Object Detection Model Optimization and Deployment on Edge Devices

Master's Thesis

Vishvajit Basagonda Jambuti

Department of Electrical and Computer Engineering
Chair of Electromobility

Supervisors:
Prof. Dr.-Ing. Daniel Görges
Julian Hund, M.Sc.

June 26, 2023

Affidavit (Eidesstattliche Erklärung)

This thesis has been prepared on initiative and under guidance of my advisors. For the preparation I have not used others than the indicated aids.

Kaiserslautern, June 26, 2023

Vishvajit Basagonda Jambuti

Abstract

This research work explores the task of object detection to improve the safety of the construction site by using Convolutional Neural Network. The thesis considers "COCO data set" and "Europe City Data-set ", these two data sets are pre-processed and combined. Furthermore, the datasets are trained using YOLOv5 model with encoder-decoder architecture. For the Next step trained model Model is Optimized by using Quantization Method. The model is optimized using TensorRT and for the optimization, FP16 quantization is used. Model Optimization allowed to increase in overall inference speed For Object detection. Tor this particular research work images from four cameras have been collected and pre-processed. And these pre-processed Images are used for object detection. Images are received and pre-processed parallel and also objects are detected parallel in real time.

Kurzfassung

Diese Forschungsarbeit befasst sich mit der Aufgabe der Objekterkennung zur Verbesserung der Sicherheit auf der Baustelle mit Hilfe eines Convolutional Neural Network. In der Arbeit werden der "COCOData-set" und der "Europe-City-Dataset" betrachtet, diese beiden Datensätze werden vorverarbeitet und kombiniert. Außerdem werden die Datensätze mit dem YOLOv5-Modell mit Encoder-Decoder-Architektur trainiert. Im nächsten Schritt wird das trainierte Modell mit Hilfe der Quantisierungsmethode optimiert. Das Modell wird mit TensorRT optimiert und für die Optimierung wird die FP16-Quantisierung verwendet. Die Modelloptimierung ermöglichte es, die Gesamtgeschwindigkeit der Inferenz für die Objekterkennung zu erhöhen. Für diese spezielle Forschungsarbeit wurden Bilder von vier Kameras gesammelt und vorverarbeitet. Diese vorverarbeiteten Bilder werden dann für die Objekterkennung verwendet. Die Bilder werden parallel empfangen und vorverarbeitet, und auch die Objekte werden parallel in Echtzeit erkannt.

Contents

List of Figures

List of Tables

List of Algorithms

Notation, Indices and Abbreviations

Abbrevations

AP	Average Precision
mAP	Mean Average Precision
BCELoss	Binary Cross Entropy Loss
BCEL	Binary Cross-Entropy with Logits Loss
CNN	Convolutional Neural Network
CIoU	Complete Intersection over Union
CSP	Cross Stage Partial
DDS	Data Distribution System
FPN	Feature Pyramid Networks
FPS	Frames Per Second
GPU	Graphics Processing Unit
HPC	high-performance computer
IOU	Intersection Over Union
MPC	Model Predictive Control
ML	Machine Learning
ONNX	Open Neural Network Exchange
PAN	Path Aggregation Networks
PP	Parameter Predictor
PND	Parallel Network Duplication
RHC	Receding Horizon Control
ROS	Robotic Operating System
RSTP	Real Time Streaming Protocol
SiLU	Sigmoid Linear Unit
SVM	Support Vector Machine
SGD	Stochastic Gradient Descent
SSD	Single Shot Detector
YOLO	You Only Look Once

1 Introduction

According to World Health Organization(WHO), every year approximately 1.3 million people die in road traffic accidents.[Mur22]This number is rising every year because of the rise in the number of vehicles on the road. The above-mentioned death number is only taken into consideration for accidents that happened on the road. Also, there are large numbers of vehicles operated in different places such as factory floors, construction sites, farming, etc. To operate vehicles in different applications and different conditions, Advance Driving Assistance System(ADAS) has become a vital part of the driving experience [Mur22]. Having advance warning even seconds before the incident can help the driver to manage a situation in a more secure manner. Because of this ADAS has become an extremely important device in terms of ensuring safety. The current ADAS technologies are based on visual cameras, RADARs, and LiDARs to detect objects [Mur22], [WZW17]. These technologies focused on characteristics like speed, precision, affordability, and energy efficiency. Apart from this ADAS should work effectively in all conditions, such as different light conditions, and different weather conditions[NB20; Liu22; KAZ22]. So, to detect stuff around you while driving, we can use multiple sensors(such as RADAR, and LiDARs)[WZW17], but the reality, good sensors that work super-fast cost a lot of money and consume a lot of power, and require a large amount of computational capacity. Plus, if they keep running all the time, they wear out quickly(RADAR and LiDARs have some moving parts). Using a camera sensor can be a cost-effective solution for object detection in comparison to other high-rate sensors, such as LiDARs or RADARs. Cameras can capture and process images at a high rate of speed and have relatively low power consumption compared to other sensors. Moreover, camera sensors are widely available in the market, making them a more practical and easily accessible option for many applications. Because of the above-mentioned reason For this research work, Deep Learning based YOLOv5 object detection algorithm is used.

Currently, Deep Learning object detection methods are roughly divided into two categories: The first category includes single-stage detection methods such as YOLO and SSD [Liu16; Red16]. And the second category is a two-stage detector which includes Faster R-CNN, Mask R-CNN, and R-FCN [Che18; He17]. Object detection response in one stage detector is faster than in a two-stage detector, but two-stage detectors compensate by being more accurate as compared to one-stage

detectors.

1.1 Objective

The thesis is an attempt to create a safety system for the construction vehicle which can be deployed on the edge device(embedded device) on the vehicle and provide object detection visualization to the driver for safety purposes. The environment use-ase concentrates on the construction site in the urban area.

The objective of this work includes the following tasks:

- Select Appropriate Data-set for Object Detection
- Select Appropriate Algorithm for Object Detection
- Train the Data-set for Selected Object Detection Model
- Optimize the trained Model to be used on the ARM64-based board
- Create an Image data Pipeline and remove the distortion in images
- Implement and deploy the model on Jetson Nano

1.2 Thesis Overview

The thesis setup as follows chapter 2 encapsulates basic concepts behind the thesis and covers fundamental aspects of machine learning, deep learning, and object detection. Later in chapter 3 list out the related work in the field of object detection, TensorRT, and ROS2. The concept of implementation is discussed in chapter 4, Also this chapter covers the system design, data pre-processing and training of the object detection model. The chapter 5 contains the evaluation of the training process and comparing the model before and after the optimization. The final chapter 6 summarizes the thesis and discusses the future work which can expand the current finding.

2 Basic Concepts

This chapter of the thesis provides a comprehensive discussion on the concept of Machine Learning (section 2.1) and its application in the field of computer vision, specifically for the task of object detection. In order to understand this task, the chapter covers the necessary background information and basic concepts required to understand object detection.

The section 2.1 discusses the concept of Machine Learning and how it extends to the field of computer vision for object detection. In section 2.2, the research proceed with a supervised learning approach and select a Neural Network algorithm as the foundation for this work. The Convolutional Neural Network (CNN) is further explored in section 2.4, with a specific focus on the YOLOv5 algorithm in section 2.6.

The section 5.1, emphasizes the importance of data and training behavior of the dataset, which is crucial for the success of the object detection task. The trained model is then integrated into the Robotic Operating System (ROS) framework in the introduction workflow of the project, with a brief discussion on ROS2 in section 2.7.

The hardware used in the project work is introduced in section 4.8, where we have provided details on the advantages and limitations of the hardware. To overcome these limitations, we explained how to use Docker in section 2.8. The model optimization method, which is an important aspect of the thesis work, is discussed in section 2.9. The trained model is deployed using the TensorRT model, with reasons and advantages explained in a subsequent section.

2.1 Machine Learning

In the late 1950s, computer scientist Arthur Samuel made a landmark contribution to the field of artificial intelligence by defining machine learning as a discipline that allows computers to learn and improve their performance without explicit instructions. His definition emphasized the significance of the machine's ability to learn and adapt, rather than relying solely on human-written code. This idea was further expanded upon by Alan Turing, who is widely regarded as one of

the pioneers of computer science. In a seminal paper published in 1950, Turing proposed a way to evaluate the intelligence of a machine, by setting a standard for determining if a machine's behavior and responses are indistinguishable from those of a human being. This paper served as an important milestone in the development of machine learning, laying the foundation for future research and advancements in the field. Tom M. Mitchell (1997) provided a more technical definition of machine learning as follows

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [[Mit07]]

The definition refers to the concept of machine learning, where a computer program can improve its performance on a certain set of tasks (T) by learning from experience (E) through exposure to data and using a performance measure (P) to evaluate its success. In other words, as the program encounters more and more examples of the tasks it is trying to perform, it can adjust its algorithms to better perform these tasks based on its past experiences. The performance measure (P) is used to quantify the success or accuracy of the program's performance on the tasks in T. For example, consider a computer program designed to identify objects in images. The class of tasks T could be object recognition, the experience E could be the images and corresponding object labels used for training the program, and the performance measure P could be the accuracy of the program's predictions on a separate set of images. Over time, as the program is exposed to more and more training images, its performance on the tasks in T (object recognition) will improve, as measured by P (accuracy), because it has learned from its experience E.[[Mit07]]

Author [SB14] has provided rough classifications of learning paradigms, with the aim of providing context for the broader field of machine learning. Four parameters are described to facilitate the classification of learning paradigms as follows.

- Supervised versus Unsupervised
- Active versus Passive Learners
- Helpfulness of the Teacher
- Online versus Batch Learning Protocol

2.1.1 Supervised versus Unsupervised

Supervised Learning

Supervised learning is an approach to machine learning where the algorithm learns from labeled data. The labeled data is used to train the algorithm to predict the correct output for new, unseen data. The main goal of supervised learning is to find a mapping algorithm that can accurately predict the output of unseen data based on the input features. In other words, it aims to generalize the learned pattern so that it can be applied to new data. This approach is useful in many areas, such as image and speech recognition, natural language processing, and predictive analytics. By using labeled data, the algorithm can learn from the examples and improve its accuracy in predicting the output for new, unseen data. To put it differently, supervised learning is a machine learning approach where a known set of input-output pairs, called the training dataset (x_i, y_i), is used to train an algorithm to recognize patterns in data and predict outputs for new, unseen data. In this approach, the input data(x_i) is converted into feature vectors that contain multiple features of the corresponding object. The algorithm searches for patterns or features in the training data to learn a mapping algorithm that can accurately predict the output ($y_i^{'}$) for new input data. The predicted output ($y_i^{'}$) is then compared to the actual target output (y_i) from the training dataset, and the algorithm adjusts its parameters to minimize the error between the predicted and target outputs. This iterative process continues until the algorithm achieves the desired level of accuracy. In short, supervised learning is like a teacher teaching a student, where the student learns by being shown examples of the correct answers and adjusting their understanding to minimize errors[[Kim17]]. The mapping function(f) can be represented as :

$$y^{'} = \theta^T f(X) \tag{2.1}$$

where ($y^{'}$) represents the predicted output of input X; θ corresponds to parameter of mapping algorithm. The Figure 2.1 illustrates the supervised learning algorithm.

Supervised learning can be further classified into two categories:

- **Classification:** A classification technique predicts discrete outcomes by categorizing the input into different classes i.e it sorts the input into different categories or classes. For example, categorizing customer feedback as either positive or negative

  Classification problems can be further divided into:

  1. Binary Classification: Binary classification is the process of dividing

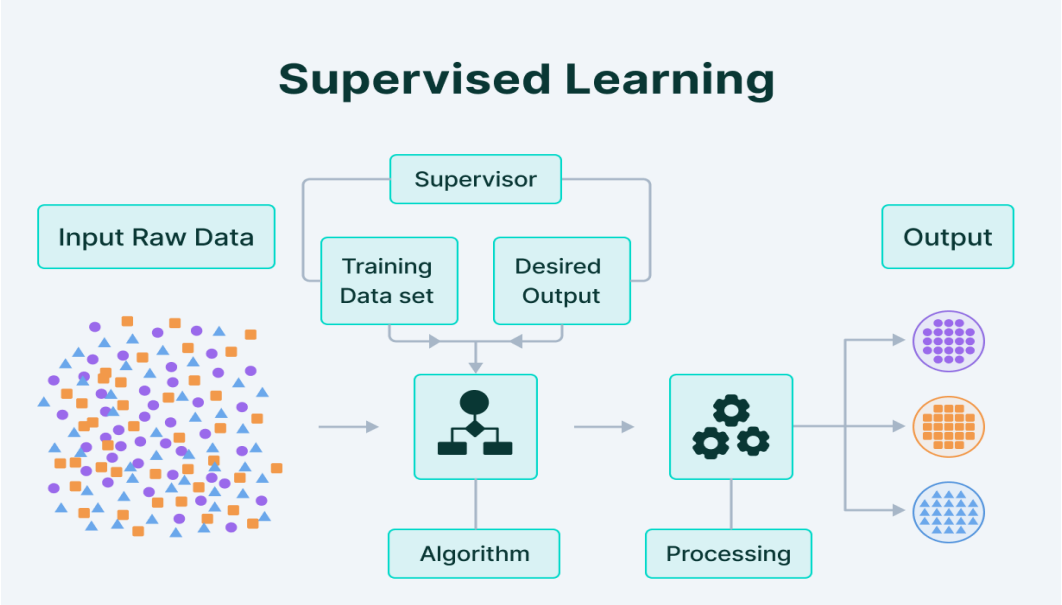


Figure 2.1: Supervised Learning
[Lab21]

a group of elements into two separate classes using a set of rules for classification. For example, A medical diagnosis involves using a binary classifier that takes into account a patient's symptoms as input features. Based on this, the classifier predicts whether the patient is healthy or has a specific disease. The potential results of this diagnosis are either positive (indicating the presence of the disease) or negative (indicating the absence of the disease).

2. Multi-class Classification: Multi-class classification refers to a type of machine learning classification where there are more than two classes or outputs. To illustrate, using a model to classify animal types in images from an encyclopedia is an example of multi-class classification because there are numerous animal classifications, and each image is classified into a single category. It is important to note that in multi-class classification, each sample is classified into one class only.

Classification tasks in machine learning can be performed by several algorithms, including support vector machines, decision trees (with boosting or bagging), k-nearest neighbor method, Bayes classification, discriminant analysis, logistic regression, and neural networks.

- **Regression:** Machine learning regression is a method utilized in predictive modeling that examines the correlation between independent variables or features and a dependent variable or outcome. It predicts continuous out-

comes by utilizing an algorithm. A few common examples of regression tasks include predicting electricity bills using historical load data and estimating the price of a house based on selected features such as the number of bedrooms, location, and other relevant factors.

Common regression algorithms are linear model, Non-linear model, regularization, decision tree with boosting and bagging, and Neural Networks

Unsupervised Learning

Unsupervised learning is a type of machine learning method that doesn't require a user to supervise or label the data being processed. This technique involves allowing the machine learning model to explore and identify patterns and hidden information in the data on its own. Unlike supervised learning, where the algorithm is trained using labeled data, unsupervised learning involves using unlabelled data.[PT16] The purpose of unsupervised learning is to explore and identify underlying patterns in data, and this technique is commonly used in areas such as data mining, anomaly detection, and clustering. The Figure 2.2 illustrates unsupervised learning. For example, unsupervised learning can be used to identify similar patterns in customer data for a business to group customers into different segments based on their behavior, preferences, or purchase history. Another example is using unsupervised learning to identify patterns in financial data to detect fraudulent activities.

The unsupervised learning technique can be further distinguished into the following:

- **Clustering:** Clustering involves categorizing a population or set of data into groups, with the goal of making the data points within each group as similar to one another as possible and as dissimilar to the data points

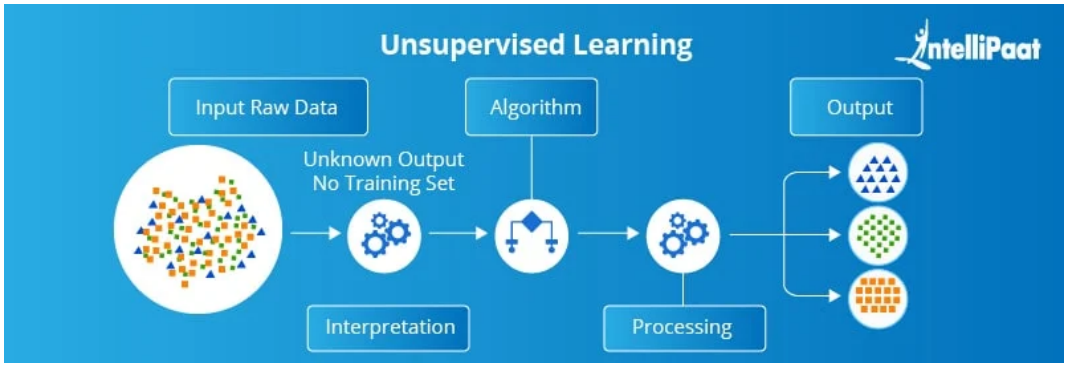


Figure 2.2: Supervised Learning
[Int21]

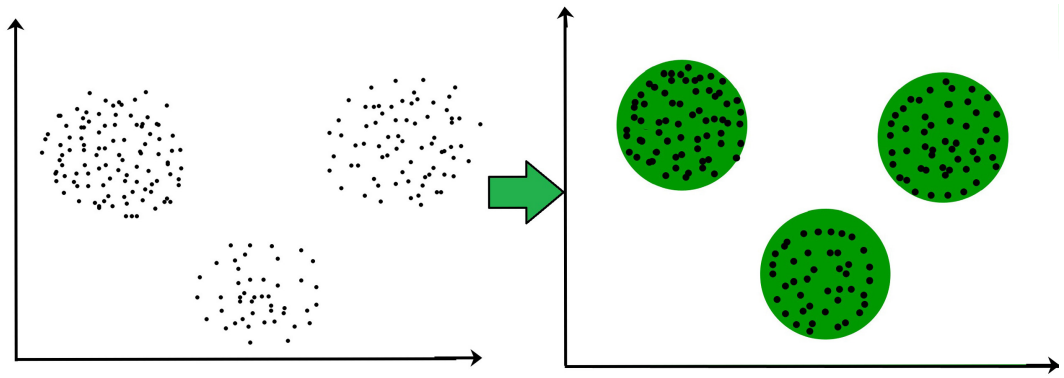


Figure 2.3: Clustering
[Gee21]

in other groups as possible. In essence, it is a way of gathering objects based on how similar or dissimilar they are to each other. Figure 2.3 simple clustering example.

Common algorithms used for clustering are k-means and k-Meanoid, hierarchical clustering, Gaussian mixture model, Spectral Clustering, Centroid-based model, Neural network, and fuzzy c-means clustering

- **Association:** Association rules are a type of unsupervised technique that help identify relationships between variables in large databases. This approach aims to uncover interesting patterns and associations among data objects. For instance, it can reveal that people who purchase a new home are likely to buy new furniture as well. Another use case is grouping shoppers based on their browsing and purchasing histories.

Other learning problems in addition to supervised and unsupervised learning include semi-supervised learning and reinforcement learning. Semi-supervised learning is a machine learning approach that utilizes both labeled and unlabeled data. It works by using labeled data to build a model, which can then be used to make predictions on unlabeled data. The goal is to improve the accuracy of the model by leveraging the large amount of unlabeled data that is often available. Reinforcement learning, on the other hand, is a type of machine learning that is focused on training agents to make decisions based on trial and error. The agent learns by receiving feedback in the form of rewards or punishments based on the actions it takes. The goal is to learn a policy that maximizes the total reward over time. Both semi-supervised learning and reinforcement learning are important areas of machine learning that have many practical applications in fields such as robotics, game playing, and natural language processing.

