visual Studio 2022

Visual Studio 2022 represents a significant evolution in Microsoft's integrated development environment (IDE) offerings, providing a robust, feature-rich platform tailored for modern software development. This version introduces several enhancements over its predecessors, focusing on improving productivity, performance, and support for new technologies [14].

Performance and Scalability

One of the most notable improvements in Visual Studio 2022 is its performance and scalability. The IDE is now a fully 64-bit application, allowing it to utilize more memory and resources. This upgrade significantly reduces the chances of running out of memory while working on large, complex solutions, a common limitation in previous versions. The transition to 64-bit architecture means developers can handle larger projects with greater efficiency, leading to faster builds and more responsive IDE performance [14].

Enhanced Code Editing and Debugging

Visual Studio 2022 introduces several enhancements to code editing and debugging, aimed at making the development process smoother and more intuitive. The new IntelliCode feature leverages AI to provide smarter code completions based on the patterns found in open-source projects, as well as the developer's own codebase. This feature helps accelerate coding by suggesting entire lines or blocks of code, reducing the amount of repetitive typing. The debugging experience has also been improved, with new capabilities like Hot Reload, which allows developers to apply code changes to a running application without needing to restart it. This feature is particularly useful in accelerating the iterative process of debugging and testing. Additionally, Visual Studio 2022 has introduced more advanced profiling tools that provide deeper insights into application performance, helping developers identify and resolve bottlenecks more efficiently [14].

Improved Collaboration and Version Control

Collaboration is a key focus in Visual Studio 2022, with integrated support for Git and GitHub, making it easier for teams to manage their code repositories. The Git tooling is built directly into the IDE, providing a seamless experience for version control operations. Developers can view and manage branches, stage changes, commit code, and even resolve merge conflicts without leaving the IDE. The integration with GitHub Actions also allows developers to trigger CI/CD pipelines directly from Visual Studio, streamlining the development workflow. The Live Share feature continues to be a standout for remote collaboration, allowing multiple developers to work on the same codebase in real-time, sharing their code, debugging sessions, and even terminal instances. This capability has

become increasingly valuable in distributed development environments, enhancing team productivity and reducing the friction of remote collaboration [14].

Support for New Technologies

Visual Studio 2022 supports a wide array of programming languages and frameworks, with enhanced support for the latest versions of C#, .NET 6, and .NET MAUI. The IDE is well-equipped for developing cross-platform applications, providing tools and libraries that facilitate the creation of applications that run seamlessly on Windows, macOS, Linux, iOS, and Android.

The support for containerized development has also been improved, with integrated Docker support that simplifies the process of developing, debugging, and deploying containerized applications. Additionally, Visual Studio 2022 includes tools for developing cloud-native applications, with built-in support for Azure services, making it easier to develop, deploy, and manage cloud-based applications [14].

User Interface and Customization

Visual Studio 2022 features a refreshed user interface that is both modern and customizable. The new UI is designed to reduce clutter and make it easier to focus on the code, with a cleaner look and feel that aligns with Windows 11's design language. The customization options have also been expanded, allowing developers to tailor the IDE to their workflow with personalized color themes, font settings, and layout configurations[14].

Learning Resources and Community Support

Microsoft has continued to invest in making Visual Studio 2022 accessible to developers of all skill levels. The IDE comes with a wealth of integrated learning resources, including tutorials, code snippets, and sample projects that help developers quickly get up to speed with new technologies and features. The integration of Visual Studio into Microsoft Learn provides a direct link to a broader set of learning paths and resources, further supporting continuous learning and skill development.

The Visual Studio community is large and active, offering extensive support through forums, GitHub repositories, and third-party extensions. This community-driven ecosystem ensures that developers can find plugins and tools to extend the IDE's capabilities, making Visual Studio 2022 adaptable to virtually any development need[14].

2.2 Software Technologies Utilized

This section covers the software-related technologies used to develop the project, from programming languages to their libraries, data transmission and databases.

2.2.1 C++

C++ is an enhanced high-performance, object-oriented form of the C programming language, allowing for better modularity in code and, hence, more reusable code. It was developed by Bjarne Stroustrup in the year 1985. The programming languages it supports are procedural, object-oriented, and generic paradigms. Being very strong in low-level memory manipulation, C++ is best suited for system programming, game development, real-time simulations, and other performance-intensive applications. The Standard Template Library provides potent tools for data structures and algorithms. C++ has an extensive range of applications, both in operating systems and compilers, embedded systems, with fine-grained control over system resources and memory for execution of applications. It's pretty complex, but at the same time, C++ is one of the common programming languages because of its versatility in both high-system-level and application-level software development[15].

2.2.2 Python

Python is a widely used programming language celebrated for its simplicity, versatility, and robust community support. Its straightforward syntax makes it easy to learn and use, which is why it’s popular among both beginners and experienced developers. Python's flexibility allows it to be applied in various domains, including web development, data science, automation, scientific computing, and more [16].

One of Python's greatest strengths is its extensive ecosystem. With a vast array of libraries available, developers can quickly implement solutions for a wide range of tasks without building everything from scratch. For instance, frameworks like Django and Flask are commonly used for web development, while libraries such as NumPy, Pandas, and TensorFlow are essential for data science and machine learning projects [17].

Although Python is an interpreted language, which generally means slower execution compared to compiled languages like C++ or Java, it offers several ways to mitigate performance issues. Developers can enhance performance using C extensions, Just-In-Time (JIT) compilation tools like PyPy, and the multiprocessing module [18].

Python 3 is now the standard version of the language, offering better features and improvements over Python 2, which is no longer supported. This transition has allowed Python to continue evolving, incorporating modern features that make it even more powerful [19].

Python's popularity spans many industries, from tech companies and finance to scientific research and education. Companies like Google, Netflix, and NASA rely on Python for various tasks, from backend development to data analysis. Its role in cutting-edge fields like artificial intelligence and big data ensures that Python will remain a key player in the programming world for the foreseeable future.

2.2.3 Darknet

Darknet is an open-source neural network framework that stands out for its efficiency, flexibility, and simplicity in deep learning tasks, especially in computer vision. Written in C and CUDA, Darknet is highly optimized for performance, allowing it to run efficiently on GPUs, which is essential for managing the computational demands of modern deep learning models [20].

One of Darknet's key strengths is its modular design, which makes it easy for developers and researchers to experiment with different network architectures, loss functions, and data augmentation techniques. This flexibility has made Darknet a popular choice for developing and deploying real-time object detection models. The YOLO (You Only Look Once) series of models, widely adopted for tasks requiring fast and accurate object detection, such as in autonomous vehicles, surveillance, and robotics, is a prime example of Darknet's capabilities [21].

Although Darknet is less feature-rich compared to other deep learning frameworks like TensorFlow or PyTorch, its lightweight nature ensures accessibility and ease of modification. This simplicity is a significant advantage for researchers who need to iterate quickly on their models without the complexity of a more elaborate framework [22].

Darknet also benefits from a robust community of users and contributors who continuously enhance the framework and share pre-trained models, making it easier for newcomers to get started. Its ongoing use in academic research and industry applications highlights its relevance and utility in the fast-evolving field of deep learning [23].

2.2.4 YOLOv4

YOLOv4, or "You Only Look Once, version 4," is an advanced object detection model developed on the Darknet framework. Building on its predecessors, YOLOv4 introduces significant enhancements in speed, accuracy, and efficiency, making it well-suited for real-time object detection.

The YOLOv4 model continues the YOLO tradition of framing object detection as a single regression problem, which allows for high efficiency and fast processing. Key architectural improvements include the use of CSPDarknet53 as the backbone, which improves feature

extraction by splitting and then merging feature maps to reduce computational load and enhance earning [24]. The model also incorporates Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PANet) to improve multi-scale object detection and feature hierarchy [25][26].

YOLOv4 employs several optimization techniques to boost performance. The Bag of Freebies (BoF) includes data augmentation strategies like Mosaic and CutMix, which enhance model accuracy without additional computational cost [24]. The Bag of Specials (BoS) features improvements such as the Mish activation function and CIoU loss, which contribute to better performance [27][28]. The AdaBelief optimizer is used to keep training fast and stable [29].

YOLOv4 offers a strong balance between accuracy and speed, achieving high performance on datasets like MS COCO while running efficiently on consumer-grade hardware. This makes it suitable for a range of applications including autonomous vehicles, security systems, robotics, and medical imaging. Despite its advantages, YOLOv4 has limitations such as the need for substantial computational resources for training and challenges in detecting small objects [24].

Further explanation and in-depth discussions on these aspects and the broader capabilities of YOLOv4 will follow.

2.2.5 CUDA

The Compute Unified Device Architecture, CUDA in abbreviation, is a parallel computing platform and programming model that has been invented by NVIDIA, hoping to enable developers using NVIDIA GPUs to achieve general-purpose computing beyond graphics rendering [30]. CUDA has an extension to the C/C++ language interface with a number of tools, including libraries highly optimized for GPU acceleration. This programming model enables thousands of codes to execute concurrently across the GPU cores, giving a notable gain in computational speed compared to classic CPU processing [30]. CUDA is critically important for the optimization of the model's performance in the Darknet YOLOv4 context. Darknet is an open-source neural network framework written in C and CUDA, which uses CUDA for parallel computations-something critically important for deep learning operations within YOLOv4. This includes the parallel computation of feedforward and backpropagation calculations, therefore accelerating training and inference times [30]. CUDA also has the provision for using highly optimized libraries like cuDNN that help in operations such as convolutions and activation functions, among others, which go a long way in further enhancing the efficiency of the model [30].

Further, efficient memory management is achieved on the GPU through CUDA, which makes YOLOv4 able to handle big data effectively, such as those of weights and activations, without much performance overhead. Efficient use of GPU resources is one that is paramount for real-time applications, thus rendering YOLOv4 suitable in autonomous vehicle perception and real-time surveillance tasks. With the help of CUDA capabilities, YOLOv4 achieves high accuracy and speed, making it possible to assert that CUDA has brought significant performance impact for modern object detection technologies.

2.2.6 CuDNN

CuDNN (CUDA Deep Neural Network library) is a GPU-accelerated library developed by NVIDIA specifically for deep learning applications. It is designed to optimize performance for deep neural networks by providing highly efficient implementations of standard operations such as convolutions, pooling, normalization, and activation functions [33]. cuDNN is integral to leveraging the full power of NVIDIA GPUs, offering improved speed and efficiency for training and inference tasks in deep learning models.

In the context of Darknet YOLOv4, cuDNN plays a crucial role in enhancing the model's performance. YOLOv4, an advanced object detection model developed using the Darknet framework, benefits from cuDNN’s optimized operations for convolutional layers and other deep learning functions [33]. cuDNN accelerates these operations by using GPU resources efficiently, which is vital for the high-speed requirements of real-time object detection.

By integrating cuDNN, YOLOv4 can achieve faster training times and more efficient inference. This is particularly important for handling the large-scale computations required by deep neural networks, as cuDNN’s optimized routines significantly reduce the computational overhead compared to CPU-only implementations [33]. As a result, YOLOv4 can perform complex object detection tasks in real-time applications such as autonomous vehicles and surveillance systems, demonstrating the effectiveness of cuDNN in improving deep learning performance [33].

2.2.7 OpenCV

OpenCV (Open Source Computer Vision Library) is a general-purpose library designed for performing real-time computer vision and image processing activities. It contains function libraries and tools for capturing video and images, extracting features, detecting objects, and manipulating images. Being developed in C++, OpenCV is efficient and scalable and supports multiple platforms

such as Windows, macOS, Linux, iOS, and Android [32]. Most use-cases of Darknet YOLOv4 combine pre-processing and post-processing with OpenCV.

YOLOv4 mainly focuses on the detection of objects, while OpenCV presents image reading and display, image transformation, such as resizing and cropping, and the integration of detection results in an application. Optionally, OpenCV could be used for capturing video frames and pre-processing them to prepare video frames that meet the input requirements of YOLOv4; then, the detection results can either be presented or further processed in real-time.

This fusion with OpenCV would multiply the usability and versatility of the object detection system of YOLOv4. It has tools for image data manipulation, which is really important for tasks of visualization of detection results and management of input data streams. With the use of OpenCV, the developer can easily exploit object detection possibilities in YOLOv4 and its robustness with regard to both images and video processing.

2.3 YOLOv4's Architecture and AI Training System

This topic explores the intricacies of YOLOv4, a cutting-edge AI model for object detection, and delves into its operational mechanisms. Understanding these complexities is key to grasping the functionality and performance of the primary application. A detailed examination of YOLOv4's architecture, training methodologies, and GPU utilization is essential for evaluating the effectiveness and efficiency of the implemented object detection system. This foundational knowledge provides the necessary context for assessing the practical application and impact of YOLOv4 within the project's scope.

YOLOv4's architecture is designed to maximize both detection accuracy and processing speed. The model is divided into three main components: the backbone, the neck, and the head.

2.3.1 Backbone

The backbone of YOLOv4 is CSPDarknet53, which acts as the feature extractor. CSPDarknet53 builds on Darknet-53 and incorporates Cross-Stage Partial (CSP) connections. It enhances feature extraction efficiency by splitting the feature map into two segments, processing each separately, and then concatenating them. This design reduces computational complexity and improves the network’s ability to learn robust features by addressing issues like gradient vanishing. Understanding feature maps-outputs of convolutional layers that represent various aspects of the

input image-is crucial for appreciating how CSPDarknet53 achieves efficient feature extraction [24][25][26].

CSP connections further optimize this process by reducing redundancy and enhancing the model's learning capacity, maintaining effective gradient flow during training and contributing to more accurate feature learning [24][25][26].

2.3.2 Neck

The neck of YOLOv4 includes two key modules: Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PANet). These components enhance the network's ability to handle variations in object sizes and improve detection accuracy [24][27][28].

The SPP module introduces a multi-scale pooling approach to feature maps, capturing features at varying resolutions and ensuring the network retains critical spatial information. This approach is essential for detecting objects of different sizes and aspect ratios. PANet, on the other hand, enhances feature hierarchy by aggregating information from multiple network layers, refining feature representation, and optimizing feature aggregation. Together, SPP and PANet enable YOLOv4 to achieve high accuracy in object detection by effectively leveraging both detailed and abstract feature information [24][27][28].

2.3.3 Head

The head of YOLOv4 is responsible for generating predictions, specifically bounding boxes and class probabilities. It builds on the anchor-based detection strategy of previous YOLO versions, with refinements to enhance prediction accuracy and robustness [24][29].

YOLOv4 uses multi-scale detection layers and the CIoU (Complete Intersection over Union) loss function for bounding box regression. Multi-scale layers ensure accurate predictions across various object sizes, while CIoU refines these predictions by incorporating additional geometric factors, such as aspect ratio and center distance. This approach improves the precision of object localization, leading to better overall detection performance [24][29].

2.3.4 Training

YOLOv4 integrates advanced techniques to optimize model performance, focusing on data quality and learning efficiency. Data augmentation techniques like Mosaic augmentation and CutMix

diversify the training dataset, enhancing the model's ability to generalize and reducing overfitting [24][25][27].

YOLOv4 also employs Bag of Freebies (BoF) and Bag of Specials (BoS) strategies to improve training without significantly increasing computational costs. BoF includes data augmentation and regularization methods that enhance accuracy, while BoS introduces slight computational overhead but offers significant performance gains, such as with the Mish activation function and CIoU loss function [24][25][27][29].

The training process is further optimized using stochastic gradient descent (SGD) and GPU acceleration, enabling efficient handling of large datasets and complex models. These elements collectively contribute to YOLOv4's high performance and efficiency in real-world object detection applications [24][29].

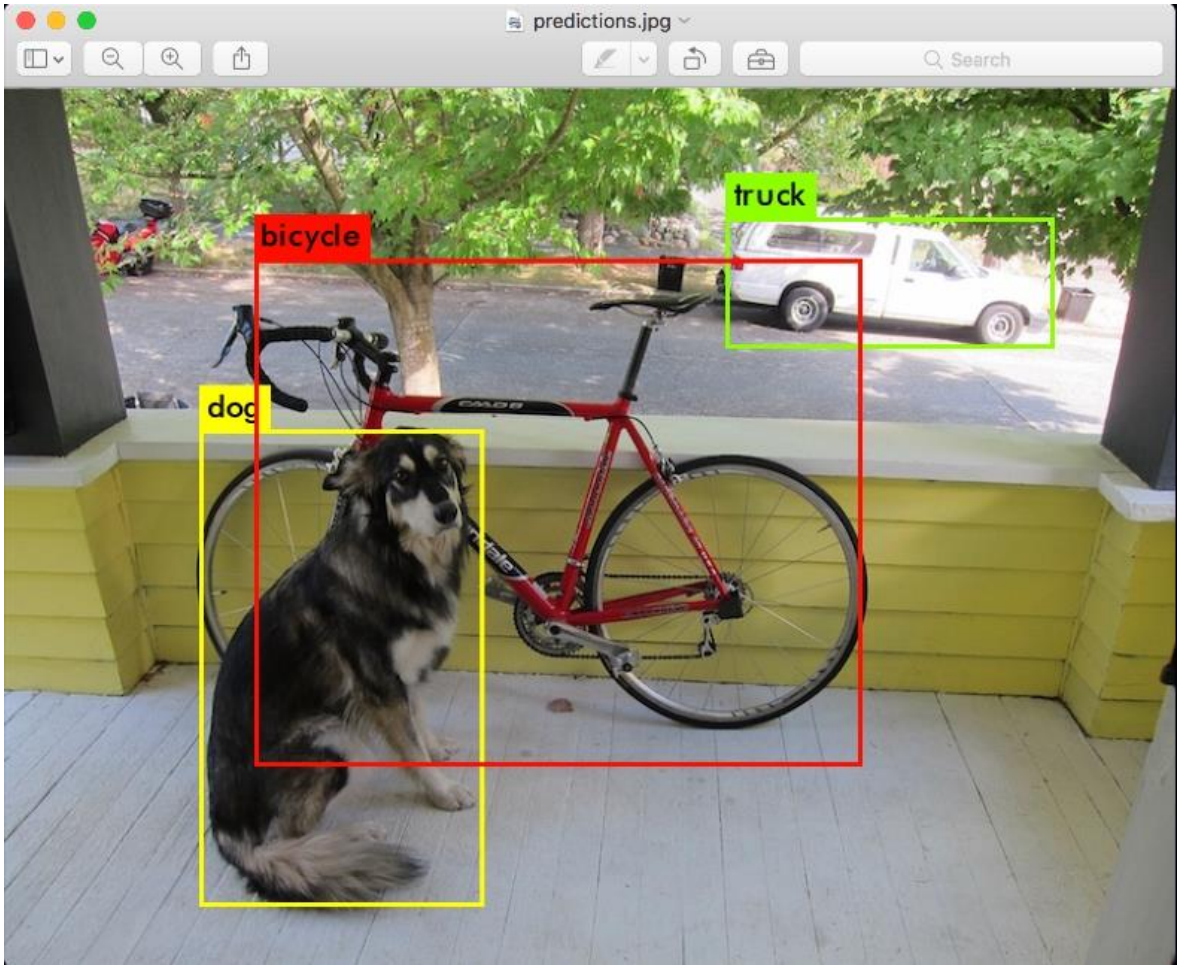

Figure 1 Ai Predictions exemple

When training deep learning models using Darknet, the accompanying loss chart serves as a crucial indicator of the model's progress throughout the training process. The y-axis of the chart, representing the loss, plays a significant role in evaluating how effectively the model's predictions align with the actual ground truth values.

2.3.4.1 Understanding the Loss Metric

Definition and Purpose: The loss function quantifies the discrepancy between the model's predictions and the true values. It is designed to gauge how far off the model's predictions are from the expected results. The core objective during training is to minimize this loss, with a lower value signifying that the model's predictions are becoming more accurate and closely aligning with the actual data [24][25][27].

2.3.4.2 Types of Loss Functions in YOLOv4

Bounding Box Loss: This aspect of the loss function measures how well the predicted bounding boxes correspond to the actual ground truth boxes. YOLOv4 employs the CIoU (Complete Intersection over Union) loss, which evaluates the accuracy of bounding boxes by considering overlap, aspect ratio, and center distance between predicted and true boxes [24][29].

Class Loss: This component assesses the error in predicting the correct class for the object within the bounding box, often measured by cross-entropy loss [24][29].

Objectness Loss: This evaluates the model's confidence in whether an object is present within a bounding box, contributing to the overall assessment of prediction quality [24][29].

2.3.4.3 Interpreting the Chart

Y-Axis (Loss): The y-axis displays the loss values, with higher values indicating poorer model performance due to less accurate predictions. Lower values, on the other hand, reflect better performance as the predictions approach the ground truth [24][25].

X-Axis (Epochs/Iterations): The x-axis represents the number of training epochs or iterations. An epoch signifies one complete pass through the entire training dataset, allowing the model to learn and adjust based on the provided data [24][25].

2.3.4.4 Analysis of the Loss Chart

Initial Phase: At the start of training, it is common for the loss to be high. This occurs because the model begins with randomly initialized weights and has not yet learned to produce accurate predictions [24][25].

Training Progress: As training progresses, a decrease in loss is expected. This reduction indicates that the model is improving its predictions and effectively minimizing errors [24][25].

Convergence: The ideal outcome is a stabilization of the loss value at a lower bound, demonstrating that the model has learned effectively and is making accurate predictions [24][25].

Plateaus or Fluctuations: If the loss chart shows plateaus or fluctuations, it may suggest issues such as an unsuitable learning rate, overfitting, or insufficient training. In such scenarios, adjusting hyperparameters or extending the training duration might be necessary to address these concerns [24][25].

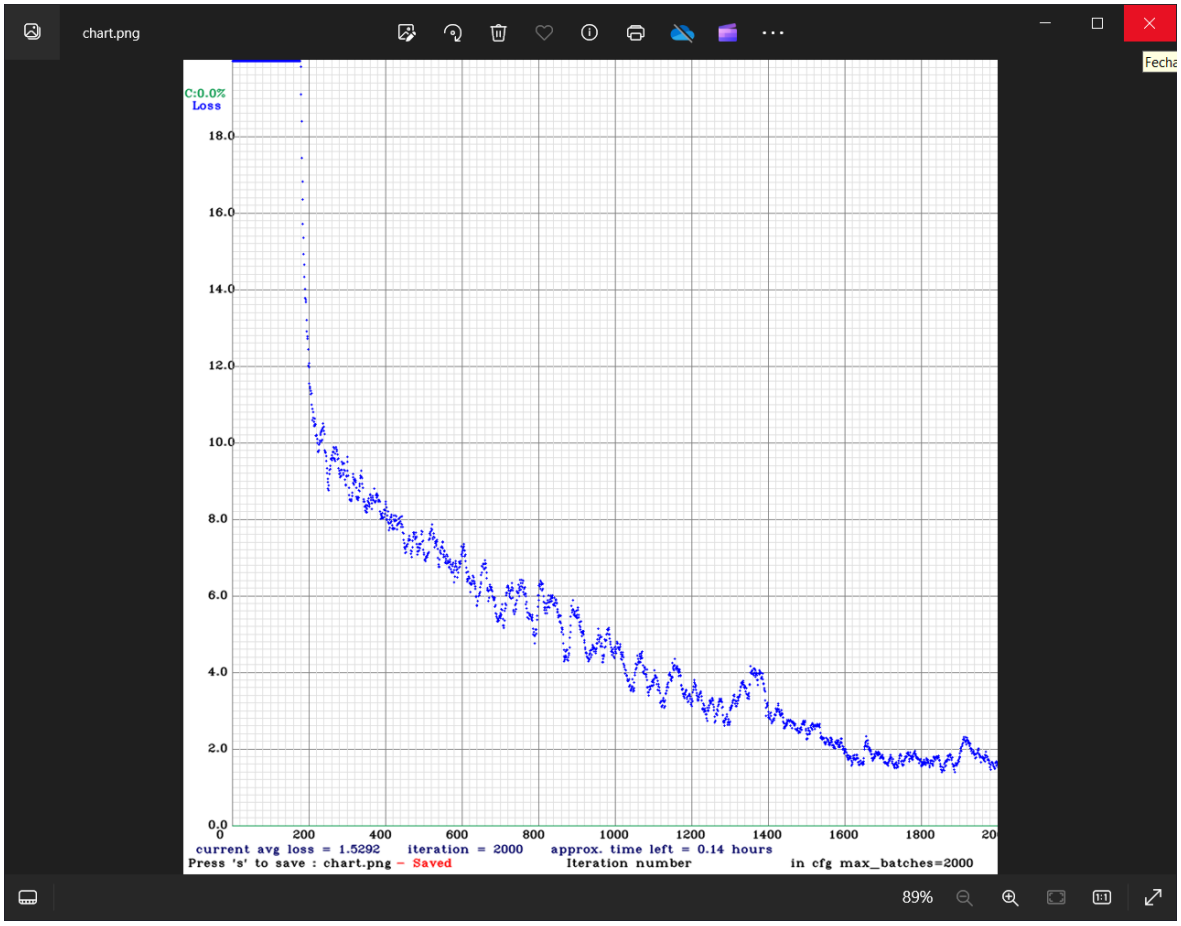

Figure 2 Loss chart of Ai training

2.3.5 Impact of the GPU

The use of GPUs for training deep learning models, such as those implemented in Darknet, offers significant advantages over CPUs due to their specialized architecture and parallel processing capabilities [30][36]. GPUs have thousands of cores designed for high-efficiency parallel processing, unlike CPUs, which are optimized for sequential tasks with fewer cores [30][36]. This parallelism is essential for deep learning, where tasks like matrix multiplications and convolutions are performed on large datasets. The GPU's ability to execute multiple operations simultaneously accelerates the computations involved in training neural networks, leading to faster convergence and reduced training times [30][36].

Additionally, GPUs are specifically designed to handle the repetitive, high-dimensional calculations common in deep learning. While CPUs excel in single-threaded performance, GPUs are superior in scenarios requiring the simultaneous processing of numerous data points [30][36]. This results in a significant boost in overall training efficiency. The practical impact of using GPUs is evident in the drastic reduction of training times; models that might take weeks or days to train on CPUs can often be trained in hours or even minutes on GPUs [30][36]. This efficiency accelerates model development, allows for the exploration of more complex architectures and larger datasets, and enables real-time experimentation and tuning [30][36].

In summary, GPUs enhance deep learning model training by leveraging their parallel processing capabilities and computational power, significantly reducing training time and enabling more efficient experimentation and model development [30][36

2.4 Scientific archive

2.4.1 Deep Learning in Skin Disease Image Recognition: A Review [34]

The review "Deep Learning in Skin Disease Image Recognition: A Review" analyzes 45 research works regarding the application of deep learning for the identification of skin diseases from images, considering studies from 2016. The paper evaluates such studies according to the types of disease addressed, datasets used, processing of data, augmentation of data, models of deep learning applied, evaluation metrics, and overall performances. They concluded that deep learning methods, particularly when multi-model fusion is implemented, have higher overall performance compared to the conventional approaches, including dermatologists' methods.

Meanwhile, these studies found one of the major challenges was how to get enough-scale datasets for training deep learning, primarily due to privacy and some rare disease problems. Further, preprocessing techniques such as denoising, size alteration, normalization, and grayscale conversion are also crucial in enhancing model accuracy. The lack of data has also resulted in the adaptation of many data augmentation methods like rotation and noise addition, and GAN seems to be a good approach for the generation of new training data.

Other deep learning models widely employed in these studies use Convolutional Neural Networks, such as AlexNet, VGG, Inception, ResNet, and DenseNet. These multi-model fusions, combining the strengths of many models, exhibit much better performance than single-model methods. Common evaluation measures include precision, mean average precision, sensitivity (actual positive rate), specificity (actual negative rate), and area under the ROC curve (AUC). Most studies evidenced high accuracy and AUC, emphasizing the recognition of skin diseases using deep learning models.

The deep-learning models for skin disease recognition provide a few key advantages. They can extract features from raw data, help eliminate the process of manually engineering features, and improve diagnostic accuracy and robustness when dealing with large and complex datasets. However, there are still major challenges associated with the use of deep learning, especially in the need for very large, annotated datasets. This motivation was to explore new data augmentation techniques that could involve developments in the direction of GANs and further research on the interpretability of deep learning models making them reliable for clinical applications.

The review concludes that deep learning holds great promise for revolutionizing the diagnosis and treatment of skin diseases, potentially leading to AI-driven diagnostic tools embedded in mobile devices or wearables. However, there are still several technical, ethical, and legal issues that remain to be resolved before the development of a wide range of such tools for use in routine clinical practice. In general, the study underlines the potential application of AI in dermatology, as well as in other fields, to bring meaningful improvement in diagnostic accuracy and efficiency.

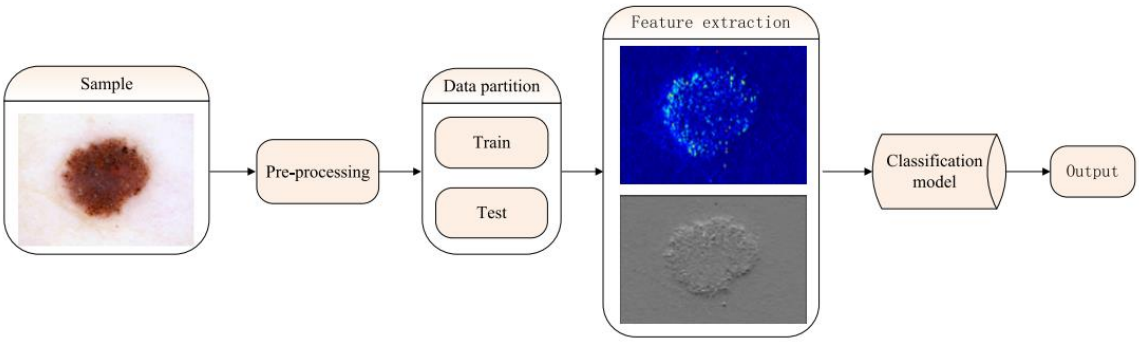

Figure 3 Flow chart of skin disease image recognition based on machine learning. Image processing is divided into image acquisition, image preprocessing, and dataset division. Image preprocessing includes image size adjustment, normalization, and noise removal.

2.4.2 Deep Facial Diagnosis: Transfer Learning from Face Recognition to Facial Diagnosis [35]

The work investigates the use of deep transfer learning for computer-aided facial diagnosis from uncontrolled 2D face images, aiming to transfer knowledge from face recognition models to identify various diseases. The study conducts experiments in diagnosing conditions such as beta-thalassemia, hyperthyroidism, Down syndrome, and leprosy, achieving accuracies greater than 90%, which significantly surpass the performance of traditional machine learning methods and human clinicians.

One of the key challenges identified is the complexity and expense associated with collecting patient-specific face image datasets due to ethical limitations and privacy concerns. Consequently, the datasets used in the study are relatively small. The research employs two primary approaches in deep transfer learning (DTL): tuning pre-trained Convolutional Neural Networks (CNNs) and using CNNs as fixed feature extractors. Models pre-trained on face recognition tasks, such as VGG-Face, were fine-tuned for facial diagnosis, demonstrating superior performance compared to models pre-trained on generic image datasets like ImageNet.

Experimental results show that the proposed model, using CNN as a feature extractor, achieved a top-1 accuracy of 95% for single disease detection (beta-thalassemia). For multi-disease detection, including beta-thalassemia, hyperthyroidism, Down syndrome, and leprosy, the model attained a top-1 accuracy of 93.3%. These results highlight the significant advantages of deep learning methods over traditional machine learning techniques, particularly in both binary and multiclass classification problems.

The study underscores the practical implications of deep transfer learning, marking it as a promising approach for developing noninvasive and low-cost methods for disease screening. Such advancements could improve early diagnosis and treatment accessibility, especially in less developed regions with limited medical resources. The ability of deep learning models, particularly through transfer learning, to achieve high accuracy and robustness in diagnosing diseases based on facial images is notable. Fine-tuning pre-trained models helps mitigate the need for large labeled datasets, addressing one of the critical barriers in medical AI applications.

However, despite these promising results, the study acknowledges the need for further research to enhance the interpretability and transparency of the models, making them more suitable for clinical use. Ethical considerations, including data privacy and informed consent, are also critical in the responsible deployment of AI in medical diagnostics. Integrating AI-driven diagnostic tools into mobile and wearable devices has the potential to revolutionize healthcare delivery, but it also presents technical and regulatory challenges that must be addressed.

In summary, this paper explores the potential of AI, particularly deep transfer learning, to transform facial diagnosis into a reliable, efficient, and accessible medical tool. The findings suggest that AI could lead to substantial changes in digital healthcare, paving the way for more widespread and equitable access to medical diagnostics.

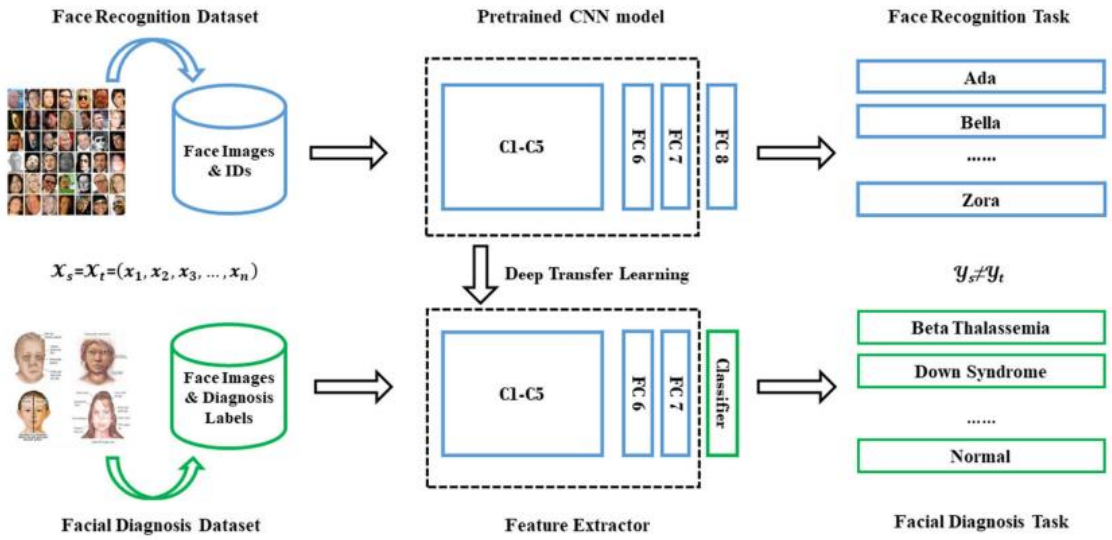

Figure 4 The schematic diagram of facial diagnosis by deep transfer learning [35]

3 Software development

This chapter covers in detail all of the software components developed in this project, from front-end to back-end and the API that establishes communication between both, and all the testing done with general examples and the adaption to the project in question.

3.1 Software Choice

As outlined in the Tools section, the Integrated Development Environment (IDE) selected for this project was Visual Studio 2022. This choice was not made directly by the intern, but rather by the company, due to the commercial nature of the Qt Toolkit, which requires a paid license. Consequently, Visual Studio 2022 was utilized in conjunction with a Qt extension, enabling the use of Qt's tools, libraries, and interface designer (Qt Designer) within the Visual Studio environment. This setup facilitated the integration of Qt's functionalities while leveraging the robust development features provided by Visual Studio 2022.

