DEVELOPMENT OF A SMART EDGE DEVICE FOR FIRE DETECTION

NG WEI YUAN

UNIVERSITI TUNKU ABDUL RAHMAN

DEVELOPMENT OF A SMART EDGE DEVICE FOR FIRE DETECTION

NG WEI YUAN

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

May 2023

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	[illegible]
Name	:	Ng Wei Yuan
ID No.	:	1801391
Date	:	9/8/2022

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"DEVELOPMENT OF A SMART EDGE DEVICE FOR FIRE DETECTION"** was prepared by **NG WEI YUAN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	[illegible]
Supervisor	:	Ir. Dr. Mun Hou Kit
Date	:	19/5/2023

Signature	:	
Co-Supervisor	:	
Date	:	

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Ir. Dr. Mun Hou Kit for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me encouragement for completing this project.

ABSTRACT

Conventional fire detection systems based on smoke and flame sensors have been shown to be unreliable in terms of accuracy, response time, and false alarms. This project proposes a smart edge fire detection system that overcomes the limitations of conventional fire warning systems by utilizing deep learning models and edge computing. The system is based on an object detection model for fire detection using the Improved YOLOv5s algorithm, which integrates BiFPN and an additional prediction layer for detecting small targets of fire. The system is designed to be deployed on edge devices, specifically the Jetson Nano B01, and includes a user-friendly interface accessible via Telegram. The workflow of the project is divided into three sections: Google Colab, a workstation, and the Jetson Nano B01. The Google Colab section is used to train the models and generate the Improved YOLOv5s model, while the workstation is used to download the trained model weights and generate the Flask server and MQTT client. The Jetson Nano B01 section is used to install the dependencies, set up the system, optimize the trained model with TensorRT and TorchScript, and connect the Flask server to Node-RED. The final system is capable of starting with user input and sending alerts to Telegram when the fire is detected, with end users able to access the system via Telegram to receive alerts and view the video stream. The results show that the Improved YOLOv5s model is the best option, with an average FPS of 10.5 and a high recall of 0.5, indicating a high capture rate of fire, although with lower precision of 0.541. The latency of the edge computing system is 554ms, making it an efficient and reliable option for detecting fires in real time. Future works could explore the use of a larger dataset and ensemble models to increase accuracy, optimize the model for devices with constrained resources, and improve the user interface and messaging service integration. Overall, this project contributes to the development of efficient and reliable fire detection systems that can be deployed in various settings.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

RM	Malaysian Ringgits
IoT	Internet of Things
IP	Internet Protocol
YOLO	You Only Look Once
BiFPN	Bidirectional Feature Pyramid Network
Faster R-CNN	Faster Region-based Convolutional Neural Network
HTTP	Hypertext Transfer Protocol
FPS	Frame Per Second
mAP	mean Average Precision
UI	User Interface
AWS	Amazon Web Services
CCTV	Closed Circuit Television
SMS	Short Message Service
VPN	Virtual Private Network
HD	High Definition
DECIoT	Distributed Edge Computing Internet of Things
NG112	Next Generation 112
CPSS	Cyber-Physical Social System
QoS	Quality of Service
SMOKE	Scalable edge coMputing framewOrK for early firE detection
SSD	Single Shot Multi Box Detection
CNN	Convolutional Neural Network
SPPFP	Spatial Pyramid Pooling-Fast-Plus
SPPF	Spatial Pyramid Pooling-Fast
CBAM	Convolutional Block Attention Module
VST	Very-Small-Target detection layer
PANet	Path Aggregation Network
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Unit
HSV	Hue, Saturation, Value
MCCL	Multi-scale Context Contrasted Local Feature module
DPPM	Dense Pyramid Pooling module

GPU	Graphics Processing Unit
CUDA	Compute Unified Device Architecture
ONNX	Open Neural Network Exchange
FYP	Final Year Project
GC	Google Collaboratory
TPU	Tensor Processing Units
NAT	Network Address Translation
MQTT	Message Queuing Telemetry Transport
MO	Model Optimizer
IR	Intermediate Representation
OpenCV	Open-Source Computer Vision
FPGA	Field Programmable Gate Arrays
IoU	Intersection over Union
NMS	Non-Max Suppression
FP32	Single-Precision Floating-Point
FP16	Half-Precision Floating-Point
%	Percentage
s	second
ms	millisecond
mbps	Megabits per second
YAML	Yet Another Markup Language

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Fire has existed since prehistoric times and fire-related accidents always follow suit with human activities. Even when technological advancement has provided humanity with a way to reduce the risk of fire outbreaks in this modern era, many fire-related accidents still occurred around the globe and leading to losses, injuries, and casualties. According to Statistics on Fire Breakouts in Malaysia from 2000 to 2019, the cumulative number of fire accidents from 2000 to 2019 in Malaysia reach an astounding number of 602271 cases with 3410 deaths, 9950 injuries, and a loss of RM64,213,600,000 worth of assets (Atikah, 2021).

To cope with the occurrence of fire outbreaks, humanity has developed numerous safety measures for fire accidents. One of the key components of fire safety measures is a fire detection system. Fire detection systems mainly function to detect early signs of fire and give out alerts to notify the people nearby and also the firefighting agency. By detecting the fire early on, the damages done to the property are significantly reduced, and also maximize the fire control effort. Thus, a fire detection system is vital in countering fire outbreaks.(Shokouhi et al., 2019)

In recent studies, Edge computing has recently gained more attention in various Internet of Things (IoT) Applications. This situation is due to the characteristics of Edge computing being responsive, latency independent and data privacy compared to Cloud computing. The incorporation of edge computing in fire detection systems provides a more efficient, reliable, and cost-effective approach to fire safety measures. By detecting fires early, providing real-time alerts, and enabling predictive maintenance, edge computing ensures that potential fire hazards are identified and addressed promptly, minimizing the risks of fire incidents and their associated damages. It is an ideal computing paradigm for designing a fire detection system where an IoT sensing device like an Internet Protocol (IP) camera is used and fast response is required to detect early fire. Thus, a smart edge fire detection system is proposed.

1.2 Importance of the Study

The result of this project is significant for several reasons.

The YOLOv5-based model proposed in this project is a valuable contribution to the advancement of deep learning fire detection systems by integrating both Bidirectional Feature Pyramid Network (BiFPN) and additional prediction layer into the YOLOv5s model, which is part of the You Only Look Once (YOLO) one stage object detection family.

Both YOLOv5s and YOLOv8s are YOLO's family object detection deep learning models. While there have been various methods proposed using YOLOv5s architecture for fire detection (Ahn et al., 2023) (An et al., 2022) (Xue et al., 2022) (Yang et al., 2023). However, currently, there is no detailed comparison between YOLOv8s and YOLOv5s in small target fire detection on edge devices. Thus, this project fulfilled the void in the present studies.

A dataset with numerous optimisations and hundreds of actual cases involving fires has been created. Noise images from a comparable environment to the test site were also included to increase the trained model's accuracy and robustness. We also added unfavorable samples, like those from the sun, the setting sun, reflections, and fluorescent lighting, among others, to lessen the effect of environmental influences on the model.

1.3 Problem Statement

Despite being frequently employed in high-rise buildings and public spaces, conventional fire warning systems based on smoke and flame sensors have been shown to be unreliable in terms of accuracy and response time. According to Pincott et al. (2022), the limitations of the present fire detection system include the inability to detect the exact position of the fire or provide detailed information on its size or spread pattern. As a result, locating the fire's origin becomes difficult because the sensors only detect a general region rather than a specific location, such as a kitchen or bedroom. As a result, the lack of precise information on the fire's origin and behavior hinders the effectiveness of fire control operations.

Furthermore, false alarms are also an acute issue with conventional systems because they have poor accuracy and could provide building occupants with a false sense of security. False alarms also cause individuals to become less attentive to the fire alarm, thereby raising the danger of accidents and fatalities.

In addition, the conventional system has a slow response time, which prevents the detection of fire during the ignition stage, when it is small and controllable. This issue is compounded by the fact that most fire outbreaks are only detected if there are people near the site who can contact the firefighting agency, resulting in slower response times and increased difficulty in saving lives and property.

Various deep learning models like YOLOv3 models and Faster Region-based Convolutional Neural Networks (Faster R-CNN) have been proposed for fire detection by various studies. For instance, Faster R-CNN, despite its ability to detect objects of varying sizes accurately, does not perform well on edge devices due to its complex architecture. YOLOv3 and YOLOv4 models while having a high detection rate of fire, have a large model size. Making it hard for both models to be implemented on edge devices where storage is limited.

Cloud computing is another computing paradigm considered for fire detection systems, but it also has its limitations. Cloud-based fire detection systems have high computational power and can process vast amounts of data, but they require a stable internet connection and can be prone to latency issues. Even though, IoT can benefit from cloud computing's scalability, elasticity, multitenancy, storage capacity, and resource sharing, among other advantages. It is anticipated that the unreliable connection between the cloud and mobile devices will prohibit IoT devices from operating at their peak capability.(Maltezos, E, et al., 2022).

Therefore, there is a need for a smart edge fire detection system that can address the limitations of conventional systems and deep learning models while providing accurate, real-time fire detection with the use of edge computing.

1.4 Aim and Objectives

This project aims to develop a fire detection system deployed with deep learning models on edge computing framework that are capable to detect small target fires and send alerts and footage to users.

Based on the aim, several objectives can be outlined as follows:

a) To develop an Improved YOLOv5s object detection model that performs can detect the small target of fire in real-time.
b) To develop a deep learning edge framework architecture system to send fire alerts and display real-time video streams.
c) To evaluate the performance of the proposed system and model in terms of accuracy, speed, and latency.

1.5 Scope and Limitations of the Study

The scope of this project is development of a deep learning-based fire detection system on an edge computing platform and also the development of an Improved YOLOv5s model. The system takes input from IP camera nodes and produces output through a Flask server with Hypertext Transfer Protocol (HTTP) stream. The system is designed to detect both small and large fire targets in daytime and nighttime. The performance of the system will be evaluated using metrics such as Frame Per Second (FPS), mean Average Precision (mAP), and confusion matrix. Additionally, the system utilizes Node-RED to generate a User Interface (UI) dashboard that takes HTTP stream from Flask server as input. Ngrok is used to expose the Flask server to the internet for remote access to the Node-RED dashboard. Lastly, Node-RED is also used to generate an alert system when fire is detected and sends an alert through Telegram. The Improved model utilize YOLOv5s as base model and add in BiFPN block and additional prediction layer to improve detection on small target of fire.

Different restrictions are imposed for this project in order to focus on improving specific system features. This project is restricted to utilising Node-RED to create a UI dashboard and an alarm system when fire is detected. In this study, other dashboard and alerting systems like Grafana or Slack are not taken into consideration.

Moreover, this project did not utilize cloud services such as Amazon Web Services (AWS) or Google Cloud, which limits the system to the edge computing platform of Jetson Nano B01. Furthermore, the project is focused solely on detecting fires and cannot detect gas leakage. The project is also limited to the use of IP camera nodes as the input source for the system, and other sources such as Closed-Circuit Television (CCTV) cameras or thermal cameras are not considered in this study. The project is limited to the use of Flask server with HTTP stream as the output mechanism for the system, and other output mechanisms such as Short Message Service (SMS) notifications or email alerts are not considered in this study. The project is limited to the use of Ngrok for exposing the server to the internet, and other methods of remote access such as port forwarding, or Virtual Private Network (VPN) are not considered in this study.

1.6 Contributions of the Study

In this project, a working fire detection system will be implemented on an edge device in a real-world setting, together with an object detection model based on YOLOv5s for fire detection. The contributions of this project are listed below.

This project discussed prior research on object detection and edge computing-based fire detection systems.

Other than that, this project also developed a YOLOv5s based model which integrate BiFPN and additional prediction layer to detect small target of fire.

Moreover, a user-friendly user interface that can be accessed from telegram and also issue command to the system from telegram is also proposed in this project.

Lastly, this project proposed a live streaming fire detection system utilizing Flask Server.

1.7 Outline of the Report

There are five chapters in this report. The information in each chapter will be sufficient for the readers. Below is a description of each chapter's outline to give the reader a fundamental grasp of the project,

Chapter 1 describes its background, aims, and objectives, as well as its scope and limitations and the contribution of the study.

The previous research on the related subject is reviewed in Chapter 2 which includes the indoor fire cycle, data preparation, fire detection using edge computing, fire detection using object detection, deep learning framework, and edge device.

The project's technique is described in Chapter 3. Starting with the planning of project activities, moving on to the tools necessary to duplicate this project, and then into the processes necessary to create comparable custom datasets used for training as well as the improved model proposed in this project. This chapter also covers the workflow, system architecture, and model optimizer. The project can be replicated by the readers by following the directions in this chapter.

Chapter 4 describes the evaluation procedures and displays the project's results. The outcomes of the proposed model and the original model as well as models from other studies are then compared and analysed.

This project is concluded in Chapter 5, which also offers ideas for additional research.

CHAPTER 2

LITERATURE REVIEW

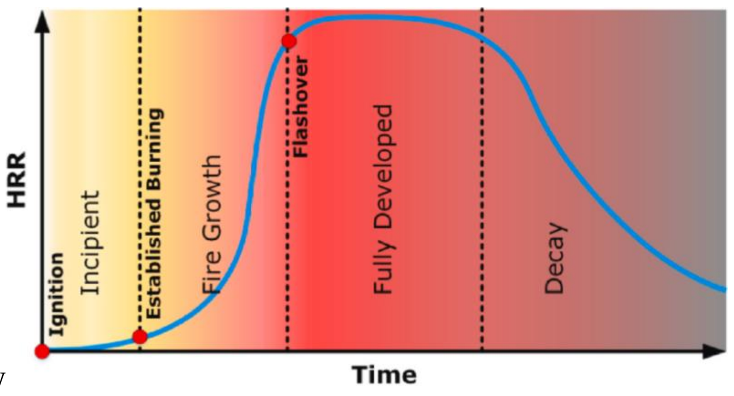
2.1 Introduction

The goal of this research is to provide an edge computing fire detection system and object detection model for small target fire detection. As a result, this chapter will begin with a review of the literature on the indoor fire cycle, data preparation, and then fire detection using two key techniques: (i) edge computing and (ii) object detection. Moreover, this chapter will also review Deep Learning Framework as the platform for deep learning development for fire detection. Lastly, Edge Device will be reviewed later in this chapter as the device for real-world deployment of the proposed system.

2.2 Indoor Fire Cycle

According to J. Pincott et al. (2022), all fires follow a similar life cycle consisting of four stages, as illustrated in Figure 2.1. The first stage is inception, during which the fire may be extinguished if there are fire retardants or insufficient fuel (J. Pincott et al. ,2022). If conditions are favorable, the fire progresses to the growth stage, where a feedback loop of burning fuel leads to increasing heat and further fuel consumption (Ottawa Fire Services, 2021, cited in J. Pincott .et .al, 2022, pg 3). This stage is crucial because it determines the rate of propagation, which depends on factors such as fuel type, quantity, and airflow (J. Pincott et al. ,2022). Fast detection of a fire in the growth stage is important to minimize damage and prevent loss of life.

Once the fire has developed enough, a flashover occurs, where the temperature becomes high enough to ignite exposed surfaces and the fire spreads rapidly. This is the most hazardous stage for firefighters, but a vision-based system could provide information about the fire's stage and location, reducing the risk to firefighters and helping them make informed decisions. After the flashover, the fire burns steadily as long as there is sufficient fuel and air supply. Eventually, the decay phase sets in as the heat and smoke reduce the oxygen supply. This phase leads to extinguishment unless more fuel or air becomes available. Vision-based systems could also be used to aid in the response to fires, particularly through controlled ventilation systems, which can accelerate the decay phase. Figure 2.2 illustrates the normal fire development (solid line) and the potential divergent paths if uncontrolled ventilation or controlled ventilation is in place (dotted lines), as cited from Ottawa Fire Services (2021) by J. Pincott et al. (2022).



w

Figure 2.1: The Lifecycle of A Fire (Ottawa Fire Services, 2021, cited in J. Pincott .et .al, 2022, pg 3)

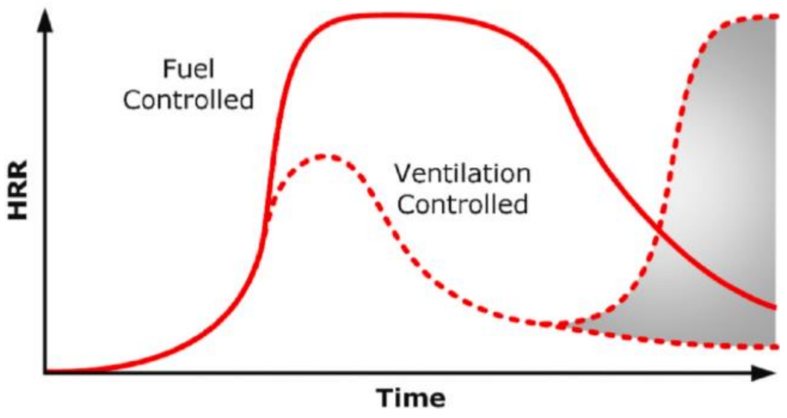


Figure 2.2: Possible Fire Lifecycle Response to Controlled Airflow (Ottawa Fire Services, 2021, cited in J. Pincott .et .al, 2022, pg 3)

In this project, we mainly focus on the detection of fire during the ignition and growth stage to achieve early fire detection.

2.3 Data preparation

To train a object detection model for indoor fire detection, datasets regarding indoor fire must be prepared. According to Ahn et. al, there still lacking in research on indoor fire detection. Only limited research papers have proposed methods regarding the data optimization and image transformation method on datasets to achieve better recall and precision of indoor fire objects. The following Table 2.1 have shown all the papers reviewed for indoor fire datasets.

Among all these studies regarding fire image datasets, one of the most notable study is the one by Pincott, J. et al. (2022), All stages of ignition, propagation, and flashover within the fire images are covered in the datasets by Pincott, J. et al. (2022). This optimization allows the trained model to be able to detect all stage of fire more accurately and also increased the recall of fire detection.

Another optimization that are seen in Pincott, J. et al. (2022), Ahn et al. (2023) and Jain & Srivastava (2022)'s datasets are the addition of images containing region of possible error. These images include reflections and light fitting which easily mistaken as fire. With the addition of these images, a more robust model that can detect fire in all types of lighting environments can be trained (Pincott, J. et al, 2022).

Other than that, both Pincott, J. et al. (2022) and Jain & Srivastava (2022) also have included images of varying image pixel densities or direction to the datasets. These addition helps images to be less sensitive to image focus and also distance. These additions also help to increase the robustness of the final trained model, which further increase the accuracy and recall of the model during testing.

Lastly, both Norkobil Saydirasulovich et al. (2023) and Li et al. (2023) have been seen using data augmentation during the generation of datasets. Data augmentation is the most efficient way to increase the quantity of dataset photos and raise the final accuracy rate, claimed Norkobil Saydirasulovich et al. (2023). The quantity and resolution of the training image datasets have an impact on the performance of trained models.

Table 2.1: Comparison Between Data Preparation Methods Proposed By Various Studies

	Reference	Year	Dataset (Indoor)	Main Highlights
1	Pincott, J. et al.	2022	• The images also include items and scenes found in the inside of various buildings, including workplaces. • Number of images ○ 480 Training ○ 120 Testing • Number of Labels ○ 628 Training ○ 156 Testing	• The use of technique based on TensorFlow Object Detection API workflow process to eliminate duplicate images. • The use of images with a wide range of pixel densities, ranging from 1080p High Definition (HD) to lower than typical CCTV resolutions. • Images were taken throughout the various stages of a fire, such as flashover, propagation, and igniting. • Reflections and light fixtures were present in several training images in places that can lead to mistakes by model. • Bounding boxes were assigned to images that included smoke and fire in close vicinity and overlapped one another.
2	Li et al.	2023	• Small image frames in indoor environments	• Apply geometric transformation of flipping of images only.

			like labs or rooms • Number of images ○ 92 images Training ○ 20 images Testing	• Apply saturation, brightness and contrast enhancement. • Apply data normalization. • Images used as the training input were sized to 320 320
3	Jain & Srivastava	2022	• fire and non-fire images collected from different location of private spaces (residential area). • Images at 640 × 480	• Image dataset at Privacy Level 4, • Images collected with varying distances and directions. • Images converted into grayscale. • Include non -fire images easily mistaken as fire.
4	Ahn et al.	2023	• 10,163 photos in total met the requirements for early fires and indoor fires. • 2 class of flame and smoke were used	• Include non -fire images easily mistaken as fire.
5	Norkobil Saydirasulovich et al.	2023	both diurnal and nocturnal fire images: • Fire Images: ○ 7700 Training ○ 2300 Testing • Non-Fire Images: ○ 2300 Training	• Apply geometric transformations such as flipping and rotation of images. • Apply only brightness and contrast enhancement. • Apply data normalization

6	Jin et al.	2023	• natural fire scenes and web video screenshots • negative samples that cause possible of error such as sun and lighting • Total images ○ 13,843 Training ○ 1061 Validating • 116,709 target objects of fire and smoke were discovered overall.	• Apply Data Augmentation: ○ random rotation ○ random scaling ○ random cropping ○ random fusing ○ change of saturation and chromaticity

2.4 Fire detection using Edge Computing

The current trend of the computing paradigm is cloud computing and edge computing. Edge Computing, being one of the distributed computing models aims to offer scalable and real-time resources near the data sources. Edge computing is typically implemented on the same hardware that gathers the data.

