Terra Inspectioneering

Thesis project Machine Learning algorithm in drone camera

Thesis report

Student: Siyu Chen

Student number: 91704

In-company mentor: David Smits

HZ-tutor: Peter van der Heide

02/06/2023

Abstract

This thesis project presents a novel approach for detecting corrosion using drone-based surveillance systems by modifying the YOLOv5 object detection model and deploying it to a VOXL 2 flight controller for real-time image processing. The central goal is to enhance the model's accuracy in detecting corrosion instances and make the developed solution easily replicable for future use within the company.

The first part of the project involves the construction of a comprehensive dataset using drone-captured images of vapor pipes, further enriched with data augmentation techniques. Following this, the YOLOv5 model is modified by adding a Convolutional Block Attention Module (CBAM) to better capture the fine-grained details of corrosion. The modified model is then optimized through a rigorous training and evaluation process, resulting in significantly improved performance metrics, as evidenced by the comparative analysis with the original YOLOv5 model.

The developed model is successfully deployed to the VOXL 2 flight controller, demonstrating its real-time corrosion detection capabilities in drone-captured images. Lastly, a toolkit is developed to facilitate the company's ability to train and adapt the model for future needs, ensuring the project's sustainability.

Despite the project's success, certain limitations are identified, such as dataset diversity and real-world performance evaluation, providing a direction for future research. This work contributes to the field of corrosion detection methodologies, presenting a robust, efficient, and adaptable model.

Foreword

I am privileged to have spent the past five months completing my thesis internship at Terra Inspectioneering. As an engineering student, this experience allowed me to apply the academic knowledge I acquired over the years to practical, real-world problems during the project on corrosion detection and drone-based systems. Both the creation of a comprehensive dataset and the modification and optimization of the YOLOv5 model for corrosion detection provided me with constant challenges and growth opportunities. This internship experience has made me more acutely aware of my capabilities, the areas for improvement, and the challenges involved in blending theory with practice - lessons that will undoubtedly prove valuable in my future professional pursuits.

Firstly, I would like to extend my heartfelt gratitude to HZ University of Applied Science and Terra Inspectioneering for providing the platform for my graduation internship. The opportunity to learn and grow in a professional environment has been invaluable.

Secondly, I owe a great deal of my project's success to the mentors at Terra Inspectioneering: David Smits. His timely advice, constant support, and insightful feedback has been instrumental in facilitating the successful completion of my project.

Lastly, I would like to express my appreciation to my examiners at HZ University of Applied Science: Peter van der Heide and Willem Haak. Their invaluable guidance during the preparation of this thesis report and their direction in the methodology design have been paramount to the progress and success of my project.

This period of internship has been an enriching journey of learning and personal development, and I am grateful to all who have contributed to it.

Siyu Chen
Middelburg, the Netherlands
27th May, 2023

Contents

1 Introduction

This section provides background information for the research.

1.1 Background

Terra Inspectioneering is a company which started as RoNik Inspectioneering on May 13th, 2016. Since then, Terra Inspectioneering is known for their inspections using aerial robots, also known as drones.

Figure 1. Company Terra Inspectioneering's Logo

Terra Inspectioneering specializes in conducting inspections in confined spaces, such as storage tanks, catering mainly to clients in the oil and gas sectors. However, the company also extends its services to companies operating in the food, beverage, power, and chemical sectors. The primary focus of Terra Inspectioneering's services is to measure the thickness of tank walls. To accomplish this, Terra Inspectioneering has developed and patented a drone-based technology for measuring wall thickness. In addition to wall thickness measurement, Terra Inspectioneering also offers visual and thermal inspections. The inspection data collected by the drone is stored on a cloud-based 3D model of the inspection object, which is accessible to the client.

VOXL 2 (ModalAI, 2023) is an autonomous computing platform that has been developed by ModalAI. This platform boasts of a SWAP-optimized design that is powered by Qualcomm QRB5165, featuring eight cores that can operate at speeds of up to 3.091 GHz and 8GB LPDDR5. Additionally, the platform has an embedded Neural Processing Unit (NPU) that can deliver 15 tera operations per second (TOPS) AI. By integrating a single board computer, depth camera, flight controller, and cellular modem, VOXL 2 (ModalAI, 2023) enables the creation of fully autonomous, connected drones. The platform can also run advanced algorithms for tasks such as object detection, mapping, and localization. In summary, VOXL 2 is a vital component of an autonomous drone architecture as it provides advanced processing capabilities and interfaces with the drone's various sensors and control systems.

Figure 2. Overview of VOXL 2. Reprinted from MODALAI (ModalAI, 2023).

VOXL 2 Features (ModalAI, 2023)

- 16 grams, 70x36mm SWAP-optimized design.
- Powered by Qualcomm® QRB5165: 8 cores up to 3.091 GHz, 8GB LPDDR5.
- 15 TOPS AI-embedded Neural Processing Units.
- Integrated flight controller on digital signal processor (DSP) with TDK® ICM-42688 IMU and ICP-10111 Barometer.

- 5G, 4G/LTE, Wi-Fi, Microhard add-on connectivity.

Software Features

- PX4, ROS 1 / 2, Ubuntu 18.04, Open CV, MAVROS, MAVSDK.
- Open-Source Linux kernel, cross-compilers.
- Docker build environment for graphics processing unit (CPU), graphics processing unit (GPU) (OpenCL), and DSP (Hexagon SDK) heterogeneous computer vision and deep learning processing.

1.2 Problem analysis

The aerial robot from Terra-Inspectioneering needs a new function of framing the defects automatically on the inspecting area through inspection.

From a bird's eye view, the structure of an automatic detection system deployed on VOXL 2 would involve several components working together. The VOXL 2 platform would serve as the central processing unit, receiving input from sensors such as cameras or a gyroscope. The onboard AI capabilities of VOXL 2 would then process this input in real-time to detect objects or events of interest.

The object detection algorithms used depend on the requirements given by the client. In this project, the object detection algorithm is to detect defects such as corrosion in the inspecting area through inspection. A TensorFlow Lite model is trained based on the object detection algorithm. The trained model is then deployed to the VOXL 2 through the VOXL software development kit (SDK). And all the detection is processed by this model.

Once a detection is made, the system could be configured to take various actions such as sending an alert or triggering a response. The system could also be connected to other devices or systems for further processing or analysis.

The hardware part should be constructed first and then the development of the training model. To achieve this, it is essential to learn how to build a TensorFlow Lite model as well as deploy the model to the aerial robot.

Hardware Theory of Operation

1. Power
   The VOXL Power Module (MDK-M0041) is designed to accept 2S-6S batteries as input, directly channeling this to the VOXL ESC (MDK-M0117). Additionally, it can regulate the voltage down to 5VDC.
   The power module (M0041-J1) forms the output. The 5V/6A rated regulator feeds the VOXL2 (MDK-M0054) and delivers power monitoring information over I2C, which is channeled to voxl-px4 via DSP, not apps_proc. This is done through the power connector, designated as M0054-J4.
2. Debug Connections
   After powering on, a user can connect to a Linux-based terminal via the USBC connector (M0054-J9) by employing the Android Debug Bridge (adb).
3. Image Sensors

A tracking sensor (ov7521, VGA, black and white, 30FPS) along with a high-resolution sensor (imx214, 4K, 30FPS) are connected to the VOXL2 M0054-J7 utilizing the M0084 dual camera flex.

4. Flight Controller
   Communication between the VOXL ESC (MDK-M0117) and voxl-px4 is facilitated over UART, with the connection established between M0117-J2 and M0054-J18.

Figure 3 shows the detailed hardware block diagram of this project. Figure 4 shows the software block diagram covering the voxl-camera-server.

Overall, the structure of an automatic detection system deployed on VOXL 2 would involve a combination of hardware and software components working together to achieve real-time detection and response. The following is the block diagram.

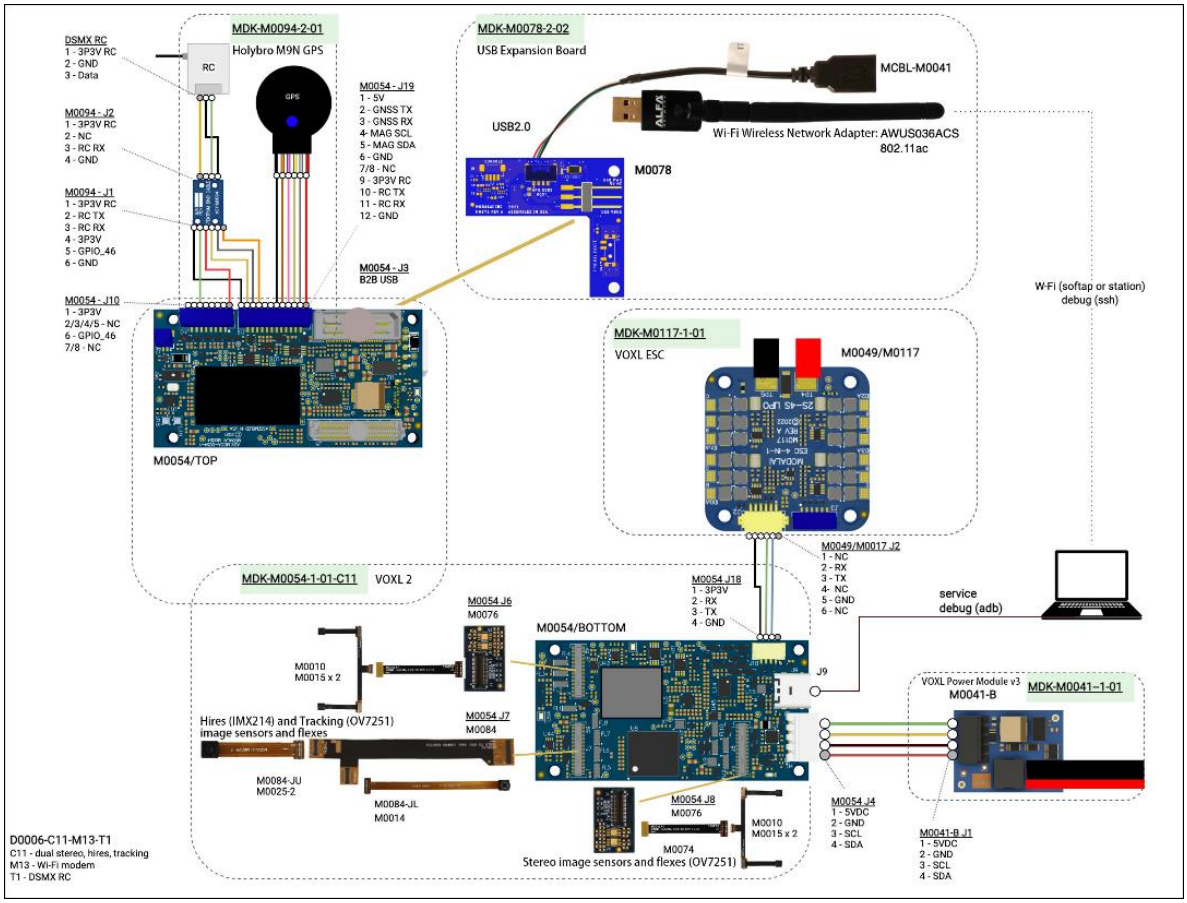


Figure 3. Overview of the hardware of the system

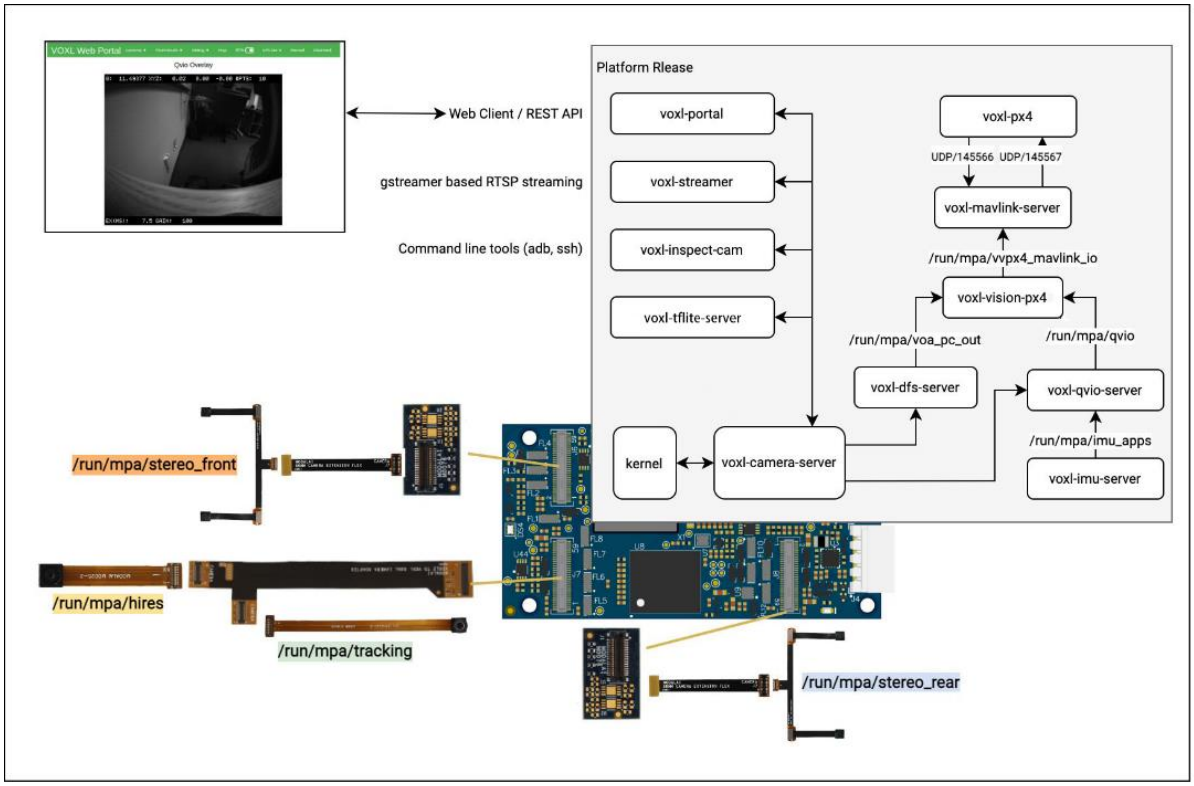


Figure 4. Overview of the software of the system

1.2.1 Goal and objectives

The system contains two main objectives: The first is to develop an automatic defect detection function that can be implemented on VOXL 2. Currently the VOXL 2 platform will be provided by the company. The objectives include building a dataset for detecting corrosion and an object detection model. What's more, it's also necessary to optimize the system for speed and accuracy and ensuring that the system could be easily integrated into the client's existing workflow.

In addition to the object detection model, the second objective of this project is to establish a comprehensive repository or tool that empowers the company to independently develop their own models in the future. This repository/tool serves as a foundational resource, allowing the company to create and train object detection models tailored to their specific requirements and domain expertise, even after the departure of the project developer.

The deployment of such a system would yield multiple benefits for Terra Inspectioneering, including cost savings from the reduction of personnel required for aerial robot operation and increased market competitiveness.

1.2.2 Current and desired situation

The current challenge faced by Terra Inspectioneering is the need for manual inspection and defect detection. This is particularly common in inspection scenarios involving the surfaces of pipes and painted and unpainted tanks. See the figures below.

Figure 5. photo of pipe with corrosion

While it may be relatively straightforward for an inspector to identify defects on the surface being inspected, in practical situations, the volume of information displayed on the monitor can be overwhelming. Fatigue and human error can lead to mistakes being made. Thus, it is imperative for Terra Inspectioneering to develop a new function for their aerial robot, enabling it to automatically identify the location of all defects. Upon completion of the inspection, it would be possible to mark all identified defects and transfer them to a figure format. The expected marked figure is shown as follows.

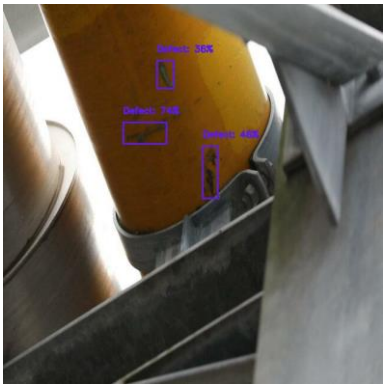


Figure 6. Figure after the process of automatic framing defect function

In the long term, the new function will not only increase the market competitiveness for Terra-Inspectioneering and reduce the workload of the inspectors but also reduce the workload for the person who is responsible for categorizing, which is important to show the data to clients.

According to the analysis above, the following comes out with the main research question and sub-research question with the reference to the system engineering phase.

1.2.3 Main research question

What is the system design of an automatic framing defects function on the VOXL 2 platform?

1.2.4 Sub-research question

1. What system could proceed with training and developing of an object detection model?
2. What is the sub-system design of this project?

3. What components are needed to build the subsystems?
4. What are the procedures to integrate these subsystems and components?
5. What are the test plans for system, subsystems, and components?

1.3 Problem statement

Terra Inspectioneering makes drones for confined space inspections. These drones contain measuring instruments and a camera. Terra Inspectioneering wants their aerial robot to have the function of automatically detecting the defects on the target surface. This needs to be developed using deep learning technology.

1.3.1 Ideal situation

Terra Inspectioneering should apply an automatic detecting defects function to their aerial robots to prevent missed inspections during the operation.

1.3.2 Problem

Currently, Terra Inspectioneering makes the inspection work manually using aerial robots. The drone operators need to visually inspect the surface for defects on the monitor. But in some cases, the drone operators may not be able to find the defects due to many reasons. By developing the function of automatic detection for their aerial robot. It would greatly increase the accuracy of the result of the inspection and reduce the workload for drone operators.

1.4 Criteria

This section primarily outlines the metrics that can serve as indicators of the performance of deep learning models. Further elaboration on these criteria is provided in section 2.6.

1. Accuracy
2. Precision
3. Recall
4. F1-score
5. mAP (mean Average Precision)

2 Theoretical Framework

This chapter mainly describes the theoretical framework of the research, including definitions and differences between machine learning and deep learning.

2.1 Definition of machine learning

The widely quoted definition of Machine learning by Tom Mitchell best explains machine learning in a nutshell. Here is what it says: “A computer program is said to learn from experience E with respect to some class of tasksT and performance measureP if its performance at tasks inT, as measured by P, improves with experience E.” (Shaikh, 2021)

In conclusion, the more experience E that being input, the better the performance is.

A typical example of machine learning is predicting weights based on the height of human beings. The following is the example data.