3.2 Overview of The Application

The objective of the application is to facilitate the selection of portions of the images by having the possibility of using an AI custom trained dataset on going. Figure 5 presents the global architecture of the application.

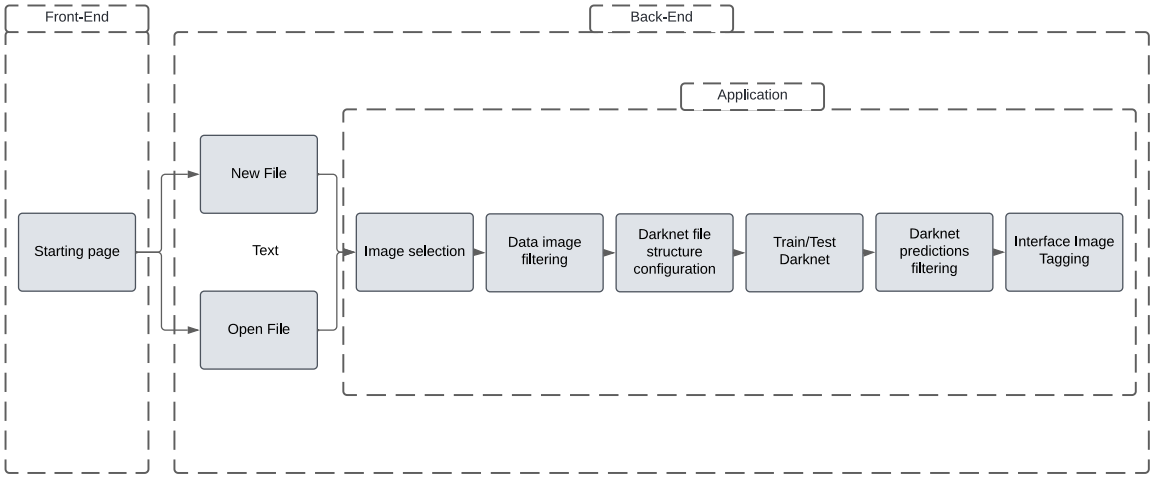

Figure 5 Project overview

The corporation's initial requirement for the application was to develop a functional and user-friendly interface in Qt, considering the large-scale image training that would be involved.

When opting to open an existing file, the application requires a specific folder type to initialize, a modified Darknet folder. This folder is a streamlined version of the original Darknet repository from GitHub, containing only essential components and excluding unnecessary files.

3.3Front-End Development

The front-end development involves designing and implementing the application's graphical user interface, adhering to the wireframe depicted in Figure 6. The interface comprises a combination of several buttons, labels, graphic views, line edits, etc..

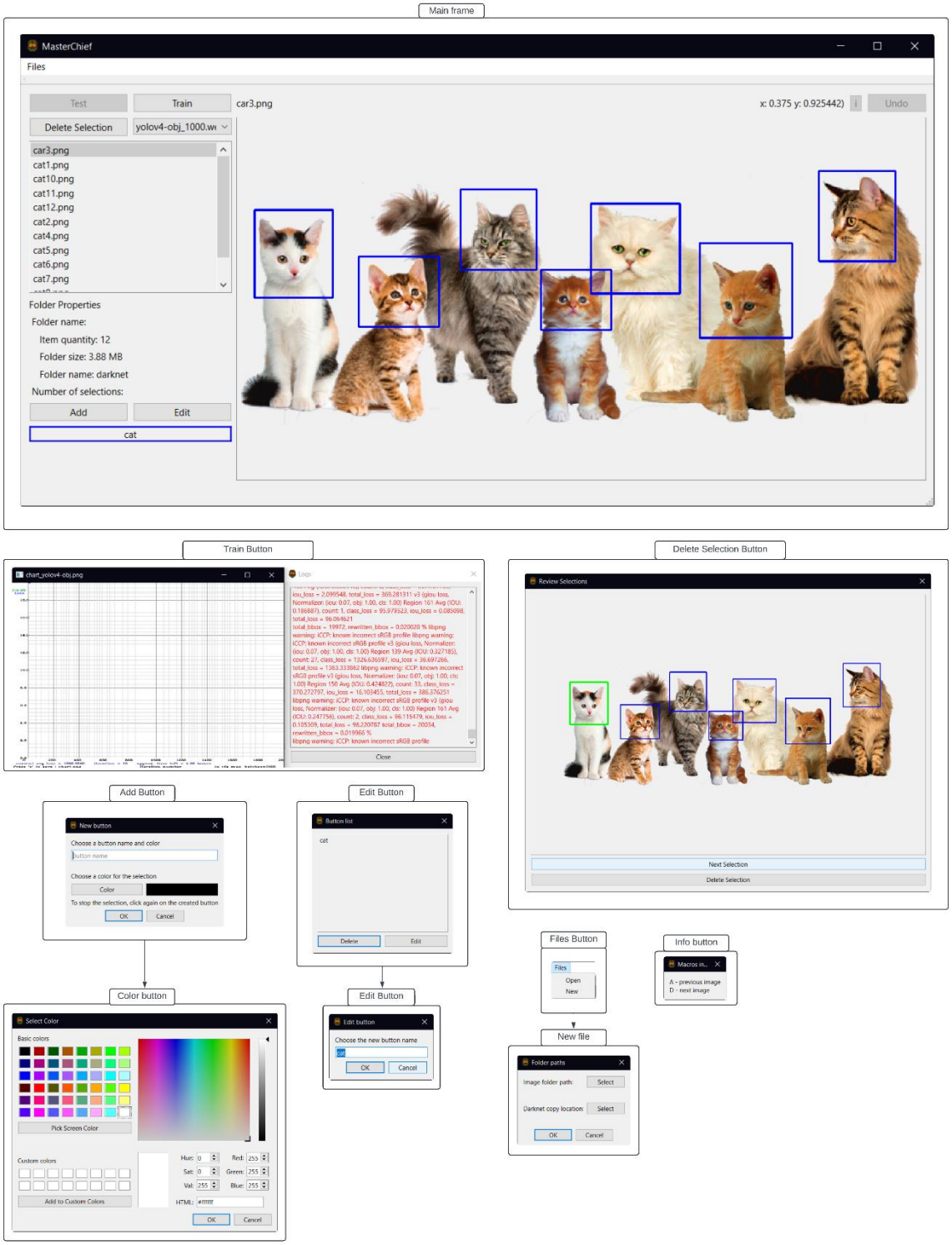

Figure 6 All the Project windows and interactable elements

After opening a new or preexisting file, the application aims to create selections on approximately 10% of the total images for tagging. The tagging process begins by creating a class that names the tags. This is done by clicking the "Add" button, which generates a new button below it. Activating this button initiates the tagging stage, where a cursor appears, allowing the user to click once on the image, followed by a second click to create a tag.

Once 10% of the images have been tagged, the "Train" button should be clicked to initiate the training process. This process typically takes between 1.5 to 3 hours. Upon completion, clicking the "Test" button updates the ComboBox with the newly trained weights. Activation of the "Test" button triggers a message in the uppermost label.

The AI detection system is controlled exclusively by pressing the "D" macro on the keyboard. For optimal detection, the image must be free of any pre-existing tags. During testing, any incorrect AI detections can be corrected using the "Delete Selection" button, which removes the unwanted selections.

The application includes additional buttons for managing and correcting tags during the image labeling process. The "Edit" button allows for modifications to the class buttons. It provides functionality to delete existing class buttons or rename them if necessary.

Located in the upper right corner, the "Undo" button facilitates error correction during the selection stage. If a selection is made incorrectly, clicking the "Undo" button will remove the most recent selection, enabling precise adjustments to the tagging process.

3.4 Darknet folder structure

This application is built on a modified Darknet folder that adheres to a specific structure.

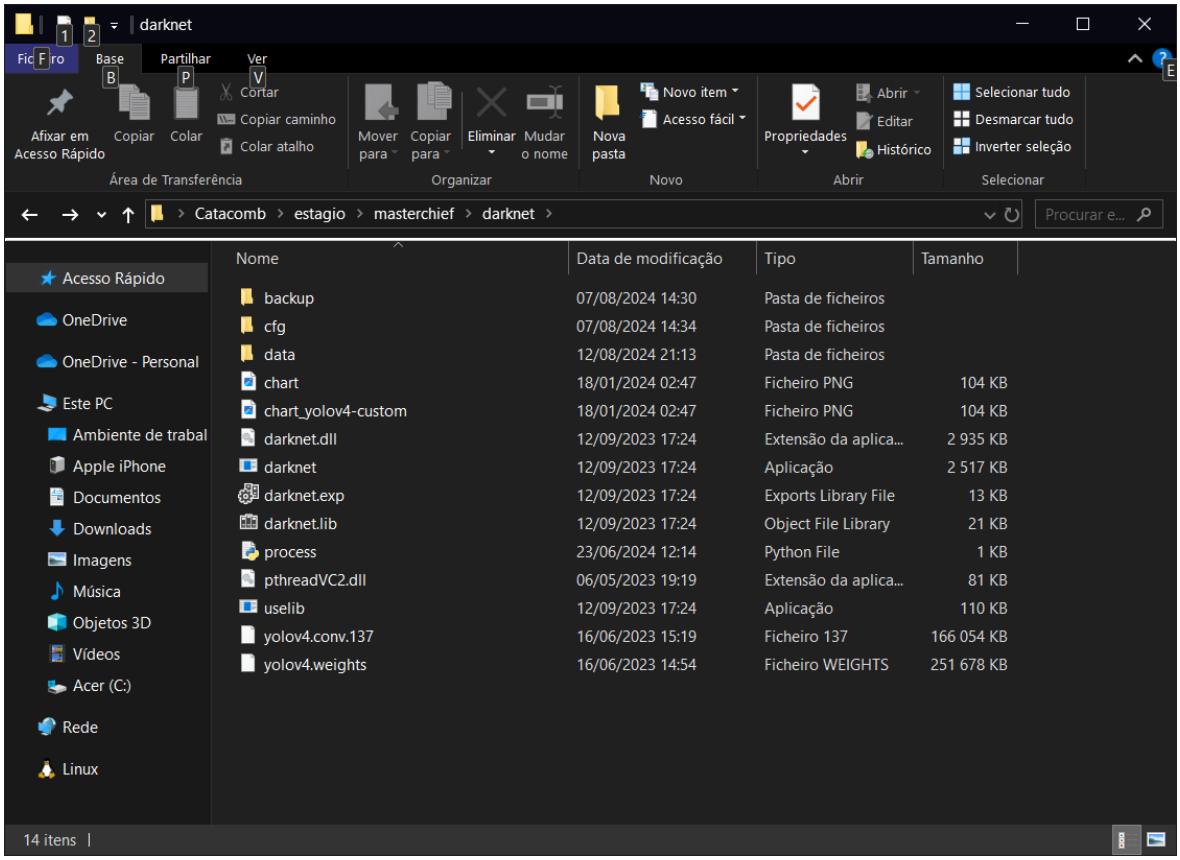

Figure 7 Darknet folder structure

This folder includes essential components such as the data folder, cfg folder, backup folder, and the process.py script in the main directory, along with other key features inherited from the original GitHub repository. Notably, it also contains the yolov4.conv.134 file, which is a pre-trained convolutional layer file used as a starting point for training the YOLOv4 model, allowing the network to fine-tune weights more efficiently during the training process.

3.4.1 Data Folder

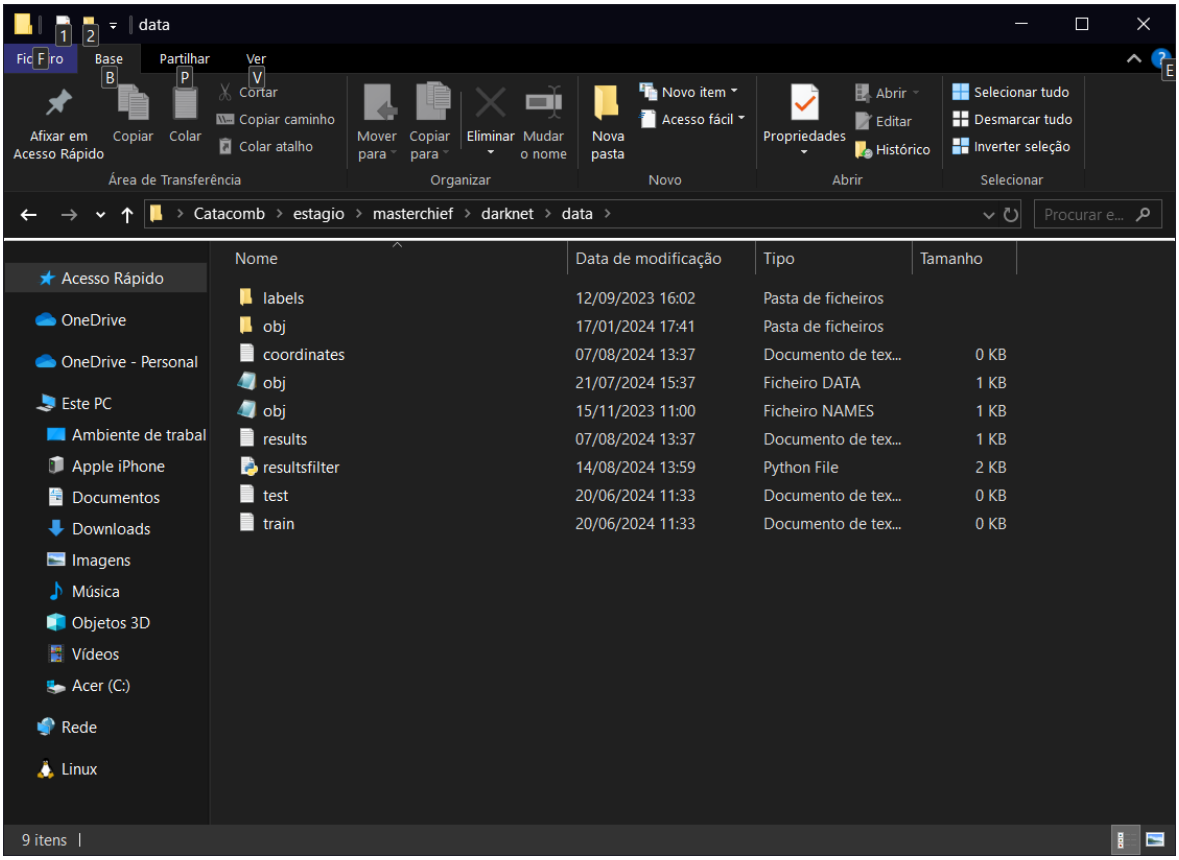

Figure 8 Data folder structure

3.4.1.1 Labels Folder

Inside the data folder in Darknet, the labels folder contains text files that correspond to the images used for training or testing the model. These text files include annotations that describe the bounding boxes and class labels for the objects within each image.

Each text file usually shares the same name as the corresponding image file, but with a .txt extension instead of an image file extension like .jpg or .png. The content of these text files is crucial for the model's training process, as they provide the ground truth data that the model uses to learn object detection. The annotations typically follow a specific format, including the class number, the coordinates of the bounding box, and sometimes additional information like object confidence scores.

This setup allows the YOLOv4 model to understand which objects are present in the images and where they are located, enabling it to learn to detect these objects in new, unseen images.

3.4.1.2 Obj Folder

The folder is structured to hold the images used for training and testing alongside corresponding text files that annotate each image. These text files share the same name as the images and contain the coordinates of the objects within the image, formatted as follows:

<index of the tag class> <x-coordinate of the tag center> <y-coordinate of the tag center> <width of the tag> <height of the tag>

This format is critical for the YOLOv4 model as it provides precise information about the location and size of each object in the image, allowing the model to learn to accurately detect and classify objects during training.

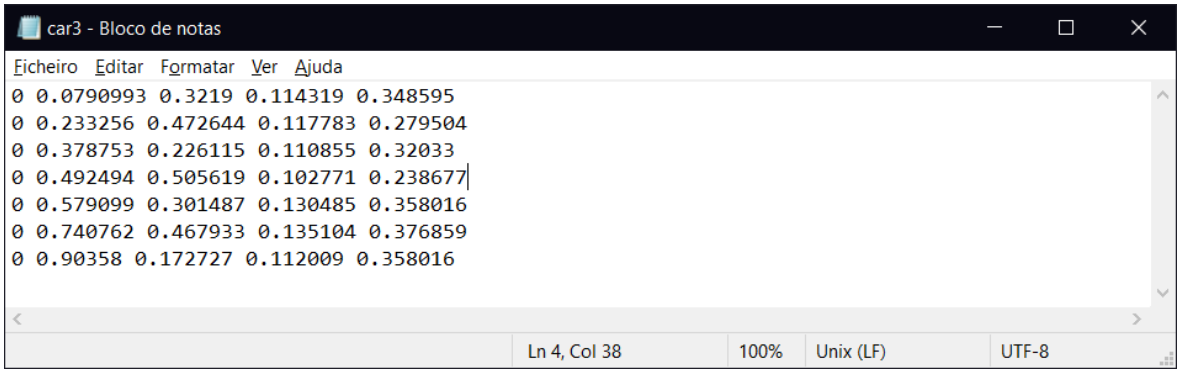

Figure 9 Selections storage example

3.4.1.3 ResultFilter.py

After the AI completes its predictions using the trained weights, it generates a grayscale JPG file named predictions in the main folder. This file contains the last tested image along with the predictions made by the AI. The predictions file plays a key role in converting the AI's predicted coordinates into a structured format for the results.txt file.

The coordinates of these predictions can be extracted due to the -ext_output argument appended to the training command. This argument ensures that detailed output, including the coordinates of detected objects, is provided during inference. However, the output generated includes more information than needed. To isolate the relevant data, the resultFilter Python script is employed. This script filters out unnecessary details, extracting only the essential information required for further processing.

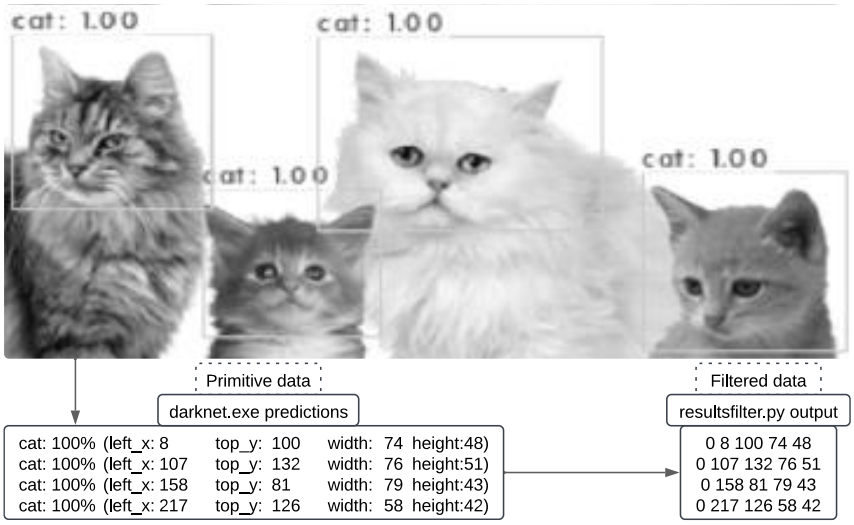

Figure 10 Ai coordinates predictions Graphic representation

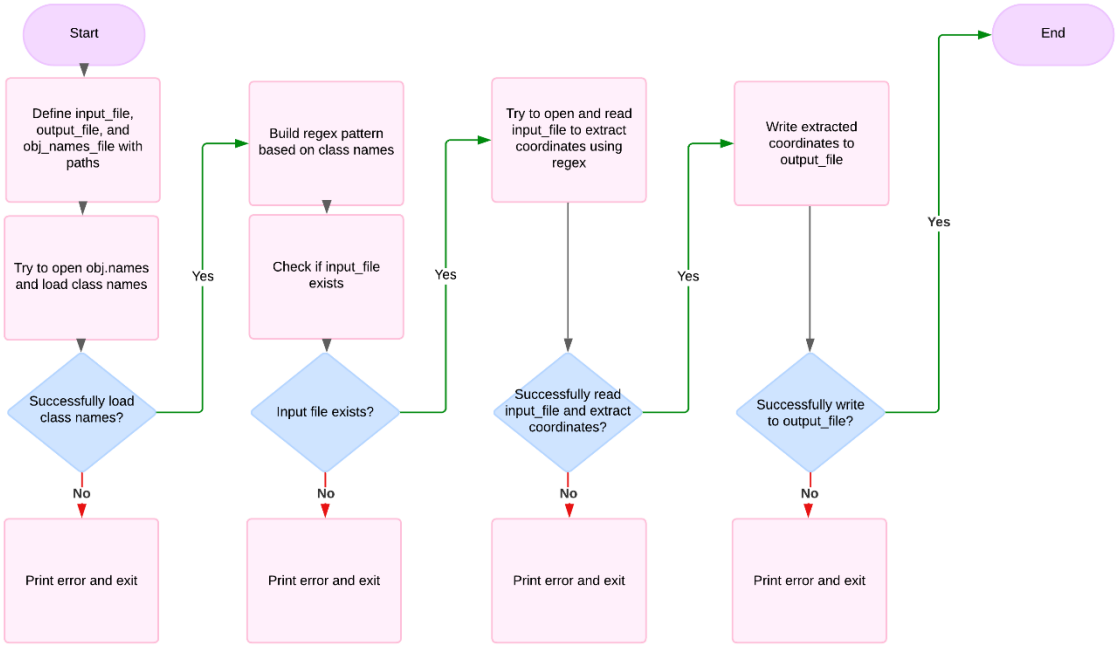

Figure 11 ResultFilter.py fluxogram

3.4.1.4 Obj.names and Obj.data

All files within this folder are essential for testing, training, and the overall functionality of the application. Among these, obj.names and obj.data play critical roles.

The obj.names file contains a list of class names, with each class name on a separate line. This file defines the categories that the model is trained to recognize.

The obj.data file includes several key pieces of information. It specifies the number of classes the model is trained to recognize. It provides the path to the file listing the training images and the path to the file listing the validation images. Additionally, it indicates the path to the file containing the class names, which links back to the obj.names file.

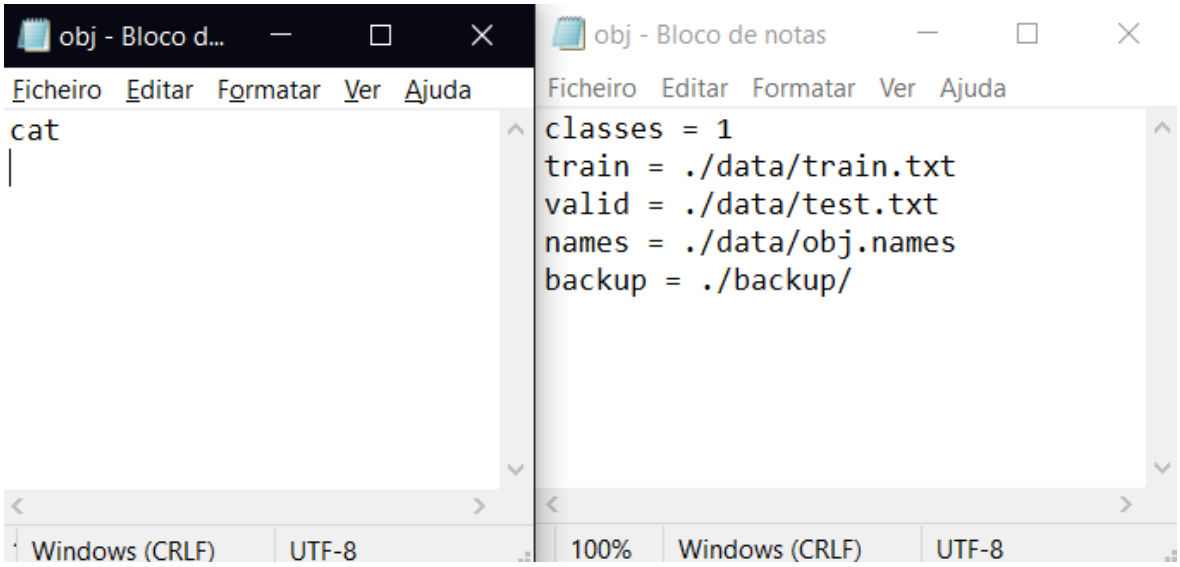

Figure 12 Obj.names and obj.text example

3.4.1.5 Test and train text files

In the data folder, there are two crucial text files: test.txt and train.txt. These files serve specific roles in the training and evaluation of the model.

train.txt: This file contains a list of file paths to the images used for training the model. Each line in train.txt specifies the path to a single training image. This allows the training process to load and process each image file during model training. The images listed here are used to teach the model to recognize and classify objects based on the annotations provided in corresponding label files.

test.txt: This file serves a similar purpose but for evaluation. It includes a list of file paths to images used for testing the model. Each line specifies the path to a single test image. These images are used to assess the performance of the trained model and ensure that it generalizes well to new, unseen data. The test set is separate from the training set to provide an unbiased evaluation of the model's accuracy and performance.

Both train.txt and test.txt are essential for properly managing the dataset and ensuring that the model is trained and evaluated using the correct images.

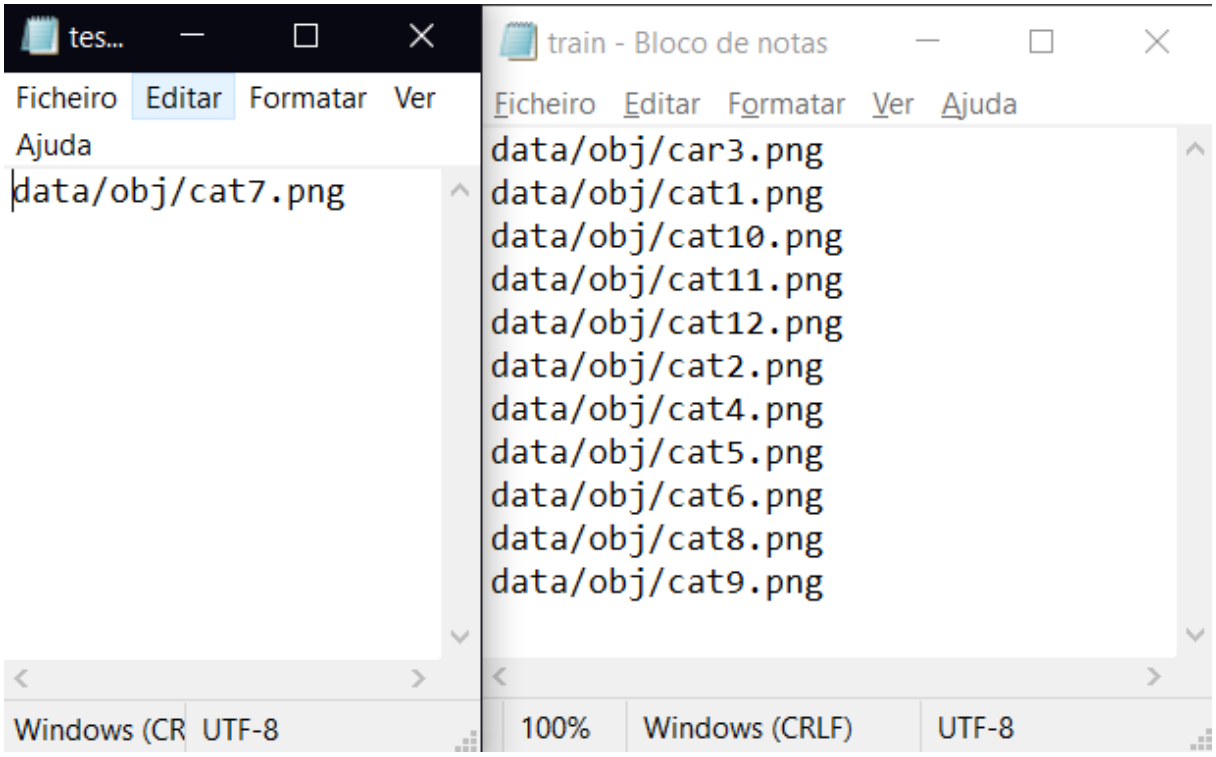

Figure 13 test.txt and train.txt example

3.4.2 Cfg folder

Inside the Darknet folder, the cfg folder holds crucial configuration files that define the architecture and parameters of neural networks. These files specify the network's structure, including the arrangement and types of layers such as convolutional and pooling layers, as well as activation functions. They also configure training parameters like the learning rate, batch size, and number of iterations.

For example, the yolov4.cfg file details the YOLOv4 model's layout and training settings, including the number of layers and their types. It provides the necessary instructions to build and train the network. Additionally, some configuration files may refer to pre-trained weights, which initialize the network with features learned from previous training, aiding in more efficient training for new tasks.

Overall, the cfg folder is integral to defining how neural networks are structured and trained, enabling users to customize and optimize their models.

3.4.3 Backup folder

Within the Darknet folder, the `backup` folder plays a crucial role in storing intermediate and final versions of the model's weights. During training, Darknet saves the model's weights periodically into this folder. This practice ensures that if training is interrupted or needs to be resumed, progress is preserved, and the model can continue from the last saved checkpoint.

The weights files in the `backup` folder, usually with a `.weights` extension, contain the network's learned parameters at different stages of training. Each file represents a checkpoint, allowing for the evaluation of the model's performance at various points. This functionality facilitates the comparison of different model versions and assists in selecting the best-performing weights based on validation results.

The `backup` folder ensures that the training process is resilient to interruptions, safeguarding computational investments and allowing for recovery and continuation of training without loss of progress.

3.5 Back-End Development

This chapter delves into the development of the API, focusing on its core components including C++, Python integration, and Darknet functionalities. The section will detail the implementation of various functions and other critical elements of the API, explaining how they interconnect and contribute to the overall system.

3.5.1 Opening Functions Logic

This program provides two options for starting a project: either creating a new file from scratch or opening an existing one.

3.5.1.1 Creating a new project

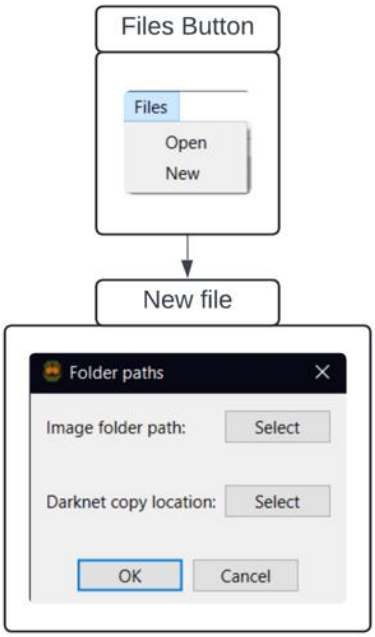

Figure 14 New project interface

To start a new project, two requirements must be fulfilled: a folder containing the images for tagging and training, and a location where the base Darknet folder will be placed. Upon specifying these paths, the images from the provided folder are copied to the data directory inside the Darknet folder. Additionally, the base Darknet folder, which is contained within the MasterChief directory, is copied to the specified local path.

In the application, a vector named imagelist is populated with the paths to the images, which facilitates the management and access of these images throughout the tagging and training stages.

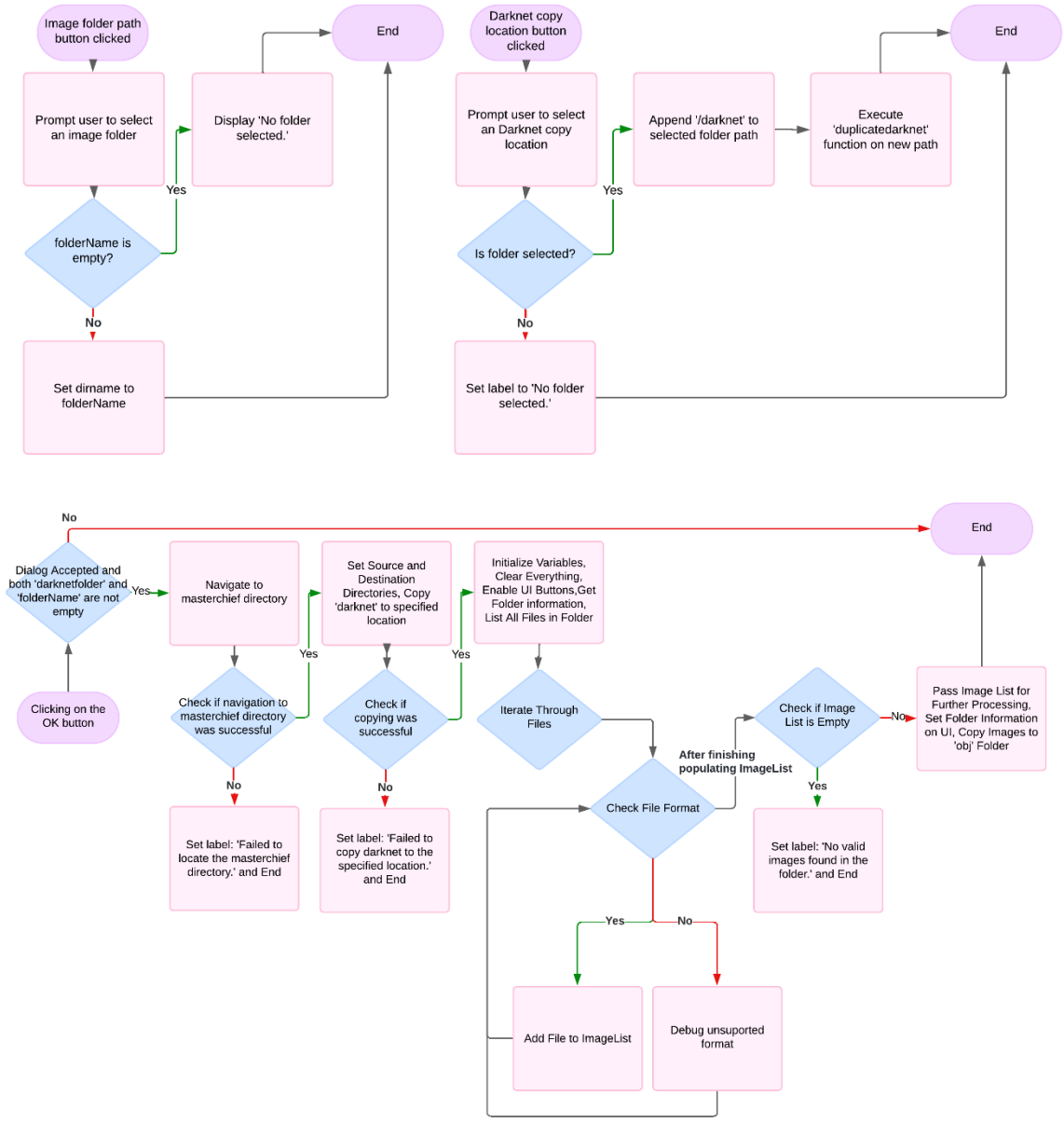

Figure 15 Image folder path, darknet copy path and ok button fluxograms

3.5.1.2 Opening an existing project

To open an existing project with preloaded images, tags, or trained weights, use the "File >> Open File" option from the menu bar. This action prompts a folder selection dialog. The selected folder must already contain images; otherwise, the project will open without images. Once the folder is chosen, all relevant variables are populated, and the graphical view is updated accordingly to reflect the contents of the folder (Figure 16).