In contrast, cloud computing is another style of computing paradigm in which real-time scalable resources are accessed via the Internet with Web Browser. Edge computing is particularly useful for areas with unreliable connectivity and restricted bandwidth as most of the data processing is carried out at the edge.

There are presently multiple methods being proposed by various studies for fire detection in indoor environments. This section will discuss the general approaches for Indoor Fire Detection and their advantages and limitation.

2.4.1 General Approaches of Fire Detection System with Edge Computing Framework

As mentioned before in section 1.1, a fire detection system is crucial and act as a vital role in countering fire outbreaks, especially indoors. As mentioned above, edge computing provides a lot of convenience and advantages over cloud computing and even conventional systems. Up until the time of this study, there are multiple implementations of edge computing in fire detections but not limited to indoor fires only. Table 2.1 show all the papers reviewed and their main highlight.

Among the edge computing framework reviewed, the most notable methods that are proposed are by Li et al., (2023) and Maltezos, E, et al., (2022). By using Jetson Nano, Li et al., (2023) are able to deploy a custom fully convolutional one-stage object detection framework which uses YOLO architecture, which shows the possibility of deployment of newer versions of YOLO object detection model such as YOLOv5, YOLOv6 and YOLOv7 on Jetson Nano B01.

Whereas for Maltezos, E, et al., (2022), the introduction of IoT sensor nodes to detect fire, smoke and gas leakage to the edge framework eliminates the limited detection range of sensors and the need for wired connections, in turn, provided insight into the deployment of IoT camera nodes which are wireless near the edge device can be an advantage when detecting a fire and during deployment as seen in Figure 2.3.

Another notable point is the use of latency for the evaluation of edge computing framework in Maltezos, E, et al. (2022) and ZHAO et al. (2022).

Table 2.2: Comparison Between Proposed Method For Fire Detection With Edge Computing

Reference	Year	Main Highlight	Type of detectors	Latency	Alert System
Maltezos, E, et al.	2022	• The study combines NG112 emergency call function, edge computing framework, and sensor-node detection approach. • The system can publish geolocated raw sensor values and real-time fire and gas leak alarms. • The study proposes DECIoT, an open-source architecture called Distributed Edge Computing Internet of Things, built on EdgeXFoundry. DECIoT is scalable, secure, adaptable, controllable, and potentially interoperable and modular.	Sensors Nodes	32	NG112 emergency call
Avgeris et al.	2019	• The study proposes a three-level CPSS for early fire detection to assist public authorities in recognizing and addressing emergency situations, such as forest fires. • It suggests the SMOKE Framework, a scalable edge framework for early fire detection. It dynamically allocates resources, supports edge-to-edge and edge-to-cloud applications, manages containerized applications, and maintains	IoT Sensor Nodes	-	-

		Quality of Service (QoS). • It employs a horizontal scaling mechanism that activates and deactivates edge servers based on performance criteria, preventing a loss in QoS.			
Li et al.	2023	• Deployed on Jetson Nano • A fully convolutional one-stage object detection framework for real-time surveillance films used in fire detection is presented in this paper. • Utilises two cameras as part of the system to locate the fire inside a building	YOLOv5s	-	
ZHAO et al.	2022	• multi-sensor-assisted detection is used to solve image detection errors and missed detections caused by poor underground lighting conditions, dust, and camera angles in coal mines. • Utilized improved YOLOv5s model	YOLOv5-as (Improved YOLOv5s)	238 (edge) 338(cloud)	Conventional Fire Alarm

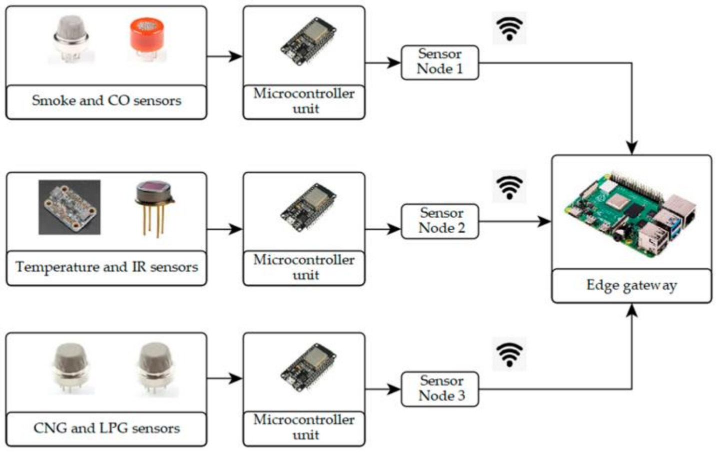


Figure 2.3: SB112 smart building sensor system (Maltezos, E, et al., 2022)

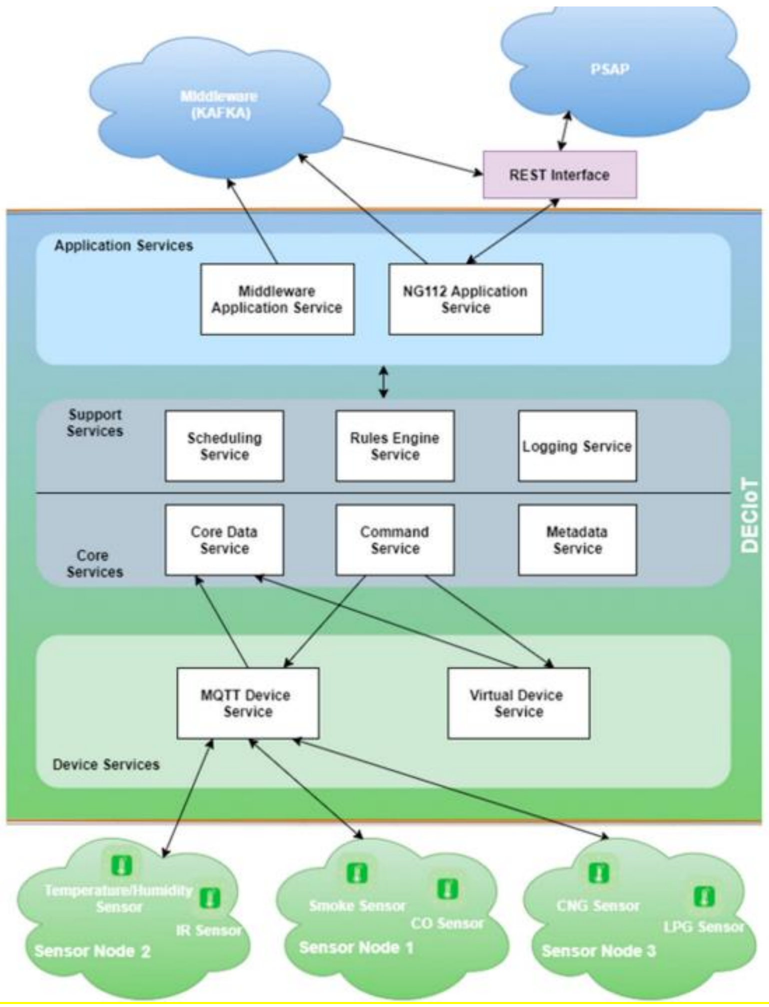


Figure 2.4: DECIoT architecture (Maltezos, E, et al., 2022)

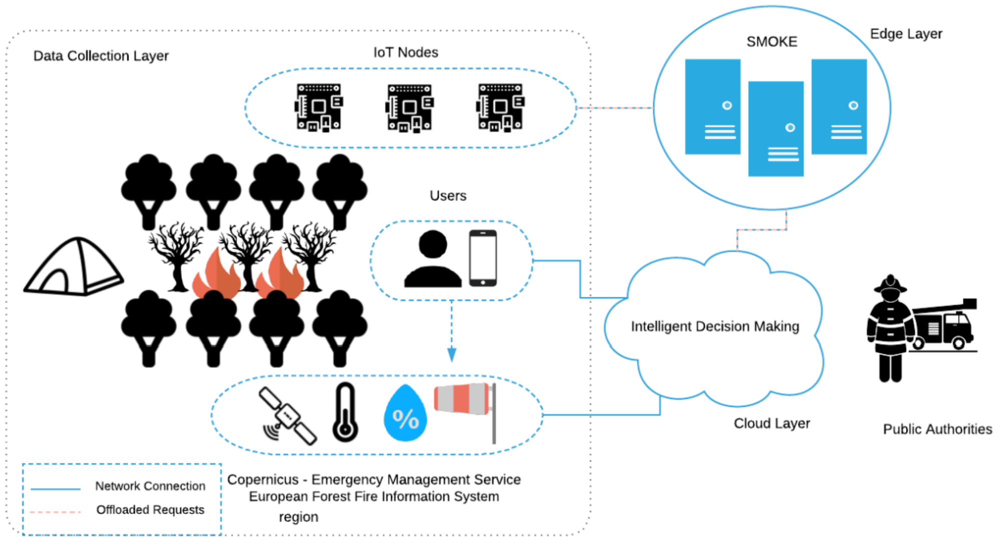


Figure 2.5: Proposed CPSS Architecture (Avgeris et al., 2019)

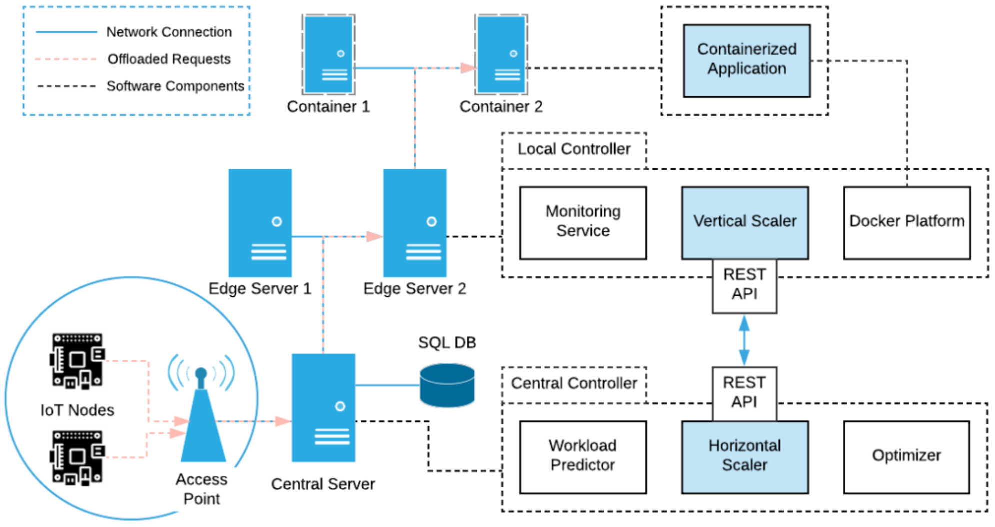


Figure 2.6: Proposed SMOKE Framework (Avgeris et al., 2019)

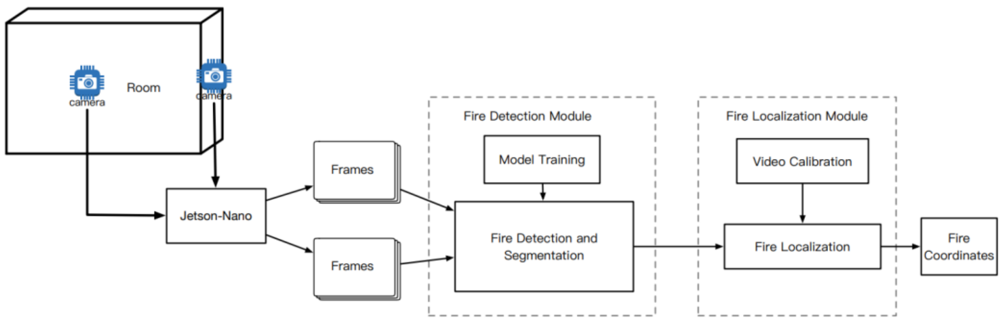


Figure 2.7: The design of the proposed Real-Time Fire Detection and Localization Framework (Li et al., 2023)

2.5 Fire detection using Object Detection

In this section, object detectors using state-of-the-art convolutional neural network (CNN) models proposed by various studies to detect fire are reviewed. Object detection involves both object classification and localisation processes, where the object is classified as a specific class and localise the object in an image using bounding boxes respectively. There are currently 2 types of object detector models, one stage and two-stage.

2.5.1 One Stage Detectors

A One Stage Detector applies a single neural network to the entire image CNN for object detection inferred in a single pass (Cantero, Esnaola-Gonzalez, Miguel-Alonso and Jauregi, 2022). This form of architecture allows higher computational speed but lower accuracy than Two Stage Detectors. The most commonly found One Stage Detectors are YOLO and Single Shot Multi Box Detection (SSD). In this section, relevant works involving fire detection with One Stage Detector are reviewed and displayed in Table 2.2. The reviews are not limited to indoor fire scenarios but also include forest fires due to the limited studies.

Table 2.3: Comparison Between Proposed Method For Fire Detections With One Stage Detectors

	Reference	Year	Detector	Main Highlights
1	Zhao, et al.	2022	Fire-YOLO a) Based on YOLOv3	• A model for small target fire object detection • Uses Improved EfficientNet instead of DarkNet-53 as the backbone. ○ Depthwise separable convolution utilised in MBConv allows the depth wise separable convolution to extract small target features at various granularities while requiring less computation. ○ Achieved improvement in feature extraction ability for small target detection and model size minimization
2	Norkobil Saydirasulovich et al.	2023	YOLOv6	• To evaluate the system's ability to recognise fire-related objects, multi-class object identification utilising random forests, k-nearest neighbours, support vector, logistic regression, naive Bayes, and XGBoost was done on the SFSC dataset. • XGBoost classifier provides the highest accuracy when attempting to identify objects
3	Li et al.	2023	Fully Convolutional One-Stage Object Detection • tested with different	• They proposed their own propose fully convolutional one-stage object detection framework based on YOLO architecture. • Include a fire localization framework that can calculate the position of fire in the image.

			backbones: a) EfficientNet b) ShuffleNet c) RepVGG d) CSPNet	
4	Mukhiddinov et al.	2022	Improved YOLOv4	• A Convolutional block attention module is added. • A H-swish activation function is used to ensure the elimination of gradient explosion
5	Ahn et al.	2023	YOLOv5	• The suggested "video image fire detector" in UL 268 B was used to test the EFDM's ability to detect fire. • The ISO 7240 test standard for fire detection and alarm systems was used to compare the EFDM's detection time to that of conventional fire detectors.
6	An et al.	2022	YOLOv5 • With dynamic convolution	• The clustering of anchor boxes is optimised using the K-mean++ technique, which results in a significantly lower classification error rate. • Network heads are pruned to improve detection speed. • Applicable for both short-range indoor fire identification and long-range outdoor fire detection
7	Xue et al.	2022	YOLOv5	• The study developed the SPPFP module by modifying the SPPF module of YOLOv5.

				This enhancement enables the model to gather global data about miniature forest fire targets. • The model was further improved by incorporating the CBAM attention module to enhance the identification of small forest fire targets. • To enhance the detection of extremely small forest fire targets, the model includes a very-small-target detection layer (VST) and adjusts the PANet in the BiFPN. • Transfer Learning is used in the study.
8	Yang et al.	2023	KPE-YOLOv5	• The model is assisted in learning the characteristics of small targets by an additional prediction layer that has fewer downsampling times and improved resolution of small targets. • scSE attention module was added, which increases the network's capacity to learn crucial features by summing the channel- and spatial-level information from the input feature maps to increase their excitation. • The clustering of anchor boxes is also optimised using the K-mean++ technique, which results in a significantly lower classification error rate.

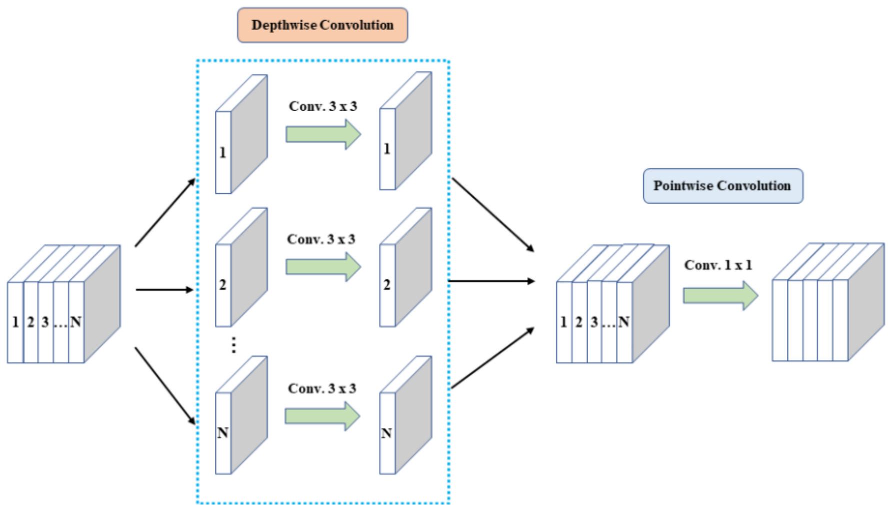


Figure 2.8: Architecture of deep separable convolution (Zhao, et al. ,2022)

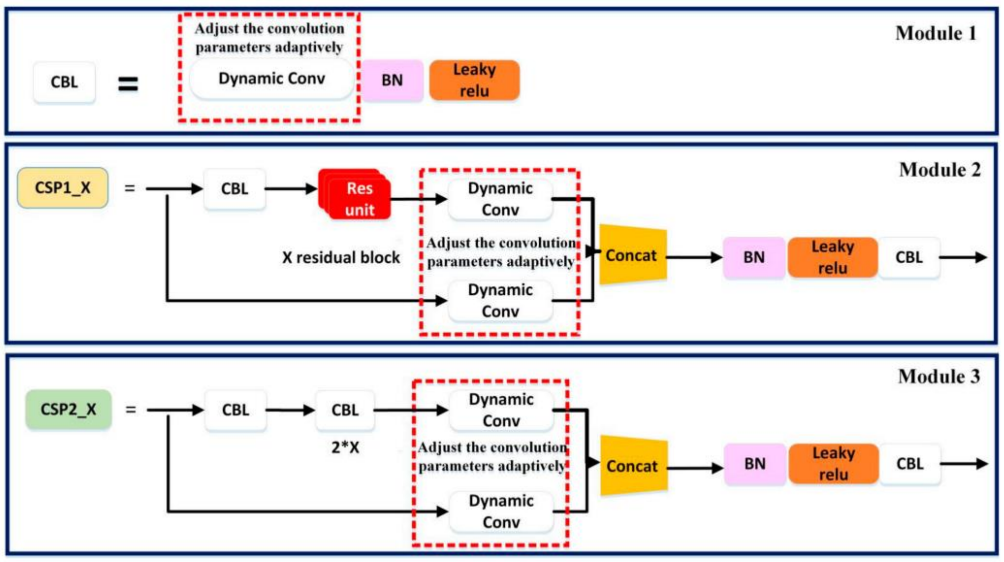


Figure 2.9: Improved YOLOv5 convolution layers with dynamic convolution (An et al. ,2022)

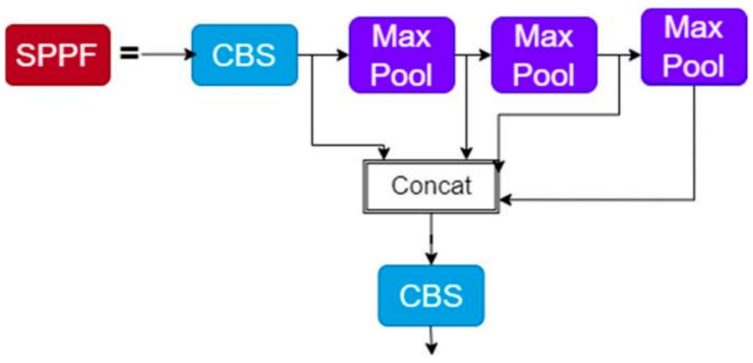


Figure 2.10: The structure of SPPF (Xue et al., 2022)

