UNIVERSITY OF SCIENCE
ADVANCED PROGRAM IN COMPUTER SCIENCE

HOANG VU-THIEN

A VISUAL FIRE DETECTION SYSTEM ON EDGE DEVICE

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

HO CHI MINH CITY, 2024

UNIVERSITY OF SCIENCE
ADVANCED PROGRAM IN COMPUTER SCIENCE

HOANG VU-THIEN 19125043

A VISUAL FIRE DETECTION SYSTEM ON EDGE DEVICE

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

THESIS ADVISOR
DUC CHAU-THANH

HO CHI MINH CITY, 2024

ACKNOWLEDGMENTS

I would like to thank my instructor and supervisor, Dr. Duc Chau-Thanh for his guidance throughout the thesis progress. Without his support and advice, this work would not be possible.

I would also like to thank to my family for being very supportive financially and emotionally in my journey to the graduation thesis.

Last but not least, I would like to thank my friends for accompanying throughout my years in college.

TABLE OF CONTENTS

ACKNOWLEDGMENTS i

TABLE OF CONTENTS ii

LIST OF FIGURES vi

LIST OF TABLES viii

ABSTRACT ix

Chapter 1 Introduction 1

1.1 Problem definition and motivation 1

1.2 Objectives 2

1.3 Thesis Outline 3

Chapter 2 Background and related works 4

2.1 Deep learning for object detection and recognition 4

2.1.1 Image Classification 5

2.1.2 Object Detection 8

2.2 Deep learning for visual fire detection 11

2.2.1 Fire recognition methods 12

2.2.2 Fire object detection approaches 13

2.2.3 Multi-stage/multi-task methods 14

2.3 Fire detection on smart edge device 15

Chapter 3 Proposed solution 17

3.1 Solution overview 17

3.2 Workflow breakdown 18

3.2.1 Detection module 18

3.2.2 Verification module 19

3.2.3 Stacking-and-voting mechanism 19

3.2.4 Notification System 20

3.3 Hardware configuration 21

3.3.1 Design 21

3.3.2 Hardware adaptation . . . 22
Chapter 4 Experiments . . . 24
4.1 Datasets description . . . 24
4.1.1 FASDD dataset . . . 24
4.1.2 Self-recorded dataset . . . 25
4.2 Experiment design . . . 26
4.2.1 Detection Model . . . 26
4.2.2 Classification Model . . . 28
4.3 Metrics . . . 28
4.3.1 Detection model evaluation . . . 28
4.3.2 Classification model evaluation . . . 29
4.3.3 Combined workflow evaluation . . . 30
4.4 Training environment . . . 31
Chapter 5 Results and discussion . . . 32
5.1 Detection . . . 32
5.2 Classification/Verification . . . 33
5.3 Combined Detection Flow . . . 33
5.4 Discussion . . . 36
Chapter 6 Conclusion . . . 38
6.1 Contributions . . . 38
6.2 Future work . . . 39
REFERENCES . . . 40
APPENDICES . . . 44
Chapter A Stacking-and-Voting Mechanism Analysis . . . 44
A.1 Preliminaries . . . 44
A.2 Example . . . 45
A.3 Conclusion . . . 45
Chapter B YOLOv5, YOLOv5-P6, YOLOv8 architecture . . . 47
B.1 YOLOv5 architecture . . . 47

B.2 YOLOv5-P6 architecture 48
B.3 YOLOv8 architecture 49
Chapter C Code snippet for Telegram bot API 50
C.1 Introduction 50
C.2 Basic usage 50
C.3 Use with python multi-threading 52
Chapter D Mean Average Precision in Object Detection 54
D.1 Precision and Recall 54
D.2 Intersection Over Union (IoU) 54
D.3 Mean Average Precision (mAP) 55

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
FASDD	Flame And Smoke Setection Dataset
IOU	Intersection over union
YOLO	You Only Look Once
ONNX	Open Neural Network Exchange
TP	True Positive
TN	True Negative
FN	False Negative
FP	False Positive
mAP	Mean Average Precision
FAR	False Alarm Rate
FNR	False Negative Rate
ViT	Visual Transformer
DEiT	Data-efficient Image Transformer
BEiT	Bidirectional Encoder representation from Image Transformers
SSD	Single Shot MultiBox Detector
FCOS	Fully Convolutional One-Stage object detection
UAV	Unmanned Aerial Vehicle
FLOPS	Floating-point operations per second
SGD	Stochastic Gradient Descent
CPU	Central Processing Unit
GPU	Graphics Processing Unit
API	Application Programming Interface

LIST OF FIGURES

1.1 Fire incidents in Vietnam in 2022 and 2023 2
1.2 Fire casulties in Vietnam in 2022 and 2023 2
1.3 Fire property damage in Vietnam in 2022 and 2023 2

2.1 AlexNet architecture [1] 5
2.2 Residual block in ResNet [2] 5
2.3 MobileVit [3] architecture 7
2.4 Traditional Multi-headed self-attention [4] and Separable self-attention on MobileVitV2 [5] 7
2.5 Faster R-CNN architecture [6] 8
2.6 Xu et al. proposed solution 15
2.7 FireNet [7] hardware setup and model archtecture 15

3.1 Fire detection workflow 17
3.2 Our proposed hardware setup 21
3.3 ONNX high-level architecture 22

4.1 FASDD example images, figure are taken from [8] 24
4.2 Self-recorded dataset samples with annotations 25

5.1 Result for combined MobileNetV3 (Finetuned with FASDD) with YOLOv5n6 33
5.2 Result for combined MobileNetV3 (Finetuned with self-recorded data) with YOLOv5n6 33
5.3 Result for combined MobileVitV2 (finetuned with FASDD) with YOLOv5n6 34
5.4 Result for combined MobileVitV2 (finetuned with self-recorded data) with YOLOv5n6 34

5.5 Comparison two combined systems with the same detection model, different verification model and same verification finetuning data. 34

B.1 YOLOv5l architecture [9] 47

B.2 YOLOv5-P6 architecture, the block used in this diagram is the same as the YOLOv5 with additional P6 output 48

B.3 YOLOv8 architecture by GitHub user RangeKing 49

C.1 BotFather informations 50

D.1 Intersection over Union 54

LIST OF TABLES

3.1 Comparision of YOLO models 18

3.2 Proposed hardware specifications 21

4.1 Hyperparamaters for training detection models. 27

4.2 Hyperparameter for classification training 28

5.1 Detection model performance benchmark. Model speed is recorded on Raspberry Pi 4 model B 4GB, both models are exported to ONNX format, input image dimension is set to 640. All test are performed on test splits of corresponding data. 32

5.2 Classification model performance. Model speed is recorded on Raspberry Pi 4 model B, both are exported to ONNX, input image dimension is set to 224. All test are performed on test splits of corresponding data. 33

5.3 Performance of 4 combination of our system. Test conducted on the test split of self-recorded dataset. Result of our system is highlighted in bold text. 35

ABSTRACT

Fire incidents pose a significant societal challenge, causing not only substantial financial damage but also leading to a high number of casualties. This issue is particularly acute in Vietnam, where residential areas are increasingly affected. Traditional sensor-based methods, while accurate, are limited by factors such as cost and operational range. With the advent of artificial intelligence, deep learning-based fire detection methods have emerged as effective alternatives. This thesis presents a novel visual fire detection system designed for deployment on edge devices with limited computational resources. The proposed system comprises four main components: a lightweight detection model, a classification model, a Stacking-and-Voting mechanism, and a notification system. The models was trained and tested using a large-scale open-source dataset and a self-recorded dataset. The final combined system achieved a high recall rate along with a very low false positive rate per frame on the self-created dataset, while also demonstrating its ability to operate in real-time on a Raspberry Pi 4.

Chapter 1
Introduction

1.1 Problem definition and motivation

Fire accidents have always been a serious problem, causing deaths and injuries along with significant financial damage. These incidents, often unexpected and rapid, can devastate communities, leaving behind a trail of destruction. They not only pose a threat to human life and physical infrastructure, but also have profound psychological impacts on the victims and their families. The economic implications are equally severe, with businesses and individuals suffering substantial losses that can take years to recover from.

The escalating severity of fire-related incidents in Vietnam has become a matter of grave concern. In 2023 [10], the country witnessed a surge in fire cases, with the count reaching 3,440, marking a significant increase from the 3,234 cases reported in 2022. The human toll of these incidents has also seen a worrying rise, with the number of fatalities and injuries in 2023 standing at 146 and 109 respectively, reflecting an increase of 22.69% from the previous year. In 2022, the figures were slightly lower but still alarming, with 119 deaths and 90 injuries, marking an increase of 21%. The financial implications of these incidents are equally distressing. Fire-related property damage in 2023 was 38% higher than the previous year, amounting to a staggering 878 billion VND, compared to 634 billion VND in 2022. Furthermore, residential fire incidents, which constitute 29.5% of total incidents in Vietnam, underscore the urgent need for improved fire detection measures, particularly in domestic households. While the market offers a variety of traditional sensor-based detectors, these solutions are not without their limitations. Being sensor-based, these devices rely heavily on close proximity for effective operation, making it challenging to detect fires in large areas. Another byproduct of such reliance on proximity is the slow detection rate or long delayed detection. Lastly, conventional sensor-based detector does not offer the ability to connect to the actual user through internet, making alarm may not be handle

in time or at all. Therefore, there is a pressing need for a faster-acting solution with the ability to alarm owner remotely, thereby minimizing the damage caused by fires.

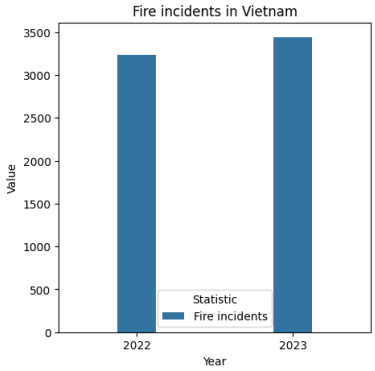

Figure 1.1. Fire incidents in Vietnam in 2022 and 2023

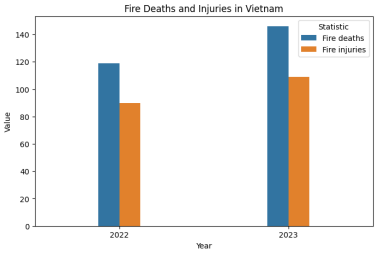

Figure 1.2. Fire casulties in Vietnam in 2022 and 2023

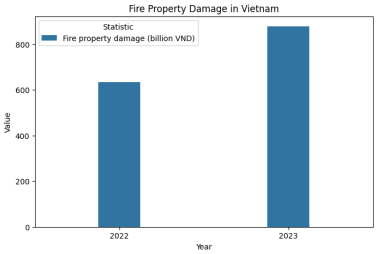

Figure 1.3. Fire property damage in Vietnam in 2022 and 2023

1.2 Objectives

The main goal of this study is to design and implement a fire detection system to replace such traditional solution. To achieve this, the study has been divided into four specific objectives:

First, the system should be designed to accurately detect fires. This involves two key aspects. First, the system should have minimal instances of false negatives as this en-

sures alarm in case of fire. Second, the chance of false alarm should be manage well enough to avoid unnecessary panics and annoyances to the owner.

Second, the system have to be ensured that the system can keep up with real-time inputs and output early alarm. This means that the system should be capable of processing and analyzing data as it comes in, without any significant delay. This is crucial for a fire detection system as fires can spread rapidly, and any delay in detection can have severe consequences.

Third, the system should be flexible and compact enough to be installed in indoor environments, considering this solution is a better alternative for the traditional system.

Last but not least, the system should have an actual output for long-range alarm, preferably a notification system. This is crucial for practical usage as this allows owner to take action whenever there is fire in the scene.

By focusing on these objectives, the study aims to develop a lightweight fire detection system that is not only accurate and reliable but also efficient and timely. This will significantly contribute to enhancing fire safety measures, particularly in residential areas where the risk of fire accidents is high.

1.3 Thesis Outline

The structure of this thesis is as follows: Chapter 2 delves into the related works and concepts, providing an overview of state-of-the-art solutions to deep learning fire detection problems, as well as a concise survey on deep learning with edge devices. Chapter 3 introduces our proposed solution, detailing each component for a comprehensive understanding of the system. Chapter 4 is dedicated to the dataset and the training strategies we adopt for our proposed system, as well as the design of experiments. Chapter 5 covers the experiments conducted on each main component, as well as the performance of the combined system. This chapter also includes a discussion section where we reflect on the results and their implications. Finally, Chapter 6 concludes the thesis, summarizing the work and providing our insights for future research and suggestions for improvement.

Chapter 2
Background and related works

2.1 Deep learning for object detection and recognition

In every aspect of life in the post COVID-19 era, artificial intelligence has become a familiar concept. It started with machine learning, a set of algorithms that can learn and improve based on data, or experience from a human perspective.