2.1.2 Active versus Passive Learners

The success of the machine learning process is heavily reliant on the learner during the training phase. Active learning is a machine learning approach where an algorithm can ask a user to label data interactively with the desired outputs. The algorithm selects a subset of examples to be labeled from the pool of unlabeled data. The core idea of active learning is that an ML algorithm could achieve higher accuracy while using fewer training labels if it can choose the data to learn from. Active learning involves interactive queries during the training phase, where an unlabeled data instance is presented to a human annotator for labeling. This makes active learning a powerful example of success in the human-in-the-loop paradigm. While in passive learning the learner does not need any feedback from the environment.

Passive learning is a machine learning technique where the learner does not require any external feedback or guidance from the environment during the training phase. The learning algorithm is only given a set of labeled examples to learn from, and it passively observes and absorbs the information in those examples without any further intervention. This technique is useful when the cost of obtaining feedback or the availability of such feedback is limited, and the learner must make the most of the given data to learn as much as possible. However, passive learning can be less efficient than other forms of learning that actively seek out feedback or interact with the environment to improve the learning process.

2.1.3 Helpfulness of the Teacher

The teacher is responsible for providing knowledge to the learning machine, enabling it to create a model that can accurately approximate concepts. Patrice Y Simard et al. from Microsoft has created the following component[Sim17]; interaction between these components helps to achieve appropriate knowledge sharing.

1. **Concept:** A concept refers to the relationship between a given example or input and the corresponding output or label. It is the process of mapping an input to a specific output. For instance, in a classification problem where the input is an image and the output is a label indicating the object in the image, the concept is the mapping between the image and its corresponding label. Similarly, in a regression problem where the input is the temperature and the output is the sales of ice cream, the concept is the mapping between temperature and ice cream sales. Therefore, the concept defines the relationship between input and output in a given machine-learning problem.[Sim17]

2. **Teacher:** To transfer knowledge of a concept to a learning machine, a teacher is required. The teacher's role is to provide labeled examples of data, which the learning machine can use to create a model that can predict the labels of new data. In machine learning, the teacher is often responsible for selecting the features that are most informative for the learning machine to achieve the best performance in a given task. The teacher's expertise and experience in the domain of interest are crucial in selecting appropriate features and guiding the learning process. [Sim17]

   Methods of knowledge transfer include:

   a) Example selection

   b) Labeling

   c) Schema definition

   d) featuring

   e) Concept decomposition

3. **Selection:** Selection refers to the procedure in which teachers obtain an instance that illustrates important aspects of a concept. In machine learning, selection involves the careful selection of examples that best represent the concept to be learned. These examples are then used to train the model, allowing it to learn to recognize similar patterns in new, unseen data. The effectiveness of selection can greatly impact the accuracy and generalization ability of the learning model.[Sim17]

4. **Label:** a label is the output of the process of assigning a concept value to an example by a teacher. For instance, a teacher may label a set of images of animals as either "cat" or "dog" depending on the visual features of each image. These labeled examples are then used by a learning machine to develop a model that can predict the concept value of new, unlabeled examples. The accuracy of the model's predictions is evaluated by comparing them to the true labels assigned by the teacher. In summary, labels are essential for supervised learning, as they provide the necessary feedback to the learning machine, allowing it to adjust its model and improve its performance.[Sim17]

5. **Schema:** A schema is a visual representation of the connections between different concepts. It shows how different concepts are related to each other and how they interact. For instance, in a schema of a car, the concept of an engine would be connected to other concepts such as transmission, fuel system, and exhaust system. This helps in understanding the complex relationships and dependencies between various concepts and can be used

to design more efficient and effective machine learning models.[Sim17]

6. **Features:** A feature is a characteristic or attribute of an example or object that is assigned a numerical value. Features are used to represent data in machine learning algorithms, where they play a crucial role in training models to recognize patterns and make predictions. By using generic features, machine learning models can more efficiently and accurately learn from data, as they can leverage the shared structure and patterns across different datasets.[Sim17]

7. **Decomposition:** Decomposition refers to the process of breaking down complex concepts into simpler ones. This approach involves expressing a complex concept in terms of its constituent parts, which are typically easier to understand and manipulate. By breaking down complex concepts into simpler ones, it becomes easier to analyze and learn from. In machine learning, decomposition techniques are often used to represent complex data sets in a simpler form, which can then be used for classification or prediction tasks.[Sim17]

2.1.4 Online versus Batch Learning Protocol

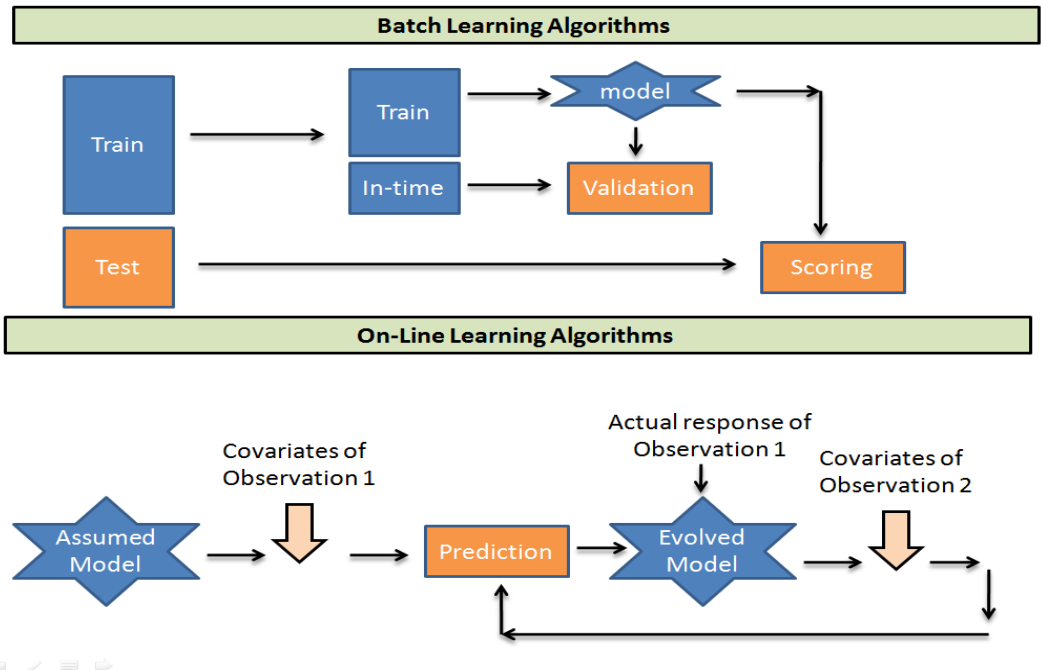


Figure 2.4: Online Verses Batch Learning [Vid15]

This machine learning distinction is based on data availability for training the model. In the Batch or Offline learning approach to train a model, a dataset is inputted into it all at once. An offline approach is often used because the system may not be capable of incremental learning, possibly due to limited computational resources in a production system. In this approach, the entire dataset is processed at once, and the model is trained on it, which may take a longer time. Once the training is complete, the model is ready for deployment and can be used to make predictions on new data. As shown in Figure 2.4 A model that has been trained and launched into production is not updated through further training. To illustrate, if a task involves classifying animals, batch learning can be used because the animal dataset is fixed and it's unlikely that new species will be added.

Online learning is a machine learning technique where data is fed to the model in small batches or as it arrives. The training happens in an incremental manner, where the model continuously updates itself to learn about new data. Each learning step is fast and cheap, which makes it suitable for scenarios where the data keeps changing frequently. For example, a machine learning model that predicts stock prices requires constant updates to incorporate changes in new data. As shown Figure 2.4, Online learning can help improve the model's accuracy over time, making it more efficient and effective. The key advantage of online learning is its ability to adapt quickly to changes, making it suitable for real-time applications such as fraud detection, spam filtering, and recommendation systems.

2.2 Neural Network

Neural networks are a widely used approach in machine learning that take inspiration from the way information is processed in a chain of neurons in the human brain. Neurons receive information through dendrites in the form of weak electrical signals, which are then transmitted to other neurons at synapses via the axon. As shown in Figure 2.5

The earliest example of an artificial neural network can be traced back to the work of an author who tried to replicate the "all or none" characteristic of neural activity by representing neural events using propositional logic. McCulloch and Pitts [MP43] proposed a model of the neuron, known as the MCP neuron, which made certain assumptions and served as the basis for the development of the threshold logic unit. The MCP neurons([MP43]) were combined to create a neural network, which was capable of performing complex tasks by processing information in a way similar to the human brain. This work was a significant milestone in the development of artificial intelligence and the inspiration for future research in the

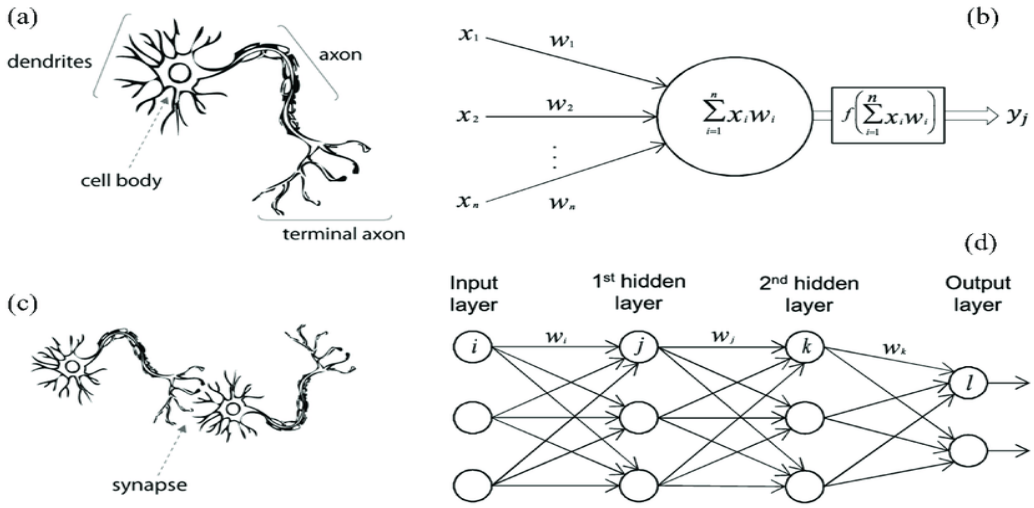


Figure 2.5: A biological neuron in comparison to an artificial neural network [MHA20]

field of neural networks. The mathematical summation of the product of a set of inputs X_1, X_2, X_3,..... X_n and their corresponding weights W_1, W_2,W_3, ... W_n are passed through the binary threshold function with threshold constant T thus the product is a single output of binary nature. However, weights in MCP model are fixed and not trainable

$$Sum = \sum_{n=1}^{N} X_i W_i \tag{2.2}$$

$$y = f(Sum) \tag{2.3}$$

Later, an improved version of the MCP model was introduced by Frank Rosenblatt [Ros61]. Although the Perceptron has a structure similar to that of the MCP model, it allows for flexible weights, which allows for trainable weights capable of learning classification tasks. The Perceptron's output is a bipolar binary[[Ros61]].

$$Sum = \sum_{n=1}^{N} X_i W_i + b \tag{2.4}$$

In addition, It is possible to stack perceptrons in order to make predictions for multiple outputs. This arrangement is called single-layer perceptron and is the motivation behind the modern machine learning process. However, as the use-case

expanded in later years single-layer perceptron (Figure 2.6)found its limitation in the implementation of the XOR gate. This motivated development of the multi-layer perception with a hidden layer between input and output as shown Figure 2.7.

2.2.1 Activation Function

The role of the activation function in neural networks is to determine the significance of a neuron's input and introduce nonlinearity for developing complex functions that cannot be achieved with a linear model. Different types of nonlinear activation functions have been proposed throughout the history of neural networks. Without activation and bias, a neural network would only be capable of solving simple linear regression problems. Activation functions can be categorized based on their properties.

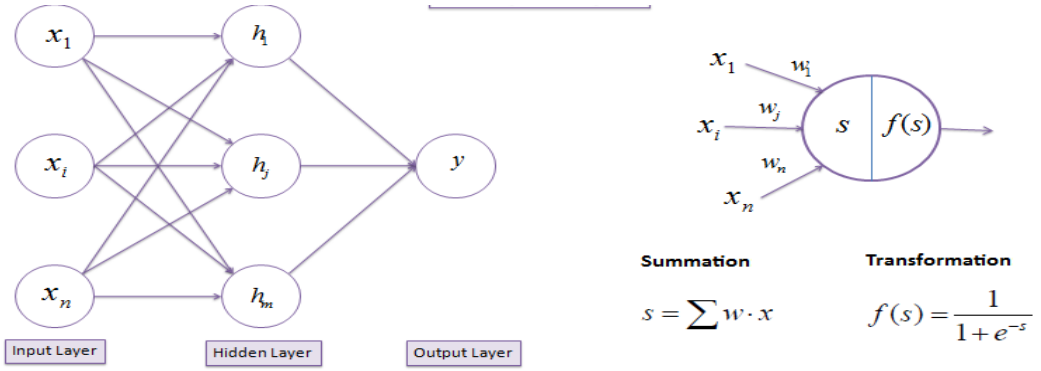


Figure 2.6: Single-layer Perceptron
[Say14]

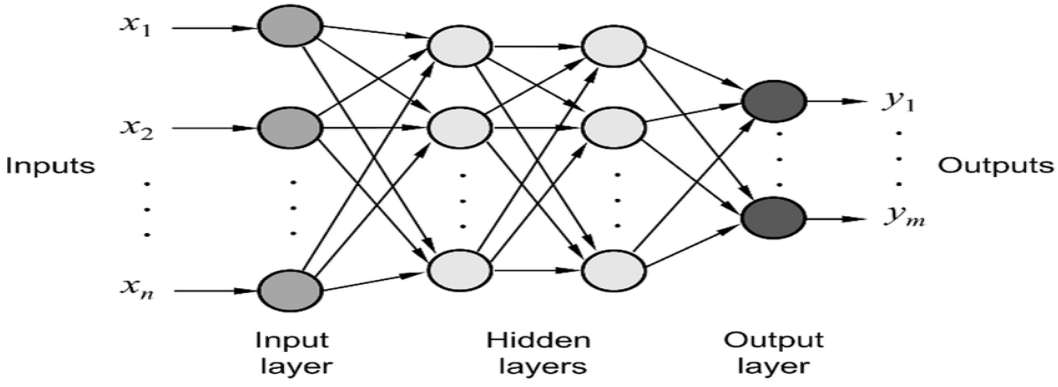


Figure 2.7: Multi-layer Perceptron
[Pou16]

- **Binary Activation function:** the binary activation function, which is also called a step function, triggers a neuron only when the input exceeds a specific threshold. In such cases, the output is binary, where it can only be either True (1) or False (0). This type of activation function is commonly used in simple binary classification problems, where the objective is to predict whether the input belongs to a particular class or not. When the input is above the threshold, the neuron fires a signal, and when it is below the threshold, it remains dormant. The threshold can be set manually or learned automatically through training the neural network. Overall, the binary activation function is a simple yet useful tool in machine learning, especially in solving binary classification problems.[Mit07; SB14; PT16]

- **Linear Activation function:** The Linear Activation Function produces an output proportional to the input. Unlike the binary output of the step function, it allows multiple outputs. However, this activation function has a limitation. If we differentiate it to introduce non-linearity, the result will be a constant value that has no association with the input value x. As a result, during backpropagation, the error rate remains constant, and the neural network's performance does not improve due to the constant gradient.[SB14; PT16]

- **Non-Linear Activation function:** To overcome the limitations of linear activation functions, non-linear activation functions were introduced. They solve these limitations in two ways: Firstly, they enable backpropagation since their derivative functions are related to the input, making it possible to identify which weights in the input neurons can improve the prediction. Secondly, non-linear activation functions allow the stacking of multiple layers of neurons, as the output is a non-linear combination of inputs that pass through multiple layers. In a neural network, because of non-linear activation functions any output can be represented as a functional computation. [Mit07; SB14; PT16]

  bellow are a few popular non-linear activation functions:

  1. Sigmoid/Logistic:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

     Range : [0,1]
     Pros: It is easy to understand and interpret the output.
     Cons: Computationally expensive, Slow conversion, vanishing gradient problem, which can slow down or even prevent the convergence of the neural network during training

2. TanH/Hyperbolic Tangent:

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.6}$$

Range:[-1, 1]
Pros: A smooth and continuous function that is differentiable, making it useful in backpropagation algorithms for updating weights
Cons: Vanishing gradient, saturation problem when its inputs are very large or very small

3. Rectified Linear Unit (ReLU):

$$f(x) = max(0) = 0 \tag{2.7}$$

Range: [0, ∞]
Pros: computationally efficient, faster convergence during training
Cons: Dying Ralu (can not accommodate for a negative value of neuron)

4. Leaky ReLU

$$f(x) = max(0) = 0 \tag{2.8}$$

Range: [0.001. ∞]
Pros: computationally efficient, Avoid the "dying ReLU"
Cons: Leaky ReLU can result in "exploding gradients" during training if the weights become too large

5. Softmax:

$$f(x_i) = \frac{e^{x_i}}{\sum_{n=1}^{N} e^{x_i}} \tag{2.9}$$

Pros: Useful for multi-class classification

2.2.2 Loss Function/Cost Function

To achieve high accuracy in Machine Learning models, it is necessary to have a method to evaluate the model's performance. A cost function is a mathematical tool used for this purpose. It helps in determining how well a model has learned the relationship between the input and output parameters. The cost function calculates the difference between the predicted output and the actual output, also known as the error or loss. The goal is to minimize this error, which can be

achieved by adjusting the model's parameters through a process called optimization. Different cost functions are used for different types of problems, such as regression or classification. Choosing the right cost function is important to ensure the model's accuracy and reliability. Therefore, understanding the concept and importance of cost functions is crucial for developing efficient and effective machine learning models.[Mit07], [SB14; PT16]

In the case of supervised learning deviation in labeled output is known as cost(also errors or loss)

$$Cost = C(y, exp) = C(W, b, \sigma, x, exp) \tag{2.10}$$

i.e. predicted outcome (**y**), weights(**W**), bias(**b**), set of activation function (σ), input(**x**) and the expected value(**exp**)

Broadly loss functions are distinguished as follows:

- **Regression Losses:** Regression losses are used in machine learning to train models for regression tasks. Primary categories for regression losses are as follows:

  1. **Mean Square Error, Quadratic loss, L2 Loss:** MSE measures the sum of squared distances between the predicted values and the target variable.

  $$MSE = \frac{\sum_{n=1}^{N}(y_i - y'_i)^2}{n} \tag{2.11}$$

  where,

  n - Number of training examples
  i - i^{th} training example in dataset
  y_i - label of i^{th} training data
  y'_i prediction of i^{th} training example

  2. **Mean Absolute Error, L1 Loss:** . MAE is the sum of absolute differences between our target and predicted variables.

  $$MAE = \frac{\sum_{n=1}^{N}|y_i - y'_i|}{n} \tag{2.12}$$

- **Classification Losses**

1. **Hinge Loss/Multi class SVM Loss:** For a correct category, its score needs to exceed the scores of all incorrect categories combined by a safety margin of typically one.

$$SVMLoss = \sum_{j \neq y_i} max(0, 1 - y_i.y'_i) \tag{2.13}$$

2. **Cross Entropy Loss/Negative Log Likelihood:** The cross-entropy loss function grows when the forecasted probability moves away from the true label.

$$CE = -(y_i log(y'_i) + (1 - y_i) log(1 - y'_i)) \tag{2.14}$$

3. **Binary Cross-Entropy with Logits Loss:** This loss combines a Sigmoid layer and the BCELoss in one single class. This version loss is numerically more stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer. Which gives the advantage of the log-sum-exp trick for numerical stability.

$$BCEWithLogitsLoss = \frac{1}{N} \sum_{i}^{N} \sum_{j}^{M} y_{i,j} \log(y'_{i,j}) \tag{2.15}$$

In YOLOv5, the default loss function used for object detection is the "Binary Cross-Entropy with Logits Loss," which is a modification of the cross-entropy loss.