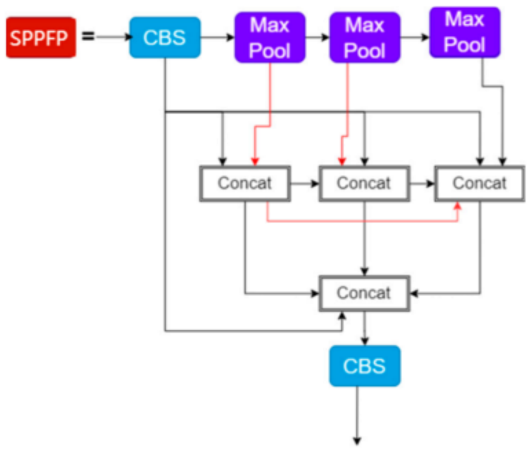


Figure 2.11: The structure of SPPFP (Xue et al., 2022)

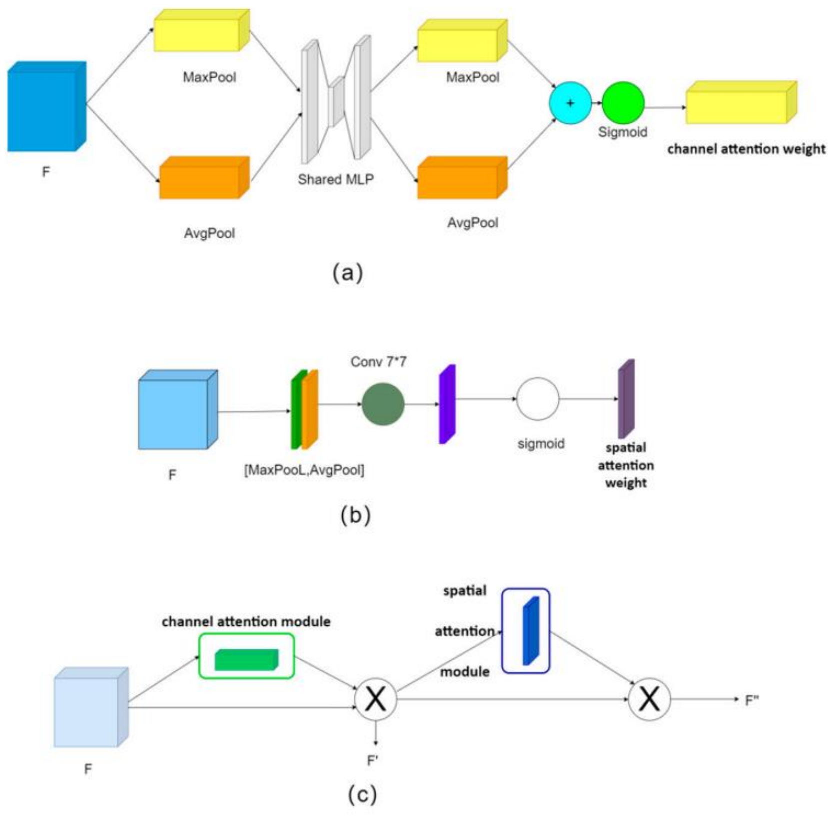


Figure 2.12: (a) The structure of the channel attention module. (b) The structure of the spatial attention module. (c) The structure of the CBAM attention module (Xue et al., 2022)

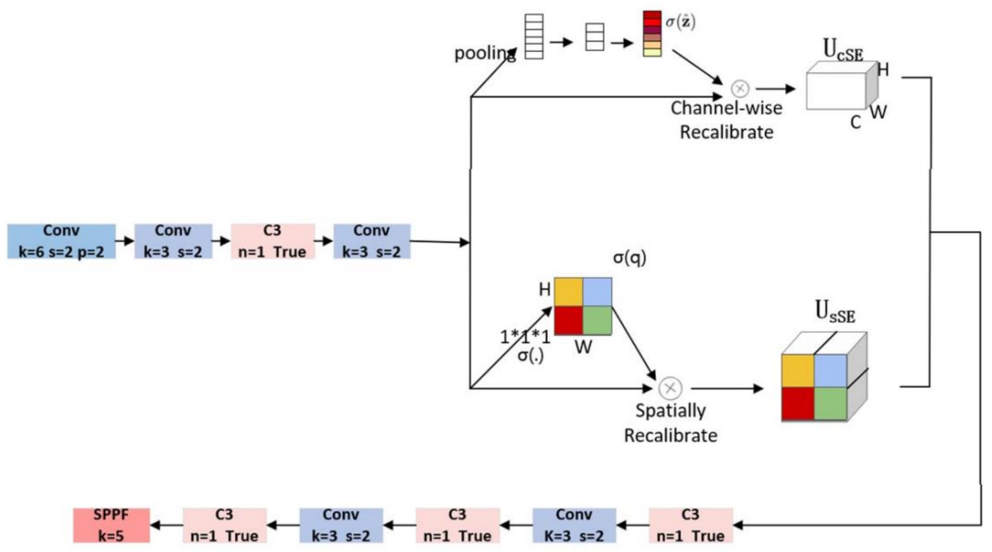


Figure 2.13: Backbone network after adding scSE module (Yang et al.,2023)

Among all one stage detectors, one of the methods used most is the use of K-mean++ algorithm to optimize anchor box to reduce classification error by both Yang et al. (2023) and An et al. (2022). With this optimization they can reduce error when detecting small target object.

Another notable method is the adjustment of the PANet to the BiFPN by Xue et al., (2022). This change allows simpler and rapid multi-scale feature fusion.

Lastly, the way of Yang et al. (2023) creating a prediction layer is also highly noteworthy. By having additional prediction layer, the model has fewer downsampling times and improved resolution of small targets.

2.5.2 Two Stage detectors

A Two Stage Detector are made up of 2 stages. At the initial stage, region proposals for object detection are generated. Following that, computes each proposed region is computed and both the classification result and the bounding boxes is extracted at the second stage (Cantero, Esnaola-Gonzalez, Miguel-Alonso and Jauregi, 2022). In contrast to One Stage detector, Two Stage Detector have higher accuracy but also lower computational speed than one stage model. Faster R-CNN is one of the most popular among Two Stage Detectors.

Table 2.4: Comparison between proposed methods for fire detection with two stage detectors

Reference	Date	Detector	Main Highlight
Khan & Khan	2022	FFireNet • The model utilizes the pre-trained convolutional base of the MobileNetV2 architecture. • Adding fully connected layers on top of the convolutional base • The model makes use of the feature extraction capabilities of its first layers, which were taught generic features using the ImageNet dataset.	• Used the Stochastic Gradient Descent (SGD) as an optimizer • Used the sigmoid and rectified linear unit (ReLU) as activation functions
Pincott et al.	2022	• Faster R-CNN with InceptionV2 • SSD MobileNet V2	• Both detectors were tested on videos that included a simulated bedroom and living room as well as a CCTV video of an office space in order to assess the efficacy of the method. • SSD MobileNet V2 model showed lower missed detection

			results but also lower accuracy than Faster R-CNN with InceptionV2
Ryu & Kwak	2022	a) Inception-v3 model (CNN model)	• The image is preprocessed using Hue, Saturation, Value (HSV) color conversion and Harris corner detector before object detection. • The motion of smoke is detected using optical flow algorithm based on Lucas -Kanade Method.
Huang et al.	2023	deformable End-to-End Object Detection with Transformers (deformable DETR) a) Based on R-CNN	• A Multi-scale Context Contrasted Local Feature module (MCCL) and a Dense Pyramid Pooling module (DPPM) were integrated to improve the feature extraction process in order to better detect small or undetectable smoke features. • To create precise bounding boxes that completely enclose the smoke object, a unique iterative bounding box combination method was presented. • Detect both forest fire and smoke

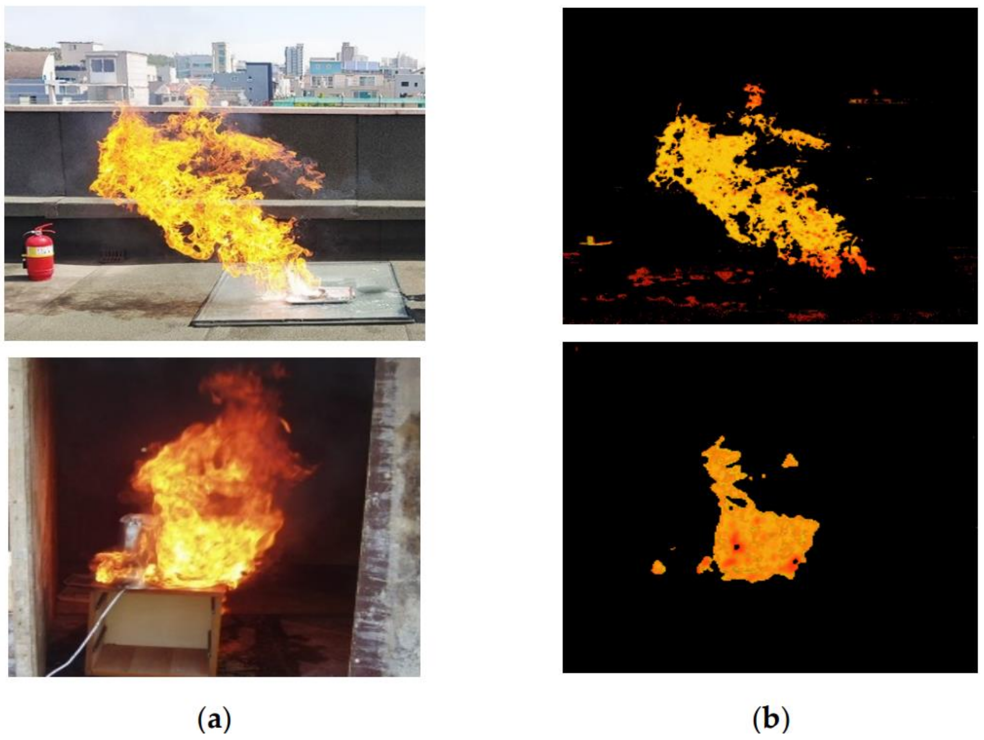


Figure 2.14: HSV color conversion of flame image. (a) original images; (b) HSV color conversion in the specified range.

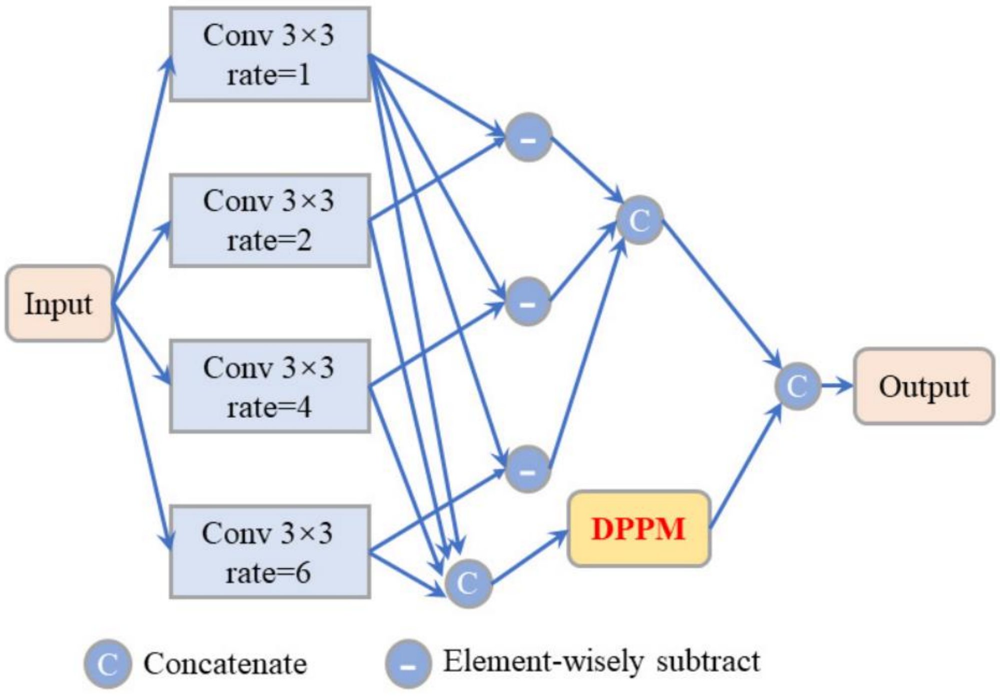


Figure 2.15: Illustration of MCCL Feature module. (Huang et al., 2023)

The most notable methods among the two stage detectors are the use of SGD as an optimizer by Khan & Khan (2022). Another popular optimizer being

Adam. Adam is a more complex optimization algorithm than SGD and able to handle complex models like YOLOv5 and YOLOv8, but it has more hyperparameters that need to be tuned. While SGD show faster convergence and more memory efficient, still more sensitive to learning rate and higher variance.

Another notable method is the use of pre-processing before inputting into the model, which allow higher accuracy and recall rate as the object in the image

2.6 Deep Learning Framework

2.6.1 TensorFlow

Unquestionably, TensorFlow is one of the most widely used deep learning frameworks. The Google Brain team started working on it, and it now supports wrapper libraries and a number of programming languages, such as Python, C++, and R. Both desktop and mobile platforms can access TensorFlow. It also comes loaded with top-notch documentation and walkthroughs to guide you.

TensorFlow is designed to be more scalable and optimized for production use cases that require high performance and distributed training across multiple Graphics Processing Unit (GPU)s and even multiple machines. TensorFlow also provide visualization toolkit, TensorBoard, which allow effective data visualization of network modeling and performance.

However, TensorFlow has a static computation graph that requires developers to define the structure of a model upfront, which can be more challenging for beginners. TensorFlow also use more memory than PyTorch as TensorFlow stores the entire computation graph in memory. This make TensorFlow less memory efficient and can lead to longer training times, especially when working with large models and datasets.

2.6.2 PyTorch

Torch is a framework for scientific computing that offers a wide range of support for machine learning techniques. It is a framework for deep learning built on the Lua programming language. On the other hand, PyTorch is a modern Torch implementation that is based on the Python programming language. This suggests that users of PyTorch who have a basic understanding of Python can start creating deep learning models.

PyTorch, being implementation of torch, also employs Compute Unified Device Architecture (CUDA) along with C/C++ libraries for the processing on GPUs PyTorch is designed to take advantage of the parallel processing capabilities of GPUs, which can significantly speed up the training and inference of deep neural networks.

PyTorch is also designed to be scalable, flexible and user-friendly, making it well-suited for a variety of use cases, from research to production. It provides a range of powerful tools and libraries for building and training deep neural networks. Tools such as automatic differentiation, dynamic computation graphs, and distributed training across multiple GPUs and machines allow users to build complex models with ease, while having a short training time. One of the powerful libraries that PyTorch offers is the TorchVision library, which provides a collection of datasets, model architectures, and image processing utilities for computer vision tasks.

Still, PyTorch have limited visualization for the results of trained models and require third parties integration with TensorBoard.

2.6.3 Summary of deep learning framework

Table 2.5: Comparison between TensorFlow and PyTorch

	TensorFlow	PyTorch
Compatibility	Compatible with most embedded device	
Learning Curve	Steep learning curve	Reasonable learning curve
User Friendly	No	Yes
Community Support	Excellent documentation and community support	
GPU /CPU Support	Strong support for GPUs	
Training Time	Long	Short
Visualization	Graph visualization and queues using TensorBoard	Limited visualization (require 3rd party integration for TensorBoard)
Optimization	○ TensorFlow Lite ○ Open Neural Network	○ TorchScript

available	Exchange (ONNX)	○ ONNX

Thus, **PyTorch** deep learning framework are considered using in this project due to its more reasonable learning curve, user friendliness and short training time.

2.7 Edge Device

Currently in the market, there are currently 2 edge devices for consideration, which are Jetson Nano B01 and Raspberry Pi 4B.

Table 2.6: Comparison between Raspberry Pi 4B 4GB and Jetson Nano B01

	Raspberry Pi 4B 4GB	Jetson Nano B01
Processor	Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit Central Processing Unit (CPU). @ 1.5GHz	Quad-Core ARM Cortex-A57 64-bit @ 1.43 GHz.
Memory	4GB SDRAM	4-GB LPDDR4 RAM
GPU	VideoCore VI GPU @ 500MHz	128-core NVIDIA Maxwell GPU @ 921MHz
Ethernet	Gigabit Ethernet Bluetooth and Wi-Fi	

Although Raspberry Pi 4B is more energy efficient in comparison to Jetson Nano due to its low computation GPU on aboard, Jetson Nano have 2 advantages over Raspberry Pi.

One of the key advantages of Jetson Nano is the GPU. As opposed to the Raspberry Pi, which uses a low-performance GPU (Cortex A72 @ 700MHz), the Jetson Nano uses a high-performance GPU (Maxwell-128 core @ 912MHz), resulting in it being more suited to executing high-end applications in the fields of AI, ML, robotics, and other technologies. The Maxwell GPU could be utilised for deep learning and offers complete support for graphic content. It is equipped with the capacity to handle up to 1080p video feeds at once while processing numerous video streams efficiently. On the other hand, the Raspberry Pi's on-chip Cortex GPU lacks the processing capacity necessary to handle complex computational tasks.

Another advantage of Jetson Nano over Raspberry Pi 4B is being compatible with various deep learning tools developed by Nvidia, which include TensorRT and CUDA. These tools allow the deep learning task to be done more efficiently. Thus, in this project Jetson Nano B01 is chosen as the edge device as seen in Figure 2.16.

Figure 2.16: Jetson Nano B01 (Cytron Technologies, n.d.).

2.8 Summary

Based on the review from J. Pincott .et .al, (2023), fire is the easiest to stop during inception stage, when the fire is just started and weak. Thus, the proposed system must be capable to detect small fire. There aren't many works in the field of fire detection that combine deep learning and edge computing. Thus, a fire detection model using deep learning and edge computing will be developed.

Besides, based on the study reviewed, it can be seen that the custom dataset used must include images containing region of possible error to reduce the possibilities of false positive and also uses data augmentation during the generation of datasets to ensure higher accuracy.

In addition, the uses of YOLO architecture in Jetson Nano B01 by also show possibility of deploying newer model such as YOLOv5 and YOLOv8 models.

Among the studies reviewed regarding deep learning CNN models, a choice is made to implement both addition of concat BiFPN layer by Xue et al.

(2022), additional prediction layer by Yang et al. (2023), the use of SGD optimizer by Khan & Khan (2022) on proposed model.

Furthermore, in between PyTorch and TensorFlow framework, PyTorch is chosen due to its user friendliness and faster training time.

Lastly, after reviewing the edge device available, Jetson Nano B01 is chosen.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

In this chapter, the methodology of the project will be explained. The details such as tools to be used, data preparation and preprocessing, model architecture, system architecture, evaluation matrix, and training of the models required to replicate this project and obtain reproducible results will also be discussed in this chapter. Through careful planning and managing, project activities and the budget used needed to accomplish the objectives will also be discussed and justified.

3.2 Project Activities Planning

The whole project is being divided into 2 parts (Project Part I and II) with a period of 2 long semesters. In this semester, only Project Part I, which consisted of 4 major activities will be completed.

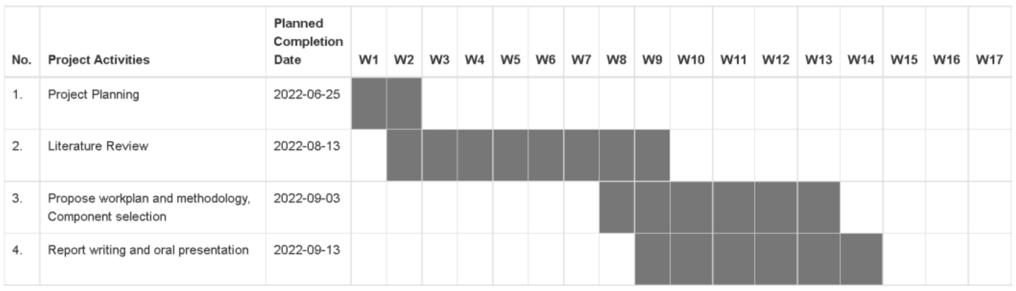
3.2.1 Project Part I

First and foremost, at the beginning of the project, a detailed discussion with the Final Year Project (FYP) supervisor, Ir. Dr. Mun Hou Kit was conducted to have a better understanding of the requirements and expected outcomes of this project. Through this discussion, possible problems encountered, and project duration are also being identified, while the Gantt chart shown in Figure 3.1 is being completed.

Following the discussion, a thorough literature review was carried out to incorporate ideas from other researchers in order to innovate on the currently available proposed method for the project. Multiple works that cover object localization relevant to fire detection as an input model as well as the implementation of edge computing were included and analysed professionally. In the literature review, these works are being reviewed in depth while all necessary aspects such as advantages and limitations are being highlighted. The literature review started right after the first meeting with Dr. Mun, and a time period of eight weeks was spent to accomplish it.

Following a thorough investigation of the works of other researchers, the methodology and work plan for this project were brainstormed. The project's methodology, programming environment, and types of hardware and software were all suggested. Moreover, every component used in the project had gone through a series of filtering before being proposed. The standard for filtering in this project includes the power consumption, price-to-performance ratio and size. To confirm the suitability of the project's use of the components, preliminary testing was also conducted. It took six weeks to reach this milestone.