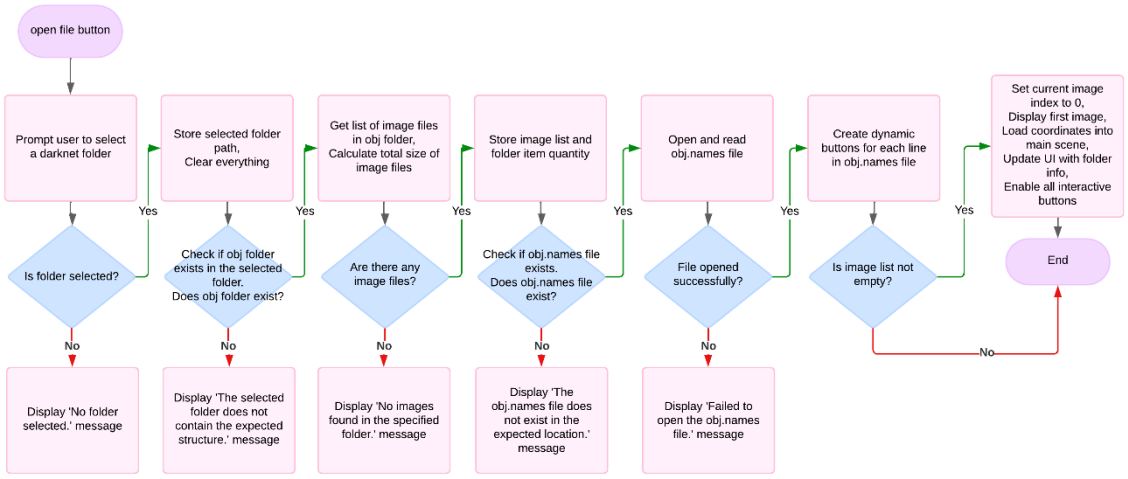

Figure 16 Existing project Opening fluxogram

3.5.2 Image Control Functions Logic

The application features a graphical view for displaying images, along with a widget list that organizes and allows searching through potentially large image datasets. Navigation is facilitated by macros that let users move to the next or previous image. The graphical view is integrated with a specially designed class that normalizes coordinate values to a range between 0 and 1, ensuring consistent scaling and accurate tagging across different images.

3.5.2.1 Image Loading

The function responsible for loading images into the `QGraphicsView` component within the application is encapsulated within a custom class named `GV`, the function name is images. This class is a subclass of `QGraphicsView`, and its primary function is to manage the display of images and handle mouse events, particularly focusing on the normalization of coordinates.

The `GV` class is designed to provide enhanced interaction with the graphical view, specifically catering to the needs of an image tagging application where precise coordinate tracking and normalization are essential. The class inherits from `QGraphicsView` and overrides key event-handling functions to customize behavior.

In this implementation, `GV` handles mouse movements and clicks through the overridden `mouseMoveEvent` and `mousePressEvent` methods. These methods capture the mouse's position within the scene, normalize these coordinates to a [0, 1] range, and then emit signals corresponding to these actions (Figure 17).

The `mouseMoveEvent` method is triggered whenever the mouse is moved within the view. It first converts the mouse position from the view's coordinates to the scene's coordinates using the

`mapToScene` method. The scene coordinates are then passed to the `normalizeCoordinates` method, which converts them into a normalized form. This normalization process involves adjusting the coordinates relative to the scene's dimensions, ensuring that the coordinates are expressed as a fraction of the scene's total size. The normalized coordinates are then emitted via the `mouseMoved` signal, allowing other components of the application to respond to mouse movement with precise, scalable data.

Similarly, the `mousePressEvent` method responds to mouse clicks by following the same process: it converts the click's position to scene coordinates, normalizes these coordinates, and emits the `mousePressed` signal. This approach ensures that interactions with the graphical view are consistently processed with normalized coordinates, facilitating more accurate image tagging and manipulation.

The normalization function, `normalizeCoordinates`, is a key component of this class. It operates by first checking if a scene exists; if not, it returns an empty point. If a scene is present, it calculates the normalized X and Y coordinates by determining their relative positions within the scene's bounding rectangle (`sceneRect`). This method effectively ensures that all mouse events are interpreted in a consistent manner, regardless of the actual size or scaling of the scene.

By promoting the `GV` class as a specialized handler for graphical interactions, the application ensures that all mouse events are accurately captured and processed, providing a robust foundation for image tagging and related tasks. The design of this class reflects an emphasis on precision and consistency, both of which are critical in the context of handling large datasets of images where exact coordinate tracking is required (Figure 18).

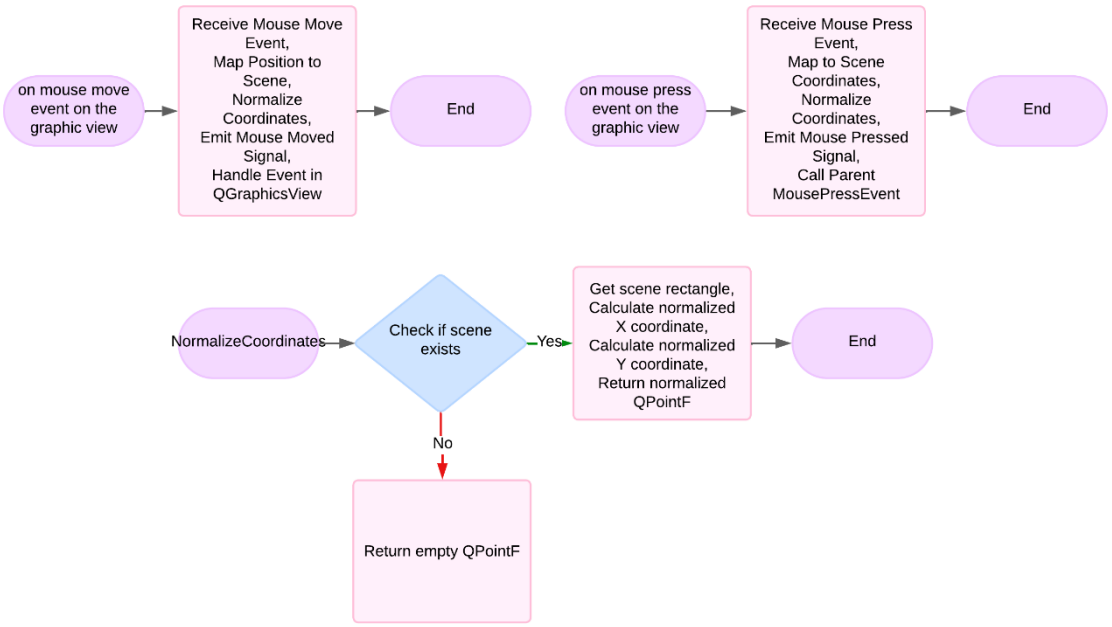

Figure 17 GraphicView Mouse Events and normalization of coordinates fluxograms

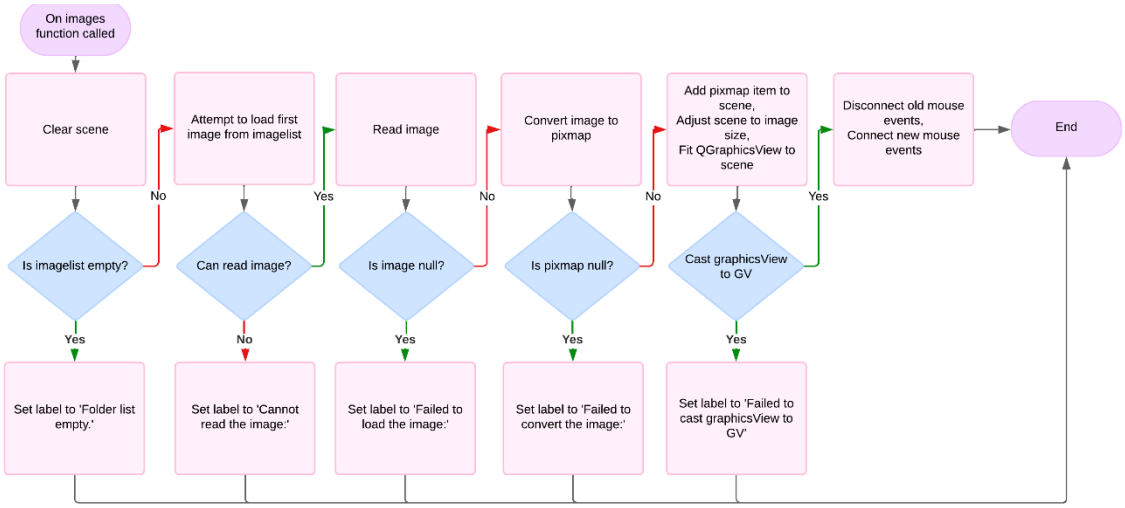

Figure 18 GraphicView image control function fluxogram

3.5.2.2 Image List

To streamline navigation through potentially extensive image datasets, a QListWidget was integrated into the application. This widget offers a user-friendly interface, enabling efficient management and interaction with images that are being tested, trained, or processed. The implementation of two key functions, on_widgetlist_clicked() and WidgetListset(), supports this functionality.

The on_widgetlist_clicked() function is triggered whenever an image is selected from the list. It ensures that the selected image is valid and within the appropriate index range. Upon successful

selection, it updates the display to show the chosen image, clears any previous selections if necessary, and loads the associated coordinates. This function also handles error conditions gracefully, such as when no image is selected or the index is out of range, by providing informative messages to the user.

The WidgetListset() function is responsible for adding images to the list. It takes a file path, extracts the file name, and adds it as an item to the QListWidget, allowing users to see and select the images they are working with. This function is kept simple yet efficient, ensuring that the interface remains responsive even when dealing with large numbers of images.

Together, these functions enhance the usability of the application, making it easier for users to navigate and work with their image datasets (Figure 19).

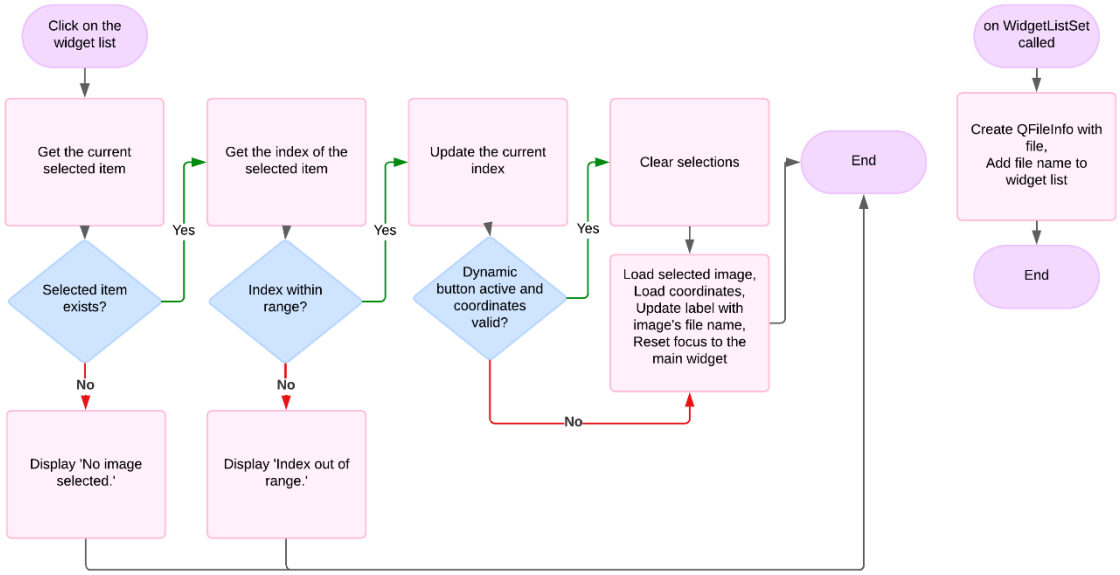

Figure 19 Widgetlist interaction fluxograms

3.5.2.3 Image navigation

To provide a more intuitive way to navigate through image selections, macros were integrated with functions designed to move between images easily: previousImage() and nextImage(). These functions enable quick transitions to the next or previous image in the list and are essential for handling large datasets where manual navigation would be cumbersome.

The previousImage() function is responsible for loading the previous image in the sequence. It clears the current scene, decrements the imageIndex if it's greater than zero, or wraps around to the last image if it's already at the first one. The new image is then displayed, the corresponding list item is selected, and the label is updated to show the image's file name. It also clears any previous selections and reloads the coordinates associated with the image.

The nextImage() function operates similarly but advances to the next image in the sequence. If it's already at the last image, it wraps around to the first one. After updating the image display, it also checks if the Darknet detection process is enabled. If so, it verifies whether the current image has already been processed (using alreadySelected). If the image has not been processed, the function attempts to send the image path to the Darknet process for AI-based selection.

Both functions ensure smooth navigation through the image dataset, handle edge cases (like wrapping around when reaching the end or start of the list), and integrate AI detection capabilities when applicable. This setup facilitates an efficient workflow, particularly in applications involving large numbers of images (Figure 20).

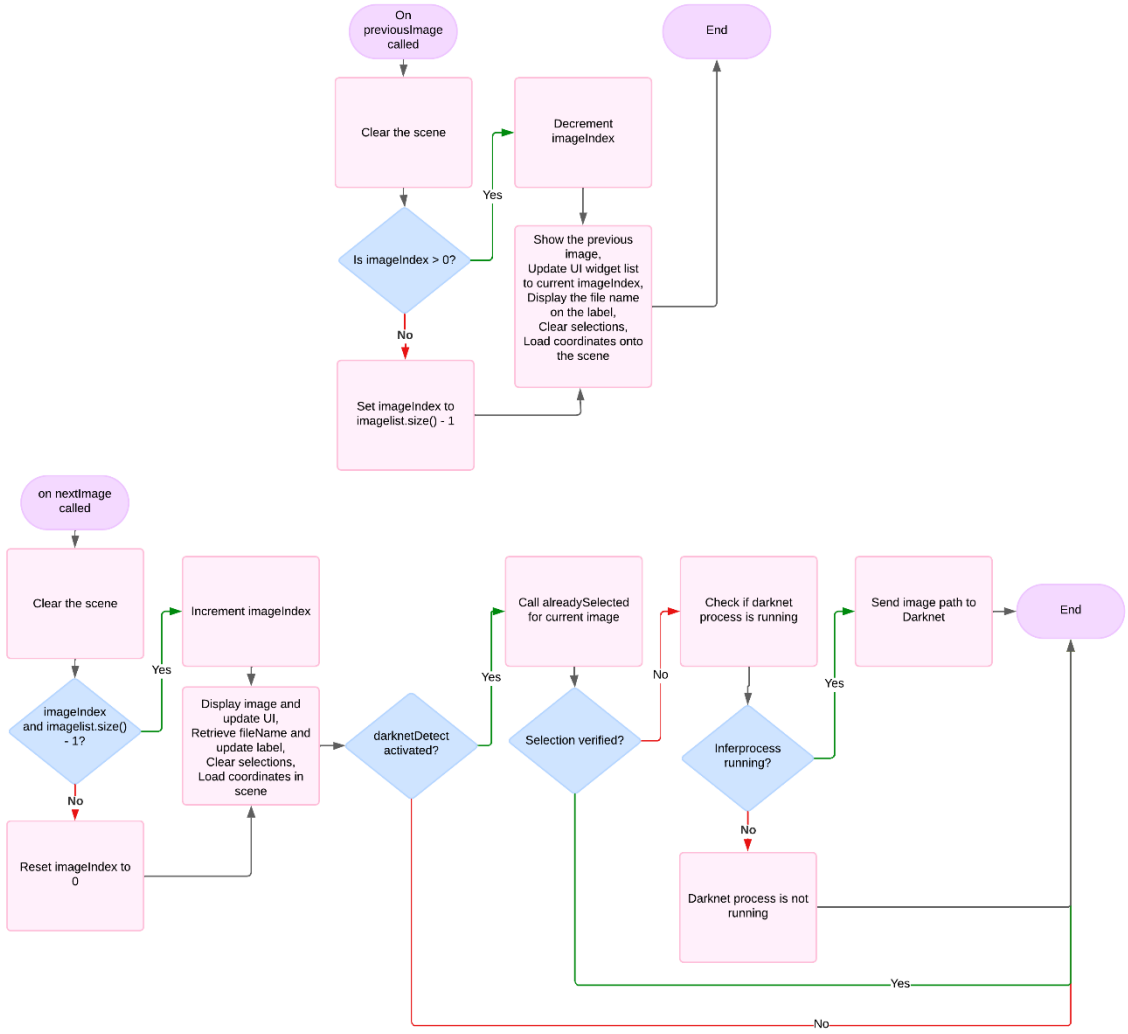

Figure 20 Image navigation functions fluxograms

This function ensures that when the test button is activated, the AI does not redundantly reprocess images that have already been tagged.

The function begins by creating a `QFileInfo` object to verify the existence of the image file. If the image does not exist, it logs an error and returns `false`. If the image is found, the function constructs the path for the corresponding text file, which should contain the selection coordinates. It checks for the presence of this text file using another `QFileInfo` object.

If the text file exists, the function logs that the image has already been tagged and returns `true`. If not, it returns `false`, indicating the need for AI-based selection. This approach optimizes the workflow by preventing unnecessary reprocessing of images. The function is efficiently designed with no apparent issues (Figure 21).

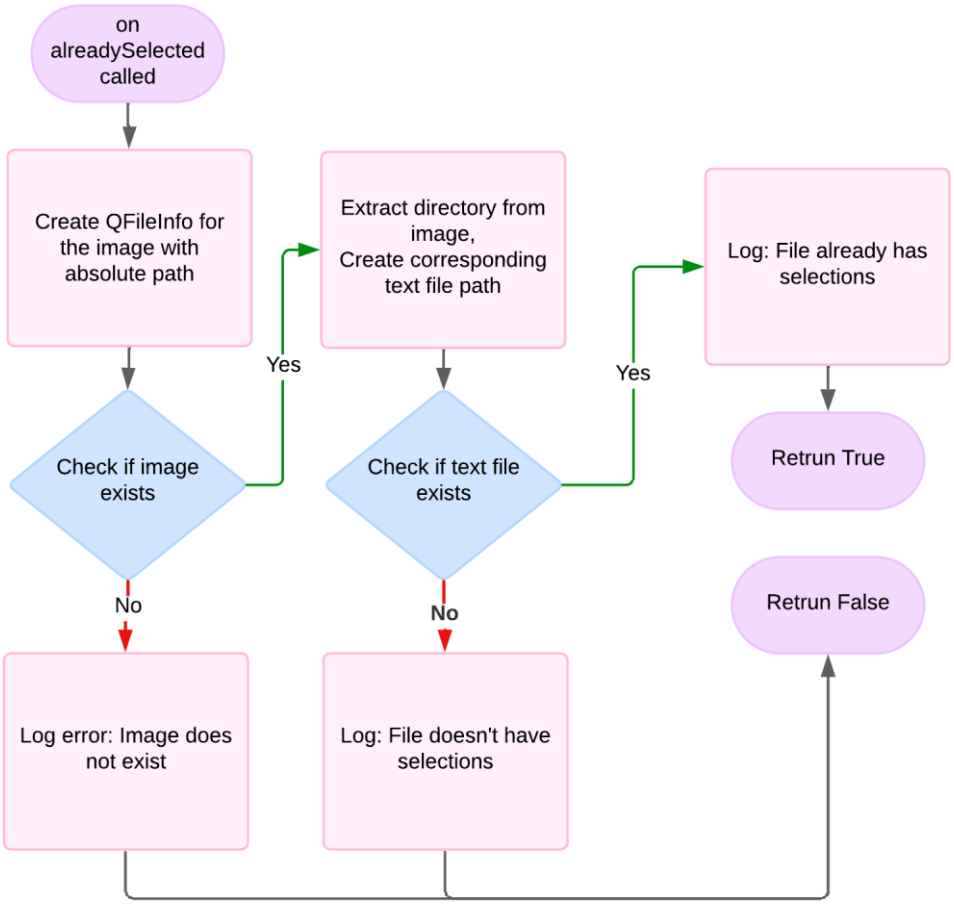

Figure 21 AlreadySelected function fluxogram

3.5.2.4 Images Reallocation

When a new project is created, the images are copied from the selected image folder to the Darknet data folder as described earlier. The function responsible for this operation is copyImagesToObjFolder, which ensures that the images are transferred correctly to the appropriate destination for further processing.

The function begins by setting up the source and destination folder paths. The duplicatedarknet function is called to potentially modify these paths, although this is not clearly necessary within the provided context.

The function then verifies the existence of the destination directory. If the destination folder does not exist, the function attempts to create it using mkpath("."). Failure to create the directory is logged, and the function returns early to prevent further errors.

Next, the function retrieves a list of all files in the source directory using entryList, filtering for files only. It then iterates over each file, constructing the full source and destination file paths. Before copying, the function checks if the destination file already exists. If it does, the existing file is removed to ensure that the copy operation does not fail due to a file conflict.

Finally, the function attempts to copy each image file from the source to the destination. Success or failure of each copy operation is logged (Figure 22**)**.

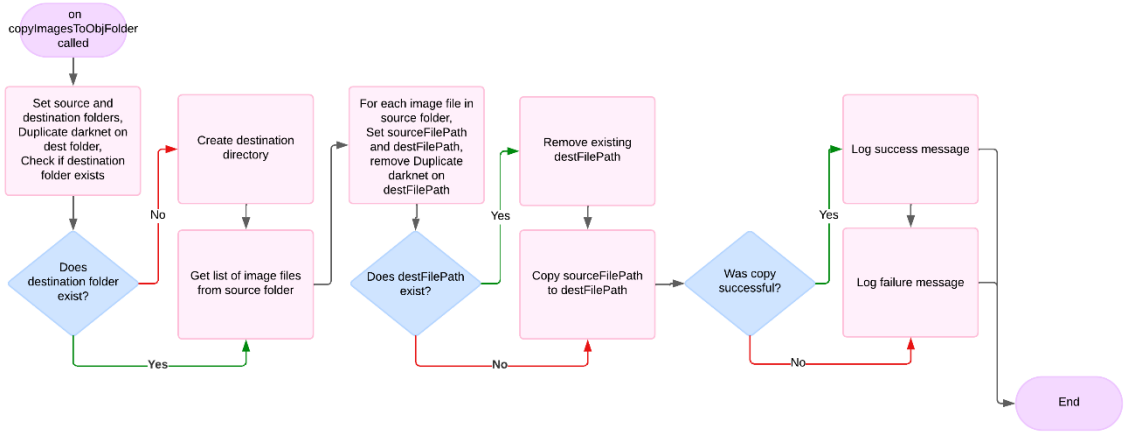

Figure 22 CopyimagestoObjFolder fluxogram

3.5.3 Class Control Functions Logic

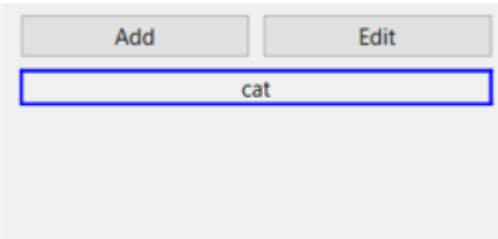

Figure 23 Class button and class control buttons

After opening or creating a project, the first task is to start tagging the images. To facilitate this, a method for organizing and naming the selections is required.

In the application layout, two key elements for managing tags are the "Add" and "Edit" buttons. Beneath these buttons is an area designated for class buttons. These class buttons, dynamically created through user interaction, represent different categories or types of selections on the graphic view.

The "Add" button allows users to define new classes for tagging, while the "Edit" button enables modifications to existing classes. This setup ensures that users can efficiently categorize and manage their selections, streamlining the process of annotating images within the application (Figure 23**).**

4.5.3.1 Class Add Button

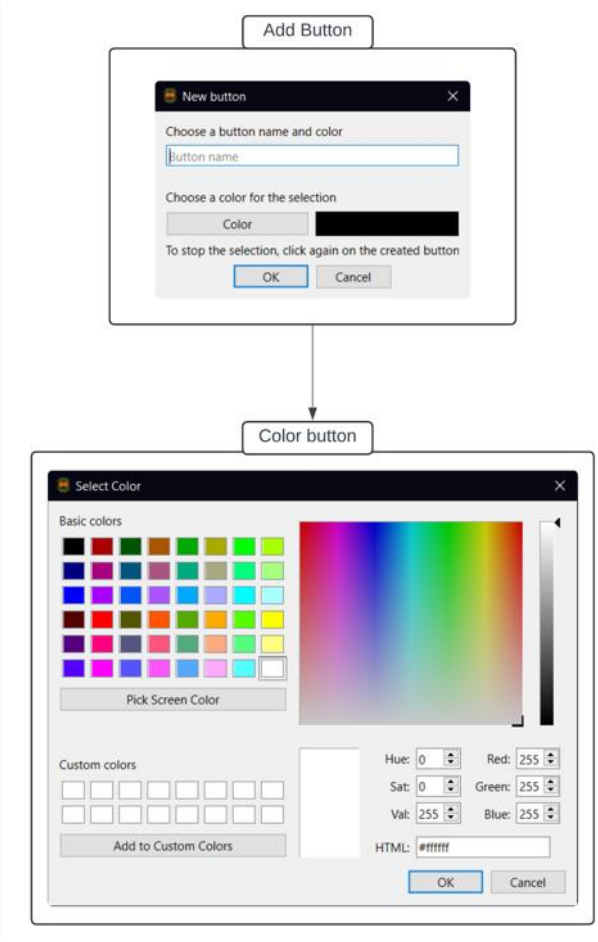

Figure 24 Add Button interface

The add button function handles the creation of new class buttons in the application, allowing users to define and customize new categories for tagging images (Figure 24, Figure 25**).**

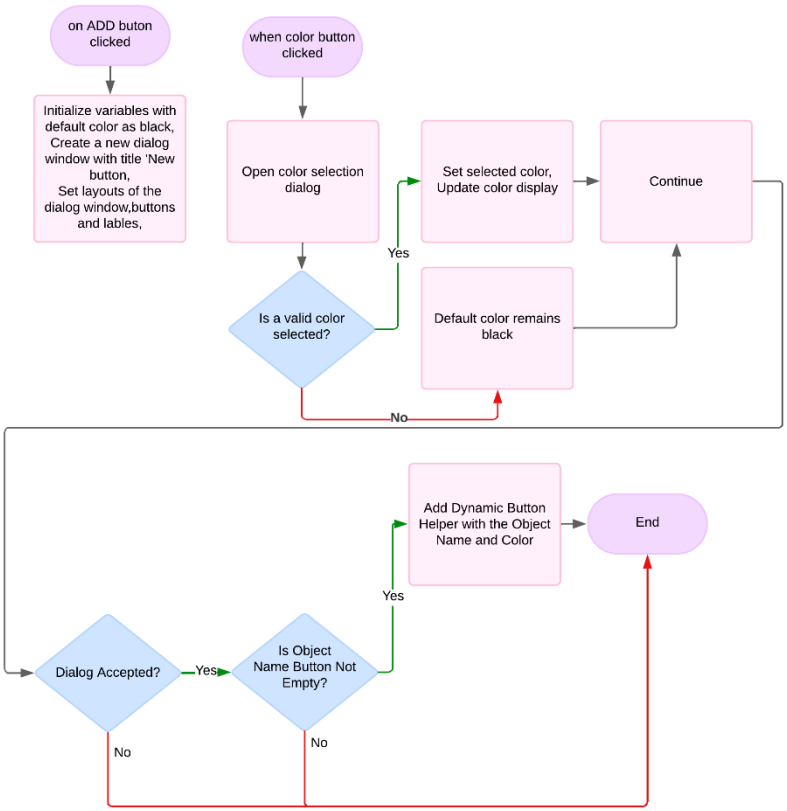

Figure 25 Add button Fluxograms

4.5.3.2 Class Edit Button

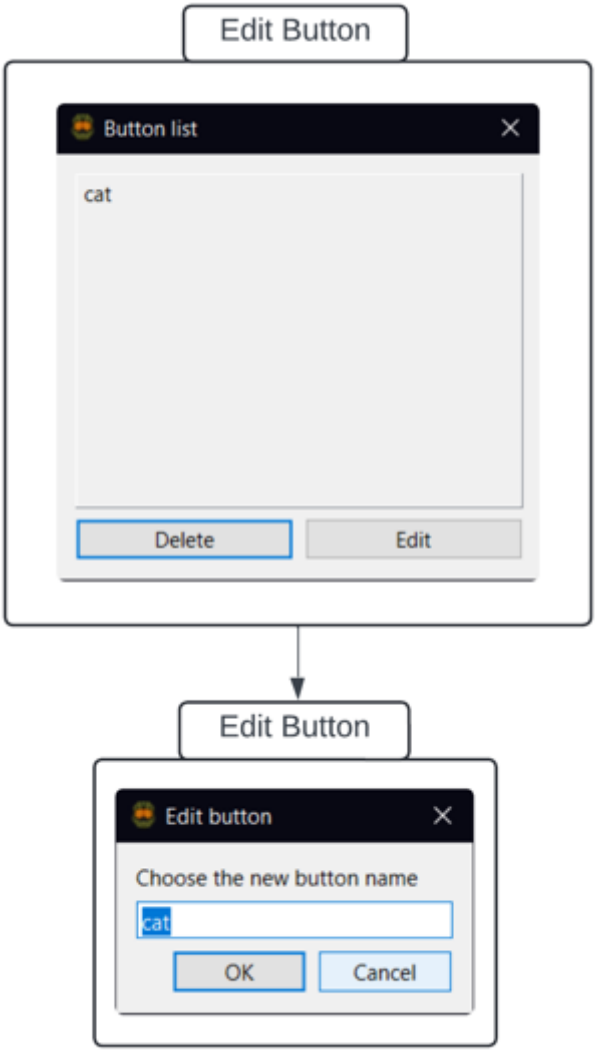

Figure 26 Edit button interface

After the class creation stage, it may become necessary to modify a class name to ensure clear identification and organization. To address this need, the Edit button was implemented to provide users with the ability to rename or delete class buttons (Figure 26**)**.

When the Edit button is activated, it opens a dialog listing all existing class buttons. Users can select a button to either rename it or remove it from the system.

The Edit action allows users to input a new name for the selected button, updating both the button's label and its associated directory. This ensures that the changes are consistent across the application.

The Delete action removes the selected button and its related data files, providing feedback on the success or failure of the operation. This functionality helps maintain an organized and accurate tagging system(Figure 27**)**.

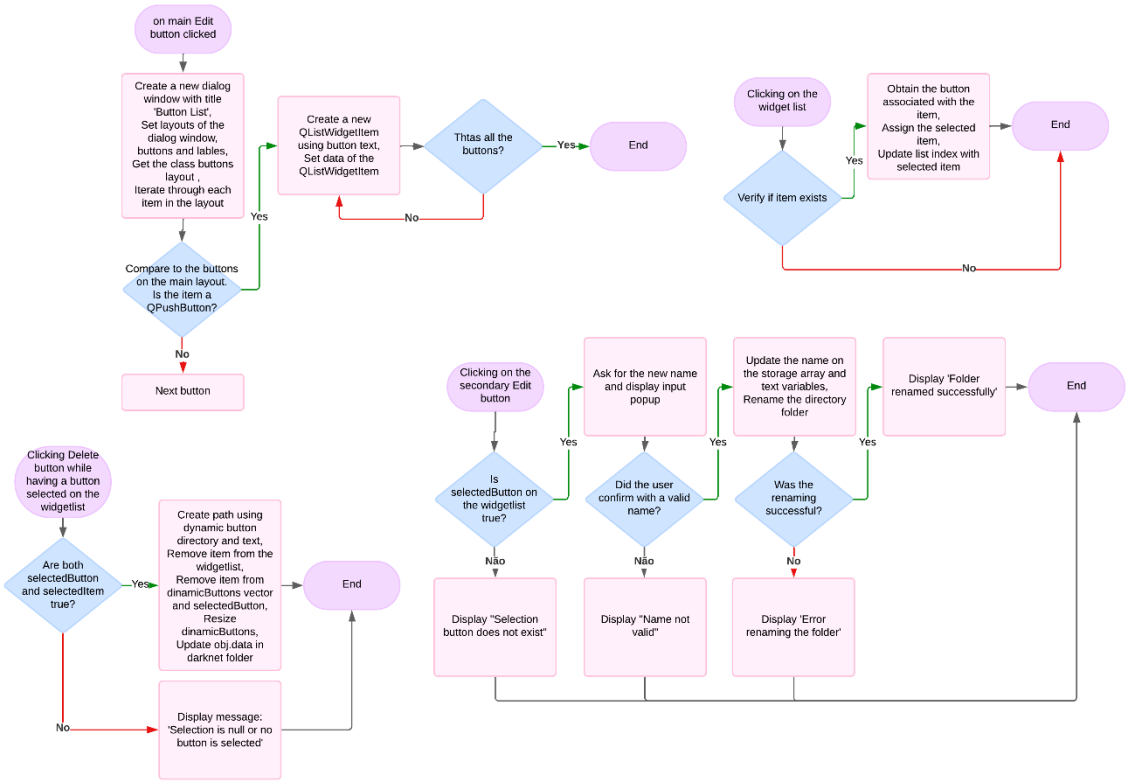

Figure 27 Edit button fluxograms

4.5.3.3 Class Button Action

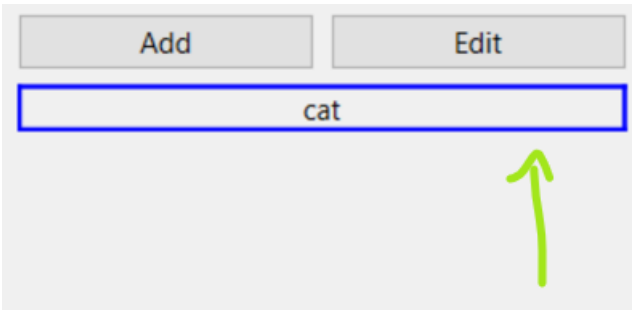

Figure 28 Class button highlighting

After a class button is added, clicking it initiates the selection process for images (Figure 28**)**. This process involves two clicks: the first to set the starting point and the second to define the selection area.

The on_button_clicked() function manages this process. When a button is clicked, the function determines the button's index in the dinamicButtons list. If selection mode isn't active, it starts the selection process by enabling mousePressEventEnabled = true. This enables the custom cursor and disables other buttons and controls to ensure the user can focus on making precise selections.

If selection mode is already active, the function ends the selection by saving the data, restoring all buttons and controls, and updating the graphic view. It ensures only one selection process is active at a time and transitions smoothly between modes (Figure 29**)**.