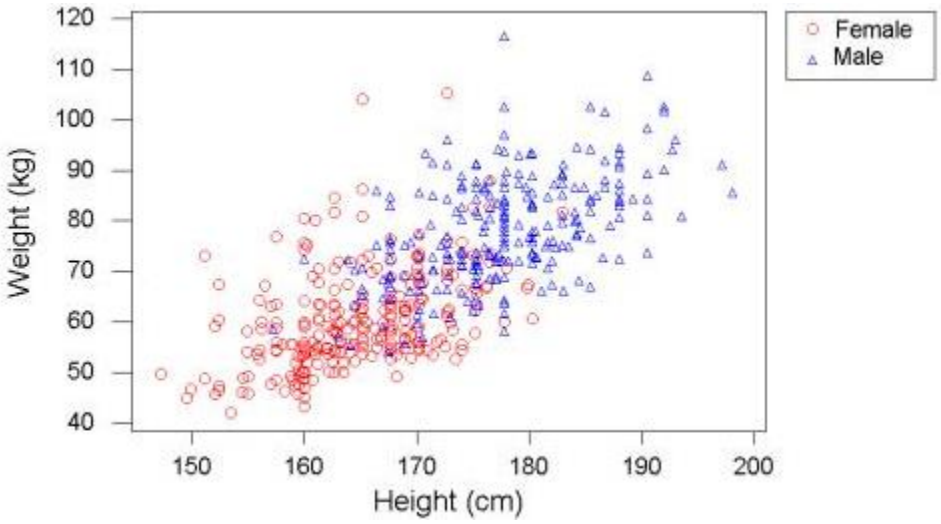


Figure 7. Weights related to height. Reprinted from Analytics Vidhya.

On the graph, the points represent the weights of females and males related to heights. The bunch of points is known, and the purpose of machine learning is to make a straight line to fit the sample points as much as possible, then this straight line is defined as learned by the machine.

The second example of machine learning is a storm prediction system.

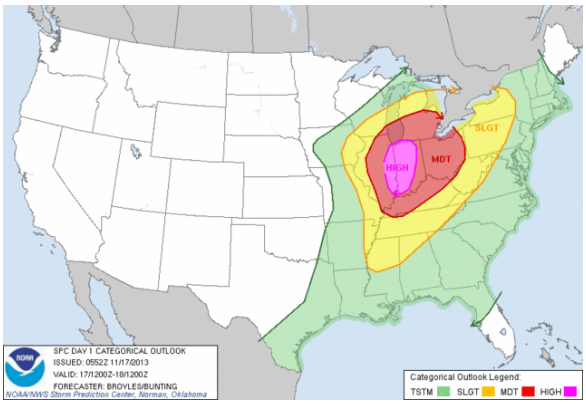


Figure 8. Storm Prediction. Reprinted from Analytics Vidhya.

First, the computer read all the historical storm data and from this vast amount of data, the computer learns certain 'patterns' of what conditions lead to storms.

For instance, the computer may be able to find out by studying historical data that when the temperature exceeds 40 degrees and the humidity is between 80 to 100, storms are prone to occur. The indicators such as 'temperature' and 'humidity' are the 'features' in machine learning, and these features are set manually. It is important to analyse which 'features' are important before making such a prediction system because the machine finds the corresponding pattern by analysing the data of these characteristics in the historical data.

It is important to understand the above point because it is an essential difference from deep learning.

2.2 Definition of deep learning

"Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones." (Gupta, 2022)

The following is an example that shows how people distinguish objects at the cognitive level.

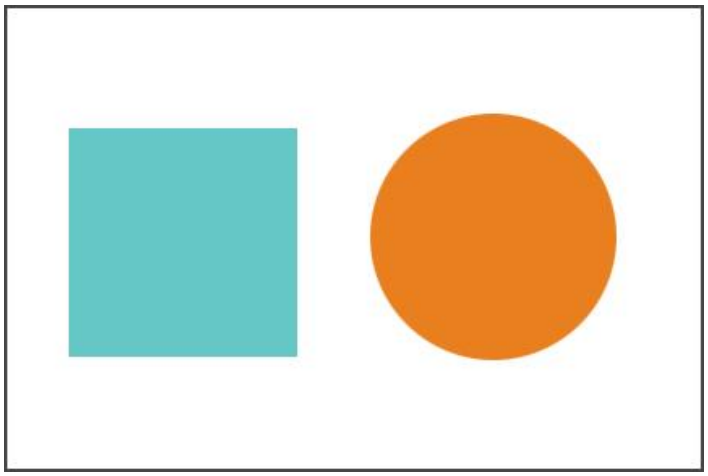


Figure 9. Shape detection. Reprinted from Analytics Vidhya.

The purpose for shape detection is to distinguish the shapes above. The first thing that eyes do is to see if the shape has four sides. If so, further check whether the four sides are connected and whether they are equal in length and whether they are perpendicular to each other. If the above conditions are met, the shape can be considered as a square.

The basic logic is to take a complex and abstract task and split it into simple and less abstract tasks (sides, angle, length, etc.). Deeping learning is doing this work to a large extent.

The second example is object detection. The system must recognize the object by the given image.

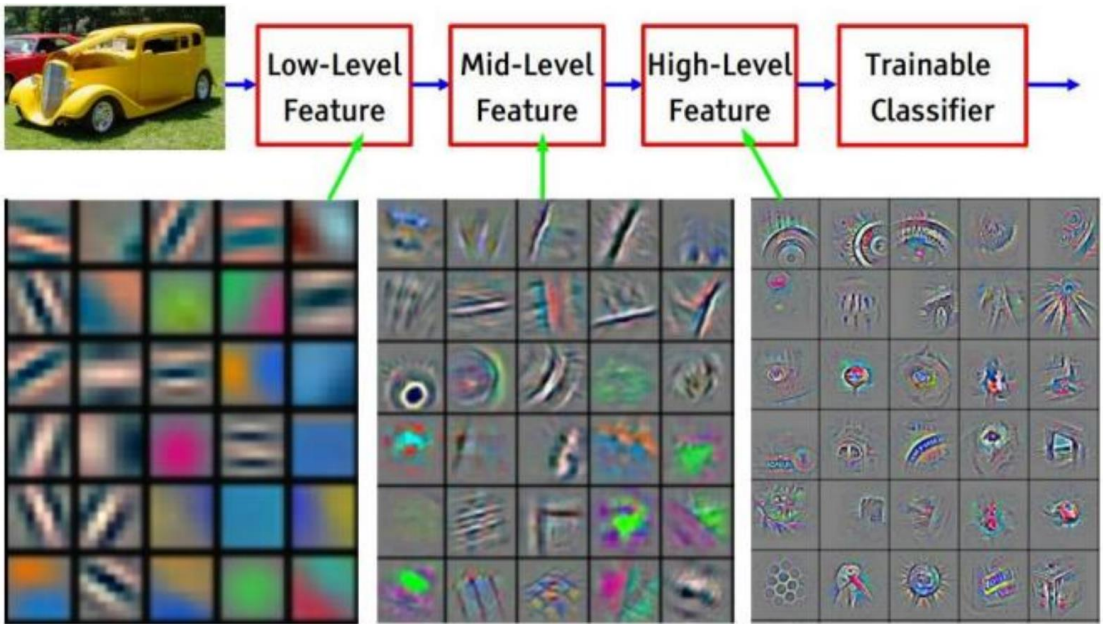


Figure 10. Feature visualization of convolutional net trained on ImageNet. Adapted from 'Visualizing and Understanding Convolutional Networks' by D.Matthew and FergusRob, 2014, New York, NY: Dept. of Computer Science.

For this car recognition example, according to the following steps:

1. First determine which sides and angles have the greatest relationship with identifying cars.
2. Build a hierarchical network based on the many small elements (edges, angles, etc.) found in the previous step, and find out various combinations between them.
3. After constructing the hierarchical network, it is possible to determine which combinations can identify cars.

The hierarchical network shown above has four layers. The input is called raw data, which cannot be directly understood by the machine. Therefore, deep learning first finds all kinds of edges related to this car as much as possible. These edges are the low-level features, which is the first step written above. And the next step is to combine these low-level features (Mid-level features), which is the second step written above. The next step is to combine these low-level features. The last step is to combine all the mid-level features and that is called high-level features. The machine will learn how to identify a car by these abstract patterns, and this is called feature engineering.

2.3 Difference between machine learning and deep learning

The two sections above describe a general understanding of the working principles of machine learning and deep learning. This section compares deep learning and machine learning by examining their key differences across several important aspects.

2.3.1 Data dependency

As the amount of data increases, the performance of the two is very different:

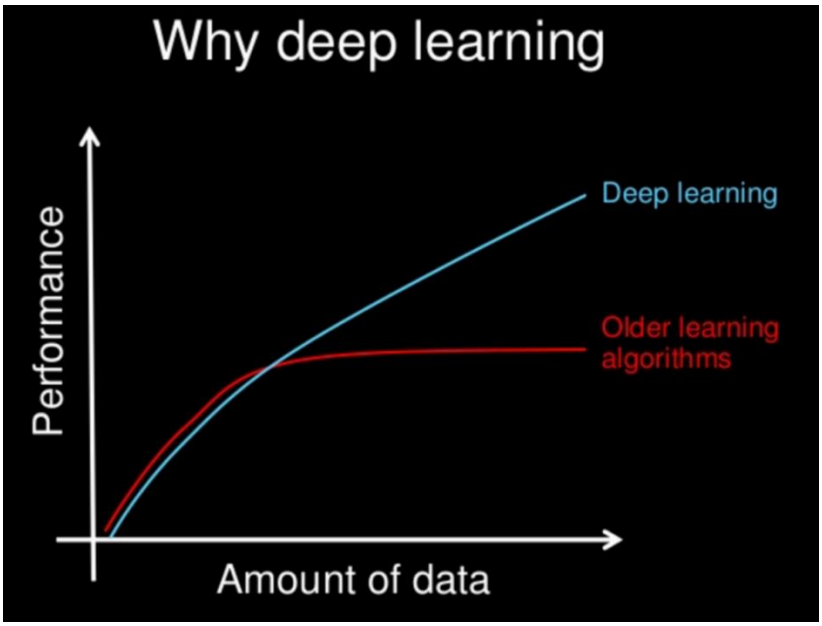


Figure 11. Performance/Data of DL and ML. Reprinted from Analytics Vidhya.

When the amount of data increases, the performance of deep learning also increases while the performance of traditional learning algorithms tends to be saturated (Garg, 2022). Therefore, when the database is small, using an older learning algorithm may be more appropriate.

2.3.2 Hardware dependency

Deep learning algorithms heavily depend on high-end hardware facilities, because the amount of calculation is too large. High-end hardware refers to computer components that are faster or better than what is typically purchased. High-end hardware is usually the most expensive and technically sophisticated.

Deep learning algorithm involves a lot of matrix multiplication operations. Therefore, most of the deep learning algorithms require GPU to participate in the operation, because the GPU is specially designed for matrix operations. On the contrary, traditional machine learning algorithms can work on low-end machines.

2.3.3 Feature engineering

Feature engineering is mentioned in the previous section (2.2). When training a model, the machine needs to know which features are there.

In machine learning, almost all features need to be identified by industry experts, therefore the features are manually encoded.

While deep learning algorithms try to learn high-level features from data. This is a very distinctive part of machine learning. Therefore, feature engineering is a tedious and labour-intensive task. The emergence of deep learning has greatly reduced the cost of discovering features.

2.3.4 Problem-Solving approach

When solving a problem, traditional machine learning algorithms usually divide the problem into several pieces, solve them one by one, and then reassemble them. But deep learning is an end-to-end solution. It allows all the different parts of a process to be trained simultaneously instead of sequentially. This allows for the transformation of raw input data into a desired output without the need for intermediate processing steps. For instance, the following is an example of object detection.

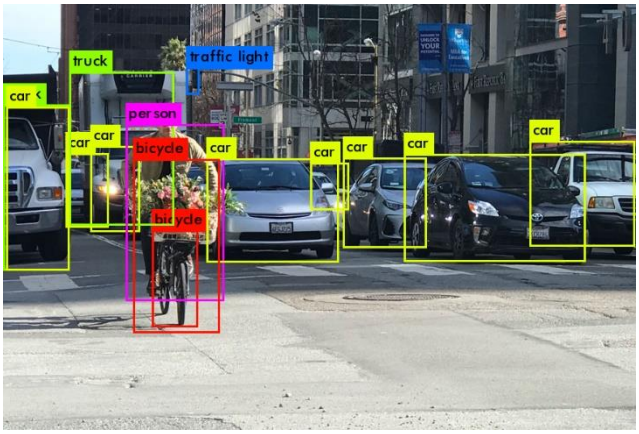


Figure 12. Overview of object detection. Reprinted from GagsHub.

The task is to identify which objects are in a picture and point where they are. A typical machine-learning approach divides the problem into two steps, object detection, and object recognition. Firstly, the machine uses a bounding box detection algorithm like a grab cut, to skim through the image and find all the possible objects. Secondly, an object recognition algorithm such as support vector machine (SVM) with histogram of oriented gradients (HOG) to recognize relevant objects.

While in deep learning, it will directly identify the corresponding object and at the same time mark the name of the corresponding object when given an image. For example, the You Only Look Once (YOLO) net can do real-time identification in a video.

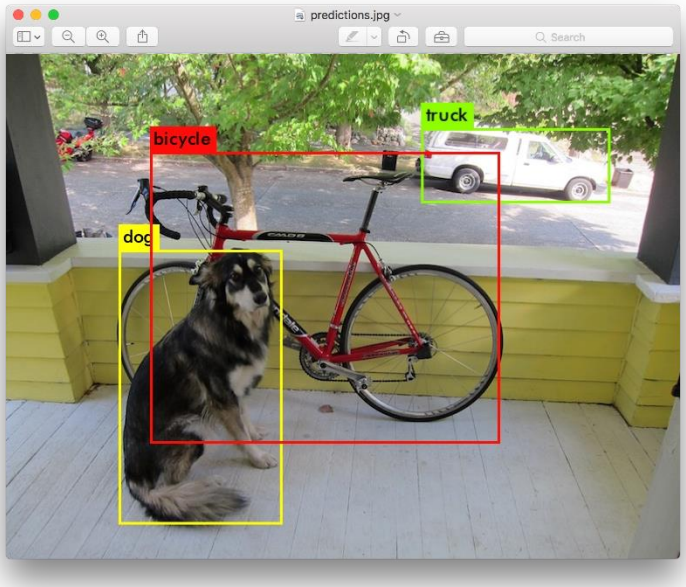


Figure 13. A prediction of YOLO net. (Redmon, 2023)

2.3.5 Execution time

Deep learning takes a lot of time in training part because there are too many parameters needs to be learnt. Deeping learning algorithms like ResNet usually takes weeks for training, under normal circumstances, the longer the training time, the higher the performance of the model, but as the training time increases, the convergence speed of the loss will be greatly slowed down and it will take exponentially longer time to obtain higher performance. (Training time of deep learning models can vary depending on several factors such as the hardware used, the size of the dataset, and the specific architecture of the model)

But machine learning algorithm can generally be trained in a few seconds at most a few hours. Although deep learning takes much longer time than machine learning. The advantage of deep learning is that once the model is trained, it runs very fast on the prediction task.

2.3.6 Interpretability

Last but not the least, deep learning is impossible to understand. In a deep learning network, each layer represents a feature, and if there are too many layers, it's not possible to know what features they represent at all, and the trained model can't be used to explain the prediction task. For example, assume someone uses deep learning to give automated scoring to essays. The performance it gives in scoring is quite excellent and is near human performance. But it can't give the reason why it has given that score. That is because the model is too complicated, and the internal rules are difficult to understand.

On the other hand, machine learning algorithms like decision trees give clear rules as to why it chose what it chose so that it is particularly easy for researchers to interpret the reasoning behind it.

2.4 Machine Learning framework

There are many machine learning frameworks in the market, some of the most popular frameworks include:

1. TensorFlow: An end-to-end open-source machine learning platform developed by Google with a comprehensive and flexible ecosystem that can help researchers advance advanced machine learning technologies.
2. PyTorch: A deep learning framework released by the Facebook team in January 2017, which is currently more popular than Theano, Caffe, MXNet and other frameworks on GitHub.
3. Keras: A high-level neural network API that can run on top of TensorFlow, CNTK or Theano.
4. Caffe: A deep learning framework developed by the University of California, Berkeley for tasks such as image classification, segmentation, and object detection.
5. MXNet: A deep learning framework developed by Amazon that supports multiple programming languages and platforms.

The following table indicates the advantages and disadvantages of these frameworks.

Table 1.Advantages and disadvantages for machine learning frameworks in the market

Framework	Advantages	Disadvantages
TensorFlow	Flexibility and scalability, large community of developers	Steep learning curve
PyTorch	Ease of use, dynamic computational graph, strong community of developers	Limited pre-built models
Keras	Simplicity and ease of use, large number of pre-built models	Limited flexibility
Caffe	Speed and efficiency, large number of pre-built models	Limited flexibility
MXNet	Scalability and flexibility, large community of developers	Steep learning curve

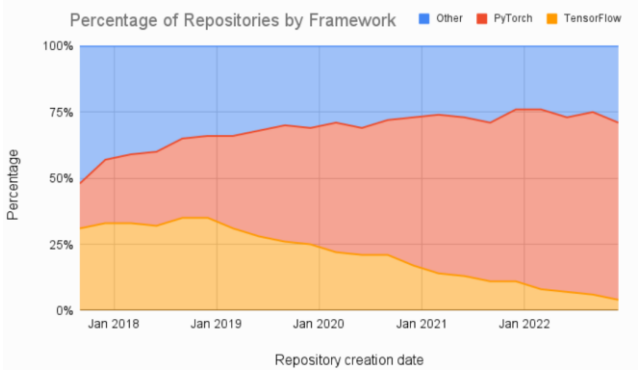


Figure 14. Percentage of Repository by Framework of 2023. (O., 2023)

According to the conducted research, TensorFlow is widely recognized as a framework extensively employed in the industry, while PyTorch primarily emphasizes its utility in research-oriented settings. Considering ModalAi's official endorsement of the tflite file as the recommended object detection model format, TensorFlow has been selected as the preferred framework for deployment in this project.

In Figure 17, the diagram presents the distribution of repositories categorized by framework usage in the year 2023. Notably, approximately 70% of the analyzed papers opted for PyTorch as their framework of choice. This statistical representation underscores the advantageous nature of utilizing PyTorch, particularly when it comes to making modifications to the models employed in this project. Therefore, PyTorch has been selected as the preferred framework for model building in this project. Figure 18 shows the decision tree, the machine learning model trained by PyTorch can be converted to TensorFlow using ONNX.