Throughout part I of the project, many explored ideas and concepts have been approved and criticised. Those ideas and concepts that went through the screening were then documented in this proposal. Since this proposal only constitutes a portion of the whole project, only brief information is provided for the methodology and preliminary result sections. To ensure that there are no grammatical errors, this report was revised multiple times before submission. The report writing process took five weeks in total. In order to communicate the project's main idea and the most recent developments in project part I, an oral presentation was given during the final week of that component.



No.	Project Activities	Planned Completion Date	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17
1.	Project Planning	2022-06-25	■	■															
2.	Literature Review	2022-08-13		■	■	■	■	■	■	■	■								
3.	Propose workplan and methodology, Component selection	2022-09-03								■	■	■	■	■	■				
4.	Report writing and oral presentation	2022-09-13									■	■	■	■	■	■			

Figure 3.1:Gantt chart for part I of this project

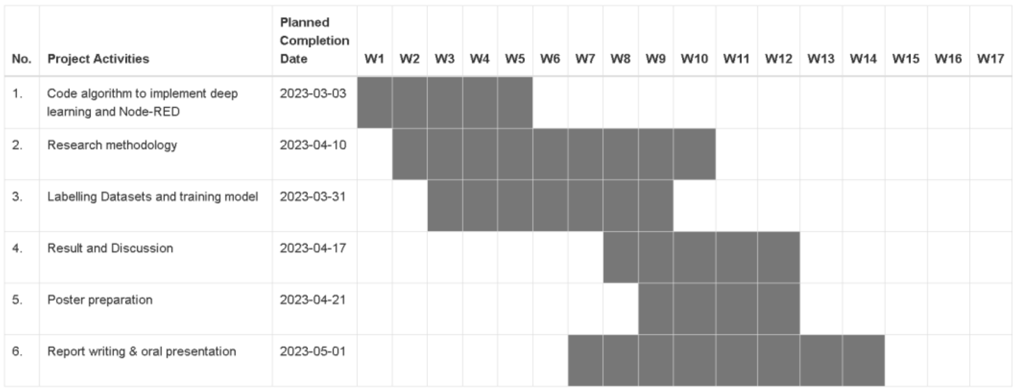
3.2.2 Project Part II

Project part II will be carried out within 14 weeks of January semester 2023. In project part II, six activities are to be accomplished.

Firstly, the software development and prototype building will be carried out starting from the first week. The works done in activity 1 include adopting deep learning models on Jetson Nano B01, setting up UI with Node-RED and building a prototype. As software development requires a long time period, five weeks are given, and this activity is expected to be done on 3 March 2023. Following activity 1, research methodology was carried out from the second week to the tenth week to determine possible improvements to fill up the technological gaps. Along with researching methodology, labelling datasets and training object detection models will be carried out from the third week to the ninth week. By labelling datasets and training models, the model's performance can be evaluated, and the problems can be recognized earlier. Also, the model can be improved to overcome the problems encountered and boost the prototype’s performance. The training also helps to improve the methodology proposed.

Right after labelling datasets and training the model, an experiment is carried out and the result is obtained starting in the eighth week. A discussion is then made on the obtained results to analyse the fire detection system’s performance. Besides, an evaluation of the adopted algorithm performance was carried out in this activity. The performance of YOLOv5 will be evaluated in terms of mAP and FPS. This activity was conducted until the twelfth week as the prototype may be modified or improved, which may affect the performance. All the results will be obtained before 7 April 2023. The deadline for the submission of the poster is week 12. The final design of the poster will be done on 21 April 2023 as well before the due date.

Finally, the report writing started in the seventh week to ensure the time is sufficient to complete the report properly. Along with the final report, the presentation slides also be prepared. The report and oral presentation are expected to be done on 25 April 2023, completing the project flow. All the activities mentioned will be shown in Figure 3.2 below.



No.	Project Activities	Planned Completion Date	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17
1.	Code algorithm to implement deep learning and Node-RED	2023-03-03	■	■	■	■	■												
2.	Research methodology	2023-04-10		■	■	■	■	■	■	■	■	■							
3.	Labelling Datasets and training model	2023-03-31			■	■	■	■	■	■	■								
4.	Result and Discussion	2023-04-17								■	■	■	■	■					
5.	Poster preparation	2023-04-21									■	■	■	■					
6.	Report writing & oral presentation	2023-05-01							■	■	■	■	■	■	■	■			

Figure 3.2: Gantt chart for part II of this project

3.3 Tools to use

3.3.1 Training Platform

The model's training and development will be done on two different platforms: a workstation and Google Collaboratory (GC). This section will define the difference between both platforms.

3.3.1.1 Google Collaboratory

GC is a free web-based interactive computing platform developed by Google Research. Without having to set up a local development environment, it is intended to give researchers, teachers, and developers a means to write and run Python code in a web-based environment through their Jupyter Notebook-based interface. GC also provide computer resources like GPU and Tensor Processing Units (TPUs), which are essential for training and testing machine learning algorithms such as deep learning and object detection.

GC also includes several other features that are useful for developers and researchers, including a visualization function for analyzing the results of trained models through Tensorboard and ClearML. It also allows us to mount our google drive to store model weight files, and results and gain access to personal datasets. Different GPU models, including Nvidia K80s, T4s, P4s, P100s, and Tesla V100s, can be accessed via GC. However, the availability of GC resources cannot be relied upon, and they are not limitless. To continue offering its service without charging, GC must be able to change the hardware's availability to support more customers and offer the required computing power (Google, 2023b). Free tier GC only provides runtime containing all the computing resources needed for deep learning for only 12 hours a day, the runtime will be halted and generated files are deleted if not saved in google drive.

GC Pro and GC Pro+ are the two premium variations of GC that Google also provides. Access to faster GPUs and TPUs, more memory, and longer runtimes are the main advantages of these commercial editions (Google, 2023a). There is no guarantee as to the kind of GPU offered, even though better GPUs are only available to paying users. Additionally, only 24 hours, or roughly twice as long as the free version, is the maximum runtime for paid versions. Any GC version may experience problems when storing the generated result CSV file and model weight file if the session is not continuous. However, the majority of well-known Python packages and dependencies are easily accessible in GC, so we'll keep the training epoch count at 50 for this project. Thus, **training of deep learning models will be performed in GC as the model weight will be stored in google drive immediately.**

3.3.1.2 Workstation

Although GC have the benefits of high computational power and user friendly interface, it is incapable to access the local network or storage in your workstation. Every files that are not stored in Google Drive is delected after runtime stopped. Thus, the workstation is used for testing the IP camera and the algorithm.

The advantages of a workstation include continuous runtime and precise hardware specs. We used a workstation with a GeForce GTX 1650 graphics card as the GPU for this project. The onboard Random Access Memory (RAM) is 8GB, while the GPU has 4 GB of memory. The workstation's Solid State Disc (SSD) has a capacity of 250GB, which is sufficient for storing all the required datasets and software dependencies. Python, C++ and Java environments can be deployed on workstations without compatibility issues.

In this project, we will be generating a Python script using Pycharm, a Node-RED flow and also testing the functionality of the proposed system in the workstation. After downloading the necessary dataset and programmes' dependencies, 200 GB of drive space remains.

Since there is no time limit restriction, validation and testing of the proposed system will be done on the workstation.

Table 3.1: Comparison between GC and Workstation

	GC	Workstation
GPU	Nvidia Tesla V100	GTX1650 4GB
Available storage size	Roughly 77GB	Roughly 520GB
Advantages	• Fast training speed • No computational burden on the host	• Unlimited runtime • Have support for libraries and software needed. • No upload required, directly access webcam and storage for testing
Limitation	• A limited runtime of 12 hours a day • Runtime will be stopped after idling for more than 4 hours. • Unable to access host’s webcam or storage, require file to be uploaded to google drive.	• Required to download dependencies manually. • Local storage is occupied.

3.3.2 Ngrok

With the use of Ngrok, a secure server tunnelling service, developers can expose their locally hosted servers to the public internet through secure tunnels, even when they are behind NATs and firewalls. Developers can distribute their web applications, APIs, and webhooks with others for testing and development while eliminating the need to configure port forwarding or firewall restrictions.

Ngrok offers both free and paid plans that come with different features and capabilities. The paid plans offer advanced features like custom domains, reserved IP addresses, and additional tunnel endpoints. This tool is particularly useful for developers who need to test their applications in a production-like environment before deploying them to a live server.

In this project, we will be used generating a Dashboard UI with Node-RED that is running locally. In order to allow end users to access the dashboard anywhere with internet connections. The dashboard's internet protocol (IP) is encrypted and expose to the internet through the use of Ngrok. This addition of Ngrok server allow the privacy of the end users to be ensured while providing services.

3.3.3 Mosquitto Message Queuing Telemetry Transport

IoT and machine-to-machine (M2M) applications benefit greatly from the use of the messaging protocol Message Queuing Telemetry Transport(MQTT). MQTT is intended to be compact and effective, making it appropriate for usage in circumstances with constrained resources. Messages are exchanged via MQTT through a broker, an intermediary. Publishers deliver their communications to the broker, who then shares them with subscribers.

MQTT operates on a publish/subscribe messaging model, which involves devices being divided into two groups: publishers and subscribers. Publishers are responsible for sending messages to the broker, while subscribers receive the messages from the broker. This model enables efficient communication between devices, as messages are only sent to those devices that have expressed an interest in receiving them. Overall, MQTT's lightweight design and efficient messaging model make it an attractive choice for IoT and M2M applications.

Mosquitto MQTT is an open-source message broker that implements the MQTT protocol. It is a widely used messaging system that enables efficient communication between devices. Mosquitto MQTT offers various features, such as message persistence, authentication, access control, and

SSL/TLS (Secure Sockets Layer/Transport Layer Security) encryption, which enhance the security and reliability of the messaging system.

The combination of MQTT and Mosquitto MQTT provides developers with a lightweight, efficient, and secure messaging system that facilitates real-time communication between devices. This combination is particularly useful for IoT and M2M applications, including home automation, industrial automation, healthcare, and transportation.

As both Flask and Node-RED can be configured to act as MQTT clients that can publish and subscribe to MQTT topics on an MQTT broker. This allows simple integration of Mosquitto MQTT as the bridge between the flask server and Node-RED flows.

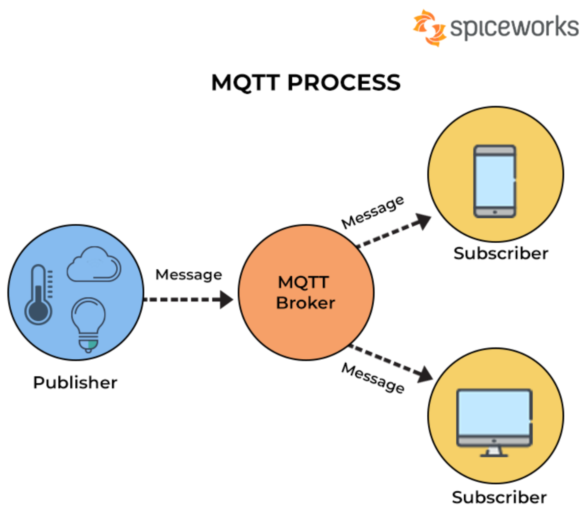


Figure 3.3:MQTT Process (BasuMallick, 2022)

3.3.4 Node- RED

Node-RED is a flow-based development tool that allows users to bridge connections between hardware devices, application programming interface (API) and online services as part of IoT. Node-RED also provides a web browser-based flow editor, which can be used to create JavaScript functions as a node. In order to use Node-Red, JavaScript runtime environment, Node.js must be preinstalled.

In this project, Node-RED will be set to receive processed video footage and alert from Flask Server onboard Jetson Nano B01 through Mosquito MQTT. If an alert is received, Node-RED will send an alert to the end users through telegram and stream the footage on the UI dashboard.

Figure 3.4 show the Node-RED flow used in this project. The injection node at the left-hand side function to initiate the system.

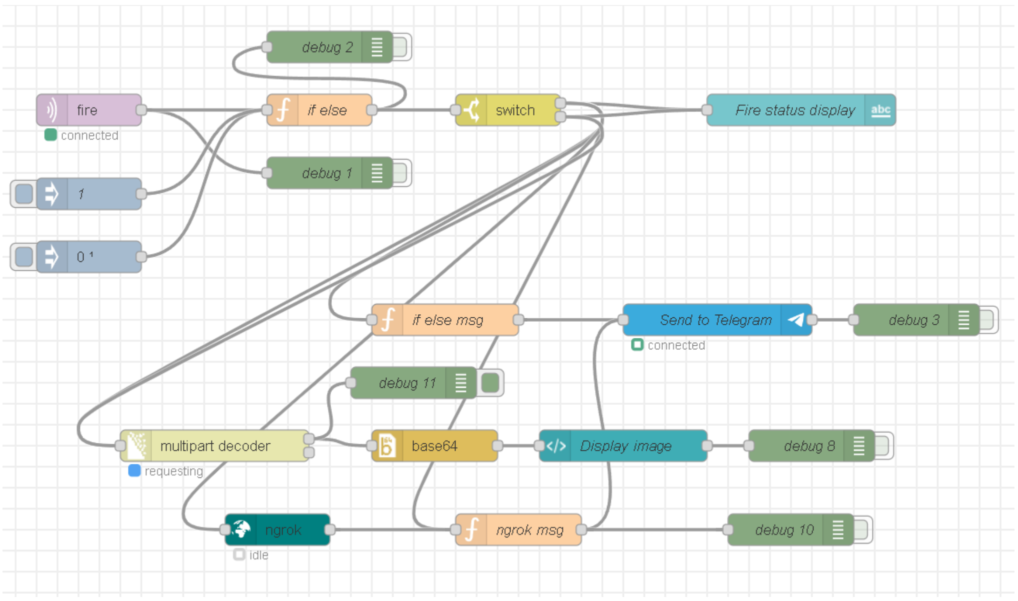
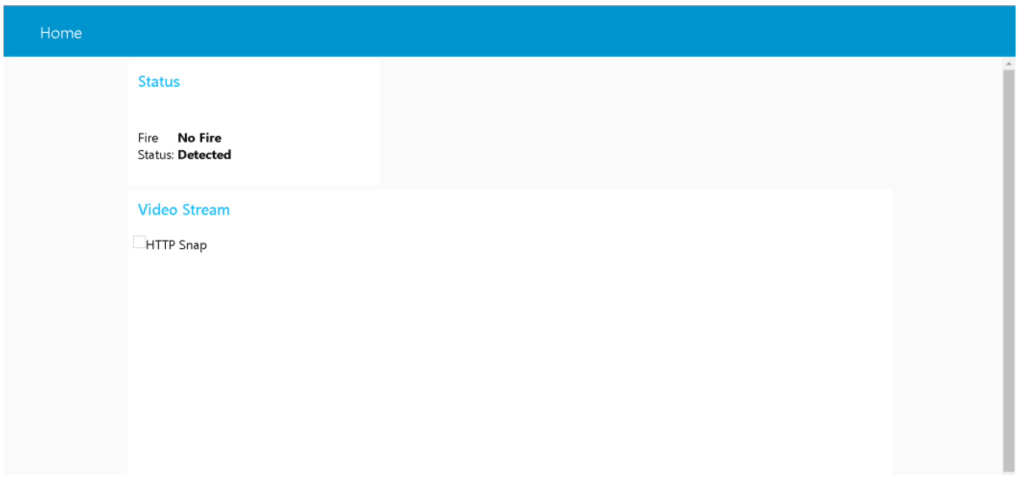


Figure 3.4: Node-RED Flow

A dashboard-like User Interface will also be made with Node-RED to show the video footage and also the message “fire detected” when a fire is detected. The Dashboard UI produced using Node-RED can be shown in Figure 3.5.



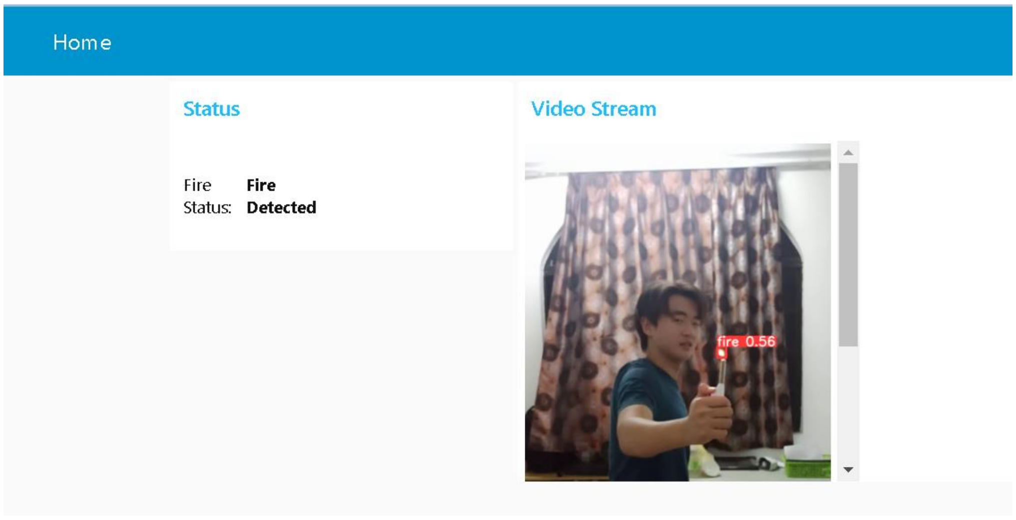
Status

Fire Status: **No Fire Detected**

Video Stream

HTTP Snap

Figure 3.5: Dashboard UI produced using Node-RED when fire not detected.



Fire Status: Fire Detected

Figure 3.6: Dashboard UI produced using Node-RED when fire is detected.

3.3.5 Flask server

Flask is a Python web framework that allows developers to build web applications quickly and easily. Flask provides a built-in development server that can be used to run and test web applications on a local machine.

The Flask development server is lightweight and simple, making it suitable for development and testing purposes; Flask is also an ideal choice for beginners who are new to web development. Moreover, Flask is also a flexible web framework that allows developers to choose the tools and libraries they want to use in their projects, giving them more control over the development process. Flask server also has a large community base, which has a large and active community of developers, thus large quantity of resources and libraries are available to help with development and problem-solving.

However, it is not designed to be used in production environments, as it lacks the performance, security, and scalability features required for serving web applications to a large number of users. While a more robust server is needed in production environments, the Flask development server is still useful for testing locally.

In this project, the Flask server is used to generate an HTTP stream server for object detection results and the output stream can be requested by Node-RED through the “GET” function..

3.3.6 Model Optimizer

A tool dubbed the Model Optimiser (MO) allows it to be simpler to transfer a deep learning model from a training environment to a deployment environment. It can transform a model that has been trained using a well-known framework, such as PyTorch, into an Intermediate Representation (IR) format, such as the ONNX format, and then into TensorRT format. Additionally, MO analyses static models in order to prepare them for effective execution on the intended hardware.

3.3.6.1 TorchScript

A PyTorch model can be serialised and saved in a format that can be loaded and operated in a different environment with TorchScript. The serialized TorchScript allow PyTorch models to be deployed in contexts without PyTorch. This is helpful for delivering models to the production environment or executing them on limited-resource devices like the Jetson Nano B01.

PyTorch models can perform far more efficiently when optimized using TorchScript, especially when used in production or on low-powered hardware. This is because TorchScript transforms PyTorch models into a highly optimised format for faster execution.

Thus, for our trained models, TorchScript can be used to optimize the models for deployment on the Jetson Nano B01, improving inference speed and reducing memory usage.

3.3.6.2 Open Neural Network Exchange

Microsoft and Facebook collaboratively created the open-source ONNX deep learning ecosystem, which acts as a framework for easy platform switching. Its key benefit is the prevention of framework lock-in by making hardware optimisation easy to acquire and allowing the exchange of deep learning models between other frameworks.

Popular deep learning frameworks including The Microsoft Cognitive Toolkit, Caffe2, MXNet, and PyTorch all natively accept ONNX models. It is also made simpler for developers to connect models across several platforms by the availability of converters for several machine learning frameworks, including TensorFlow, CoreML, Keras, TensorRT, and Scikit-learn. Using

ONNX, pre-trained models of the different frameworks can be compiled into a file, which can then be merged with required applications.

In this project, the ONNX framework acts as an intermediate framework for the trained PyTorch model. After converting to ONNX, the model will be converted into TensorRT optimized engine.

3.3.6.3 TensorRT

TensorRT is a high-performance deep learning inference optimizer and runtime library developed by NVIDIA for use on their GPUs, including the Jetson Nano B01. It is designed to optimize trained neural network models for deployment in production environments, allowing them to run with maximum efficiency on NVIDIA GPUs.