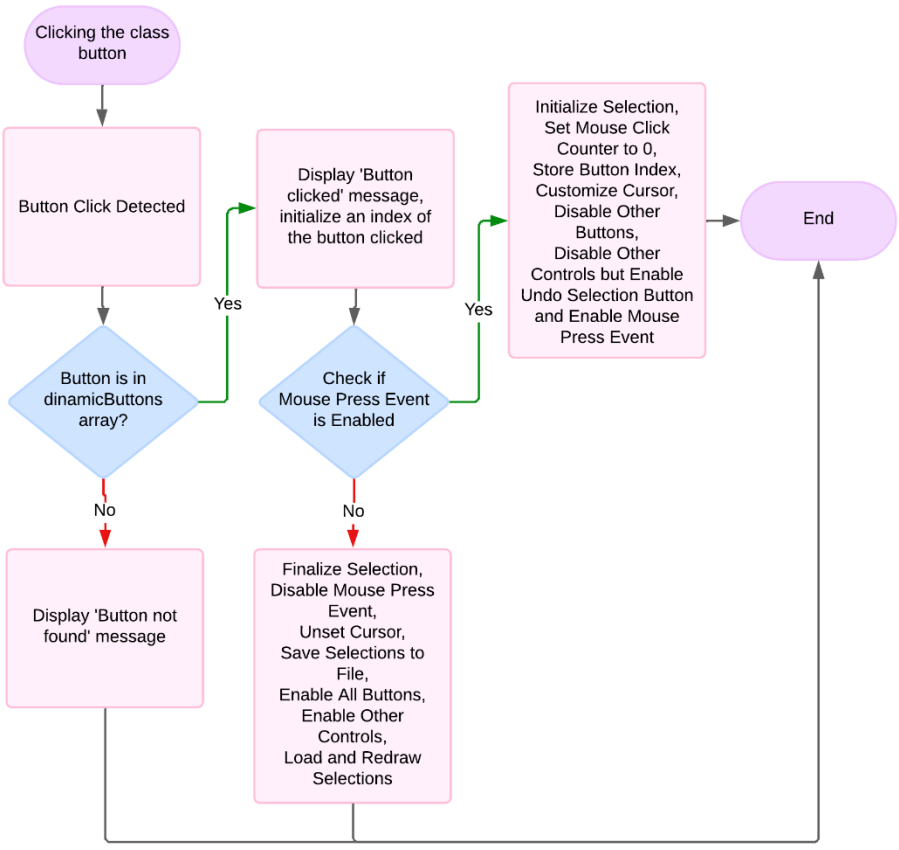

Figure 29 Class button interaction fluxogram

4.5.3.4 Class Button Aditton

When loading a project, the previously created classes need to be restored. This is done using the on_openFile_clicked() function, which retrieves image files and class names from the specified

darknet folder. Central to this process are the addDynamicButton() and addDynamicButtonHelper() functions, which recreate the class buttons dynamically with their respective colors.

The addDynamicButton() function begins by generating a unique color for each button using a hue offset, ensuring each class is visually distinct. It then calls addDynamicButtonHelper(), which handles the actual creation and styling of the button. The addDynamicButtonHelper() function updates the list of buttons, ensuring the layout and data structures are correctly modified to accommodate the new button.

The function also updates the obj.data file to maintain the link between the classes and the image annotations. The new button is inserted into the layout, and a click event is connected to the button to enable image selection based on that class. If the system failed to increment the hueOffset correctly or mismanaged the connection of the button's click event, it could result in overlapping colors or unresponsive buttons.

In terms of improvements, the functions seem well-structured. However, ensuring the hueOffset is properly cycled and confirming that all connections are made without errors is crucial. Additionally, verifying that the layout insertion accounts for edge cases, such as an empty or

improperly configured layout, would help avoid potential issues when adding dynamic buttons (Figure 30).

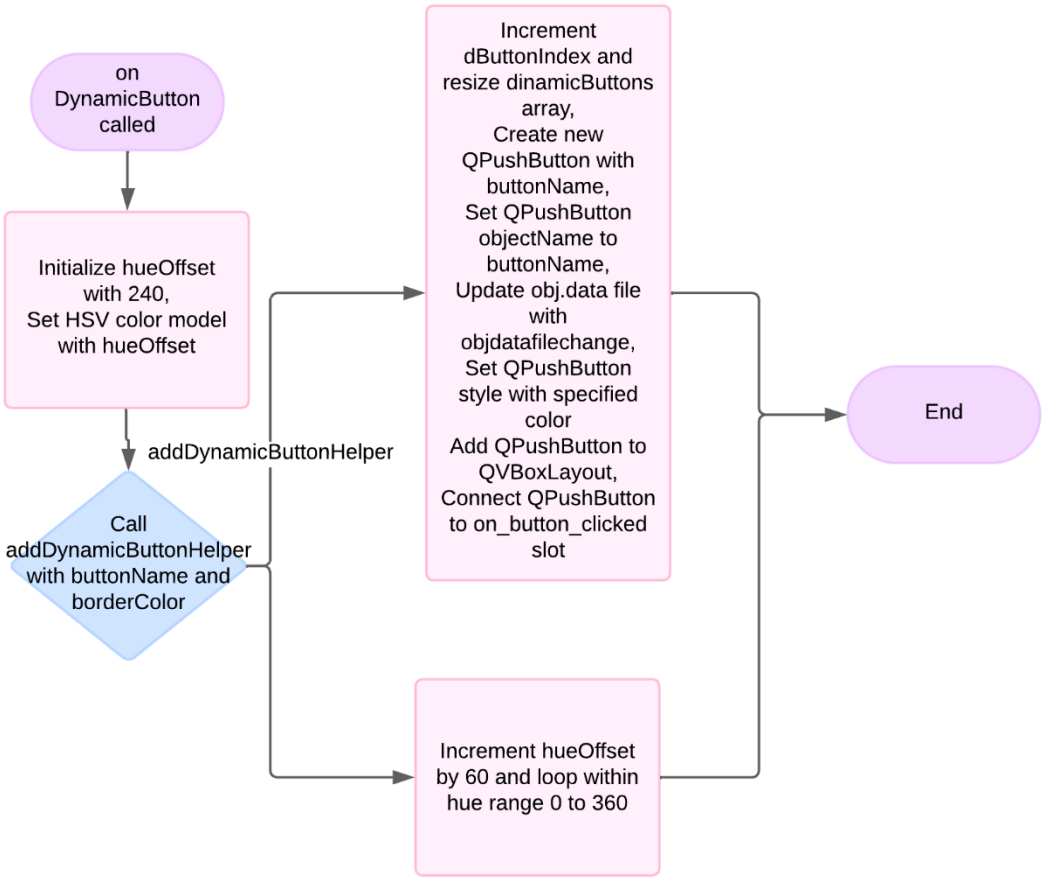

Figure 30 Class button addition fluxogram

3.5.4 Mouse Events Control Functions Logic

To effectively track mouse events within the graphical interface, a custom QGraphicsView class, named GV, was developed. This class is crucial for managing image interactions, particularly in tasks that require precise coordinate tracking, such as image tagging. The GV class is responsible not only for loading images but also for handling mouse events, ensuring that all interactions are processed with accuracy and normalized coordinates.

To enhance user interaction, a custom cursor was implemented. The customCursor() function creates a transparent, 600x600 pixel pixmap with crosshairs at the center, which replaces the default cursor when the user interacts with the QGraphicsView. This visual aid is designed to

improve the accuracy of selections made by the user, making the tagging process more intuitive and precise (Figure 31).

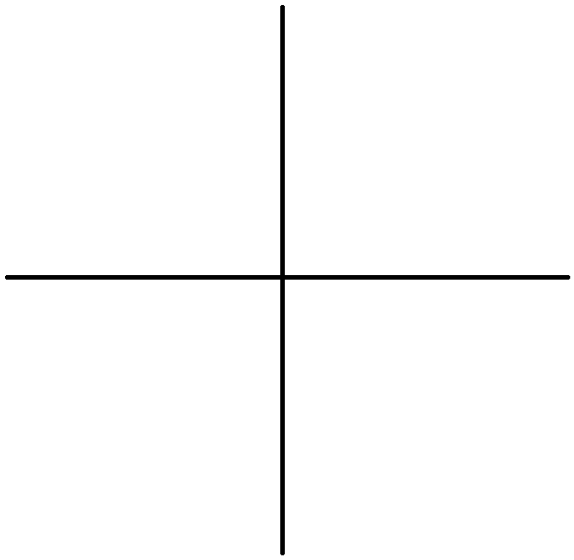

Figure 31 Custom cursor

Mouse events within the graphical view are managed by two key functions: updateMousePosition() and mouseClicked(). The updateMousePosition() function updates the UI label with the current mouse coordinates whenever the mouse moves, providing the user with real-time feedback on the cursor's position. This is critical for ensuring precise selections, especially when working with detailed images.

The mouseClicked() function is responsible for handling click events within the graphical view. This function operates under the condition that mousePressEventEnabled is set to true, indicating that the application is ready to capture and process mouse clicks for making selections. The selection process itself is divided into two stages. The first click captures the initial coordinates of the selection rectangle and stores them in the firstc vector of the currently selected button's data structure. The second click captures the opposite corner of the rectangle, storing these coordinates in the secondc vector. After the second point is recorded, the selection is visually represented on the scene, and the selection counter is updated.

The use of mousePressEventEnabled is crucial for managing the state of the application during the selection process, preventing unintended clicks from disrupting the user's workflow. However, the

system could benefit from additional feedback mechanisms to guide the user, especially when the application is not in selection mode. Furthermore, while the current implementation includes error handling for unexpected values in the selection process, there is room for improvement, such as offering recovery options if an error occurs (Figure 32).

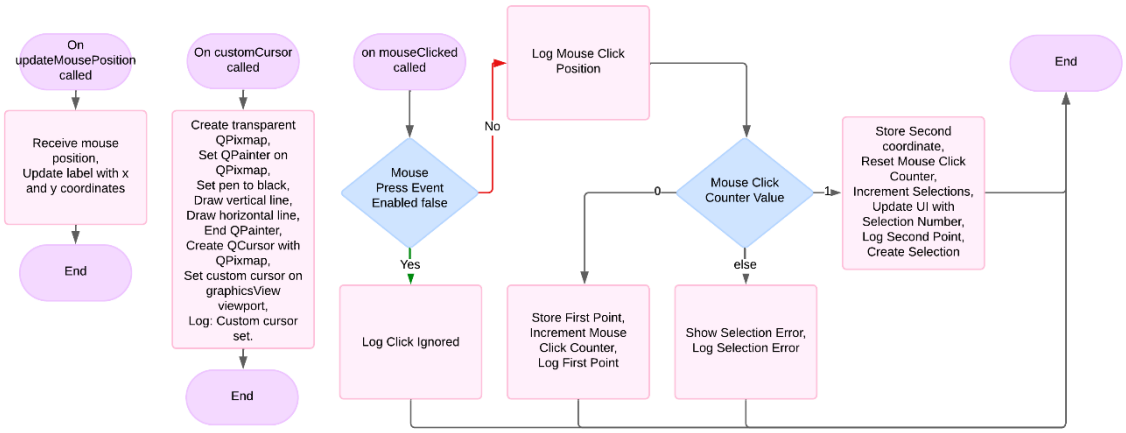

Figure 32 Mouse functions fluxograms

3.5.5 Selections Control Functions Logic

The Selections stage is a crucial part of the application, designed to allow users to interact with images by selecting specific areas for tagging or other purposes. This stage involves several processes that ensure accurate and efficient selection, providing users with visual feedback and control over their actions. Below, we explore the key steps that take place before, during, and after the Selections stage.

3.5.5.1 Selection Creation

The `createSelection` function is pivotal in rendering user-defined selections onto a graphical view. It retrieves selections stored in the program's variables, normalizes them to match the image dimensions, and visually represents them within a `QGraphicsScene`. This function is integral to the image annotation process, providing precise visual feedback on the selections made by the user.

The function begins by clearing any previous drawings from the `QGraphicsScene`, ensuring that the new selections are displayed on a clean slate. It loads the current image, adds it to the scene, and calculates its dimensions-critical for scaling the normalized selection coordinates.

Next, the function iterates through the stored selection data, which are pairs of coordinates associated with each dynamic button, representing different classes or labels. These coordinates, initially normalized to a [0, 1] range relative to the image's size, are scaled back to the actual pixel dimensions based on the image's width and height.

For each selection, a rectangle is created using these scaled coordinates, ensuring that the selection is accurately positioned and sized on the image. The rectangle is then normalized to correct any inconsistencies in coordinate order. Each rectangle is visually styled with a colored border, determined by the associated dynamic button's color, providing clear visual differentiation between various selection types or classes. Finally, the rectangles are added to the scene, making them visible on top of the image.

This function is called at various points throughout the code, ensuring that selections consistently appear in the graphical view whenever necessary. Unlike other functions, `createSelection` does not include built-in error protections because the functions that invoke it already manage error handling. This design choice keeps `createSelection` focused solely on its primary task of rendering selections, relying on external safeguards to ensure its inputs are valid (Figure 33).

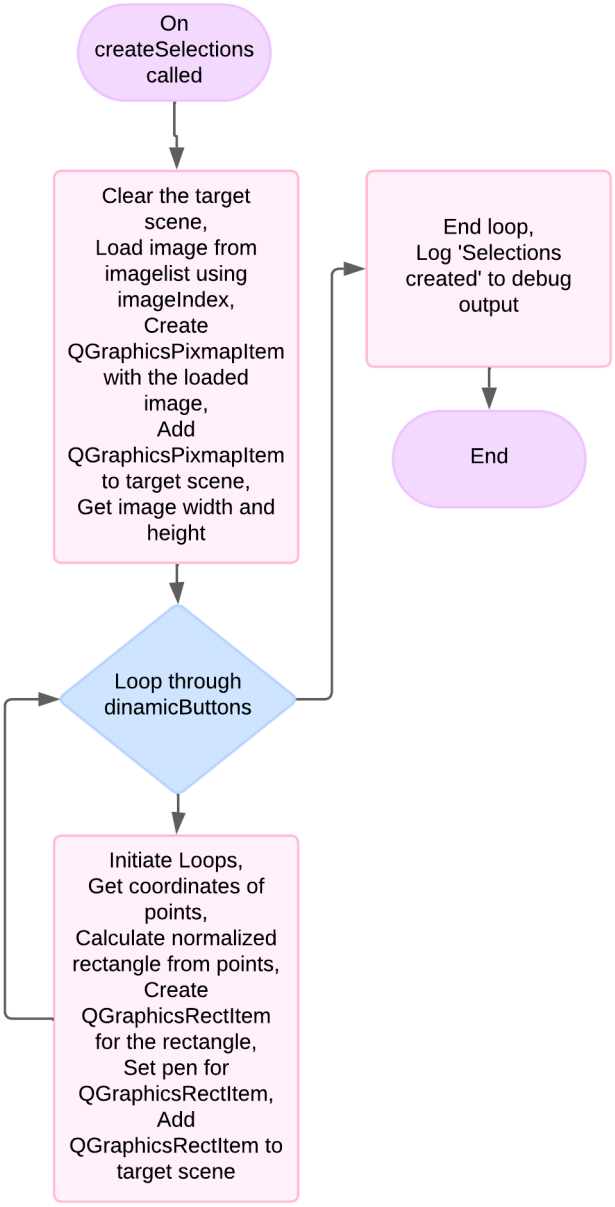

Figure 33 Selection Creation fluxograms

3.5.5.2 Load Coordinates

The `loadCoords` function plays a vital role in managing the loading and reloading of selection coordinates within the application. It ensures that any selections previously made are accurately represented both in the program's internal variables and visually within the graphical view.

The function begins by clearing the current graphical scene and resetting the coordinates associated with each dynamic button. This step is crucial to prevent any old or conflicting data from interfering with the new set of coordinates that will be loaded.

Next, it constructs the file path for the current image's coordinate file, which contains the selection data. The function includes a safeguard to correct any potential directory naming issues, such as the presence of redundant directory paths. Once the correct path is determined, the function checks if the coordinate file exists and attempts to open it for reading.

As the function reads the file line by line, it parses the selection data, ensuring that each line contains a valid set of coordinates. The coordinates are expected to be in a specific format, and the function checks this format to prevent errors. If the data is valid, the function calculates the top-left and bottom-right points of each selection box. These points are stored in the `firstc` and `secondc` lists of the appropriate dynamic button, corresponding to the class index.

Finally, after all coordinates have been processed and stored, the `createSelection` function is called. This ensures that the selections are rendered on the target scene, allowing users to visually confirm the selections corresponding to the current image. By doing so, the `loadCoords` function integrates seamlessly with the rest of the application, providing a reliable method to manage and display selection data across sessions (Figure 34).

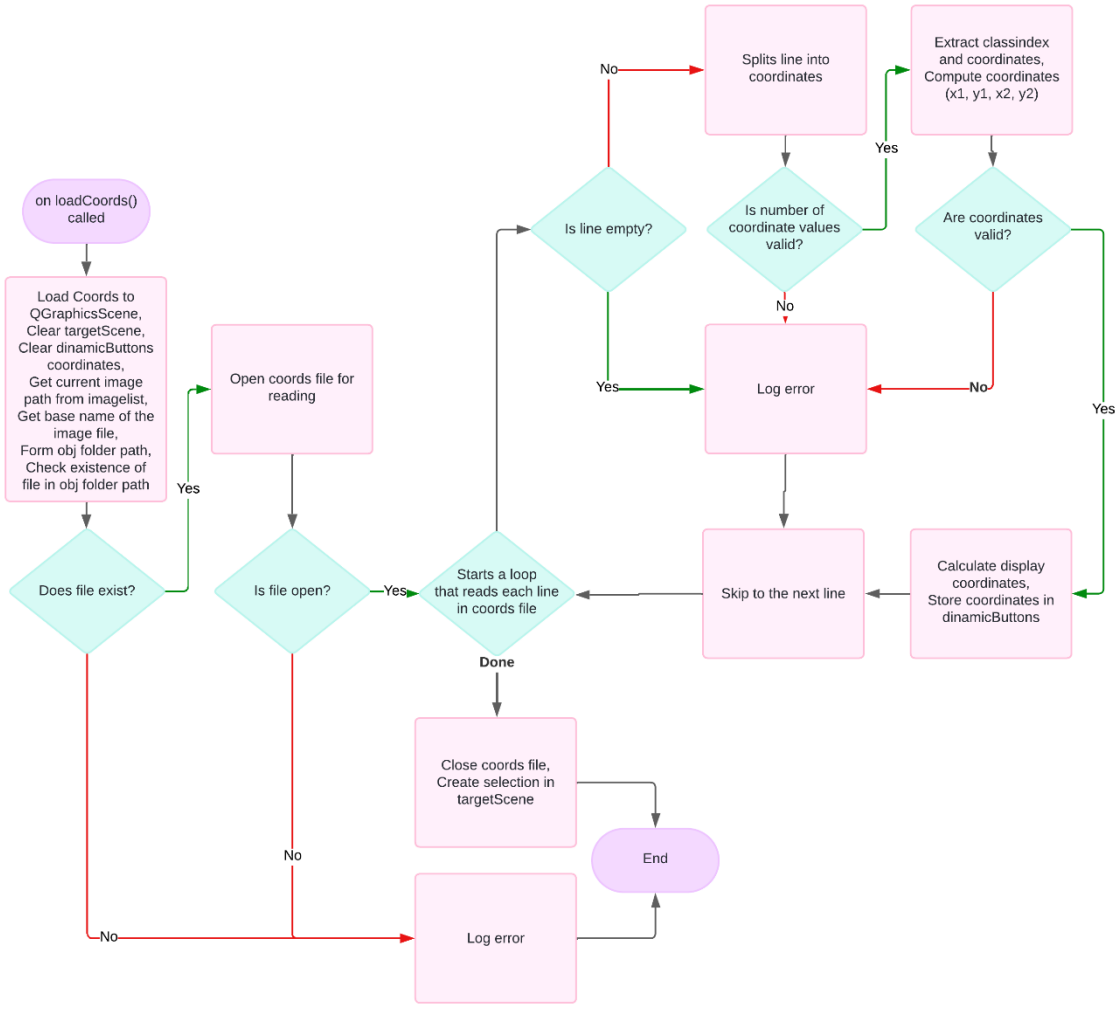

Figure 34 Coordinates loading fluxogram

3.5.5.3 Add Coordinates to a file

This function is designed to add coordinates generated by AI predictions to a corresponding `<imagename>.txt` file, which is essential for maintaining consistent data across the image tagging and labeling process.

The function begins by determining the path to the file that contains the AI-predicted coordinates. It attempts to open this file, reading all the lines into a `QStringList` called `coordinatesList`. If the file is empty or fails to open, the function logs an appropriate error message and exits.

Once the coordinates are loaded, the function retrieves the path of the current image and extracts its base name and dimensions. This information is necessary for correctly placing the coordinates

within the context of the image's size. The image file is then copied to the appropriate directory within the project's structure to ensure consistency.

Next, the function opens a corresponding `.txt` file where the coordinates will be stored. If this file cannot be opened for writing, the function logs an error and exits.

For each line of coordinates in the `coordinatesList`, the function parses the class index and the bounding box coordinates (x, y, width, and height). It then converts these coordinates into a center-based format, calculating the center point of the bounding box. These center-based coordinates are further normalized by dividing them by the image's width and height, ensuring that they fit within a [0, 1] range.

Finally, the normalized coordinates, along with the class index, are written to the output `.txt` file. The function provides debug output to display both the original and normalized coordinates, helping to verify the accuracy of the transformation.

This function is crucial for integrating AI-generated data into the labeling workflow, ensuring that predictions are correctly aligned with the image data. It handles file operations, data transformation, and normalization, ensuring the data is consistent and usable across different stages of the project (Figure 35).

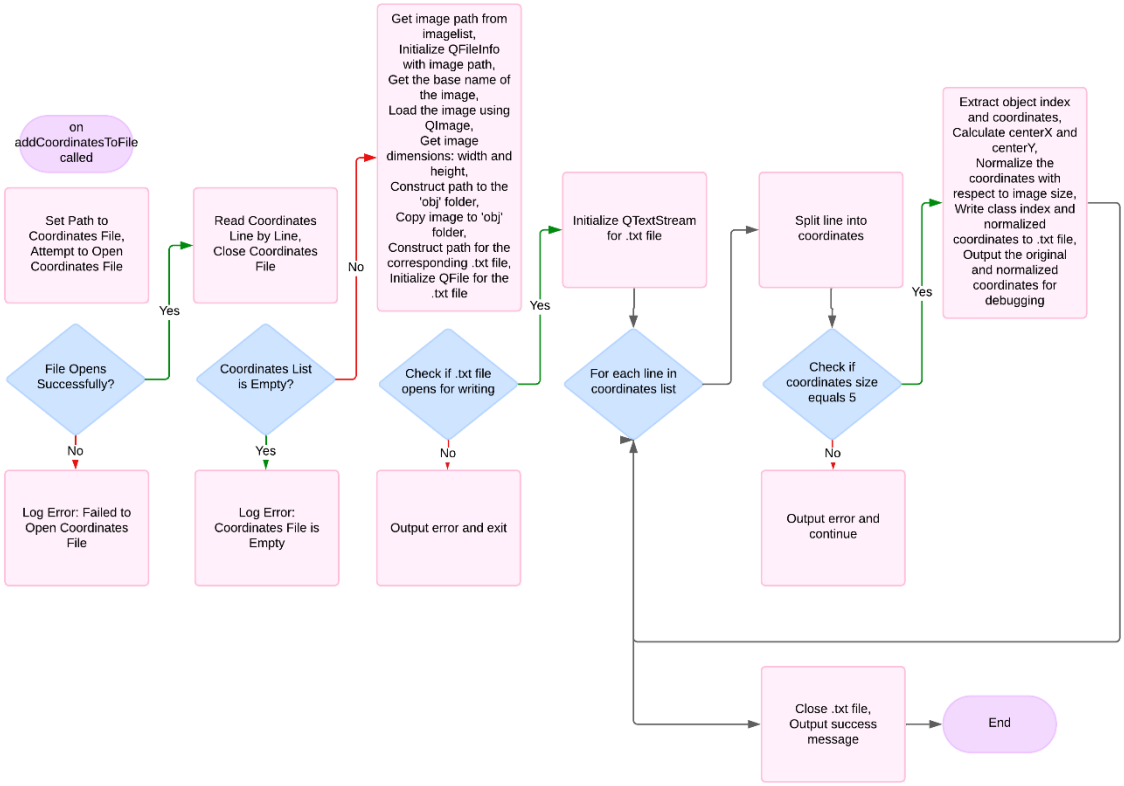

Figure 35 Coordinates addition to file function fluxogram

3.5.5.4 Delete Coordinates from a File

The `` function is designed to remove specific coordinate entries from a text file associated with an image within an annotation system. This function ensures the accuracy and currency of the image's annotation data.

The function initiates by constructing the path to the text file that contains the coordinates for the currently selected image. It derives this path using the image's base name and the location of the darknet folder. The function includes a check to adjust the path if it contains a specific directory structure, ensuring the correct file path is utilized.

Upon determining the file path, the function verifies the existence of the file. If the file is not found, it logs an error message and terminates. If the file exists, it opens the file in read-only mode and reads all lines, which represent various annotations or selections, into a list.

The function then validates the `selectionIndex` to ensure it falls within the valid range of lines read from the file. This step prevents attempts to delete a line that does not exist, which could lead to errors.

If the `selectionIndex` is valid, the function removes the corresponding line from the list. This line represents the coordinates of a particular selection or annotation.

Finally, the function reopens the file in write mode, truncates its contents, and writes the modified list of lines back to the file. This operation updates the file to reflect the deletion of the specified selection, ensuring the file's data remains accurate and current.

In summary, the `deleteCoordinatesFromFile` function manages the deletion of specific coordinate data from an image's annotation file efficiently, maintaining the integrity and relevance of the annotation data (Figure 36).

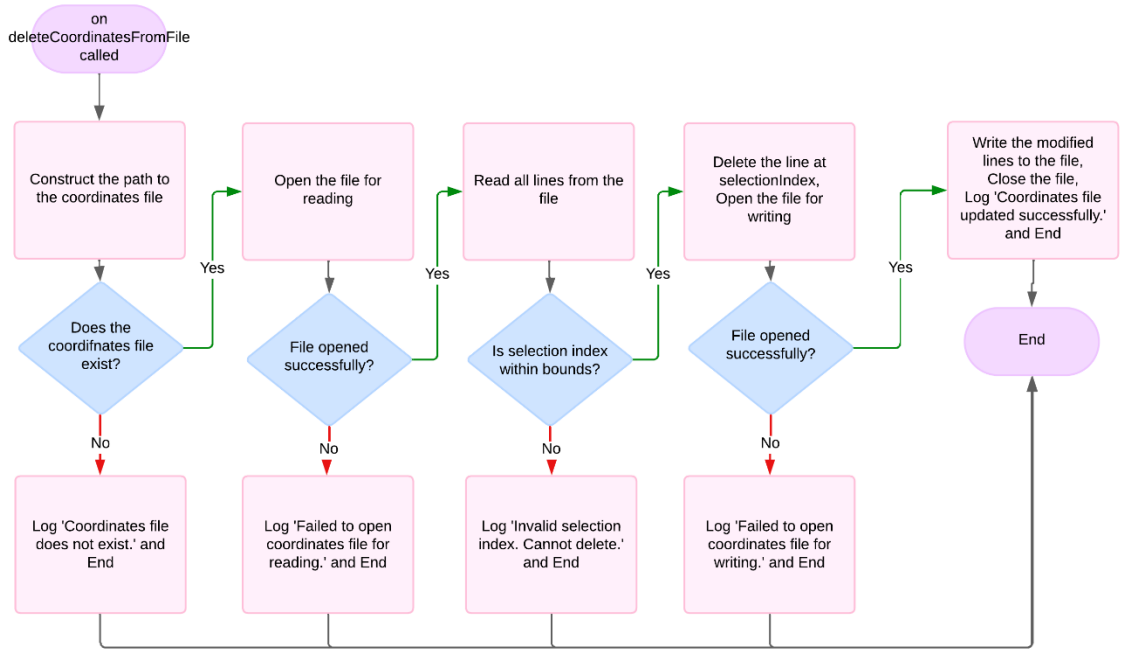

Figure 36 Coordinates addition to file function fluxogram

3.5.5.5 Undo Selection Button

A button was implemented to handle possible user errors during the selection stage, allowing the user to undo the last selection made if it was done incorrectly. This functionality is accessible only while still in the selection stage.

The `on_undoSelec_clicked` function is responsible for undoing the last selection made by the user. When the button is clicked, the function first checks if the current `dButtonIndex` is within the valid range of `dinamicButtons`, which are the dynamically created buttons representing different selection categories.

If the selected button index is valid and there are existing selections associated with it, the function proceeds to remove the last set of coordinates (the most recent selection) from both the `firstc` and `secondc` lists of that button. These lists store the coordinates that define each selection.

After removing the last selection, the function calls `createSelection` to redraw the remaining selections on the main graphics scene, ensuring the view is updated to reflect the changes.

The function then updates the displayed number of selections for the current button, ensuring the user is aware of how many selections remain. If there are still selections left, the Undo button remains enabled; otherwise, it is disabled to prevent further undo attempts. If no valid dynamic button is selected, or if there are no selections to undo, the function updates the UI to notify the user and disables the Undo button to avoid any invalid operations.

This function effectively allows users to correct mistakes during the selection process, enhancing usability and ensuring that selections are accurately managed (Figure 37**).**

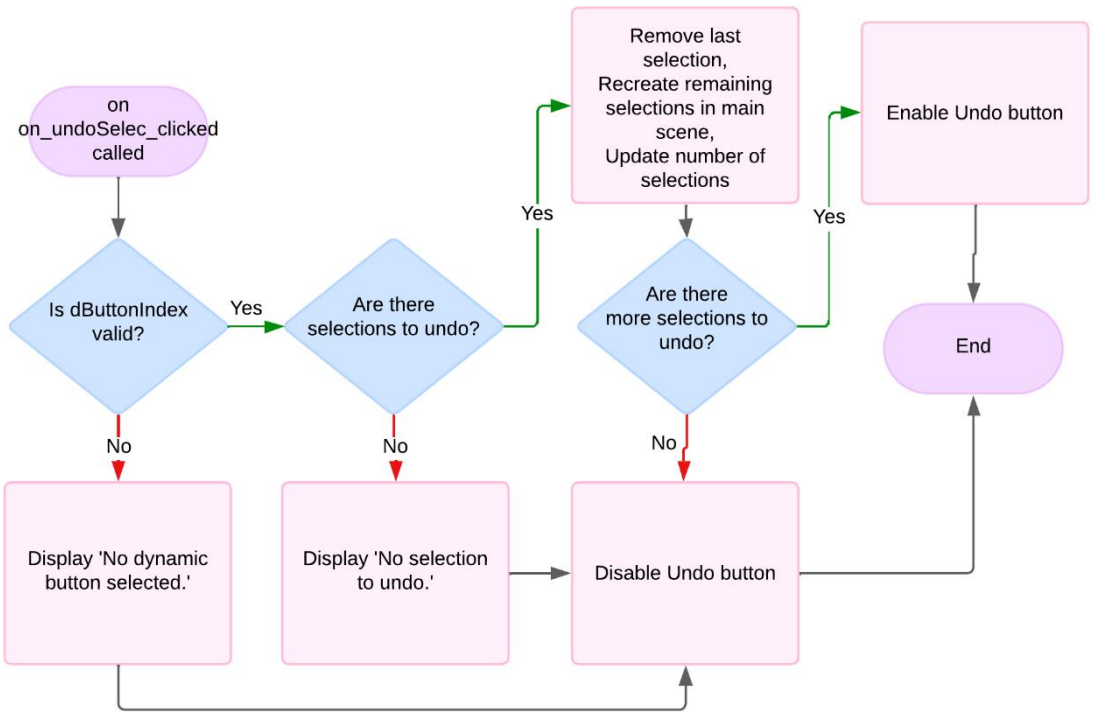

Figure 37 Undo selection button fluxogram

3.5.5.6 Clear Selections

The `clearSelections` function is widely used in the code to reset all existing selections made across dynamic buttons. It loops through each button in the `dinamicButtons` list, clearing the `firstc` and `secondc` lists that store the selection coordinates. This action effectively removes any previously made selections. Additionally, the function resets the `selectionNr` for each button to zero, ensuring that the selection count is updated to reflect that no selections are currently stored. Finally, it logs a message to confirm that all selections have been cleared. This function is essential for maintaining a clean state before new selections are made (Figure 38).

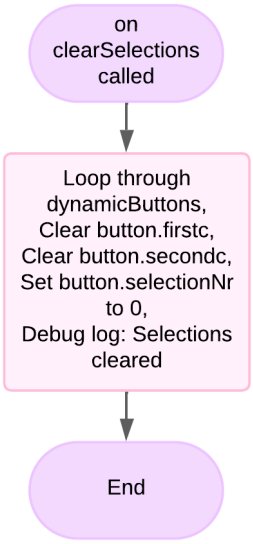

Figure 38 ClearSelections fluxogram

3.5.5.7 Clear Everything

The `clearEverything` function completely resets the interface, clearing all associated data and resources. It starts by clearing the widget list in the user interface, removing all items from it. Next, it checks if the `dinamicButtons` list is not empty. If it contains items, the function iterates through each button, deletes the button widget, clears the lists storing coordinates, resets the selection count to zero, and deletes the associated scene if it exists. After processing all items, it clears the `dinamicButtons` list itself. The function then clears the `imagelist`, resets the `imageIndex` to 0, sets the item quantity to 0, and resets the dynamic button index to -1, indicating that no button is

currently selected. Finally, it clears the main scene, removing any graphical items displayed. The `clearEverything` function is a comprehensive reset mechanism that ensures all elements of the interface, including dynamic buttons, images, and scenes, are completely cleared, leaving the interface in a fresh state, ready for new input or tasks (Figure 39**).**

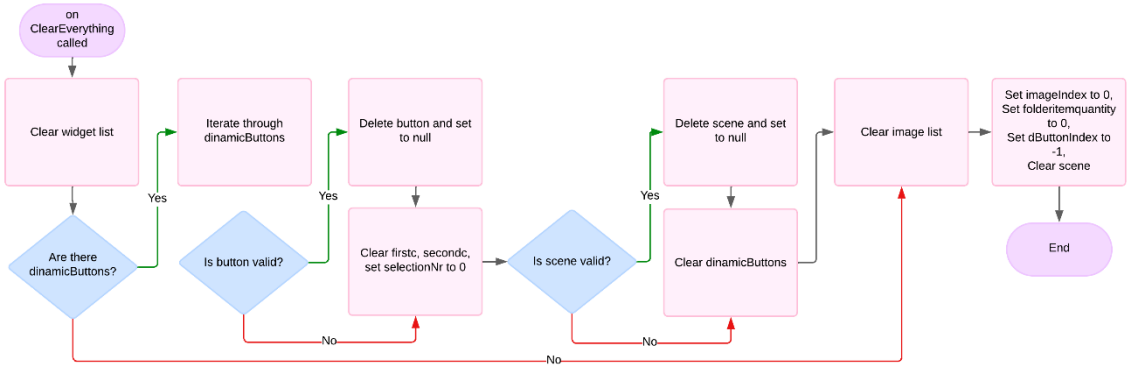

Figure 39 ClearEverything fluxogram

3.5.5.8 Image Coordinates File Creation

The `fileCreate` function is responsible for generating and saving a text file that contains annotation data for the current image in the image annotation system. It starts by determining the file path of the image currently being worked on, extracting its base name (without the extension), and constructing the path to the `obj` folder within the darknet directory.