Deep learning, a subset of machine learning, has shown remarkable results in various visual tasks. It leverages neural networks with many layers (hence ”deep”) to learn representations of data with multiple levels of abstraction. It has proven to be the robust solution for many field such as computer vision, natural language processing, speech recognition and many more. As the advances of hardware power and optimization method, deep learning is accessible by any organizations and individuals and fuel a lot of idea to improve human well-being.

When it comes to visual tasks, Convolutional Neural Networks (CNNs) has been well-known for their effectiveness. They process images through a series of neural blocks, each consisting of convolution layers for feature extraction and pooling layers for dimension reduction. The network learns to detect simple features in the early layers and more complex patterns in the deeper layers. Fully connected layers are used for high-level reasoning, and modern CNNs often include additional layers such as batch normalization and dropout for improved performance.

Recently, Transformer, a model known for its effectiveness for natural language processing tasks as it considers dependencies of every aspect of the input with self-attention mechanism, has been introduce to visual tasks. It shows promising results compared to traditional CNNs in various computer vision tasks, as it enables global features consideration. However, as the self-attention mechanism requires considerably more computational power, CNN is still a great choice for many projects.

2.1.1 Image Classification

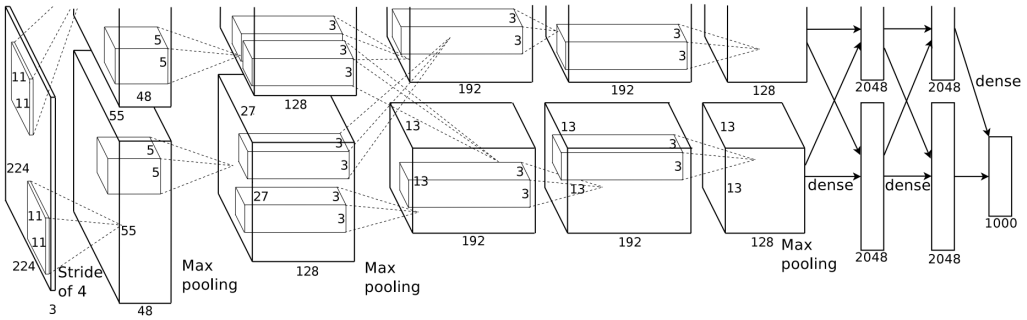

Figure 2.1. AlexNet architecture [1]

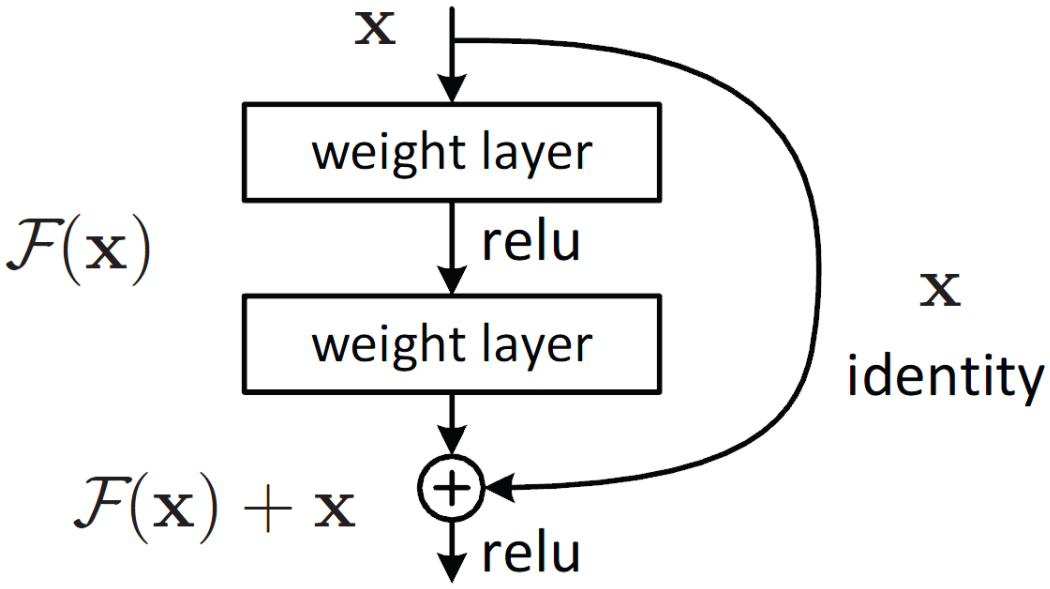

Figure 2.2. Residual block in ResNet [2]

Image classification is a fundamental task in computer vision, where the goal is to categorize an image into one of several predefined classes. After the success of AlexNet [1] in 2012, CNNs became the backbone of most image classification tasks. However, as networks became deeper, they started facing problems like vanishing gradients and increased computational complexity. VGGNet [11], for instance, with its 19-layer architecture, was computationally intensive and had a large number of parameters, making it difficult to train. The introduction of ResNet [2] in 2016 addressed these issues with the concept of residual networks. ResNet [2] introduced

”shortcut connections” or ”skip connections” that allow the gradient to be directly backpropagated to earlier layers, mitigating the vanishing gradient problem and enabling the training of much deeper networks. Following ResNet [2] and the development of residual networks, EfficientNet [12] was proposed as a systematic approach to network scaling. Instead of arbitrarily increasing the depth or width of the network, EfficientNet [12] scales all dimensions of the network (depth, width, and resolution) based on a set of fixed scaling factors, leading to better performance with fewer parameters. With the success in natural language processing, researcher has began to apply transformer for visual tasks, and the first success emerged when Visual Transformer (ViT) [13] was introduced by Google. ViT treats an image as a sequence of patches and applies self-attention to capture dependencies between them, showing promising results. It goes on par with ResNet given the same model size in ImageNet and surpass ResNet with larger dataset, making them scale very well with the size of the dataset [13]. It still, however, comes with various weaknesses, mainly in its demanding training process. Researchers also works on that and make significant progress to work on that weakness, with the introduction of self-supervised learning (Bidirectional Encoder representation from Image Transformers (BEiT) [14]), knowledge distillation (Data-efficient Image Transformer(DEiT) [15]).

Meanwhile, in the pursuit of models that can operate efficiently on mobile devices, architectures like MobileNet [16] were developed. These models introduced techniques like depthwise separable convolutions to reduce computational complexity while maintaining competitive performance. With the rise of visual transformers recently, hybrid models like MobileViT [3] and MobileViTv2 [5] have emerged. These models combine the strengths of CNNs and Transformers, using a CNN as a feature extractor at the lower layers and a Transformer at the higher layers to capture global dependencies. For later version of these models, the self-attention mechanism is also modified to further reduce the runtime complexity, making these models an even more suitable. MobileVitV2 [5] is a clear example for that: it adjusts the transformer block of the predecessor MobileVit by changing multi-headed self-attention mechanism to

separable self-attention mechanism, reducing the complexity of attention modules to linear complexity while also avoiding the use of costly operation (i.e batch-wise matrix multiplication) by utilizing element-wise operations.

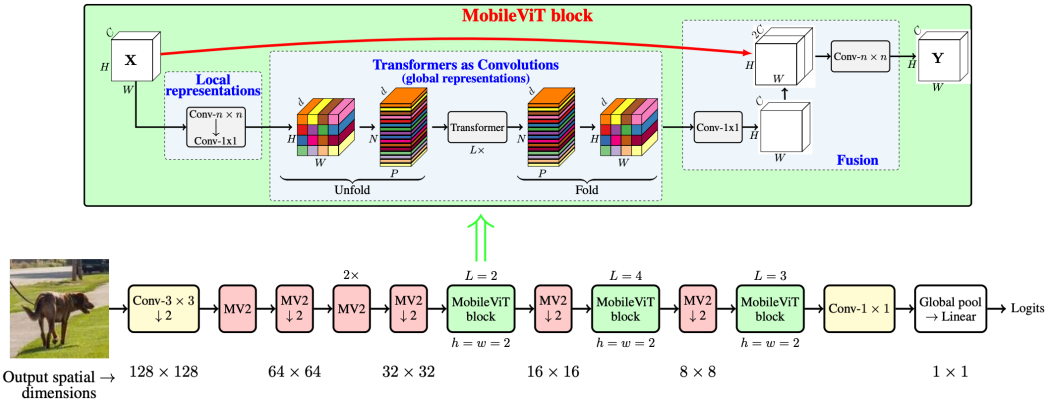

Figure 2.3. MobileVit [3] architecture

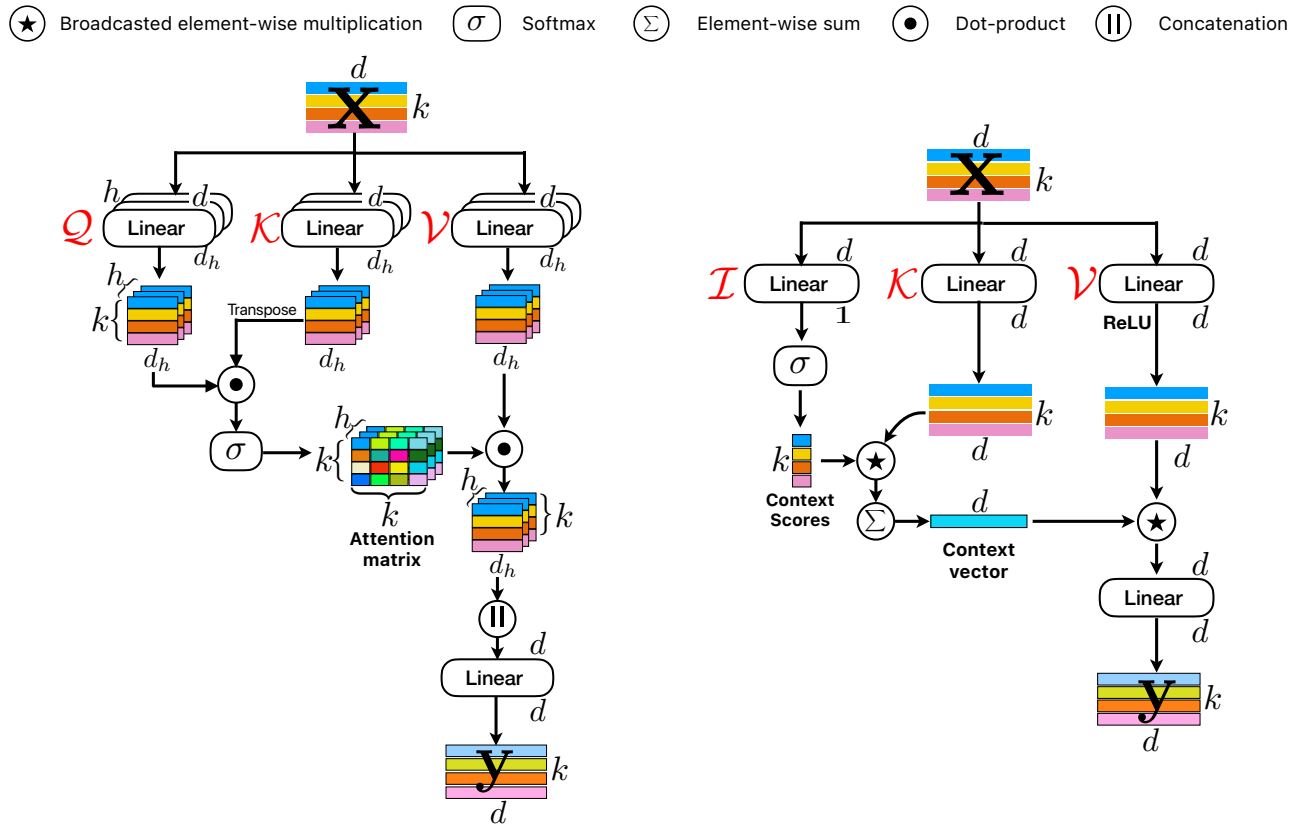

Figure 2.4. Traditional Multi-headed self-attention [4] and Separable self-attention on MobileVitV2 [5]

2.1.2 Object Detection

Object detection, which extends image classification, involves not only identifying objects but also locate them within the image. State-of-the-art object detection models are categorized by two main approaches: one-stage detection models, (including You Only Look Once (YOLO) [17], Single Shot MultiBox Detector (SSD) [18], Fully Convolutional One-Stage object detector (FCOS) [19], etc) and two-stage detection models (including Regions with CNN features (R-CNN) [20], Fast R-CNN [21], Faster R-CNN [6], etc). One-stage detectors focus on speed while two-stage detector tend to yield more accurate result. We will go through each well-represented model family of both categories, namely R-CNN and YOLO, with the focus on YOLO.