In Jetson Nano B01, TensorRT can be used to accelerate the execution of deep learning models on the device's GPU, allowing for faster and more efficient inference performance. This is particularly useful in applications such as object detection, image classification, and natural language processing, where the speed and accuracy of inference are critical. TensorRT achieves this optimization by fusing multiple layers of the network into a single operation, minimizing data movement and memory usage, and applying precision calibration to reduce computational overhead.

To use TensorRT with the models, the trained models need to be exported to the ONNX format and then optimized using the TensorRT API. This process involves configuring the TensorRT engine, parsing the ONNX model, and applying optimizations such as layer fusion, precision calibration, and dynamic tensor memory allocation. Once the TensorRT engine has been built, it can be loaded onto the Jetson Nano B01 and used for inference with the CNN model.

Thus, for the trained models, TensorRT can be used to optimize the models for deployment on the Jetson Nano B01, improving inference speed and reducing memory usage.

3.3.7 OpenCV

Open-Source Computer Vision (OpenCV) is a free and open-source computer vision and machine learning software library. It was originally developed by Intel and is now maintained by the OpenCV community.

OpenCV has support for multiple platforms and operating systems, as it can be used with various programming languages such as C++, Python, Java, and MATLAB on platforms such as Windows, Linux and MacOS, and it provides tools and libraries for image and video processing, object detection and tracking, machine learning, real-time processing capabilities, easy integration with other libraries and tools, and support for popular hardware platforms such as GPUs, Field Programmable Gate Arrays (FPGA)s, and embedded devices.

Object detection using CNNs often involves preprocessing and postprocessing steps to analyze images, extract features, and identify objects of interest. OpenCV provides a convenient and powerful set of tools for these tasks, including image loading and manipulation, colour space conversion, feature extraction, and object tracking. Additionally, OpenCV's integration with deep learning frameworks like TensorFlow and PyTorch allows for easy deployment of CNN models for object detection and recognition.

In this project, OpenCV is used for image preprocessing and post-processing during model inferencing. OpenCV functions that are used by this project included in Table 3.2.

Table 3.2: Functions of OpenCV used in this project.

Function	Uses
cv2.resize	Used to resize the input and output image size.
cv2.flip:	Flip image either vertically or horizontally. In this project image is flipped before inputting into model to generate 0° image and increase accuracy.
cv2.cvtColor	Convert image from 1 color model to another. As OpenCV convert RGB image to BGR image during processing, in order to convert back to RGB image, this function is used.

cv2.imencode	Used to encode the image into simpler data to be transmit though http stream
cv2.VideoCapture	To capture the input from IP Webcam

3.3.8 Roboflow

For an object detection model to train on image datasets, every image in the datasets must be annotated to generate a label. A label allows the object detection model to identify the class of a specific object or region. In this project, Roboflow will be used as an annotation tool. Roboflow is a platform that provides tools and services for managing, annotating, and transforming image data for machine learning applications, including object detection, segmentation, and classification. It's had support for all popular models and formats, from YOLO's family to Faster R-CNN and PASCAL VOC format. Other than that, data augmentation also be conducted on this platform as it provide a variety of data augmentation such as flipping, contrast, exposure and brightness variation.

3.3.9 ClearML

For data science teams around the world, ClearML is an open-source platform created to streamline and automate the creation and maintenance of machine learning solutions. It allows user to manage their datasets, models and workflow directly on their local server. By integrating ClearML into the GC's training notebook, we will be using it to track the accuracy and runtime of the trained models for each run in this project.

3.4 Data Preparation

3.4.1 Datasets Used

There are many fire detection-related datasets from various studies. However, the dataset contains some outdoor wildfires, which do not suit small fire detection requirements. Thus, a custom dataset is proposed in this study.

In order to train a model that could detect small fires in both outdoor & indoor scenarios, we need to have a large number of images with outdoor & indoor scenarios. Fire and Smoke BBox COCO Dateset, BoW Datasets and Robmarkcole datasets have been chosen as they have contained a large number of indoor fire images. However, low-resolution images and forest fire images must be filtered out to ensure accurate results on indoor fires.

Similar to Pincott et al. (2022), this research uses training images from Google and Flickr that include locations of potential inaccuracy, such as reflections in screens and light fixtures. By exposing the model to demanding data during training, the inclusion of these regions aims to increase the robustness of the model. To lessen the frequency of false positives, non-fire photos of indoor environments from the MIT indoor collection are also included.

To ensure high mAP is achieved in the test environment, a few noise images that are captured in the testing environment have been added to the training and valid sets. It is not added to the test set as it will make the testing unbiased. Currently, the total of images is shown in Table 3.3.

Table 3.3: Allocation of Fire and Non-Fire images in the custom dataset

	Train	Valid	Test
Fire images	546	150	85
Non-Fire images	364	115	44
Total	910	265	129

Unfortunately, the training set have 910 images, which is below the recommended number of 1000 training images by Jocher (n.d). Thus, data augmentation is used to increase the number of training images.

3.4.2 Data Augmentation

In considering the fact that custom datasets contain a finite number of images, Li et al., (2022), Norkobil Saydirasulovich et al., (2022), and Jin et al., (2023) advocate using data augmentation to enhance the dataset's training image count.

The training images were horizontally flipped with a probability of 0.5 during the data augmentation stage. Additionally, the images' exposure and brightness were changed by up to 25% in order to increase the model's resistance to changes in camera settings and lighting. In order to improve the model's durability, the saturation of the photos was changed by up to 40%.

The images were then given a random Gaussian blur effect of up to 1px to assist the model in better adapting to changes in camera focus. The resulting dataset is in Table 3.4.

Table 3.4: Allocation of Images after performing data augmentation on custom dataset.

	Train	Valid	Test
No. of Images	2700	265	129
Ratio (%)	87	8	4

Thus, the custom dataset has fulfilled the requirement for training and is ready for training in GC.

3.5 Improvement On YOLOv5s

3.5.1 Overview

YOLOv5 proposed by Jocher (n.d) unlike its predecessors like YOLOv4 and YOLOv3, uses a PyTorch framework instead of the Darknet framework. YOLOv5 has four different model sizes, which are YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x, YOLOv5s is the smallest with the fastest inference speed but lowest accuracy, while YOLOv5x is the largest with slowest inference speed but most accurate. In this project, YOLOv5n is used due to the low computational power of edge devices.

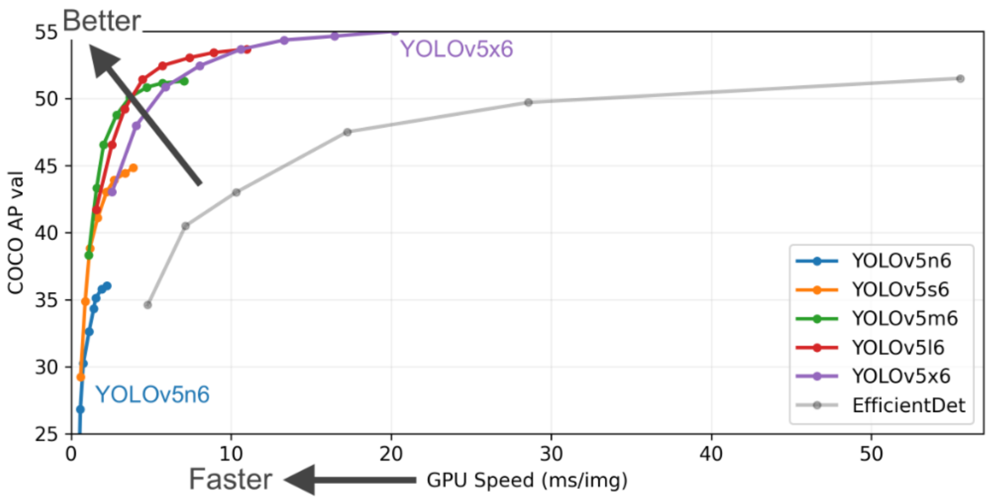


Figure 3.7:Comparison of speed and accuracy on COCO datasets (Jocher, n.d)

3.5.2 Addition of Bidirectional Feature Pyramid Network Concat

In the proposed model, a 1 BiFPN block is integrated into the YOLOv5s model at the P4 layer. In the classic YOLOv5s model, PANet is used instead of BiFPN, which is a variant of FPN as shown in Figure 3.8.

For identifying small targets like fire, the addition of a BiFPN block enhances the model's accuracy by more effectively integrating data from various scales and levels of abstraction. Even though the BiFPN block adds more calculations and parameters, its effect on the model's overall computational needs is minimal, especially when contrasted with growing the number of layers or the size of the feature maps.

By incorporating a single block of BiFPN into the PANet of YOLOv5, the model's performance on tiny item recognition tasks can be improved without noticeably boosting the computing requirements.

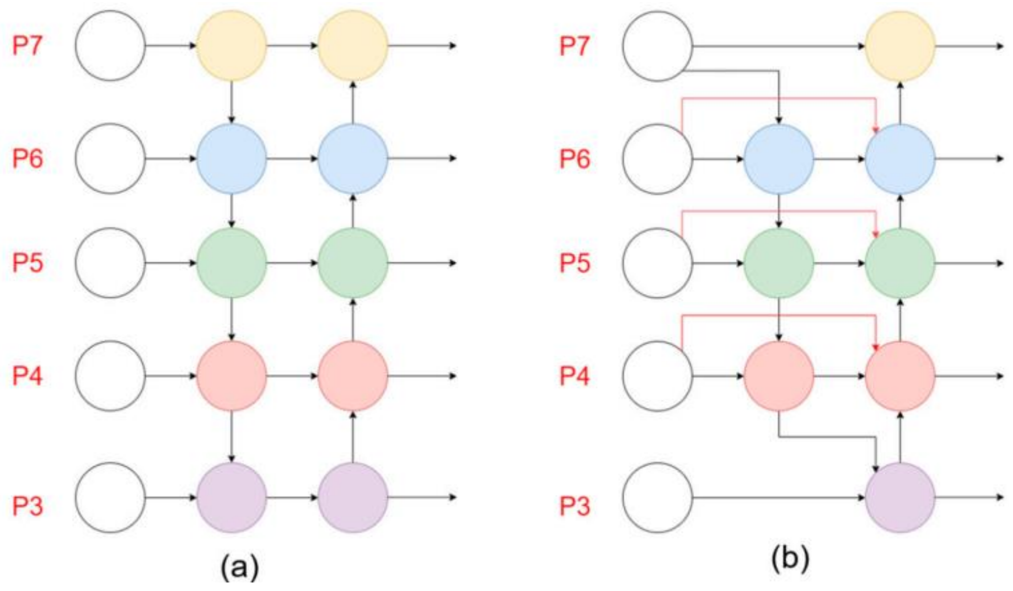


Figure 3.8: The structure of (a) PANet & (b) BiFPN (Xue et al., 2022)

3.5.3 Addition of prediction layer

In the YOLOv5 PyTorch Hub repo, it has implemented an addition of a neck layer in YOLOv5p6.yaml to extract data from a larger image. However, in this project, we are required to detect small targets of fire.

Due to the smaller size of these targets and the greater down-sampling factor utilised in the model, YOLOv5 may not detect little flames efficiently in the context of fire detection. The feature maps' resolution is decreased throughout the downsampling process, making it difficult for the model to pick up on the particular characteristics of little flames. To solve this problem, an extra prediction layer can be added to shallower feature maps with better resolution in order to find these targets according to the method of Yang et al. (2023). The model is able to better capture the important properties of little flames for precise detection because of the additional layer, which lowers the downsampling ratio (Yang et al., 2023).

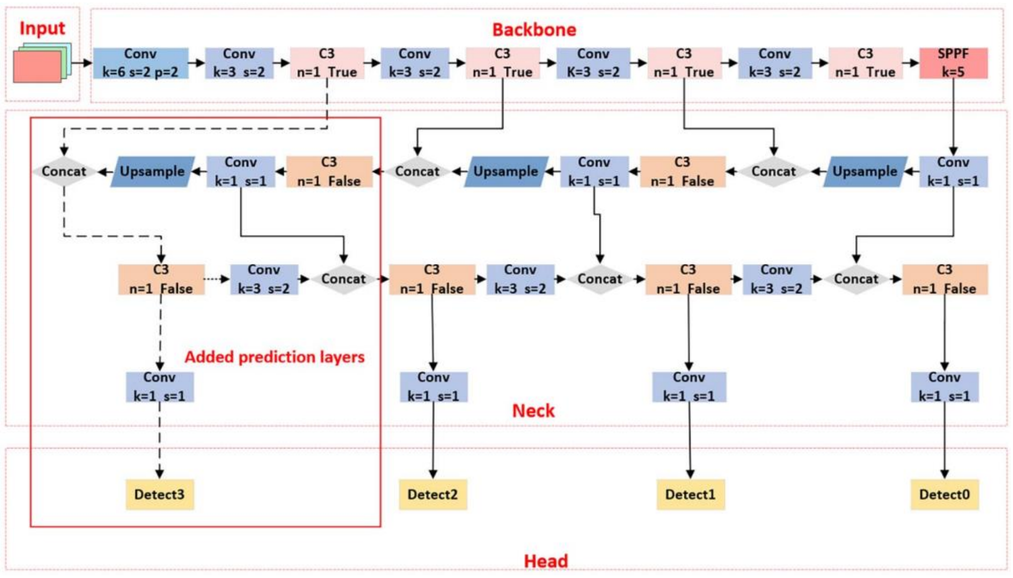


Figure 3.9: Additional prediction layer in the highlighted red box on top of original YOLOv5s model (Yang et al., 2023)

The model YAML script of the improved model is in appendix, and it is tested for YOLOv5s model. Application on other YOLOv5 model such as YOLOv5x or YOLOv5m required manual tweaking. This also applies to newer version of YOLOv5, currently tested YOLOv5 version is YOLOV5 p5 model.

3.6 Training the model

3.6.1 Batch Selection

YOLOv5 and YOLOv7 have a considerably large size. The largest batch size that can be utilised for training deep learning models depends on how much memory is allotted to the model. The workstation used for this project's model training has enough CPU and GPU RAM to support training batches larger than 32. However, YOLOv7 are unable to train at a batch size of 32 due to limited CUDA memory, while a batch size of eight is not optimal and will result in a slow training rate, as most training of object detection models usually requires at least a batch size of 32. Thus, a batch size of sixteen is chosen for YOLOv5 and YOLOv7 models.

Table 3.5: Type of setting during training for both YOLOv5s and YOLOv8s

Type of setting	
Batch	16
Epochs	50
Training Image Size	640x640
Learning Rate	0.01
Intersection over Union (IoU)	0.4
Conf threshold	0.2
Optimizer	SGD

3.6.2 Transfer Learning Approach

Transfer learning involves taking a pre-trained model on a large dataset and using it as a starting point to train a new model on a smaller, more specific dataset. YOLOv5 and YOLOv7 are pre-trained on a COCO dataset, which contains thousands of images of various objects. The transfer learning with the pre-trained model is employed in this project for improving performance due to the marginal number of images in the custom fire dataset.

There are 3 strategies to transfer the learning approach. The first strategy trained the entire model which was applied the most. Strategy 2 freeze the backbone layer and train the head. Strategy 3 freezes all the convolutional layers and uses them as the image feature extractor.

Since the COCO dataset does not have a fire object class, strategy 1 is used, which uses the pre-trained model as a feature extractor and trains a new classifier on top of the extracted features. Taking this strategy, the pre-trained model's ability to extract useful features is exploited while training a new classifier that is specific to the custom fire dataset. As the transfer learning method is integrated with YOLOv5 and YOLOv7 to train the model, it can directly be used in GC to perform the transfer learning.

3.7 Conversion of the Trained Model into Intermediate Representation to load in Inference Engine

In this project, the MO is used to (i) optimize the PyTorch model into the respective format, (ii) Reduce the model's floating-point precision from single precision (FP32) to half precision (FP16), (iii) add Non-Max Suppression (NMS) technique into the model and (iv) reduce IoU and Confidence threshold.

Intersection over Union (IoU) is a metric that compares two sets of data, such as the predicted bounding box and the actual ground truth bounding box of an object in an image, to determine the similarities between both data. Higher IoU indicates a better match between the predicted and ground truth bounding boxes.

The confidence threshold is a figure that is used to exclude predictions with low confidence scores. A confidence score is the probability that an object is, in fact, present in the projected bounding box in object detection tasks. The confidence threshold is used to manage the trade-off between recall and precision, whereby lowering the threshold results in higher recall but poorer precision, and vice versa.

The intial model modifications reduces the model size by a factor of two and speeds up inference.. Whereas the second is a technique used mainly in object detection that aims at selecting the best bounding box out of a set of overlapping boxes. By using this technique, the overlapping of bounding boxes on the same object can be eliminated. Lastly, the latter model adjustment reduce

The output files from the fire detection model's conversion into IR are then imported into the inference engine (IE), in this case, TensorRT. TensorRT is specifically made to optimize and accelerate deep learning inference on a particular hardware platform, in contrast to MO, which optimizes the model based on its complexity to improve memory and compute times. It optimizes and accelerates inference for CNN models on specific Nvidia GPUs. The supported devices for TensorRT includes all Nvidia hardware, including all Jetson device line and Workstation. Furthermore, TensorRT supports Nvidia CUDA which comes on board Jetsons Device and Workstation, thus allowing the use of CUDA libraries to accelerate deep learning inference, the use of high memory bandwidth that can significantly improve the speed of data transfer between the CPU and GPU and lastly using the massive parallel computing platform provided by CUDA to accelerate computations by utilizing the thousands of processing cores available on the Nvidia GPU.

In this project, TensorRT will optimize the YOLOv5 and YOLOv7 models for Nvidia GPU on Jetson Nano B01. The optimization must be made on the edge device for the optimization to be compatible with the hardware specification. The resulting file will be the TensorRT engine for FP32 and FP16.

3.8 System Architecture

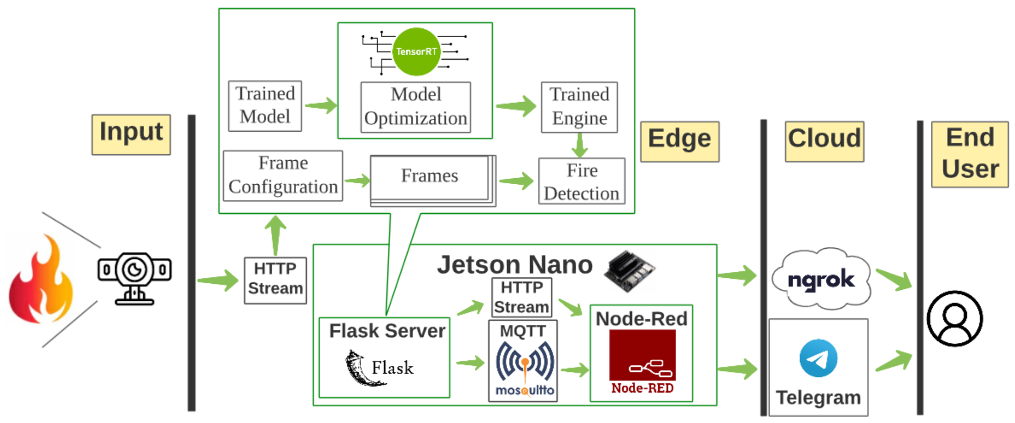


Figure 3.10: System Architecture

The proposed system has three unique layers, including four main components of the sensing, processing, and visualising units. The IP camera that is responsible for collecting video inputs represents the sensing unit of the system. The sensing unit is situated in the input layer, where the source is located.

To achieve minimum latency, the edge device, also the processing unit is situated at the edge layer, where it is close to the source but still able to access the cloud. As the processing unit, Jetson Nano B01 is responsible for controlling the operation of the whole system. In the Jetson Nano B01, Flask Application, Mosquitto MQTT and Node-RED are installed and deployed. The video footage from the IP camera is sent to the Flask server via HTTP to be separated into frames and waiting to be processed by YOLOv5. The trained YOLOv5 model from GC is imported into Jetson Nano B01 and optimized with TensorRT to generate TensorRT Engine. By serializing the engine, the frames are processed, and bounding boxes are generated for each frame. In the Flask application, the processed data is then encoded and streamed via HTTP through the Flask server.

Lastly, the visualising unit is Node-RED. With Node-RED, a dashboard UI is deployed that can be accessed through both browser and phone. In the dashboard UI, the end user can observe the fire outbreak footage. In order to protect the privacy of end users, before sending out the HTTP address of the dashboard UI, the address goes through encryption in the Ngrok cloud server.

3.9 Experimental setup

The experiment uses test split from the custom dataset mentioned in section 3.4. In this experiment, the improved YOLOv5s, the original YOLOv5s, and the YOLOv8s versions of YOLO are employed for fire detection. The V380 IP camera was chosen because it can record crystal-clear, HD footage at a distance of 107 cm (42 inches), even in low light. The camera is positioned to see the entire area of interest where fires may develop and are mounted 200 cm above the base level. The experimental setup is shown in Figure 3.11 and 3.12.