Once the path is set, the function copies the image file into the `obj` folder. Following this, it attempts to create or clear a corresponding text file, named after the image, to store the coordinates of the selections made by the user. This file is essential as it records the bounding boxes for the annotations.

The function then loops through the `dinamicButtons` array, which stores the selections made by the user. For each selection, it calculates the center point, width, and height of the bounding box by averaging the coordinates of the opposite corners. These values, along with the class index (indicating the type of object), are written to the text file in a format that can be used by object detection models.

If the file creation and writing process is successful, the function updates the user interface to indicate that the file has been successfully created and logs this outcome for debugging purposes. If it fails at any point, it logs an error and informs the user through the interface that the file creation process did not succeed.

This function plays a critical role in the annotation system by ensuring that all selections made during the annotation process are accurately saved in a text file, which can later be used for training or validating object detection models (Figure 40**)**.

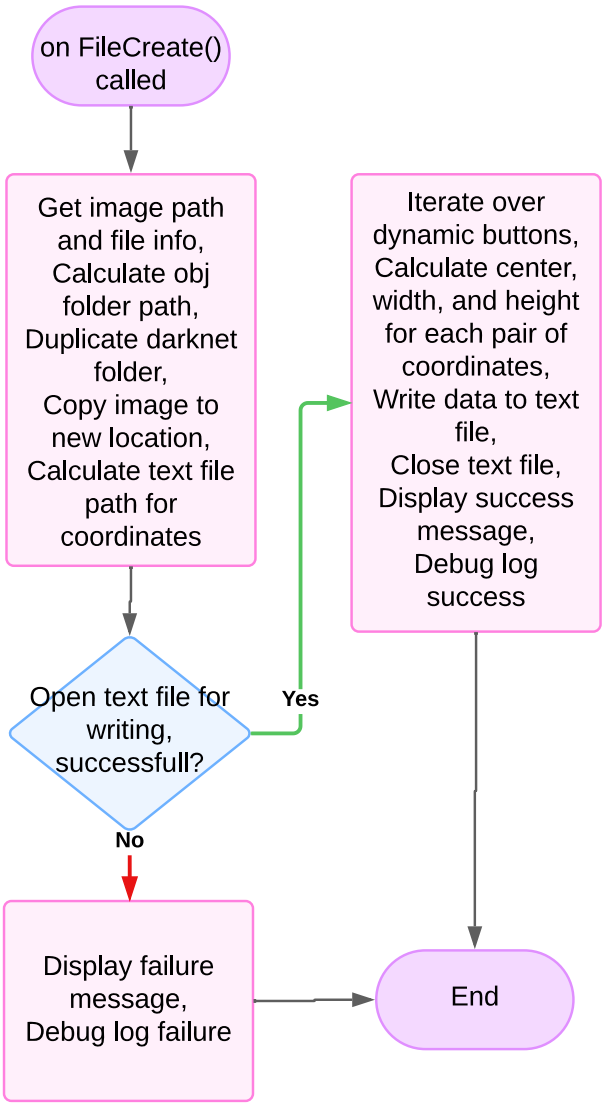

Figure 40 File creation fluxogram

3.5.5.9 Redo Selections Button

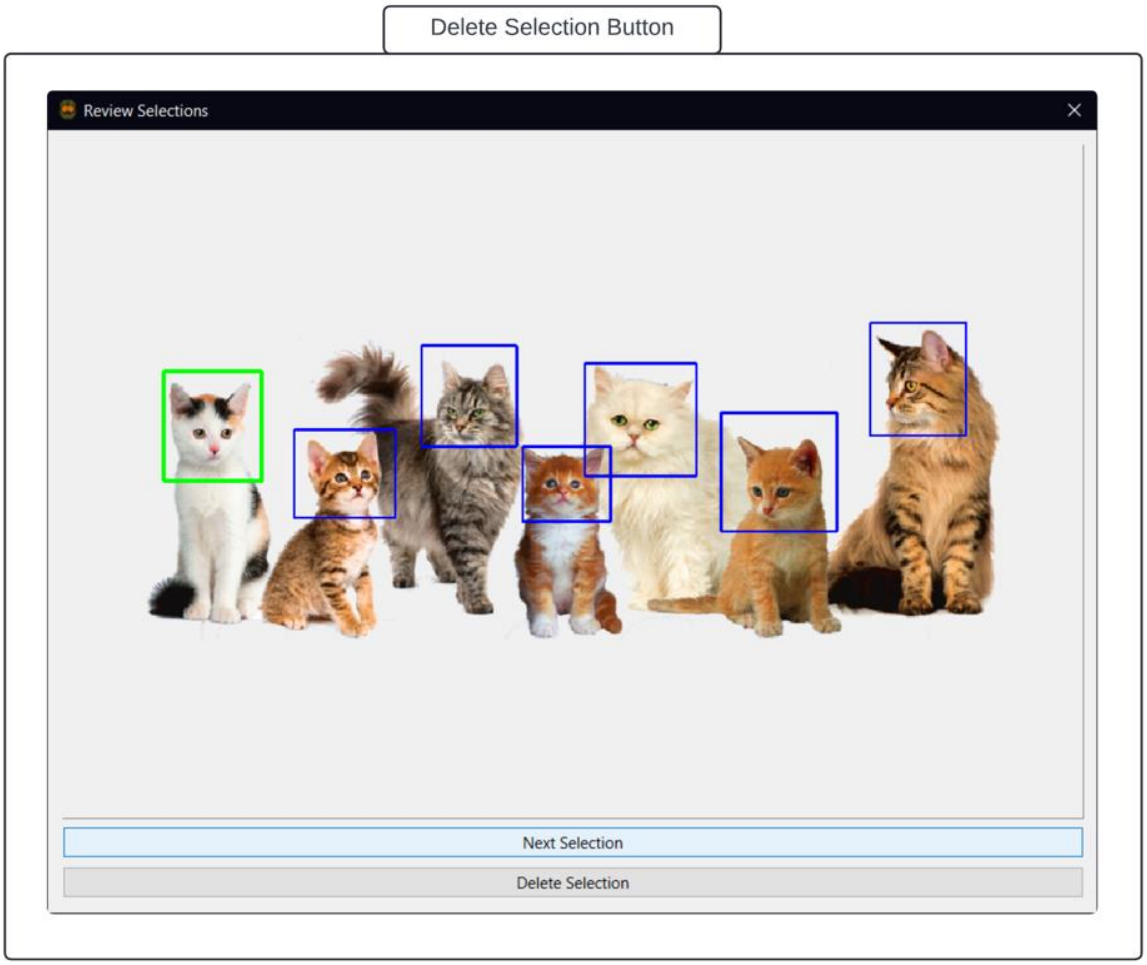

Figure 41 Delete Selections button interface

After performing numerous selections on an image or AI registering predictions in the graphic view, there is a possibility of errors, such as a poorly made selection. To address this, a button was created that opens a window with a `QGraphicsView` displaying the current selections, accompanied by two buttons: one to cycle through and highlight each selection, and another to delete the erroneous one (Figure 41).

The `on_redo_clicked` function initializes this window, configuring it to display the image and the associated selections. It uses a `QGraphicsScene` to manage the graphical content and a `QGraphicsView` to display it. The function loads the image and its coordinates into this scene, allowing the user to visually inspect each selection.

The function keeps track of the currently highlighted selection and provides the ability to cycle through them. As the user clicks to cycle through selections, the highlighted selection is visually

distinguished using a thicker, contrasting green pen. When a selection is found to be incorrect, the user can delete it by clicking the corresponding button. This action removes the selected rectangle from both the scene and the underlying data structure (`dinamicButtons`), as well as from the associated coordinates file, ensuring the annotation remains accurate.

Once the deletion is confirmed, the function updates the selection count, adjusts the highlighted selection if necessary, and reflects these changes in the user interface. After the review dialog is closed, the main view is refreshed to display the updated set of selections. This function provides a robust mechanism for reviewing and correcting selection errors, thereby maintaining the integrity of the annotation process (Figure 42**).**

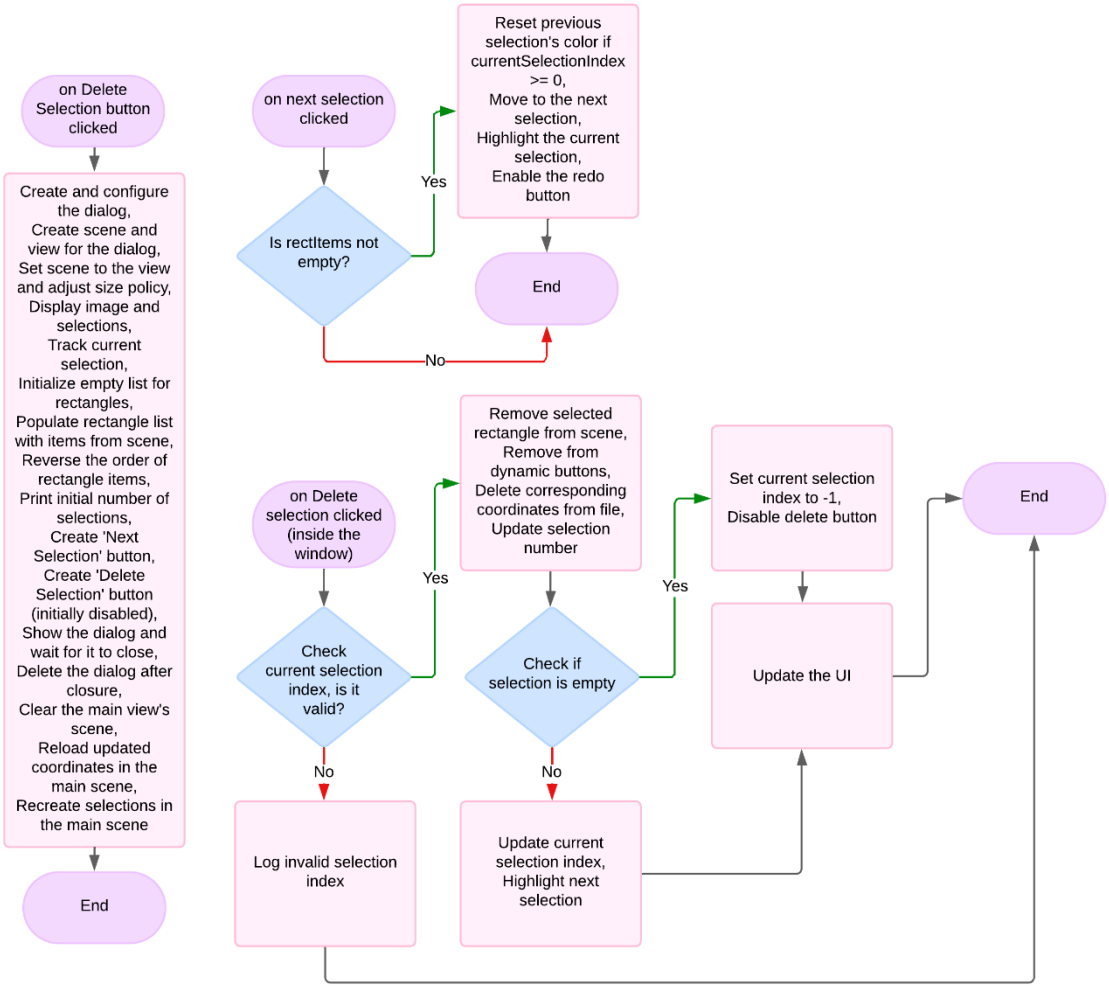

Figure 42 Delete Selections button fluxogram

3.5.6 Darknet Control Functions Logic

Darknet Control is arguably the most critical and complex component of the project. This section will introduce its key aspects and functionality.

3.5.6.1 Object Train Button

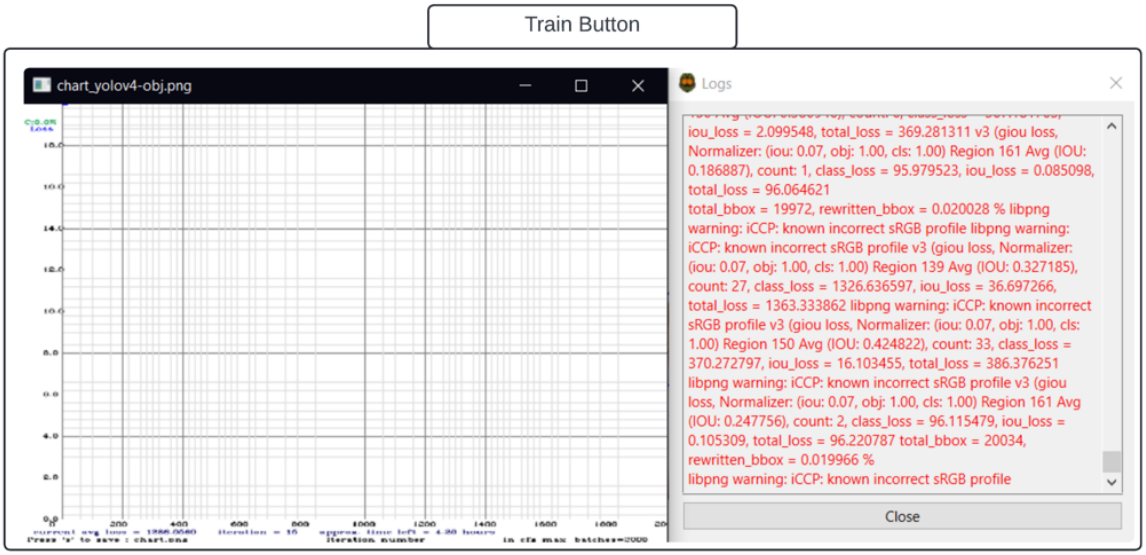

Figure 43 Train button interface

After making all the necessary selections and preparing to train the model to create a custom AI, the next step is to use the train button (Figure 43**)**. This function orchestrates the training process by executing three key functions: `yolov4Config`, which adjusts the YOLOv4 configuration file to fit the project's specific requirements; `runPythonScript`, which runs a Python script that populates the

`data.name` and `data.obj` files with the appropriate information; and `runDarknetexe`, which initiates the actual training process using the Darknet framework (Figure 44**)**.

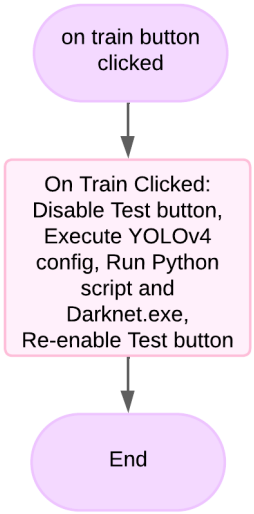

Figure 44 Train button fluxogram

3.5.6.2 Yolov4 Cfg Configuration

The `yolov4Config` function is designed to customize the YOLOv4 configuration file based on the number of object classes defined in the dataset. It begins by locating the `obj.data` file, from which it reads the number of classes. This number is critical as it influences various parameters in the YOLOv4 configuration.

The function then modifies the YOLOv4 configuration file, setting the subdivisions parameter to 64, which manages how the training data is divided and processed in smaller chunks, balancing memory usage and computational efficiency. It calculates the maximum number of training iterations (`max_batches`) using the formula `2000 * num_classes`, ensuring that the training duration is proportional to the complexity of the task, which is dictated by the number of object classes.

Next, the function updates the learning rate steps, recalculating them as 80% and 90% of `max_batches`, respectively. These steps control when the learning rate decreases, helping to refine the model as training progresses. The number of filters in the convolutional layer immediately preceding each YOLO layer is adjusted to `(num_classes + 5) * 3`, which accounts for the output

channels required for each class, along with bounding box coordinates, confidence scores, and the objectness score.

Finally, the function updates the number of classes within each YOLO layer to match the value from the `obj.data` file, ensuring that the network is configured to detect the correct number of object categories. After all these adjustments, the updated configuration is written back to the file, readying the YOLOv4 model for training on a customized dataset. This process is essential for adapting the generic YOLOv4 architecture to specific tasks, allowing it to effectively recognize and classify the defined objects in new images (Figure 45**).**

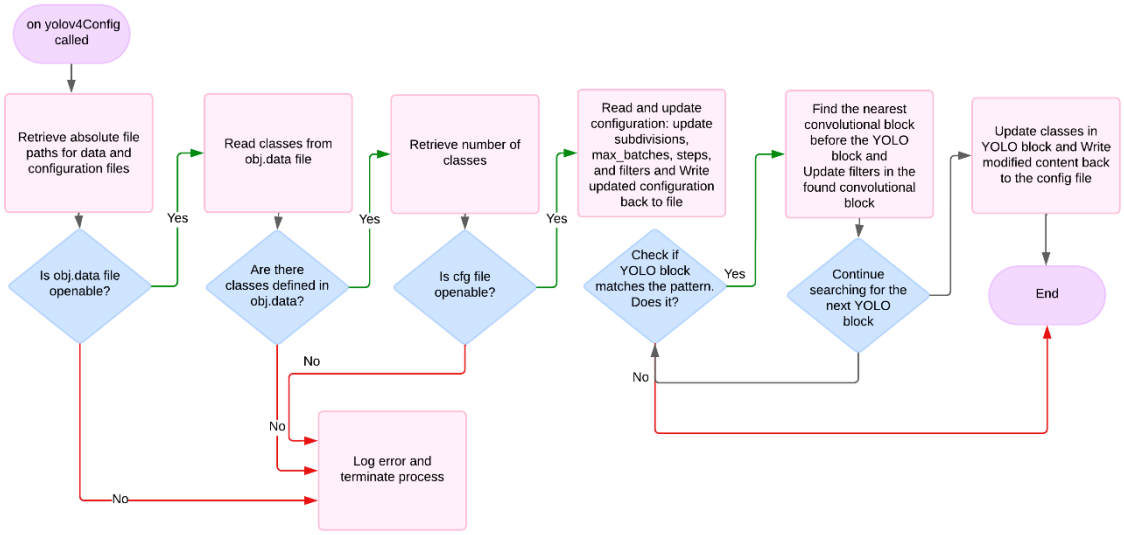

Figure 45 Yolov4 cfg configuration fluxogram

3.5.6.3 Populate Train and Test files

The Darknet framework requires two crucial files, `train.txt` and `test.txt`, located in the `data` folder. These files play distinct roles in the process of training and evaluating the YOLOv4 model. The `train.txt` file contains the paths to the images that the model will use for training, while the `test.txt` file lists the images for validation and testing purposes. The contents of these files are essential for the proper functioning of the training pipeline, as they instruct the model on which images to use in each phase.

To automate the creation of these files, a Python script named process was provided by the company but it was modified in order to populate only the files where selections were created. This script is specifically designed to work with images stored in the `data/obj` folder, where all training images

must be placed before execution. The script assumes that the images are in `.png` format, and it scans the entire `data/obj` directory to generate a list of these image paths.

The script is straightforward in its operation: it reads the images in the `data/obj` folder, and then writes their paths to either `train.txt` or `test.txt`, depending on a predefined percentage split. By default, the script allocates 10% of the images to the test set, meaning that for every 10 images, one is assigned to the `test.txt` file and the remaining nine to `train.txt`. This is controlled by a simple counter that resets after reaching the specified test index.

The script begins by opening or creating the `train.txt` and `test.txt` files in write mode. It then iterates through each `.png` file in the `data/obj` directory. For each image, it determines whether the image should be added to the training or test set based on the counter value. If the counter matches the calculated index for the test set, the image path is written to `test.txt`; otherwise, it goes to `train.txt`. After writing the path, the counter is updated, ensuring the correct distribution of images between the two sets.

This division is crucial for the YOLOv4 model to learn effectively, as the training set teaches the model to recognize patterns and objects, while the test set provides an independent dataset to evaluate the model's performance. The `.png` format is explicitly handled in the script, and if other formats are used, the script would need modification to accommodate those.

Finally, after the script completes its operation, it confirms the successful population of the `train.txt` and `test.txt` files by printing a message. This step ensures that the files are correctly generated, allowing the training process to proceed smoothly with the appropriate datasets (Figure 46).

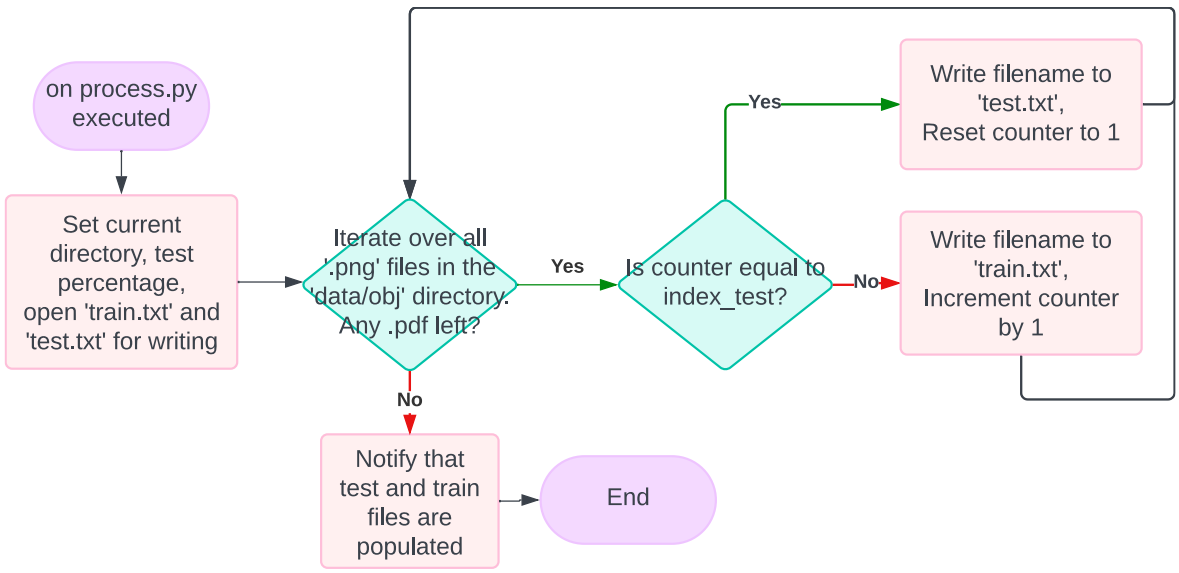

Figure 46 process.py fluxogram

The `runPythonScript` function automates the execution of a Python script within the Darknet folder. It uses `QProcess` to run the script in the specified directory, capturing real-time output and errors for logging. After starting the process, the function waits for it to complete, ensuring the script runs successfully or logs any issues encountered during execution (Figure 47**).**

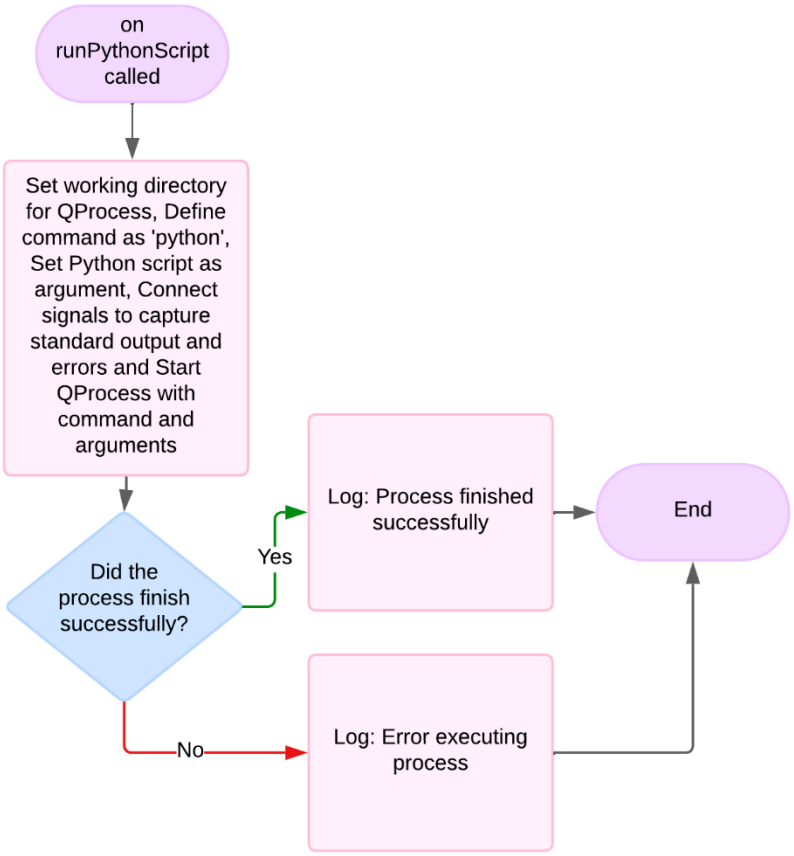

Figure 47 Run python script fluxogram

3.5.6.4 Darknet Image Training

After modifying the YOLOv4 configuration file and populating the train and test files, the training process can be initiated using the `runDarknetexe` function. This function automates the execution of the training process for YOLOv4 by running the `darknet.exe` application within a specified directory.

The function first verifies the existence of `darknet.exe` and checks the validity of the working directory. It then sets up a `QDialog` window with a `QTextEdit` to display the output logs, providing a visual representation of the training process, which is crucial for monitoring progress and detecting potential errors.

A `QProcess` is created to run the Darknet training command with appropriate arguments, including paths to the data file, the configuration file, and the pre-trained weights. The process output, including standard output and errors, is captured in real-time and displayed in the dialog window.

The dialog includes a close button that allows the user to terminate the training process if necessary. If the process is still running when the dialog is closed, it is terminated to ensure proper cleanup.

This design not only automates the training but also provides a user-friendly interface to monitor and control the process effectively (Figure 48**)**.

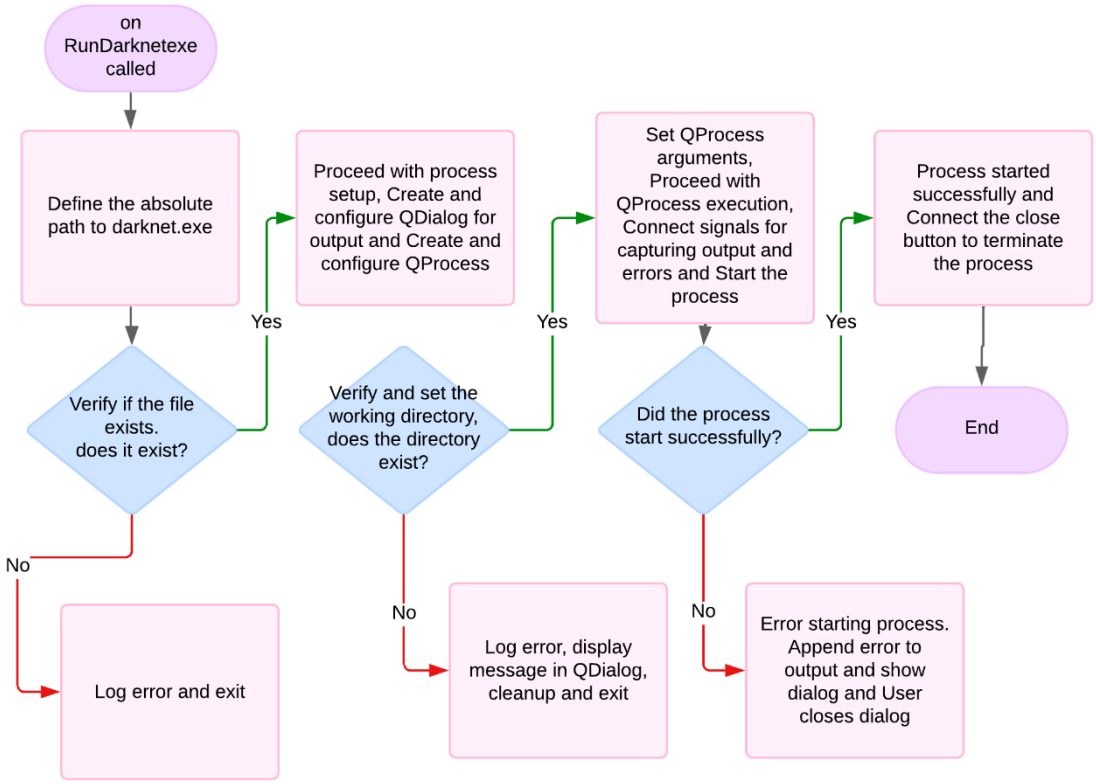

Figure 48 Rundarknetexe fluxogram

3.5.6.5 Weights ComboBox

After the training process is completed, the comboBox next to the buttons becomes available, populated with the trained weights. These weights are generated at every 1000 iterations during the training.

The `yolov4ComboBox` function handles the dynamic update of this comboBox. Initially, it clears any existing items to prevent duplication. The function then constructs the absolute path to the `backup` folder where the weights are stored. If the path contains an incorrect directory structure, it is corrected.

If the darknet directory is not set, the comboBox is populated with a placeholder item, "No weights," indicating that no weights are available. If the directory exists but is empty, it adds "no target" to the

comboBox, signifying that no weight files are found. If the directory contains files, each weight file is listed in the comboBox.

Once the comboBox is populated, the function sets the `weights` variable to the path of the first weight file, unless "no target" is selected, in which case `weights` is set to an empty string.

The function also connects the comboBox's selection change signal to update the `weights` variable whenever a different weight file is selected by the user. This ensures that the correct weight file is always available for any subsequent operations, and the layout is updated to reflect the changes in the UI (Figure 49**)**.

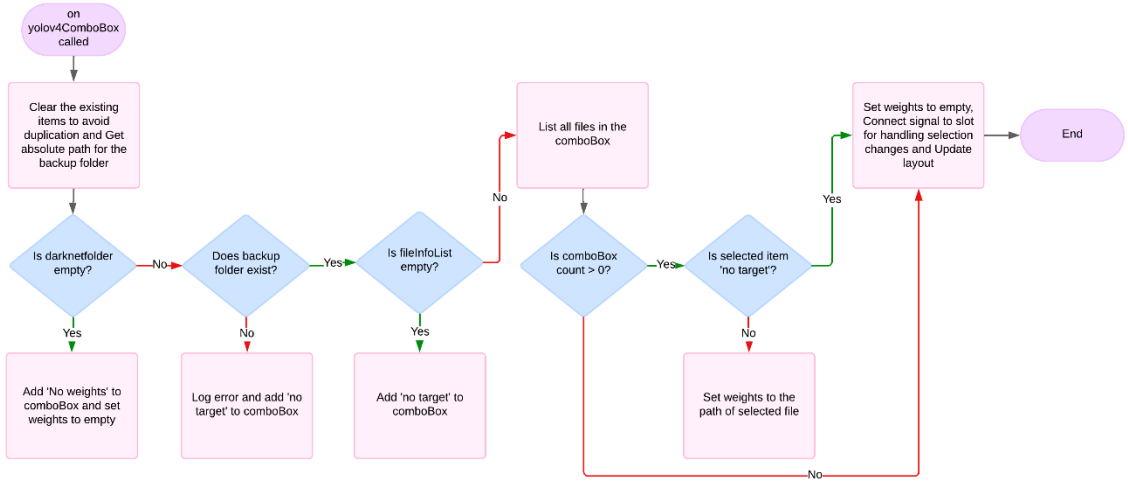

Figure 49 Yolov4ComboBox fluxogram

3.5.6.6 AI Test Button

Once the AI has been trained and the images are prepared for analysis, the process of allowing the AI to make predictions can begin. This process is initiated when the user clicks the "Test" button, triggering the `on_test_clicked` function, which activates the AI's detection capabilities.

The function starts by toggling the `darknetDetect` boolean variable, switching between enabling and disabling the AI's detection mode. It then refreshes the available weights by invoking the `yolov4ComboBox` function, ensuring that the most recent weights are available for selection.

Next, the function updates the UI label to display "Wait," indicating that the system is processing. If `darknetDetect` is enabled, the AI detection process is initialized by calling `initializeDarknet`. A timer is set for 7 seconds to simulate processing time, after which the label text updates to reflect whether AI detection is enabled or disabled, depending on the current state of `darknetDetect`. If `darknetDetect` is not enabled, the function simply updates the label to show that AI detection is disabled.

After this setup, the user can press the "D" key to switch images. This action triggers the AI to analyze the new image using the selected weights, testing the image and displaying the results (Figure 50).

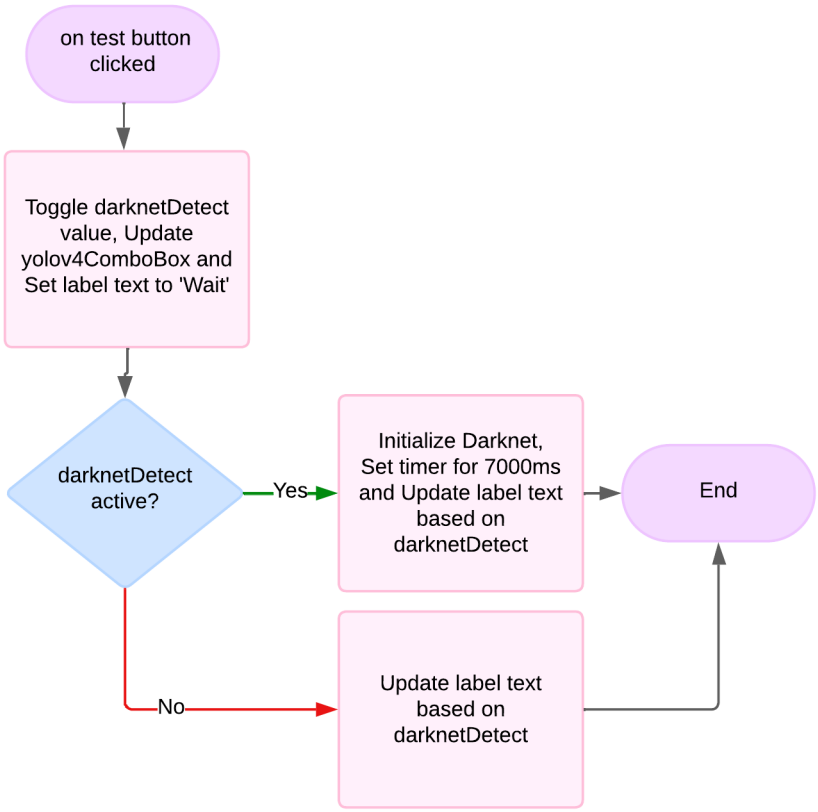

Figure 50 Test button fluxogram

3.5.6.7 Process Darknet Output

The `processDarknetOutput` function was designed to handle the output generated by the AI after it processes images using the Darknet framework. This function is responsible for capturing the predictions made by the AI, filtering out irrelevant data, and preparing the results for further use.

When the Darknet process completes, the function retrieves the standard output generated by the AI, which contains the raw predictions. This output is then written to a results file located at `darknetfolder/data/results.txt`. The file is opened in write-only mode, which ensures that any

previous content is cleared before writing the new data, effectively resetting the results file for each run.

Once the predictions are saved, the function executes a Python script, `resultsfilter.py`, by calling the generalized `runPythonScript` function. The purpose of `resultsfilter.py` is to process the raw AI predictions, filtering out any noise or irrelevant information, and refining the results to make them more accurate and actionable. After filtering, the refined results are further processed by calling the `addCoordinatesToFile` and `loadCoords` functions, which integrate the filtered predictions into the application's main scene, making them ready for visual display or further analysis (Figure 51).