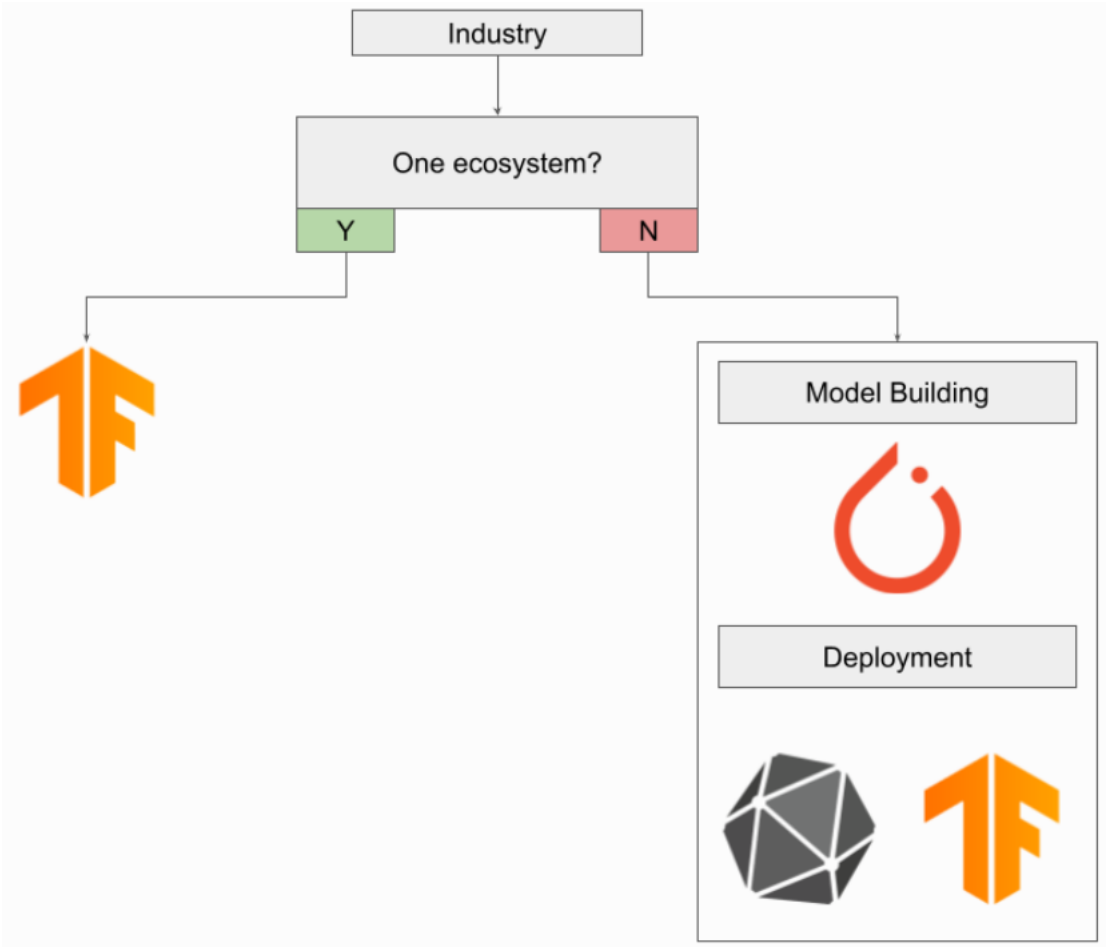


Figure 15. Decision Tree of Machine Learning Project in Industry

2.5 Related research results

2.5.1 Rail failure risk assessment

There is similar research which is about detecting the failure risk of rails using a deep learning technology. The rail risk assessment involves detecting the rail defects that can potentially result in rail break and derailment in extreme cases (Jamshidi, et al., 2017). The following figure is the flowchart of the proposed methodology.

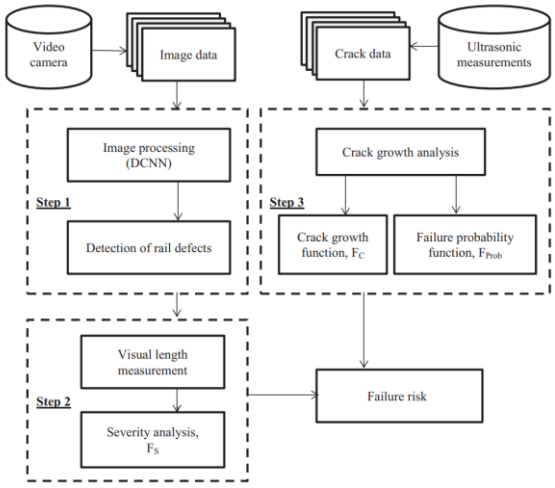


Figure 16. Flowchart of the proposed methodology. (Jamshidi, et al., 2017)

As is shown in this flowchart, the data is generated by a video camera. After gathering the database it's processed by DCNN[8] to get a trained model. Then step two is to implement the proposed framework and make the analysis for failure risk.

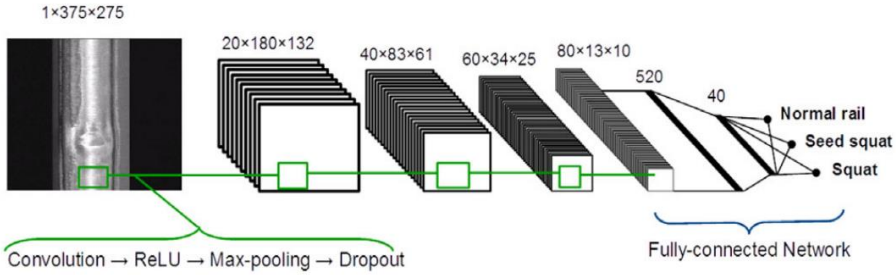


Figure 17. The architecture of the proposed DCNN model. (Jamshidi, et al., 2017)

In this structure, the input is one image which has a size of 375x275. It can be stored as a list like [b, h, w, c], and processing this input through 4 convolution layers and 2 fully connected networks it comes out with three outputs, corresponding to three categories: normal rail, seed squat, and squat.

B: Batch, in this case, the number is 1.

H: Height, it's the height of the image.

W: Width, it's the height of the image.

C: Channel, in this case, c can be ignored because it's a greyscale image and only has one dimension of colour.

In conclusion, by relying on the estimated failure risk, the infrastructure manager can act at the right time and the right place to prevent any unexpected consequences induced by rail breaks.

According to this project and all the sections mentioned above, it is convincing that developing a function of automatic detection and applying it to VOXL 2 is possible.

2.5.2 YOLOv5 improvement

The original Yolov5 model structure is normally not suitable for project requires single object category and with dataset that does not has enough data for train. Zhu in his paper (Zhu, Lyu, Wang, & Zhao, 2021) explains that applying object detection model to a drone-captured scenarios mainly has three problems:

1. The object scale varies significantly because the flight altitude of drone changes greatly.
2. The images captured by drones contain objects with high density, which bring in occlusion between objects.
3. Drone-captured images always contain confusing geographic elements because of covering large area.

The problems mentioned above make this project very challenging. Therefore, it is necessary to adjust the original of Yolov5 to have a better performance.

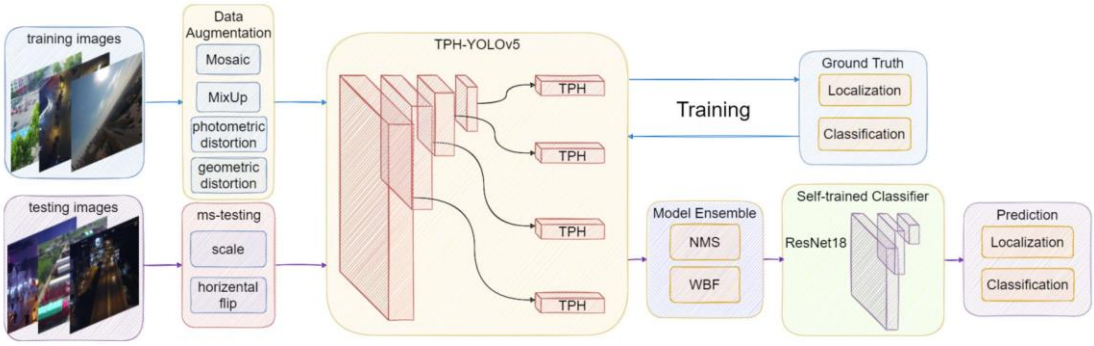


Figure 18. The overview of working pipeline using TPH-YOLOv5. Compared to original version, the authors manly improve the head by applying Transformer Prediction Head (TPH). They also add one more head to better detect different scale objects. In addition, they employ bag of tricks like data augmentation, multi-scale testing, model ensemble and self-trained classifier to make TPH-YOLOv5 stronger (Zhu, Lyu, Wang, & Zhao, 2021).

2.6 Criteria

The quality of the results of machine learning and deep learning can be measured by different criteria depending on the type and goal of the task. The following are criteria for testing the performance of an object detection model (Zhu, Lyu, Wang, & Zhao, 2021).

2.6.1 Accuracy

The accuracy metric determines the proportion of correctly classified values, indicating its correctness rate. It is calculated as the sum of true positive (TP) and true negative (TN) values divided by the total number of values.

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

True positive (TP): It is an outcome where the model correctly predicts the positive class.
True negative (TN): It is an outcome where the model correctly predicts the negative class.
False positive (FP): It is an outcome where the model incorrectly predicts the positive class.
False negative (FN): It is an outcome where the model *incorrectly* predicts the *negative* class.

2.6.2 Precision

Precision is a performance metric that evaluates the ability of a machine learning model to accurately classify positive values. It is calculated as the ratio of true positives to the total number of predicted positive values.

$$Precision = \frac{TP}{TP + FP}$$

2.6.3 Recall

Recall is a metric that evaluates the model's ability to identify positive values correctly. It is calculated as the true positives divided by the total number of actual positive values.

$$Recall = \frac{TP}{TP + FN}$$

2.6.4 F1-score

The F1 score is defined as the harmonic mean of Recall and Precision, and it is commonly used to evaluate the overall performance of a binary classification model, especially when both Precision and Recall are considered.

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

2.6.5 mAP (mean Average Precision)

mAP is a very popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. If the model only has one class, then it could be considered as the same as F1-score.

2.7 Benefit-cost analysis

This section discusses which machine learning algorithm (DL of ML) is the most beneficial. According to the analysis in the previous sections, deep learning can be an effective technique for defect detection. Deep learning models for object detection such as YOLO, Faster R-CNN and SSD are quite popular in the market. Based on a paper (Zhao, Zhao, Xu, & Wu, 2019), deep learning methods have significantly outperformed machine learning methods on several object detection benchmarks, such as PASCAL VOC, MS COCO, and ImageNet. For example, on the PASCAL VOC 2012 test set, the best machine learning method achieved a mean average precision (mAP) of 53.3%, while the best deep learning method achieved a mAP of 86.6%. On the MS COCO 2017 test-dev set, the best machine learning method achieved a mAP of 25.1%, while the best deep learning method achieved a mAP of 48.4%. On the ImageNet 2015 test set, the best machine learning method achieved a mAP of 33.3%, while the best deep learning method achieved a mAP of 56.4%. These results show that deep learning methods can achieve much higher accuracy and robustness than machine learning methods for object detection.

MODALAI itself also has a development tool kit (SDK) for deep learning models which is friendly for the developer to customize their model. While if using the traditional machine learning algorithms, considering that the VOXL 2 doesn't support the traditional machine learning algorithms natively. The developer needs to design an application programming

interface (API) for it, this poses a significant challenge for the developer of this project. If using deep learning technology, it's also maintenance friendly for the company in the future to improve this function.

Considering all these advantages and disadvantages, deep learning would be the one to realize the defect detection deep learning model.

3 Method

The Vee model method is selected for this project. The Vee model is a widely recognized and accepted approach in the engineering field that consists of four main phases: System Design and System test, Sub-system Design and Sub-system test, Component Design and Component test. Each phase has well-defined objectives, activities, and deliverables that will ensure the successful implementation of the project (Blanchard & Fabrycky, 2011). The utilization of the Vee model will provide a systematic approach to the development process, facilitating the management of project risks, reducing uncertainties, and ensuring that the final product meets the specified requirements (Veenvliet, Broenink, & Bonnema, 2015).

3.1 Reason for applying Vee model

The Vee model is particularly well-suited for software design and testing. Its logical structure promotes thorough test coverage, and its step-by-step breakdown of the system into smaller components minimizes the risk of neglecting essential elements. As the focus of this project lies in the development of a machine learning algorithm for a drone-based monitoring system, the Vee model is apt for ensuring effective and efficient attainment of the objectives.

3.2 Activities and deliverables

Based on the analysis of previous problems, the deliverables can be summarized as follows.

1. Training dataset for corrosion detection
2. Modified YOLOv5 model architecture
3. Model performance validation results
4. Object detection package

The focus of this phase lies in the design of the system at various levels and the subsequent formulation of corresponding test plans for their execution, in accordance with the framework of the Vee model. All sub-questions can be solved upon completion of the tasks within this phase. The V model encompasses two key phases: Design and integration.

3.2.1 Design and Test plan

The primary objective of the Design and Test Plan phase is to provide a detailed description of the system and develop corresponding test plans.

3.2.1.1 System design

In the system design phase, the comprehensive design of the system will be presented in the form of a system description, closely resembling the analysis of the existing situation. Following the completion of the system description, the system test plan will be developed and delivered. This phase marks the resolution of the initial sub-question, setting the foundation for further exploration and development.

3.2.1.2 Sub-system design

To thoroughly analyse the system from diverse functional perspectives, a strategic approach involves dividing the entire system into multiple subsystems. Each of these subsystems is accompanied by a detailed description, providing insights into its specific characteristics and functionalities. Additionally, these subsystems are linked to their respective experiment plans, outlining the necessary steps to conduct experiments and gather valuable data. This phase of the analysis process aims to address the second sub-question, contributing to a comprehensive understanding of the system's complexity and dynamics.

3.2.1.3 Component design

Within a data analysis project, the focus lies on the components that encompass the programs employed to realize the specific functions related to data analysis. During the process of component design, an important aspect involves the utilization of digital twin technology to establish a mapping or connection between the desired functionalities and the corresponding code implemented within the program. Consequently, this phase of the project aims to solve the third sub-question, thus contributing to the overall advancement of the data analysis effort.

3.2.2 Integration and Test executing

This phase focuses on following the predefined test plans established during the preceding phase. Its primary objective is to combine the components or subsystems that successfully pass the tests, ensuring their seamless integration. By undertaking this step, the integration and test executing phase aims to determine the feasibility of the designed components, subsystems, or the entire system, addressing the relevant sub-question in the process. Consequently, upon completion of this phase, fourth sub-question can be solved.

3.2.2.1 Component build phase

During this phase, the construction of the dataset and training program will be made. In this phase, the objective is to test each version of dataset and training program that achieves assigned tasks based on the test plan. The test results will be recorded, and the most suitable dataset and training program are selected for further use.

3.2.2.2 Subsystem integration

The selected modules from the previous phase will be integrated into the subsystem, considering the analysis aspects of each metrics. The test plan for the subsystem in the design phase will be employed to assess if the assembled subsystem meets the design requirements and make necessary optimizations.

3.2.2.3 System integration

In a manner analogous to the subsystem integration phase, the individual subsystems are assembled into a comprehensive system, followed by testing and comparison with the expected outcome. Ultimately, the final delivery is attained through essential system

optimization. The integration of the entire system can be viewed as the evaluation of the object detection model.

4 Result

This chapter systematically describes the results of this project based on the guidance of the Vee model. It will follow the order under system design, sub-system design, component design, components integration, sub-system integration and system design integration to present the outcomes of this project.

4.1 List of requirements

The list of requirements is listed as follows to give the reader a general idea of what this project is going to accomplish.

Table 2. List of requirements

No.	Domain	Requirements
1	Software	The model should have the ability to frame out corrosions and defects.
2		The performance of the deep learning model shall meet the minimum standard of criteria.
3		The repository of this project should be possible for future personnel to continue to develop.
4	Software & Hardware	Configure the VOXL 2 to make the model compatible with the hardware.
5		The result shall run smoothly on the VOXL 2.
6		An instruction file shall be made for both compiling and installing the package.

As is shown in this table, this project mainly focuses on developing an object detection task algorithm. To achieve this, the requirements will be converted to several tasks to make it clearer for the author to develop. The following diagram gives a general idea about what tasks and knowledge are needed in this project.

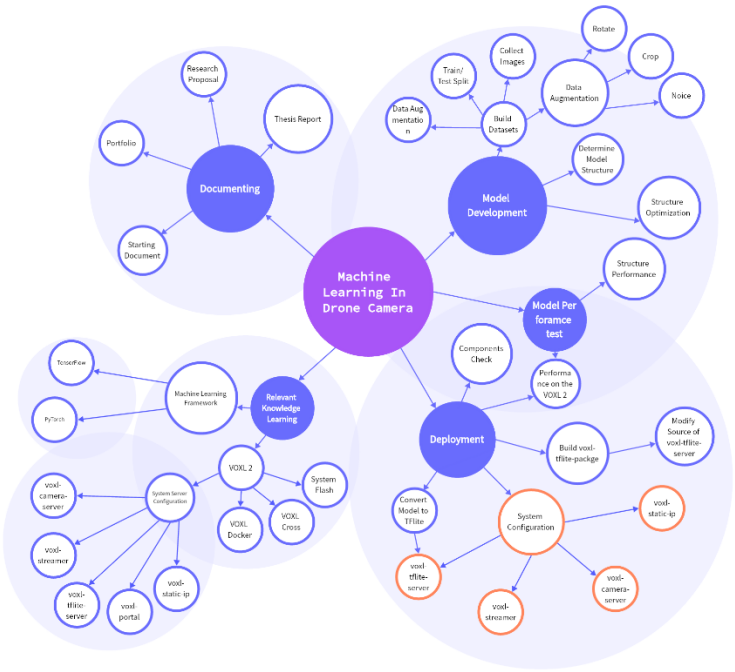


Figure 19. Overview of the project process

4.2 System design

The system design is created to analyse and satisfy the client's requirements. In this chapter, it shows the final design and system test plan.

4.2.1 System description

The main objective of this project is to design an object detection model that can identify corrosion in real-time on drone-captured images. The system is intended to be deployed on the VOXL 2 platform, enabling drones to autonomously detect corrosion and mark its location on the surface under inspection.

The system consists of four subsystems: Dataset Construction, Model Training and Optimization, Model Deployment, and Model Usage (Detection Phase).

A secondary objective of the project is to create a repository or a toolkit for the company. This toolkit will contain all the resources and documentation necessary to allow the company to replicate the entire process of model creation and deployment. This will include resources for data collection and annotation, scripts for model training and evaluation, code for model deployment on the VOXL 2, and guides on how to use the model for real-time corrosion detection. Appendix 1.1 indicates the system description in detail.

4.2.2 System test plan

The system test plan aims to validate whether the system meets the requirements. It focuses on the coherence of each subsystem and the feasibility for future maintenance. Appendix 1.2 indicates the system test plan in detail.

4.3 Sub-system design and components design

This chapter shows the sub-system and components design and the test plans. In this section it will show the solution of how these sub-questions are solved. There are four subsystems involved in this project: Data Collection and Preparation, Model Training and Optimization, Model Deployment, Toolkit Development. Each of these subsystems can be treated as individual components. In this project, components refer to different building block or parts of the system that can be developed and tested independently. These subsystems are interconnected, and each one contributes to the overall success of this project.