2.2.3 Optimization

In machine learning, the goal is to consistently monitor the predicted outputs of a feed-forward network and reduce errors in the model. This is done by monitoring the loss function, which measures the difference between the predicted output and the actual output. To reduce this loss, the weights in the neural network are updated during an iterative process, a technique known as backpropagation. The optimization function calculates the gradient of the cost function with respect to the weights, allowing the weights to be modified in a way that minimizes the loss function. The optimization function starts from the last hidden layer and updates the weights until the initial layer is reached, resulting in a more accurate and optimized model.[[PyT]]

Some popular optimization algorithms are as follows:

- **Algorithm using Gradient Decent:** Gradient descent is a popular optimization technique for machine learning models. Its primary objective is to adjust the model parameters in a way that minimizes a given cost function. This is achieved by tweaking the parameters iteratively in the direction of the negative gradient of the cost function. The gradient is like the slope of a curve and represents the direction and rate of change in the cost function relative to changes in the model parameters. The larger the gradient, the steeper the slope, and the more rapidly the model can learn. However, if the gradient is zero, the model will stop learning. The gradient is calculated using partial derivatives with respect to the input variables, and it guides the updating of the model weights during the learning process. The goal of gradient descent is to find the optimal set of model parameters that minimize the cost function and make accurate predictions on new data. However, the gradient descent algorithm is computationally expensive since it computes all the data points in a single epoch, which leads to the evaluation of the Batch gradient descent and stochastic. gradient descent.

$$\theta = \theta - \alpha * \Delta_\theta E|J(\theta)| \tag{2.16}$$

where parameters θ of the objective J(θ)(weight minimization/derivation of cost function) and α is learning rate and Δ_θ represents gradient

1. **Batch Gradient Descent:** Batch gradient descent is an optimization algorithm used in machine learning to update the model parameters by calculating the error for each example in the training dataset. However, it updates the model only after evaluating all training examples in one training epoch. This process is cyclical, and multiple epochs are usually needed to reach the desired level of accuracy. Batch gradient descent has some benefits, such as computational efficiency, stable error gradient, and stable convergence. However, there are also some disadvantages, such as the possibility of converging to a suboptimal state, as the stable error gradient may not always represent the best possible model parameters. Another drawback is that batch gradient descent requires the entire training dataset to be loaded into memory, which can be challenging for large datasets.[SB14; PT16]

$$\theta = \theta - \alpha * \Delta_\theta J(\theta) \tag{2.17}$$

2. **Stochastic Gradient Descent:** Stochastic Gradient Descent (SGD) is a machine learning optimization algorithm that updates the model's parameters for each training example within the dataset, one by one. This can make the training process faster than Batch Gradient Descent, depending on the problem being solved. One advantage of SGD

is that the frequent updates allow us to see the rate of improvement more clearly.

$$\theta = \theta - \alpha * \Delta_\theta J(\theta : x_i, y_i) \tag{2.18}$$

X_i and y_i from training dataset

However, frequent updates can be more computationally expensive than the Batch Gradient Descent approach. Also, the frequent updates may produce noisy gradients, which may cause the error rate to jump around instead of slowly decreasing. This can make it difficult for the model to converge to an optimal solution.

3. **Mini batch Gradient Descent:** Mini-batch gradient descent is a popular optimization algorithm that combines the benefits of both stochastic gradient descent (SGD) and batch gradient descent. It involves splitting the training dataset into smaller batches and then updating the model parameters based on the average gradient of each batch. This approach strikes a balance between the computational efficiency of batch gradient descent and the robustness of SGD. By updating the model parameters after processing a batch of examples, mini-batch gradient descent achieves faster convergence compared to batch gradient descent and is less prone to noisy gradients compared to SGD.

In this research work, YOLOv5 object detection is employed and the optimization algorithm used in YOLOv5 is Stochastic Gradient Descent (SGD). SGD updates the model parameters using a single or a few randomly selected training examples in each iteration, which reduces the computational cost compared to the batch gradient descent approach. The stochastic nature of SGD can sometimes result in noisy gradients, causing the error rate to jump around instead of decreasing smoothly. However, this issue can be addressed by adjusting the learning rate and implementing techniques such as momentum or adaptive learning rate. Overall, SGD is an effective optimization algorithm for YOLOv5 object detection, allowing the model to learn and improve through iterations while minimizing the loss function.

- **Adaptive learning Algorithm:**

  Adaptive learning algorithms are optimization methods for neural network training that can adjust the learning rate, momentum, or other hyperparameters during the training process. The goal is to improve the efficiency of the optimization and achieve better performance.[SB14; PT16]

  1. **Adam** The most common adaptive learning algorithm is called Adaptive Moment Estimation (Adam), which is a combination of two other

optimization algorithms, stochastic gradient descent with momentum and RMSprop. Adam is designed to work well with large datasets and high-dimensional parameter spaces, which are common in deep learning. The Adam algorithm computes individual adaptive learning rates for different parameters by using estimates of first and second moments of the gradients. The first moment is the mean of the gradients, and the second moment is the variance of the gradients. The learning rate is then scaled by the first moment, and the second moment is used to perform normalization. This helps to make the optimization more efficient and less sensitive to the choice of hyperparameters.

Adam also incorporates bias correction, which helps to overcome the initialization bias problem that can occur in the early stages of training. This is achieved by using a moving average of the moments rather than the moments themselves.

Other adaptive learning algorithms include Adagrad, Adadelta, and Nadam. Adagrad adapts the learning rate for each parameter based on the historical gradient information. Adadelta is similar to Adagrad but uses an estimate of the second moment of the gradients to determine the learning rate. Nadam is a combination of Nesterov Accelerated Gradient (NAG) and Adam, which adds the momentum feature of NAG to Adam to help the optimization process.

Overall, adaptive learning algorithms are powerful tools for optimizing neural network training, and they have been shown to achieve state-of-the-art performance on many deep learning tasks.

2.3 Computer Vision

Computer vision is an aspect of artificial intelligence that enables machines to understand and interpret visual inputs, such as images and videos, just like humans do. Its goal is to teach machines to recognize objects, detect motion, and understand their surroundings through digital images. However, unlike humans, who learn to recognize objects and their features over a lifetime, computer vision algorithms use cameras, data, and machine learning techniques to rapidly learn and improve their abilities. By processing large amounts of data, computer vision algorithms can quickly detect patterns and make informed decisions based on them. One of the main advantages of computer vision is its ability to process large amounts of data quickly and accurately.[IBM23a]

To teach computers to recognize and understand visual inputs, such as images or videos, computer vision requires large amounts of data. The data is analyzed

repeatedly until differences between objects and features can be identified, and ultimately the computer can recognize images. For instance, to teach a computer to recognize car tires, it must be fed massive amounts of tire images and tire-related objects so that it can learn the differences and recognize a tire, even one with no defects.[IBM23a]

There are two main technologies used in computer vision: deep learning and convolutional neural networks (CNNs) (section 2.4). Deep learning is a type of machine learning that involves training a computer to learn from large amounts of data by building complex, layered neural networks. CNNs, on the other hand, are a specific type of deep learning architecture designed to analyze visual inputs by processing data in a way that mimics the human brain's visual cortex. By using deep learning and CNNs, computers can learn to recognize and understand visual inputs with high accuracy and speed.

2.4 Convolutional Neural Network(CNN)

Convolutional neural networks (CNNs) were first introduced by By author [LeC89] in the early 1990s. Their work described a system that could recognize handwritten digits from the US Postal Service's Zip Code data set with high accuracy using a convolutional neural network. Since then, CNNs have become one of the most popular and widely used deep learning architectures in the field of computer vision.

2.4.1 Convolutional neural networks (CNN) Architecture

Convolutional neural networks (CNN) are specialized types of neural networks that perform exceptionally well when given image, speech or audio inputs. CNNs are composed of three types of layers: convolutional, pooling, and fully-connected (FC) layers. The first layer of a CNN is a convolutional layer, and subsequent layers can either be additional convolutional or pooling layers. The last layer of a CNN is always a fully-connected layer. Each layer of a CNN increases in complexity, with earlier layers identifying simple features such as colors and edges, and later layers identifying larger elements or shapes of an object until it recognizes the intended object. The convolutional layer is responsible for detecting patterns in the input image using filters, while the pooling layer is used to reduce the size of the feature maps. Finally, the fully-connected layer takes the output from the previous layer and produces the final output of the network.[PT16] [Yam18]

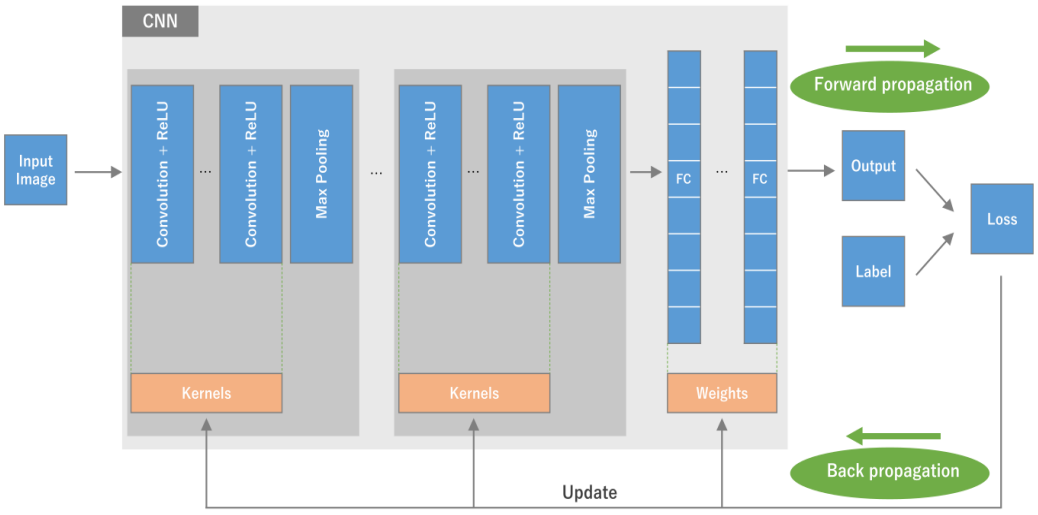


Figure 2.8: Overview of CNN Architecture and training Process [Yam18]

Convolutional Layer

The Convolution layer is a fundamental component of the CNN architecture that is typically placed after the input layer. It uses mathematical operations like convolution and activation functions to combine linear and non-linear functions. By applying a set of filters to the input data, the Convolution layer is able to identify specific patterns and features in the data. The activation function is then applied to transform the output of the Convolution layer in a non-linear way. By stacking multiple Convolution layers in a CNN, the network can recognize increasingly complex patterns and features in the input data.

In the true mathematical sense, convolution is a principle that defines how two functions overlap to produce a third function. It involves a mathematical operation that combines the input functions, allowing us to obtain a new function that describes their interaction. Convolution is widely used in signal processing and image analysis to extract features from data by applying a convolutional filter or kernel to it. By convolving two functions together, we can analyze their relationship and extract meaningful information from their interactions.

Pooling Layer

Pooling layers, which are also referred to as downsampling layers, are used to perform dimensionality reduction, reducing the number of parameters in the input[Yam18]. The pooling operation is similar to the convolutional layer in that

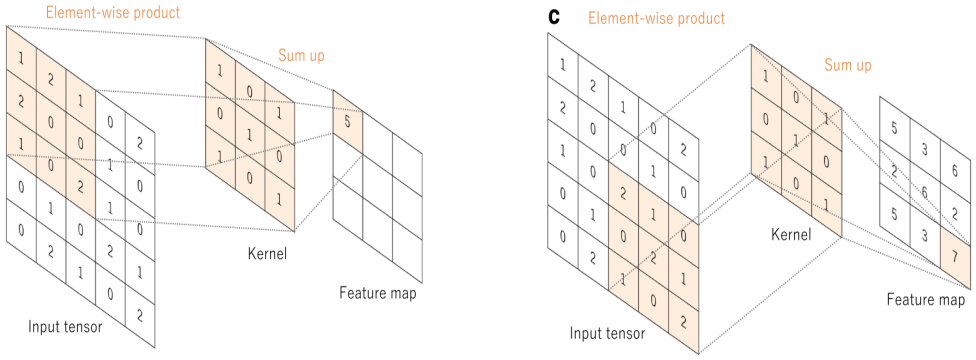


Figure 2.9: Convolutional Operation with kernel of size 3x3, no padding and stride 1
[Yam18]

it applies a filter to the entire input. However, unlike the convolutional layer, the pooling filter does not have any weights. Instead, the filter applies an aggregation function to the values within its receptive field, and the resulting output array is populated with these aggregated values. There are two primary types of pooling:

- **Max Pooling:** Max pooling is achieved by sliding a window of size n*n across the feature map and selecting the maximum value from the elements present in the window.
- **Global Average pooling:** is achieved by sliding a window of size n*n across the feature map and selecting the average value from the elements present in the window.

Fully Connected Layer

A fully connected layer (Figure 2.7) is a type of feed-forward network that is utilized to convert the downsampled output from the pooling layer into a flattened one-dimensional array of numbers, regardless of whether the output was originally 3D or 2D. A fully connected layer consists of neurons that are arranged in layers, and each neuron in a layer shares weights with every other neuron in that layer. After passing through the fully connected layer, the final layer uses the softmax activation function, which is used to obtain the probability of the input belonging to a particular class, rather than the ReLU function which is commonly used in other layers of the CNN.

2.5 Object Detection

Object detection is a computer vision technique that involves detecting and localizing objects within an image or video. The localization aspect of object detection refers to identifying the exact position of an object or objects within an image, typically through the use of bounding boxes that surround the object(s). This is distinct from image classification or recognition, which involves identifying the class or category of an entire image or an object within an image, rather than identifying its precise location. For instance, object detection could be used to identify and locate cars within an image of a busy street, while image classification might simply classify the entire image as a "street scene".

Object detection can be classified into two categories: machine learning-based and deep learning-based approaches.

- **Machine Learning Based:** In traditional machine learning-based approaches, computer vision techniques are used to identify certain features of an image, such as color histogram or edges, and groups of pixels that may belong to an object. These features are then fed into a regression model that predicts the location of the object along with its label.
- **Deep Learning Based:** Deep learning-based approaches use convolutional neural networks (CNNs) (section 2.4) to perform unsupervised object detection. CNNs have the ability to automatically learn features from images, eliminating the need for feature extraction. The end-to-end approach of deep learning-based methods allows for a more accurate and efficient detection process, as the network is trained to detect objects in an optimized manner.

2.5.1 Method Selection

To choose the most suitable approach for object detection, it is necessary to consider the specific application and problem at hand. Machine learning and deep learning are two approaches that can be used for object detection. The decision between them is influenced by the availability of resources such as a powerful Graphics Processing Unit (GPU) and a large number of labeled training images. If these resources are not available, then a machine-learning approach might be the better option. Deep learning techniques are more effective when more images are available for training, and when GPUs can be used to decrease the time needed to train the model. In summary, the choice between machine learning and deep learning depends on the specific needs and resources of the project. In this research work, Deep Learning Based approach is selected and

used.

2.5.2 Working of Deep learning Approach

In deep learning-based object detection, there are usually two components to the model. The first is an **encoder** that takes an image as input and uses multiple layers to learn important statistical features that help locate and classify objects within the image. The second component is a **decoder** that uses the output from the encoder to predict the labels and bounding boxes for each detected object. The simplest type of decoder is a regressor, which directly predicts the location and size of each bounding box. However, this type of model has limitations since you need to specify the number of objects to predict in advance. If there are more or fewer objects in the image than expected, some objects may go undetected. A more flexible approach is the region proposal network, where the model proposes regions of the image where objects may be located. These regions are then fed into a classification subnetwork to determine if they contain an object or not. This method allows for arbitrary numbers of regions that may contain a bounding box, but it is computationally more expensive than the regressor approach. When choosing a decoder, the number of objects in an image and the computational resources available are key considerations.

Single shot detectors (SSDs) take a different approach than pure regressors or region proposal networks in object detection. SSDs use a set of predetermined regions, instead of a subnetwork, to locate objects within an image. These regions are determined by laying a grid of anchor points over the input image and creating boxes of various shapes and sizes around each point. For each box at each anchor point, the model predicts whether an object exists in the region and adjusts the location and size of the box to fit the object. Since there are multiple boxes at each anchor point and anchor points may be close together, SSDs produce many potential detections that overlap. To eliminate redundant detections, a post-processing technique called non-maximum suppression is applied to SSD outputs. This technique removes overlapping bounding boxes by selecting the box with the highest confidence score for a given object and suppressing all other boxes that overlap with it. This approach strikes a balance between accuracy and efficiency, making SSDs a popular choice for real-time object detection applications.

2.5.3 Evaluation

To evaluate the performance of an object detection model, it needs to determine how accurately it is able to identify and locate objects in an image. The accuracy of the object location is typically measured using a metric called intersection-over-

union (IOU). This metric calculates the overlap between the predicted bounding box and the ground truth bounding box. The intersection is divided by the union of the two bounding boxes, producing a value between 0 (no overlap) and 1 (perfect overlap). A high IOU score indicates that the model has accurately located the object.

To evaluate the accuracy of the object label, a simple "percent correct" metric can be used. This involves comparing the predicted labels with the ground truth labels and calculating the percentage of correct predictions. However, this metric may not be appropriate in all cases, especially when dealing with imbalanced classes or when certain classes are more important than others. In such cases, other metrics such as precision, recall, and F1 score may be more suitable.

2.6 YOLOv5

YOLO is an object detection algorithm that operates in real-time and recognizes different objects in an image. It is based on regression analysis and provides class probabilities for detected images. YOLO uses convolutional neural networks (CNN) to achieve real-time object detection. Unlike other object detection algorithms, YOLO requires only one forward propagation through the neural network to detect objects in the entire image. The CNN predicts class probabilities and bounding boxes simultaneously, making it faster than other object detection techniques. YOLO comes in various variants and has become popular because of its speed, high accuracy, and learning capabilities. Its speed is due to the ability to predict objects in real-time. YOLO provides accurate results with minimal background errors, making it a popular choice in object detection. Additionally, it has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.[Li22]

The YOLO algorithm works using three techniques: residual blocks, bounding box regression, and intersection over union (IOU). Residual blocks are used to learn the features of objects in the image, bounding box regression is used to predict the location and size of objects, and intersection over union (IOU) is used to evaluate the accuracy of the algorithm's predictions. Overall, YOLO is an important algorithm in object detection due to its speed, accuracy, and learning capabilities.

2.6.1 YOLOv5 Architecture

Single-stage object detectors (like YOLO) architecture are composed of three components: A backbone, a Neck, and a Head to make dense predictions(Figure 2.10).