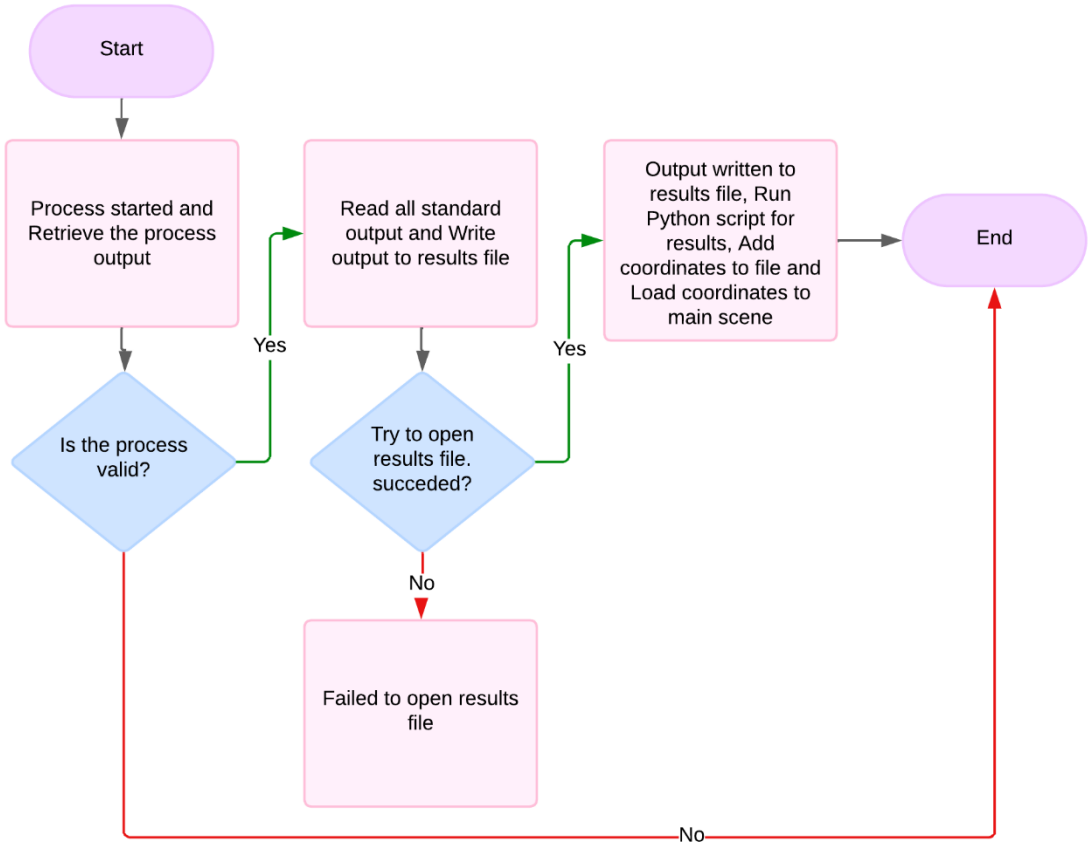

Figure 51 processDarknetOutput fluxogram

3.5.6.8 Darknet Detection Initialization

The function `initializeDarknet()` is essential for initiating the object detection process using the Darknet framework. This function sets up and executes the necessary command to run Darknet with the appropriate configurations and weights, enabling the AI to make predictions on the provided images.

The function builds a command to execute `darknet.exe` in "test" mode, which is specifically used for running inference on images, meaning that it tests the trained model on new, unseen data. The command includes several key arguments:

1. detector test: Specifies that the model should run in detection mode, using a trained YOLOv4 model.

2. data/obj.data: Points to the data configuration file, which contains metadata about the dataset, such as the number of classes and the paths to the training and validation files.

3. cfg/yolov4-obj.cfg: Specifies the configuration file for the YOLOv4 model, detailing the architecture of the neural network and various hyperparameters.

4. <weights trained>: Refers to the file containing the trained weights that the model will use for inference.

5. -dont_show: Prevents the Darknet tool from displaying the results in a window, which is useful for running processes in a command-line environment without requiring user interaction.

6. -ext_output: Ensures that the output includes detailed information about the predictions, such as the detected objects' coordinates and confidence scores, which is crucial for further processing by the application.

Once the command is prepared, the `initializeDarknet()` function launches the Darknet process using a `QProcess` object. This object is responsible for handling the execution of external programs within a Qt application. The function also connects several signals from the `QProcess` object to handle standard output, errors, and process completion events. Specifically, the output from the Darknet process is handled by the `processDarknetOutput()` function, which processes and saves the predictions.

The `initializeDarknet()` function plays a critical role in triggering the detection process. After it starts, the Darknet tool enters a state where it waits for an image path to be provided. When an image path is supplied, the tool uses the trained model to make predictions on the image, generating output that includes the detected objects, their locations, and associated confidence scores. This output is then captured and processed to integrate the predictions into the application's workflow (Figure 52**)**.

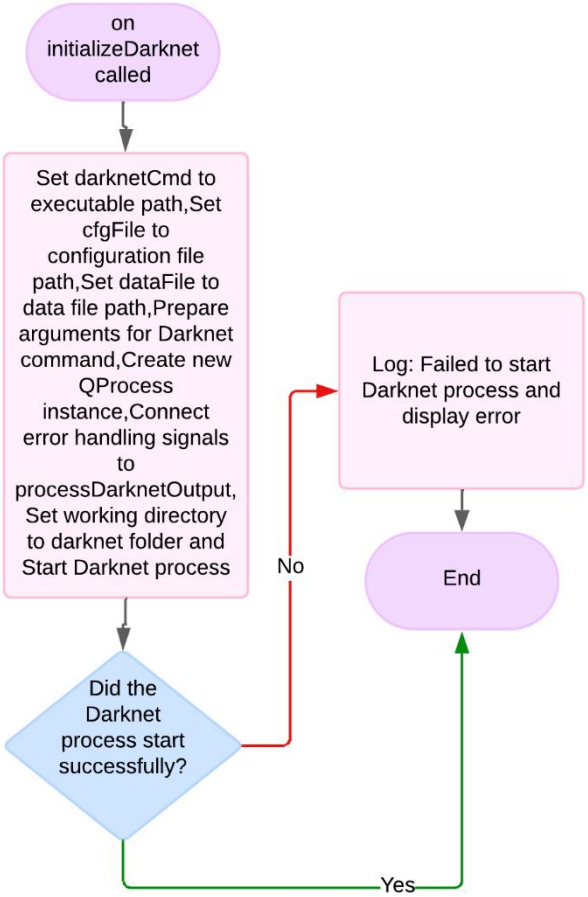

Figure 52 InitializeDarknet fluxogram

4 Results

This chapter presents the interface results of a single-case simulation using images of some of the world's cutest animals: cats. Machines deserve to enjoy them too! A small sample of images was selected as an example. Using a small dataset is advantageous for demonstration, as it accelerates the training process. Naturally, a comprehensive dataset would typically include thousands of images, capturing various angles, positions, and lighting conditions to ensure the training model is robust against diverse scenarios. All outputs and actions resulting from this simulation are documented in detail.

4.1 Images dataset

A total of nineteen images were selected, of which nineteen were manually tagged for a total of sixty-two selections (Figure 53). With such a small dataset, maximizing the number of images available for training is crucial. Consequently, five images were allocated for training (Figure 54).

Figure 53 Simulation Images chosen to be manually tagged

Figure 54 Simulation Images chosen to be tested

4.2 Manually tagged images

All manually tagged images were curated to focus exclusively on the head of the cat. This preference was chosen deliberately, as the head represents a significantly smaller portion of the image compared to the full body (Figure 55). Utilizing the full body could introduce visual ambiguity, particularly in images containing numerous objects.

Figure 55 Simulation Manually tagged images with actual selections

4.3 Darknet Training

On the training side, two experiments were conducted: one utilizing the preexisting dataset “yolov4.conv.137” and one without it. The objective was to assess whether the use of the preexisting dataset provides any notable advantages. Given that “yolov4.conv.137” contains millions of tagged images and is intended to assist the training dataset, there is a possibility that the training process could become "distracted" by attempting to match irrelevant features, such as bicycle wheels. This distraction could result in an increased loss rate.

4.3.1 Darknet Training with Pre-existing dataset

After the images were prepared, the training process commenced with the use of the pre-trained model “yolov4.conv.137”, resulting in the generation of a loss chart, as shown in Figure 56.

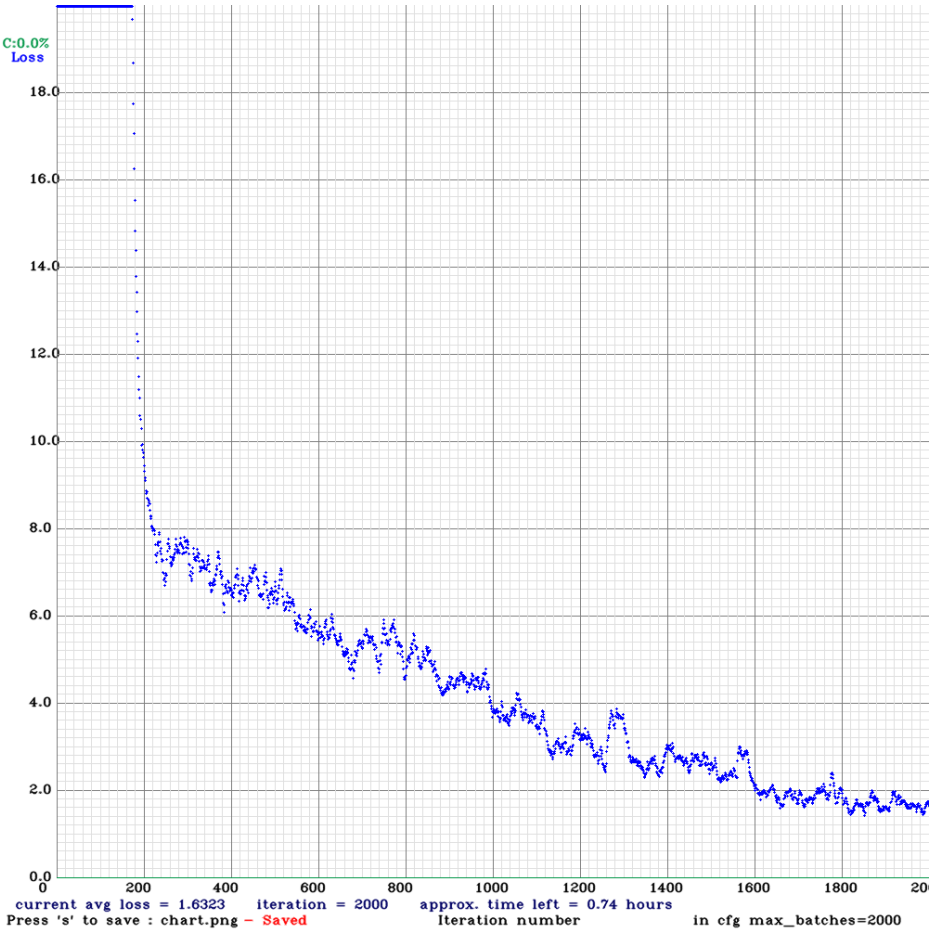

Figure 56 Simulation with pre-existing dataset loss chart

4.3.2 Darknet Testing with Pre-existing dataset

The ML tagging process was then initiated, producing the outputs shown in Figure 57. Upon analyzing the results, it is evident that some images were tagged incorrectly, which is understandable given that the training dataset consisted of only 15 images. Additionally, the module was trained using a diverse dataset, including mixed breeds of cats with various colors and patterns, different ages (ranging from adorable kittens to charming senior cats), and varied lighting conditions and positions. Despite these challenges, the outputs were remarkably good. Furthermore, the presence of well-executed selections highlights that the training process was overall successful.

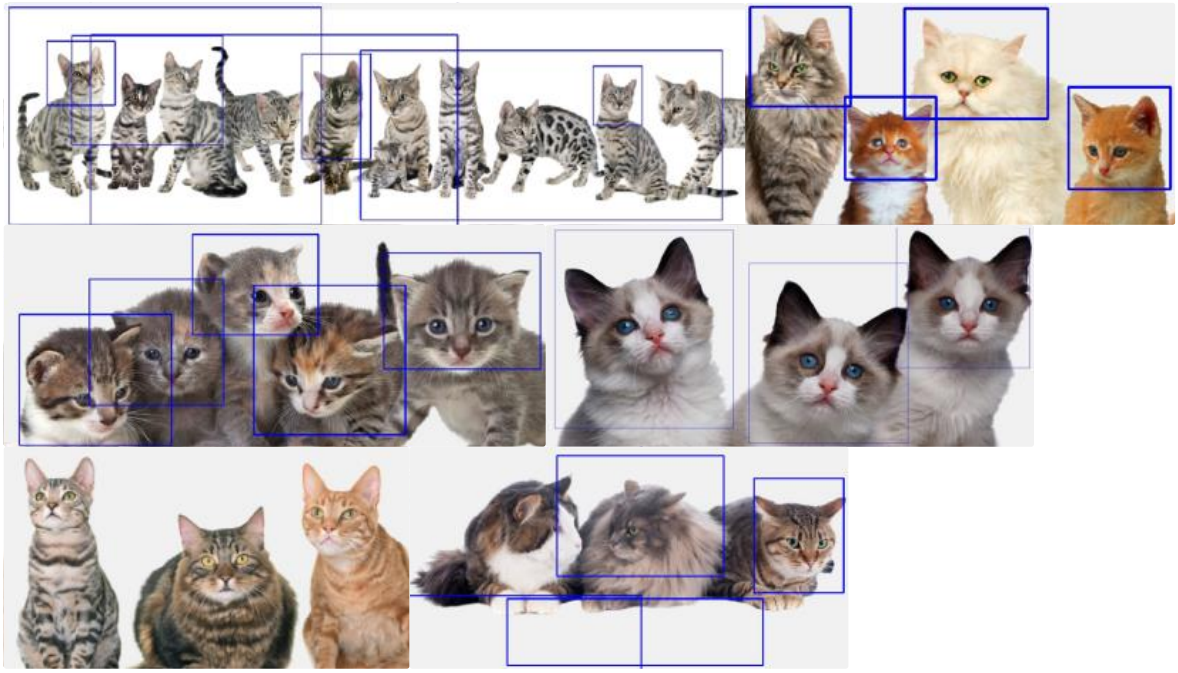

Figure 57 Simulation ML tagged images with pre-existing dataset

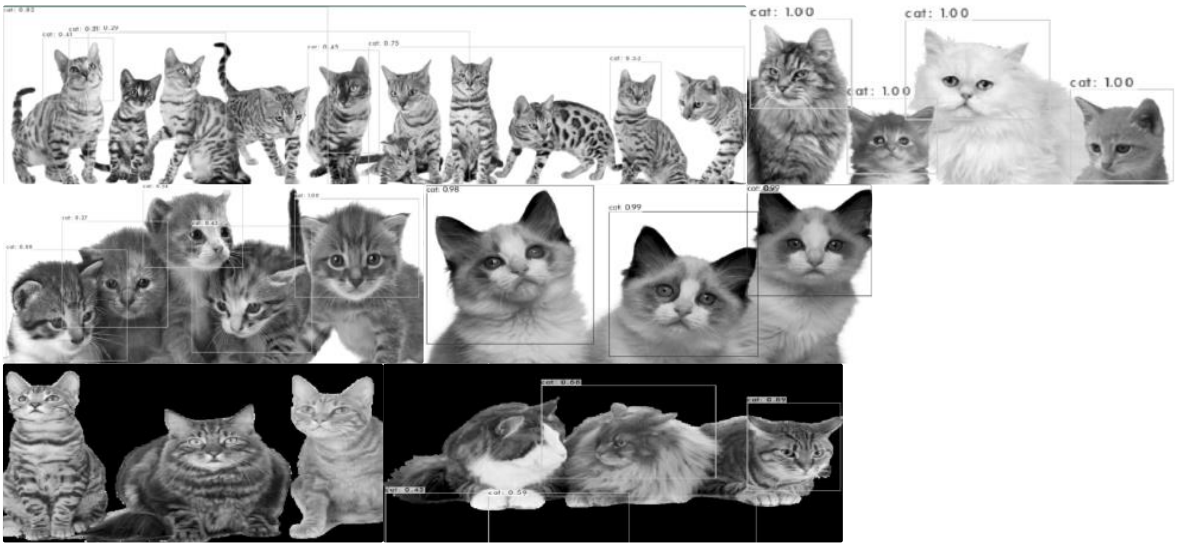

Figure 57 ML Predictions certainty with pre-existing dataset

4.3.3 Darknet Training without Pre-existing dataset

After the images were prepared, the training process commenced without the use of the pre-trained model “yolov4.conv.137”, resulting in the generation of a loss chart, as shown in Figure 58.

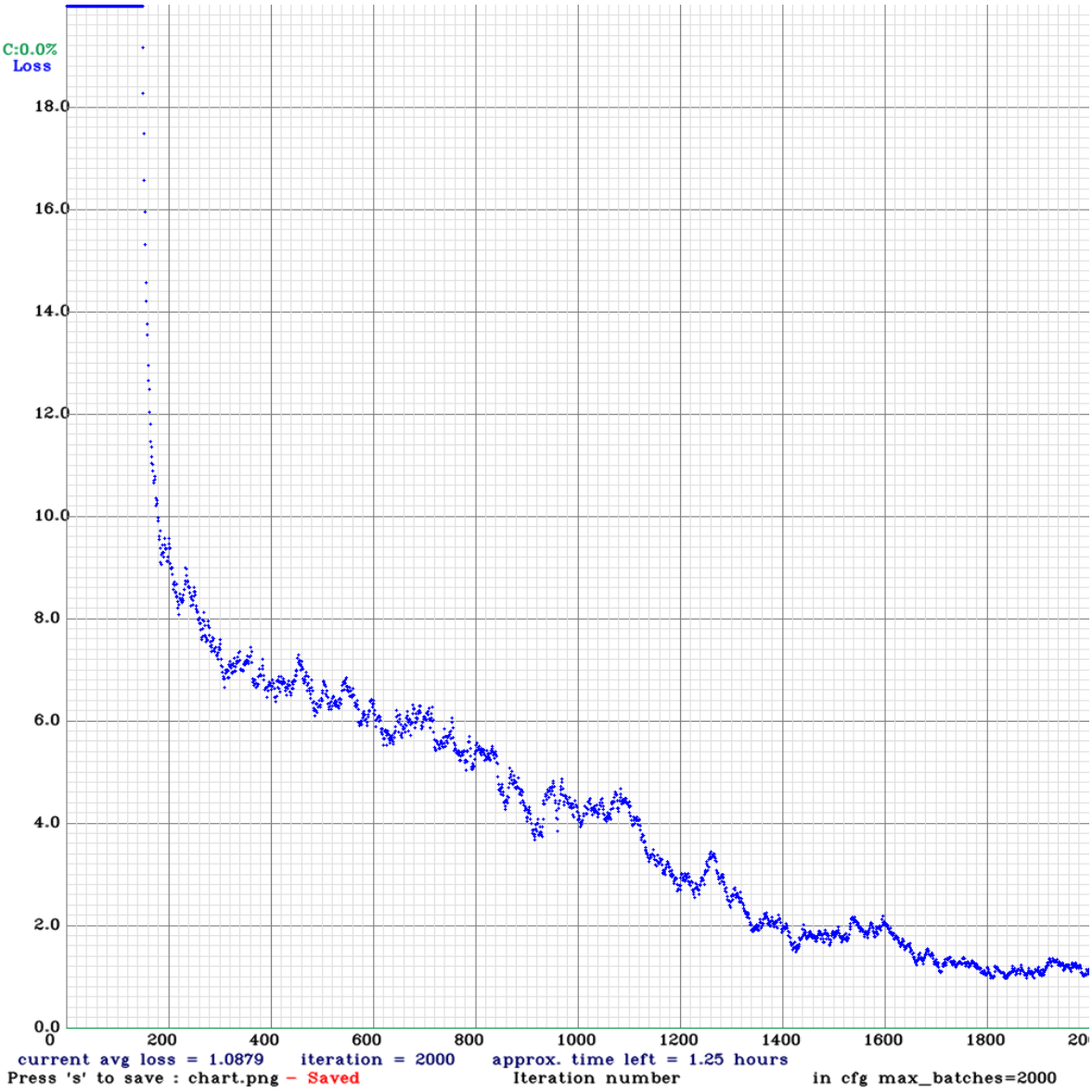

Figure 58 Simulation without pre-existing dataset loss chart

4.3.4 Darknet Testing without Pre-existing dataset

The ML tagging process was then initiated, producing the outputs shown in Figure 59 and Figure 60.

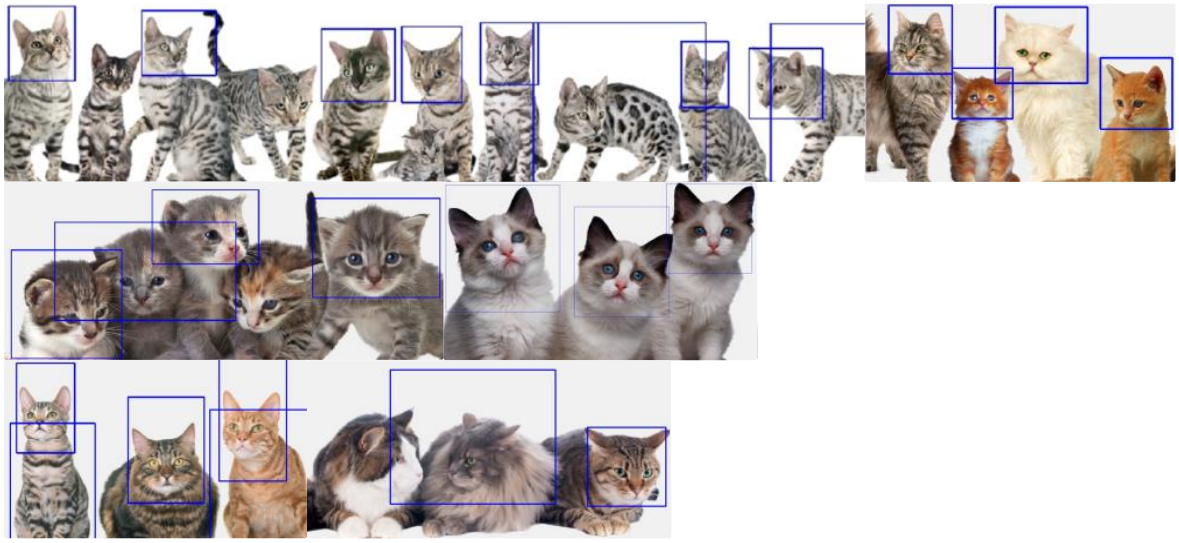

Figure 59 Simulation ML tagged images without pre-existing dataset

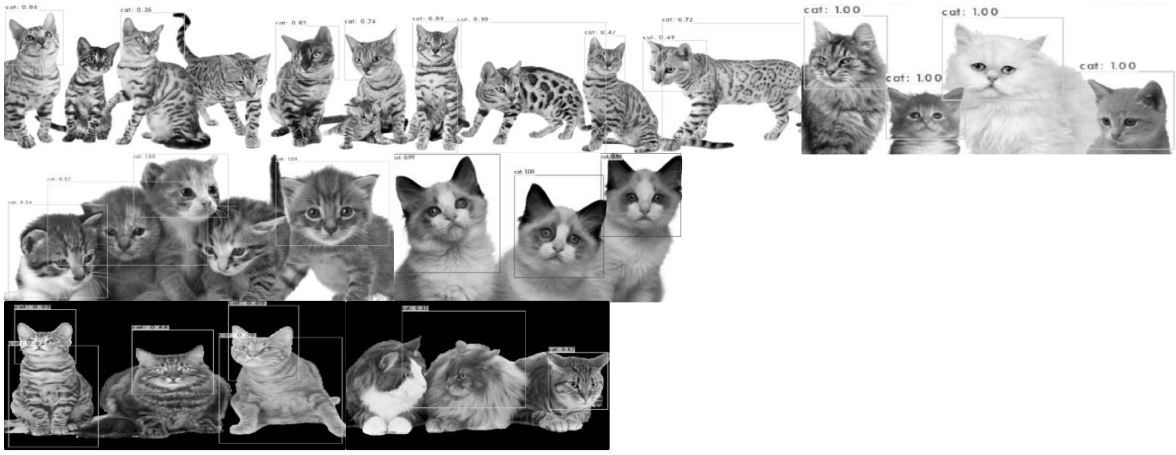

Figure 60 Simulation ML tagged images without pre-existing dataset

4.3.5 Yolov4.conv.137 usage conclusions

The most notable difference observed was the training time. The use of the pre-trained weights "Yolov4.conv.137" resulted in a training duration of three hours, whereas training without these weights took twenty-two hours.

The loss charts showed a slight variation at the end of the training process. The model trained without the pre-trained weights exhibited less ripple (Figure 58, Figure 56). This could be attributed to the extensive variety of tagged images in "yolov4.conv.137," which may introduce some confusion during training, such as misclassifications when comparing cats to other objects, potentially leading to suboptimal training.

In terms of predictions, the model trained without the pre-trained weights appeared to be more precise but had lower detection rates (Figure 59, Figure 57).

In conclusion, the results did not demonstrate any significant advantage or disadvantage to using the pre-trained weights for this dataset. These changes were not evident in the dataset used for this study; however, in larger datasets, the differences could potentially be more pronounced, though this remains unverified in the context of the current analysis.

5 Conclusion

This project was centered on software development, with a strong emphasis on back-end implementation. The back-end was developed in C++ using the Qt libraries and extensions, while Python scripts were employed for various tasks. The front-end was created using Qt Design Studio, integrated with Visual Studio. However, the primary focus remained on the back-end functionality, which formed the project's foundation.

One of the major challenges encountered during development was configuring Darknet and OpenCV, which proved to be more complex than anticipated. Despite these hurdles, the overall development process was engaging and manageable.

The project's specific objectives were defined by the project lead, with progress reviewed by both my ISEP tutor and the GISLOTICA tutor. The development process was particularly challenging as I had to self-learn all the programming languages involved while working on the project.

5.1 Future Work Projects

Future work will focus on experimenting with the pattern detection system in the cordage industry and potentially optimizing the system's performance by integrating existing advanced YOLO models. It would then be useful to use the validation loss during

the training phase to detect overfitting, select an appropriate learning rate, and apply an

early stopping criterion whenever the validation loss stabilizes, even if the training loss

continues to decrease.

Documental References

[1] Qt Company-Qt: Leading Cross-Platform Development Framework. Available at: https://www.qt.io. Accessed June 6, 2024.

[2] The Qt Company. "Qt Licensing." Available at: https://www.qt.io/licensing. Accessed August 27, 2024.

[3] The Qt Company. "Qt for Device Creation." Available at: https://www.qt.io/product/device-creation. Accessed August 27, 2024.

[4] The Qt Company. "Qt for Application Development." Available at: https://www.qt.io/product/application-development. Accessed August 27, 2024.

[5] The Qt Company. "Qt Documentation." Available at: https://doc.qt.io/. Accessed August 27, 2024.

[6] The Qt Company. "Qt Ecosystem." Available at: https://www.qt.io/qt-ecosystem. Accessed August 27, 2024.

[7] Qt Documentation. "Getting Started with Qt." Available at: https://doc.qt.io/qt-6/gettingstarted.html. Accessed August 27, 2024.

[8] The Qt Company. "Qt Licensing." Available at: https://www.qt.io/licensing. Accessed August 27, 2024.

[9] Qt Blog. "Why Choose Qt?" Available at: https://www.qt.io/blog. Accessed August 27, 2024.

[10] React. "React – A JavaScript library for building user interfaces." Available at: https://reactjs.org. Accessed August 27, 2024.

[11] Telerik. "Telerik UI Components for .NET and JavaScript." Available at: https://www.telerik.com. Accessed August 27, 2024.

[12] Kendo UI. "Kendo UI: Modern UI Components for JavaScript." Available at: https://www.telerik.com/kendo-ui. Accessed August 27, 2024.

[13] Syncfusion. "Essential Studio: Comprehensive UI Controls for Web, Desktop, and Mobile." Available at: https://www.syncfusion.com/products/essential-studio. Accessed August 27, 2024.

[14] Microsoft. "Visual Studio 2022." Available at: https://visualstudio.microsoft.com/vs/2022/. Accessed August 27, 2024.

[15] Stroustrup, Bjarne. The C++ Programming Language. 4th ed., Addison-Wesley, 2013.

[16] Python Software Foundation. Python Official Documentation. Available at: https://docs.python.org/3/. Accessed August 27, 2024.

[17] Django Software Foundation. Django Official Documentation. Available at: https://www.djangoproject.com/. Accessed August 27, 2024.

[18] PyPy Project. PyPy: The Python Interpreter. Available at: https://www.pypy.org/. Accessed August 27, 2024.

[19] Van Rossum, Guido. Python 3 Documentation. Available at: https://docs.python.org/3/whatsnew/3.0.html. Accessed August 27, 2024.

[20] Darknet. "Darknet: Open Source Neural Networks in C." Available at: https://pjreddie.com/darknet/. Accessed August 27, 2024.

[21] Redmon, Joseph. *YOLO: Real-Time Object Detection.* Available at: https://pjreddie.com/darknet/yolo/. Accessed August 27, 2024.

[22] TensorFlow. "TensorFlow vs. Darknet: Comparing Deep Learning Frameworks." Available at: https://www.tensorflow.org/. Accessed August 27, 2024.

[23] GitHub. "Darknet GitHub Repository." Available at: https://github.com/pjreddie/darknet. Accessed August 27, 2024.

[24] Joseph Redmon and Santosh Divvala. "YOLOv4: Optimal Speed and Accuracy of Object Detection." Available at: https://arxiv.org/abs/2004.10934. Accessed August 16, 2024.

[25] Joseph Redmon and Santosh Divvala. "Spatial Pyramid Pooling for Object Detection." Available at: https://arxiv.org/abs/1504.00940. Accessed August 16, 2024.

[26] Joseph Redmon and Santosh Divvala. "Path Aggregation Network for Object Detection." Available at: https://arxiv.org/abs/1803.01534. Accessed August 16, 2024.

[27] Joseph Redmon and Santosh Divvala. "Mish Activation Function for Neural Networks." Available at: https://arxiv.org/abs/1908.08681. Accessed August 16, 2024.

[28] Joseph Redmon and Santosh Divvala. "CIoU Loss for Object Detection." Available at: https://arxiv.org/abs/1907.07184. Accessed August 16, 2024.

[29] Joseph Redmon and Santosh Divvala. "AdaBelief Optimizer for Fast and Stable Training." Available at: https://arxiv.org/abs/2010.07468. Accessed August 16, 2024.

[30] CUDA Official Documentation: NVIDIA. Available at: https://developer.nvidia.com/cuda-zone. Accessed August 16, 2024.

[31] Darknet Framework: Joseph Redmon and Santosh Divvala. Available at: https://github.com/pjreddie/darknet. Accessed August 16, 2024.

[32] OpenCV Official Documentation: OpenCV. Available at: https://opencv.org/. Accessed August 16, 2024.

[33] NVIDIA. cuDNN: GPU Accelerated Library for Deep Neural Networks. Available at: https://developer.nvidia.com/cudnn. Accessed August 27, 2024.

[34] Asan, O., & Yoon, H. (2020). "Deep Learning in Skin Disease Image Recognition: A Review." *Sensors*, 20(19), 5487. Available at: https://www.mdpi.com/1424-8220/20/19/5487. Accessed August 27, 2024.

[35] Zhen, J., Zhao, Y., & Zhang, C. (2021). "Deep Facial Diagnosis: Transfer Learning from Face Recognition to Facial Diagnosis." *IEEE Transactions on Biomedical Engineering*. Available at: https://ieeexplore.ieee.org/document/9355498. Accessed August 27, 2024.

[36] NVIDIA Developer Blog. "Why GPUs Are Better for Deep Learning." Available at: https://developer.nvidia.com/blog/why-gpus-are-better-for-deep-learning/. Accessed August 16, 2024.