2.1.2.1 R-CNN

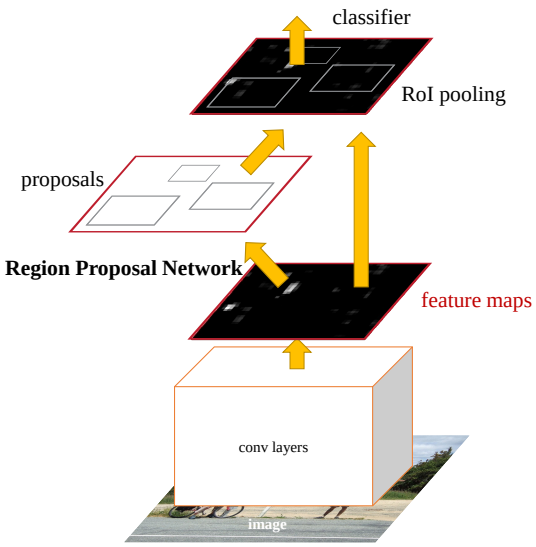

Figure 2.5. Faster R-CNN architecture [6]

R-CNN [20], a two-stage detector, initially employs selective search to extract regions of interest (RoIs) and computes features from each region for classification. This process, however, is time-consuming due to the vast number of RoIs. Fast R-CNN [21] addresses this issue by computing the features of the entire image instead of each individual region. Despite this improvement, the model's runtime remains lengthy, up to 2 seconds on a NVIDIA K40 Graphics Processing Unit GPU, pri-

marily due to the continued use of selective search. Faster R-CNN further improves upon this by proposing a Region Proposal Network, which allows shared computation across both stages, resulting in a higher detection speed suitable for real-time detection. However, this model may not be ideal for budget-restricted environments due to its substantial computational power requirements.

2.1.2.2 YOLO

As mentioned above, YOLO [17] is a single-stage object detection model that predicts the bounding boxes and class probabilities of objects in an image simultaneously, unlike traditional two-stage models that require region proposal networks and feature extraction. In its initial version, YOLO divided the input image into a grid of cells, and each cell was responsible for predicting the bounding boxes and class probabilities of objects in an image. In the evolution of YOLO, a significant enhancement was introduced in YOLOv2 [22] with the concept of anchor boxes. These predefined boxes of certain shapes and sizes were assigned to each grid cell. The anchor boxes improved the model's ability to detect objects of various sizes and aspect ratios, as each anchor box was responsible for predicting objects that had similar shapes and sizes.

With the introduction of YOLOv3 [23], another significant enhancement was made with the integration of the Feature Pyramid Network (FPN). FPN is a structure for multi-scale object detection. It enhances the ability of the model to detect objects at different scales by constructing a feature pyramid with an additional top-down pathway and lateral connections. This allows the model to use high-level semantic feature maps at all scales, improving performance especially on detecting smaller objects. In YOLOv3, this concept is realized through the use of three different scales during prediction, each responsible for detecting objects of different sizes. This multi-scale prediction is one of the reasons why YOLOv3 improved upon its predecessors in terms of detection performance across a variety of object sizes.

2.1.2.3 YOLOv5

YOLOv5 [24], an active development by the open-source community led by Glenn Jocher and Ultralytics, represents a significant evolution in object detection. Building upon the foundation laid by YOLOv3, YOLOv5 further refines the use of the Feature Pyramid Network (FPN) for detecting objects at different scales, thereby enhancing the model's efficiency and effectiveness. In addition to refining the use of FPN, YOLOv5 introduces a new module known as C3. The C3 module, a modified version of the CSPNet [25], includes three convolutional layers and a bottleneck structure. This structure increases computational speed and reduces parameter complexity, making the model more efficient. Another significant feature of YOLOv5 is AutoAnchor. AutoAnchor automatically generates anchor boxes using K-means clustering and Genetic Algorithm. This feature improves the model's performance in detecting objects of various sizes and shapes, addressing one of the limitations of the anchor boxes in earlier versions. YOLOv5 also incorporates the concept of model scaling, similar to EfficientNet in image classification, providing multiple version of the model (YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x). Each version of model varies in size and complexity, allowing user to select the suitable one that optimally balances their requirements for speed and accuracy.

Building upon these features, an enhanced variant known as YOLOv5-P6 was introduced. YOLOv5-P6 is specifically designed to handle a common challenge in YOLOv5 - the detection of extra-large objects in images. By adding a P6 layer, YOLOv5-P6 extends the capabilities of the original model, making it more versatile and effective on datasets with a wide range of object sizes. This enhancement is particularly useful in real-world scenarios where objects in images can vary significantly in size. The architecture of YOLOv5 and YOLOv5-P6 is included in the Appendices

2.1.2.4 YOLOv8 and YOLOv5u - the anchor-free YOLO

Building on the success of YOLOv5, the Ultralytics team has advanced the field of object detection with the introduction of YOLOv8 [26]. YOLOv8 emerges as the state-of-the-art model, outperforming its predecessors by incorporating significant architectural improvements and optimization techniques. YOLOv8 distinguishes itself with a more efficient backbone. YOLOv8 introduces the use of the C2f module replacing the previously used C3 module. The C2f module, a novel architectural component, contributes to the model's enhanced performance. This module is designed to facilitate more effective feature fusion, thereby improving the model's ability to detect objects across a range of scales and complexities. The introduction of the C2f module in YOLOv8 underscores the model's innovative design and the continuous efforts to optimize object detection performance.enhanced feature extraction capabilities, and a refined loss function that collectively improve detection accuracy and speed. In addition to the change in the backbone, YOLOv8 also employ a anchor-less detection head. With this new head, the amount of candidate bounding boxes is reduced, therefore speeding up the Non-max Suppression (NMS) process. This will reduce the inference time substantially due to the focus for parallel computing (less sequentially CPU runtime).

Inspired by YOLOv8, Ultralytics has released YOLOv5u [24] model family, with the most noticeable change is the anchor-less detect head while still keeping the backbone and neck of the original YOLOv5. By eliminating the need for predefined anchor boxes, YOLOv5u simplifies the training process and reduces the model's reliance on hyperparameters. This change enhances the model's ability to generalize across diverse datasets and object sizes, resulting in more robust and accurate detections. The architecture of YOLOv8 is included in the Appendices.

2.2 Deep learning for visual fire detection

In this section, we delve into the application of deep learning techniques in the realm of fire detection, particularly focusing on visual data. The recent advancements

in deep learning have revolutionized computer vision, leading to remarkable improvements in image recognition and analysis. We will explore how these advancements are leveraged for fire detection, transforming traditional methods by enhancing accuracy and speed. This includes a survey on various deep learning models, their architecture, and their performance in the respective designated environment.

2.2.1 Fire recognition methods

In these method, a classification model is usually employed to tell if fire incident is present in the image. CNNs are frequently used as a solution. Muhammad et al. [27] proposed a method that utilize a modified lightweight SqueezeNet, as it uses smaller convolution kernels and drops the fully connected layer. The method achieve precision of 86% and F-measure score of 91% on BoWFire dataset, while still being more lightweight than the state-of-the-art of the model at its time. In addition to the proposed change to model, the authors also develop an algorithm to segment fire for better analysis, which allow appropriate action. Majud et al. [28] proposed a system utilizing EfficientNet-B0 with attention mechanism to enhance the GRAD-CAM visualization output as attention mechanism is claimed to focus better on fire objects. The method achieve 95.40% at accuracy and a recall rate of 97.61% on a self-created dataset including fire and non-fire images.

With the rise of visual transformer in recent years, researchers has started to incorporate the state-of-the-art model to their works. Zhang et al. [29] proposed a solution for fire recognition in power plant employing ViT as the classification model. On the authors' self-crawled dataset, ViT outperform 4 others CNN options, with 97.03% accuracy rate and 93.12% F1-score. Shahid et al. [30] proposed a fire identification system with ViT. The authors claimed that ViT show better performance than the other CNN methods, including the SqueezeNet solution by Muhammad et al. [27]. The best model, ViT-B/32 scores 97.09%, 99.10%, 98.08% in recall, precision and F1-Score on BoWFire [31] dataset, respectively. It also achieves 2.15% false positive rate, 1.02% false negative rate, and 94.03% accuracy on Foggia dataset [32].

The above-mentioned methods have shown significant progress in fire recognition, especially with the incorporation of visual transformers. However, it is important to note that these methods are not without their limitations. For instance, while the SqueezeNet modification proposed by Muhammad et al. [27] achieved impressive results on the BoWFire dataset, it may not perform as well on other datasets due to differences in image characteristics. Similarly, the attention mechanism utilized by Majud et al. [28] may not always focus on the correct fire objects, leading to potential inaccuracies. Furthermore, while the visual transformer models proposed by Zhang et al. [29] and Shahid et al. [30] outperformed other CNN methods, they are computationally expensive and may not be suitable for real-time applications.

2.2.2 Fire object detection approaches

Similar to the object detection scene, methods for fire-object detection are also divided into two main approach: two-stage detectors and one-stage detectors. For two-stage detection, Faster R-CNN is one of the popular choice [33, 34]. However, given the urgent nature of fire detection problems and the rapid advancements in the accuracy of one-stage detectors, the latter has garnered increased attention recently, particularly the YOLO-based approach. Miao et al. [35] proposed an improved YOLOv5s with the usage of Squeeze-and-Excitation channel attention mechanism and Ghost [36] layer, creating a more efficient yet more accurate than the original model, as the proposed method achieve 80.6%, which improved 3.2% compared to the original YOLOv5s. Chen et al. [37] proposed another modified YOLOv5s, in which the Contextual transformer block was use along with Coordinate attention block is used to better capture the global feature of the image. The author also employs the Bi-directional Feature Pyramid Network, which even further utilize the global feature extracted from those attention/transformer block earlier. The proposed model achieve a mAP@0.5 (Mean Average Precision at 0.5 Intersection over Union threshold) score of 87.7%, which is 6.2% higher than the unmodified version, while maintaining a comparable inference speed (36.6 FPS compared to 43.8 FPS with YOLOv5s).

Visual transformer with self-attention mechanism also make its appearance in recent fire-object detection methods. Li et al [38] proposed a CNN-fused DETR, in which the CNN paired with normalization-based attention [39] acts as a backbone for feature extraction, combined with deformable attention used in encoder-decoder modules. The proposed method aims to accelarate the training process while also achieve decent detection rate - with AP_{smoke} score of 76.0% and AP_{fire} score of 81.7% on a dataset of 26060 images including fire and smoke instances. However, akin to other visual transformers model, this proposed solution requires great computational power, which may be a hinder to run real-time without large servers or powerful GPUs.

2.2.3 Multi-stage/multi-task methods

In the subsequent phase of this study, we delve into the utilization of an ensemble or two-stage system, a methodology frequently employed in prior research. This system often leverages a classification model to verify the scene for detected objects to go through. Bahhar et al. [40] proposed a system comprising two modules: detection and classification. The detection module utilizes two YOLOv5 models for a two-stage detection process, one model for smoke detection and the other for fire detection. The classification module consists of three parts: a data processing component, a feature extractor paired with a global average pooling layer, and a predictor at the end of the module. The proposed model reportedly yields robust results: the classification module achieves a F1-score of 95%, an accuracy rate of 99%, and a recall rate of 98%. Meanwhile, the detection module attains an mAP@0.5 of 85% for smoke and 76% for the combined model. Xu et al. [41] proposed an ensemble learning solution. The authors employ YOLOv5 and EfficientDet for fire object detection, paired with EfficientNet as a global recognition system to guide the detection module. The proposed solution achieves a detection rate of 98.9% and a false positive rate of merely 0.3%. Overall, the approaches in this category offer a promising performance - low false alarm rate and high detection rate - as well as the ability to customize for one's own need, especially for budget-restricted projects with low-end device settings.