Bellow explained the high-level architecture of the YOLOv5

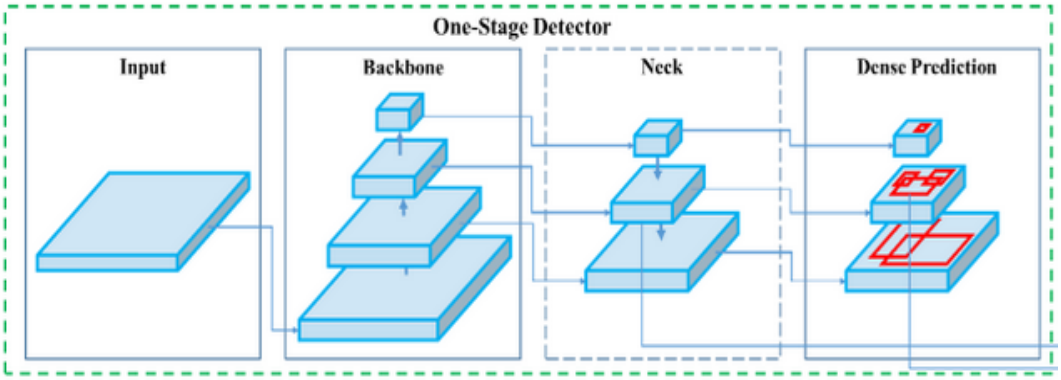


Figure 2.10: YOLOv5 Architecture
[IQ21]

- **Model Backbone:** The backbone of a neural network is a pre-trained model that is used to extract important features from images. These features are then used by the model to make predictions about the objects present in the image. The backbone network works by reducing the spatial resolution of the image while increasing its feature resolution, making it easier for the model to identify important features. This is achieved through a series of convolutional layers that apply filters to the image, extracting information about edges, corners, textures, and other features that are important for object detection. The backbone network is typically pre-trained on a large dataset, such as ImageNet, and can be fine-tuned for a specific object detection task. By using a pre-trained backbone network, the model can benefit from the knowledge and insights gained from previous training on large datasets, reducing the amount of time and resources needed to train a new model from scratch.[IQ21]

- **Model Neck:** The ”neck” component of a single-stage object detector model, such as YOLOv5, is responsible for extracting feature pyramids. A feature pyramid is a set of features maps of the same image at different scales. This allows the model to effectively detect objects of different sizes and scales, which is crucial in real-world scenarios where objects can appear in various sizes and orientations. The neck component typically employs different techniques such as FPN (Feature Pyramid Networks) and PAN (Path Aggregation Networks) to extract these feature pyramids. By using a neck component in the model, the model can generalize well and improve its accuracy in detecting objects of varying sizes and scales.[IQ21]

- **Model Head:** In the YOLOv5 architecture, the model head is responsible for the final stage operations. It takes the feature maps produced by

the neck and applies anchor boxes on them to generate the final output. The anchor boxes are pre-defined bounding boxes of different sizes and aspect ratios that are placed at different locations on the feature maps. The model head adjusts the anchor boxes to fit the objects in the image by predicting their class probabilities, objectness scores, and bounding box coordinates.[IQ21]

The class probabilities represent the likelihood of each object belonging to a particular class, while the objectness scores indicate the likelihood of an object being present in the given location. The bounding box coordinates specify the location of the object in the image. The model head performs these operations in a dense and efficient manner, allowing for real-time object detection.

2.6.2 Activation Function

The choice of activation function is an important decision when building a deep learning model, and for YOLOv5, the authors decided to use both SiLU and Sigmoid activation functions. SiLU, or Sigmoid Linear Unit, is also known as the Swish activation function. It is applied to the convolution operations in the hidden layers of the YOLOv5 model. SiLU is a smooth and nonlinear activation function that is easy to compute, and it has shown to perform better than other activation functions such as ReLU in some cases.

On the other hand, Sigmoid activation function is used with the convolution operations in the output layer of the YOLOv5 model. Sigmoid is a mathematical function that maps any input value to a value between 0 and 1. It is commonly used to model the probability of a certain event occurring. In the context of object detection, the output of the YOLOv5 model will provide the probability of an object belonging to a certain class, and Sigmoid is used to convert this output into a probability value between 0 and 1.

Overall, the combination of SiLU and Sigmoid activation functions in YOLOv5 contributes to the model's high accuracy and efficient performance in real-time object detection.

2.6.3 Loss Function

YOLOv5 is an object detection algorithm that produces three outputs: the classes of the detected objects, their bounding boxes, and the objectness scores. To compute the loss, the algorithm uses three different loss functions. For the classes and objectness, it uses Binary Cross-Entropy with Logits Loss (BCEL), while for

the location loss, it uses Complete Intersection over Union (CIoU). The final loss is calculated as the sum of these three losses, each multiplied by a weighting factor. The weights are determined empirically and are used to balance the contribution of each loss to the final loss. The overall goal of computing the loss is to minimize it during training so that the algorithm can learn to accurately detect objects in images.

2.7 Robotic Operating System 2(ROS2)

The Robot Operating System, or ROS, is a collection of open-source software tools, hardware drivers, and algorithms that are designed to facilitate the development of robot control software. Despite its name, ROS is not an operating system in the traditional sense. Instead, it provides a set of communication systems, frameworks, and tools that enable developers to build and manage complex robot applications.

The core features of ROS include a communication system that uses publish-subscribe and remote method invocation protocols, as well as a framework for managing dependencies, building software, and visualizing data. Additionally, ROS provides an ecosystem of libraries, drivers, and simulation tools such as Gazebo, which enable developers to build complex robotic systems more easily.

One of the key benefits of ROS 2 is its distributed real-time system architecture. This architecture allows different components of a robot, such as sensors, motion controllers, detection algorithms, and navigation algorithms, to be treated as separate nodes in a distributed system. These nodes communicate with each other using a middleware called the Data Distribution System (DDS), which is integrated into ROS 2. DDS provides a reliable and efficient way for nodes to exchange data in a distributed environment, enabling robots to operate seamlessly in complex, real-world environments.

- **Nodes:** In ROS 2, nodes communicate with each other through various mechanisms such as topics, service invocation, and actions. Topic-based communication follows a publish-subscribe architecture, where a node publishes data to a topic, and multiple nodes can subscribe to that topic to receive the data. Similarly, a node can also subscribe to multiple topics to receive data from different nodes.

  ROS 2 topics are built on top of DDS topics, which are a standardized way of exchanging data between distributed systems. This ensures that ROS 2 can work seamlessly with other DDS-compliant systems.

  Service calls, on the other hand, use a remote method invocation approach

where a node requests a service from another node, and the requested node provides a response. The service can be either synchronous or asynchronous. ROS 2 also provides the concept of actions, which are used for long-running service calls. In this case, the client sends a request with a goal to the action server, which then performs a long-running process and publishes intermediate results as it progresses toward the goal. When the goal is achieved, the server reports the final result to the client.

Both services and actions are implemented using topics over the DDS middleware, which provides a reliable and efficient way for nodes to communicate with each other in a distributed environment. This communication mechanism allows different nodes to work together to accomplish complex tasks, enabling robots to perform a wide range of functions in real-world scenarios.

- **Topic:** ROS 2 nodes communicate with each other using three main mechanisms: topics, service invocation, and actions. Topic-based communication follows a publish-subscribe architecture, where data produced by a node is

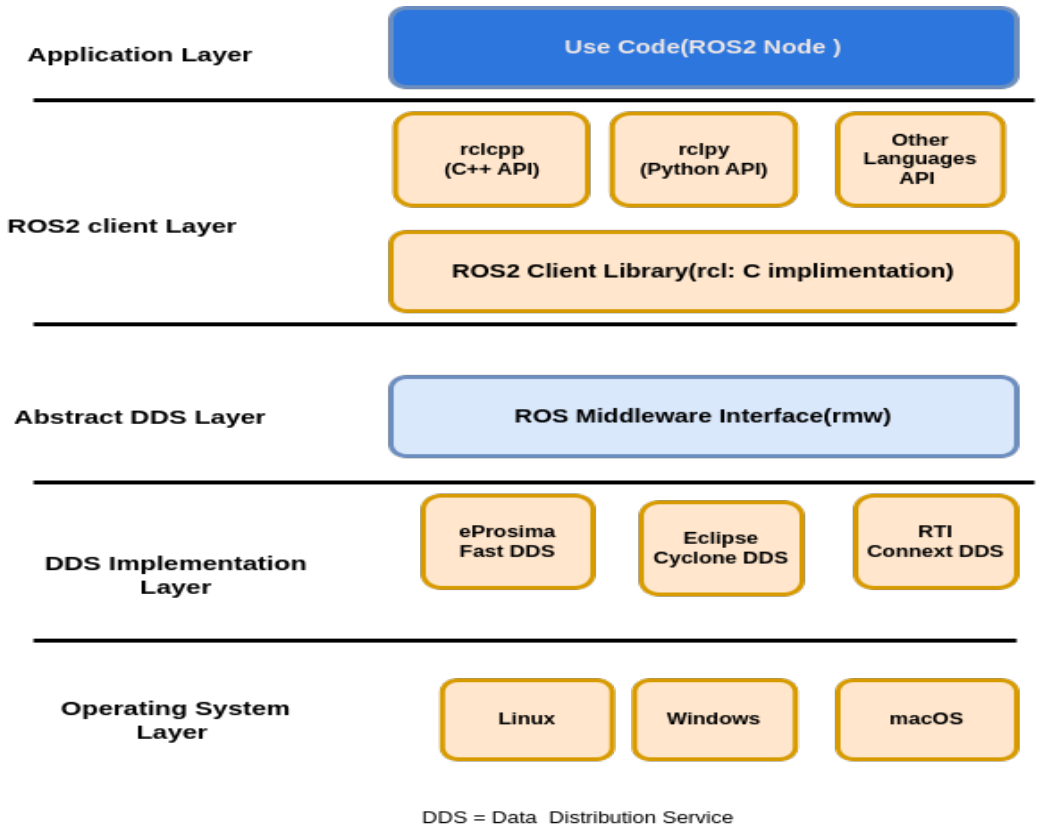


Figure 2.11: ROS2 Architecture
[Add21]

published to a topic and can be subscribed to by multiple nodes. Similarly, a node can subscribe to multiple topics to receive data from different nodes. It's important to note that ROS 2 topics are compatible with DDS topics, ensuring that ROS 2 can work seamlessly with other DDS-compliant systems.

Service calls in ROS 2 use a remote method invocation approach where a node requests a service from another node, and the requested node provides a response. Actions are used for long-running service calls, where the client sends a request with a goal to the action server, which then performs a long-running process, publishes intermediate results, and reports the final result when the goal is achieved.

The use of a decentralized architecture of topic structure provides several advantages in terms of performance and fault tolerance. Additionally, the rich quality of service features provided by the DDS middleware has also played a significant role in the success of the ROS 2 architecture. These features ensure that nodes can communicate with each other reliably and efficiently in a distributed environment. This enables robots to perform complex tasks while adapting to different hardware configurations and operating environments.

The decentralized architecture of the Topic structure creates advantage in terms of both performance and fault tolerance. In addition, rich quality of service features provided by the DDS middleware have also been a factor that raised the ROS 2 architecture to higher levels.

- **Messages:** In ROS 2, messages are used to transmit data between nodes via topics. These messages are data structures that correspond to DDS data types, ensuring compatibility between ROS 2 and other DDS-compliant systems. Standard messages are particularly important for interoperability, as they enable different devices from various manufacturers to communicate with each other using a common language. For example, cameras from different companies can be integrated into the system and communicate with standard messages regardless of the camera model. The node component developed for a particular camera converts the camera's special communication protocol messages into standard DDS data to enable seamless communication with other nodes in the system.

2.8 Docker

Docker is a platform that allows developers to create containers that are self-contained units of software that can run on any system, regardless of the under-

lying operating system or hardware. Containers are made up of the application source code and all the dependencies required to run that code. Containers created with Docker are portable, meaning they can be run on any system that supports Docker, regardless of the underlying infrastructure. This makes it easy to move applications between different environments, such as development, testing, and production.

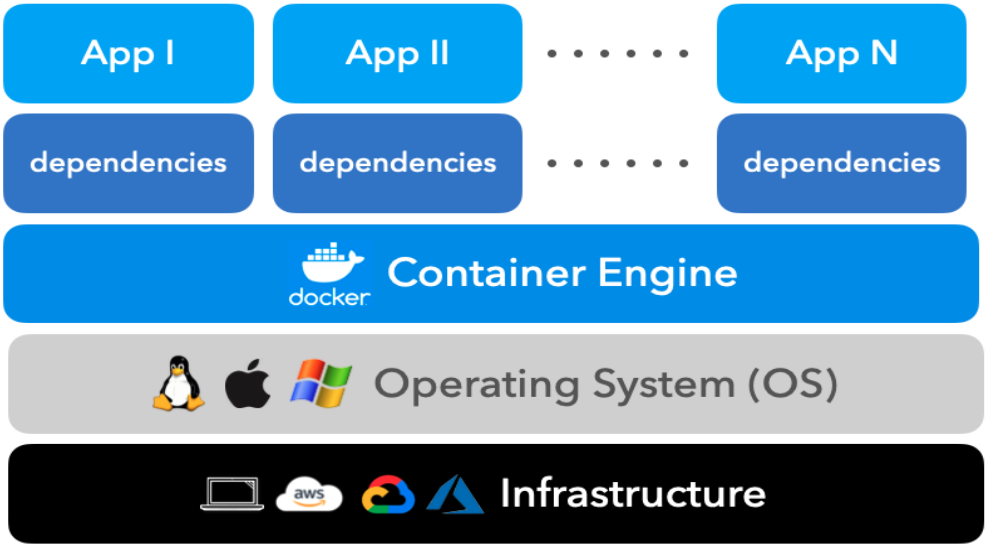


Figure 2.12: Docker Container
[IBM23b]

Containers are possible because of the process isolation and virtualization features that are built into the Linux kernel. These features, such as control groups (Cgroups) and namespaces, allow multiple application components to share the resources of a single instance of the host operating system in the same way that virtual machines (VMs) share the resources of a single hardware server. These capabilities enable containers to offer many of the same benefits as VMs, such as application isolation, cost-effective scalability, and disposability. However, containers offer additional advantages, such as faster startup times and less overhead, because they do not require a separate operating system for each container. Overall, container technology allows developers to create and manage isolated environments for their applications while sharing the resources of the host operating system. This results in more efficient and cost-effective use of resources and improved application performance. [IBM23b]

Docker simplifies the process of building and managing these containers by providing an easy-to-use interface. Developers can use Docker to package their applications and dependencies into self-contained images. These images can then

be shared and run on any system with Docker installed. Docker is open-source, which means that it is free to use and can be modified and improved by developers around the world. It has become a popular tool for modern application development and deployment due to its ability to simplify the process of building, shipping, and deploying complex applications and infrastructure.[Dev21]

Docker containerization is compatible with Microsoft Windows and Apple MacOS in addition to Linux. Developers are able to run Docker containers on any operating system, and popular cloud providers like Amazon Web Services (AWS), Microsoft Azure, and IBM Cloud offer specialized services to assist developers with building, deploying, and running Docker containerized applications.(As illustrated in Figure 2.12)

Following some terms and tools used for docker containers and their use will be explained in the Methodology chapter 4

- **DockerFile:** To create a Docker container, a basic text file is needed with instructions on how to build the container image. This process can be automated with DockerFile, which is essentially a set of command-line instructions that Docker Engine uses to construct the image. The Docker commands are standardized and work in the same way, regardless of the environment, infrastructure, or other variables. While there is a large list of Docker commands available, they all function uniformly.[IBM23b]
- **Docker Images:** Docker images include the application source code and all the necessary tools, libraries, and dependencies for running the application as a container. Running the Docker image creates an instance or multiple instances of the container. While it is possible to create a Docker image from scratch, most developers prefer to download them from common repositories. Using a single base image, developers can create multiple Docker images that share the same underlying stack.[IBM23b]
- **Docker Containers:** Docker containers are live and operational versions of Docker images. While Docker images are static files, containers are active, temporary, and executable content. Users can interact with containers, and administrators can modify their settings and conditions using Docker commands.[IBM23b]
- **Docker Hub:** Docker Hub is a public repository that stores various Docker images. The repository is maintained by Docker, Inc. and contains both official images created by Docker, Inc., as well as other images contributed by the wider community. In addition, Docker Hub also offers certified images that belong to the Docker Trusted Registry. These certified images undergo a rigorous testing and verification process to ensure that they meet specific quality standards, security measures, and compatibility requirements.

[Doc23]

2.9 Model Optimization

To use a deep learning model, it requires significant computational resources such as CPU, GPU, memory, and power. As previously discussed, deep learning models can be trained for tasks such as face recognition and object detection. In the modern scenario, computer vision and language processing applications are also being used on resource-constrained devices such as mobile devices, IoT, and other embedded devices. Our use case involves deploying a pre-trained model on an embedded device. A deep learning-based model can be compressed using two techniques

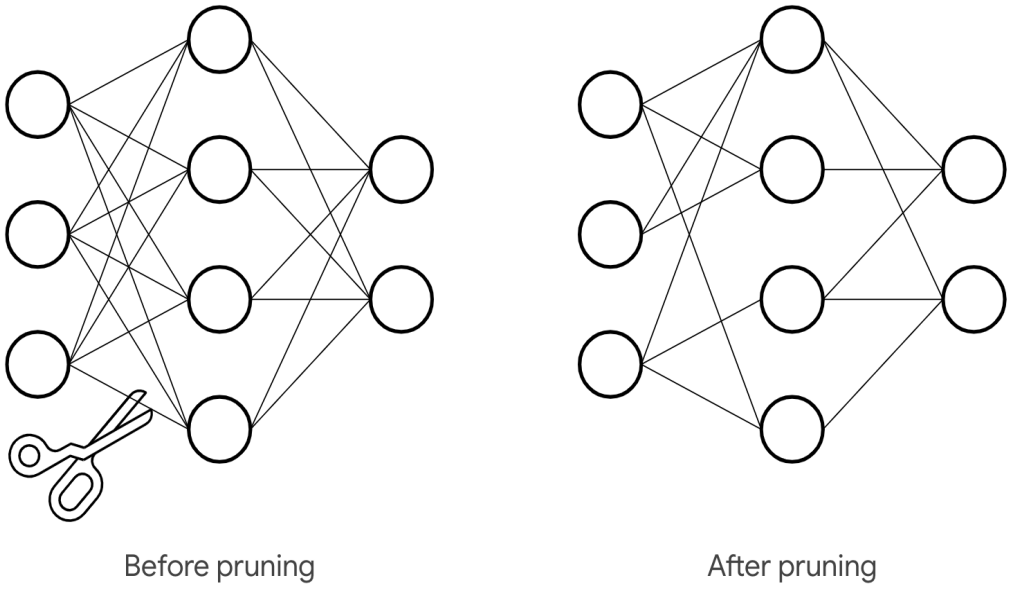


Figure 2.13: Pruning
[TF19]

- **Pruning:** Pruning is a technique used to reduce the size of a trained neural network by removing connections between neurons, channels, or filters. This is achieved by setting the weights of certain connections or groups of weights to zero, effectively removing them from the network. The idea behind pruning is that neural networks are often too complex and contain redundant connections, which can be removed without significantly impacting their performance. Pruning can be either unstructured or structured, depending on whether individual weights or entire channels/filters are removed from the network. Unstructured pruning involves removing individual weights

or neurons, while structured pruning involves removing entire channels or filters.[TF19]

- **Quantization:** To reduce the storage required for model weights, a technique is used to decrease the number of bits used to store them. The advantage of this technique is that it does not require retraining of the pre-processed model. Further details are provided below.[TF19]

In this research work, we are utilizing TensorRT to optimize the model. TensorRT is a high-performance deep learning inference optimizer and runtime engine developed by NVIDIA. The engine optimizes the model by quantizing the weights, which means reducing the number of bits used to represent the weights of the model. This quantization process reduces the model size and memory usage, thereby improving its performance on resource-constrained devices. We have chosen TensorRT to perform the quantization process because it offers an efficient and optimized platform for the deployment of deep learning models on GPUs, particularly for real-time applications such as object detection and recognition. TensorRT offers automatic precision calibration, which helps in minimizing accuracy loss during the quantization process. Overall, the use of TensorRT in the research work allows for efficient model compression and deployment on resource-constrained devices. Detailed explained of TensorRT is given in thesection 4.4

3 Related Work

This chapter discusses YOLOv5-based object detection models with different variations and optimized TensorRT models and techniques used for the deployment of object detection models on edge devices, which is a fundamental aspect of this work. Firstly, in section 3.1 discusses and acknowledges the work done in the YOLO object detection model and different variations of the model. section 3.2 discusses and acknowledges the work done on the TensorRT model optimization. And section 3.3 discuses and acknowledges the work done in the ROS2 framework.

3.1 Object Detection And YOLO Darknet

As stated in the introduction, AI has become a tool for enhancing safety in construction sites by using camera sensors to detect objects, people, and other elements. Specifically, this section will provide a brief overview of the application of deep learning's YOLOv5 algorithm in object detection.