By breaking down the project into these subsystems, the author can focus on testing and evaluating each component separately. This allows the author to identify potential issues and ensure the quality and performance of each subsystem before integrating them into the complete system. The following shows the workflow of sub-systems.

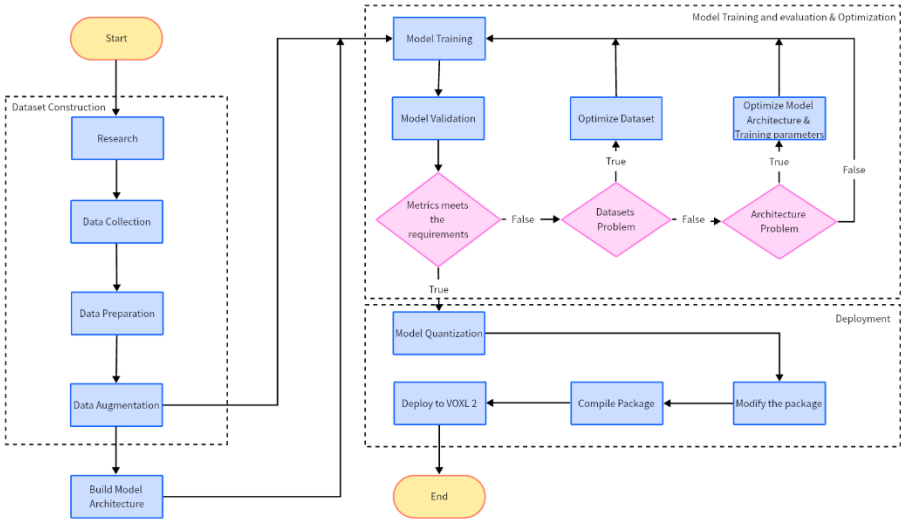


Figure 20. Subsystems workflow

4.3.1 Data Collection and Preparation

4.3.1.1 Data Collection and Preparation Subsystem Description

The aim of this subsystem is to establish a robust and representative dataset that the object detection model can learn from.

The subsystem involves the collection and preparation of the image dataset used for training the object detection model. The process starts with selecting images of vapor pipes with visible corrosion from the company's database. These images are then resized to a standard size of 640x640 pixels using an image conversion tool. The resized images undergo annotation using an online annotation tool, highlighting regions of corrosion in each image. The dataset is then expanded via data augmentation techniques (rotation, cropping, and noise addition), yielding a total of 2.7k images for model training. Appendix 2.1 indicates the sub-system description in detail.

4.3.1.2 Data Collection and Preparation Subsystem test plan

The subsystem test plan aims to confirm the integrity and readiness of the dataset for the subsequent stages of model training and optimization. The test ensures the images are properly formatted, accurately annotated, and the dataset is sufficiently diverse through the applied data augmentation techniques. For further details, please refer to Appendix 2.2 for a comprehensive description of the subsystem test plan.

4.3.2 Model Training and Optimization

4.3.2.1 Model Training and Optimization Subsystem Description

This subsystem revolves around model training and optimization. This process uses an improved YOLOv5 model architecture, with added Convolutional Block Attention Module (CBAM) to enhance its ability to capture fine-grained details related to corrosion. The model is trained on the pre-processed dataset that mentioned in the previous subsystem, and its performance is optimized through iterative training and hyperparameter tuning. Appendix 3.1 indicates the subsystem description in detail.

4.3.2.2 Model Training and Optimization Subsystem test plan

The subsystem test plan for model training and optimization aims to verify the functionality and performance of the trained model. It checks the model's operational readiness, evaluates performance metrics against project requirements, confirms effective hyperparameter optimization, and validates successful model training convergence. The success of this subsystem is assessed by the improved performance of the model in detecting corrosion instances accurately. The model should have a high precision-recall curve and an increased mAP, chapter 2.6 indicates the details of these metrics. For a comprehensive understanding of the test plan, please refer to Appendix 3.2 in the document.

4.3.3 Model Deployment Subsystem Description

This subsystem pertains to model deployment. In this phase, the trained model is integrated into the VOXL 2 device. This is accomplished by modifying the SDK (voxl-tflite-server) provided by ModalAI, building the environment in the voxl-cross docker image, and creating a .deb package that is then deployed onto the VOXL 2 via an ADB command. Appendix 4.1 indicates the subsystem description in detail.

4.3.4 Model Deployment Subsystem test plan

The subsystem test plan for model deployment aims to verify the successful deployment of the object detection model onto the VOXL 2 device. It involves confirming the installation of the .deb package, checking the real-time operation of the model on the device, and ensuring its performance meets project requirements. The subsystem can be considered as successful if the package can be installed and run correctly in the VOXL 2 platform after configuration. For an in-depth understanding of the test plan, refer to the relevant Appendix 4.2 in the document.

4.3.5 Toolkit Development Subsystem description

This subsystem revolves around toolkit development. This phase encompasses modification of the YOLOv5 model, addition of the attention module, architecture adjustments, and format conversion adaptations for further quantization. A Jupyter notebook is created for process simplification, and a comprehensive instruction manual (README file) is developed to facilitate the self-training of customized models for users. Appendix 5.1 indicates the subsystem description in detail.

4.3.6 Toolkit Development Subsystem test plan

The subsystem test plan for toolkit development aims to verify the successful modification of the YOLOv5 model, its correct conversion, and the functionality of the Jupyter notebook, along with the comprehensiveness and user-friendliness of the instruction manual. The success of this subsystem compiles to the requirements which hinge on the creation of a user-friendly toolkit that facilitates the training and application of the model for corrosion detection. This test ensures the toolkit is accessible and effective in enabling users to train their own models. For a detailed understanding of the test plan, refer to the relevant Appendix 5.2 in the document.

4.4 System integration

System integration in the context of this project involves ensuring that each subsystem interacts coherently with others to create a unified, functioning system that can effectively detect corrosion from drone-captured images and provide a toolkit for future model development.

1. **Integration of Dataset Construction and Model Training & Optimization:** The Dataset Construction subsystem, responsible for creating an augmented dataset for model training, works seamlessly with the Model Training & Optimization subsystem. The augmented dataset is used to train and optimize the YOLOv5 model, which incorporates an attention module to enhance the model's detection accuracy.

Figure 21. The dataset for this project is saved on the company's shared drives.

2. **Integration of Model Training & Optimization and Model Deployment:** After the model is trained and optimized, it enters the Model Deployment subsystem. The trained model is converted to TensorFlow Lite (tflite) format and packed into a .deb package for deployment on the VOXL 2 device. These two subsystems need to work in harmony to ensure the trained model can be effectively deployed and function in real-time on the device.
3. **Integration of Toolkit Development with Dataset Construction, Model Training & Optimization, and Model Deployment:** The Toolkit Development subsystem integrates with all other subsystems. It simplifies the model training, conversion processes, and provides instructions for users to train their own models, ensuring that the benefits of the project can continue even after its conclusion.

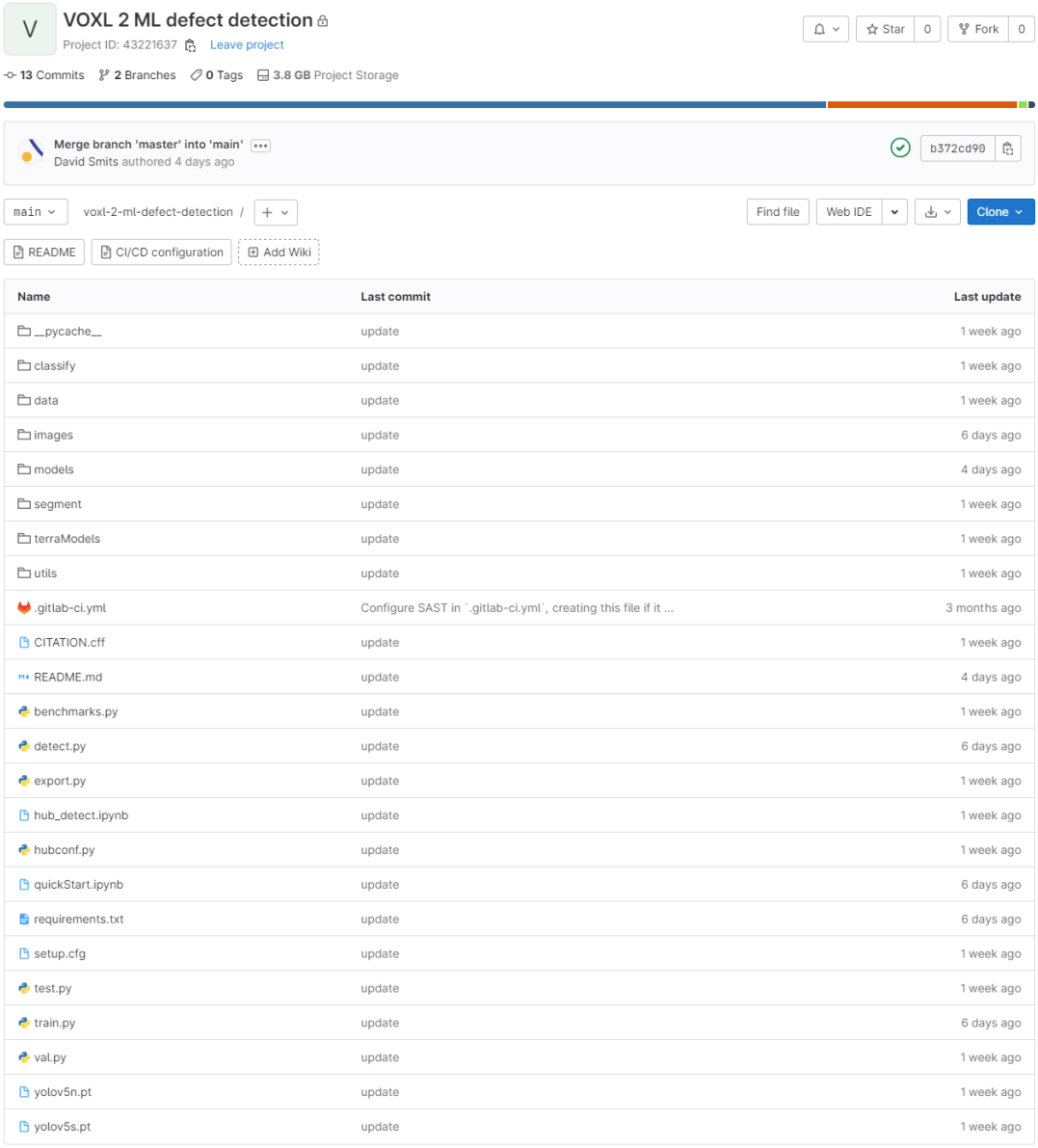


Figure 22. The Develop Toolkit is saved in the company's GitLab Repository

4. **Overall System Integration:** Once all subsystems are integrated, the whole system should operate cohesively. The end system will provide real-time object detection capabilities on the VOXL 2 device and empower users to develop their own object detection models.

4.5 Test results

This chapter delves into the results obtained from testing the subsystems and the overall system to evaluate their performance and ensure that they meet the established project objectives.

4.5.1 Sub-system and components test results

All sub-systems are tested according to the test plan.

4.5.1.1 Data collection and preparation test result

In testing the dataset construction subsystem, the generated dataset contains the expected number of augmented images. Each image maintains a resolution of 640x640 pixels, preserving the corrosion details. The annotations are precise, and the image transformations, such as rotation, cropping, and noise addition, are successfully implemented, as it fulfils the requirements that emphasizes a diverse and comprehensive dataset for effective model training. Therefore, the dataset construction subsystem is validated as functioning as intended. Appendix 6.1 indicates a detailed result.

4.5.1.2 Model Training and Optimization Subsystem test result

The subsystem test confirms that the modified YOLOv5 model, integrated with the CBAM module, has trained successfully on the augmented dataset. The model demonstrates an improved detection accuracy during training, suggesting the effective functioning of the attention module. The optimization process further enhanced the model's performance. A comparative evaluation with the base YOLOv5 model indicates an improvement in the corrosion detection ability of the modified model. Hence, this subsystem is validated to have effectively fulfilled its purpose of training and optimizing the model to detect corrosion. Details of the tests are presented in Appendix 6.2.

4.5.1.3 Model Deployment Subsystem test result

This subsystem is validated through a series of tests. Successful SDK modifications are carried out, as it complies the requirement that it can build the package correctly and the VOXL 2 can run this program without any problem, followed by successful build environment setup and dependency installation inside the docker. The build scripts execute correctly, and a .deb package containing the model in the required tflite format is created. This package is successfully deployed on the VOXL 2. In real-time testing, the model demonstrated the ability to process drone-captured images and detect corrosion instances. Hence, the Model Deployment subsystem is verified to have achieved its goal of deploying the model to a real-world device for real-time operations. Details of the tests are provided in Appendix 6.3.

4.5.1.4 Toolkit Development Subsystem test result

This subsystem has been successfully validated through rigorous testing. The YOLOv5 model modifications are carried out without errors, as is the adaptation for PyTorch to TensorFlow format conversion. The created Jupyter notebook executes correctly, streamlining the model training process. The readme file provides clear and comprehensive instructions, aiding in

user understanding. The test confirms the effectiveness and user-friendliness of the toolkit, as it includes a user-friendly interface and a detailed instruction guide and facilitates the training and application of the model. Thus, this subsystem has proven its capability to facilitate efficient training of customized models for corrosion detection. The detailed test results are elaborated in Appendix 6.4.

4.5.1.5 System test result

In the system test, the integrated system successfully processed the drone-captured images and accurately detected instances of corrosion. The performance metrics of the system meet the requirements, demonstrating improved mAP.

Moreover, the system functioned seamlessly in real-time, processing drone-captured images and accurately detecting and analysing corrosion. The integration of subsystems and their collective performance are up to the expectations, fulfilling all the requirements.

Thus, the system test is successful because it demonstrates that the integrated system meets all the requirements and aligns with requirements. The detection and analysis of corrosion instances in drone-captured images show the practical applicability of the system, highlighting its potential for real-world deployment in drone-based systems. The detailed system test result is elaborated in Appendix 6.5.

5 Conclusion and recommendations

This project successfully achieves its objectives of developing an object detection model to detect corrosions and deploying it to a flight controller, VOXL 2, for real-time processing of drone-captured images. By modifying the YOLOv5 model, adding a Convolutional Block Attention Module (CBAM), and building a comprehensive dataset, it improves the model's ability to accurately detect corrosion instances especially for drone-captured images.

Further, the development of a comprehensive toolkit has ensured the company's ability to create and train their own models, thus contributing to the sustainability and scalability of the project. A comparative analysis with the original YOLOv5 model clearly shows the effectiveness of the modifications, leading to an improved performance of the model.

The project has taken a significant step in the field of corrosion detection methodologies, particularly when integrated with drone-based surveillance systems. The improved precision and adaptability of the detection model promise numerous opportunities for efficient and proactive corrosion management.

5.1 Discussion

The study carried out in this project provides a comprehensive analysis of the application of a modified YOLOv5 model for corrosion detection in drone-based monitoring systems. The results demonstrate the effectiveness of the modifications, highlighting the model's improved precision and mAP when compared to the original YOLOv5 architecture.

The project highlights how deep learning models, when fine-tuned with domain-specific augmentations, can offer robust performance in specific applications such as corrosion detection. It further underlines the potential of integrating such AI-driven techniques into existing drone-based systems, enabling enhanced detection.

However, it is important to contextualize the findings within the scope and limitations of the project. The training and evaluation of the model are based on a specific dataset, which could potentially affect its generalization to unseen instances of corrosion in different environmental conditions or varying severity levels. Hence, while the performance metrics are promising, they might not fully represent the model's capability to handle real-world complexity and diversity.

What’s more, the project focuses on drone-captured images for corrosion detection, and the model's performance in varied real-world scenarios, such as different lighting conditions, angles, and distances, is not extensively evaluated. Therefore, future implementations should consider these factors to ensure the model's reliability and effectiveness in practical applications.

Finally, although the modified YOLOv5 model demonstrates a significant improvement in corrosion detection, its true potential could be better harnessed when integrated with other decision-making systems, such as automated reporting, maintenance scheduling, and risk assessment frameworks. This would maximize the utility of the AI-driven detection system.

In essence, the project results are encouraging, opening new avenues for leveraging AI in the field of corrosion detection. However, further research addressing the limitations and

exploring opportunities in diverse real-world conditions and integrative systems would be crucial to fully realize the potential of the modified YOLOv5 model in corrosion detection.

5.2 Future Directions

Future directions for this project include addressing these limitations, expanding the applicability of the dataset, and conducting comprehensive field trials. The potential integration of the corrosion detection algorithm with other monitoring systems and maintenance management frameworks should also be explored.

References

Blanchard, B. S., & Fabrycky, W. J. (2011). *Systems Engineering and Analysis.* Upper Saddle River, New Jersey: Prentice Hall.

Garg, A. (2022, November 7). *Advance Guide on Interview Questions of Deep Learning*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2022/11/advance-guide-on-interview-questions-of-deep-learning/?utm_source=related_WP

glenn-jocher, AyushExel, Borda, alexstoken, & NanoCode012. (2023). *ultralytics/yolov5*. Retrieved from GitHub: https://github.com/ultralytics/yolov5

Gupta, A. (2022, February 24). *Deep Learning vs Machine Learning for Regression*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2022/02/deep-learning-vs-machine-learning-for-regression/?utm_source=related_WP

Jamshidi, A., Fafhih-Roohi, S., Hajizadeh, S., Nunez, A., Babuska, R., Dollevoet, R., . . . Schutter, B. D. (2017). *A Big Data Analysis Approach for Rail Failure Risk Assessment.*

ModalAI. (2023). *ModalAI Technical Documentation*. Retrieved from ModalAI: https://docs.modalai.com/voxl-flight-deck-getting-started/

ModalAI. (2023). *Unleashing 5G with Qualcomm Flight RB5 5G Platform*. Retrieved from ModalAI: https://www.modalai.com/pages/qualcomm-flight-rb5-5g-platform

ModalAI. (2023). *VOXL® 2*. Retrieved from ModalAI: https://www.modalai.com/products/voxl-2?variant=39914779836467

ModalAI. (2023). *VOXL® 2 - Figure 1. Overview of VOXL 2*. Retrieved from ModalAI: https://www.modalai.com/pages/voxl-2

O., R. (2023). *AssemblyAI-Examples / pytorch-vs-tensorflow*. Retrieved from GitHub: https://github.com/AssemblyAI-Examples/pytorch-vs-tensorflow/tree/main/2023

Redmon, J. C. (2023). *YOLO: Real-Time Object Detection*. Retrieved from pijreddie: https://pjreddie.com/darknet/yolo

S, W., J, P., & Y, L. J. (2018). *CBAM: Convolutional Block Attention Module.* European Conference on Computer Vision (EVCC).