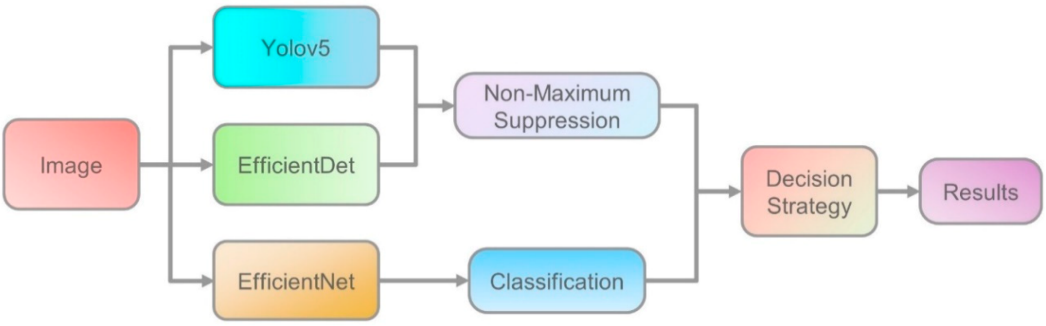

Figure 2.6. Xu et al. proposed solution

2.3 Fire detection on smart edge device

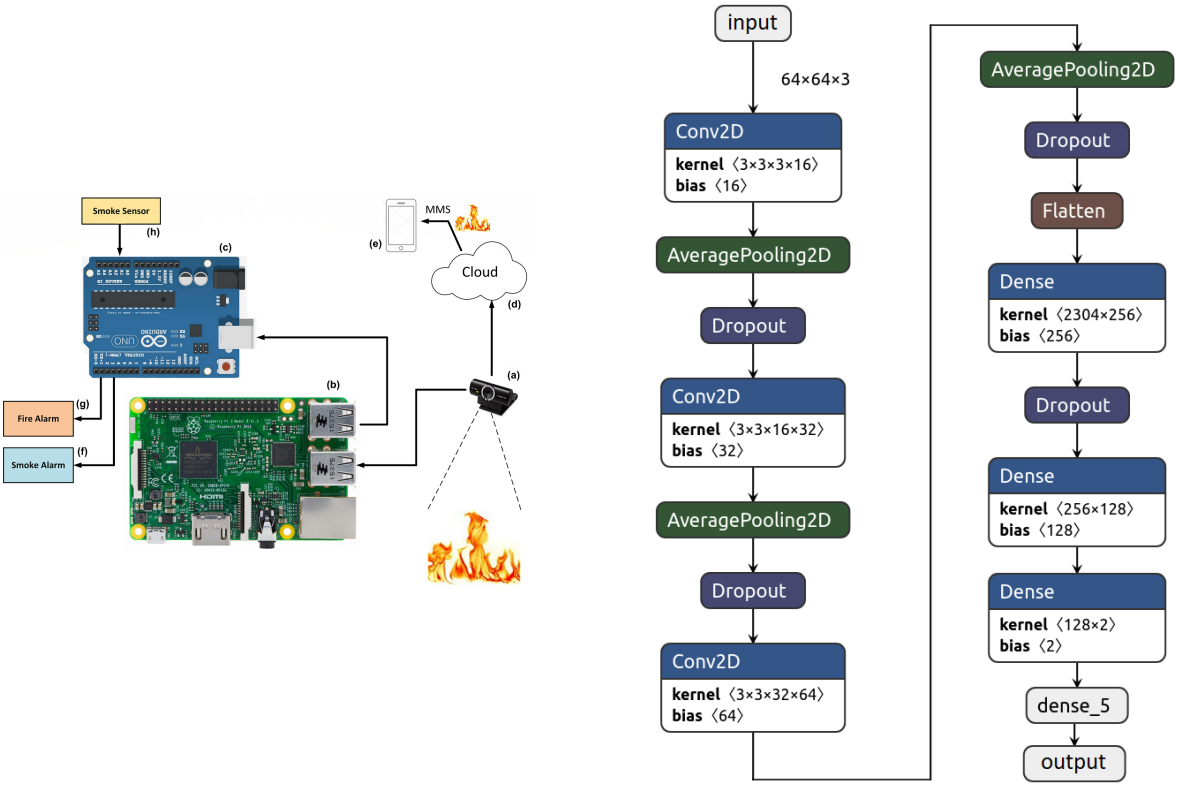

Figure 2.7. FireNet [7] hardware setup and model archtecture

As deep learning solutions are being employed in the IoT systems, specifically in smart edge devices where the edge device is capable of running deep neural networks, research for viability in such devices take place. Nguyen et al. [42] proposed a real-time fire detection system for UAV(Unmanned aerial vehicle). The author deployed the SSD model, with MobileNet as the base model, to the Nvidia Jetson Nano. The

system achieve 92.7% mAP with 26 FPS on the operating device. Jadon et al. [7] proposed a lightweight model on Raspberry Pi 3 Model B. The author also connect the main device to a microcontroller that integrate a smoke sensor and two different alarms to distinct fire and smoke scenario. The model achieve 96.53% on Foggia's [32] dataset and an impressive 24 frames per second on the intended device.

Chapter 3
Proposed solution

3.1 Solution overview

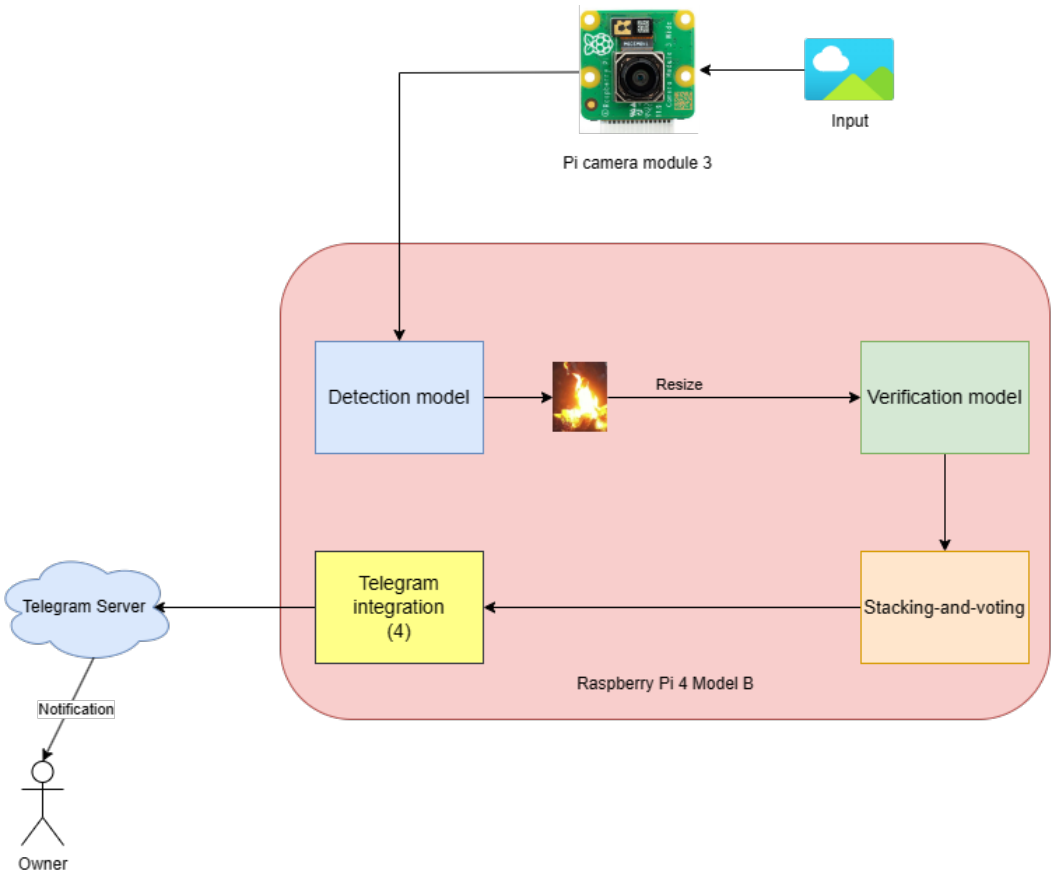

Figure 3.1. Fire detection workflow

In previous section, we have mentioned that multi-models/multi-stage approach achieve promising results while being highly customizable. Based on the statement, in this study, we propose a two-stage fire detection workflow on a Raspberry Pi 4 Model B. This approach allows us to use lightweight models while maintaining high level of detection precision. The first stage involves a lightweight detection model that identifies potential fire regions in video frames. The second stage is a verification model that classifies these regions as fire or non-fire, thereby reducing the false positive rate. We also introduce a voting system that leverages the continuity of video detection to further decrease the false alarm rate and improve the overall detection

rate. Finally, when an incident is detected, a notification is sent to the user via Telegram bot. This two-stage detection workflow, combined with a voting system and a notification feature, offers a promising solution for reliable and efficient fire detection on our device.

The following sections will discuss each components of the system, as well as its proposed running environment.

3.2 Workflow breakdown

3.2.1 Detection module

Model	mAP^{val}_{50-95}	Params (M)	FLOPs (B)
YOLOv5n	28.0	1.9	4.5
YOLOv5n6	36.0	3.2	4.6
YOLOv5nu	34.3	2.6	7.7
YOLOv5n6u	42.1	3.2	7.8
YOLOv8n	37.3	3.2	8.7

Table 3.1. Comparision of YOLO models

In the detection stage of our solution, we aim to use a lightweight model due to the constraints of our choice of devices. Among the various models available, the YOLO family stands out for its speed and compactness, making it particularly suitable for our task. We decide to employ the model from Ultralytics (i.e YOLOv5, YOLOv5u and YOLOv8) as they has been well developed for easy training for comprehensive comparison as well as rapid deployment due to . As can be seen in 3.1, the anchor-based YOLOv5n is the lightest model, with the least parameters and the smallest floating-point operations per second (FLOPs), following by YOlOv5n6 with only 2% difference in terms of FLOPs. In terms of accuracy, the anchor-less YOlOv5n6u and YOLOv8 come up on first and second respectively, with YOLOv5n6 comes in third. After consider all factor of speed-accuracy trade-off, we decide to employ YOLOv5n6

model. However, for more insights and better recommendations, we will include both YOLOv5n6u, as an anchor-free representative, and YOLOv5n6 in comparison in section 5.

3.2.2 Verification module

In our study, we also employ a classification module as a verification gate for the initial detection step. However, we propose a modification compared to previous methods: instead of forwarding the entire image to the model, we classify the bounding boxes created by detection module. We hypothesize that this approach could enhance accuracy, as downsizing the image might result in the loss of features related to potential flames or clouds of smoke. By focusing on the object, we can leverage the detection model's ability to handle large inputs, ensuring that the bounding boxes of suspicious objects maintain sufficient resolution for the classification model. Given the constraints of our proposed lightweight workflow, we are limited to small, mobile-friendly classification models. Despite their size, these models have demonstrated effective performance in fire detection tasks, as can be seen in [41] with EfficientNet. Moreover, as mentioned in previous chapter, some studies have suggested that visual transformers may outperform traditional CNNs in this task. This leads us to consider CNN-transformer hybrid models and MobileVitV2 is our final choice for its wide range model variant, including the tiny variant, namely MobileViTV2-0.5. However, we suspect that most lightweight models, even the older MobileNet model family, would work well in this workflow, and the performance across such model in the task does not deviate much from each other. Hence, we will compare MobileViTV2 with a traditional CNN model for better judgement of our hypothesis.

3.2.3 Stacking-and-voting mechanism

As far as two-stage detection go, it will eventually make mistakes, either false alarms or missed detections. However due to the nature of real-time inference that make incidents can be detect in any window of time while streaming inputs, missed detection rarely happens if not to be non-existent at all with such system, given the

scenario is in the data distribution. However, also due to that characteristics, false alarm happens much more frequently. For example, with such a system that have 99.9% false alarm rate, false alarm will raise once every 1000 inputs. Given the input of 1 frame per second, it will raise 1 false alarm per 1000 second or approximately 17 minutes in average. Such alarm rate will not be operable in practice due to the immense annoyance it brings to end users.

In order to address the problem, we employ a stacking-and-voting system after the verification module. This system continuously queue inference result, with the number limit of results in the queues up (which means when the queue if full, the least recent result gets push out). If half of the queue capacity is positive, it raises an alarm to the end user and empties the queue. As fire incidents are linear events, sacrificing delay time to exchange for exponentially better judgement of the scenario is worth considering. This also mean the mechanism will not affect recall rate of the system if the number are high enough. We will discuss about how the mechanism work mathematically in the Appendices.

3.2.4 Notification System

After the voting step confirming a fire incident, a notification will be sent to end user. To do such action, we use Telegram Bot API, a free service provide by Telegram, and implement it using `telepot` Python module. In order to not interrupt the workflow, the action is push to another thread to be executed. Code snippet for example usage will be included in the Appendices.

3.3 Hardware configuration

3.3.1 Design

	Specification
CPU	Cortex A72 (ARM v8) 64 bit SoC @ 1.8 Ghz
RAM	4GB LPDDR4 SDRAM
Camera	Raspberry Pi Camera Module V3 12MP Wide 120^o
Other	Ethernet, 2.4Ghz/5Ghz Wireless

Table 3.2. Proposed hardware specifications

Figure 3.2. Our proposed hardware setup

As previously mentioned, we have opted for the Raspberry Pi 4 Model B with 4GB of RAM as our device of choice. It has a quad-core ARM Cortex-A72 processor, which can operate up to 1.8 GHz. The device also supports Ethernet and dual-band wireless for internet connection. We also integrate a Raspberry Pi Camera Module

V3 as our input. This camera is capable of outputting 12 MP images and capturing video up to 1080p at 50 FPS. It also includes phase detection autofocus, which allows the input to auto-adjust in out-of-focus scenarios.

We choose this setup mainly because generally, Raspberry Pi 4 is a popular device among IoT projects which make our solution can be adopt to widely. We believe this setup provides a good starting point for future adopters, which we will mention in chapter 6, Future Work section.