[ZZN21] has done examination of a YOLOv5-based safety helmet detection method, with a focus on the YOLOv5 network architecture, classifier configuration, and dataset processing. Tests were conducted using YOLOv5 models of different sizes (s, m, l, x) with varying parameters, resulting in exceptional accuracy and speed. The use of pre-trained weights resulted in a 1% improvement in the mAP values of the models. In future work, the focus will be on collecting a substantial number of positive and negative samples from real-world scenarios to increase the network's generalizability.[ZZN21]

[Dua22] has done some work in development dataset called SODA for object detection on construction sites. SODA is a VOC-format image dataset with 19,846 images and annotation information for 286,201 objects, covering 15 categories of common construction objects. It was tested on two popular one-stage object detection algorithms, resulting in training results and benchmarks. SODA makes several contributions to the field of construction industry object detection. Firstly, the SODA dataset is designed specifically for the construction industry and features high-quality image data collected from real construction sites, resulting in improved practicality and performance compared to general datasets. It also contains more images and categories than existing construction datasets.

Secondly, the study provides a comprehensive analysis of the dataset, including statistical analysis of images and objects, image resolution and perspective, clustering analysis of bounding boxes, and box-plot analysis of pixel length and width. A comparison with other construction datasets is also included. Finally, the effectiveness of deep learning-based object detection on construction sites using SODA was tested using YOLOv3 and YOLOv4, and results showed good speed and accuracy. This study is the first to detect multiple 4M1E categories in a single object detection model and offers insights for various tasks in construction site safety and progress analysis. SODA is the first open dataset that covers a wide range of construction object categories and will continue to be updated with new categories and capabilities.[Dua22]

[JWL20] Had modified existing YOLOv3 object detection neural network and deployed it on embedded device to implement pedestrian detection on embedded device. The importance of this study is in its application of pedestrian detection on embedded platforms. The algorithm speeds up calculation time by 40%, increasing the original YOLOv3's 6 frames per second to 9-10 frames per second.[[JWL20]]

[Mur22] proposed work to improve the speed and accuracy factor which ware limited to R-CNN, Fast-RCNN, Faster RCNN, YOLOv3, and YOLOv4, YOLOv5 was selected. The YOLOv5 model was not pre-trained and aimed to include in system for assisting drivers for detecting pedestrian with significance increase in speed and accuracy.The framework consists of three main modules: extraction, detection, and visualization. The extraction module extracts features from the input, the detection module identifies and locates objects, and the visualization module provides alerts and warnings. This framework has been used to develop the Android application "ObjectDetect," which alerts the user of significant events. The application primarily uses a camera and can operate in challenging weather conditions.[Mur22]

The authors[Pen21]have introduced CORY-Net in their paper. A CORY-Net model is modified version of the YOLOv5 to use the pre-trained feature extraction encoder to detect the related object from the dataset on power grid construction site. A new object detection model, which integrates contrastive self-supervised pre-training and YOLOv5 for the purpose of monitoring safety on power grid construction sites. They have also created a high-quality dataset of practical power grid construction scenes, which includes 22 categories of objects that pose potential safety risks, to support the training of CORY-Net. The experimental results showed that CORY-Net requires less annotated training data and performs better than the supervised model on the power grid construction dataset, due to the contrastive self-supervised pre-training. In comparison to YOLOv5, CORY-Net improves the mAP@.5 by 1.32% and mAP@.5:.95 by 3.54%. Furthermore, CORY-Net outperforms the state-of-the-art supervised models in terms of

mAP@.5 for most categories of objects.[Pen21]

[Liu22] has came across the problem of detection pedestrians in the adverse weather condition such as fogy weather or rainy weather. to solve this problem they proposed image adaptive object detection method called IA-YOLO. In this study, a fully differentiable image processing module (DIP) is proposed, whose hyper-parameters are adaptively learned by a small CNN-based parameter predictor (CNN-PP). The CNN-PP predicts the DIP's hyper-parameters based on the input image's brightness, color, tone, and weather-specific information. After being processed by the DIP module, the interference from weather-specific information in the image is suppressed, while latent information is restored. A joint optimization scheme is presented to learn the DIP, CNN-PP, and YOLOv3 backbone detection network in an end-to-end manner. To improve the image for detection, the CNN-PP is weakly supervised through the use of bounding box annotations to learn an appropriate DIP. The proposed network is trained using images under both normal and adverse weather conditions. With the aid of the CNN-PP network, the proposed IA-YOLO approach adaptively handles images affected by different weather conditions. The experimental results showed that proposed method performed much better tin both fogy and low light scenarios.[Liu22]

[Ben21] has proposed a model to detect tiny objects. the proposed model was slightly modified version of YOLOv5 model. Author has methodology for modifying the structure of YOLOv5 to enhance performance for a specific task is being proposed. This is done through experimentation, where the various components of YOLOv5 are analyzed. In this study, the effects and reasoning behind various architectural and model modifications to the well-known YOLOv5 object detector have been examined in order to enhance its ability to detect small objects. The techniques have been validated in the context of autonomous racing, emphasizing the unique requirements and limitations of this scenario, and suggesting avenues for future research. The results have led to the creation of a new YOLO-Z family of models that can improve small object detection by nearly 6% (as measured by the mAP at 50% IoU) with only a slight increase in inference time of around 3 ms. These findings have the potential to upgrade existing systems to detect smaller objects in situations where current models are ineffective. This improvement can extend the detection range and perception reliability of autonomous vehicles, leading to more effective planning and decision-making strategies and providing a significant advantage in autonomous racing.[Ben21]

[Son21]has presented an improved object detection method based on YOLOv5 that enhances the accuracy of object positioning and recognition for a grasping robot. The method optimizes the depth and width of the YOLOv5 network using network pruning, resulting in a lighter network that can be applied in industrial equipment. The optimal parameters of the YOLOv5-ours(Presented model)

model were determined through experiments on the learning rate and batch size and were found to have the lowest loss function value compared to other YOLO series models. The results show that the proposed YOLOv5-ours(Presented model) model has a precision of 99.35%, recall of 99.38%, mAP value of 99.43%, and F1 score of 99.41%. Compared to the original YOLOv5s, YOLOv5m, and YOLOv5l models, the mAP of the YOLOv5-ours(Presented model) model increased by 1.12%, 1.2%, and 1.27%, respectively, and the size of the model was reduced by 10.71%, 70.93%, and 86.84%, respectively.[Son21]

[AOC22]has done transfer learning by changing light conditions. In this study, the YOLOv5 deep learning algorithm was utilized to detect two classes: Koko Krunch and Lady's Choice. A custom dataset was collected and augmented to improve accuracy in varying lighting and orientations. The model was trained using the YOLOv5s COCO pre-trained weights for 100 epochs, and the best result of mAP 0.9948 at 87 epochs was selected for validation. The optimal distance between the objects and the camera was found to be 4ft with a confidence level of 0.96 for the Koko Krunch class and 0.94 for the Lady's Choice class. The best luminance for both classes was determined to be 475lm and 600lm, respectively. The optimal orientation was found to be upside-down for the Koko Krunch class and normal for the Lady's Choice class. The results showed a 100% detection rate for both classes at all luminance levels and an average confidence level of 96% for the Koko Krunch class and 91% for the Lady's Choice class. Additionally, the real-time inference was achieved, with an average time of 1.704 seconds on the Raspberry Pi Model 4B.[AOC22]

3.2 TensorRT And Deployment

[HPC18] has done some work in optimizing YOLO algorithm to detect objects in real-time using non-GPU device. The author used YOLO-Lite model for their work. The author has used YOLOv2 and modified the model architecture with a trial and error basis at the trial 3 they changed 9 convolutional layers from YOLOv2 Tiny and a total of 3,181 filters, resulting in 6.97 billion FLOPS to 7 layers and 749 filters, resulting in 482 million FLOPS. The FLOPS of Tiny-YOLOv2 is 14 times higher compared to that of YOLO-Lite Trial 3-no batch. The reduced number of layers in YOLO-LITE contributes to its faster performance. Author came to the conclusion that YOLO-LITE is successful in enabling object detection on non-GPU computers. The system made several important contributions to the field of object detection. Firstly, it demonstrated that shallow networks have great potential for creating lightweight real-time object detection networks. The performance of YOLO-LITE at 21 FPS on a non-GPU computer was significant for such a small system. Secondly, YOLO-LITE highlights the

need to reevaluate the use of batch normalization for smaller shallow networks in the field of object detection. The advancement in this area of lightweight real-time object detection is crucial in making object detection practical in everyday use.[HPC18]

[HPC18] has provided some insight on Deep Learning inference parallelization on Heterogeneous Processors With TensorRT. In their work the have provided a methodology for the acceleration of a single application through the exploitation of various forms of parallelism including multi-threading, multi-streaming, pipeline inference network, and PND(Parallel Network Duplication) is presented. In their methodology mentioned three types of parallelization first one is pre-processing parallelization, the second one is Intra-PE Parallelization and the third is Post-process parallelization. By using one or a combination of two or a combination of all three they experimented and came to the following conclusion. In this study Author used Jetson AGX Xavier device. The Author has tested the methodology of parallelization with six actual object detection networks on an NVIDIA Jetson AGX Xavier board. The results have shown significant improvements in throughput ranging from 81% to 391% compared to the baseline inference that solely relies on the GPU. In this work, TensorRT is converted from the ONNX model. The conversion process optimizes performance through quantization and kernel auto-tuning, which takes into account the number of CUDA cores. However, a TensorRT model cannot be used on different TensorRT versions or GPU types.[HPC18]

[HPC18] and their team has worked on the deployment of the YOLOv5 model on Triton Inference Servers. In this study Authors first converted PyTorch Model into the ONNX model and Then this ONNX model was converted into TensorRT Model for Final deployment on the Triton Inference servers. IN their study they experimented with the ONNX model and TensorRT Model. ONNX acts as an intermediary representation for models from different frameworks. The authors used the ONNX module integrated into PyTorch to convert the YOLOv5 network into the ONNX format. The TensorRT library is designed for faster neural network inference and is built using CUDA technology. It only runs on Nvidia GPUs. In this study Author has observed that the TensorRT models deliver better performance in terms of throughput compared to the un-optimized ONNX model. The TensorRT fp32 model achieves a slightly higher throughput than the ONNX model due to layer fusion and kernel auto-tuning. The use of fp16 quantization is found to significantly enhance throughput, and incorporating dynamic batching results in even further improvement. The author also observed that the latency of both ONNX and TensorRT models increases with an increase in concurrency. The TensorRT fp32 model has been observed to perform faster than the non-optimized ONNX model, by 24.6%. The adoption of fp16 quantization has been seen to significantly improve the speed of inference. Furthermore, the

use of dynamic batching has been found to further reduce latency by 15.7% and 24.7% when the concurrency is equal to 30 and 100 respectively.[HPC18]

[JWL20] detailed work and experimentation of Nr [10] is explained above in section 3.1 improvement has achieved by implementing techniques through embedded devices, such as adjusting the input image resolution, although this caused image distortion. Author used Jetson TX2 which comes with better CUDA capabilities.[JWL20]

[CJW22] and the team were worked on developing an automated garbage collection system by using deep learning and robotics. The aim of their research was to use TensorRT acceleration technology which is provided by YOLOX low-cost embedded platform and achieve good results in detecting and separating the garbage. For the embedded device author used Jetson Nano 4GB for for object detection. Arduino UNO was used for the control part of the mechanical jaws and used PC For managing hardware devices and viewing real-time recognition in the browser. The team also developed their own data set for detecting the garbage which contains 17 categories. In this paper main aspects of the YOLO series are analyzed, including accuracy, speed, FLOPs, and number of parameters. The results show that YOLOX has the best mAP (97%) and the shortest inference time (0.30ms) for a single image frame. The commonly used YOLOv5 and YOLOv3 are slightly lower in terms of accuracy and speed, but YOLOX has higher memory utilization and a shorter training time, making it ideal for quick iteration of the model. When a new category of garbage needs to be added, it can be easily adapted. With the support of the TensorRT acceleration technology, the speed wad doubled to achieve real-time recognition. The model is compatible with various complex environments, enhancing the accuracy and efficiency of garbage classification.[CJW22]

[Stä21] has done a case study on the deployment of the RatinaNet model parallel with PointPillars to detect objects and localize the objects in runtime. RetinaNet is a 2D object detection algorithm based on images. It is a pioneer in the field of one-stage object detectors and has surpassed previous two-stage networks in terms of running time while maintaining similar detection performance. PointPillars is a 3D object detection network that utilizes LiDAR technology and has demonstrated state-of-the-art results on publicly accessible benchmarks. It has the ability to perform real-time inferences. they compared the runtime of the RetinaNet with TensorRT and TorchScript while considering quantization for deployment. The author had come to the conclusion that TensorRT has a lower inference time than TorchScript. The author used the NVIDIA Jetson AGX Xavier device in their study and also came to the conclusion that the runtime is also heavily influenced by the available power supply for the device. And Author made necessary modifications and tools for two object detection algorithms, RetinaNet and PointPillars, which have been analyzed and discussed. The runtime

performance of TensorRT and TorchScript was evaluated, and it was concluded that TensorRT should be used for RetinaNet and the convolutional part of PointPillars, while TorchScript should be used for the fully connected part of PointPillars. The runtime of both algorithms can be reduced through quantization, with TensorRT supporting Int8 and TorchScript supporting Float16. The impact of quantization on detection performance was found to be low for RetinaNet with Int8, while PointPillars should only be quantized to Float16. The design parameters of the algorithms were also studied, and a good runtime-performance trade-off was found with an input resolution of 576x1024 for RetinaNet and 12000 pillars with 24 points per pillar for PointPillars. The available power supply in the deployment environment was noted as having a significant impact on runtime and must be considered when choosing a setup.[Stä21]

3.3 ROS2

[Rek20] has presented an architecture for a self-driving car that utilizes ROS2. The architecture is designed to handle real-time decision-making based on sensory input, ensuring high reliability and strong functional safety requirements. This paper describes an architecture for self-driving vehicles that is designed to be largely vehicle independent. The design takes into consideration the requirements for real-time performance and calculation capabilities. To address the differences in control interfaces for steering, acceleration, and braking, the architecture introduces a wrapper node known as ROS2Car. Nodes for standard tasks in automated driving are also defined. The architecture is evaluated in various scenarios to assess its usability and variability. The first evaluation results are reported in the paper.[Rek20]

[Puc20] has conducted study on distributed and synchronized setup for real-time robot control using ROS2. In this study, the operating system is prepared and evaluated to meet real-time requirements. The robustness of the system is established through the examination of various benchmark scenarios. Multiple evaluation methods are utilized to account for different perspectives and reduce the chance of missing critical configurations. The long-term evaluation is conducted by measuring response time and communication latencies through the use of directly addressed GPIOs(General Purpose Input Output).In this research, it has been established that the Linux network stack may cause non-deterministic latencies and jitter. It is therefore recommended to avoid overloading the local network and to use a dedicated network for time synchronization with PTP(Precision Time Protocol) when feasible. The study showcases the configuration and versatile validation of real-time capabilities using consumer-based hardware and open-source software. To validate the robustness of the system, even under extreme

stress and network conditions, external measurements are conducted using the serial ports of the hosts as digital outputs to assess response time and synchronization.[Puc20]

In the research conducted by [MKA16], the potential and limitations of ROS2 are explored by using Data Distribution Service (DDS). The authors highlight the capabilities of DDS, which is well-suited for real-time embedded systems with various transport configurations such as deadline and fault-tolerance, and scalability. An exploration and evaluation of the potential and limitations of DDS and ROS2 is carried out. The Object Management Group defines the DDS specification as a publish/subscribe data-distribution system. DDS has the capability to support various types of applications, from small embedded systems to large-scale infrastructures, including distributed real-time embedded systems. The foundation of DDS is based on a Data-Centric Publish-Subscribe (DCPS) model, which aims to efficiently transfer data between processes in distributed heterogeneous platforms. Author concluded through various experiments, the capabilities of ROS2 and the performance characteristics of ROS1 and DDS through ROS2 in different scenarios were evaluated and compared in terms of latencies, throughput, number of threads, and memory consumption. The impact of switching between DDS implementations and QoS(Quality of Service) policies in ROS2 was also measured. The results of the study showed that different DDS implementations should be used in different situations, and that DDS provides fault tolerance and flexibility for various platforms. These findings are valuable for those interested in using DDS and ROS2.[MKA16]

[Ich22] has done extensive work on Launching ROS2 applications from Cloud. FogROS2 is introduced in this paper as an adaptive platform for cloud robotics that allows for the execution of compute-intensive parts of ROS2 applications in the cloud.FogROS2 resolves the 9 limitations of FogROS1 which is created and maintained by the same team for the ROS1 framework, seamlessly integrates with the launch and communication systems of ROS 2, providing the ability to set up cloud computers, establishing secure network communication, installing the necessary code and dependencies for the robot, and launching both the robot and cloud-robotics code. The introduction of FogROS2 as a successor to FogROS addresses its limitations and offers various improvements such as compatibility with ROS 2, integration with ROS 2 launch and communication systems, provisioning and starting of cloud computers, configuring and securing network communication, installing robot code and dependencies, launching robot and cloud-robotics code, transparent video compression, improved performance, and security, access to more cloud computing providers, and remote visualization and monitoring. In experiments, it has been observed that the use of cloud computing provides significant performance benefits and transparent video compression results in further improvement. The design of FogROS2 allows for easy integration with other

cloud computing providers, DDS providers, and message compression. Its architecture is flexible and adaptable. FogROS2 automates the process of constructing virtual machine images on the cloud, resulting in faster startup times.[Ich22]

The authors of this study [Kro21] performed an examination of the end-to-end latency of ROS2 in distributed systems with various Data Distribution Service (DDS) middlewares and default settings. They also analyzed the performance of the ROS2 stack, identifying areas where latency was slowing down. In this paper author also experimented with the effect of latency on ROS2 scalability, which is very crucial to new technology. In this study, the scalability of a ROS2 system was evaluated with respect to its latency. The results showed that the latency increases with payload size when it exceeds the fragmentation size of UDP(User Datagram Protocol). The frequency of messages was found to have an inverse relationship with latency, with minor fluctuations observed for the 95% percentile of latency as the number of nodes increased. The hardware used and parameter settings were found to have a significant impact on latency. The performance of different DDS middleware was also evaluated, and it was found that no single middleware provided the lowest latency in all cases. The biggest contributors to overall latency were found to be the DDS middleware and the delay between message notification and message retrieval by ROS2. The results also showed that latency was higher on Raspberry Pi and the fluctuations in latency were more significant compared to a desktop PC. However, the performance of different middlewares varied between the Raspberry Pi and desktop PC.[Kro21]

4 Methodology

This section covers the Methodological aspect of the work. The section 4.1 explains the workflow of the thesis work. The section 4.2 illustrates the dataset concerned with the current work. The section 4.3 encapsulates the information detailed information about the model used for the training and training parameters used in the process. It further discusses data augmentation and the data manipulation process. The section 4.4 explains the detailed process of model compression and optimization. The section 4.5 includes the process of removal distortion in Image. And section 4.6 explains the creation of ROS2 nodes for image undistort and object detection nodes. And the section 4.7 includes the all detailed information about the docker image used for deployment of the ROS2 nodes. In section 4.8 we discussed tools used in project work.