Shaikh, J. (2021, January 4). *Deep Learning vs. Machine Learning - the essential differences you need to know!* Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/

Veenvliet, K. T., Broenink, J. F., & Bonnema, G. M. (2015). *SYSTEMS DESIGN AND ENGINEERING.* CRC Press.

Woo, S., Park, J., Lee, J.-Y., & Kweon, I. S. (2018). *CBAM: Convolutional Block Attention Module.* Springer. doi:10.1007/978-3-030-01270-0_48

Zaidi, S. S., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M., & Lee, B. (2022). *A survey of modern deep learning based object detection models.* Elsevier. doi:10.1016/j.jclepro.2022.131312

Zeiler, M. D., & Fergus, R. (2014). *Visualizing and Understanding Convolutional Networks.* New York: Dept. of Computer Science.

Zhao, Z., Zhao, P., Xu, S., & Wu, X. (2019). *Object Detection With Deep Learning: A Review.* IEEE. doi:10.1109/TNNLS.2018.2876865

Zhu, X., Lyu, S., Wang, X., & Zhao, Q. (2021). *TPH-YOLOv5: improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Secenarios.* Beijing: Beihang University. doi:10.7527/S1000-6893.2022.27106

Appendix 1 System Design

Appendix 1.1 System description

System overview

The purpose of this project is to design and deploy an object detection model capable of accurately detecting corrosions. The system is specifically developed to be deployed on the VOXL 2, which serves as a flight controller. The VOXL 2 plays a crucial role in processing real-time images captured by drones and identifying the presence of corrosions within these images. The system encompasses two main objectives:

1. Real-time Corrosion Detection: The primary objective of the system is to achieve real-time and precise detection of corrosions in the drone-captured images. By leveraging a customized variant of the YOLOv5 object detection model, the system effectively identifies and localizes corrosions, enabling prompt inspection and timely remediation efforts.

2. Repository for Model Development: In addition to the object detection model, this project aims to establish a comprehensive repository or tool that empowers the company to independently develop their own models in the future. This repository/tool serves as a foundational resource, allowing the company to create and train object detection models tailored to their specific requirements and domain expertise, even after the departure of the project developer.

By accomplishing these objectives, this project significantly enhances the company's capabilities in efficient and reliable corrosion detection utilizing the VOXL 2. Moreover, the establishment of a repository/tool ensures the sustainability of model development efforts, enabling the company to continue advancing their expertise in the field of object detection even in the absence of the original project developer.

Appendix 1.2 System test plan

I. Aim

The goal of the system test is to validate that the entire system functions as expected and fulfils the predefined requirements. This includes the detection of corrosion in real-time on drone-captured images with satisfactory accuracy, successfully deploying the model on the VOXL 2 flight controller, ensuring the model works in real-time conditions, and that the company can replicate the model creation and deployment process using the provided toolkit.

II. Hypothesis

If the system and toolkit have been designed and developed correctly, then the system should detect and accurately mark corrosion instances in drone-captured images in real-time when deployed on a VOXL 2 flight controller, and the company should be able to recreate this process using the provided resources.

III. Tools required for the System Test

1. VOXL 2 equipped with a camera and HereLink.
2. Test dataset of drone-captured images including both normal and corrosion target.
3. The deployment environment: voxl-tflite-server SDK and necessary hardware.
4. Performance measurement tools for model evaluation.
5. Log recording tool for system logs during the test.
6. The repository containing all resources and documentation for replicating the model creation and deployment process.

IV. Method

This phase presents a sequential guide outlining all the necessary actions to be carried out during the test to infer an outcome.

1. Preparation: Ensure the drone, VOXL 2, and all other equipment are ready and functioning correctly. Prepare the test dataset that has been kept separate from the training data for unbiased evaluation.

2. Deployment: Deploy the trained model on the VOXL 2 using the adb shell.

3. Real-Time Testing: Startup the process and record the output from the model (corrosion detection and marking) as well as system logs for further analysis.

4. Performance Evaluation: Evaluate the model's performance based on the real-time detection results. Calculate mAP50-95.

5. Functional Testing: Ensure all parts of the system are functioning as intended. This includes the real-time detection, marking of detected areas, handling of image data.

6. Toolkit Testing: Test the toolkit by attempting to recreate the process of model creation and deployment. Verify that all necessary resources and documentation are

present, accurate, and sufficient for someone unfamiliar with the project to recreate the process.

7. Validation: Validate the system test by comparing the actual results with the expected results.

V. Checklist

The checklist serves as an essential tool to ensure that each requirement has been addressed and verified.

1. The voxl-tflite-server is operational on VOXL 2.
2. The model has been correctly deployed on VOXL 2.
3. Real-time detection is functional during operation.
4. The model correctly marks areas of corrosion in real-time.
5. mAP meets predefined performance thresholds.
6. All parts of the system function as intended during drone operation.
7. Toolkit contains all necessary resources and documentation.
8. Toolkit allows for successful replication of the model creation and deployment process.

VI. Expected Results

1. The model should be capable of detecting and marking corrosion targets in real-time with a high degree of accuracy.
2. There should be no functional issues or unexpected behaviors during the system test.
3. The VOXL 2 should successfully handle the model and process drone-captured images in real-time.
4. During the entire operation, the system should function in a stable manner.
5. The toolkit should contain all necessary resources and documentation for replicating the model creation and deployment process.
6. The toolkit should be sufficient for someone unfamiliar with the project to successfully recreate the process.

Appendix 2 Data Collection and Preparation Subsystem Design

Appendix 2.1 Data Collection and preparation subsystem description

This subsystem focuses on gathering and preparing the necessary data for training the object detection model.

The following lists the steps that are normally taken to build a customized dataset:

1. **Image Collection:** The initial step involves retrieving images containing corrosion from the company's database. These images specifically focus on the corrosion in vapor pipes, ensuring the model is trained to identify similar instances.
2. **Image Resizing:** To maintain consistency and ensure compatibility with the object detection model, all collected images are resized to a standard size of 640x640 pixels. This process utilizes an online image conversion tool which allows for bulk resizing, preserving the original aspect ratio and ensuring no critical information is lost during the resizing process.
3. **Image Annotation**: After the images are resized, they are annotated using an online annotation tool. This step is crucial as it provides the "ground truth" labels that the model will use for learning. The tool is used to manually draw bounding boxes around instances of corrosion in each image and label them accordingly. In total, 1079 instances of corrosion are annotated across the different images.
4. **Data Augmentation:** The final step involves augmenting the dataset to increase its size and improve the model's ability to generalize. Techniques such as rotation, cropping, and noise addition are applied to the original images, creating variations that simulate different perspectives, image quality, and potential occlusions that the model might encounter in real-world scenarios. This step effectively expands the dataset to 2.7k images, providing a robust and diverse set of examples for the model to learn from.

The initial stage entails the acquisition of images containing the desired targets from the company's extensive database. Following the completion of this step, a total of 1046 images have been procured. Subsequently, the subsequent stage involves the annotation of the acquired data, which is recognized as the most time-intensive undertaking. Deep learning methodologies necessitate a substantial volume of meticulously annotated data, thereby demanding the annotator's utmost concentration and attentiveness during the annotation process.

The subsequent stage encompasses the application of data augmentation techniques. Augmentation plays a pivotal role in enhancing the model's ability to generalize its performance by augmenting the diversity of learning instances accessible to the model. In this project, the author employs three distinct methods for augmentation: rotation, cropping, and noise addition. Incorporating rotations aids in cultivating the model's insensitivity towards variations in camera orientation. Cropping facilitates the model's robustness against subject translations and deviations in camera positioning. Lastly, the inclusion of noise serves the purpose of fortifying the model's resilience against potential camera-related artifacts.

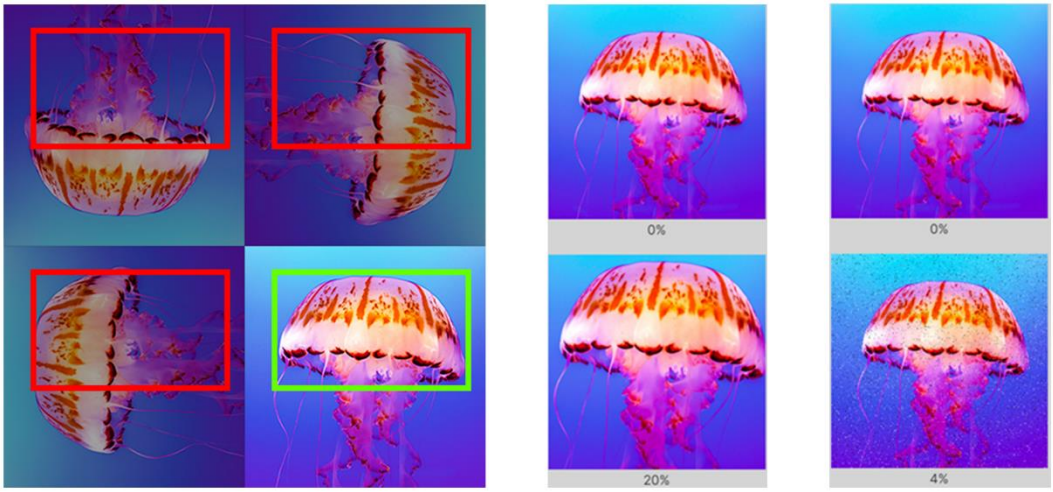


Figure 23. Image Augmentation: Rotate, Crop, and Noise.

After the augmentation, the final dataset has 2.7k images which is almost triple the original size. It is then divided into two sets: training sets and the testing set. The ratio of the training sets and the testing set is 9:1. Testing set is a dataset that would never be involved in training the model, it is used to test the performance of the model after training.

The training sets need to be divided into two sets, the training sets and the validation set. The ratio between the training set and the validation set is 9:1. Training set is used for training; the number of images should be as many as possible. Validation set is used to validate the performance after each training round is complete, the change of the variables is based on the result of the validation in each round.

Appendix 2.2 Data Collection and preparation subsystem test plan

I. Aim

The objective of this test plan is to ensure that the data collection and preparation subsystem functions correctly and produces a dataset that is suitably formatted, adequately diverse, and correctly annotated for training the object detection model.

II. Hypothesis

If the data collection and preparation subsystem has been implemented correctly, it should provide a sufficiently large and diverse dataset with accurate annotations that can be used to effectively train the object detection model.

III. Tools required for the System Test

1. The dataset of 2.7k images created by the subsystem.
2. Image viewing software to visually inspect the images and annotations.
3. Data analysis software (development toolkit) for analysing the dataset.

IV. Method

1. Image Check: Inspect a random sample of images to verify that all images are correctly resized to 640x640 pixels.

2. Annotation Check: Use an image viewer to visually inspect the annotations in a subset of images. The annotations should accurately highlight the corroded areas in each image.
3. Data Augmentation Check: Examine the augmented images to ensure the data augmentation techniques (rotation, cropping, and noise addition) have been correctly applied and that they add meaningful diversity to the dataset.
4. Dataset Diversity Check: Use data analysis tools to evaluate the diversity of the dataset, checking for a reasonable balance in the variety and distribution of the augmented images.

V. Checklist

1. All images are correctly resized to 640x640 pixels.
2. Annotations accurately highlight the corroded areas in each image.
3. Data augmentation techniques are correctly applied.
4. The dataset is sufficiently diverse and balanced in terms of the variety and distribution of the augmented images.

VI. Expected Results

1. All images should be of the correct size (640x640 pixels).
2. Annotations should correctly and accurately mark the instances of corrosion in each image.
3. Data augmentation techniques should be correctly applied, creating diverse variations of the original images.
4. The dataset should be adequately diverse, ensuring a robust set of examples for the model to learn from.

Appendix 3 Model Training and Optimization Subsystem Design

Appendix 3.1 Model Training and Optimization subsystem description

Appendix 3.1.1 Original YOLOv5n Model Architecture

YOLOv5 belongs to the You Only Look Once (YOLO) family of computer vision models and is widely utilized for object detection. It is available in four primary versions: small (s), medium (m), large (l), and extra-large (x), with each version offering improved accuracy rates. Furthermore, each variant requires varying training durations.

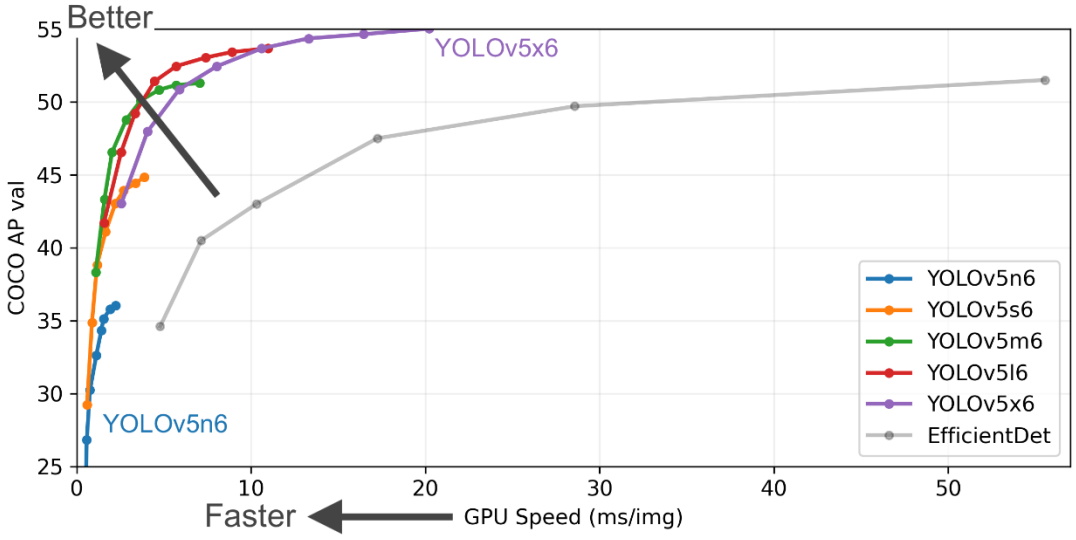


Figure 24. The release of YOLOV5 shows promise of state of the art object detection. (glenn-jocher, AyushExel, Borda, alexstoken, & NanoCode012, 2023)

In the chart, the goal is to produce an object detector model that is very performant (Y-axis) relative to its inference time (X-axis). Preliminary results show that YOLOv5 does exceedingly well in the COCO dataset to this end relative to other state of the art techniques.
In the provided chart, it is evident that all versions of YOLOv5 demonstrate faster training speeds compared to EfficientDet. The YOLOv5x model, which exhibits the highest level of accuracy among the YOLOv5 variants, can process images with comparable precision to the EfficientDet D4 model while achieving multiple times faster processing times. In this project, the YOLOv5n model has been selected due to its superior performance in terms of inference speed and reduced computational requirements.
YOLOv5 derives most of its performance improvement from PyTorch training procedures, while the model architecture remains close to YOLOv4.

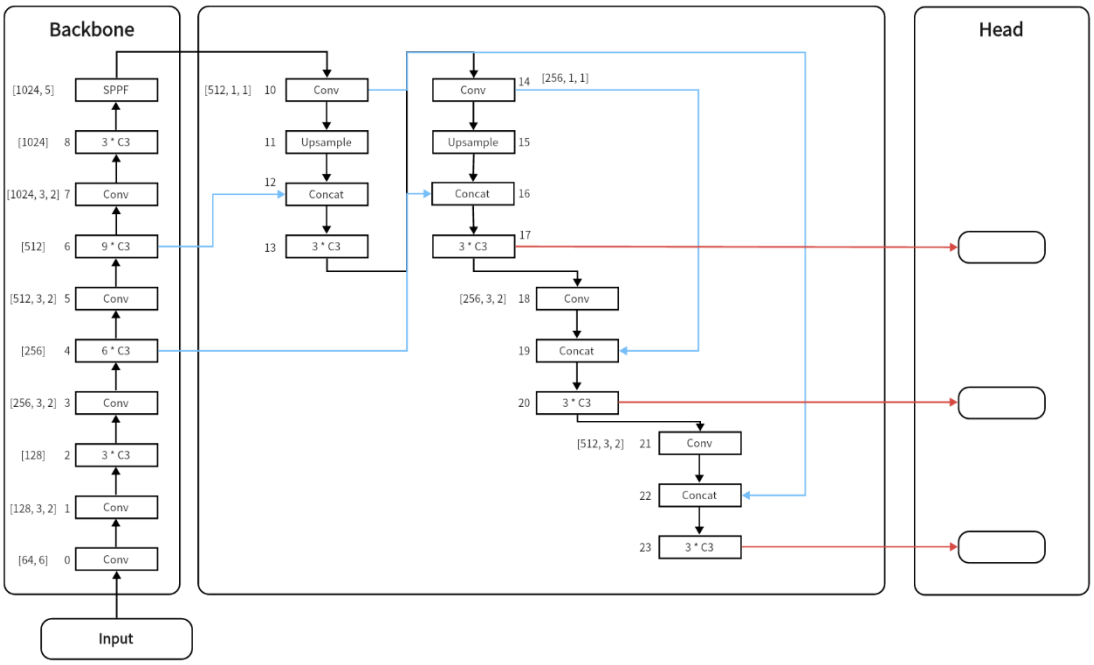


Figure 25. Original YOLOv5n Architecture

Appendix 3.1.2 Model Training and Optimization

This subsystem is dedicated to the development, training, and optimization of the object detection model.

1. **Model Development**: To enhance the base YOLOv5 model's ability to capture fine-grained details related to corrosion, the model architecture is modified by adding a Convolutional Block Attention Module (CBAM) (S, J, & Y, 2018). CBAM is a spatial-channel attention module that considers both spatial and channel-wise features. This project refers to two of the convolutional attention modules of CBAM: channel attention module (CAM) and spatial attention module (SAM). CAM pays more attention to the semantic features of the feature map. The Channel Attention Module applies average pooling in the spatial dimension to aggregate spatial information and max pooling to collect finer target features on the feature map Y of size H×W×C. The simultaneous use of these two pooling operations can reduce the size of the feature map and the amount of computation while improving the expressive capability of the network. After pooling, the two one-dimensional vectors are sent to the fully connected layer for computation, where a 1×1 convolution kernel is used to implement the weight sharing between the feature vectors. Finally, through the addition operation and sigmoid activation, the channel attention Zc is generated.

$$z_c = sigmoid\big(MLP\big(AvgPool(Y)\big) + MLP(MaxPool(Y))\big)$$

SAM pays more attention to the positional information of the features, focusing on areas of the feature map with more effective features, thus supplementing channel attention. Average pooling and max pooling are used to compress the feature map Yc in the channel dimension, resulting in two two-dimensional feature maps, which

are then concatenated based on the channel to produce a feature map with a channel number of 2. To ensure that the final feature is consistent with the input Yc in the spatial dimension, a hidden layer containing a single convolution kernel is used to convolve the concatenated feature map. Finally, the spatial attention weight Zs is generated through the sigmoid operation.

$$z_s = sigmoid(conv(AvgPool(Y), MaxPool(Y)))$$

The attention module tells the model where to concentrate more computation, improving the expressive power of the region of interest. The SCAM module emphasizes focusing on meaningful features in both channel and spatial dimensions, which can be used to focus on important features and suppress invalid features, improving the effective flow of feature information in the network. SCAM is a very lightweight general module that can be seamlessly embedded into the YOLOv5 network architecture for end-to-end training. This project also compares the performance of different attention modules such as SE (Squeeze and Excitation) Networks. The following is the improved YOLOv5 model architecture.