3.3.2 Hardware adaptation

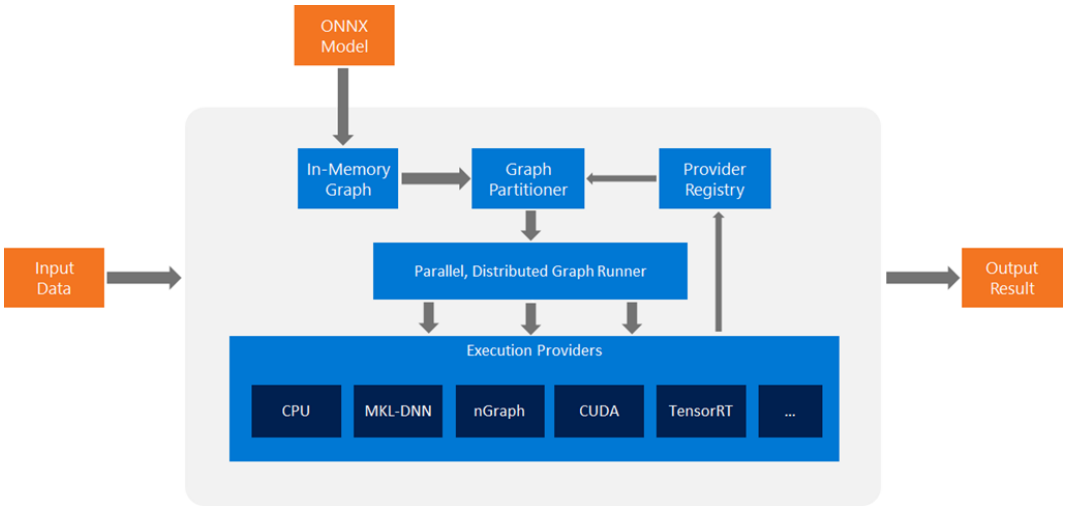

Figure 3.3. ONNX high-level architecture

As deep learning models become more complex and require more computational power, optimization methods and frameworks are becoming increasingly important. One such framework is the Open Neural Network Exchange (ONNX) Runtime [43]. ONNX Runtime is a performance-focused engine for running machine learning models, which supports a wide range of architectures and platforms. ONNX runtime take a exported model and convert it into its in-memory graph representation, therefore reduce the computational operation required. ONNX runtime offers various graph optimizations, therefore further increase the performance.

Although the Cortex-A72 is a capable CPU that can allow daily usage, it is not quite a suitable processor for running deep learning model. To make the proposed system

be able to do inference in real-time, we optimize the models using ONNX runtime. In addition to better inference time, ONNX also provide easier deployment since it is a cross-platform framework, enabling us to train model in different environment settings.

Chapter 4
Experiments

4.1 Datasets description

4.1.1 FASDD dataset

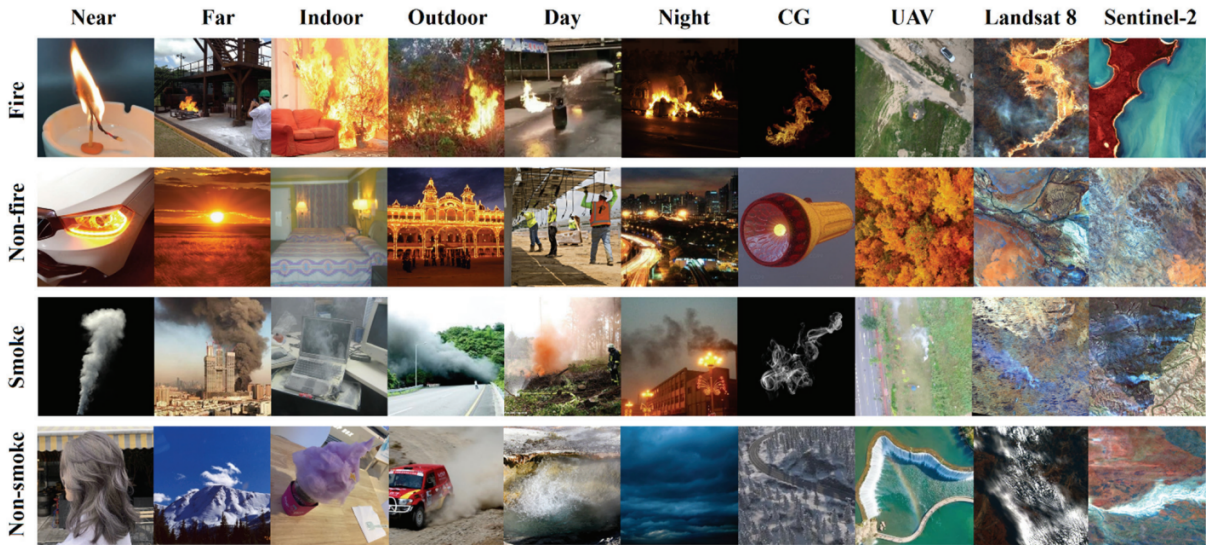

Figure 4.1. FASDD example images, figure are taken from [8]

In this study, we will make use of the Flame and Smoke Detection Dataset (FASDD) [8], an open-source dataset of over 100,000 images for fire and smoke detection. The FASDD is split into two categories: Computer Vision, which contains RGB images from various sources, and Remote Sensing, which contains images from sensor cameras. This dataset has been claimed to achieve the highest heterogeneity and the most significant difference in feature distribution. It can be used for both tasks, image classification and object detection.

Our focus is on the Computer Vision part of the FASDD, as our target is to detect with regular RGB input. This part consists of 95,314 images with 56,115 positive samples, which contain 73,297 flame objects and 53,080 smoke objects, and 39,199 negative samples. The images range from various aspect ratios (1:6.6 to 1:0.18) and different sizes (78 to 10,600 for width, 68 to 8,858 for height).

The FASDD_CV, the computer vision subdataset of FASDD, is meticulously curated, ensuring a wide variety of fire and smoke instances across different individuals, lighting conditions, and angles. This diversity is crucial for training robust models that can generalize well to unseen data. Moreover, the FASDD_CV is annotated with detailed semantic descriptions of each fire and smoke instance, providing rich contextual information that can be leveraged during model training. These annotations not only include the type of instance but also its intensity and temporal dynamics, offering a comprehensive understanding of the fire and smoke instances in the dataset.

4.1.2 Self-recorded dataset

Figure 4.2. Self-recorded dataset samples with annotations

To make the model suitable for practical purposes, we also create a dataset which consists of images recorded from our camera (i.e the Raspberry Pi Camera Module 3). The dataset consists of 1385 images, with 763 images for training, 358 for validation and 262 for testing. There are 700 negative samples, 685 positive samples, which contains 962 annotations of 713 flame objects and 249 smoke instances. All the training images are being preprocessed, with static crop and contrast stretching, and augmented, with brightness change (between -15% and +15%) and noise addition (up to 0.61% of pixel).

4.2 Experiment design

In this section, we describe the experimental design and methods used to assess the performance of our proposed fire detection system on a Raspberry Pi 4 Model B. We will also compare our model selection with another option for each neural network component. Namely, we will evaluate YOLOv5n6 and its anchor-free version, YOLOv5n6u, for detection module, and MobileVitV2 and a conventional CNN model for mobile device, namely MobileNetV3 (large variant) [44]. After that we will combine each of every results yielded from the individual assessments and further assess the performance of each combination.

We also intend to further assess the importance of our self-recorded dataset in this method. Therefore, we will also detail how we attempt to evaluate with the training strategy for verification/classification models.

4.2.1 Detection Model

We use the original training flow for both YOLOv5n6 (with a forked YOLOv5 repository from GitHub) and YOLOv5n6u (with the `ultralytics` library). For both models, we utilize their pretrained weights on the COCO datasets provided by Ultralytics and fine-tune them on the FASDD dataset for 50 epochs. In this process, we use images that have been resized such that one dimension measures 640 pixels. The other dimension can be any size, but it must not exceed 640 pixels. This specific image size is chosen to balance between computational efficiency and the level of detail necessary for the model to perform well. Following this, we conduct single class training for both of our models on our self-recorded datasets. We trained both models for 30 epochs. As for the optimizer, we apply Stochastic Gradient Descent (SGD) for YOLOv5n6 training and the auto optimizer, which adapts to the training process (AdamW for the first 10000 iterations, SGD for all later iterations), for YOLOv5n6u training. Both also use linear learning rate scheduler with warmup. Table 4.1 shows the hyperparameter settings for both of our models.

Hyperparameter	Model 1 (YOLOv5n6u)	Model 2 (YOLOv5n6)	Description
lr0	0.01	0.01	Initial learning rate.
lrf	0.01	0.01	Final learning rate (fraction of lr0).
momentum	0.937	0.937	Momentum
weight_decay	0.0005	0.0005	Adam weight decay
warmup_epochs	3.0	3.0	Number of epochs for the learning rate warmup.
warmup_momentum	0.8	0.8	Initial momentum during the warmup phase.
warmup_bias_lr	0.1	0.1	Initial bias learning rate during the warmup phase.
box	7.5	0.05	Box loss gain.
cls	0.5	0.5	Classification loss gain.
cls_pw	-	1.0	Classification BCELoss positive weight.
obj	-	1.0	Object loss gain (scale with pixels).
obj_pw	-	1.0	Object BCELoss positive weight.
iou_t	-	0.2	Intersection over Union training threshold.
anchor_t	-	4.0	Anchor-multiple threshold.
fl_gamma	-	0.0	Focal loss gamma.
hsv_h	0.015	0.015	Image HSV-Hue augmentation (fraction).
hsv_s	0.7	0.7	Image HSV-Saturation augmentation (fraction).
hsv_v	0.4	0.4	Image HSV-Value augmentation (fraction).
degrees	0.0	0.0	Image rotation (+/- deg).
translate	0.1	0.1	Image translation (+/- fraction).
scale	0.5	0.5	Image scale (+/- gain).
shear	0.0	0.0	Image shear (+/- deg).
perspective	0.0	0.0	Image perspective (+/- fraction), range 0-0.001.
flipud	0.0	0.0	Image flip up-down (probability).
fliplr	0.5	0.5	Image flip left-right (probability).
mosaic	1.0	1.0	Image mosaic (probability).
mixup	0.0	0.0	Image mixup (probability).
copy_paste	0.0	0.0	Segment copy-paste (probability).
closing_mosaic	0	-	Number of last epochs to disable mosaic augmentation.

Table 4.1. Hyperparamaters for training detection models.

4.2.2 Classification Model

Hyperparamter	Value
Batch size	100
Training image size	224
Optimizer	Adam
Initial learning rate (lr0)	0.001
Final learning rate scaling	0.01
Decay factor	0.000005
Label smoothing	0.1

Table 4.2. Hyperparameter for classification training

As for classification/verification models, we train them on both datasets separately and benchmark result of 4 outcome weights for MobileNetV3 and MobileVitV2. We use the ImageNet1k pretrained weights for as our training starting point for both models. We also crop all objects from both dataset, since our verification module only classifies detected objects, and resize them to 224x224 resolution (for ”neither-fire-or-smoke” class, we take image without fire or smoke as the data). For models train with FASDD, we train 30 epochs with batch size 100. For models train with practical data, we train 20 epochs with batch size 100. All training process employ Adam as optimizer with linear learning rate scheduler. Table 4.2 shows the hyperparameters settings for both of our models.

4.3 Metrics

4.3.1 Detection model evaluation

In the process of evaluating object detection models, we adhere to the established standards within the field. Specifically, we utilize the mAP metrics, which is calculated at two different Intersection over Union (IOU) thresholds: 0.5 and 0.5-0.95. The mAP at an IOU of 0.5, often denoted as mAP@0.5, is a widely used metric in object detection tasks. It measures the model’s performance at a relatively lenient

IOU threshold. In other words, a prediction is considered a 'match' if it has an IOU of 0.5 or higher with the ground truth bounding box. This metric is particularly useful for applications where approximate localization is sufficient. On the other hand, mAP@0.5:0.95 is a more stringent metric. It calculates the mAP over a range of IOU thresholds from 0.5 to 0.95 with a step size of 0.05. This metric provides a more comprehensive evaluation of the model's performance across different levels of localization precision. It is especially important in scenarios where precise object localization is critical. The use of these two metrics allows us to comprehensively evaluate the model's capability to accurately identify objects at different levels of precision in localization, thus offering a thorough understanding of its overall effectiveness. For better insight about the metrics, we will provide a detailed description on Appendices. Additionally, we conduct benchmarking of inference speed utilizing the Raspberry Pi 4 platform. This benchmarking process involves deploying optimized models on the Raspberry Pi 4 device, enabling us to gauge the efficiency and practical viability of the models in real-world scenarios. By assessing inference speed on the proposed device, we aim to provide a deep understanding of the performance capabilities of both the detection models on the hardware platform.