4.1 Project Workflow

As shown in Figure 4.1 project work is started with select the appropriate data for the analysis. This involves identifying relevant data sources for the task, assessing data quality, and selecting the most appropriate data for the project. Once data sources have been identified, it is important to assess the quality of the data. This involves checking for missing values, data integrity, and other data quality issue. Detailed about the selected datasets and information is explained in the section 4.2. The next step is pre-processing the data. This involves cleaning the data, removing missing values or outliers, and transforming the data to a format suitable for model training. detailed explanation about data pre-processing is given in subsection 4.2.2. After the pre-processed dataset is in place, the next step is to train the model. This involves selecting an appropriate model architecture, defining hyperparameters, and training the model using the pre-processed dataset. Training details and hardware used for the training are explained in section 4.3. Next step is as per the project's main requirement optimize the model for a high frame rate used by using the Jetson Nano and TensorRT this is explained in section 4.4. The Object detection model will be used in the ROS2 framework where the image is received from RSTP camera(Real Time Streaming Protocol) for this task ROS2 Environment setup is explained in section 4.6. In this research work four cameras are used for the experimental set as shown in

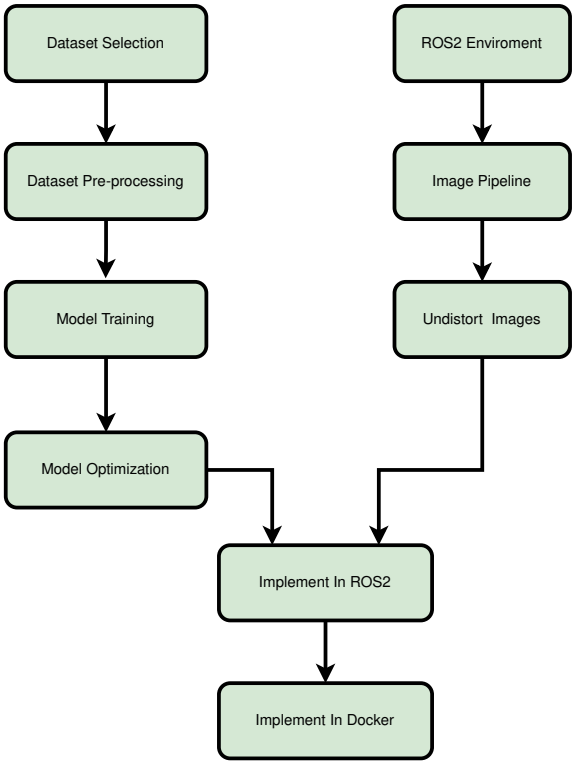


Figure 4.1: Project Workflow

Figure 4.2, image pipeline created for the input to model i.e. preprocessing in the input stream and the need for input stream is explained in the section 4.5. For the final step, everything is implemented in Docker because Jetson Nano does not support the ROS2 Implementation in Docker, and building the Docker image is explained in section 4.7.

Figure 4.2: Experimental test bench

4.2 Data Preparation

Object detection is a computer vision activity that requires recognizing and locating specific objects in images or videos. A dataset containing labeled images is essential to develop an object detection model. This dataset serves as the base for the model, and its quality is crucial to the model's accuracy. The dataset should include images with different objects, backgrounds, and lighting conditions to help the model learn and perform better.

When building an object detection model, the dataset used for training is crucial to its performance. A dataset with a diverse range of images and videos can help the model to generalize better, which means that it will be able to recognize objects in new and unseen data more accurately. On the other hand, if the model is trained on a small dataset that has limited variation, it may not perform well when presented with new data that has different lighting conditions, camera angles, and object sizes. In such cases, the model may fail to identify the objects of interest, resulting in reduced accuracy. Therefore, having a diverse dataset that covers a wide range of scenarios is essential to train a robust object detection model that can perform well in various conditions.

4.2.1 Dataset Selection

For this research work, two datasets are combined and then the model is trained. Detailed information about the datasets is given bellow

- **Coco Dataset:**

  The COCO dataset is a popular dataset used for training and evaluating computer vision models. It is widely used for tasks such as object detection and instance segmentation. It is a large-scale dataset that includes image recognition, segmentation, and captioning. Microsoft created the COCO dataset, which is maintained by the COCO Consortium, a group of institutions that includes Carnegie Mellon University. The dataset was developed to address some of the limitations of previous datasets, such as ImageNet, which only focused on object classification.[Con23]

  The COCO dataset, which includes 81 object categories, contains more than 330,000 images and 2.5 million object instances. Professional annotators labeled each object instance with a bounding box and an object category label using a polygon annotation tool. The dataset is utilized not only for object detection but also for other computer vision tasks such as instance segmentation and image captioning. Instance segmentation identifies each object instance's pixels in addition to detecting objects, and image caption-

ing produces a descriptive caption for an image. The COCO dataset has become a benchmark for object detection and segmentation models, with many state-of-the-art models using it as a standard evaluation dataset. It has also been used in several computer vision challenges, such as the COCO Detection Challenge and the COCO Captioning Challenge, which encourage researchers to develop more accurate and robust models for object detection and image captioning.[Con23]

In summary, the COCO dataset is a large-scale image recognition, segmentation, and captioning dataset that has become a standard benchmark for object detection and instance segmentation models. Its size, diversity, and accuracy have made it an invaluable resource for researchers and developers working on computer vision tasks.

- **Europe City Data-set:**

  Europe city dataset is created by the Department of Cognitive Robotics Delft, The Netherlands. The aim of the research was to increase pedestrian detection performance by utilizing a bigger dataset. The EuroCity Persons dataset is introduced, providing detailed annotations of pedestrians, cyclists, and other riders in urban traffic scenes. The dataset was created by collecting images from a moving vehicle in 31 cities across 12 European countries, resulting in over 238,200 labeled person instances in over 47,300 images. Additionally, the dataset includes over 211,200 person orientation annotations. Four deep learning approaches (Faster R-CNN, R-FCN, SSD, and YOLOv3) were optimized as baselines for the new object detection benchmark. The dataset was used to analyze the generalization capabilities of these detectors, as well as the effects of training set size, dataset diversity, dataset detail, and annotation quality on detector performance.[Bra19]

The primary reason for selecting these particular datasets is because it matches our use case

4.2.2 Data Pre-Processing

For this research work, we are using the YOLOv5 algorithm for object detection. Importance of YOLOv5 has been explained in the section above. To use YOLO v5, the annotations for an image must be provided in a .txt file(see the Figure 4.4). Each line of the file represents a bounding box for an object in the image. In the Figure 4.3, there are three objects (two persons and one tie), and each object is described in one line of the text file. The format for each line is: class, x_center, y_center, width, and height, where the box coordinates must be normalized between 0 and 1 based on the image dimensions. The class numbers

Figure 4.3: Coco Annotation Example
[Pap21]

start from 0.

The Europe City dataset and COCO dataset are annotated using different formats since they were created by different organizations. The Europe City dataset uses Pascal VOC format and the labels are saved in a .json file example shown in Figure 4.5. To use the labels with the YOLO algorithm, we created a python script that converts the labels from Pascal VOC format to YOLO's .txt format. On the other hand, for the COCO dataset, the image annotation labels are already in the .txt format and can be used directly with YOLO.

Once the annotation was converted from the pascal VOC format to YOLO format. We also needed to address the difference in image size between the two datasets.

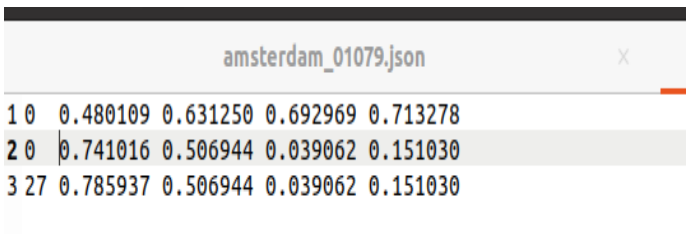


Figure 4.4: Coco label Example
[Pap21]

The COCO dataset has images of size 640×480 while the Europe city dataset has images of size 1200×720. Therefore, we wrote a Python script to change the size of the images in the Europe city dataset to match the size of the images in the COCO dataset. This was necessary to ensure that the YOLO algorithm can process the images correctly and generate accurate results.

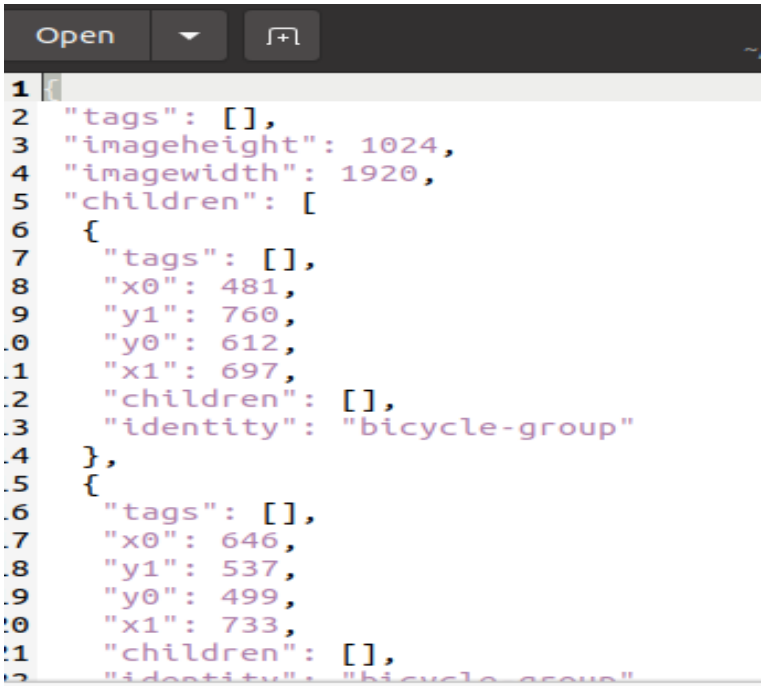


Figure 4.5: Pascal VOC label for Europe city Data-set

4.3 Training Concepts

This section covers the process of training an object detection model on a dataset, providing information on the image augmentation parameters and the training methodology. The first part of this section discusses the parameters used for image augmentation. The later subsections of this section focus on the methodology of training the model, providing information on the architecture and training parameters used. This includes details on the specific neural network architecture used, as well as the learning rate, batch size, and other hyperparameters. The training methodology encapsulates this information and explains how it is used to optimize the model's performance during the training process. Overall, this section provides a comprehensive overview of the training process for object detection models on a dataset.

No	Augmentation	Data Type
1	Random Crop	str2bool
2	Crop Width	int
3	Crop Height	int
4	Vertical Flip	str2bool
5	Horizontal Flip	str2bool
6	Brightness	float
7	Rotation	float
8	Zoom Range	float
9	Channel Shift	float

Table 4.1: Augmentation Parameters

4.3.1 Image Augmentation

Yolov5 is an object detection model that utilizes image augmentation to enhance the model's performance during training. Image augmentation is the process of creating new variations of the original images by applying various modifications such as cropping, resizing, flipping, and rotation. By applying these modifications randomly, the model can see more diverse examples of the same objects during training. This can help to reduce over-fitting and improve the model's ability to recognize objects in various orientations and scales. For example, random cropping can generate multiple images by randomly cropping different areas of the input images. This can help to generate variations of the same object at different scales, which can be useful for detecting objects that are farther away or closer to the camera. Random resizing can also be applied to generate images of different sizes, which can help to improve the model's robustness to variations in image resolution.[Ult20]

Overall, image augmentation is a powerful technique that can significantly improve the performance of object detection models like Yolov5, by increasing the amount of data available for training and improving the model's ability to generalize to new, unseen data. Some important image augmentation parameters are listed in the Table 4.1

4.3.2 Training Details

This work approaches dataset training on YOLOv5 object detection architecture-based work of [Ult20], which is based on the work of Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi in their research paper "You Only Look

Once: Unified, Real-Time Object Detection". YOLOv5 was introduced in 2020 by [nr 44], and its implementation is in the Pytorch framework. A brief overview of the YOLOv5 architecture is explained in the section 2.6. In this section, we will see the detailed model architecture layer-by-layer construction of the model. And hardware used to train the model.

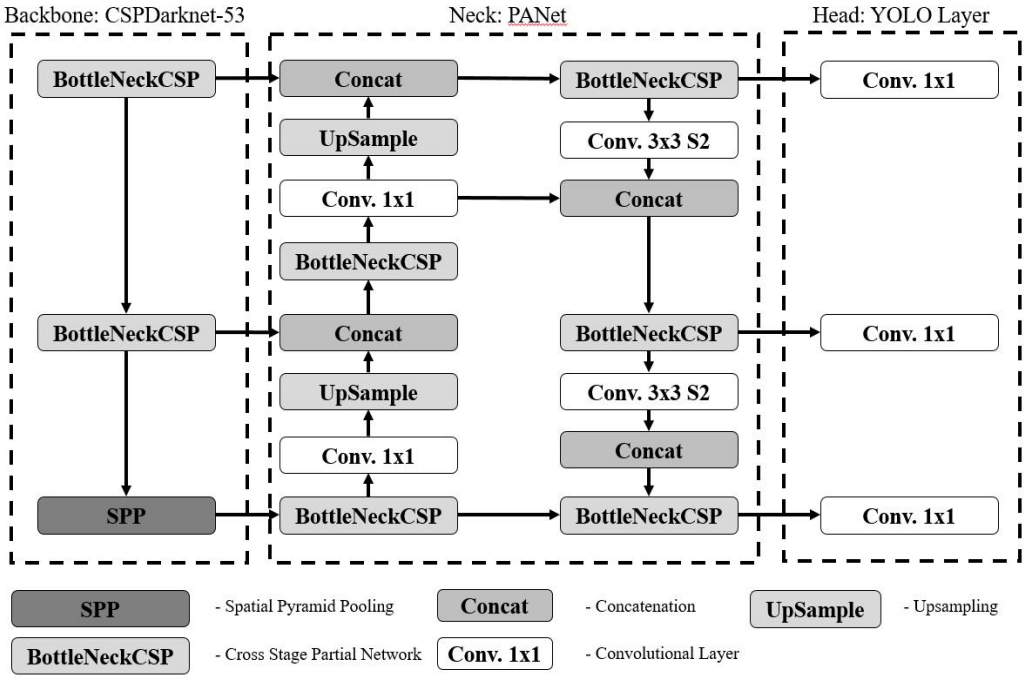


Figure 4.6: YOLOv5 Architecture
[SP21]

Model Builder

YOLOv5 architecture uses CSPDarknet(Cross Stage Partial) [SP21] i.e. Model backbone as a feature extractor. and PANet i.e. Neck as a detector. finally, for the model head, it used YOLO layer for drawing bounding and predicting class probabilities [SP21]. As you can see in the Figure 4.6

There are 4 sub-versions of YOLOv5 architecture. YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5n. These models are distinguished by their number of convolutional layers and subsamples in the CSP(Cross Stage Partial) layer blocks of the feature extractor. This results in different varying sizes of occupied disk space and recognition quality. In other words, the models' architecture affects the size of the model, its ability to accurately recognize objects, and its inference speed.

As you can observe from Table 4.2 comparison of the sub-variation of the YOLOv5 models, the Mean average precision (mAP@0.5:0.95) increase size of the model

Model	mAP@0.5:0.95	Weights	No. of Parameters	Speed(CPU b1 in ms)
YOLOv5n	28.0	3.9 MB	1.9 Million	45
YOLOv5s	37.4	14.5 MB	7.2 Million	98
YOLOv5m	45.4	42.3 MB	21.2 Million	224
YOLOv5l	49.0	92.9 MB	46.5 Million	430

Table 4.2: Model Comparison
[Ult20]

increases and so does the inference speed of the model. By comparing the inference speed and precision of the n, s, m, and l we decided to go with s i.e YOLOv5s for training and optimizing the model for further use. As shown in Figure 4.7 YOLOv5s.ymal configuration file that contains information about the architecture of the corresponding neural network and the number of classes recognized by the model.

Hardware Used For Training

Training deep learning models is a computationally very expensive task. To train, the deep learning model needs NVIDIA GPUs in order to achieve good results in terms of training speed and accuracy. To train our object detection YOLOv5 model we used Elwetritsch GPU clusters provided by the university. Specification for the devices is as follows

- Operating System: Red Hat Enterprise Linux Server 7.9 (Nitrogen)
- GPU : NVIDIA V100 TENSOR CORE GPU(125 teraFLOPS, 300 WATTS)
- CPU: Intel® Xeon® Silver Prozessor 4215, 2.5 GHz , 8 cores
- Memory: 64 GB 64 bit LPDDR4 25.6GB/s

4.3.3 Training Conceptualization and Parameter

The model used in this work (YOLOv5) is based on the work of [Red16] and conceptualized by [Ult20]. The model is Trained using the PyTorch framework. The process of training is detected by a set of constants known as hyper-parameters. Some of the most used hyper-parameters are as follows:

- Learning Rate

```yaml
#
# YOLOv5s
# Parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # 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
  ]

# 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, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Figure 4.7: Content of file yolov5s.yaml
[Ult20]

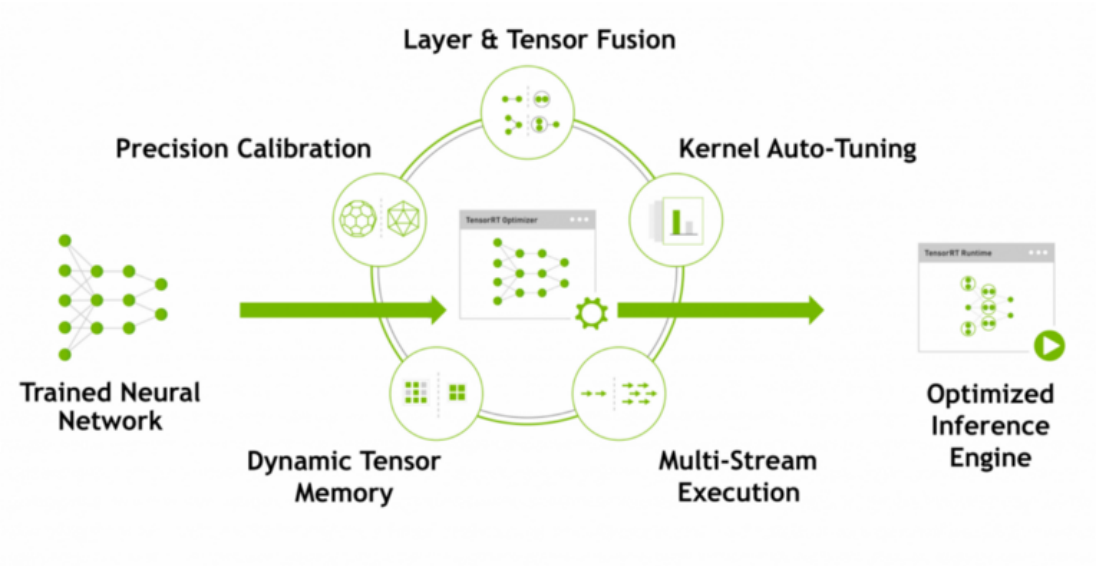


Figure 4.8: Types of Optimization Performed by TensorRT [Ten23]

- Number of Epochs
- Hidden Layer
- Hidden Unit
- Activation Function
- Batch Size
- Optimizer

The effect of these parameters on training behavior is explained in the section 5.2

4.4 TensorRT:Model Optimization

A basic introduction about TensorRT and the reason it is used in this particular application is explained in section 2.9. Five types of optimization are performed in the TensorRT to increase the throughput of the deep learning model(Figure 4.8). And these optimizations are explained as follows [Ten23]

- **Weight and Activation Precision Calibration:** During the Training process model saves the parameters(weights) and activation in FP32(floating point 32), to achieve better precision and accuracy. While optimizing the model using TensorRT it converts them into FP16 or INT8, this optimization

reduces latency and reduces model size. While converting from FP32 to FP16 or INT8 some weights get shrunk, This results in a reduction in accuracy but this effect(less accurate model) is not significant.

- **Layers and Tensor Fusion:** When using deep learning software to process a graph, the same calculation is repeated over and over. TensorRT solves this problem by combining layers and tensors in a smart way to save GPU memory and speed up processing. TensorRT uses the layer and tensor fusion by fusing nodes in the kernel vertically or horizontally which results in a reduction in the cost of reading and writing the tensor data in each layer.
- **Kernel auto-tuning:** In general Deep learning models do not take into consideration GPU platform architecture. In this Step of the optimization process, some kernel-specific optimization is performed. For example, it selects the optimal batch size based on the GPU platform. For example, if we optimized the model using NVIDIA RTX2080(hardware GPU platform) this optimized model will not work in Jetson Nano.
- **Dynamic Tensor Memory:** TensorRT makes better use of memory by allocating memory to tensors only when they are in use, which helps to reduce the amount of memory used and avoids unnecessary memory allocation cost, which results in faster and more efficient processing.
- **Multiple Stream Execution:** TensorRT is built to handle several input streams at the same time, using NVIDIA's CUDA stream as its foundation. this is one of the most important factors comes in the practice, For our application because we are receiving four video streams from the cameras.

4.5 Image Undistorted

In our experimental setup, we used a wide-angle camera. The lens distortion effect is observed in Images captured by the wide-angle camera. This camera distortion can be categorized into five distortion coefficients. And these five confinements are used to remove distortion effects produced by a wide-angle camera. By using camera calibration, it is possible to obtain the camera matrix and distortion coefficient which can be further used to correct the distortion in created by the lens. The process includes comparing distorted images with their corrected versions. Camera calibration is a technique that enables the precise measurement of the camera's intrinsic and extrinsic parameters, which are required to correct the distortion in images captured by the camera. This correction process can improve the quality and accuracy of the images.

Algorithm 1 Distortion correction

1: **Input:** Distorted images, CameraMatrix, Distortion coeff.
2: undistort images with cv::undistort()
3: **Output:** Distortion corrected images

As shown in the above Algorithm 1. distortion correction algorithm takes input such as raw (distorted image coming from the wide-angle camera), camera matrix(which is calculated from the camera calibration) distortion coefficient(calculated during camera calibration) and returns the un-distorted image. This un-distortion algorithm is used in the ROS2 nodes. For the calibration of the cameras following Algorithm 2 is used.

Algorithm 2 Camera Calibration

1: **Input:** Good captured calibration images
2: Declares calibration pattern information
3: Declare object points and image points vector
4: Convert color RGB image to Gray image
5: Declare corner points vector
6: Find corner points with findChessboardCorners()
7: Push-back corner points into image points
8: Find camera parameter with calibrateCamera()
9: **Output:** Camera matrix, distortion coefficient, translation and rotational vectors

This Algorithm 2 returns the returns Camera Matrix and Distortion coefficient which is the input parameter for the Algorithm 1.

4.6 ROS2 Nodes

As explained in the workflow of the project (section 4.1), the experimental setup contains four wide-angle cameras. Raw wide-angle images are received in the image publisher node and distortion in images is removed, This un-distorted image is published on the topic. Next, this un-distorted image is subscribed in the TensorRT node and objects are detected from the image and bounding boxes are drawn on the image. And TensorRT node publishes the object-detected image(bounding boxes are drawn) and Bounding Box. In the next image subscription, node object-detected images are subscribed. Node arrangement is shown in Figure 4.9

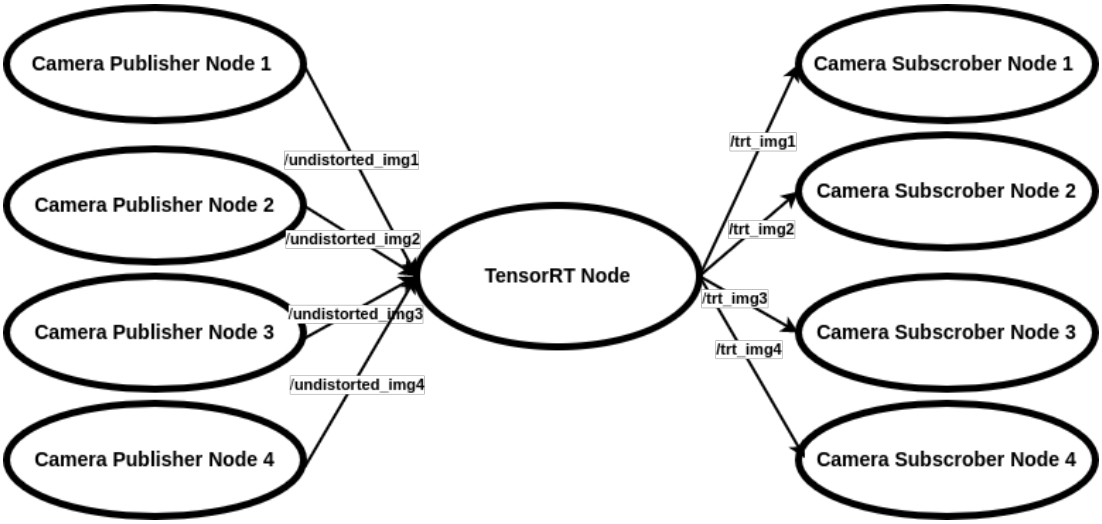


Figure 4.9: ROS2 Node Structure

4.6.1 Image Publisher Node

The image publisher node is created using the programming language C++, which is a commonly used language for developing applications that prioritize high performance and direct management of hardware resources. Additionally, the main function of a node is to remove distortion from the image and publish the image (topic) for further use, as shown in Algorithm 3.

The publishing data type is an image that has the following attribute