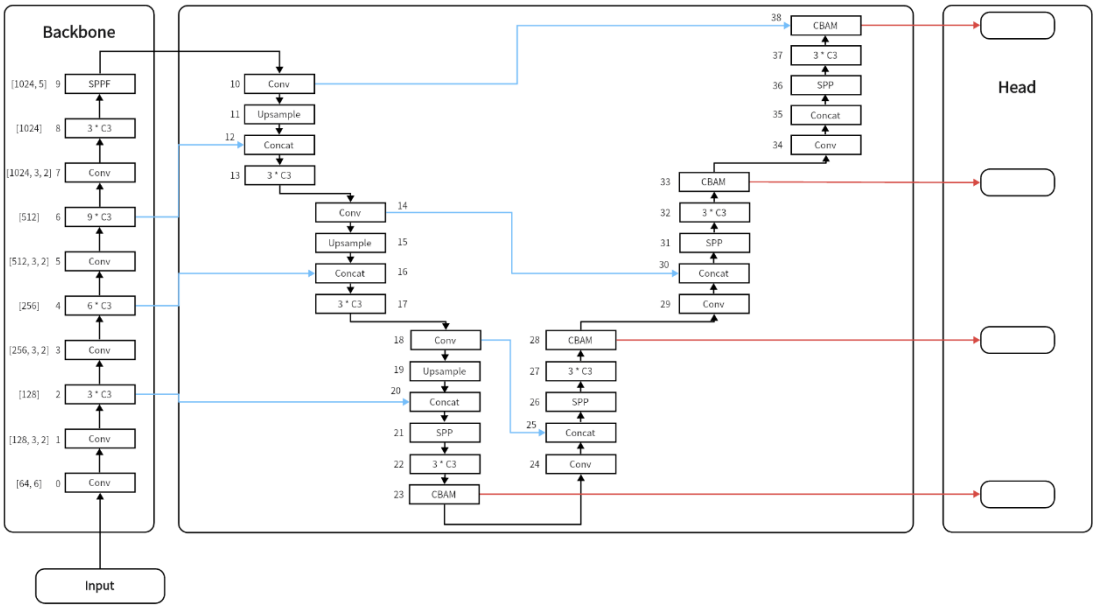


Figure 26. Improved YOLOv5n architecture

2. **Model Training:** The prepared dataset from the previous subsystem is used to train this modified YOLOv5 model. The training process is a supervised learning approach, where the model learns to recognize corrosion instances from the annotated images. This process involves forward propagation (making predictions), calculating the loss (difference between prediction and ground truth), and backpropagation (updating model weights to minimize loss).
3. **Model Optimization:** After initial training, the model undergoes optimization to improve its performance. This includes hyperparameter tuning, where parameters that govern the training process (learning rate, batch size, etc.) are adjusted to find the most effective configuration. Additionally, techniques to prevent overfitting (early stopping and dropout) is employed.

4. **Performance Evaluation:** During and after training, the model's performance is evaluated using precision-recall curve and mAP. These evaluations help track the model's progress, identify potential issues, and determine the effectiveness of optimization strategies.

Overall, the Model Training and Optimization subsystem focuses on developing a high-performing object detection model that can effectively detect and mark instances of corrosion in drone-captured images. This subsystem is crucial as the model's accuracy and reliability have direct implications on the success of the entire project.

Appendix 3.2 Model Training and Optimization subsystem test plan

I. Aim

The purpose of this test plan is to confirm that the model training and optimization subsystem operates effectively, resulting in an object detection model that can accurately identify and localize corrosion in images.

II. Hypothesis

If the model training and optimization subsystem has been executed correctly, then the modified YOLOv5 model should be able to detect and accurately mark instances of corrosion in the test datasets.

III. Tools required for the System Test

1. The trained YOLOv5 model
2. Test dataset (a subset of the original dataset not used in training)
3. Object detection evaluation metrics (mAP, Precision-Recall curve)
4. Program involved in the repository of this project that based on Python for running the model and computing the evaluation metrics.

IV. Method

1. Model Functionality Check: Run the model with a few test images and verify that it operates without errors and produces expected outputs (bounding boxes and class predictions).
2. Performance Evaluation: Evaluate the model's performance on the test dataset. This can be done by computing common evaluation metrics like precision, recall, F1 score, and possibly Mean Average Precision (mAP) if multiple classes are involved.
3. Hyperparameter Optimization Check: Review the training logs to confirm that hyperparameters were tuned and that the optimization improved the model's performance.
4. Training Gradient Check: Review the training loss curve to ensure that the model training is gradient, i.e., the loss decreased over time and eventually stabilized.

V. Checklist

1. The model operates without errors and produces expected outputs.

2. The model's performance metrics meet or exceed the project requirements.
3. Hyperparameters have been optimized and the optimization improved the model's performance.
4. The model training is gradient, as evidenced by the training loss curve.

VI. Expected Results

1. The model should function correctly without any errors, producing bounding boxes and class predictions for input sources.
2. The model's performance metrics should meet or exceed the project requirements, indicating effective learning and generalization ability.
3. The training logs should show evidence of hyperparameter tuning and improvement in the model's performance due to optimization.
4. The training loss curve should show a trend of decreasing loss over epochs, demonstrating that the model training has converged.

Appendix 4 Model Deployment Subsystem Design

Appendix 4.1 Model Deployment Subsystem Description

This subsystem focuses on integrating the trained object detection model into the VOXL 2 device, a key component of the drone system.

1. **SDK Modification:** The provided SDK from ModalAI, specifically voxl-tflite-server, is modified according to the project's requirements. The primary modifications are made in the main.cpp file and the inference_helper.cpp file to integrate the trained model.
2. **Building the Environment:** The voxl-tflite-server project builds within the voxl-cross docker image (version 1.7 or above). The docker image is launched, and the required dependencies are installed within the docker environment. For dependency installation, the hardware platform 'qrb5165' and the target binary repo 'dev' are specified.
3. **Building Scripts:** Scripts are built for the 64-bit hardware platform using the appropriate cross compilers.
4. **Creating a .deb Package:** A Debian (.deb) package is created within the docker environment. This package encapsulates the modified voxl-tflite-server project with the integrated object detection model, enabling easy installation onto the target device.
5. **Deploying to VOXL 2:** The created .deb package is transferred to the VOXL 2 using the ADB command "./deploy_to_voxl.sh". This command installs the package on the VOXL 2, effectively deploying the object detection model onto the device.

In summary, the Model Deployment subsystem involves a series of technical steps to deploy the trained object detection model onto the VOXL 2 device. This allows the device to run the model in real-time and frame the targets (corrosions) from the drone-captured images. Successful deployment is critical to the functional completion of the project.

Appendix 4.2 Model Deployment Subsystem test plan

I. Aim

The aim of this test plan is to ensure that the model deployment subsystem operates effectively, leading to a successful installation of the object detection model on the VOXL 2 device.

II. Hypothesis

If the model deployment subsystem has been executed properly, the object detection model should be successfully deployed onto the VOXL 2 device and should be able to process input images in real-time.

III. Tools required for the System Test

1. The VOXL 2 platform
2. ADB (Android Debug Bridge)
3. HereLink

4. Test images for real-time testing

IV. Method

1. Deb Package Deployment Check: After deploying the .deb package to the VOXL 2 device using the ADB command "./deploy_to_voxl.sh", check if the package is successfully installed on the device.
2. Real-time Functionality Check: Run the model with real-time inputs and confirm that the model operates without errors and processes the images as expected.
3. Performance Check: Evaluate the model's real-time performance on the VOXL 2 device by providing it with test images and ensuring it accurately detects and frames corrosion instances in a timely manner.

V. Checklist

1. The .deb package is successfully deployed and installed on the VOXL 2 device.
2. The model operates without errors and processes real-time images on the VOXL 2 device.
3. The model's real-time performance on the VOXL 2 device meets or exceeds the project requirements.

VI. Expected Results

1. The .deb package should be successfully deployed and installed on the VOXL 2 device.
2. The model should function correctly on the VOXL 2 device, processing real-time images without errors.
3. The model's real-time performance on the VOXL 2 device should meet or exceed the project requirements, demonstrating successful deployment and operation.

Appendix 5 Toolkit Development Subsystem Design

Appendix 5.1 Toolkit Development Subsystem Description

This subsystem pertains to the development of a toolkit that facilitates easier and more efficient training of object detection models for users who may not be familiar with the process.

1. **Model Modification:** The YOLOv5 model, based on the Ultralytics version, is adjusted. This involves incorporating an attention module and altering the model's architecture to enhance its ability to detect fine-grained details related to corrosion.

2. **Format Conversion Adaptations:** To facilitate further model quantization and conversion, adjustments are made to ensure the model can be converted from PyTorch format to TensorFlow format. This conversion is a vital step as the model deployed on the VOXL 2 device is required to be in TensorFlow Lite (tflite) format.

3. **Jupyter Notebook Creation:** To simplify the model training and conversion process, a Jupyter notebook is created. This notebook houses the code and serves as an interactive platform for executing the model training, conversion, and related processes.

4. **Instruction Manual (README file):** A detailed instruction manual, provided in the form of a README file, is developed to guide users through the process of training a customized model. This document provides step-by-step instructions, effectively making the toolkit accessible and user-friendly, even for individuals who might not have a deep understanding of the underlying processes.

In summary, the Toolkit Development subsystem is aimed at making the process of training and deploying a customized object detection model more straightforward and accessible. It helps to ensure that the project's benefits can continue to be realized, even after the project's conclusion. This subsystem is integral to meeting the second objective of the project, fostering a more sustainable model of technology use and development within the company.

Appendix 5.2 Toolkit Development test plan

I. Aim

The aim of this test plan is to verify the successful operation of the toolkit development subsystem, ensuring the YOLOv5 model's modifications, conversions, and notebook operations are functioning as intended, and the instruction manual is comprehensive and user-friendly.

II. Hypothesis

If the toolkit development subsystem has been correctly implemented, then users should be able to train their own customized model following the instructions and using the provided Jupyter notebook.

III. Tools required for the System Test

1. Jupyter notebook
2. Test model for conversion and training
3. A user unfamiliar with the process (to test the instruction manual)

IV. Method

1. Model Modification Check: Confirm the YOLOv5 model has been correctly modified, with the attention module and architectural changes effectively implemented.

2. Conversion Check: Verify the successful conversion of the modified YOLOv5 model from PyTorch format to TensorFlow format, preparing it for further quantization and conversion to TensorFlow Lite (tflite) format.

3. Jupyter Notebook Functionality Check: Validate the functionality of the Jupyter notebook in facilitating model training and conversion processes.

4. Instruction Manual Review: Test the instruction manual with a user unfamiliar with the process, evaluating its comprehensiveness and user-friendliness.

V. Checklist

1. The YOLOv5 model has been correctly modified and can be converted from PyTorch to TensorFlow format.
2. The Jupyter notebook operates effectively, simplifying the model training and conversion processes.
3. The instruction manual (README file) is comprehensive, user-friendly, and enables users unfamiliar with the process to train their own customized model.

VI. Expected Results

1. The modified YOLOv5 model should successfully convert from PyTorch format to TensorFlow format.
2. The Jupyter notebook should effectively simplify the model training and conversion processes.
3. The instruction manual should be comprehensive and user-friendly, allowing users unfamiliar with the process to train their own customized model.

Appendix 6 Test results

Appendix 6.1 Data Collection and Preparation Subsystem test result

During the detailed testing process for the Dataset Construction subsystem, a variety of tests are carried out to confirm the successful creation of the dataset, its subsequent augmentation, and the precise annotation of the target corrosion areas in the images.

1. **Image Sourcing and Resizing:** The first step of the subsystem involved sourcing images that contained instances of corrosion in vapor pipes from the company's database. A total of 1079 original images are sourced, each containing corrosion instances of varying degrees and sizes. These images are resized to 640x640 pixels using an online image converter tool. The images maintained a high degree of detail following resizing, preserving the essential features of the corrosion.

Figure 27. Original Image, 6720 x 4480

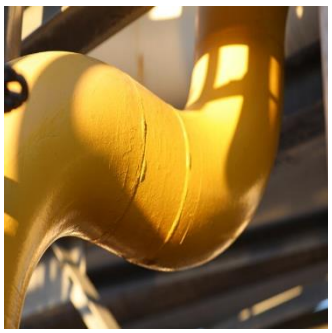


Figure 28. Resized Image, 640 x 640

2. **Annotation:** The next phase involves the annotation of the images. Each image is annotated. The corrosion areas are carefully marked, ensuring accurate boundary identification for the machine learning model.
3. **Data Augmentation:** After annotation, the images undergo augmentation, which involved rotation, cropping, and noise addition. The augmentation process successfully generates an augmented dataset of 2.7k images, thereby enhancing the dataset's size and variability.

Figure 29. Annotated Images with data augmentation

4. **Validation:** The resulting dataset is tested by feeding it into the Model Training & Optimization subsystem. The model can learn effectively from this dataset and detect the annotated corrosion areas, validating the successful creation and augmentation of the dataset.

Figure 30. Model predictions on augmented dataset

The detailed tests concluded that the Dataset Construction subsystem is functioning as expected, with the dataset being of high quality, correctly annotated, and sufficiently augmented. The system is ready for use in the following stages of model training and optimization.

Appendix 6.2 Model Training and Optimization Subsystem test result

The Model Training and Optimization subsystem undergo thorough testing to ascertain the successful adaptation of the YOLOv5 model and the incorporation of the Convolutional Block Attention Module (CBAM).

1. **Model Training:** After adaptation, the model is trained on the augmented dataset created by the previous subsystem. The training phase is monitored to ensure the model is learning effectively and enhancing its detection capabilities.

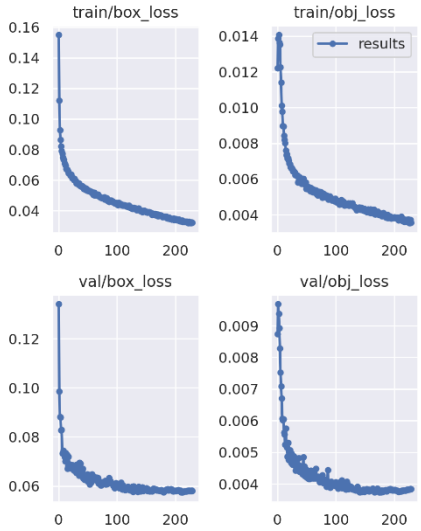


Figure 31. Training Loss vs Epochs

2. **Model Evaluation:** Upon completing the training, the model is evaluated for its detection accuracy. This is done using a validation dataset containing drone-captured images of corrosion.

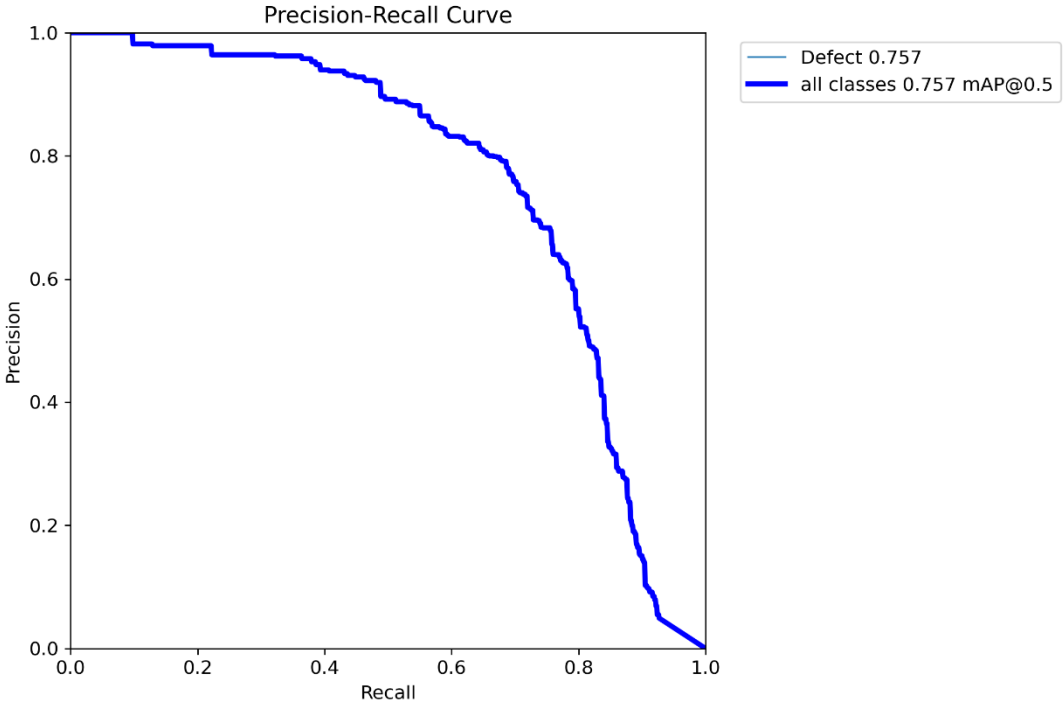


Figure 32. Precision-Recall Curve of improved YOLOv5 model

3. Comparison with Base Model: The modified model's performance is compared with the base YOLOv5 model. This comparison indicates that the improved model improved 0.04 in mAP@0.5.