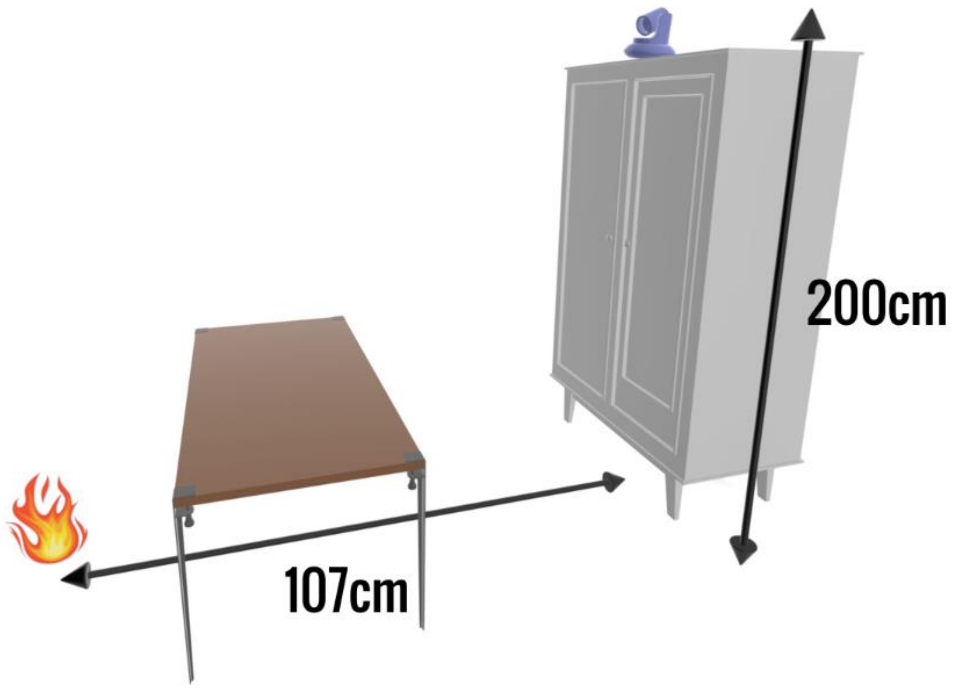


Figure 3.11: Experimental Setup (Isometic view)

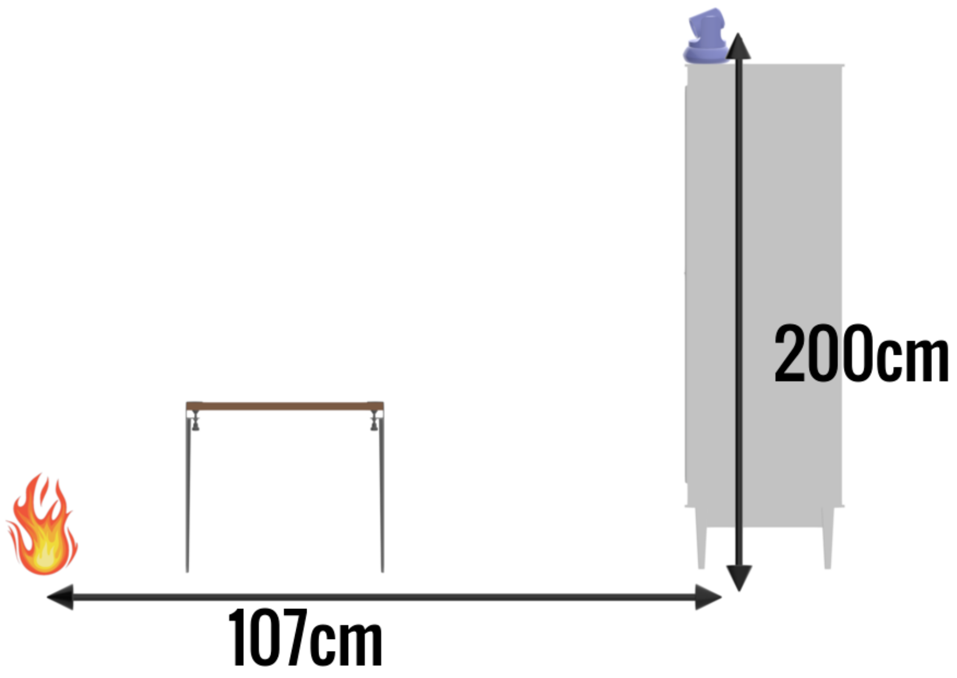


Figure 3.12: Experimental Setup (side view)

Starting with the fire's origin and continuing through its growth and extinguishment, the V380 IP camera records a continuous video stream. About five minutes should be allotted for the recording. To evaluate the system's capability to identify flames from hazy images or small fire targets, a second test video is also captured.

In the testing site, a table is placed in front of the camera to partially impede the view of the fire, and the second test video is purposefully blurred to imitate occlusion and other noise and disruption in the video stream. These circumstances are intended to test the fire detection system's robustness in real-world situations when environmental influences could impair fire detection.

The lights are turned on and off during the recording to represent nighttime and daytime situations, respectively, to replicate changes in lighting conditions. To test the fire detection system's performance in various lighting environments, the time and length of the illumination changes are changed. There is enough oxygen flowing through the area where the experiment is being conducted, and all combustible objects have been removed from the area.

Finally, all three versions of YOLO are used to analyse the video streams in order to spot any potential fires and take the necessary precautions to put them out. The testing data are utilised to assess the precision and effectiveness of the many YOLO iterations and to improve the fire detection system.

3.10 Workflow

The workflow of this project is separated into 3 sections, one on GC, one on the workstation and another on Jetson Nano B01. This step is taken to reduce the workload of Jetson Nano B01 by pushing most computational power needed for the training model onto the cloud and workstation. Moreover, training of the model does not require real-time 24-hour streaming of data, thus GC is fully capable of carrying out this task.

At the start of the first section, the environment will be set up in GC to train the models. Although google collab are used , repositories and libraries must be imported. One of the crucial libraries is PyTorch as both YOLOv5s & YOLOv8s are based on the PyTorch framework. PyTorch can be directly imported into GC because GC is GPU enabled and uses a Python environment. Another item to be set up is the repository of YOLOv5 and YOLOv8, which can be cloned directly from GitHub onto GC (Jocher, G., n.d). After cloning, a custom YAML file is produced to generate an Improved YOLOv5s model. After setting up the environment in google collab, we will compile a custom fire dataset from Fire and Smoke BBox COCO, BoW and Robmarkcole datasets, while filtering out low-resolution images as well as smoke and forest fire images. To increase the detection rate in the testing environment, noise image from the testing ground is added to the datasets. In RoboFlow, the images in the dataset are then annotated with bounding boxes for each fire object and marked null images for non-fire images. To increase the number of training images, data augmentation mentioned in 3.4.2 is used. After data augmentation, the dataset is uploaded into GC and starts training YOLOv5s and YOLOv8s models according to the parameter in section 3.6.1 as well as Improved YOLOv5s with custom.YAML file. The result can then be displayed on ClearML for comparison. If the model is not satisfying, the data augmentation step will be repeated until a satisfying model is determined. The trained model weight is downloaded directly from ClearML into the workstation.

Now entering section 2, a Python script containing the MQTT client and Flask framework setup is generated with Pycharm. With the script, the usability of the Flask server HTTP stream is tested. If the HTTP stream can be accessed locally on the browser, then the Python script as well as the trained model will be imported into Jetson Nano B01.

In section 3, Jetson Nano B01 is installed with JetPack for ease of use. After setting up OS, the PyTorch framework and OpenCV library required for running the Python script as well as Node.js required for running Node-RED are installed. With these dependencies installed, YOLOv8 and YOLOv5 repository is cloned from Github. Following that, Mosquitto MQTT and Node-RED are installed and set up. IP Camera is also set up and connected to WiFi. Using the Node-RED, a dashboard UI is developed that is capable to display the HTTP stream from the Flask server and also allows the end user to start the system as well as send an alert to the end user when a fire is detected. With the UI completed, we will start optimizing the trained model with ONNX then TensorRT as well as TorchScript to generate the engine file and TorchScript file. By installing the MQTT library and node in Jetson Nano and Node-Red respectively, a client and broker relationship is set up between the Python script and Node-RED.

Finally, the experiment will start here. The custom dataset is first imported into Jetson Nano B01. Using benchmarks.py provided by YOLOv5 and YOLOv8 repositories, results can be obtained and saved for each format of the model.

For real-life deployment, the user can start the system by typing "/flask" in the firefighterbot. If no fire is detected, there will be no changes. However, if a fire is detected by the Improved YOLOv5s model, an alert will be sent to Node-RED via Mosquitto MQTT. The alert is then forwarded to the end users through Telegram. Users can choose to access the UI with the video stream by typing "/ngrok". A WIFI module is installed on the Jetson Nano B01 to bridge the connection between IP Camera, User's phone and Jetson Nano B01.

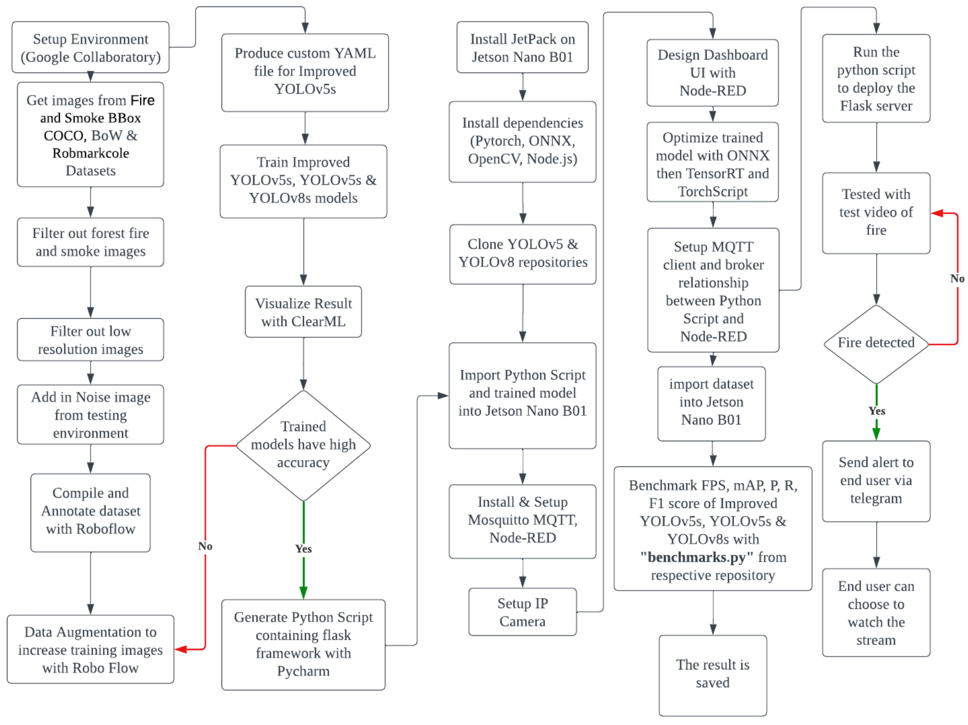

Setup Environment (Google Collaboratory)
Get images from Fire and Smoke BBox COCO, BoW & Robmarkcole Datasets
Filter out forest fire and smoke images
Filter out low resolution images
Add in Noise image from testing environment
Compile and Annotate dataset with Roboflow
Data Augmentation to increase training images with Robo Flow
Produce custom YAML file for Improved YOLOv5s
Train Improved YOLOv5s, YOLOv5s & YOLOv8s models
Visualize Result with ClearML
Trained models have high accuracy
No
Yes
Generate Python Script containing flask framework with Pycharm
Install JetPack on Jetson Nano B01
Install dependencies (Pytorch, ONNX, OpenCV, Node.js)
Clone YOLOv5 & YOLOv8 repositories
Import Python Script and trained model into Jetson Nano B01
Install & Setup Mosquitto MQTT, Node-RED
Setup IP Camera
Design Dashboard UI with Node-RED
Optimize trained model with ONNX then TensorRT and TorchScript
Setup MQTT client and broker relationship between Python Script and Node-RED
import dataset into Jetson Nano B01
Benchmark FPS, mAP, P, R, F1 score of Improved YOLOv5s, YOLOv5s & YOLOv8s with "benchmarks.py" from respective repository
The result is saved
Run the python script to deploy the Flask server
Tested with test video of fire
Fire detected
No
Yes
Send alert to end user via telegram
End user can choose to watch the stream

Figure 3.13: Flowchart of workflow

3.11 Summary

In this section, the project activities planned are shown and the tools used in this project are defined. Moreover, the system architecture is proposed, and the methodology is generated. The Improved YOLOv5s for small target fire detections is designed based on PyTorch Framework. The custom dataset is generated and used to train the models. The GC, Workstation and Jetson Nano B01 are used as platforms for the project. Google Collab, is used to train the Improved YOLOv5s, YOLOv5s and YOLOv8s models. While for Jetson Nano B01, the system is configured to be the testing platform for the project, whereas the workstation is used for testing the Python script. Lastly, TensorRT and TorchScript are used to optimize the trained model and deploy it on Jetson Nano B01.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

In this chapter, the fire detection systems have been taken for testing to determine whether the method proposed is usable and results are obtainable. The testing will be using the same custom datasets used in training but will be using the test split that has unbiased data. The testing will be done on YOLOv5s, YOLOv8s and Proposed YOLOv5s models. Finally, the performance of the Proposed YOLOv5s, YOLOv5, and YOLOv8 models on the Jetson Nano B01 will be tested and discussed. As there are no publicly available detection datasets for fire benchmarking, a comparison will solely be made with other studies on their performance on respective datasets.

4.2 Evaluation Metric

Since the fire detection models were trained using an exclusive dataset, there is no benchmark against which to measure how effectively they performed.

In this project, the performance of the proposed system instead is evaluated using mAP, confusion matrix, precision (P), recall (R), FPS, F1 score and Latency.

The level of precision indicates how accurate the model's forecasts are, whereas Recall is a metric for how well a model can identify all occurrences of positive data. Equation 4.1 and Equation 4.2 show the calculation of precision and recall respectively. False Positive (FP) refers to the proportion of occurrences where the model misclassifies a situation as positive. The quantity of cases that the model properly classifies as positive is known as True Positive (TP). The number of instances that the model misclassifies as negative is known as False Negative (FN). The number of cases that the model correctly classifies as negative is known as True Negative (TN).

$$Precision = \frac{TP}{TP+FP} \quad (4.1)$$

$$Recall = \frac{TP}{TP+FN} \quad (4.2)$$

Where:

TP = True Positive

FP = False Positive

FN = False Negative

An object detection model's overall accuracy across all classes is measured by mAP. It is calculated by averaging the average precision (AP) values of all classes in a dataset. On the other hand, AP is a measure of the accuracy of the model for a specific class. Based on Precision and Recall, the AP can be expressed as the integral of function Precision of Recall, as shown in Equation 4.3. However, since this project only include 1 class, both mAP and AP are the same.

$$mAP(single\ class) = AP \ = \int_{1}^{0} Precision(Recall)d(Recall) \quad (4.3)$$

There are 2 variations of mAP, which are mAP50 and mAP50-95. The mAP calculated at an IoU threshold of 0.5 is known as mAP50, whereas the mAP calculated over the 0.5 to 0.95 IoU threshold range with a 0.05 step size is known as mAP50-95. When objects are relatively small or have little overlap with the ground truth boxes, the mAP50 can be used to assess how effectively the model is functioning. mAP50-95 on the other hand is a metric that takes consideration of the model's performance across various IoU thresholds and shows an overall performance. In this project, mAP50 is given the primary focus compared to mAP50-95 as we are focusing on detecting small target fires.

The F1 score calculates the balance between precision and recall and evaluates the performance of the model. If the F1 score is high, precision and recall are high, and vice versa. The equation is shown in Equation 4.4.

$$F1\ score = \frac{2\times(Precision\times Recall)}{Precision+Recall} \qquad (4.4)$$

Only a model with a higher FPS than 10 FPS is required to calculate the F1 score as the F1 score does not consider FPS in its metric.

The performance evaluation was conducted using 2 test videos featuring flames, as well as images separated from the datasets that feature both fire and non-fire images. According to Ryu & Kwak (2022), when detecting using optical flow, the performance of the model should be evaluated through a video, or in our case a live stream, the image should never be static. Thus, following their example, in this project, we also included 30 frames from the test video that we will be inferencing to calculate the result. We also are evaluating the FPS of both models. FPS indicate the time require for a model to process and analyse a video frame in real-time. As a fire detection system needs to be able to execute timely and accurate detections of fire in real-time, by having a higher FPS can allow faster detection of fire and minimize the damages caused by fire.

.

Table 4.1: Comparison of Improved YOLOv5s, YOLOv5s and YOLOv8s on Jetson Nano B01

	Framework	PyT	TrS	TRT (FP32)	TRT (FP16)		PyT	TrS	TRT (FP32)	TRT (FP16)		PyTorch	TrS	TRT (FP32)	TRT (FP16)
	FPS	4.7	4.794	6.58	10.6		6.01	6.22	9.04	**13.5**		5.409	5.598	8.654	12.3
	mAP (50)	0.514	0.515	0.515	0.508		0.571	**0.572**	**0.572**	0.564		0.454	0.452	0.452	0.452
	Precision (P)	0.542	0.541	0.541	0.541		0.667	0.666	0.666	0.666		**0.926**	**0.926**	**0.926**	**0.926**
Improved v5s	Recall (R)	**0.5**	0.502	0.502	0.5	v5s	0.4	0.386	0.386	0.385	v8s	0.29	0.29	0.29	0.29
	Model Size (MB)	15.5	30.9	35	17.1		**14.1**	28.1	31.8	15.6		22	43.8	49.6	24.6
	F1 Curve				**0.52**					0.479					0.444
	Increase in FPS compared to PyTorch	0	0.02	0.4	1.25		**0**	0.03	0.5	1.26		0	0.03	0.6	**1.28**

Where:

PyT is PyTorch

TRT is TensorRT engine.

TrS is TorchScript

From the Table 4.1, it can be seen that **YOLOv5s show higher overall performance then YOLOv8s** and even **Improved YOLOv5s** when trained on identical custom dataset and also during testing. However, **Improved YOLOv5s** and **YOLOv8s** shown **highest Recall rate and Precision respectively**.

Another notable observation in Table 4.1 is the difference between the FPS of optimized model and their PyTorch counterparts. In Table 4.1, the models are optimized using TorchScript, ONNX and TensorRT mentioned in 3.3.6 are compared using the performance before and after optimization. Overall, TorchScript optimization show average of 2.7% increase in FPS, while TensorRT at FP32 and FP16 show average increase of 50 % and 126% respectively. YOLOv8s show the largest increase in FPS when optimized with TensorRT, followed by YOLOv5s and lastly Improved YOLOv5s. The improved YOLOv5s show lower FPS because of added complexity in the model.

With a recall of 0.5, the lowest FPS of 10.6 and a precision of 0.541, improved YOLOv5s has a marginal FPS, a good recall, but a lower accuracy, which implies it might generate more false alarms. Due to its average recall, it only missed half of the fires.

While having a mediocre FPS of 12.15, YOLOv8 has the highest precision of 0.926, which means it generates very few false alarms. Of the three models, it has the lowest recall of 0.29, which means it missed 71% of fires.

As opposed to Improved YOLOv5s, YOLOv5s has a greater precision of 0.666, a recall of 0.385, and the highest FPS of 13.5. This indicates that it has a greater fire detection range than YOLOv8 and generates fewer false alarms than Improved YOLOv5s. Due to its lower recall compared to Improved YOLOv5s, it still failed to detect 61% of fires.

In the context of fire detection, both recall and precision are more important than mAP, as high recall rate ensure that all instances of fire are detected, while precision minimize the instances of false alarm and ensure that only actual fires are detected. Still, the cost of missed fire detection is larger than the cost of false alarms in fire detection as false alarm only resulted in more notification, while missed fire detections mean damage of property in a indoor fire detection. Thus, recall is taken priority over precision.

F1 score for the 3 models, are 0.520, 0.479 and 0.444 for Improved YOLOv5s, YOLOv5s and YOLOv8s respectively. By focusing just on precision and recall in this test, Improved YOLOv5s have the highest F1 score and with a difference of 8.55% and 17% difference from YOLOv5s and YOLOv8s.

Nevertheless, FPS is also a vital parameter when evaluating a fire detection on edge Devices. In Fire detection on edge device such as Jetson Nano B01, FPS will affect the latency of the system and latency further affect the responsiveness towards fire outbreak. Having higher FPS means faster respond from user. Still, having an FPS higher than 10 is sufficient for household indoor used as this project is targeting indoor fire detection in residential area instead of industrial area where responsiveness is taken priority then fire capture rate.