Annex A. C++ Main Header File Code

```
#pragma once
```

```
#ifndef INTERFACE_H
#define INTERFACE_H

#include <QtWidgets/QMainWindow>
#include "ui_interface.h"
#include "./ui_interface.h"
#include "ui_interface.h"
#include "GV.h"
#include <QObject>
#include <QFile>
#include <QAbstractButton>
#include <QPushButton>
#include <QFileDialog>
#include <QProgressDialog>
#include <QDirIterator>
#include <QMenu>
#include <QDir>
#include <QLabel>
#include <QFileInfo>
#include <QGuiApplication>
#include <QGraphicsPixmapItem>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QPixmap>
#include <QImage>
#include <QImageReader>
#include <QInputDialog>
#include <QVBoxLayout>
#include <QLayout>
#include <QInputDialog>
#include <QTranslator>
#include <QMessageBox>
#include <QVBoxLayout>
#include <QDialog>
#include <QMouseEvent>
#include <QGraphicsSceneMouseEvent>
#include <QColorDialog>
#include <QPoint>
#include <QKeyEvent>
#include <QProcess>
#include <QRandomGenerator>
#include <QTextEdit>
#include <opencv2/opencv.hpp>
#include <QRegularExpression>
#include <QComboBox>
#include <QTimer>


class interface : public QMainWindow
{
   Q_OBJECT
```

```
protected:
    void keyPressEvent(QKeyEvent* event) override;

public:
    interface(QWidget* parent = nullptr);
    ~interface();

    QGraphicsView* graphicsView = nullptr;
    QGraphicsScene* scene = new QGraphicsScene();
    QGraphicsPixmapItem* pixmapItem = nullptr;

    QString folderName = nullptr;
    QString darknetfolder = nullptr;
    QString weights = nullptr;

    QPushButton* selectedButton = nullptr;
    QListWidgetItem* selectedItem = nullptr;

    int buttonCounter = -1;
    int index = 0;
    int currentIndex = 0;
    int listbIndexSelected = 0;
    int imageIndex = 0;
    int dButtonIndex = -1;
    int folderitemquantity = 0;
    int mouseClickCounter = 0;

    //all the variables related to the dinamic buttons
    struct dinamicB
    {
        QColor color;
        QVector<QPointF> firstc;
        QVector<QPointF> secondc;
        QPushButton* button = nullptr;
        QString dirButtons;
        int selectionNr = 0;
        QGraphicsScene* scene;
    };
    std::vector<dinamicB> dinamicButtons;

    std::vector<QString> imagelist;

    bool mousePressEventEnabled = true;
    bool darknetDetect = false;

    QProcess* inferprocess;

private slots:
    void images(std::vector<QString> imagelist);
```

```
    void on_info_clicked();
    void nextImage();
    void previousImage();
    void on_button_clicked();
    void on_newFile_clicked();
    void on_addB_clicked();
    void on_editB_clicked();
    void on_undoSelec_clicked();
    void on_train_clicked();
    void on_openFile_clicked();
    void on_redo_clicked();
    void yolov4ComboBox();
    void initializeDarknet();
    void addDynamicButton(const QString& buttonName);
    void addDynamicButtonHelper(const QString& buttonName, const QColor& color);
    void addCoordinatesToFile();
    void runPythonScript(const QString& scriptPath, QString script);
    void runDarknetexe(const QString& scriptPath);
    void on_test_clicked();
    void processDarknetOutput();
    bool alreadySelected(const QString& imageName);
    void yolov4Config();
    void deleteCoordinatesFromFile(int selectionIndex);
    QString duplicatedarknet(QString duplicate);
    void copyImagesToObjFolder(const QString& srcFolder, const QString& dstFolder);
    void objdatafilechange();
    void fileCreate();
    void loadCoords(QGraphicsScene* targetScene);
    void clearEverything();
    bool copyDirRecursively(const QString& srcPath, const QString& dstPath);
    void on_widgetlist_clicked();
    void WidgetListset(const QString& file);
    void mouseClicked(const QPointF& pos);
    void createSelection(QGraphicsScene* targetScene);
    void clearSelections();
    void updateMousePosition(const QPointF& pos);
    void customCursor();

private:
    Ui::interfaceClass ui;
};

#endif
```

Annex B. C++ Custom GraphicView Header File Code

```
#ifndef GV_H
#define GV_H

#include <QGraphicsView>
#include <QMouseEvent>

class GV : public QGraphicsView {
   Q_OBJECT

public:
   GV(QWidget* parent = nullptr) : QGraphicsView(parent) {
      setMouseTracking(true);
   }

signals:
   void mouseMoved(const QPointF& pos);
   void mousePressed(const QPointF& pos);

protected:
   void mouseMoveEvent(QMouseEvent* event) override {
      QPointF scenePos = mapToScene(event->pos());
      QPointF normalizedPos = normalizeCoordinates(scenePos);
      emit mouseMoved(normalizedPos);
      QGraphicsView::mouseMoveEvent(event);
   }
   void mousePressEvent(QMouseEvent* event) override {
      QPointF scenePos = mapToScene(event->pos());
      QPointF normalizedPos = normalizeCoordinates(scenePos);
      emit mousePressed(normalizedPos);
      QGraphicsView::mousePressEvent(event);
   }

private:
   QPointF normalizeCoordinates(const QPointF& pos) const {
      if (!scene()) {
         return QPointF();
      }
      QRectF sceneRect = scene()->sceneRect();
      double normalizedX = (pos.x() - sceneRect.left()) / sceneRect.width();
      double normalizedY = (pos.y() - sceneRect.top()) / sceneRect.height();
      return QPointF(normalizedX, normalizedY);
   }
};

#endif // GV_H
```

Annex C. C++ File Code

```
#ifndef INTERFACE_CPP
#define INTERFACE_CPP

#include "interface.h"
#include "./ui_interface.h"

interface::interface(QWidget* parent)
   : QMainWindow(parent)
{
   ui.setupUi(this);
   setWindowTitle("MasterChief");

   setFocusPolicy(Qt::StrongFocus);

   ui.info->setEnabled(false);
   ui.widgetlist->setEnabled(false);
   ui.addB->setEnabled(false);
   ui.editB->setEnabled(false);
   ui.undoSelec->setEnabled(false);
   ui.test->setEnabled(false);
   ui.redo->setEnabled(false);

   yolov4ComboBox();

   QObject::connect(ui.openFile, &QAction::triggered, this, &interface::on_openFile_clicked);
   QObject::connect(ui.newFile, &QAction::triggered, this, &interface::on_newFile_clicked);

   ui.graphicsView->setScene(scene);
}

interface::~interface()
{
   clearEverything();
   delete scene;
}
//---------------------- PROJECT MANAGMENT ---------------------------
void interface::on_newFile_clicked() {
   QDialog dialog(this);
   dialog.setWindowTitle("Folder paths");

   QGridLayout* dialogLayout = new QGridLayout(&dialog);

   // Label e botão de seleção do primeiro diretório
   QLabel* dirnameLabel = new QLabel("Image folder path:", &dialog);
   dialogLayout->addWidget(dirnameLabel, 0, 0);

   QPushButton* dirselecB1 = new QPushButton("Select", &dialog);
```

```
    dialogLayout->addWidget(dirselecB1, 0, 1);

    QLabel* dirname = new QLabel(&dialog);
    dialogLayout->addWidget(dirname, 1, 0, 1, 2);

    connect(dirselecB1, &QPushButton::clicked, [&]() {
        folderName = QFileDialog::getExistingDirectory(nullptr, "Select an image folder", "", QFileDialog::ShowDirsOnly);
        if (folderName.isEmpty()) {
            ui.label->setText("No folder selected.");
            return;
        }
        dirname->setText(folderName);
        });

    // Label e botão de seleção do segundo diretório
    QLabel* darknetLabel = new QLabel("Darknet copy location:", &dialog);
    dialogLayout->addWidget(darknetLabel, 2, 0);

    QPushButton* dirselecB2 = new QPushButton("Select", &dialog);
    dialogLayout->addWidget(dirselecB2, 2, 1);

    QLabel* darknetpath = new QLabel(&dialog);
    dialogLayout->addWidget(darknetpath, 3, 0, 1, 2);

    connect(dirselecB2, &QPushButton::clicked, [&]() {
        darknetfolder = QFileDialog::getExistingDirectory(nullptr, "Darknet copy location", "", QFileDialog::ShowDirsOnly);
        if (darknetfolder.isEmpty()) {
            ui.label->setText("No folder selected.");
            return;
        }
        darknetfolder = darknetfolder + "/darknet";
        darknetpath->setText(darknetfolder);
        });

    // Botões OK e Cancelar
    QDialogButtonBox* buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel, Qt::Horizontal, &dialog);
    dialogLayout->addWidget(buttonBox, 4, 0, 1, 2, Qt::AlignCenter);

    // Conecta os botões OK e Cancelar para aceitar ou rejeitar a janela de diálogo
    connect(buttonBox, &QDialogButtonBox::accepted, &dialog, &QDialog::accept);
    connect(buttonBox, &QDialogButtonBox::rejected, &dialog, &QDialog::reject);

    // Executa a janela de diálogo
    if (dialog.exec() == QDialog::Accepted && !darknetfolder.isEmpty() && !folderName.isEmpty()) {
        QDir currentDir(QDir::currentPath());
        if (!currentDir.cd("..") || !currentDir.cd("..") || !currentDir.cd("..") || !currentDir.cd("masterchief")) {
            ui.label->setText("Failed to locate the masterchief directory.");
            return;
        }
```

```
    QString sourceDir = currentDir.absoluteFilePath("darknet");
    QString destDir = darknetfolder;

    qDebug() << "Source Directory:" << sourceDir;
    qDebug() << "Destination Directory:" << destDir;

    if (!copyDirRecursively(sourceDir, destDir)) {
        ui.label->setText("Failed to copy darknet to the specified location.");
        qDebug() << "Failed to copy from" << sourceDir << "to" << destDir;
        return;
    }

    int i = 0;
    qint64 totalSize = 0;

    clearEverything();

    // Desbloquear os botões
    ui.widgetlist->setEnabled(true);
    ui.addB->setEnabled(true);
    ui.editB->setEnabled(true);
    ui.info->setEnabled(true);
    ui.redo->setEnabled(true);

    QFileInfo folderInfo(folderName);
    QDir directory(folderName);

    QStringList files = directory.entryList(QDir::Files);
    folderitemquantity = files.size();
    imagelist.resize(folderitemquantity);

    foreach(const QString & file, files) {
        QString filePath = directory.filePath(file);
        QFileInfo fileInfo(filePath);
        totalSize += fileInfo.size();

        // Detectar formato baseado na extensão do arquivo
        QString extension = fileInfo.suffix().toLower();
        QString format = QImageReader::imageFormat(filePath);

        if ((extension == "png" || extension == "jpg" || extension == "bmp" || extension == "jpeg") &&
(!format.isEmpty() || extension == "jpg" || extension == "jpeg")) {
            qDebug() << "File:" << filePath << "Format based on extension:" << extension << "Detected
format:" << format;
            WidgetListset(file);
            imagelist[i] = filePath;
            i++;
        }
        else {
            qDebug() << "File:" << filePath << "is not a supported format";
        }
    }
```

```
        if (imagelist.empty()) {
            ui.label->setText("No valid images found in the folder.");
        }
        else {
            images(imagelist);
        }

        ui.foldername->setText("    Folder name: " + folderInfo.fileName());
        ui.foldersize->setText("    Folder size: " + QString::number(totalSize * 0.000001, 'f', 2) + "mb");
        ui.folderitemquantity->setText("    item quantity: " + QString::number(folderitemquantity));

        copyImagesToObjFolder(folderName, darknetfolder + "/darknet/data/obj");
    }
}

void interface::on_openFile_clicked() {
    QString darknetfilepath = QFileDialog::getExistingDirectory(nullptr, "Select a darknet folder", "", QFileDialog::ShowDirsOnly);
    if (darknetfilepath.isEmpty()) {
        ui.label->setText("No folder selected.");
        return;
    }
    qDebug() << darknetfilepath;

    darknetfolder = darknetfilepath;

    clearEverything();

    QString objFolderPath = darknetfolder + "/data/obj";
    QDir objDir(objFolderPath);
    if (!objDir.exists()) {
        ui.label->setText("The selected folder does not contain the expected structure.");
        return;
    }

    QStringList nameFilters;
    nameFilters << "*.jpg" << "*.jpeg" << "*.png";
    QFileInfoList fileList = objDir.entryInfoList(nameFilters, QDir::Files);

    qint64 totalSize = 0; // Initialize total size

    for (const QFileInfo& fileInfo : fileList) {
        imagelist.push_back(fileInfo.absoluteFilePath());
        WidgetListset(fileInfo.absoluteFilePath());
        totalSize += fileInfo.size(); // Accumulate the file size
    }

    if (imagelist.empty()) {
        ui.label->setText("No images found in the specified folder.");
        return;
```

```
    }

    folderitemquantity = imagelist.size();

    QString objNamesFilePath = darknetfolder + "/data/obj.names";
    QFile objNamesFile(objNamesFilePath);
    if (!objNamesFile.exists()) {
        ui.label->setText("The obj.names file does not exist in the expected location.");
        return;
    }

    if (!objNamesFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        ui.label->setText("Failed to open the obj.names file.");
        return;
    }

    QTextStream in(&objNamesFile);
    while (!in.atEnd()) {
        QString line = in.readLine().trimmed();
        if (!line.isEmpty()) {
            addDynamicButton(line);
        }
    }
    objNamesFile.close();

    if (!imagelist.empty()) {
        imageIndex = 0;
        images({ imagelist[imageIndex] });
        loadCoords(scene);  // Load coordinates into the main scene
        ui.widgetlist->setCurrentRow(imageIndex);
        QFileInfo fileName(imagelist[imageIndex]);
        ui.label->setText(fileName.fileName());
    }

    QFileInfo folderInfo(darknetfilepath);

    ui.foldername->setText("    Folder name: " + folderInfo.fileName());
    ui.foldersize->setText("    Folder size: " + QString::number(totalSize * 0.000001, 'f', 2) + " MB");
    ui.folderitemquantity->setText("    Item quantity: " + QString::number(folderitemquantity));

    ui.widgetlist->setEnabled(true);
    ui.addB->setEnabled(true);
    ui.editB->setEnabled(true);
    ui.redo->setEnabled(true);
    ui.test->setEnabled(true);
    ui.info->setEnabled(true);
    yolov4ComboBox();
}

bool interface::copyDirRecursively(const QString& srcPath, const QString& dstPath) {
    QDir srcDir(srcPath);
```

```
    if (!srcDir.exists()) {
        return false;
    }

    QDir dstDir(dstPath);
    if (!dstDir.exists()) {
        if (!dstDir.mkpath(".")) {
            return false;
        }
    }

    foreach(QString file, srcDir.entryList(QDir::Files)) {
        QString srcFilePath = srcPath + "/" + file;
        QString dstFilePath = dstPath + "/" + file;
        if (!QFile::copy(srcFilePath, dstFilePath)) {
            return false;
        }
    }

    foreach(QString dir, srcDir.entryList(QDir::Dirs | QDir::NoDotAndDotDot)) {
        QString srcDirPath = srcPath + "/" + dir;
        QString dstDirPath = dstPath + "/" + dir;
        if (!copyDirRecursively(srcDirPath, dstDirPath)) {
            return false;
        }
    }

    return true;
}

//---------------------- BUTTONS MANAGMENT---------------------------
void interface::on_addB_clicked() {
    QColor color = Qt::black; // Define a cor preta como padrão

    // Cria uma nova janela de diálogo
    QDialog dialog(this);
    dialog.setWindowTitle("New button");

    // Layout para a janela de diálogo
    QVBoxLayout* dialogLayout = new QVBoxLayout(&dialog);

    // Info label
    QLabel* infolabel = new QLabel("Choose a button name and color", &dialog);
    dialogLayout->addWidget(infolabel);

    // LineEdit para o nome do botão
    QLineEdit* objectnameB = new QLineEdit(&dialog);
    objectnameB->setPlaceholderText("Button name");
    dialogLayout->addWidget(objectnameB);

    // Define o foco no LineEdit
```

```
    objectnameB->setFocus();

    QLabel* infolabel1 = new QLabel("\nChoose a color for the selection", &dialog);
    dialogLayout->addWidget(infolabel1);

    QHBoxLayout* colorLayout = new QHBoxLayout();
    dialogLayout->addLayout(colorLayout);

    QPushButton* colorButton = new QPushButton("Color", &dialog);
    colorLayout->addWidget(colorButton);

    QLabel* colorDisplay = new QLabel(&dialog);
    colorDisplay->setStyleSheet("background-color: black; border: 1px solid black;");
    colorLayout->addWidget(colorDisplay);

    connect(colorButton, &QPushButton::clicked, [&]() {
        QColor selectedColor = QColorDialog::getColor();
        if (selectedColor.isValid()) {
            color = selectedColor;
            colorDisplay->setStyleSheet(QString("background-color: %1; border: 1px solid
black;").arg(color.name()));
        }
        // Se nenhuma cor foi escolhida, a cor padrão permanece preta
        });

    QLabel* infolabel2 = new QLabel("To stop the selection, click again on the created button", &dialog);
    dialogLayout->addWidget(infolabel2);

    // Botões OK e Cancelar
    QDialogButtonBox* buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok |
QDialogButtonBox::Cancel, Qt::Horizontal, &dialog);
    dialogLayout->addWidget(buttonBox, 0, Qt::AlignHCenter);

    // Conecta os botões OK e Cancelar para aceitar ou rejeitar a janela de diálogo
    QObject::connect(buttonBox, &QDialogButtonBox::accepted, &dialog, &QDialog::accept);
    QObject::connect(buttonBox, &QDialogButtonBox::rejected, &dialog, &QDialog::reject);

    // Ajusta o tamanho do diálogo ao conteúdo
    dialog.adjustSize();

    // Executa a janela de diálogo
    if (dialog.exec() == QDialog::Accepted) {
        QString objectnameButton = objectnameB->text();
        if (!objectnameButton.isEmpty()) {
            addDynamicButtonHelper(objectnameButton, color);
        }
    }
}

void interface::on_editB_clicked() {
    // create a dialog window
    QDialog dialog(this);
```

```
dialog.setWindowTitle("Button list");

// main layout
QVBoxLayout* vdialogLayout = new QVBoxLayout(&dialog);

// list of buttons
QListWidget* listB = new QListWidget(&dialog);
vdialogLayout->addWidget(listB);

// obtains the buttons for the layout
QLayout* layout = ui.objetos->layout();
for (int i = 0; i < layout->count(); ++i) {
   // verification
   if (QPushButton* button = qobject_cast<QPushButton*>(layout->itemAt(i)->widget())) {
      // button add
      QListWidgetItem* item = new QListWidgetItem(button->text(), listB);
      item->setData(Qt::UserRole, QVariant::fromValue(button));
   }
}

// second layout for organization
QHBoxLayout* hdialogLayout = new QHBoxLayout(&dialog);

// delete button
QPushButton* deleteB = new QPushButton("Delete", &dialog);
hdialogLayout->addWidget(deleteB);

// edit button
QPushButton* editB = new QPushButton("Edit", &dialog);
hdialogLayout->addWidget(editB);

// add the vertical layout to the horizontal one
vdialogLayout->addLayout(hdialogLayout);

// connects the click on the list to the actual button
connect(listB, &QListWidget::itemClicked, this, [&](QListWidgetItem* item) {
   // verification
   if (item) {
      // obtains the button associated to the item
      selectedButton = item->data(Qt::UserRole).value<QPushButton*>();
      selectedItem = item;
      listbIndexSelected = listB->row(item);
   }
   });

// edit button
connect(editB, &QPushButton::clicked, [&]() {
   if (selectedButton) {
      // ask for the new name
      QString oldobjectname = selectedButton->text();
      bool ok;
```

```
        QString newName = QInputDialog::getText(&dialog, tr("Edit button"),
            tr("Choose the new button name"),
            QLineEdit::Normal,
            selectedButton->text(),
            &ok);

        // verification and update to the list
        if (ok && !newName.isEmpty()) {
            // update the name on the storage array and text variables
            dinamicButtons[listbIndexSelected].button->setText(newName);
            selectedButton->setText(dinamicButtons[listbIndexSelected].button->text());
            selectedItem->setText(dinamicButtons[listbIndexSelected].button->text());

            // rename
            QDir directory(dinamicButtons[listbIndexSelected].dirButtons);
            QString oldPath = QDir(dinamicButtons[listbIndexSelected].dirButtons).filePath(oldobjectname);
            QString newPath = QDir(dinamicButtons[listbIndexSelected].dirButtons).filePath(dinamicButtons[listbIndexSelected].button->text());
            dinamicButtons[listbIndexSelected].dirButtons = newPath;
            dinamicButtons[listbIndexSelected].button->setObjectName(QString(newName));
            if (directory.rename(oldPath, newPath)) {
                ui.label->setText("Folder renamed successfully");
            }
            else {
                ui.label->setText("Error renaming the folder");
            }
        }
        else {
            ui.label->setText("Name not valid");
        }
    }
    else {
        ui.label->setText("Selection button does not exist");
    }
    });

  // delete button
  connect(deleteB, &QPushButton::clicked, [&]() {
    if (selectedButton && selectedItem) {
        // Cria o caminho
        QString path = QDir(dinamicButtons[listbIndexSelected].dirButtons).filePath(dinamicButtons[listbIndexSelected].button->text());
        QDir directory(path);

            // Remove o item da lista
        listB->takeItem(listB->row(selectedItem));

        // Remove do vetor de armazenamento
        dinamicButtons.erase(dinamicButtons.begin() + listbIndexSelected);
```

```
            delete selectedButton;

            // size vector
            dButtonIndex--;
            dinamicButtons.resize(dButtonIndex + 1);

            objdatafilechange();//updates the obj.data in darknet folder

            // Limpa a seleção
            selectedButton = nullptr;
            selectedItem = nullptr;
            listbIndexSelected = -1;
        }
        else {
            ui.label->setText("Selection is null or no button is selected");
        }
        });

    dialog.exec();
}

void interface::on_info_clicked() {
    QDialog dialog(this);
    dialog.setWindowTitle("Macros info");

    QVBoxLayout* vdialogLayout = new QVBoxLayout(&dialog);
    QLabel* label1 = new QLabel(&dialog);
    label1->setText("A - Previous Image \nD - Next Image\n1 - First Class Button\n2 - Second Class Button\n3 - Third Class Button\nT - Enable Test");
    vdialogLayout->addWidget(label1);

    dialog.exec();
}

void interface::on_button_clicked() {
    QPushButton* clickedButton = qobject_cast<QPushButton*>(sender());
    if (clickedButton) {
        ui.label->setText("Button clicked: " + clickedButton->text());
        qDebug() << "Button clicked: " << clickedButton->text();

        int buttonIndex = -1;
        for (int i = 0; i < dinamicButtons.size(); ++i) {
            if (dinamicButtons[i].button == clickedButton) {
                buttonIndex = i;
                break;
            }
        }

        if (buttonIndex != -1) {
            qDebug() << "Button index: " << buttonIndex;
            if (mousePressEventEnabled) {
```

```
            // Inicializa a seleção
            mouseClickCounter = 0;
            dButtonIndex = buttonIndex;
            customCursor();

            // Desabilita outros botões
            for (int i = 0; i < dinamicButtons.size(); ++i) {
                if (i != buttonIndex) {
                    dinamicButtons[i].button->setEnabled(false);
                }
            }

            // Desabilita outros controles
            ui.info->setEnabled(false);
            ui.widgetlist->setEnabled(false);
            ui.addB->setEnabled(false);
            ui.editB->setEnabled(false);
            ui.undoSelec->setEnabled(true);

            // Habilita o estado de captura de mouse
            mousePressEventEnabled = false;
        }
        else {
            // Finaliza a seleção
            mousePressEventEnabled = true;
            ui.graphicsView->viewport()->unsetCursor();

            // Salva as seleções no arquivo
            fileCreate();

            // Reabilita todos os botões
            for (int i = 0; i < dinamicButtons.size(); ++i) {
                dinamicButtons[i].button->setEnabled(true);
            }

            // Reabilita outros controles
            ui.info->setEnabled(true);
            ui.widgetlist->setEnabled(true);
            ui.addB->setEnabled(true);
            ui.editB->setEnabled(true);
            ui.undoSelec->setEnabled(false);

            // Carrega e redesenha todas as seleções na cena principal
            loadCoords(scene);  // Use the main scene
        }
    }
    else {
        ui.label->setText("Button not found in dinamicButtons array.");
        qDebug() << "Button not found in dinamicButtons array.";
    }
}
```

```
}

void interface::addDynamicButton(const QString& buttonName) {
   static int hueOffset = 240;  // Start with a hue offset of 0 (which corresponds to red)

   // HSV color model: Start with red (hue 0), full saturation, and full brightness
   QColor borderColor = QColor::fromHsv(hueOffset, 255, 255);

   // Add the dynamic button with the calculated color
   addDynamicButtonHelper(buttonName, borderColor);

   // Increment the hueOffset for the next button, looping within the hue range [0, 360)
   const int hueIncrement = 60;  // Change this value to control the color shift between buttons
   hueOffset = (hueOffset + hueIncrement) % 360;
}

void interface::addDynamicButtonHelper(const QString& buttonName, const QColor& color) {
   dButtonIndex++;
   dinamicButtons.resize(dButtonIndex + 1); // Resize button vector
   dinamicButtons[dButtonIndex].color = color;

   // Cria o botão com o nome predefinido
   QPushButton* newButton = new QPushButton(buttonName);
   dinamicButtons[dButtonIndex].button = newButton;
   dinamicButtons[dButtonIndex].button->setObjectName(buttonName);

   // Atualiza o arquivo obj.data
   objdatafilechange();

   // Define o estilo do botão com a cor especificada
   QString buttonStyle = QString("border: 2px solid %1;").arg(color.name());
   dinamicButtons[dButtonIndex].button->setStyleSheet(buttonStyle);

   // Adiciona o botão ao layout
   QVBoxLayout* layout = qobject_cast<QVBoxLayout*>(ui.objetos->layout());
   if (layout) {
      int spacerIndex = layout->indexOf(ui.verticalSpacer);
      layout->insertWidget(spacerIndex, newButton);
   }

   // Conecta o botão ao slot on_button_clicked
   QObject::connect(dinamicButtons[dButtonIndex].button, &QPushButton::clicked, this,
&interface::on_button_clicked);
}

void interface::keyPressEvent(QKeyEvent* event) {
   switch (event->key()) {
   case Qt::Key_T:
      // Always active
      on_test_clicked();  // Call the on_test_clicked() function
      break;
```

```
case Qt::Key_1:
    // Always active
    if (!dinamicButtons.empty()) {
        QPushButton* firstButton = dinamicButtons[0].button;
        if (firstButton) {
            firstButton->click();  // Simulate a click on the first dynamic button
        }
    }
    break;

case Qt::Key_2:
    // Always active
    if (dinamicButtons.size() > 1) {
        QPushButton* secondButton = dinamicButtons[1].button;
        if (secondButton) {
            secondButton->click();  // Simulate a click on the second dynamic button
        }
    }
    break;

case Qt::Key_3:
    // Always active
    if (dinamicButtons.size() > 2) {
        QPushButton* thirdButton = dinamicButtons[2].button;
        if (thirdButton) {
            thirdButton->click();  // Simulate a click on the third dynamic button
        }
    }
    break;

case Qt::Key_A:
    // Active only when mousePressEventEnabled is true
    if (mousePressEventEnabled) {
        previousImage();
    }
    else {
        QWidget::keyPressEvent(event);  // Default handling for the "A" key
    }
    break;

case Qt::Key_D:
    // Active only when mousePressEventEnabled is true
    if (mousePressEventEnabled) {
        nextImage();
    }
    else {
        QWidget::keyPressEvent(event);  // Default handling for the "D" key
    }
    break;
```

```
    default:
        QWidget::keyPressEvent(event);  // Default handling for all other keys
        break;
    }
}

//------------------ IMAGES MANAGMENT---------------------
void interface::copyImagesToObjFolder(const QString& sourceFolder, const QString& destinationFolder)
{
    QString sourcfol = sourceFolder;
    QString destfol = destinationFolder;

    destfol = duplicatedarknet(destfol);

    QDir sourceDir(sourceFolder);
    QDir destDir(destinationFolder);

    if (!destDir.exists()) {
        if (!destDir.mkpath(".")) {
            qDebug() << "Failed to create destination directory:" << destinationFolder;
            return;
        }
    }

    QStringList imageFiles = sourceDir.entryList(QDir::Files);

    foreach(const QString & imageFile, imageFiles) {
        QString sourceFilePath = sourceDir.absoluteFilePath(imageFile);
        QString destFilePath = destDir.absoluteFilePath(imageFile);
        destFilePath = duplicatedarknet(destFilePath);

        if (QFile::exists(destFilePath)) {
            QFile::remove(destFilePath);
        }

        if (QFile::copy(sourceFilePath, destFilePath)) {
            qDebug() << "Copied" << sourceFilePath << "to" << destFilePath;
        }
        else {
            qDebug() << "Failed to copy" << sourceFilePath << "to" << destFilePath;
        }
    }
}

void interface::previousImage() {
    scene->clear();
    if (imageIndex > 0) {
        imageIndex--;
    }
    else {
        imageIndex = imagelist.size() - 1;
```

```
    }
    images({ imagelist[imageIndex] });
    ui.widgetlist->setCurrentRow(imageIndex);
    QFileInfo fileName(imagelist[imageIndex]);
    ui.label->setText(fileName.fileName());
    clearSelections();
    loadCoords(scene);  // Pass the main scene to loadCoords
}

void interface::nextImage() {
    scene->clear();
    if (imageIndex < imagelist.size() - 1) {
        imageIndex++;
    }
    else {
        imageIndex = 0;
    }
    images({ imagelist[imageIndex] });
    ui.widgetlist->setCurrentRow(imageIndex);
    QFileInfo fileName(imagelist[imageIndex]);
    ui.label->setText(fileName.fileName());
    clearSelections();
    loadCoords(scene);  // Pass the main scene to loadCoords
    if (darknetDetect) {
        bool selectionVerification = alreadySelected(imagelist[imageIndex]);
        qDebug() << "Selection verification:" << selectionVerification;
        if (selectionVerification == false) {
            qDebug() << "DARKNET SELECTION ACTIVATED";

            if (inferprocess && inferprocess->state() == QProcess::Running) {
                QString imageFileName = imagelist[imageIndex];
                qDebug() << "Sending image path to Darknet process:" << imageFileName;

                // Envie o caminho da imagem para o processo do Darknet
                inferprocess->write(imageFileName.toUtf8() + '\n');
            }
            else {
                qDebug() << "Darknet process is not running.";
            }
        }
    }
}

bool interface::alreadySelected(const QString& imageName) {
    // Cria um QFileInfo para a imagem com caminho absoluto
    QFileInfo imageFile(imageName);

    qDebug() << "Verificando existência da imagem:" << imageName;

    // Verifica se a imagem existe
    if (!imageFile.exists()) {
```

```
        qDebug() << "Erro: A imagem" << imageFile.filePath() << "não existe.";
        return false;
    }

    // Extrai o diretório da imagem
    QString directoryPath = imageFile.absolutePath();

    // Cria o caminho do arquivo de texto correspondente
    QString txtFilePath = directoryPath + "/" + imageFile.completeBaseName() + ".txt";
    QFileInfo txtFile(txtFilePath);

    qDebug() << "Verificando existência do arquivo de texto:" << txtFilePath;

    if (txtFile.exists()) {
        qDebug() << "File already has selections";
        return true;
    }
    else {
        qDebug() << "File doesn't have selections";
        return false;
    }
}

void interface::on_widgetlist_clicked() {
    QListWidgetItem* selectedItem = ui.widgetlist->currentItem();
    if (!selectedItem) {
        ui.label->setText("No image selected.");
        return;
    }

    int newIndex = ui.widgetlist->row(selectedItem);
    if (newIndex < 0 || newIndex >= imagelist.size()) {
        ui.label->setText("Failed to load the image: Index out of range.");
        return;
    }

    imageIndex = newIndex;  // Update the current index

    // Clear selections if there's an active dynamic button with valid coordinates
    if (dButtonIndex != -1 && !dinamicButtons[dButtonIndex].firstc.isEmpty() &&
!dinamicButtons[dButtonIndex].secondc.isEmpty()) {
        clearSelections();
    }

    // Load the selected image and its coordinates
    images({ imagelist[imageIndex] });
    loadCoords(scene);

    // Update the label with the image's file name
    QFileInfo fileInfo(imagelist[imageIndex]);
    ui.label->setText(fileInfo.fileName());
```

```
    // Reset focus to the main widget to enable macro capture again
    this->setFocus();
}

void interface::WidgetListset(const QString& file) {
    QFileInfo fileInfo(file);
    ui.widgetlist->addItem(fileInfo.fileName());
}

void interface::images(std::vector<QString> imagelist) {
    scene->clear();
    if (imagelist.empty()) {
        ui.label->setText("Folder list empty.");
        return;
    }

    QString imagePath = imagelist[0];
    qDebug() << "Attempting to load image:" << imagePath;

    QImageReader reader(imagePath);
    if (!reader.canRead()) {
        ui.label->setText(QString("Cannot read the image: %1").arg(imagePath));
        qDebug() << "Cannot read the image:" << imagePath << "Error:" << reader.errorString();
        return;
    }

    QImage image = reader.read();
    if (image.isNull()) {
        ui.label->setText(QString("Failed to load the image: %1").arg(imagePath));
        qDebug() << "Failed to load the image:" << imagePath << "Error:" << reader.errorString();
        return;
    }

    QPixmap pixmap = QPixmap::fromImage(image);
    if (pixmap.isNull()) {
        ui.label->setText(QString("Failed to convert the image: %1").arg(imagePath));
        qDebug() << "Failed to convert the image:" << imagePath;
        return;
    }

    QGraphicsPixmapItem* pixmapItem = new QGraphicsPixmapItem(pixmap);
    scene->addItem(pixmapItem);

    // Ajusta a cena ao tamanho da imagem
    scene->setSceneRect(pixmap.rect());

    // Ajusta o QGraphicsView para escalar a cena
    ui.graphicsView->fitInView(scene->sceneRect(), Qt::KeepAspectRatio);

    // Conecta os eventos do mouse
```

```
    GV* graphicsView = qobject_cast<GV*>(ui.graphicsView);
    if (graphicsView) {
        QObject::disconnect(graphicsView, &GV::mouseMoved, this, &interface::updateMousePosition);
        QObject::disconnect(graphicsView, &GV::mousePressed, this, &interface::mouseClicked);

        QObject::connect(graphicsView, &GV::mouseMoved, this, &interface::updateMousePosition);
        QObject::connect(graphicsView, &GV::mousePressed, this, &interface::mouseClicked);
    }
    else {
        ui.label->setText("Failed to cast graphicsView to GV");
        qDebug() << "Failed to cast graphicsView to GV";
    }
}

//------------------------ MOUSE EVENTS MANAGEMENT--------------------------
void interface::customCursor() {
    QPixmap cursorPixmap(600, 600);
    cursorPixmap.fill(Qt::transparent);
    QPainter painter(&cursorPixmap);
    painter.setPen(Qt::black);
    painter.drawLine(300, 0, 300, 600);
    painter.drawLine(0, 300, 600, 300);
    painter.end();
    QCursor customCursor(cursorPixmap, 300, 300);
    ui.graphicsView->viewport()->setCursor(customCursor);

    qDebug() << "Custom cursor set.";
}

void interface::updateMousePosition(const QPointF& pos) {
    ui.label2->setText(QString("x: %1 y: %2").arg(pos.x()).arg(pos.y()));
}

void interface::mouseClicked(const QPointF& pos) {
    if (!mousePressEventEnabled) {
        qDebug() << "Mouse clicked at: " << pos;

        if (mouseClickCounter == 0) {
            dinamicButtons[dButtonIndex].firstc.push_back(pos);
            mouseClickCounter++;
            qDebug() << "First point selected: " << pos;
        }
        else if (mouseClickCounter == 1) {
            dinamicButtons[dButtonIndex].secondc.push_back(pos);
            mouseClickCounter = 0;
            dinamicButtons[dButtonIndex].selectionNr++;
            ui.selectionNr->setText(QString("Number of selections:
%1").arg(dinamicButtons[dButtonIndex].selectionNr));
            qDebug() << "Second point selected: " << pos;
            createSelection(scene);  // Use the main scene
        }
```

```
        else {
            ui.label->setText("Selection error");
            qDebug() << "Selection error: Unexpected mouseClickCounter value.";
        }
    }
    else {
        qDebug() << "Mouse click ignored: mousePressEventEnabled is false.";
    }
}

//-----------------------------------------------Selection Management----------------------
void interface::deleteCoordinatesFromFile(int selectionIndex) {
    // Construct the path to the coordinates file
    QString imagePath = imagelist[imageIndex];
    QFileInfo imageFileInfo(imagePath);
    QString baseName = imageFileInfo.completeBaseName();
    QString coordsFilePath = darknetfolder + "/darknet/data/obj/" + baseName + ".txt";
    coordsFilePath= duplicatedarknet(coordsFilePath);

    qDebug() << "Coordinates file path:" << coordsFilePath;

    QFile coordsFile(coordsFilePath);
    if (!coordsFile.exists()) {
        qDebug() << "Coordinates file does not exist.";
        return;
    }

    if (!coordsFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qDebug() << "Failed to open coordinates file for reading.";
        return;
    }

    // Read all lines from the file
    QTextStream in(&coordsFile);
    QStringList lines;
    while (!in.atEnd()) {
        lines.append(in.readLine());
    }
    coordsFile.close();

    // Check if the selection index is within bounds
    if (selectionIndex < 0 || selectionIndex >= lines.size()) {
        qDebug() << "Invalid selection index. Cannot delete.";
        return;
    }

    // Remove the line corresponding to the selectionIndex
    qDebug() << "Deleting line:" << lines[selectionIndex];
    lines.removeAt(selectionIndex);

    // Write the modified lines back to the file
```

```
    if (!coordsFile.open(QIODevice::WriteOnly | QIODevice::Text | QIODevice::Truncate)) {
        qDebug() << "Failed to open coordinates file for writing.";
        return;
    }
    QTextStream out(&coordsFile);
    for (const QString& line : lines) {
        out << line << "\n";
    }
    coordsFile.close();

    qDebug() << "Coordinates file updated successfully.";
}

void interface::loadCoords(QGraphicsScene* targetScene) {
    targetScene->clear();
    for (auto& button : dinamicButtons) {
        button.firstc.clear();
        button.secondc.clear();
    }

    QString imagePath = imagelist[imageIndex];
    QFileInfo imageFileInfo(imagePath);
    QString baseName = imageFileInfo.completeBaseName();

    QString objFolderPath = darknetfolder + "/darknet/data/obj";
    objFolderPath = duplicatedarknet(objFolderPath); // remove the possible /darknet/darknet
    QString coordsFilePath = objFolderPath + "/" + baseName + ".txt";

    QFile coordsFile(coordsFilePath);
    if (!coordsFile.exists()) {
        qDebug() << "File does not exist:" << coordsFilePath;
    }

    if (!coordsFile.open(QIODevice::ReadOnly)) {
        qDebug() << "Failed to open file:" << coordsFilePath;
    }

    QTextStream in(&coordsFile);
    while (!in.atEnd()) {
        QString line = in.readLine();
        if (line.isEmpty()) {
            qDebug() << "Empty line in file:" << coordsFilePath;
            continue;
        }

        QStringList coordinates = line.split(' ', Qt::SkipEmptyParts);

        if (coordinates.size() % 5 != 0) {
            qDebug() << "Invalid number of coordinate values in line:" << line;
            continue;
        }
```

```
        int classIndex = coordinates[0].toInt();
        for (int j = 0; j < coordinates.size(); j += 7) {
            bool ok1, ok2, ok3, ok4;
            float cx = coordinates[j + 1].toFloat(&ok1);
            float cy = coordinates[j + 2].toFloat(&ok2);
            float width = coordinates[j + 3].toFloat(&ok3);
            float height = coordinates[j + 4].toFloat(&ok4);

            float x1 = cx - width / 2;
            float y1 = cy - height / 2;
            float x2 = x1 + width;
            float y2 = y1 + height;

            if (ok1 && ok2 && ok3 && ok4) {
                QPointF first(x1, y1);
                QPointF second(x2, y2);
                dinamicButtons[classIndex].firstc.push_back(first);
                dinamicButtons[classIndex].secondc.push_back(second);
            }
            else {
                qDebug() << "Invalid coordinate values in line:" << line;
            }
            qDebug() << "Coordinate values:" << line;
        }
    }

    coordsFile.close();
    createSelection(targetScene);
}

void interface::createSelection(QGraphicsScene* targetScene) {
    targetScene->clear();  // Clear the scene before adding new selections

    QPixmap pixmap(imagelist[imageIndex]);
    QGraphicsPixmapItem* pixmapItem = new QGraphicsPixmapItem(pixmap);
    targetScene->addItem(pixmapItem);

    int imgWidth = pixmap.width();
    int imgHeight = pixmap.height();

    for (int i = 0; i < dinamicButtons.size(); ++i) {
        for (int j = 0; j < dinamicButtons[i].firstc.size(); ++j) {
            QPointF first = dinamicButtons[i].firstc[j];
            QPointF second = dinamicButtons[i].secondc[j];

            qreal x1 = first.x() * imgWidth;
            qreal y1 = first.y() * imgHeight;
            qreal x2 = second.x() * imgWidth;
            qreal y2 = second.y() * imgHeight;
```

```
            QRectF rect(QPointF(x1, y1), QPointF(x2, y2));
            QRectF normalizedRect = rect.normalized();

            QGraphicsRectItem* rectItem = new QGraphicsRectItem(normalizedRect);

            QPen pen(dinamicButtons[i].color);
            pen.setWidth(2);

            rectItem->setPen(pen);
            targetScene->addItem(rectItem);
        }
    }

    qDebug() << "Selections created.";
}

void interface::on_undoSelec_clicked() {
    if (dButtonIndex >= 0 && dButtonIndex < dinamicButtons.size()) {
        // Verifica se há seleções para desfazer
        if (!dinamicButtons[dButtonIndex].firstc.isEmpty() &&
!dinamicButtons[dButtonIndex].secondc.isEmpty()) {
            // Remove a última seleção feita
            dinamicButtons[dButtonIndex].firstc.pop_back();
            dinamicButtons[dButtonIndex].secondc.pop_back();

            // Recria as seleções restantes na cena principal
            createSelection(scene);  // Use the main scene

            // Atualiza o número de seleções
            dinamicButtons[dButtonIndex].selectionNr = dinamicButtons[dButtonIndex].firstc.size();
            ui.selectionNr->setText("Number of selections: " +
QString::number(dinamicButtons[dButtonIndex].selectionNr));

            // Enable the Undo button if there are still selections to undo
            ui.undoSelec->setEnabled(!dinamicButtons[dButtonIndex].firstc.isEmpty());
        }
        else {
            ui.label->setText("No selection to undo.");
            // Disable Undo button since there's nothing to undo
            ui.undoSelec->setEnabled(false);
        }
    }
    else {
        ui.label->setText("No dynamic button selected.");
        // Disable Undo button since there's nothing to undo
        ui.undoSelec->setEnabled(false);
    }
}

void interface::clearSelections() {
    for (auto& button : dinamicButtons) {
        button.firstc.clear();
```

```
        button.secondc.clear();
        button.selectionNr = 0;
    }
    qDebug() << "Selections cleared.";
}

void interface::clearEverything() {
    ui.widgetlist->clear();

    if (!dinamicButtons.empty()) {
        for (auto& item : dinamicButtons) {
            if (item.button) {
                delete item.button;
                item.button = nullptr;
            }
            item.firstc.clear();
            item.secondc.clear();
            item.selectionNr = 0;

            if (item.scene) {
                delete item.scene;
                item.scene = nullptr;
            }
        }
        dinamicButtons.clear();
    }
    imagelist.clear();
    imageIndex = 0;
    folderitemquantity = 0;
    dButtonIndex = -1;
    scene->clear();
}

void interface::on_redo_clicked() {
    // Create and configure the dialog
    QDialog* reviewDialog = new QDialog(this);
    reviewDialog->setWindowTitle("Review Selections");
    reviewDialog->resize(800, 600);

    QVBoxLayout* layout = new QVBoxLayout(reviewDialog);

    // Create the scene and view for the dialog
    QGraphicsScene* reviewScene = new QGraphicsScene();
    QGraphicsView* graphicsView = new QGraphicsView(reviewDialog);
    graphicsView->setScene(reviewScene);
    graphicsView->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    layout->addWidget(graphicsView);

    // Display the image and selections
    images({ imagelist[imageIndex] });  // Set up the image in the scene
    loadCoords(reviewScene);            // Load the coordinates and draw the selections in the dialog's scene
```

```
// Track the current selection
int currentSelectionIndex = -1;
QList<QGraphicsRectItem*> rectItems;

// Populate rectItems with the rectangles created in loadCoords, with the last item being first
QList<QGraphicsRectItem*> tempItems;
for (QGraphicsItem* item : reviewScene->items()) {
    if (QGraphicsRectItem* rectItem = qgraphicsitem_cast<QGraphicsRectItem*>(item)) {
        tempItems.append(rectItem);
    }
}

// Reverse the order
for (int i = tempItems.size() - 1; i >= 0; --i) {
    rectItems.append(tempItems[i]);
}

qDebug() << "Initial number of selections:" << rectItems.size();

// Button to cycle through selections
QPushButton* cycleButton = new QPushButton("Next Selection", reviewDialog);
layout->addWidget(cycleButton);

// Button to redo (delete) the highlighted selection
QPushButton* deleteButton = new QPushButton("Delete Selection", reviewDialog);
layout->addWidget(deleteButton);
deleteButton->setEnabled(false);  // Initially disabled until a selection is highlighted

// Connect the cycle button to cycle through selections
connect(cycleButton, &QPushButton::clicked, this, [&]() mutable {
    if (!rectItems.isEmpty()) {
        // Reset previous selection's color
        if (currentSelectionIndex >= 0) {
            QPen originalPen(dinamicButtons[dButtonIndex].color);
            originalPen.setWidth(2);
            rectItems[currentSelectionIndex]->setPen(originalPen);
        }

        // Move to the next selection
        currentSelectionIndex = (currentSelectionIndex + 1) % rectItems.size();
        qDebug() << "Highlighting selection at index:" << currentSelectionIndex;

        // Highlight the current selection
        QPen highlightPen(Qt::green);  // Use a contrasting color (e.g., green)
        highlightPen.setWidth(3);       // Make the line slightly thicker
        rectItems[currentSelectionIndex]->setPen(highlightPen);

        // Enable the redo button now that a selection is highlighted
        deleteButton->setEnabled(true);
    }
```

```
    });

  // Connect the redo button to delete the highlighted selection
  connect(deleteButton, &QPushButton::clicked, this, [&]() mutable {
    qDebug() << "Redo button clicked";
    qDebug() << "Current selection index:" << currentSelectionIndex;
    qDebug() << "Total selections before deletion:" << rectItems.size();

    if (currentSelectionIndex >= 0 && currentSelectionIndex < rectItems.size()) {
      // Remove the selected rectangle from the scene
      QGraphicsRectItem* rectItem = rectItems.takeAt(currentSelectionIndex);
      qDebug() << "Deleting rectangle at index:" << currentSelectionIndex;
      reviewScene->removeItem(rectItem);
      delete rectItem;

      // Remove the corresponding selection from dinamicButtons
      dinamicButtons[dButtonIndex].firstc.remove(currentSelectionIndex);
      dinamicButtons[dButtonIndex].secondc.remove(currentSelectionIndex);

      // Delete the corresponding line from the coordinates file
      deleteCoordinatesFromFile(currentSelectionIndex);

      // Update selectionNr correctly
      dinamicButtons[dButtonIndex].selectionNr = rectItems.size();
      qDebug() << "Selection removed. Total remaining:" << dinamicButtons[dButtonIndex].selectionNr;

      // Adjust the current selection index
      if (rectItems.isEmpty()) {
        // No selections left
        currentSelectionIndex = -1;
        deleteButton->setEnabled(false);
      }
      else {
        // Ensure the index wraps correctly if needed
        currentSelectionIndex = std::min(currentSelectionIndex, static_cast<int>(rectItems.size()) - 1);

        QPen highlightPen(Qt::green);
        highlightPen.setWidth(3);
        rectItems[currentSelectionIndex]->setPen(highlightPen);
        qDebug() << "Next selection highlighted at index:" << currentSelectionIndex;
      }

      // Update the UI
      ui.selectionNr->setText("Number of selections: " +
QString::number(dinamicButtons[dButtonIndex].selectionNr));
    }
    else {
      qDebug() << "Invalid selection index for deletion";
    }
    });
```

```
    // Show the dialog and wait for it to close
    reviewDialog->exec();
    delete reviewDialog;

    // After the dialog closes, refresh the main view
    scene->clear();                 // Clear the main scene
    loadCoords(scene);              // Reload the updated coordinates
    createSelection(scene);         // Recreate selections in the main scene
}

void interface::addCoordinatesToFile() {
    // Path to the coordinates file
    QString coordinatesFilePath = darknetfolder + "/darknet/data/coordinates.txt";
    coordinatesFilePath = duplicatedarknet(coordinatesFilePath);
    QFile coordinatesFile(coordinatesFilePath);

    if (!coordinatesFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qDebug() << "Failed to open coordinates file for reading:" << coordinatesFile.errorString()<<
coordinatesFilePath;
        return;
    }

    QTextStream in(&coordinatesFile);
    QStringList coordinatesList;

    while (!in.atEnd()) {
        QString line = in.readLine();
        coordinatesList.append(line);
    }

    coordinatesFile.close();

    if (coordinatesList.isEmpty()) {
        qDebug() << "Coordinates file is empty.";
        return;
    }

    // Path of the current image
    QString imagePath = imagelist[imageIndex];
    QFileInfo imageFileInfo(imagePath);
    QString baseName = imageFileInfo.completeBaseName();
    QImage image(imagePath);

    // Get the dimensions of the image
    qreal imgWidth = image.width();
    qreal imgHeight = image.height();

    // Path to the obj folder
    QString objFolderPath = darknetfolder + "/darknet/data/obj";
    objFolderPath = duplicatedarknet(objFolderPath);
```

```
QString newImagePath = objFolderPath + "/" + imageFileInfo.fileName();
QFile::copy(imagePath, newImagePath);

// Path to the corresponding .txt file for the image
QString cfolderPath = objFolderPath + "/" + baseName + ".txt";
QFile txtFile(cfolderPath);

if (!txtFile.open(QIODevice::WriteOnly | QIODevice::Truncate)) {
    qDebug() << "Failed to open target file for writing:" << txtFile.errorString();
    return;
}

QTextStream out(&txtFile);

for (const QString& line : coordinatesList) {
    QStringList coordinates = line.split(" ");
    if (coordinates.size() != 5) {
        qDebug() << "Invalid coordinates format in line:" << line;
        continue;
    }

    // Extract object index (class index) and coordinates
    int classIndex = coordinates[0].toInt();
    qreal x = coordinates[1].toDouble();
    qreal y = coordinates[2].toDouble();
    qreal width = coordinates[3].toDouble();
    qreal height = coordinates[4].toDouble();

    // Convert to center format (centerX, centerY)
    qreal centerX = x + width / 2.0;
    qreal centerY = y + height / 2.0;

    // Normalize with respect to image size
    double normalizedCenterX = centerX / imgWidth;
    double normalizedCenterY = centerY / imgHeight;
    double normalizedWidth = width / imgWidth;
    double normalizedHeight = height / imgHeight;

    // Write the class index and normalized coordinates to the output file
    out << QString::number(classIndex) << " "
        << normalizedCenterX << " " << normalizedCenterY << " "
        << normalizedWidth << " " << normalizedHeight << "\n";

    // Debug: Display normalized coordinates
    qDebug() << "Original (Real): X:" << x << "Y:" << y
        << "Width:" << width << "Height:" << height;
    qDebug() << "Converted to Center (Real): CenterX:" << centerX << "CenterY:" << centerY;
    qDebug() << "Normalized (0-1): CenterX:" << normalizedCenterX
        << "CenterY:" << normalizedCenterY
        << "Width:" << normalizedWidth << "Height:" << normalizedHeight;
}
```

```
    txtFile.close();

    qDebug() << "Coordinates added to file:" << cfolderPath;
}

void interface::fileCreate() {
    QString imagePath = imagelist[imageIndex];
    QFileInfo imageFileInfo(imagePath);
    QString baseName = imageFileInfo.completeBaseName();

    QString objFolderPath = darknetfolder + "/darknet/data/obj";
    objFolderPath = duplicatedarknet(objFolderPath);

    QString newImagePath = objFolderPath + "/" + imageFileInfo.fileName();
    QFile::copy(imagePath, newImagePath);

    QString cfolderPath = objFolderPath + "/" + baseName + ".txt";
    QFile txtFile(cfolderPath);

    if (txtFile.open(QIODevice::Append | QIODevice::ReadWrite)) {
        QTextStream out(&txtFile);
        for (int j = 0; j < dinamicButtons.size(); ++j) {
            int size = std::min(dinamicButtons[j].firstc.size(), dinamicButtons[j].secondc.size());
            for (int k = 0; k < size; ++k) {
                QPointF first = dinamicButtons[j].firstc[k];
                QPointF second = dinamicButtons[j].secondc[k];

                QPointF center((first.x() + second.x()) / 2, (first.y() + second.y()) / 2);
                qreal width = std::abs(first.x() - second.x());
                qreal height = std::abs(first.y() - second.y());

                out << QString::number(j) << " " << center.x() << " " << center.y() << " " << width << " " <<
height << "\n";

            }
        }
        txtFile.close();
        ui.label->setText(QString("Created a file named: %1").arg(baseName + ".txt"));
        qDebug() << "Created a file named: " << baseName + ".txt";
    }
    else {
        ui.label->setText("Failed to create the file.");
        qDebug() << "Failed to create the file.";
    }
}

//----------------------------------------------DARKNET MANAGMENT----------------------
void interface::runPythonScript(const QString& scriptPath, QString script) {
    QProcess process;
    // Defina o diretório de trabalho para a pasta darknet
```

```
    process.setWorkingDirectory(scriptPath);

    // Comando para executar o script Python
    QString command = "python";
    QStringList arguments;
    arguments << script;

    // Conecte sinais para capturar saída e erros
    connect(&process, &QProcess::readyReadStandardOutput, [&]() {
        qDebug() << "Logs: " << process.readAllStandardOutput();
        });

    connect(&process, &QProcess::readyReadStandardError, [&]() {
        qDebug() << "Erro do processo: " << process.readAllStandardError();
        });

    // Inicie o processo
    process.start(command, arguments);

    // Aguarde o término do processo
    if (!process.waitForFinished()) {
        qDebug() << "Erro ao executar o processo: " << process.errorString();
    }
    else {
        qDebug() << "Processo terminado com sucesso.";
    }
}

void interface::runDarknetexe(const QString& scriptPath) {
    QString script = scriptPath;

    // Define the absolute path to darknet.exe
    QString absolutePathToDarknet = script + "/darknet.exe";
    qDebug() << "Initial absolutePathToDarknet:" << absolutePathToDarknet;

    // Verify if the file exists
    if (!QFile::exists(absolutePathToDarknet)) {
        qDebug() << "Error: The file" << absolutePathToDarknet << "does not exist.";
        return;
    }

    qDebug() << "File exists. Proceeding with process setup.";

    // Create and configure the QDialog for output
    QDialog* dialog = new QDialog(this);
    dialog->setWindowTitle("Logs");

    QTextEdit* outputEdit = new QTextEdit(dialog);
    outputEdit->setReadOnly(true);

    QPushButton* closeButton = new QPushButton("Close", dialog);
```

```
QVBoxLayout* layout = new QVBoxLayout(dialog);
layout->addWidget(outputEdit);
layout->addWidget(closeButton);
dialog->setLayout(layout);

dialog->adjustSize();

qDebug() << "Dialog setup complete. Creating QProcess.";

// Create and configure the QProcess
QProcess* process = new QProcess(this);

// Verify and set the working directory
QString workingDirectory = script;
qDebug() << "Setting working directory to:" << workingDirectory;
process->setWorkingDirectory(workingDirectory);

// Check if the working directory is valid
if (!QDir(workingDirectory).exists()) {
    qDebug() << "Error: The working directory" << workingDirectory << "does not exist.";
    outputEdit->append("<font color='red'>Error: The working directory does not exist.</font>");
    dialog->exec();
    delete process;  // Ensure proper cleanup
    return;
}

/*QStringList arguments;
arguments << "detector" << "train"
    << "data/obj.data"
    << "cfg/yolov4-obj.cfg";*/

QStringList arguments;
arguments << "detector" << "train"
    << "data/obj.data"
    << "cfg/yolov4-obj.cfg"
    << "yolov4.conv.137";

qDebug() << "Arguments for process:" << arguments;

// Connect signals for capturing output and errors
QObject::connect(process, &QProcess::readyReadStandardOutput, [process, outputEdit]() {
    QString output = process->readAllStandardOutput();
    qDebug() << "Standard Output:" << output;
    outputEdit->append(output);
    });

QObject::connect(process, &QProcess::readyReadStandardError, [process, outputEdit]() {
    QString error = process->readAllStandardError();
    qDebug() << "Standard Error:" << error;
    outputEdit->append("<font color='red'>" + error + "</font>");
```

```
        });

    // Start the process
    qDebug() << "Starting process with path:" << absolutePathToDarknet;
    process->start(absolutePathToDarknet, arguments);

    if (!process->waitForStarted()) {
        qDebug() << "Error starting process:" << process->errorString();
        outputEdit->append("<font color='red'>Error starting process: " + process->errorString() + "</font>");
        dialog->exec();
        delete process;  // Ensure proper cleanup
        return;
    }

    qDebug() << "Process started successfully.";

    // Connect the close button to terminate the process
    QObject::connect(closeButton, &QPushButton::clicked, [process, dialog]() {
        qDebug() << "Close button clicked. Terminating process.";
        process->terminate();
        if (!process->waitForFinished(3000)) {
            process->kill();
            qDebug() << "Process killed.";
        }
        dialog->accept();
        });

    // Show the dialog
    dialog->exec();

    // After dialog is closed, check if the process is still running and terminate if needed
    if (process->state() != QProcess::NotRunning) {
        qDebug() << "Process still running after dialog closed. Terminating.";
        process->terminate();
        if (!process->waitForFinished(3000)) {
            process->kill();
            qDebug() << "Process killed after dialog closed.";
        }
    }

    delete process;
    qDebug() << "Process and dialog cleaned up.";
}

QString interface::duplicatedarknet(QString duplicate) {
    if (duplicate.contains("/darknet/darknet")) {
        duplicate = duplicate.replace("/darknet/darknet", "/darknet");
    }
    return duplicate;
}
```

```
void interface::on_train_clicked() {

   // Disable the Test button while training
   ui.test->setEnabled(false);

   qDebug() << "yolov4 cfg config";
   yolov4Config();

   qDebug() << "Python script exe:  " << darknetfolder;
   runPythonScript(darknetfolder, "process.py");

   qDebug() << " Darknet.exe";
   runDarknetexe(darknetfolder);

   // Re-enable the Test button after training
   ui.test->setEnabled(true);
}

void interface::processDarknetOutput() {
   QProcess* process = qobject_cast<QProcess*>(sender());
   if (process) {
      QString output = process->readAllStandardOutput();
      qDebug() << "Darknet process output:" << output;

      // Caminho para o arquivo de resultados
      QString resultFilePath = QDir::toNativeSeparators(darknetfolder + "/data/results.txt");

      // Abrir o arquivo em modo de escrita, o que irá truncar o arquivo automaticamente
      QFile resultFile(resultFilePath);
      if (resultFile.open(QIODevice::WriteOnly | QIODevice::Text)) {
         QTextStream out(&resultFile);
         out << output;  // Escreve a saída no arquivo, limpando o conteúdo anterior
         resultFile.close();
         qDebug() << "Output written to results file.";
      }
      else {
         qDebug() << "Failed to open results file for writing:" << resultFile.errorString();
      }

      // Verifica se o script Python está no caminho correto e é executável
      runPythonScript(darknetfolder + "/data", "resultsfilter.py");
      addCoordinatesToFile();
      loadCoords(scene);  // Use the main scene
   }
}

void interface::yolov4ComboBox() {
   // Save the current selection
   QString currentSelection = ui.comboBox->currentText();

   // Clear the existing items to avoid duplication
```

```
ui.comboBox->clear();

// Caminho absoluto para a pasta backup
QString backupFolderPath = darknetfolder + "/backup";

// Add "no target" if the directory path is empty
if (darknetfolder.isEmpty()) {
   ui.comboBox->addItem("No weights");
   weights = ""; // Set weights to empty if the directory is not configured
}
else {
   // Open the folder and list all files
   QDir dir(backupFolderPath);
   if (!dir.exists()) {
      qDebug() << "Error: The directory" << backupFolderPath << "does not exist.";
      ui.comboBox->addItem("no target");
   }
   else {
      QFileInfoList fileInfoList = dir.entryInfoList(QDir::Files | QDir::NoDotAndDotDot);
      if (fileInfoList.isEmpty()) {
         ui.comboBox->addItem("no target");
      }
      else {
         foreach(const QFileInfo & fileInfo, fileInfoList) {
            ui.comboBox->addItem(fileInfo.fileName());
         }

         // Restore the previous selection if it exists
         int index = ui.comboBox->findText(currentSelection);
         if (index != -1) {
            ui.comboBox->setCurrentIndex(index);
         }
         else {
            // Default to the first item if the previous selection is not found
            ui.comboBox->setCurrentIndex(0);
         }

         // Update weights based on the restored selection
         QString selectedFileName = ui.comboBox->currentText();
         if (selectedFileName != "no target") {
            weights = QDir(backupFolderPath).filePath(selectedFileName);
            qDebug() << "Weights initially set to:" << weights;
         }
         else {
            weights = "";
         }
      }
   }
}

// Connect the signal for selection change to update weights
```

```
    connect(ui.comboBox, QOverload<int>::of(&QComboBox::currentIndexChanged), this, [this, backupFolderPath]() {
        QString selectedFileName = ui.comboBox->currentText();
        if (selectedFileName != "no target") {
            weights = QDir(backupFolderPath).filePath(selectedFileName);
        }
        else {
            weights = "";
        }
        qDebug() << "Weights updated to:" << weights;
        });

    // Update the layout
    ui.buttongridLayout->update();
}

void interface::yolov4Config() {
    // Caminho absoluto para os arquivos
    QString dataFilePath = darknetfolder + "/data/obj.data";
    QString configFilePath = darknetfolder + "/cfg/yolov4-obj.cfg";

    qDebug() << "dataFilePath:" << dataFilePath << "configFilePath:" << configFilePath;

    // Lê o número de classes do arquivo obj.data
    int numClasses = 0;
    QFile dataFile(dataFilePath);
    if (dataFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream in(&dataFile);
        while (!in.atEnd()) {
            QString line = in.readLine().trimmed();
            if (line.startsWith("classes =")) {
                numClasses = line.split("=").last().trimmed().toInt();
                qDebug() << "Número de classes encontrado:" << numClasses;
                break;
            }
        }
        dataFile.close();
    }
    else {
        qDebug() << "Erro ao abrir o arquivo" << dataFilePath;
        return;
    }

    // Modifica o arquivo yolov4-obj.cfg
    QFile configFile(configFilePath);
    if (!configFile.open(QIODevice::ReadWrite | QIODevice::Text)) {
        qDebug() << "Erro ao abrir o arquivo" << configFilePath;
        return;
    }

    // Lê todo o conteúdo do arquivo
```

```
    QTextStream in(&configFile);
    QString content = in.readAll();
    configFile.seek(0);  // Volta ao início do arquivo para reescrever

    // Atualiza subdivisions
    content.replace(QRegularExpression("subdivisions=\\d+"), "subdivisions=64");
    qDebug() << "Atualizando subdivisions para: 64";

    // Atualiza max_batches
    int maxBatches = 2000 * numClasses;
    content.replace(QRegularExpression("max_batches=\\d+"),
QString("max_batches=%1").arg(maxBatches));
    qDebug() << "Atualizando max_batches para:" << maxBatches;

    // Atualiza steps (já está correto)
    int eightyPercent = static_cast<int>(maxBatches * 0.8);
    int ninetyPercent = static_cast<int>(maxBatches * 0.9);
    content.replace(QRegularExpression("steps=\\d+,\\d+"),
QString("steps=%1,%2").arg(eightyPercent).arg(ninetyPercent));
    qDebug() << "Atualizando steps para:" << eightyPercent << "," << ninetyPercent;

    // Atualiza apenas o filtro antes do bloco [yolo]
    QRegularExpression yoloBlockPattern("\\[yolo\\]");
    QRegularExpression convBlockPattern("\\[convolutional\\]");
    QRegularExpression filtersPattern("filters=\\d+");

    QRegularExpressionMatch yoloMatch = yoloBlockPattern.match(content);
    while (yoloMatch.hasMatch()) {
        int yoloPos = yoloMatch.capturedStart();

        // Encontra o bloco [convolutional] mais próximo antes do bloco [yolo]
        QRegularExpressionMatch convMatch = convBlockPattern.match(content, 0);
        int lastConvPos = -1;

        while (convMatch.hasMatch() && convMatch.capturedStart() < yoloPos) {
            lastConvPos = convMatch.capturedStart();
            convMatch = convBlockPattern.match(content, convMatch.capturedEnd());
        }

        if (lastConvPos != -1) {
            // Atualiza o filters deste bloco [convolutional]
            QRegularExpressionMatch filterMatch = filtersPattern.match(content, lastConvPos);
            if (filterMatch.hasMatch() && filterMatch.capturedStart() < yoloPos) {
                content.replace(filterMatch.capturedStart(),                filterMatch.capturedLength(),
QString("filters=%1").arg((numClasses + 5) * 3));
                qDebug() << "Atualizando filters para:" << (numClasses + 5) * 3 << "no bloco antes de" <<
yoloMatch.capturedStart();
            }
        }

        // Continua procurando o próximo bloco [yolo]
        yoloMatch = yoloBlockPattern.match(content, yoloPos + 1);
```

```
    }

    // Atualiza classes dentro do bloco [yolo]
    content.replace(QRegularExpression("classes=\\d+"), QString("classes=%1").arg(numClasses));
    qDebug() << "Atualizando classes para:" << numClasses;

    // Escreve o conteúdo modificado de volta ao arquivo
    configFile.resize(0); // Limpa o conteúdo do arquivo
    QTextStream out(&configFile);
    out << content;
    configFile.close();

    qDebug() << "Arquivo de configuração atualizado com sucesso.";
}

void interface::on_test_clicked() {
    // Alterna o valor de darknetDetect
    darknetDetect = !darknetDetect;
    yolov4ComboBox();

    // Atualiza o texto do rótulo para "Wait"
    ui.label->setText("Wait");

    if (darknetDetect) {
        // Inicia o processo de inicialização do Darknet
        initializeDarknet();

        // Usa um temporizador para aguardar antes de atualizar o texto
        QTimer::singleShot(7000, [this]() {
            // Atualiza o texto do rótulo com o valor atual de darknetDetect
            QString status = darknetDetect ? "Darknet detect enabled" : "Darknet detect disabled";
            ui.label->setText(status);
            });
    }
    else {
        // Se o darknetDetect estiver desativado, apenas atualiza o texto do rótulo
        QString status = darknetDetect ? "Darknet detect enabled" : "Darknet detect disabled";
        ui.label->setText(status);
    }
}

void interface::initializeDarknet() {
    qDebug() << "initializeDarknet called.";

    QString darknetCmd = darknetfolder + "/darknet.exe";  // Nome do executável darknet
    QString cfgFile = "cfg/yolov4-obj.cfg";  // Caminho relativo para o cfg
    QString dataFile = "data/obj.data";  // Caminho relativo para o data

    QStringList args;
    args << "detector" << "test" << dataFile << cfgFile << weights << "-dont_show" << "-ext_output"<<"-
map";
```

```
    inferprocess = new QProcess(this);

    // Conectar sinais de erro
    connect(inferprocess, &QProcess::readyReadStandardOutput, this, &interface::processDarknetOutput);

    // Definir o diretório de trabalho
    QString workingDir = QDir::toNativeSeparators(darknetfolder);  // Convertendo para o formato do Windows
    inferprocess->setWorkingDirectory(workingDir);

    qDebug() << "Working directory set to:" << workingDir;

    // Verificar se o caminho completo para o executável e arquivos está correto
    qDebug() << "Starting Darknet process with command:" << darknetCmd << "and arguments:" << args;

    inferprocess->start(darknetCmd, args);

    if (!inferprocess->waitForStarted()) {
        qDebug() << "Failed to start Darknet process. Error:" << inferprocess->errorString();
    }
}

void interface::objdatafilechange() {
    QString objDataFilePath = darknetfolder + "/darknet/data/obj.data";
    QString objNamesFilePath = darknetfolder + "/darknet/data/obj.names";

    objDataFilePath = duplicatedarknet(objDataFilePath);
    objNamesFilePath = duplicatedarknet(objNamesFilePath);

    // Atualiza o arquivo obj.data.txt
    QFile dataFile(objDataFilePath);
    if (!dataFile.exists()) {
        qDebug() << "File does not exist:" << objDataFilePath;
        return;
    }

    if (!dataFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qDebug() << "Failed to open file for reading:" << objDataFilePath;
        return;
    }

    QString content;
    QTextStream in(&dataFile);
    while (!in.atEnd()) {
        QString line = in.readLine();
        if (line.startsWith("classes =")) {
            line = "classes = " + QString::number(dinamicButtons.size());
        }
        content += line + "\n";
    }
```

```
    dataFile.close();

    if (!dataFile.open(QIODevice::WriteOnly | QIODevice::Text | QIODevice::Truncate)) {
        qDebug() << "Failed to open file for writing:" << objDataFilePath;
        return;
    }

    QTextStream out(&dataFile);
    out << content;
    dataFile.close();

    // Atualiza o arquivo obj.names
    QFile namesFile(objNamesFilePath);
    if (!namesFile.exists()) {
        qDebug() << "File does not exist:" << objNamesFilePath;
        return;
    }

    if (!namesFile.open(QIODevice::WriteOnly | QIODevice::Text | QIODevice::Truncate)) {
        qDebug() << "Failed to open file for writing:" << objNamesFilePath;
        return;
    }

    QTextStream namesOut(&namesFile);
    for (const auto& button : dinamicButtons) {
        namesOut << button.button->text() << "\n";
    }
    namesFile.close();
}



#endif // INTERFACE_CPP
```