```
std_msgs/Header header
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

Algorithm 3 Image Publisher Node

1: **Input:** RSTP Camera IP Address
2: Get Camera Matrix, Get Distortion coeff.
3: Remove distortion in images with cv::undistort()
4: **Output:** Publish Image (topic)

4.6.2 Object Detection Node

This node is used to detect the objects using the TensorRT engine. Compression of the PyTorch engine is explained in the section 4.4. This node is created in the python package. The object detection node subscribes to the undistorted image(Topic). Object detection Node Algorithm is shown bellow(Algorithm 4).

Algorithm 4 TensorRT Object Detection Node

1: **Input:** Image(Topic Subscription)
2: Image Pre-processing
3: Engine Importing
4: Building the TensorRT Engine
5: Allocating Memory
6: Binding Inputs and Outputs
7: Inference
8: c
9: Drawing bounding boxes around the detected objects and displaying the class labels
10: **Output:** Publish Image (topic), Bounding Box(Topic)

Important steps in the algorithm are as follows:

- Image Pre-processing: Convert input image to the format used by the Engine, this involves resizing the image, changing the color format, and normalizing pixel value.
- Building the TensorRT Engine: TensorRT creates a runtime engine, and the engine is optimized for the hardware platform. This includes layer fusion and precision calibration.
- Allocating Memory: TensorRT engine allocates memory for the input and output tensors of the model
- Binding Inputs and Outputs: Input and output tensors are bound to the allocated memory
- Inference: Input data(Image) is passed through the TensorRT engine to perform inference. And it executes the forward pass of the neural network model to generate a set of bounding boxes and corresponding class probabilities.
- Post-Processing: In the post-process step it filter false positive, suppress redundant detection, and bounding box dimensions converted to the original image coordinate.

And this node publishes object detected image(Bounding Box drawn) and Bounding Box. For the image std image sensor datatype is used (explained in subsection 4.6.1). The custom data type is created for the bounding box.

```
std_msgs/Header header
float32 probability
uint16 xmin
uint16 ymin
uint16 xmax
uint16 ymax
uint16 id
uint16 img_width
uint16 img_height
int32 center_dist
string class_id
uint16 class_id_int
```

4.6.3 Image Subscriber Node

This node is created to subscribe to the object-detected image coming from TensorRT Node and display the image or stream of images.

4.7 Docker Image

The basic concept of Docker and its purpose of use in our work is explained in section 2.8. As it explained in workflow (section 4.1) we are using the Jetson Nano(check the details of Jetson Nano in section 4.8) for the object detection model deployment. Jetson Nano works on Jetpack 4.6 developed by the NVIDIA team and Jetpack is built on Ubuntu 18.04 (Linux Distro). The minimum operating system requirement for ROS2 is Ubuntu 20.04. for this reason in this research work, we are building the docker image with the base image of OS Ubuntu 20.04. Docker we built for this research work is derivative of the work done by [Syn22]. Basic docker building instruction is given in [Syn22]. Following are some important repository and tools frameworks installed in the docker images.

- OpenCV C++
- PyPorch 1.8.0
- torchvision 0.9.0

- ROS2 foxy
- matplotlib>=3.2.2
- opencv-python>=4.1.1
- nvidia-tensorrt
- tqdm>=4.64.0
- psutil

4.8 Tools

Current work uses multiple tools and devices to complete the objective of the research work. Presently, the concept has been implemented using programming languages like Python and C++, And libraries like Open CV and PyTorch. Important tools used in the work are illustrated as follows:

4.8.1 Hardware

For this work, there are two important hardware one is for training the model and the second one is the Embedded device on which the model is deployed i.e Edge device

Training Device:

The training of the deep learning model requires a high computing platform. This work uses the Elwetrish GPU cluster. The specification is given in the subsubsection 4.3.2

Edge Device:

The main object of this work is to deploy object detection on Jetson Nano. NVIDIA Jetson Nano Developer Kit is a small powerful computer that lets the developers run multiple machine learning applications such as image classification, object detection, and image segmentation. The device uses multiple modes of power supply and is mostly used in AI .

- AI Performance: 472 GFLOPs

- GPU: 128-core NVIDIA Maxwell™ GPU
- CPU: Quad-Core Arm® Cortex®-A57 MPCore processor
- Memory: 4 GB 64-bit LPDDR4 25.6GB/s
- Networking: 10/100/1000 BASE-T Ethernet
- Mechanical: 69.6 mm x 45 mm 260-pin SO-DIMM connector

The Schematic Diagram of the Jetson Nano is in Figure 4.10

- **Jetson Nano Setup:**

  As explained above Jetson Nano has very limited RAM and Storage Capacity.For the Jetson Nano, NVIDIA has created custom Operating system called Jetpack[NVI23]. Jetpack is based based on the Ubuntu 18.04 LTS. Installing Jetpack On SD card is similar to flashing the USB stick for the new operating sysytem. After installing the the Jetpack on Jetson Nano and setting up the device, we need to follow one more important stuff in order to operate device smoothly. After setting up the device we need to increase the swap memory in use. To increase the swap memory on the Jetson Nano we need to follow the official guide provided in Jetson forum [NVI23]

4.8.2 Software

- Deep Learning Framework For Training: The current research work involves using PyTorch [PyT] to train the object detection model such as YOLOv5. PyTorch is an open-source optimized deep learning library based on python, and PyTorch is mostly used for applications using GPU and CPUs. PyTorch is favored over other deep learning frameworks such as TensorFlow and Keras since it makes use of dynamic computation graphs. Most importantly it allows developers to debug the neural network in real time. Training is achieved through using "ultralytics/yolov5" [Ult20] repository on GitHub.
- Deep Learning Framework for Deployment: This research work is based on the use of a TensorRT-optimized model to deploy the model on the device. TensorRT is developed by NVIDIA. NVIDIA® TensorRT™, an SDK for high-performance deep learning inference, which includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for inference for real-time object detection applications. In this research work we used "wang-xinyu/tensorrtx"[Wan19] to optimize and deploy the model.

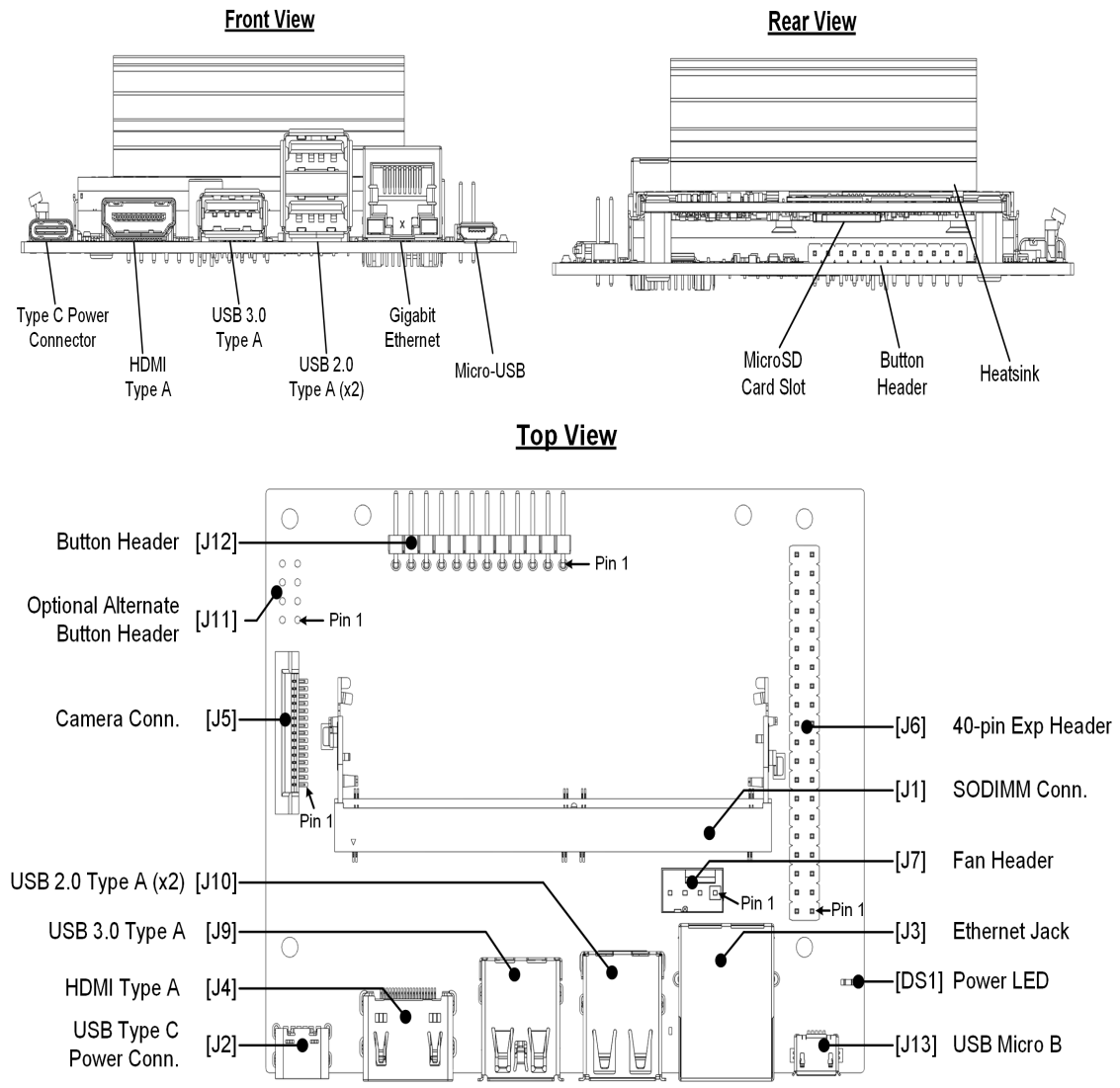


Figure 4.10: Jetson Nano Schematic
[NVI23]

5 Evaluation

The evaluation process aims at observing the training procedure and inference quality of the model. Firstly, in section 5.1 the evaluation of the process assesses the initial performance of the YOLOv5 [Ult20] model. Further, the model is optimized using the post-training quantization technique. In section 5.2 we discussed in detail the model size and inference speed and frame rate per second (FPS) for different camera outputs.

5.1 Training Process Evaluation

As discussed earlier for this research work, we choose YOLOv5 [Ult20] model for the object detection task. Moreover, the models are trained using the following parameter shown in Table 5.1. Once the model is trained it is saved in ".pt" format.

In this research work, Indexes such as Precision, Recall, F1, AP, mAP_0.5 and mAP_0.5:0.95 were selected to evaluate the performance of the project detection model(YOLOv5) after the training. Precision is the ability of the model to

No	Parameter	Value
1	Epoch	300
2	Learning Rate	0.01
3	Learning Rate Scheduler	LambdaLR
4	Optimizer	SGD
5	Data Augmentation Rate	0.5
6	Batch Size	16
7	Loss Function	Binary Cross-Entropy with Logits Loss
8	Image Size	640, 480

Table 5.1: Training Parameters

correctly predict the accuracy of the positive samples(see Equation 5.1). In the simplest term, Precision is the ratio between the True Positives and all the Positives(True Positive(TP) and False Positive(FP)). Larger the value of the precision better the performance of the model. The recall is the ability of the model to correctly identify True Positives(see Equation 5.2). The recall is the ratio of the positive sample predicted to the total positive sample(TP + FN(False Negative)). Recall and precision have similar performance metrics, as they both relate to a model's ability to correctly identify positive samples. Recall and precision can influence each other. Generally, if the accuracy rate is high, the recall rate will be low, and if the accuracy rate is low, the recall rate will be high. The F1 measure is the weighted harmonic mean of precision (P) and recall (R) of a classifier, taking $\alpha = 1$(F1 score). F1 graph for the trained YOLOv5 object detection model is show in Figure 5.1

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{5.3}$$

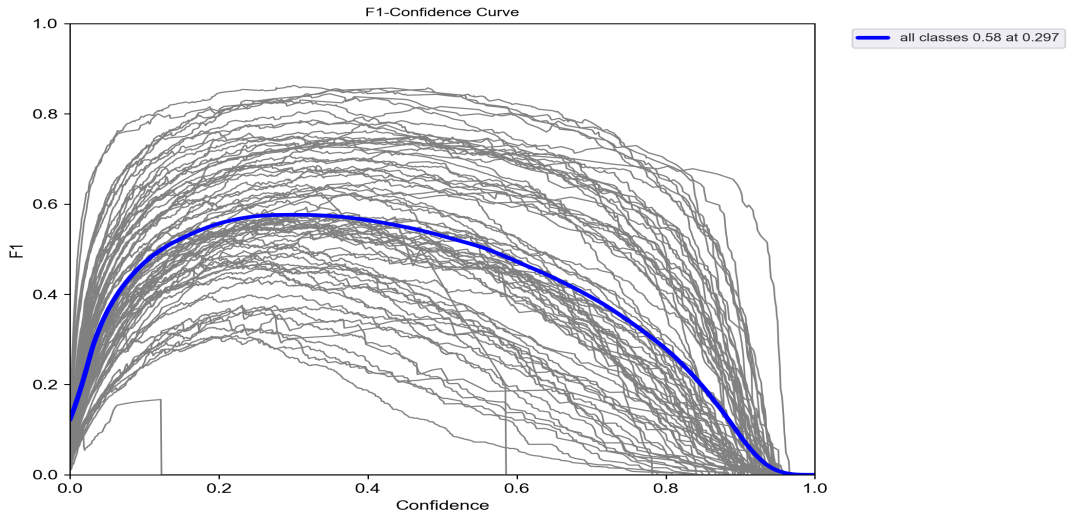


Figure 5.1: F1 Curve

In YOLOv5, average precision (AP) (see Equation 5.4) is a common evaluation metric used to assess the performance of object detection models.In YOLOv5 we can calculate two types of average precision, the first one is mean average

precision (mAP), and the second one is mean average precision at a different intersections over union (IoU) thresholds (mAP@IoU). mAP calculates the average precision of the algorithm across all object categories and IoU thresholds. mAP@IoU calculates the average precision of the algorithm for each object category at different IoU thresholds. It is calculated by computing the AP of the algorithm for each category and each IoU threshold and then taking the mean of these AP values. IoU is a measure of how well the predicted bounding box overlaps with the ground truth bounding box. The mAP@IoU metric is useful in summarizing the algorithm's prediction capability at different levels of overlap between the predicted and ground truth bounding boxes.

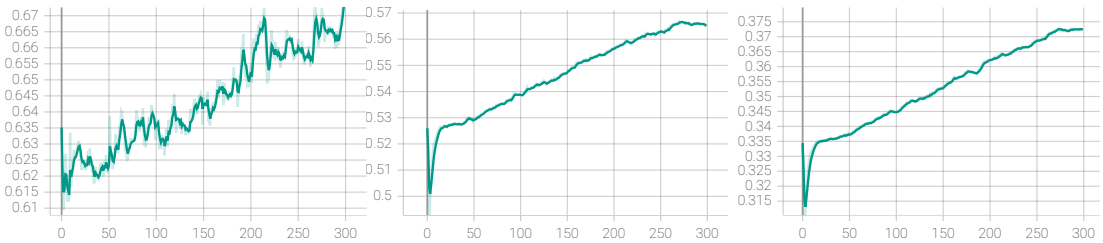
$$AP = \frac{\sum_{k=1}^{n} P(k) \cdot rel(k)}{R} \tag{5.4}$$

(5.5)

Average Precision and graph is shown in Table 5.2 and Figure 5.2

Precision	Recall	mAP@0.5	mAP@0.5:0.95
0.67491	0.52361	0.56483	0.37228

Table 5.2: Training Precising and Recall



(a) Average Precision (b) mAP@0.5 IoU (c) mAP@0.5:0.95 IoU

Figure 5.2: Average Precision Graphs for Training 300 Epochs

There are several types of losses used in the YOLOv5 object detection model, each of which plays a different role in training the model. Calculation loss is discussed in subsection 2.2.2 In this research work Class Loss, Object Loss, and Box Loss are one of the most important losses considered for evaluation. As shown in Figure 5.3 and Table 5.3.