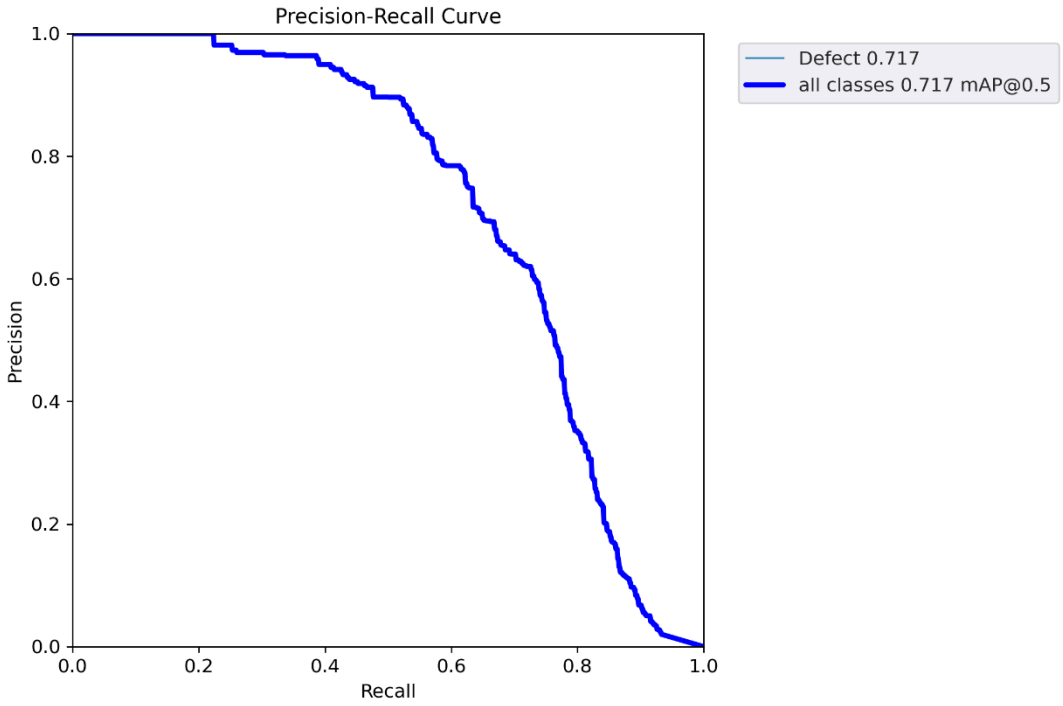


Figure 33. Precision-Recall Curve of the original YOLOv5 model

Appendix 6.3 Model Deployment Subsystem test result

This subsystem is tested in detail to confirm the successful deployment of the modified YOLOv5 model on the VOXL 2 device. The steps for the deployment process and their respective test results are as follows:

1. **SDK Modification:** The initial task involved modifying the SDK (voxl-tflite-server) as per the requirements. The main.cpp and inference_helper.cpp files are changed to adapt to the specific needs of the project. The modifications are successful and doesn't introduce any errors in the code.
2. **Building Environment**: The voxl-tflite-server project is built in the voxl-cross docker image (V2.5). This process is carried out without any issues, verifying the creation of the required environment.
3. **Dependency Installation:** The necessary dependencies are installed inside the docker. The dependencies are correctly specified and installed without errors, validating this step of the deployment process.

```
voxl-cross(2.5):~$ ./install_build_deps.sh qrb5165 dev
using qrb5165 dev debian repo
Ign:1 http://voxl-packages.modalai.com ./dists/qrb5165/dev/binary-arm64/ InRelease
Ign:2 http://voxl-packages.modalai.com ./dists/qrb5165/dev/binary-arm64/ Release
Get:3 http://voxl-packages.modalai.com ./dists/qrb5165/dev/binary-arm64/ Packages [60.7 kB]
Fetched 60.7 kB in 2s (26.2 kB/s)
Reading package lists... Done
installing:
libmodal-pipe
libmodal-json
voxl-opencv
qrb5165-tflite
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  libmodal-json:arm64 libmodal-pipe:arm64 qrb5165-tflite:arm64 voxl-opencv:arm64
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 24.7 MB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 http://voxl-packages.modalai.com ./dists/qrb5165/dev/binary-arm64/ libmodal-json 0.4.3-202209281920 [43.8 kB]
Get:2 http://voxl-packages.modalai.com ./dists/qrb5165/dev/binary-arm64/ libmodal-pipe 2.8.4-202304051615 [83.7 kB]
Get:3 http://voxl-packages.modalai.com ./dists/qrb5165/dev/binary-arm64/ qrb5165-tflite 2.8.0-2-202210071637 [11.0 MB]
Get:4 http://voxl-packages.modalai.com ./dists/qrb5165/dev/binary-arm64/ voxl-opencv 4.5.5-1-202210072305 [13.6 MB]
Fetched 24.7 MB in 6s (4006 kB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 76, <> line 4.)
debconf: falling back to frontend: Readline
Selecting previously unselected package libmodal-json:arm64.
(Reading database ... 30765 files and directories currently installed.)
Preparing to unpack .../libmodal-json_0.4.3-202209281920_arm64.deb ...
Unpacking libmodal-json:arm64 (0.4.3-202209281920) ...
Selecting previously unselected package libmodal-pipe:arm64.
Preparing to unpack .../libmodal-pipe_2.8.4-202304051615_arm64.deb ...
Unpacking libmodal-pipe:arm64 (2.8.4-202304051615) ...
Selecting previously unselected package qrb5165-tflite:arm64.
Preparing to unpack .../qrb5165-tflite_2.8.0-2-202210071637_arm64.deb ...
Unpacking qrb5165-tflite:arm64 (2.8.0-2-202210071637) ...
Selecting previously unselected package voxl-opencv:arm64.
Preparing to unpack .../voxl-opencv_4.5.5-1-202210072305_arm64.deb ...
Unpacking voxl-opencv:arm64 (4.5.5-1-202210072305) ...
Setting up qrb5165-tflite:arm64 (2.8.0-2-202210071637) ...
Setting up voxl-opencv:arm64 (4.5.5-1-202210072305) ...
Setting up libmodal-json:arm64 (0.4.3-202209281920) ...
Setting up libmodal-pipe:arm64 (2.8.4-202304051615) ...
Processing triggers for libc-bin (2.27-3ubuntu1.6) ...

Done installing dependencies
```

Figure 34. Dependency Installation Logs

4. **Build Scripts Execution:** The build scripts are executed successfully.

```
voxl-cross(2.5):~$ ./build.sh qrb5165
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/aarch64-linux-gnu-gcc-7 - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/aarch64-linux-gnu-g++-7 - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/root/build
[ 50%] Building C object src/CMakeFiles/voxl-tflite-server.dir/resize.c.o
[ 50%] Building CXX object src/CMakeFiles/voxl-tflite-server.dir/inference_helper.cpp.o
[ 75%] Building CXX object src/CMakeFiles/voxl-tflite-server.dir/main.cpp.o
[100%] Linking CXX executable voxl-tflite-server
/usr/lib64//libtensorflow-lite.a(nnapi_delegate.cc.o): In function `tflite::delegate::nnapi::NNMemory::NNMemory(NnApi const*, char const*, unsigned long)':
nnapi_delegate.cc:(.text+0xd58): warning: the use of `tmpnam' is dangerous, better use `mkstemp'
[100%] Built target voxl-tflite-server
voxl-cross(2.5):~$
```

Figure 35. Successful build scripts

5. **Package Creation:** A .deb package is created, containing the modified YOLOv5 model in the tflite format. The creation of this package is successful, verifying the successful conversion of the model to the required format.

```
voxl-cross(2.5):~$ ./make_package.sh deb
Package Name:  voxl-tflite-server
version Number:  0.3.1
Consolidate compiler generated dependencies of target voxl-tflite-server
[100%] Built target voxl-tflite-server
Install the project...
-- Install configuration: "RELEASE"
-- Installing: ../pkg/data/usr/bin/voxl-tflite-server
/home/root
starting building Debian Package
dpkg-deb: building package 'voxl-tflite-server' in 'voxl-tflite-server_0.3.1_arm64.deb'.
DONE
voxl-cross(2.5):~$
```

Figure 36. Successfully create package

6. **Real-Time Testing:** The model is tested for real-time performance on drone-captured images. The model can process the images in real-time and correctly identify the corrosion instances.

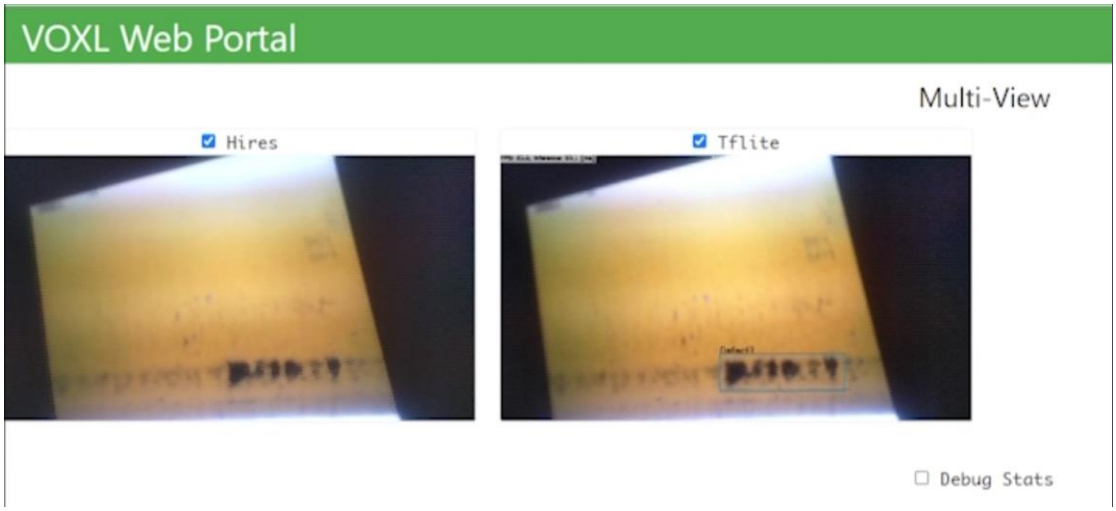


Figure 37. Real-time processing on the VOXL 2

The test concludes that the Model Deployment subsystem is functioning correctly, achieving the successful deployment and operation of the modified YOLOv5 model on the VOXL 2. This subsystem is now ready for real-world operations.

Appendix 6.4 Toolkit Development Subsystem test result

This subsystem undergoes comprehensive testing to ensure the success of the toolkit creation process and the efficient training of the customized model. Here are the results for each significant step of the process.

1. **Modification of YOLOv5 Model:** The YOLOv5 model is successfully modified, with the incorporation of the attention module and changes in the model architecture. The changes are made to suit the specific needs of the project. The modifications did not introduce any errors.

```
# CBAM
class ChannelAttention(nn.Module):

    def __init__(self, in_planes, ratio=16):
        super().__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)
        self.f1 = nn.Conv2d(in_planes, in_planes // ratio, 1, bias=False)
        self.relu = nn.ReLU()
        self.f2 = nn.Conv2d(in_planes // ratio, in_planes, 1, bias=False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = self.f2(self.relu(self.f1(self.avg_pool(x))))
        max_out = self.f2(self.relu(self.f1(self.max_pool(x))))
        out = self.sigmoid(avg_out + max_out)
        return out
```

```python
class SpatialAttention(nn.Module):

    def __init__(self, kernel_size=7):
        super().__init__()
        assert kernel_size in (3, 7), 'kernel size must be 3 or 7'
        padding = 3 if kernel_size == 7 else 1
        # (特征图的大小-算子的size+2*padding)/步长+1
        self.conv = nn.Conv2d(2, 1, kernel_size, padding=padding, bias=False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        # 1*h*w
        avg_out = torch.mean(x, dim=1, keepdim=True)
        max_out, _ = torch.max(x, dim=1, keepdim=True)
        x = torch.cat([avg_out, max_out], dim=1)
        #2*h*w
        x = self.conv(x)
        #1*h*w
        return self.sigmoid(x)
```

```python
class CBAM(nn.Module):
    # CSP Bottleneck with 3 convolutions
    def __init__(self, c1, c2, ratio=16, kernel_size=7):  # ch_in, ch_out, number, shortcut, group
        super().__init__()
        self.channel_attention = ChannelAttention(c1, ratio)
        self.spatial_attention = SpatialAttention(kernel_size)

    def forward(self, x):
        out = self.channel_attention(x) * x
        # c*h*w
        # c*h*w * 1*h*w
        out = self.spatial_attention(out) * out
        return out
```

Figure 38. CBAM module

```yaml
# Parameters
nc: 1  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.25  # layer channel multiple
anchors:
  - [7,9,  9,17,  17,15,  13,27]
  - [21,28,  36,18,  23,47,  35,33]  # P3/8
  - [58,29,43,60,82,46,66,88]  # P4/16
  - [133,77,111,135,206,137,197,290]  # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]
```

Figure 39. Modified YOLOv5 architecture

```
# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [ -1, 1, Conv, [ 128, 1, 1 ] ],
   [ -1, 1, nn.Upsample, [ None, 2, 'nearest' ] ],
   [ [ -1, 2 ], 1, Concat, [ 1 ] ],  # cat backbone P2
   [-1, 1, SPP, [128, [5, 9, 13]]],
   [ -1, 3, C3, [ 128, False ] ],  #  (P2/4-xsmall)
   [-1, 1, CBAM, [128]],            # 23

   [ -1, 1, Conv, [ 128, 3, 2 ] ],
   [ [ -1, 18, 4], 1, Concat, [ 1 ] ],  # cat head P3
   [-1, 1, SPP, [256, [5, 9, 13]]],
   [ -1, 3, C3, [ 256, False ] ],  # (P3/8-small)
   [-1, 1, CBAM, [256]],          # 28

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14, 6], 1, Concat, [1]],  # cat head P4
   [-1, 1, SPP, [512, [3, 7, 11]]],
   [-1, 3, C3, [512, False]],  #  (P4/16-medium)
   [-1, 1, CBAM, [512]],       # 33

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 1, SPP, [1024, [3, 5, 7]]],
   [-1, 3, C3, [1024, False]],  #  (P5/32-large)
   [-1, 1, CBAM, [1024]],        # 38

   [[23, 28, 33, 38], 1, Detect, [nc,anchors]],  # Detect(P2, P3, P4, P5)
  ]
```

Figure 40. Modified YOLOv5 architecture

2. **Conversion Adaptation:** The modified model is adapted to be converted from PyTorch format to TensorFlow format. This step is essential for the model to be quantized and converted into the tflite format. The model is successfully converted without any issues.

```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
export: data=[redacted] weights=['./runs/train/exp/weights/best.pt
YOLOv5  5223659 Python-3.9.13 torch-2.0.0+cu117 CPU

Fusing layers...
YOLOv5n summary: 157 layers, 1760518 parameters, 0 gradients, 4.1 GFLOPs

PyTorch: starting from runs\train\exp\weights\best.pt with output shape (1, 6300, 6) (3.7 MB)

TensorFlow SavedModel: starting export with tensorflow 2.8.0...

                from  n    params  module                                  arguments
2023-05-27 20:59:36.718618: E tensorflow/stream_executor/cuda/cuda_driver.cc:271] failed call to cuInit: CUD
2023-05-27 20:59:36.725926: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagn
2023-05-27 20:59:36.726205: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: ????C??1
2023-05-27 20:59:36.730154: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is o
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
  0               -1  1      1760  models.common.Conv                      [3, 16, 6, 2, 2]
  1               -1  1      4672  models.common.Conv                      [16, 32, 3, 2]
  2               -1  1      4800  models.common.C3                        [32, 32, 1]
  3               -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  4               -1  1     29184  models.common.C3                        [64, 64, 2]
  5               -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  6               -1  1    156928  models.common.C3                        [128, 128, 3]
  7               -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  8               -1  1    296448  models.common.C3                        [256, 256, 1]
...
Detect:          python detect.py --weights runs\train\exp\weights\best_saved_model
Validate:        python val.py --weights runs\train\exp\weights\best_saved_model
PyTorch Hub:     model = torch.hub.load('ultralytics/yolov5', 'custom', 'runs\train\exp\weights\best_saved_m
Visualize:       https://netron.app
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)

converter.allow_custom_ops = True

converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]
tflite_quant_model = converter.convert()
with open('test.tflite', 'wb') as f:
  f.write(tflite_quant_model)
```

✓ 2.0s Python

```
WARNING:absl:Importing a function (__inference_pruned_6862) with ops with unsaved custom gradients. Will lik
```

Figure 41. Successfully model conversion

3. **Jupyter Notebook Creation:** A Jupyter notebook is created to streamline the model training process. The notebook is tested, and it executes correctly, proving its utility for training the customized model.

1. Before Start

```
! pip install -r requirements.txt
! pip uninstall -y tensorflow && pip install -q tensorflow==2.8.0
! pip install protobuf==3.20.1
```

Python

1.1 Mount Google Drive (Google Colab user only)

- First please download the repo and save it in your Google Drive.

+ Code + Markdown

```
from google.colab import drive
drive.mount('/content/drive')

# Change the path to the repo, for example, ' %cd ./drive/MyDrive/the/location/of/the/downloaded/repo/'
%cd ./drive/MyDrive/
```

Python

2. Download the training dataset and save it to **./datasets/**

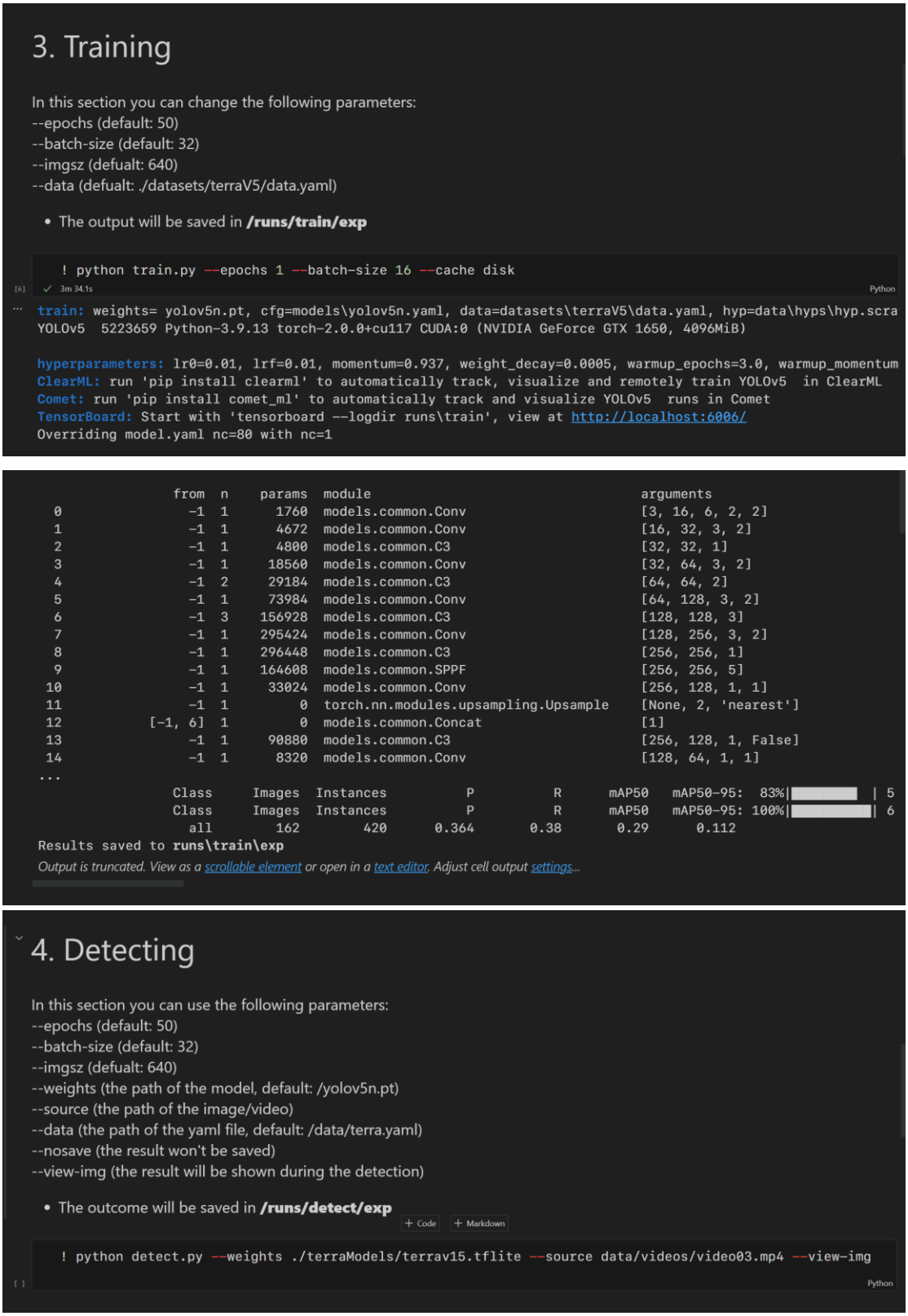


Figure 42. Jupyter Notebook Code and Outputs

5. Quatization & Convert to TfLite

- First convert the torch model to tensorflow saved_model

```
! python export.py --weights ./runs/train/exp/weights/best.pt --imgsz 320
```