After taking consideration of all the evaluation criteria, among the 3 models, Improved YOLOv5s is the best option, which has an average FPS of 10.5 while having lower precision of 0.541 but higher recall of 0.5. This indicates that it has a marginal level of accuracy and detection speed but have a high capture rate of fire which is crucial for fire detection.

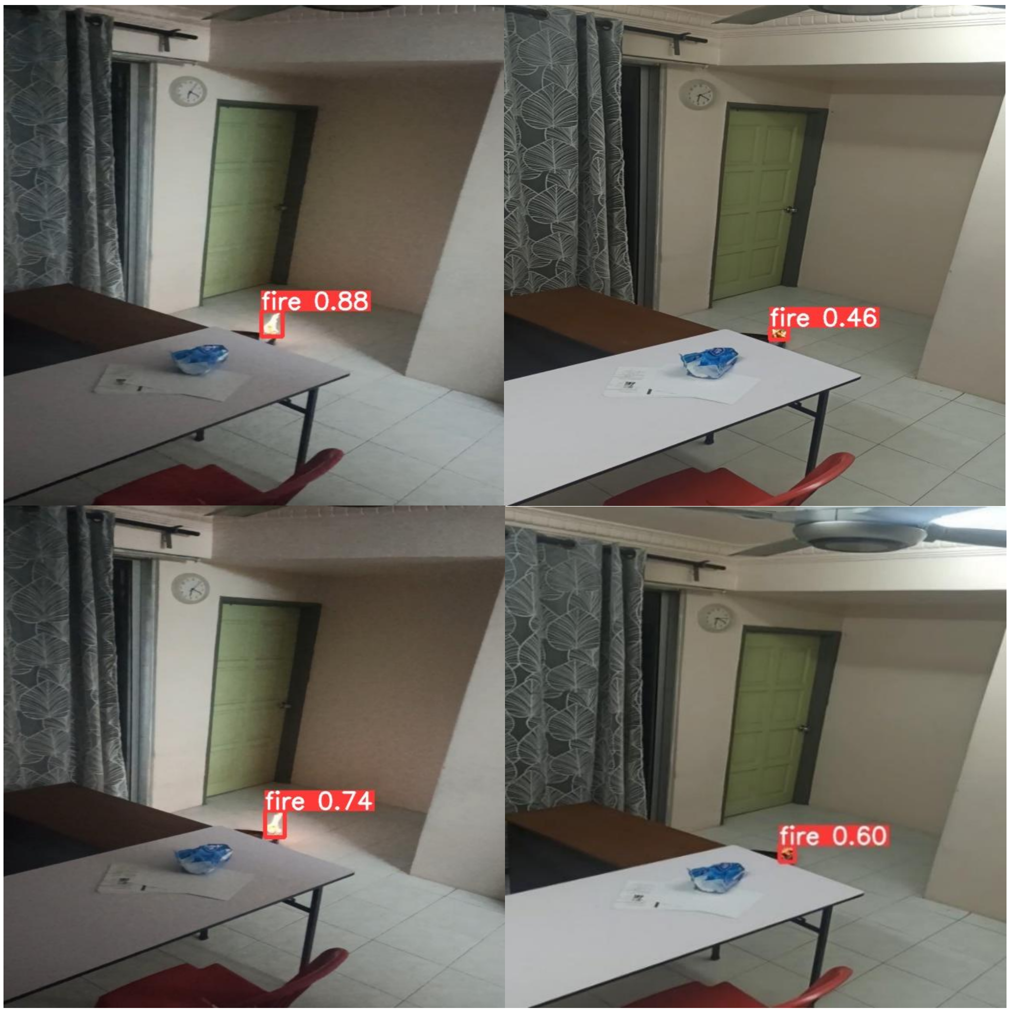
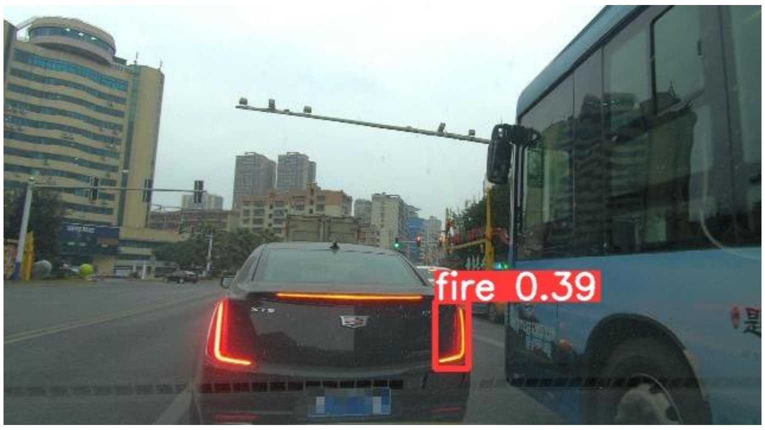


Figure 4.1: TP in both dark and light environment in YOLOv5s (Top) Improved YOLOv5s(bottom)



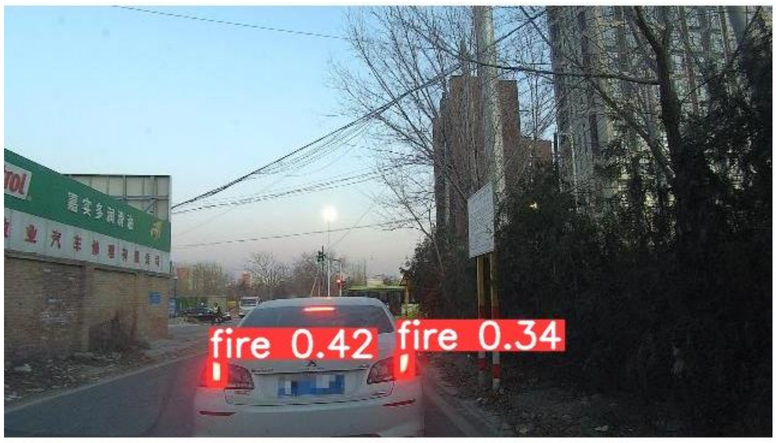


Figure 4.2: FP on car's back light in both YOLOv5s (Top) and Improved YOLOv5s (Bottom)

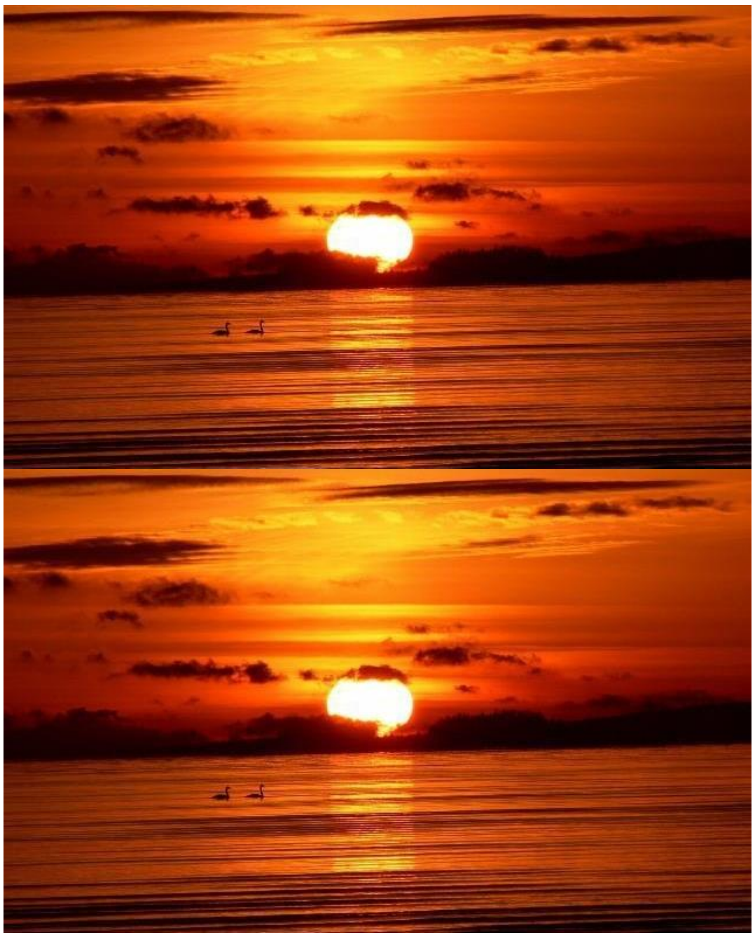


Figure 4.3: TN on sunset in both YOLOv5s (Top) and Improved YOLOv5s (Bottom)

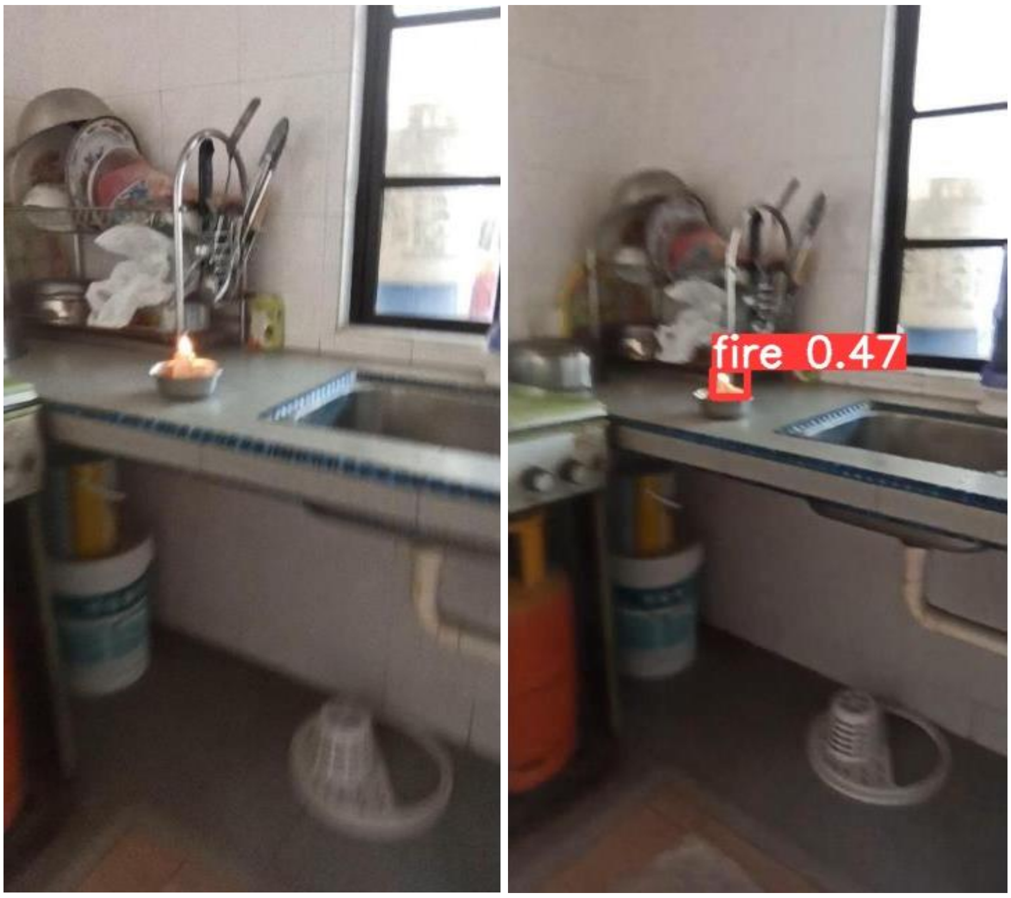


Figure 4.4: YOLOv5s(Left) compared to Improved YOLOv5s (Right)

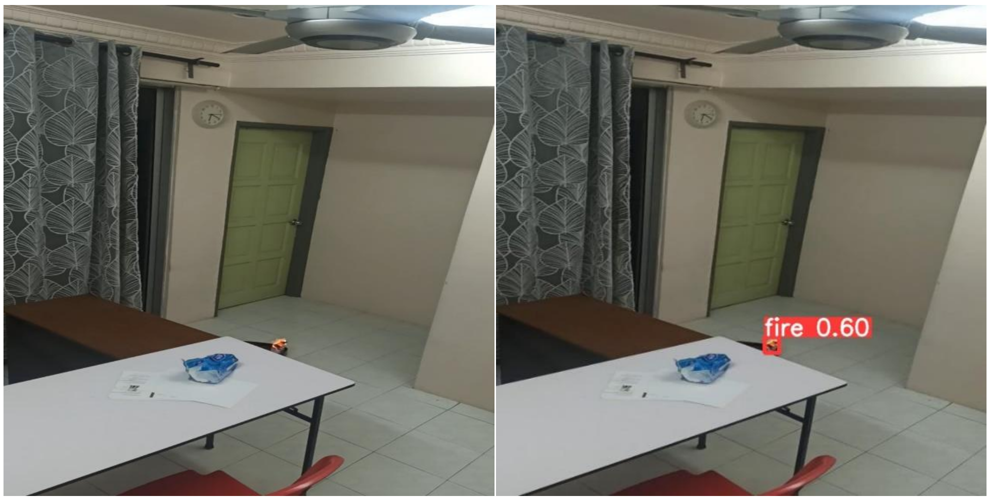


Figure 4.5: YOLOv5s(Left) compared to Improved YOLOv5s (Right)

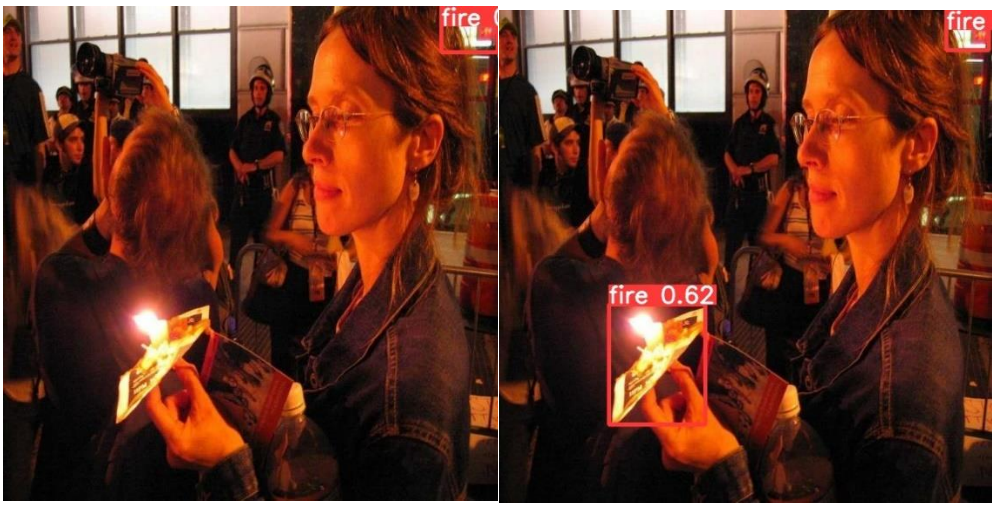


Figure 4.6: YOLOv5s(Left) compared to Improved YOLOv5s(Right)

Based on Figure 4.1 to 4.6, we can conclude that both Improved YOLOv5s and YOLOv5s can detect fire in light and non-light condition making them viable in daytime and nighttime for fire detection as seen in Figure 4.1, both models also did not show false positive on sunset in Figure 4.3. Sunset is one the image with high possibility of FP added according to Pincott, J. et al. (2022), Ahn et al. (2023) and Jain & Srivastava (2022)', which is easily mistaken as fire. However, Improved YOLOv5s show better robustness as it can detect fire in blurry images and also small flame of fire better than YOLOv5s as seen in Figure 4.4, 4.5 and 4.6. Thus, showing the robustness of Improved YOLOv5s. However, both models show FP on the car backlight in Figure 4.2, which means that, the models still lack training on background images.

Latency in this project refers to the amount of time it takes for IP camera to send input, process input by fire detection model and send the result to end user. The testing enevironemtn with parameters as shown in Table 4.2:

Table 4.2: Testing environment for Latency

Telegram latency (s)	Ip Camera' s Output	Forward time (s)	Decode time(s)	Inference FPS	Output Image	WiFi Download/Upload speed (mbps)	MQTT send time (s)
0.15	1080p	0.094	0.047	10.6	640 x 640	34 /27	0.001

Input image size: 1600x1200 (1080p)

Input size = 1600 x 1200 x 3 (RGB image)

Output size = 640 x 640 x 3 (RGB image)

Latency = (Input size / Upload speed) + (Output size / Download speed) + Forward time + Decode time + MQTT send time + Telegram send time Latency = (1600 x 1200 x 3) / (27 x 10^6) + (640 x 640 x 3) / (34 x 10^6) + 0.094 + 0.047 + 0.001 + 0.150

Latency = 0.232 + 0.030 + 0.094 + 0.047 + 0.001 + 0.150

Latency = 0.554 s or 554ms

Therefore, the total latency is approximately 554ms.

As currently there are no benchmark for fire detection. We will be comparing purely with results generated by various researchers in their studies.

Table 4.3: Comparison with other studies on mAP, P, R, F1 Score , FPS and Latency

Reference	mAP/AP (50)	P	R	F1	FPS	Latency (ms)	Detector	Edge Device	Remark
Maltezos, E, et al. (2022)	-	-	-	-	-	23	Sensors	Raspberry Pi 4	Using Edge Computing
Zhao, et al. (2022)	80.23	91.5	59.62	**73**	-	-	Fire-YOLO	PC	
Norkobil Saydirasulovich et al. (2023)	39.5	93.48	28.29	-	-	-	YOLOv6	PC	
Li et al. (2023)	95.4	-	-	-	14.28	-	EfficientNet	Jetson Nano	Using Edge Computing
	96.2	-	-	-	14.28	**-**	ShuffleNet		
	94.8	-	-	-	14.28	-	RepVGG		
	94.1	-	-	-	14.28	**-**	CSPNet		
Mukhiddinov et al. (2022)	72.6	-	-	-	-	1320	Improved YOLOv4	PC	Using Cloud Computing
An et al. (2022)	**96.3**	**95.2**	**99**	48.5	**55.56**	-	Improved & Pruned YOLOv5	PC	
Ahn et al. (2023)	96	91	93	-	-	-	YOLOv5s	PC	

Proposed system	50.8	54.1	50	53	10.6	554	Improved YOLOv5s	Jetson Nano	Using Edge Computing

Based on the comparison above in Table 4.2, the proposed system achieved mediocre results than other methods. An et al. (2022) achieved the highest recall rate of 99% and precision of 95.2%. The high recall rate and precision of An et al., (2022) is due to the introduction of dynamic convolution into YOLOv5s model. To produce adaptive dynamic convolution, attention dynamically modifies the weight of each convolution kernel in accordance with the input, allowing the model to capture small object in an image easily (An et al., 2022).

Although unable to achieve better results than the highest in each aspect, the Improved YOLOv5s still perform better than YOLOv6 in small target fire detection, achieving better mAP and recall rate. The reason is because an additional prediction layer allow prediction on higher resolution of small target.

The system proposed by Maltezos, E, et al. (2022) saw the lowest latency of 23ms, which is the proposed system is able to achieve the lower latency of 554ms than the system proposed in the study by Mukhiddinov et al. (2022) which has a latency of 1320ms. The reason for the proposed system to achieve lower latency is because Mukhiddinov et al. (2022) is using a cloud computing network where the data is sent to an AI server for computation instead of using an edge device. Thus, we can see the advantages of edge computing when performing fire detections.

4.3 Summary

In conclusion, regardless of the training approach applied, the suggested system architecture is equivalent to existing benchmarked methods for detecting small fires. The capacity of the model to identify small fire targets has been successfully enhanced by the addition of BiFPN and an additional prediction layer. These two improvements outperformed YOLOv5s trained only on a custom dataset, while inference speed can still be improved. Furthermore, it was demonstrated that the suggested system outperformed a cloud-based system in terms of latency, demonstrating the value of edge computing for fire detection. The Jetson Nano B01's on-board GPU was used to significantly accelerate the inference speed of the multi-task model using post-training quantization with TensorRT.Together, the three components

(Improved YOLOv5s, Edge Computing and TensorRT) complete the training and deployment cycle.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

This project has successfully developed an Improved YOLOv5s model and designed an edge computing system architecture for small target fire detections in residential areas. The model is trained with custom datasets that utilized data augmentation. Based on the results, it is shown that BiFPN and additional prediction layers are recommended for increasing the detection rate of small targets of fire. While the proposed model underperformed in terms of FPS and precision against the original YOLOv5s model, it still provides a comparable result, with a higher recall rate and F1 score.

Although the proposed method is unable to outperform other studies with the highest score, it is able to outperform several studies in terms of latency and Recall rate. The best model could achieve a 0.52 F1 score and a 0.5 recall rate for small target fire detections, while the system is able to carry out fire detection at a latency of 554ms. TensorRT is also shown to be capable of increasing the inferencing speed of the trained models by twice, with a small performance trade-off.

5.2 Recommendations for future work

This project has a lot of room for development. First, a larger dataset could be utilised to train the YOLOv5s-based model, and an ensemble model that incorporates the results of several models could be employed to potentially increase the accuracy of fire detection. The model can be further optimised for use on devices with constrained resources by investigating various architectures, training methods, and deployment options beyond TensorRT, including TensorFlow Lite.