4.3.2 Classification model evaluation

In the assessment of classification models, we adopt precision and recall as the primary metrics. Precision denotes the ratio of correctly identified positive instances to the total predicted positive instances, providing insight into the model's accuracy in classifying positive cases. Recall, on the other hand, signifies the ratio of correctly identified positive instances to the total actual positive instances, elucidating the model's ability to capture all positive cases within the datasets. These metrics are particularly relevant in the context of a verification module within a fire detection system, as they reflect the model's capacity to accurately identify and verify instances

of fire presence. The formulas of Precision and Recall are denoted as:

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

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

Furthermore, alongside precision and recall, we also conduct benchmarking of inference time for both models. This evaluation aspect aims to encompass the efficiency of the models in real-time scenarios, considering the practical implications of their deployment within the broader system framework. By assessing inference time alongside precision and recall metrics, we strive to provide a comprehensive understanding of the classification models' performance, facilitating informed decisions regarding their suitability for integration into the fire detection system.

4.3.3 Combined workflow evaluation

In the evaluation of the combined detection workflow within the fire detection system, we also adopt precision and recall as pivotal metrics. Given the critical nature of fire detection systems, where timely and accurate detection is paramount, precision and recall serve as fundamental indicators of system performance. The interplay between these metrics provides insights into the system's reliability in raising alarms promptly and accurately during fire incidents. We also adopt False Negative Rate (FNR) and False Alarm Rate (FAR), both are vital metrics for fire detection. The False Alarm Rate refers to the percentage of non-fire instances that are mistakenly classified as fires. On the other hand, the False Negative Rate is the percentage of actual fires that are not correctly identified. Given that fires are relatively rare events, most real-time surveillance footage does not contain any fires. Too many false alarms, in this case, will cause annoyance and frustration for the end users. Therefore, it's es-

sential to keep false alarms to a minimum in fire detection.

$$FAR = \frac{FP}{FP + TN}$$

$$FNR = \frac{FN}{FP + TN}$$

4.4 Training environment

All training is conducted on a Window 11 with Nvidia Geforce RTX 3070 Laptop 8GB GPU, Intel i7-12700H CPU, 32GB RAM. Model is trained and tested on Python 3.9 and Torch 2.1.

Chapter 5
Results and discussion

In this section, we will present the experimental results for the two fundamental components of our system: the detection module and the verification module. For each experiment, we will outline the metrics employed and provide a concise interpretation of our findings. This approach will ensure a comprehensive understanding of the system's performance and efficacy. We also access the performance of each combination of YOLOv5n6 with different verification module settings for the system to affirm our choice of models. After that, we will give our insights about the results of three experiments.

5.1 Detection

Model	**FASDD_CV**		**Self-recorded dataset**		**Inference time (ms)**
	mAP@0.5	mAP@0.5-0.95	mAP@0.5	mAP@0.5-0.95	
YOLOv5n6	86.0%	58.1%	81.9%	43.5%	333
YOLOv5n6u	86.5%	60.8%	84.6%	46.8%	507
YOLOv5x (reference model)	84.1%	-	-	-	-

Table 5.1. Detection model performance benchmark. Model speed is recorded on Raspberry Pi 4 model B 4GB, both models are exported to ONNX format, input image dimension is set to 640. All test are performed on test splits of corresponding data.

Table 5.1 not only provides a comparative analysis of the performance of the two candidate detection models, YOLOv5n6 and YOLOv5n6u, but also benchmarks them against the reference model, YOLOv5x, from [8]. The superior mAP@0.5 scores of our candidate models over YOLOv5x underscore the effectiveness of the additional layer incorporated for detecting extra-large objects. This enhancement has evidently improved the accuracy of the outputs. The experimental results reveal a close competition between YOLOv5n6 and YOLOv5n6u in terms of accuracy. The latter, albeit being the newer model, demonstrates a marginally better performance both dataset. However, when it comes to inference time, YOLOv5n6 clearly outperforms

YOLOv5n6u by being up to 174 ms faster. This significant difference in inference time can be crucial in our system as it allows more results can be stacked within a period, which enhance the system accuracy.

5.2 Classification/Verification

Model	FASDD_CV		Self-recorded data				Inference time (ms)
			FASDD_CV finetuned		Self-recorded data finetuned		
	Precision	Recall	Precision	Recall	Precision	Recall	
MobileNetV3	98.7%	98.6%	97%	98.4%	98.4%	98.1%	95.9
MobileVitV2	98.4%	98.1%	98.6%	99.2%	97.7%	96.7%	122.3

Table 5.2. Classification model performance. Model speed is recorded on Raspberry Pi 4 model B, both are exported to ONNX, input image dimension is set to 224. All test are performed on test splits of corresponding data.

Table 5.2 provide performance benchmark for our two candidate classification models: MobileNetV3 and MobileVitV2. Both model show similar metrics on the FASDD_CV dataset and the self-recorded dataset. However, MobileVitV2 show 27% higher inference time comparing to MobileNetV3.

5.3 Combined Detection Flow

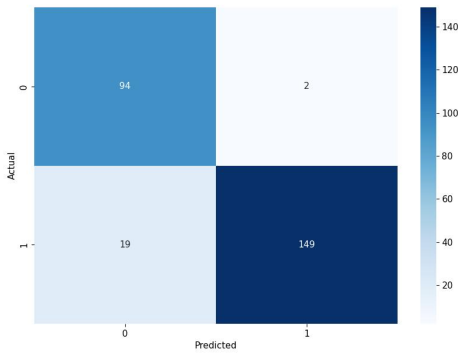

Figure 5.1. Result for combined MobileNetV3 (Finetuned with FASDD) with YOLOv5n6

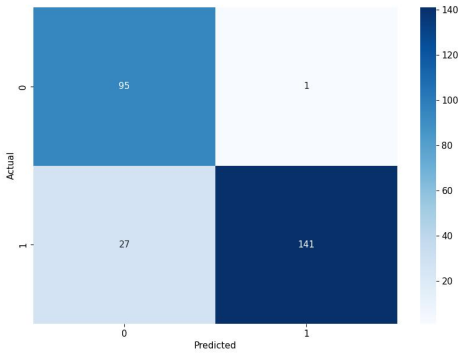

Figure 5.2. Result for combined MobileNetV3 (Finetuned with self-recorded data) with YOLOv5n6

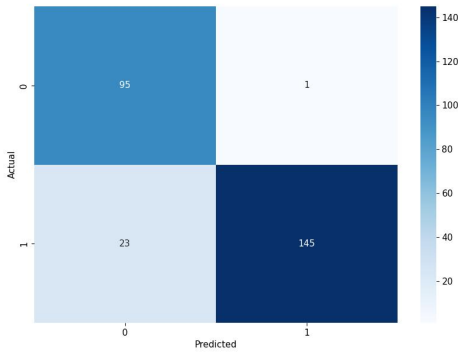

Figure 5.3. Result for combined MobileVitV2 (finetuned with FASDD) with YOLOv5n6

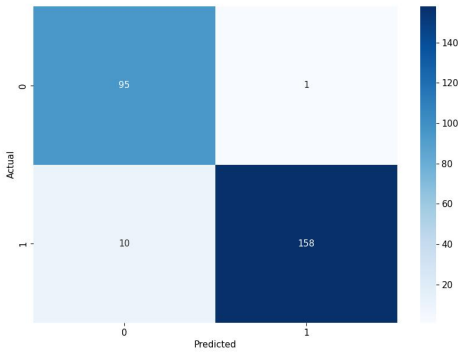

Figure 5.4. Result for combined MobileVitV2 (finetuned with self-recorded data) with YOLOv5n6

Self-recorded MobileVitV2 + YOLOv5n6

Self-recorded MobileNetv3 + YOLOv5n6

Ground truth

Figure 5.5. Comparison two combined systems with the same detection model, different verification model and same verification finetuning data.

Detection model	Verification model	Verification finetuning dataset	Precision	Recall	FAR	FNR
YOLOv5n6	MobileNetV3	FASDD	98.68%	88.69%	2.08%	11.31%
		Self-recorded	99.30%	83.93%	1.04%	16.07%
	MobileVitV2	FASDD	99.32%	86.31%	1.04%	13.69%
		Self-recorded	**99.37%**	**94.05%**	**1.04%**	**5.95%**

Table 5.3. Performance of 4 combination of our system. Test conducted on the test split of self-recorded dataset. Result of our system is highlighted in bold text.

Table 5.3 illustrate the result of 4 combination for our candidate system. The superior performance of the combination that includes MobileVitV2 finetuned by our self-recorded dataset underscores the importance of using a dataset that closely matches the conditions under which the system will be deployed. This combination achieved a high recall rate of 94.05% when combined with YOLOv5n6, indicating a high ability to correctly identify positive instances. On the other hand, combinations involving either MobileNetV3 (fine-tuned on either dataset) or MobileVitV2 fine-tuned by FASDD performed worse, with some even scoring a high FNR of over 16%. This observation leads to two conclusions: firstly, the attention mechanism in MobileVitV2 enhances the verification of YOLOv5n6 detections. The global feature consideration trait of MobileVitV2 is useful to verify large objects (which works well with models that are capable of extra-large object detection i.e YOLOv5-P6), as can be seen in Figure 5.5, as well as distinguish hard-to-see instances. Secondly, the self-recorded dataset plays a crucial role for the system to operate practically in such device settings. The result has fortified our component choice of YOLOv5n6 and MobileVitV2, both in speed and accuracy aspects. The system allows us to operate real-time (1-3 FPS). With Stacking-and-Voting mechanism set at 11 results, theoretically said, the chance of fire detection within first 10 second is more than 99.998%. Moreover, the false alarm rate should be theoretically less than a fraction of billions (approximately 5.6×10^{-10}).

5.4 Discussion

The results of our combined system show a comparable result to previous work. Compared to Xu et al. [41] with similar approach, we manage to replicate close result while omitting the use of the another detection model, which simplify the training process as well as model choice. Also, focusing on some specific regions allow us to train verification model with less data as less background is need to consider, which we believe is a slight advantage compare to the global classifier approach.

Through the execution of three distinct experiments, we have underscored the significance of self-recorded data for superior adaptation to practical applications. The FASDD dataset, despite its large-scale nature and encompassing various fire scenarios, exhibits a deficiency in data that encapsulates fire and smoke instances recorded by our input device, namely the Pi Camera Module 3. This shortfall is primarily due to the limited number of devices that employ the same sensor and software, which lead to the insufficient availability of similar data to gather. This also explained why there is no clear correlation between the classification/verification module result and the combined system result.

As for our dataset, as mentioned in the previous section, the dataset lacks variety due to our constriction in the actual fire incident collection. Consequently, the implementation of a stacking-and-voting mechanism becomes indispensable, playing a pivotal role in mitigating the false alarm rate to near-zero levels. In the same time, however, this mechanism may potentially obstruct the system's ability to detect fire in out-of-distribution scenarios, particularly those not included in the dataset, as the detection rate per frame in such incidents is not as high as expected. Nevertheless, we did try to address this by including different background and light conditions, which partially mitigate the problems. It is worth noting that this issue can be readily resolved if there exists a dedicated facility for creating such images, which allow continuous improvement of the system over time.

Since our decision on the model is heavily based on our hardware setup, we believe

that the choice of model may vary according to other aspects. One such crucial factor is hardware specifications, which may allow for the employment of more or less complex models to adapt for the need of real-time detection.

Chapter 6
Conclusion

6.1 Contributions

This thesis has proposed a deep learning system for real-time visual fire on edge device, particularly the Raspberry Pi 4 model B. The system comprises of two main components: a lightweight detection module (YOLOv5n6), and mobile-friendly verification module (MobileVitV2), as well as a stacking-and-voting mechanism for false alarm management and finally a simple notification API for end user. The thesis also studies closely about the model choice for both modules by individually assess the performance of each module and combines them for more insights about not only model choice but the dataset use for practical purpose. In the process of deciding for lightweight detection model, two candidate models were proposed: YOLOv5n6, which represents the anchor-based detector family, and YOLOv5n6u, which represents the anchor-free detector family. Two models was pretrained with FASDD_CV, a subdataset of large scale fire and smoke dataset, and further finetuned with self-created data, which recorded by our input device - Pi camera module V3 - and annotated by us. The result show slightly better performance from YOLOv5n6u in FASDD_CV and self-recorded dataset, but fall short in term of speed, which come as an deciding factor for the final decision.

As for verification module, two lightweight models were chosen as experimental options for the component: MobileNetV3 and MobileVitV2. Each model was finetuned on two datasets for two separate weights, which produce four model variants in total, and put on multiple tests. Results of the experiment show the similar performance across the board. The experiment, while showing similar performance for each of every model in the same category, does not correspond to immediate similar results in the combined system experiment. In fact, MobileVitV2 finetuned with self-recorded dataset with YOLOv5n6 show the most promising results among all the presented combinations as it has the highest recall rate of 94.05% as well as lowest false alarm

rate of 1.04%. In addition to that, the system is able to operate in real-time with 1-3 FPS. With Stacking-and-Voting mechanism, the system manage to minimize false alarm rate less than fraction of billions, while still be able to output alarm with in the first 11 seconds.