[7] ✓ 15.2s Python

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input_1 (InputLayer)        [(1, 320, 320, 3)]        0           []

 tf_conv (TFConv)            (1, 160, 160, 16)         1744        ['input_1[0][0]']

 tf_conv_1 (TFConv)          (1, 80, 80, 32)           4640        ['tf_conv[0][0]']

 tfc3 (TFC3)                 (1, 80, 80, 32)           4704        ['tf_conv_1[0][0]']

 tf_conv_7 (TFConv)          (1, 40, 40, 64)           18496       ['tfc3[0][0]']

 tfc3_1 (TFC3)               (1, 40, 40, 64)           28928       ['tf_conv_7[0][0]']

 tf_conv_15 (TFConv)         (1, 20, 20, 128)          73856       ['tfc3_1[0][0]']

 tfc3_2 (TFC3)               (1, 20, 20, 128)          156288      ['tf_conv_15[0][0]']

 tf_conv_25 (TFConv)         (1, 10, 10, 256)          295168      ['tfc3_2[0][0]']

 tfc3_3 (TFC3)               (1, 10, 10, 256)          295680      ['tf_conv_25[0][0]']

 tfsppf (TFSPPF)             (1, 10, 10, 256)          164224      ['tfc3_3[0][0]']
...
Total params: 1,760,518
Trainable params: 0
Non-trainable params: 1,760,518
```

Figure 43. Jupyter Notebook Code and Outputs

```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
export: data=C:\Users\SiyuC\VSCodeProject\yolov5\data\terra.yaml, weights=['./runs/train/exp/weights/best.pt
YOLOv5  5223659 Python-3.9.13 torch-2.0.0+cu117 CPU

Fusing layers...
YOLOv5n summary: 157 layers, 1760518 parameters, 0 gradients, 4.1 GFLOPs

PyTorch: starting from runs\train\exp\weights\best.pt with output shape (1, 6300, 6) (3.7 MB)

TensorFlow SavedModel: starting export with tensorflow 2.8.0...

                 from  n    params  module                                  arguments
2023-05-27 21:05:38.459826: E tensorflow/stream_executor/cuda/cuda_driver.cc:271] failed call to cuInit: CUD
2023-05-27 21:05:38.466983: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagn
2023-05-27 21:05:38.467312: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: ????C??1
2023-05-27 21:05:38.470936: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is o
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
  0                -1  1      1760  models.common.Conv                      [3, 16, 6, 2, 2]
  1                -1  1      4672  models.common.Conv                      [16, 32, 3, 2]
  2                -1  1      4800  models.common.C3                        [32, 32, 1]
  3                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  4                -1  1     29184  models.common.C3                        [64, 64, 2]
  5                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  6                -1  1    156928  models.common.C3                        [128, 128, 3]
  7                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  8                -1  1    296448  models.common.C3                        [256, 256, 1]
...
Detect:          python detect.py --weights runs\train\exp\weights\best_saved_model
Validate:        python val.py --weights runs\train\exp\weights\best_saved_model
PyTorch Hub:     model = torch.hub.load('ultralytics/yolov5', 'custom', 'runs\train\exp\weights\best_saved_m
Visualize:       https://netron.app
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

- In this case the tensorflow model should be in /runs/train/exp/weights/best_saved_model

```
[8]  ✓ 0.0s                                                                    Python
import tensorflow as tf

# Change the path to your model
saved_model_dir = 'runs/train/exp/weights/best_saved_model'
```

```
[9]  ✓ 2.0s                                                                    Python
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)

converter.allow_custom_ops = True

converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]
tflite_quant_model = converter.convert()
with open('test.tflite', 'wb') as f:
  f.write(tflite_quant_model)
```

```
WARNING:absl:Importing a function (__inference_pruned_6862) with ops with unsaved custom gradients. Will lik
```

- The tflite model should be in the root directory.

Figure 44. Jupyter Notebook Code and Outputs

4. **Readme File Creation:** A detailed instruction file is created to guide users in training their customized models. The instructions are found to be clear and comprehensive during testing.

VOXL 2 ML defect detection

Environment setup & Build dataset

The part below describes the environment setup and building custom dataset. It's much easier and recommanded to train in Google Colab if you have a premium account. The following instruction is based on a local machine.

▼ **Global Environment Setup**

Please be sure to follow the steps to setup the environment.

1. GPU memory >= 8Gb is recommanded
2. CUDA Version
   GTX 10xx series **CUDA >= 11.x**
   GTX 16xx series **CUDA >= 11.x**
   RTX 30xx series **CUDA >= 11.7**
   RTX 40xx series or newer **CUDA >= 12.x**
3. No need to install cudnn
4. Anaconda3-2022.10 is recommanded (Python==3.9.13), newer version should also work
5. Pytorch Version >= 1.8.x, **2.0.0** is recommanded
6. TensorFlow Version == **2.8.0**, TensorFlow is used for quantization. The current VOXL 2 only supports TensorFlow 2.8.0, please see ModelAi official page to check if there is an upgrade.

P.S. If it's running on Colab, only need to type the following command in your Notebook before training:

```
!pip uninstall -y tensorflow && pip install -q tensorflow==2.8.0
```

▼ **Training Environment Setup**

1. Create and activate a virtual environment using command:

```
conda create --name yolov5 python==3.9.12
conda activate yolov5
```

2. Download the yolov5 to local
3. Install the requirements using command:

```
pip install -r requirements.txt
```

Colab user: Follow step 2 and 3

Figure 45. Readme file section

▼ **Dataset setup**
Setup a custom dataset, you could download the prebuild datasets in Terra's shared file.

▼ **Labeling tools**
You could use **labelimg** to annotate images.

1. In the virtual environment, type the following command to install the tool:

```
pip install labelimg
```

2. In the virtual environment type the following command to open the tool:

```
labelimg
```

3. Change the save format to **YOLO**, the default should be PascalVOC

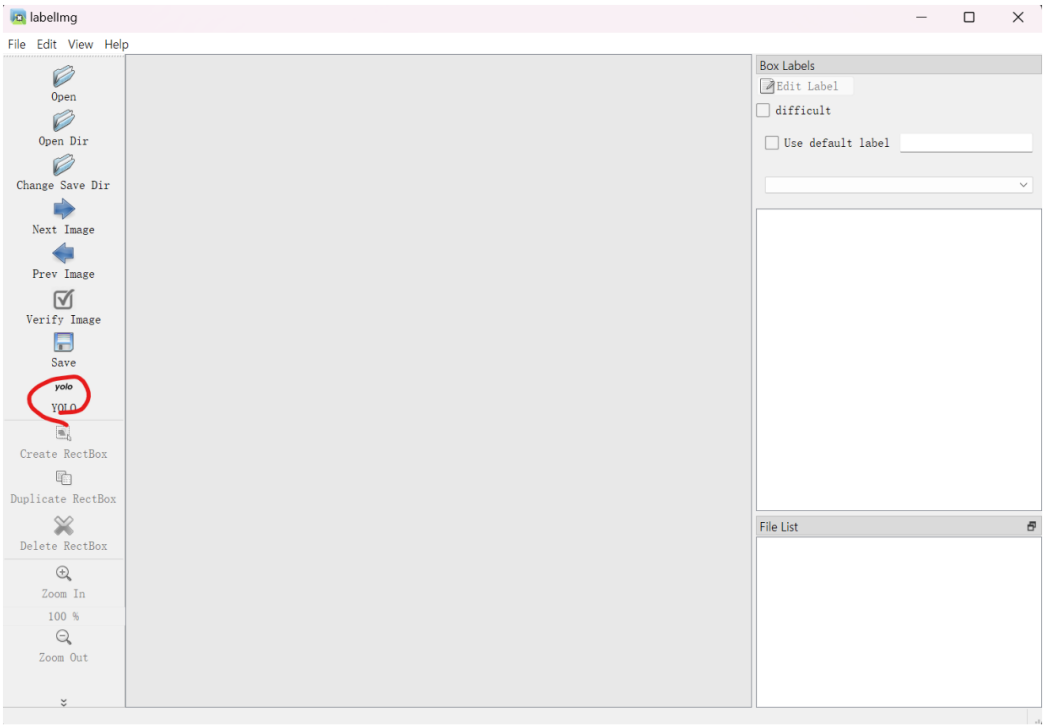


Figure 46. Readme file section

4. After labeling, save the images and annotation files to ***/datasets***

5. Split the dataset into training set and validation set and test set, the recommanded division ratio is 8 : 1 : 1, see the terra dataset for more information. The stucture of the dataset should be as follows.

▼ custom datasets

```
- train
    - images
    - labels
- valid
    - images
    - labels
- test
    - images
    - labels
```

It's not necessary to use the same structure as is shown in this instruction, you need to create a **.yaml** file to let the program know what is the structure of your dataset.

Online labeling tools like https://www.makesense.ai/ are also recommanded, be sure to choose the YOLO format.

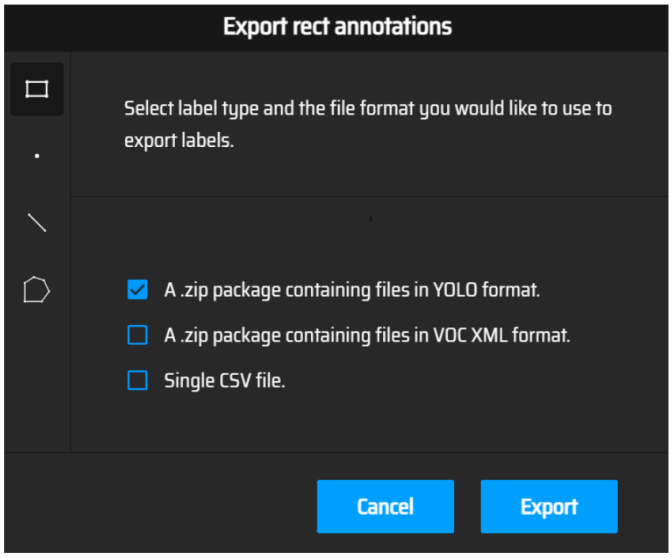
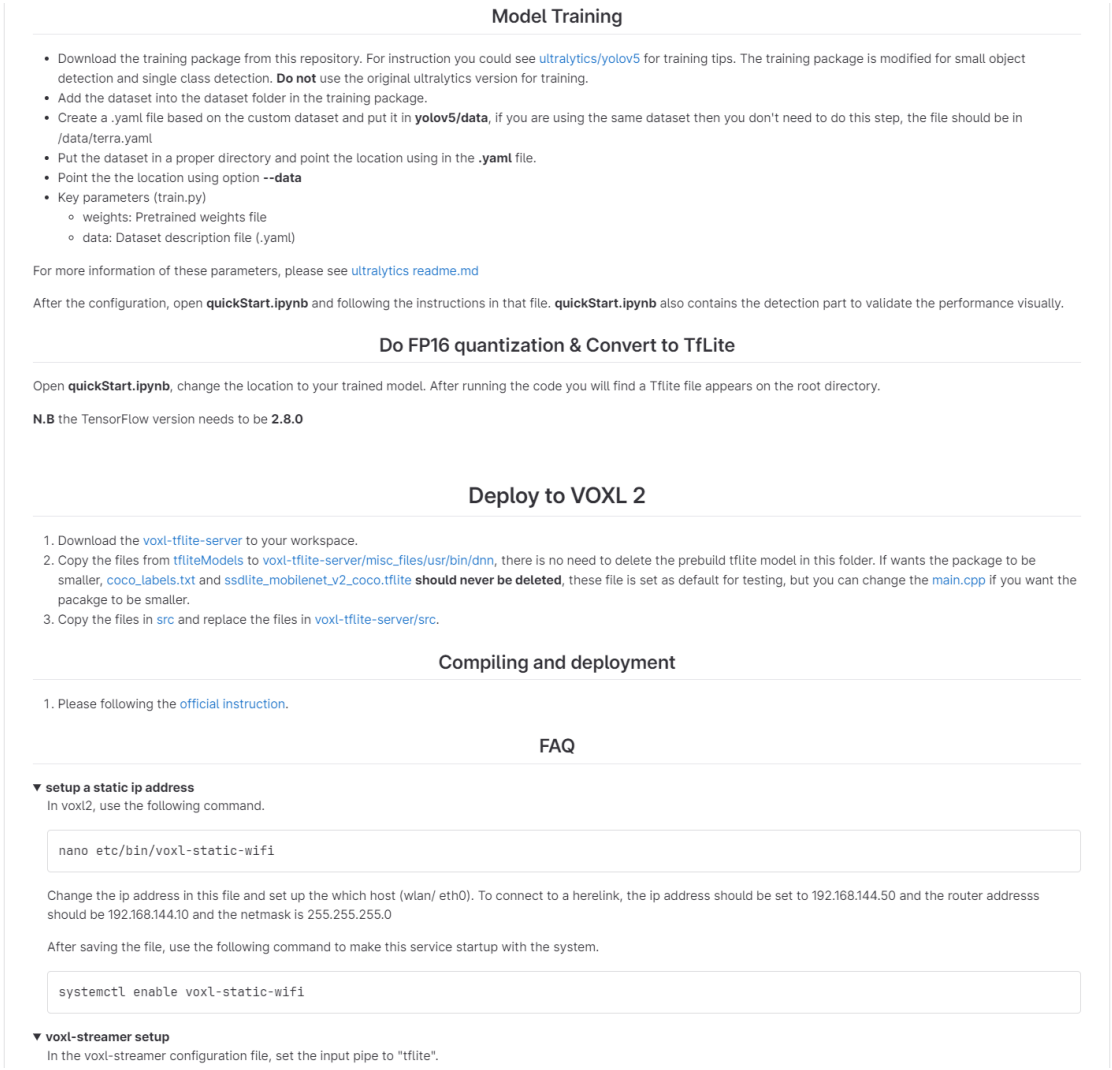


Figure 47. Readme file section



Model Training

- Download the training package from this repository. For instruction you could see ultralytics/yolov5 for training tips. The training package is modified for small object detection and single class detection. **Do not** use the original ultralytics version for training.
- Add the dataset into the dataset folder in the training package.
- Create a .yaml file based on the custom dataset and put it in **yolov5/data**, if you are using the same dataset then you don't need to do this step, the file should be in /data/terra.yaml
- Put the dataset in a proper directory and point the location using in the **.yaml** file.
- Point the the location using option **--data**
- Key parameters (train.py)
  - weights: Pretrained weights file
  - data: Dataset description file (.yaml)

For more information of these parameters, please see ultralytics readme.md

After the configuration, open **quickStart.ipynb** and following the instructions in that file. **quickStart.ipynb** also contains the detection part to validate the performance visually.

Do FP16 quantization & Convert to TfLite

Open **quickStart.ipynb**, change the location to your trained model. After running the code you will find a Tflite file appears on the root directory.

N.B the TensorFlow version needs to be **2.8.0**

Deploy to VOXL 2

1. Download the voxl-tflite-server to your workspace.
2. Copy the files from tfliteModels to voxl-tflite-server/misc_files/usr/bin/dnn, there is no need to delete the prebuild tflite model in this folder. If wants the package to be smaller, coco_labels.txt and ssdlite_mobilenet_v2_coco.tflite **should never be deleted**, these file is set as default for testing, but you can change the main.cpp if you want the pacakge to be smaller.
3. Copy the files in src and replace the files in voxl-tflite-server/src.

Compiling and deployment

1. Please following the official instruction.

FAQ

▼ **setup a static ip address**
In voxl2, use the following command.

```
nano etc/bin/voxl-static-wifi
```

Change the ip address in this file and set up the which host (wlan/ eth0). To connect to a herelink, the ip address should be set to 192.168.144.50 and the router addresss should be 192.168.144.10 and the netmask is 255.255.255.0

After saving the file, use the following command to make this service startup with the system.

```
systemctl enable voxl-static-wifi
```

▼ **voxl-streamer setup**
In the voxl-streamer configuration file, set the input pipe to "tflite".

Figure 48. Readme file section

5. **Toolkit Usability Testing:** The toolkit is tested for usability by a third-party user who is not involved in the project. The user can use the toolkit to train a customized model successfully, confirming the toolkit's usability and effectiveness.

The detailed tests affirm that the Toolkit Development subsystem is functioning as intended, facilitating the creation of a toolkit for efficient training of customized models. The subsystem is validated for future use in training customized models for corrosion detection.

Appendix 6.5 System test result

The goal of the system testing is to validate the complete functionality of the corrosion detection system when all subsystems are integrated and working in unison. Each subsystem – Data Collection and Preparation, Model Training and Optimization, Model Deployment, and Toolkit Development - undergoes a rigorous series of tests to validate their individual functionalities as well as their interoperability within the entire system. Here are the results:

1. Data Collection and Preparation Subsystem: Passes the subsystem test, demonstrating a high-quality dataset conducive to the detection of corrosion (Refer to Appendix 6.1 for more details).
2. Model Training and Optimization Subsystem: Passes the subsystem test, validating the successful training and optimization of the modified YOLOv5 model (Refer to Appendix 6.2 for more details).
3. Model Deployment Subsystem: Passes the subsystem test, proving successful model deployment and real-time performance on the VOXL 2 device (Refer to Appendix 6.3 for more details).
4. Toolkit Development Subsystem: Passes the subsystem test, affirming the usability and efficiency of the toolkit for training of customized models (Refer to Appendix 6.4 for more details).

Upon the successful completion of all subsystem tests, the corrosion detection system can be deemed to have passed the system test. The system has proven its readiness for real-world deployment and use, meeting both of its primary objectives efficiently. The detailed results and descriptions are provided in the respective appendices.