Adding extra capabilities to the Node-RED dashboard to provide more specific information about discovered fires as well as seeking other messaging services besides Telegram for alerting users can both enhance the user interface. Furthermore, the inference speed, mAP, and precision can be greatly increased by optimising the model using the Tensor Cores of the Jetson Nano B01 or by using a more compact and effective model architecture or backbones like MobileNet or ShuffleNet. Overall, these suggestions can improve the fire detection system's functionality and usability for tiny fire detections.

REFERENCES

Ahn, Y., Choi, H. and Kim, B.S., 2022. Development of early fire detection model for buildings using computer vision-based CCTV. *Journal of Building Engineering*, 65, p.105647.

An, Q. et al., 2022. A Robust Fire Detection Model via Convolution Neural Networks for Intelligent Robot Vision Sensing. *Sensors*, 22(8), p.2929.

Avgeris, M. et al., 2019. Where There Is Fire There Is SMOKE: A Scalable Edge Computing Framework for Early Fire Detection. *Sensors*, 19(3), p.639.

BasuMallick , C., 2022, *What Is MQTT (MQ Telemetry Transport)? Working, Types, Importance, and Applications* | [Online]. Available at: https://www.spiceworks.com/tech/iot/articles/what-is-mqtt/ [Accessed: 1 May 2023].

Cytron, *Official NVidia Jetson Nano B01 Kits* [Online]. Available at: https://my.cytron.io/p-jetson-nano-b01-kits [Accessed: 1 May 2023].

David Valverde Cantero, Iker Esnaola-Gonzalez, Miguel-Alonso, J. and E. Jauregi, 2022. Benchmarking Object Detection Deep Learning Models in Embedded Devices. *Sensors*, 22(11), pp.4205–4205.

Google, 2023a, *Choose the Colab plan that's right for you* [Online]. Available at: https://colab.research.google.com/signup [Accessed: 1 May 2023].

Google, 2023b, *Colaboratory - Frequently Asked Questions* [Online]. Available at: https://research.google.com/colaboratory/faq.html [Accessed: 1 May 2023].

He, L. et al., 2021. Efficient attention based deep fusion CNN for smoke detection in fog environment. *Neurocomputing*, 434, pp.224–238.

Huang, J. et al., 2023. A Small-Target Forest Fire Smoke Detection Model Based on Deformable Transformer for End-to-End Object Detection. *Forests*, 14(1), pp.162–162.

Jain, A. and Srivastava, A., 2022. Privacy-Preserving Efficient Fire Detection System for Indoor Surveillance. *IEEE Transactions on Industrial Informatics*, 18(5), pp.3043–3054.

Jocher, G., 2020, *ultralytics/yolov5* [Online]. Available at: https://github.com/ultralytics/yolov5 [Accessed: 24 October 2023].

Khan, S. and Khan, A., 2022. FFireNet: Deep Learning Based Forest Fire Classification and Detection in Smart Cities. *Symmetry*, 14(10), p.2155.

Li, Y. et al., 2023. Real-Time Early Indoor Fire Detection and Localization on Embedded Platforms with Fully Convolutional One-Stage Object Detection. *Sustainability*, 15(3), p.1794.

Maltezos, E. et al., 2022. A Smart Building Fire and Gas Leakage Alert System with Edge Computing and NG112 Emergency Call Capabilities. *Information*, 13(4), p.164.

Mukhiddinov, M., Abdusalomov, A.B. and Cho, J., 2022. Automatic Fire Detection and Notification System Based on Improved YOLOv4 for the Blind and Visually Impaired. *Sensors*, 22(9), p.3307.

NUR ATIKAH , B.Z., 2021, *Statistics On Fire Breakouts, Malaysia - MAMPU* [Online]. Available at: https://www.data.gov.my/data/en_US/dataset/statistics-on-fire-breakouts-malaysia [Accessed: 24 October 2023].

Pincott, J., Tien, P.W., Wei, S. and Calautit, J.K., 2022. Indoor fire detection utilizing computer vision-based strategies. *Journal of Building Engineering*, 61, p.105154. Available at: https://www.sciencedirect.com/science/article/pii/S2352710222011615 [Accessed: 24 October 2022].

Ryu, J. and Kwak, D., 2022. A Study on a Complex Flame and Smoke Detection Method Using Computer Vision Detection and Convolutional Neural Network. *Fire*, 5(4), p.108.

Saydirasulov Norkobil Saydirasulovich et al., 2023. A YOLOv6-Based Improved Fire Detection Approach for Smart City Environments. *Sensors*, 23(6), pp.3161–3161.

SHARMA, A., 2020, *Coco csv format fire object detection* [Online]. Available at: https://www.kaggle.com/datasets/ankan1998/coco-csv-format-fire-object-detection [Accessed: 1 May 2023].

Shokouhi, M. et al., 2019. Preventive measures for fire-related injuries and their risk factors in residential buildings: a systematic review. *Journal of Injury and Violence Research*, 11(1).

Silva, J. et al., 2022. EdgeFireSmoke: A Novel Lightweight CNN Model for Real-Time Video Fire–Smoke Detection. *IEEE Transactions on Industrial Informatics*, 18(11), pp.7889–7898.

Xue, Z., Lin, H. and Wang, F., 2022. A Small Target Forest Fire Detection Model Based on YOLOv5 Improvement. *Forests*, 13(8), p.1332.

Yang, R. et al., 2023. KPE-YOLOv5: An Improved Small Target Detection Algorithm Based on YOLOv5. *Electronics*, 12(4), p.817.

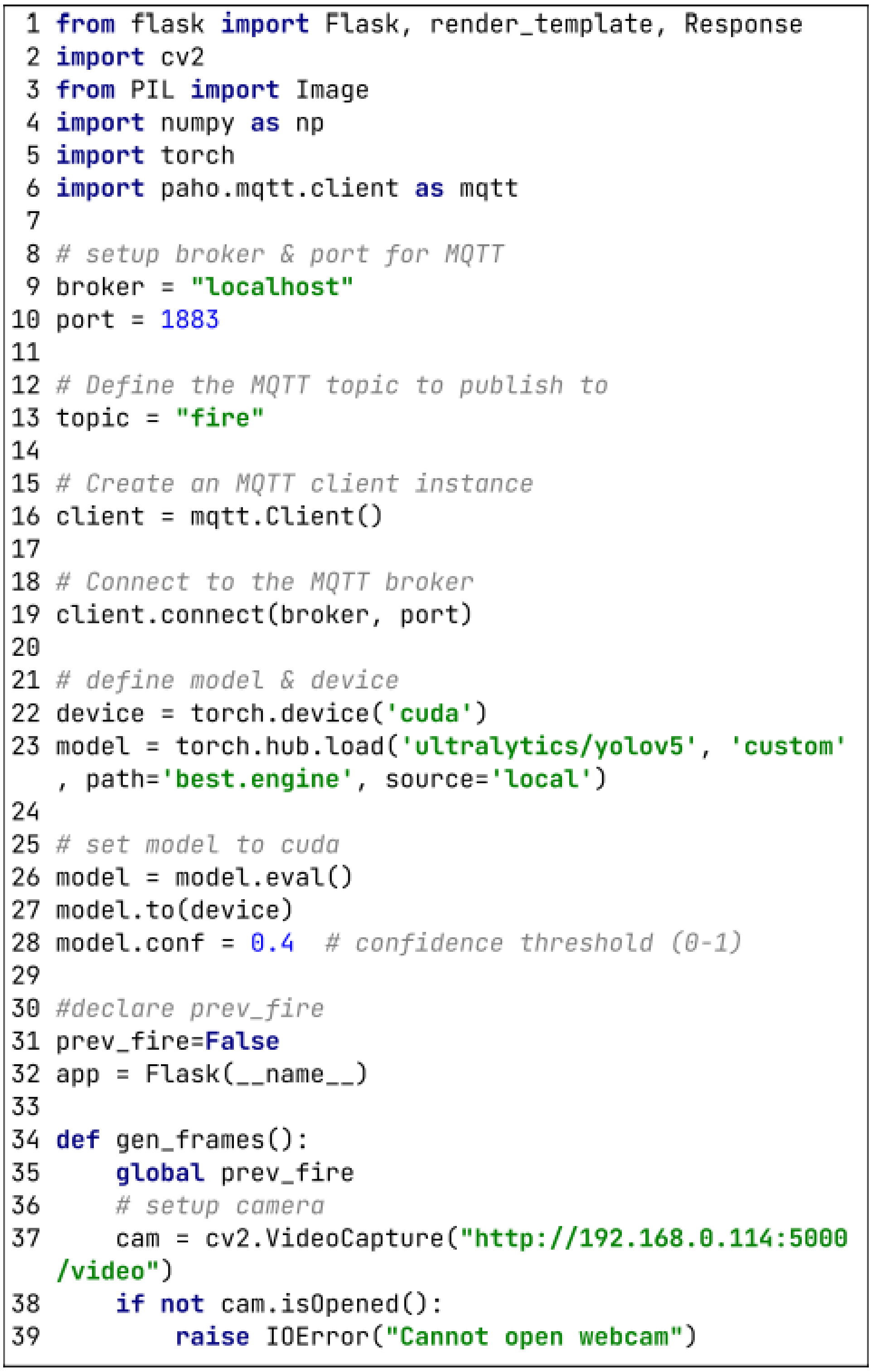
Zhao, L., Zhi, L., Zhao, C. and Zheng, W., 2022. Fire-YOLO: A Small Target Object Detection Method for Fire Inspection. *Sustainability*, 14(9), p.4930.

ZHAO, D. et al., 2022. Coal mine external fire detection method based on edge intelligence. *Journal of Mine Automation*, 48(12), pp. 108–115.

APPENDICES

APPENDIX A: Programme codes

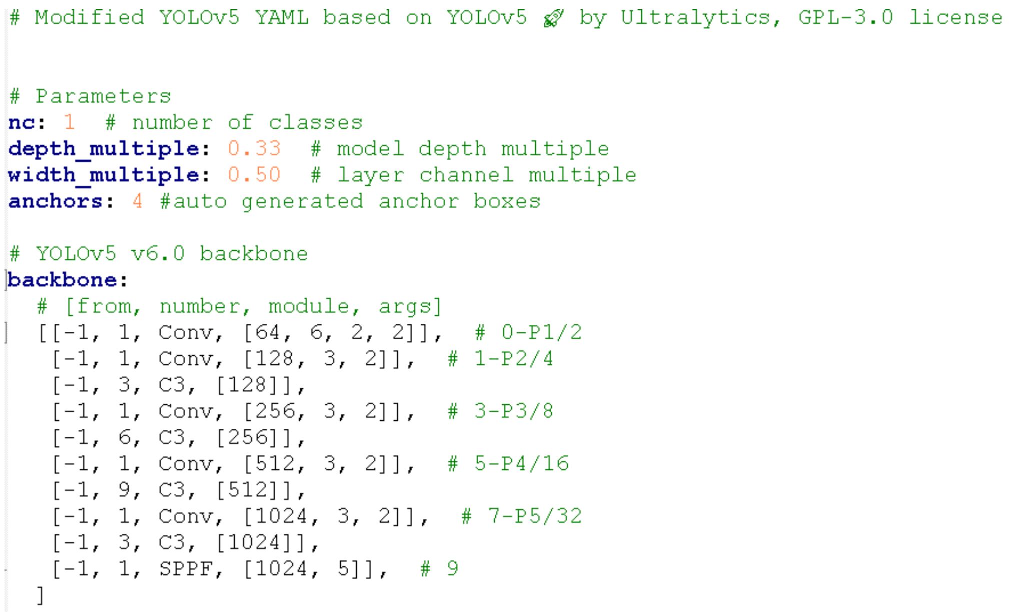
Programming code 5.1: Main.py



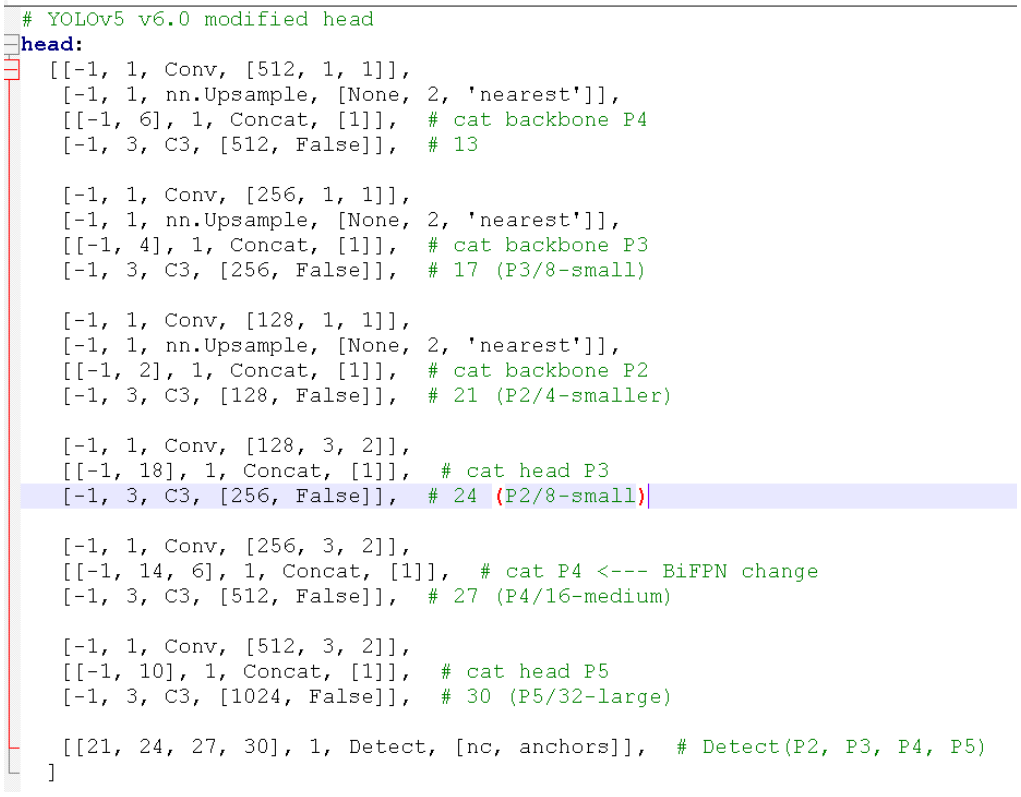
```python
from flask import Flask, render_template, Response
import cv2
from PIL import Image
import numpy as np
import torch
import paho.mqtt.client as mqtt

# setup broker & port for MQTT
broker = "localhost"
port = 1883

# Define the MQTT topic to publish to
topic = "fire"

# Create an MQTT client instance
client = mqtt.Client()

# Connect to the MQTT broker
client.connect(broker, port)

# define model & device
device = torch.device('cuda')
model = torch.hub.load('ultralytics/yolov5', 'custom', path='best.engine', source='local')

# set model to cuda
model = model.eval()
model.to(device)
model.conf = 0.4  # confidence threshold (0-1)

#declare prev_fire
prev_fire=False
app = Flask(__name__)

def gen_frames():
    global prev_fire
    # setup camera
    cam = cv2.VideoCapture("http://192.168.0.114:5000/video")
    if not cam.isOpened():
        raise IOError("Cannot open webcam")
    cam.set(cv2.CAP_PROP_FPS, 1)

    while True:
        ret, frame = cam.read()
        if not ret:
            break
        else:
            resized_frame = cv2.resize(frame, (640, 640))
            pil_image = Image.fromarray(resized_frame)
            with torch.no_grad():
                result = model(cv2.cvtColor(np.array(pil_image), cv2.COLOR_BGR2RGB),size=640)
            if len(result.xywh[0]) > 0:
                dconf = result.xywh[0][0][4]
                dclass = result.xywh[0][0][5]
                if dconf.item() >= 0.4 and dclass.item() == 0.0:
                    if not prev_fire:
                        print("got fire")
                        client.publish(topic, "1")
                        prev_fire = True
                    if prev_fire:
                        print("still got fire")
                if dconf.item() <= 0.4 and dclass.item() == 0.0:
                    print("no fire")
                    prev_fire = False
                    client.publish(topic, "0")
            result=cv2.cvtColor(np.squeeze(result.render()), cv2.COLOR_BGR2RGB)
            ret, buffer = cv2.imencode('.jpg', result)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')


@app.route('/')
```

```python
def index():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')


@app.route('/video', methods=['POST', 'GET'])
def video():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')


if __name__ == '__main__':
    app.run(port=5000, host="0.0.0.0")
```

APPENDIX B: model Yet Another Markup Language format

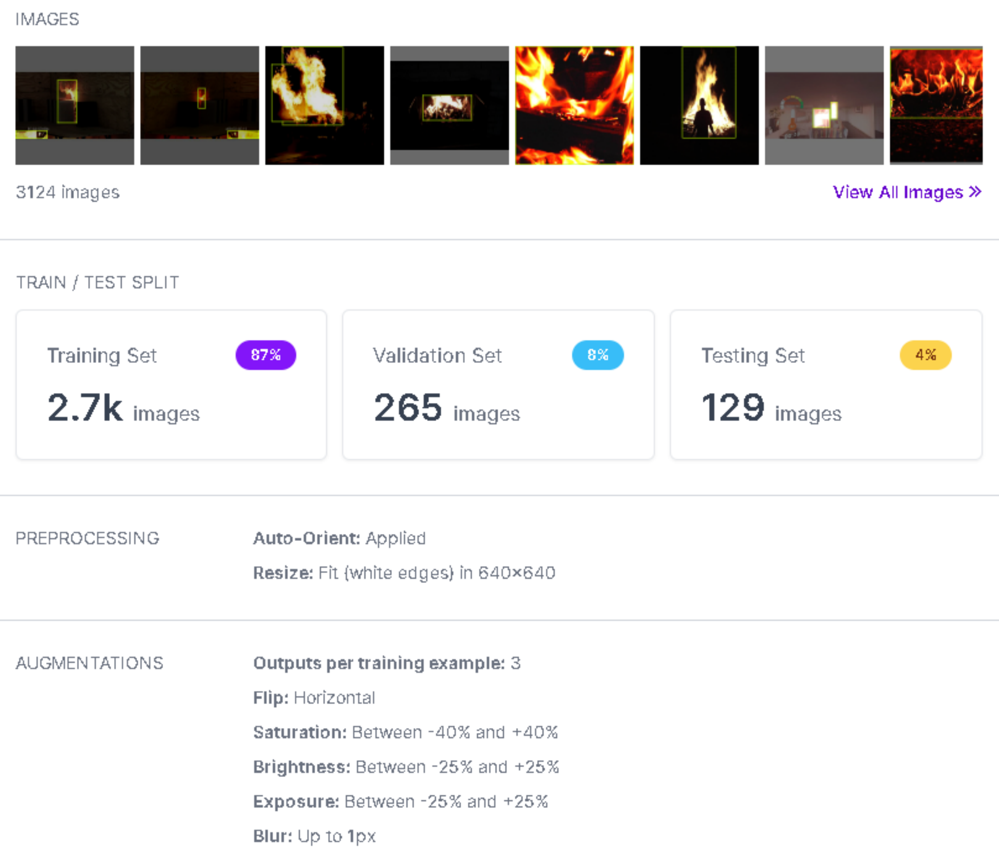


```
# Modified YOLOv5 YAML based on YOLOv5 🚀 by Ultralytics, GPL-3.0 license


# Parameters
nc: 1  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors: 4 #auto generated anchor boxes

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

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

   [-1, 1, Conv, [128, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 2], 1, Concat, [1]],  # cat backbone P2
   [-1, 3, C3, [128, False]],  # 21 (P2/4-smaller)

   [-1, 1, Conv, [128, 3, 2]],
   [[-1, 18], 1, Concat, [1]],  # cat head P3
   [-1, 3, C3, [256, False]],  # 24 (P2/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14, 6], 1, Concat, [1]],  # cat P4 <--- BiFPN change
   [-1, 3, C3, [512, False]],  # 27 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 30 (P5/32-large)

   [[21, 24, 27, 30], 1, Detect, [nc, anchors]],  # Detect(P2, P3, P4, P5)
  ]
```

APPENDIX C: Custom Dataset



IMAGES

3124 images

View All Images »

TRAIN / TEST SPLIT

Training Set 87%	Validation Set 8%	Testing Set 4%
2.7k images	265 images	129 images

PREPROCESSING	**Auto-Orient:** Applied **Resize:** Fit (white edges) in 640×640
AUGMENTATIONS	**Outputs per training example:** 3 **Flip:** Horizontal **Saturation:** Between -40% and +40% **Brightness:** Between -25% and +25% **Exposure:** Between -25% and +25% **Blur:** Up to 1px

https://app.roboflow.com/ds/VdUEWrsEUB?key=zhgAMBqSN3