Annex D. Python File Code (Process.py)

```
import glob, os

# Current directory
current_dir = os.path.dirname(os.path.abspath(__file__))

print(current_dir)

current_dir = 'data/obj'
```

```
# Percentage of images to be used for the test set
percentage_test = 10;

# Create and/or truncate train.txt and test.txt
file_train = open('data/train.txt', 'w')
file_test = open('data/test.txt', 'w')

# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.png")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))

    if counter == index_test:
        counter = 1
        file_test.write("data/obj" + "/" + title + '.png' + "\n")
    else:
        file_train.write("data/obj" + "/" + title + '.png' + "\n")
        counter = counter + 1

print("test and train file populated")
```

Annex E. Python File Code (ResultsFilter.py)

```
import re
import os

# Paths to files - Use an absolute path for obj.names
input_file = 'results.txt'
output_file = 'coordinates.txt'
obj_names_file = os.path.abspath('./obj.names')  # Absolute path

# Load class names from the obj.names file and map them to indices
try:
    with open(obj_names_file, 'r', encoding='utf-8') as file:
        class_names = [line.strip() for line in file.readlines()]
        class_name_to_index = {name: index for index, name in enumerate(class_names)}
except IOError as e:
    print(f'Error reading file: {e}')
```

```
    exit(1)

# Build a regex pattern to match any class name followed by coordinates
class_names_pattern = '|'.join([re.escape(name) for name in class_names])
regex = re.compile(rf'({class_names_pattern}):\s*\d+%\s*\(left_x:\s*(-?\d+)\s*top_y:\s*(-?\d+)\s*width:\s*(\d+)\s*height:\s*(\d+)\)')

# List to store extracted coordinates
coordinates = []

# Verify if the input file exists
if not os.path.exists(input_file):
    print(f'Input file does not exist: {input_file}')
    exit(1)

# Read the results file
try:
    with open(input_file, 'r', encoding='utf-8') as file:
        content = file.read()
        print(f"Content of results.txt: {content}")  # Debugging print
        matches = regex.findall(content)
        print(f"Matches found: {matches}")  # Debugging print
        for match in matches:
            class_name, left_x, top_y, width, height = match
            class_index = class_name_to_index[class_name]  # Get the index for the class name
            coordinates.append(f'{class_index} {left_x} {top_y} {width} {height}')

except IOError as e:
    print(f'Error reading file: {e}')
    exit(1)

# Write the coordinates to the output file
try:
    with open(output_file, 'w', encoding='utf-8') as file:
        file.write('\n'.join(coordinates))

except IOError as e:
    print(f'Error writing file: {e}')
    exit(1)

print(f'Coordinates saved to {output_file}')
```