6.2 Future work

There are a few approaches for this work in the future. In terms of enhancing the deep learning aspects of the work, two approaches can be considered. First, the use of alternative input devices such as CCTV cameras could be explored. This could potentially address the issue of the dataset's quality and quantity, making data collection more efficient and less labor-intensive, as the data might have existed and well-annotated. Second, the exploration of advanced system configurations with built-in neural computing capability such as Nvidia Jetson or Google Coral should be beneficial. These devices could enable batch inference and multi-input fire detection, thereby diversifying the visual input and increase the overall detection efficiency.

On the practical side of the system, future adopters might consider scaling the notification system to an independent signaling module. This could be achieved through integration into the IoT ecosystem, which would allow for customization with microcontrollers. This enhancement would increase the system's practicality and responsiveness in real-world applications.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[3] S. Mehta and M. Rastegari, "Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer," 2022.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[5] S. Mehta and M. Rastegari, "Separable self-attention for mobile vision transformers," *Transactions on Machine Learning Research*, 2023. [Online]. Available: https://openreview.net/forum?id=tBl4yBEjKi

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.

[7] A. Jadon, M. Omama, A. Varshney, M. S. Ansari, and R. Sharma, "Firenet: A specialized lightweight fire smoke detection model for real-time iot applications," 2019.

[8] M. Wang, L. Jiang, P. Yue, D. Yu, and T. Tuo, "Fasdd: An open-access 100,000-level flame and smoke detection dataset for deep learning in fire detection," *Earth System Science Data Discussions*, pp. 1–26, 2023.

[9] Ultralytics, "Yolov5 object detection architecture," accessed: 2024-02-28. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/architecture_description/

[10] "Press release on 2023 fire prevention and rescue in vietnam," 2024. [Online]. Available: http://canhsatpccc.gov.vn/ArticlesDetail/tabid/193/cateid/1172/id/22312/language/vi-VN/Default.aspx

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[12] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[13] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[14] H. Bao, L. Dong, S. Piao, and F. Wei, "Beit: Bert pre-training of image transformers," *arXiv preprint arXiv:2106.08254*, 2021.

[15] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," 2021.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*. Springer International Publishing, 2016, p. 21–37. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[19] Z. Tian, C. Shen, H. Chen, and T. He, "Fcos: Fully convolutional one-stage object detection," 2019.

[20] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[21] R. Girshick, "Fast r-cnn," 2015.

[22] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," 2016.

[23] ——, "Yolov3: An incremental improvement," 2018.

[24] G. Jocher, "Ultralytics yolov5," 2020. [Online]. Available: https://github.com/ultralytics/yolov5

[25] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, "Cspnet: A new backbone that can enhance learning capability of cnn," 2019.

[26] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[27] K. Muhammad, J. Ahmad, Z. Lv, P. Bellavista, P. Yang, and S. W. Baik, "Efficient deep cnn-based fire detection and localization in video surveillance applications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 7, pp. 1419–1434, 2018.

[28] S. Majid, F. Alenezi, S. Masood, M. Ahmad, E. S. Gündüz, and K. Polat, "Attention based cnn model for fire detection and localization in real-world images," *Expert Systems with Applications*, vol. 189, p. 116114, 2022.

[29] K. Zhang, B. Wang, X. Tong, and K. Liu, "Fire detection using vision transformer on power plant," *Energy Reports*, vol. 8, pp. 657–664, 2022, 2022 The 4th International Conference on Clean Energy and Electrical Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S235248472201071X

[30] M. Shahid and K.-l. Hua, "Fire detection using transformer network," in *Proceedings of the 2021 International Conference on Multimedia Retrieval*, ser. ICMR '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 627–630. [Online]. Available: https://doi.org/10.1145/3460426.3463665

[31] D. Y. Chino, L. P. Avalhais, J. F. Rodrigues, and A. J. Traina, "Bowfire: detection of fire in still images by integrating pixel color and texture analysis," in *2015 28th SIBGRAPI conference on graphics, patterns and images*. IEEE, 2015, pp. 95–102.

[32] P. Foggia, A. Saggese, and M. Vento, "Real-time fire detection for video surveillance applications using a combination of experts based on color, shape and motion," *IEEE Transactions on Circuits and Systems for Video Technology*, 2015.

[33] P. Barmpoutis, K. Dimitropoulos, K. Kaza, and N. Grammalidis, "Fire detection from images using faster r-cnn and multidimensional texture analysis," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 8301–8305.

[34] Q.-x. Zhang, G.-h. Lin, Y.-m. Zhang, G. Xu, and J.-j. Wang, "Wildland forest fire smoke detection based on faster r-cnn using synthetic smoke images," *Procedia engineering*, vol. 211, pp. 441–446, 2018.

[35] J. Miao, G. Zhao, Y. Gao, and Y. Wen, "Fire detection algorithm based on improved yolov5," in *2021 International Conference on Control, Automation and Information Sciences (ICCAIS)*, 2021, pp. 776–781.

[36] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "Ghostnet: More features from cheap operations," 2020.

[37] G. Chen, H. Zhou, Z. Li, Y. Gao, D. Bai, R. Xu, and H. Lin, "Multi-scale forest fire recognition model based on improved yolov5s," *Forests*, vol. 14, no. 2, 2023. [Online]. Available: https://www.mdpi.com/1999-4907/14/2/315

[38] Y. Li, W. Zhang, Y. Liu, R. Jing, and C. Liu, "An efficient fire and smoke detection algorithm based on an end-to-end structured network," *Engineering Applications of Artificial Intelligence*, vol. 116, p. 105492, 2022.

[39] Y. Liu, Z. Shao, Y. Teng, and N. Hoffmann, "Nam: Normalization-based attention module," *arXiv preprint arXiv:2111.12419*, 2021.

[40] C. Bahhar, A. Ksibi, M. Ayadi, M. M. Jamjoom, Z. Ullah, B. O. Soufiene, and H. Sakli, "Wildfire and smoke detection using staged yolo model and ensemble cnn," *Electronics*, vol. 12, no. 1, 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/1/228

[41] R. Xu, H. Lin, K. Lu, L. Cao, and Y. Liu, "A forest fire detection system based on ensemble learning," *Forests*, vol. 12, no. 2, 2021. [Online]. Available: https://www.mdpi.com/1999-4907/12/2/217

[42] A. Nguyen, H. Nguyen, V. Tran, H. X. Pham, and J. Pestana, "A visual real-time fire detection using single shot multibox detector for uav-based fire surveillance," in *2020 IEEE Eighth International Conference on Communications and Electronics (ICCE)*. IEEE, 2021, pp. 338–343.

[43] O. R. developers, "Onnx runtime," https://onnxruntime.ai/, 2021, version: 1.17.0.

[44] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," 2019.

[45] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

APPENDICES

A Stacking-and-Voting Mechanism Analysis

In this chapter, we will discuss the mathematical ideas behind out stacking-and-voting mechanism to reduce false alarms.

A.1 Preliminaries

The binomial distribution is a fundamental concept in probability theory that describes the likelihood of observing a specific number of successes in a fixed number of independent trials, each having only two possible outcomes: success and failure. The distribution is characterized by 2 parameters: n, the number of independent trials, and p, the probabilities of success in each trial. The probability mass function (PMF) of the distribution, or the probability of having exactly k successes in n trials, is denoted as:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

where:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

The cumulative distribution function (CDF) of the binomial distribution represent of having at most k successes in n trials is denoted as:

$$P(X \leq k) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i (1-p)^{n-i},$$

which is the sum of PMF from 0 to X. We will utilize this concept in the following examples to understand how the stacking-and-voting mechanism leverages the binomial distribution to reduce false alarms.

A.2 Example

This example, similar to the one previously presented in chapter 3, Stacking-and-Voting mechanism section, analyzes a fire detection system exhibiting a 2% false alarm rate per frame (we pick the number that is close to our final decision for the model). The system accumulates information from 10 continuous frames, triggering an alarm solely if 5 or more frames indicate the presence of fire.

We will employ the binomial distribution to calculate the false positive rate (probability of a false alarm in a normal situation). Solution step:

1. Parameter definition:
    - $n = 10$ (number of frames)
    - $p = 0.02$ (probability of false alarm per frame)
    - $k = 4$ (maximum permissible false alarms before raising the alarm)
2. Calculate the probability of having at most 4 false alarms:

$$P(X \leq 4) = \sum_{i=0}^{\lfloor 4 \rfloor} \binom{10}{i} 0.02^i (1 - 0.02)^{10-i} \approx 0.99999926$$

3. Calculate the False Alarm Rate (FAR):

$$FAR = 1 - P(X \leq 4) \approx 7.4e - 07$$

This indicate, with a combined system that achieve 2% FAR per frame, our stacking-and-voting system has effectively reduced the actual false alarm rate down by approximately 27000 times.

A.3 Conclusion

In this chapter, we have show the theoretical idea for our stacking-and-voting mechanism on reducing false alarm rate for the system. In high level, by requiring

a consensus among multiple frames before triggering an alarm, the system effectively filters out random noise and anomalies that could lead to false positives.

B YOLOv5, YOLOv5-P6, YOLOv8 architecture

B.1 YOLOv5 architecture

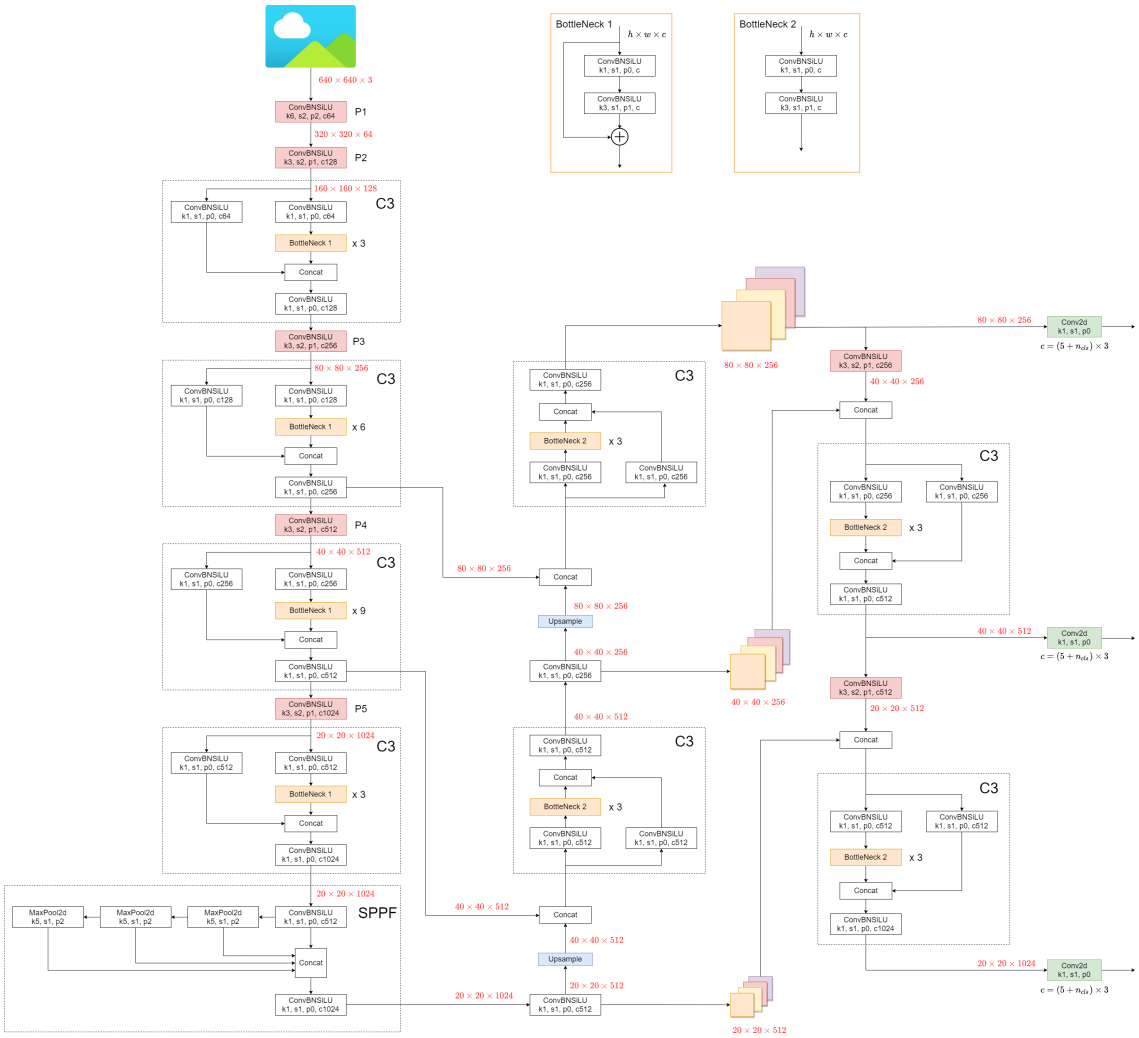

Figure B.1. YOLOv5l architecture [9]