Class Loss measures the how well model is able to classify the object it has detected. The class loss is important because it helps the model distinguish

between different types of objects in the image. Object Loss measures how well the model is able to detect the presence of objects in each grid cell. Object loss is important because it helps the model focus on the grid cells that contain objects. Box Loss measures how well the model is able to predict the bounding boxes around the objects in the image. The box loss is important because it helps the model accurately localize the objects in the image

The Table 5.3 summarizes achieved training and validation loss during the training. the main objective of the training is to minimize the training and validation loss in Object, Class, and Box detection. This can be observed from the graph in Figure 5.3

Type	Box Loss	Object Loss	Class Loss
Train	0.042873	0.063126	0.015931
Val	0.043063	0.049792	0.015715

Table 5.3: Training and Validation Loss

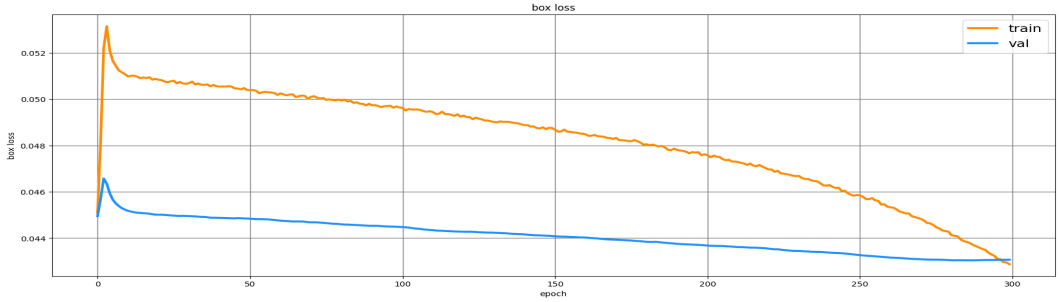
5.2 Model Evaluation

In the previous section, we discussed various metrics used to evaluate the performance of a machine-learning model during the training process. For the evaluation, we considered Hyper-parameters, Mean Average Precision (mAP) at different IoU, and Training and Validation Loss. The main goal of the training process is to increase the mAP and minimize the Training and validation loss by optimizing the Hyper-Parameter.

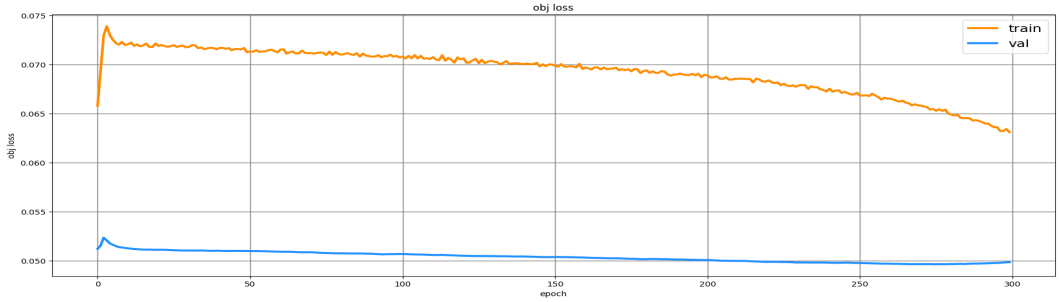
In this section, we are going to evaluate the model by size and inference speed after and before optimization by using TensorRT The process of model optimization by using TensorRT is explained in section 4.4

5.2.1 Model Size

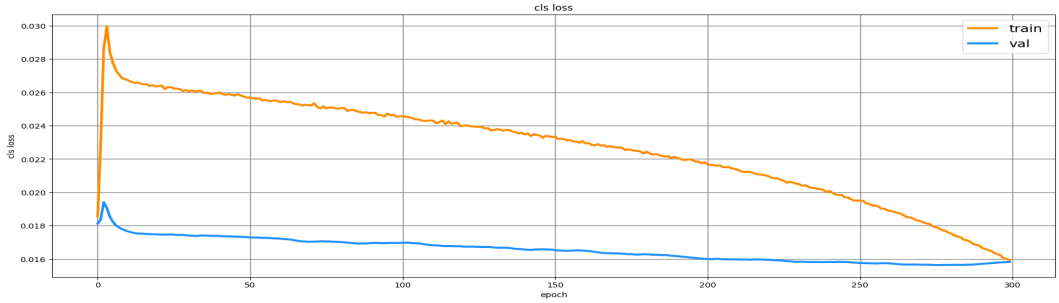
In this section, we are going to evaluate and discuss the size of the model after optimizing the model using TensorRT. When the model is optimized using TensorRT it generates a ".engine" file. In Table 5.4 explained. TensorRT engine is 54.02% bigger then that of the '.pt' file.



(a) Box Loss vs Epoch



(b) Object Loss Vs Epoch



(c) Class Loss Vs Epoch

Figure 5.3: Training and Validation Epoch vs Loss

Nr	Model	Size
1	YOLOv5 PyTorch	14.1 MB
2	TensorRT Engine	21.8 MB

Table 5.4: Model Size Comparison

5.2.2 Inference Speed Evaluation

TensorRT is a Deep Learning framework developed by NVIDIA for use with their GPUs, which is based on NVIDIA's CUDA parallel programming model. By utilizing TensorRT, it is possible to achieve a speed boost of around 4 to 5 times faster inference compared to the baseline model. It is widely acknowledged that TensorRT inference graphs can significantly improve inference speed and throughput compared to running models natively.

In Table 5.5 is show a comparison between inference speed between the PyTorch model and TensorRT optimized model. For this research work, FP16 quantization is used. As you can see that from the table for one camera object detection there is a 64.14% increase in the inference speed. As we increase the number of cameras for object detection from one to two and four, inference speeds of the PyTorch Model are increased by 2 times(or by 87.96%) and 4 times(or by 375%) respectively. But we can see that for the TensorRT model, there is no significant increase in the inference speed.

Model	One Camera	Two Camera	Four Camera
YOLOv5 PyTorch	133 ms	250 ms	500 ms
TensorRT	81 ms	82 ms	90 ms

Table 5.5: Inference Speed Comparison FP16 Quantization

For this research work camera used has a max frame rate of 25 FPS(Frames per Second). From the Table 5.6, we can see that for the PyTorch Model as the number of cameras increase frame rate for the object detection per camera drops. With an optimized model, there is no change in the frame rate. The main reason behind this is when the model is optimized it takes consideration into the architecture of the hardware platform and creates a parallel engine execution node. This way as the number of cameras increases inference speed and frame rate remain same or as close as possible.

Model	One Camera	Two Camera	Four Camera
YOLOv5 PyTorch	12 FPS	6 FPS	3 FPS
TensorRT	25 FPS	25 FPS	25 FPS

Table 5.6: FPS Comparison FP16 Quantization

6 Conclusion and Future Work

Nowadays object detection became one of the most important fields in the design of safety systems in the automotive industry and the construction side. Object detection is done in two parts object classification and localization of the object in the image. Object detection models not only need to work accurately to detect the object and localize it they also need to work fast in real-time on the device(Jetson Nano). The main aim of this research work is to optimize the object detection model and deploy the optimized model on the edge devices. This research work acknowledges the challenges in the field of object detection model optimization and deployment on the Jetson Nano(Edge Device).

The proposed system architecture is comprised of ArkCam Basic cameras (wide angle image) is interact with Jetson Nano through IP address. Later the object detected an image coming from Jetson Nano conceptualized. For this research work, Jetson Nano is used for testing and deploying of the object detection model. The Jetson Nano is a small, low-cost single-board computer designed for AI and machine learning applications. It is manufactured by NVIDIA and comes with an operating system called JetPack, which is based on Ubuntu 18.04. Jetson Nano has a major disadvantage when it comes to deploying the AI application it has limited processing power, limited storage, and limited ram. The main challenge working with Jetson Nano is it has the latest CUDA capabilities but is implemented on Ubuntu 18.04, and most of the tools and libraries needed for the work were not compatible with JetPack.

This work explores the task of an object detection-based trained model over an embedded device (edge device). The training of the object detection dataset is done by using YOLOv5s [Nr 44]. This particular model is chosen because of its ability to achieve low latency with less size and better accuracy. Furthermore compatibility with the TensorRT framework for optimization [Nr 50]. This work is based on the foundation of several experiments that were conducted using varying parameters for the selected architecture to avoid over-fitting and under-fitting. This work has considered 200 epochs and used "Binary Cross-Entropy with Logits Loss" as a loss function for the YOLOv5 architecture. In this research work, YOLOv5 object detection is employed and the optimization algorithm used in YOLOv5 is Stochastic Gradient Descent (SGD). The Model is trained on the Elwetritsch HPC(high-performance computer) Network provided by the university.

The trained model is optimized using TensorRT and optimization is done by using the post-training quantization method suggested by [Ten23] through TensorRT converter[Wan19].

The results are explained through a comparison of inference speed size and FPS(frames per second) with a different number of cameras on Jetson Nano for pre-optimization(PyTorch model) and post-optimization(TensorRT Engine). It has been confirmed that the post-optimized model(TensorRT Engine) is 54.02% bigger than that of the PyTorch model(.pt model), But when it comes to the inference speed TensorRT Engine(Post-optimized model) is 64.14% faster than the PyTorch model for one camera setup. Furthermore, for the two and four-camera setups, the TensorRT engine was 86.96% and 375% respectively faster than that of the PyTorch model. Also when it comes to Object detection FPS for one, two, and four camera setups there was no drop in frame rate for the TensorRT Engine. Finally, the work found that the TensorRT engine (Optimized model) archives comparatively better performance than the PyTorch model in two out of three domains of the evaluation.

The image data pipeline is created in the ROS2 Framework. Images are received from the RSTP cameras. And distortions(due to wide-angle camera) in these images are removed. Undistored images are sent to the Object Detection node (created by using the TensorRT engine) and object-detected images are subscribed in the image subscription node. A Docker container is used to deploy the whole setup mentioned above.

However, a major challenge is that TensorRT has limited support for custom operators, which requires users to write TensorRT plugins to support these operators. To address this issue, Tencent and NVIDIA have collaborated to develop an open-source tool called TensorRT Plugin Autogen Tool (TPAT). This tool supports all operators in the Open Neural Network Exchange (ONNX) format and can generate TensorRT plugins end-to-end, thereby making it easier to use custom operators with TensorRT.

Bibliography

[Add21] Addison Addison. *ROS 2 Architecture Overview*. `https://automaticaddison.com/ros-2-architecture-overview/`. Accessed on April 8, 2023. 2021.

[AOC22] Lalainne Anne J Abel, Toni Ceciro N Oconer, and Jennifer C Dela Cruz. "Realtime object detection of pantry objects using yolov5 transfer learning in varying lighting and orientation". In: *2022 2nd International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*. IEEE. 2022, pp. 1–7.

[Ben21] Aduen Benjumea et al. "YOLO-Z: Improving small object detection in YOLOv5 for autonomous vehicles". In: *arXiv preprint arXiv:2112.11798* (2021).

[Bra19] Markus Braun et al. "Eurocity persons: A novel benchmark for person detection in traffic scenes". In: *IEEE transactions on pattern analysis and machine intelligence* 41.8 (2019), pp. 1844–1861.

[Che18] Bowen Cheng et al. "Revisiting rcnn: On awakening the classification power of faster rcnn". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 453–468.

[CJW22] Zhu Chunxiang, Qian Jiacheng, and Binrui Wang. "YOLOX on Embedded Device With CCTV & TensorRT for Intelligent Multicategories Garbage Identification and Classification". In: *IEEE Sensors Journal* 22.16 (2022), pp. 16522–16532.

[Con23] COCO Consortium. *COCO Dataset*. `https://cocodataset.org/`. Accessed on April 8, 2023. 2023.

[Dev21] DevOpsCube. *What is Docker?* `https://devopscube.com/what-is-docker/`. Accessed on April 8, 2023. 2021.

[Doc23] Docker. *Using Docker Desktop*. `https://docs.docker.com/desktop/use-desktop/`. Accessed on April 8, 2023. 2023.

[Dua22] Rui Duan et al. "SODA: A large-scale open site object detection dataset for deep learning in construction". In: *Automation in Construction* 142 (2022), p. 104499.

[Gee21] GeeksforGeeks. *Clustering in Machine Learning*. `https://www.geeksforgeeks.org/clustering-in-machine-learning/`. Accessed on April 8, 2023. 2021.

[He17] Kaiming He et al. "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[HPC18] Rachel Huang, Jonathan Pedoeem, and Cuixian Chen. "YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers". In: *2018 IEEE international conference on big data (big data)*. IEEE. 2018, pp. 2503–2510.

[IBM23a] IBM. *Computer Vision*. `https://www.ibm.com/topics/computer-vision`. Accessed on April 8, 2023. 2023.

[IBM23b] IBM. *Docker*. `https://www.ibm.com/topics/docker`. Accessed on April 8, 2023. 2023.

[Ich22] Jeffrey Ichnowski et al. "Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2". In: *arXiv preprint arXiv:2205.09778* (2022).

[Int21] Intellipaat. *Supervised Learning vs Unsupervised Learning vs Reinforcement Learning*. `https://intellipaat.com/blog/supervised-learning-vs-unsupervised-learning-vs-reinforcement-learning/`. Accessed on April 8, 2023. 2021.

[IQ21] OpenGenus IQ. *YOLOv5: A Deep Learning-Based Real-Time Object Detection System*. `https://iq.opengenus.org/yolov5/`. Accessed on April 8, 2023. 2021.

[JWL20] Yuanzhe Jin, Yixun Wen, and Jingting Liang. "Embedded real-time pedestrian detection system using YOLO optimized by LNN". In: *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*. IEEE. 2020, pp. 1–5.

[KAZ22] Apostolos Karasmanoglou, Marios Antonakakis, and Michalis Zervakis. "Heatmap-based Explanation of YOLOv5 Object Detection with Layer-wise Relevance Propagation". In: *2022 IEEE International Conference on Imaging Systems and Techniques (IST)*. IEEE. 2022, pp. 1–6.

[Kim17] Phil Kim. *Matlab deep learning with machine learning, neural networks and artificial intelligence*. Springer, 2017.

[Kro21] Tobias Kronauer et al. "Latency analysis of ros2 multi-node systems". In: *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. IEEE. 2021, pp. 1–7.

[Lab21] V7 Labs. *Supervised vs Unsupervised Learning*. `https://www.v7labs.com/blog/supervised-vs-unsupervised-learning`. Accessed on April 8, 2023. 2021.

[LeC89] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[Li22] Ziran Li et al. "Application of low-altitude UAV remote sensing image object detection based on improved YOLOv5". In: *Applied Sciences* 12.16 (2022), p. 8314.

[Liu16] Wei Liu et al. "Ssd: Single shot multibox detector". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer. 2016, pp. 21–37.

[Liu22] Wenyu Liu et al. "Image-adaptive YOLO for object detection in adverse weather conditions". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 2. 2022, pp. 1792–1800.

[MHA20] Zhenzhu Meng, Yating Hu, and Christophe Ancey. "Using a data driven approach to predict waves generated by gravity driven mass flows". In: *Water* 12.2 (2020), p. 600.

[Mit07] Tom Michael Mitchell et al. *Machine learning*. Vol. 1. McGraw-hill New York, 2007.

[MKA16] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. "Exploring the performance of ROS2". In: *Proceedings of the 13th International Conference on Embedded Software*. 2016, pp. 1–10.

[MP43] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[Mur22] Jamuna S Murthy et al. "ObjectDetect: A Real-Time Object Detection Framework for Advanced Driver Assistant Systems Using YOLOv5". In: *Wireless Communications and Mobile Computing* 2022 (2022).

[NB20] Nipun D Nath and Amir H Behzadan. "Deep convolutional networks for construction object detection under different visual conditions". In: *Frontiers in Built Environment* 6 (2020), p. 97.

[NVI23] Jetson Nano Developer Kit NVIDIA. *Jetson Nano Developer Kit*. 2023. URL: `https://developer.nvidia.com/embedded/jetson-nano-developer-kit` (visited on 01/10/2023).

[Pap21] Paperspace. *How to Train YOLOv5 on Custom Data*. `https://blog.paperspace.com/train-yolov5-custom-data/`. Accessed on April 8, 2023. 2021.

[Pen21] Guangyao Peng et al. "CORY-net: contrastive Res-YOLOv5 network for intelligent safety monitoring on power grid construction sites". In: *IEEE Access* 9 (2021), pp. 160461–160470.

[Pou16] Abraham Pouliakis et al. "Artificial neural networks as decision support tools in cytopathology: past, present, and future". In: *Biomedical engineering and computational biology* 7 (2016), BECB–S31601.

[PT16] Michael Paluszek and Stephanie Thomas. *MATLAB machine learning*. Apress, 2016.

[Puc20] Lennart Puck et al. "Distributed and synchronized setup towards real-time robotic control using ROS2 on Linux". In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2020, pp. 1287–1293.

[PyT] PyTorch. *PyTorch*. `https://pytorch.org/`. Accessed: April 6, 2023.

[Red16] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[Rek20] Michael Reke et al. "A self-driving car architecture in ROS2". In: *2020 International SAUPEC/RobMech/PRASA Conference*. IEEE. 2020, pp. 1–6.

[Ros61] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[Say14] Saed Sayad. *Artificial Neural Networks (ANN)*. `https://www.saedsayad.com/artificial_neural_network_bkp.htm`. Accessed on April 8, 2023. 2014.

[SB14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[Sim17] Patrice Y Simard et al. "Machine teaching: A new paradigm for building machine learning systems". In: *arXiv preprint arXiv:1707.06742* (2017).

[Son21] Qisong Song et al. "Object detection method for grasping robot based on improved YOLOv5". In: *Micromachines* 12.11 (2021), p. 1273.

[SP21] Daria Snegireva and Anastasiia Perkova. "Traffic sign recognition application using yolov5 architecture". In: *2021 International Russian Automation Conference (RusAutoCon)*. IEEE. 2021, pp. 1002–1007.

[Stä21] Lukas Stäcker et al. "Deployment of deep neural networks for object detection on edge AI devices with runtime optimization". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 1015–1022.

[Syn22] Synced. *jetson-containers*. 2022. URL: https://github.com/dusty-nv/jetson-containers (visited on 09/24/2022).

[Ten23] NVIDIA TensorRT. *NVIDIA TensorRT SDK*. 2023. URL: https://developer.nvidia.com/tensorrt (visited on 01/10/2023).

[TF19] Team TensorFlow TF. *Introducing the Model Optimization Toolkit's pruning API*. https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html. Accessed on April 8, 2023. 2019.

[Ult20] Ultralytics. *YOLOv5: Implementation of YOLOv5 in PyTorch*. https://github.com/ultralytics/yolov5. Accessed: April 6, 2023. 2020.

[Vid15] Analytics Vidhya. *Introduction to Online Machine Learning - Simplified*. https://www.analyticsvidhya.com/blog/2015/01/introduction-online-machine-learning-simplified-2/. Accessed on April 8, 2023. 2015.

[Wan19] Xinyu Wang. *TensorRTX: TensorRT Examples and Demos*. https://github.com/wang-xinyu/tensorrtx. Accessed: April 6, 2023. 2019.

[WZW17] Liang Wang, Yihuan Zhang, and Jun Wang. "Map-based localization method for autonomous vehicles using 3D-LIDAR". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 276–281.

[Yam18] Rikiya Yamashita et al. "Convolutional neural networks: an overview and application in radiology". In: *Insights into imaging* 9 (2018), pp. 611–629.

[ZZN21] Fangbo Zhou, Huailin Zhao, and Zhen Nie. "Safety helmet detection based on YOLOv5". In: *2021 IEEE International conference on power electronics, computer applications (ICPECA)*. IEEE. 2021, pp. 6–11.