B.2 YOLOv5-P6 architecture

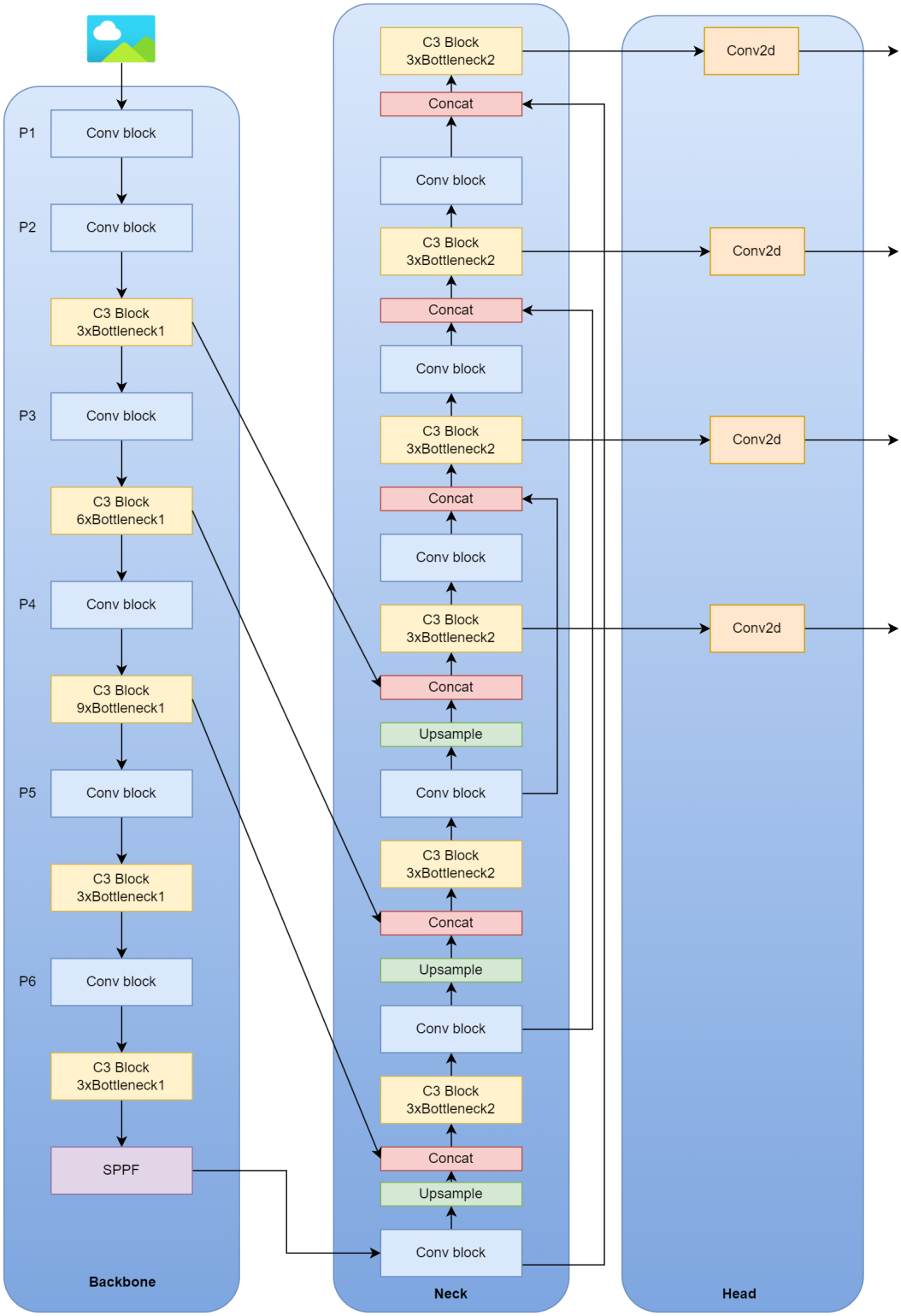

Figure B.2. YOLOv5-P6 architecture, the block used in this diagram is the same as the YOLOv5 with additional P6 output

B.3 YOLOv8 architecture

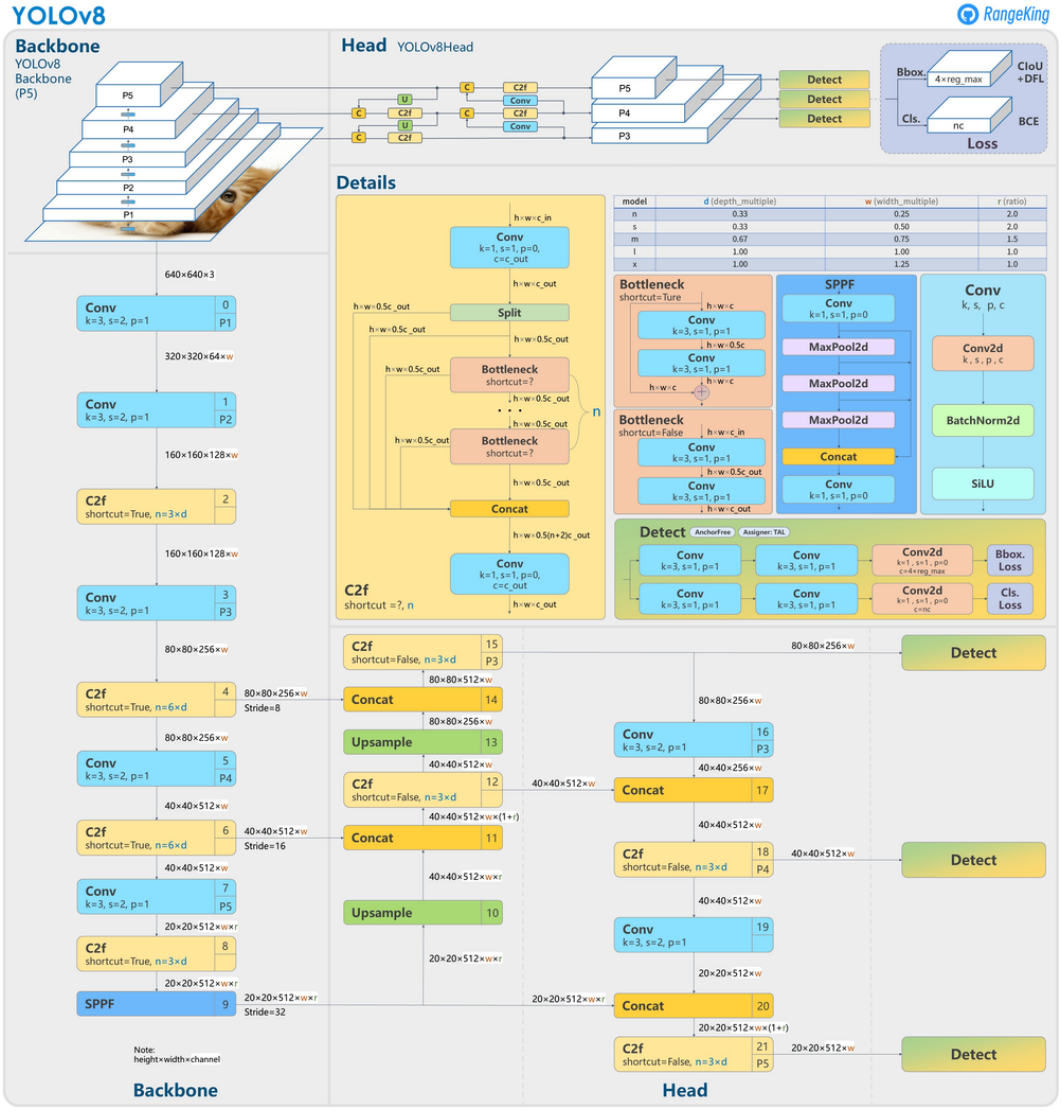

Figure B.3. YOLOv8 architecture by GitHub user RangeKing

C Code snippet for Telegram bot API

C.1 Introduction

`telepot` is a Python framework for the Telegram Bot API. It provides a simple way to develop applications that can interact with the Telegram messaging platform. With `telepot`, python developer can create bots that can receive messages, send messages, edit messages, and handle inline queries.

C.2 Basic usage

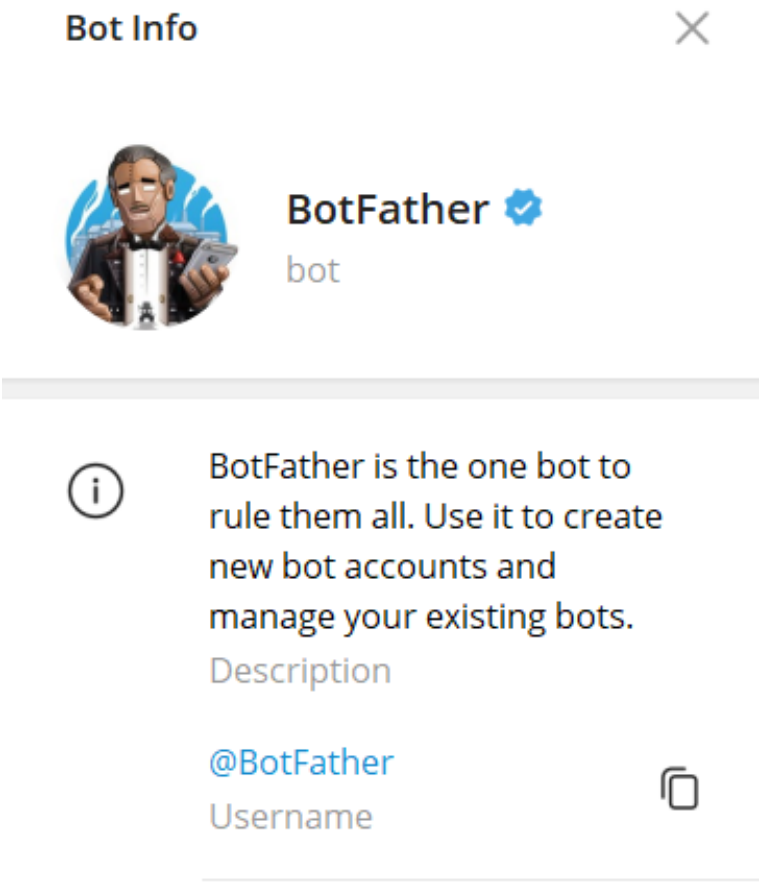

Figure C.1. BotFather informations

This snippet is for simple bot usage. The bot token is taken when creating the bot using the `BotFather` bot by telegram, the chat id can be retrieve.

```
import telepot

def send_notification(bot_token, chat_id, message):
    bot = telepot.Bot(bot_token)
    bot.sendMessage(chat_id, message)

# Usage
bot_token = 'YOUR_BOT_TOKEN'
chat_id = 'YOUR_CHAT_ID'
message = 'Hello, World!'
send_notification(bot_token, chat_id, message)
```

The `chat_id` parameters can be retrieve with this snippet. Note that user has to send message to the bot for the `getUpdates` method to work.

```
import telepot

def get_chat_id(bot_token):
    bot = telepot.Bot(bot_token)
    updates = bot.getUpdates()
    if updates:
        chat_id = updates[-1]['message']['chat']['id']
        return chat_id
    else:
        return None

# Usage
bot_token = 'YOUR_BOT_TOKEN'
chat_id = get_chat_id(bot_token)
```

```
if chat_id:
    print(f"The chat ID is: {chat_id}")
else:
    print("Send a message to the bot and try again.")
```

C.3 Use with python multi-threading

In our implementation of the workflow on our Raspberry Pi, we utilize multi-threading to not interrupt the main detection flow. This is an snippet for that implementation.

```
import telepot
import threading

def send_notification(bot_token, chat_id, message):
    bot = telepot.Bot(bot_token)
    bot.sendMessage(chat_id, message)

def threaded_notification(bot_token, chat_ids, message):
    threads = []
    for chat_id in chat_ids:
        thread = threading.Thread(
            target=send_notification,
            args=(bot_token, chat_id, message),
        )
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()
```

```
bot_token = 'YOUR_BOT_TOKEN'
chat_ids = ['CHAT_ID_1', 'CHAT_ID_2', 'CHAT_ID_3']
message = 'Hello, World!'
threaded_notification(bot_token, chat_ids, message)
```

D Mean Average Precision in Object Detection

D.1 Precision and Recall

Precision and recall are fundamental metrics in object detection. Precision is the ratio of correctly predicted positive observations to the total predicted positives. The formula for precision is

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

where TP is the number of true positives and FP is the number of false positives. Recall is the ratio of correctly predicted positive observations to all observations in actual class. The formula for recall is

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

where FN is the number of false negatives.

D.2 Intersection Over Union (IoU)

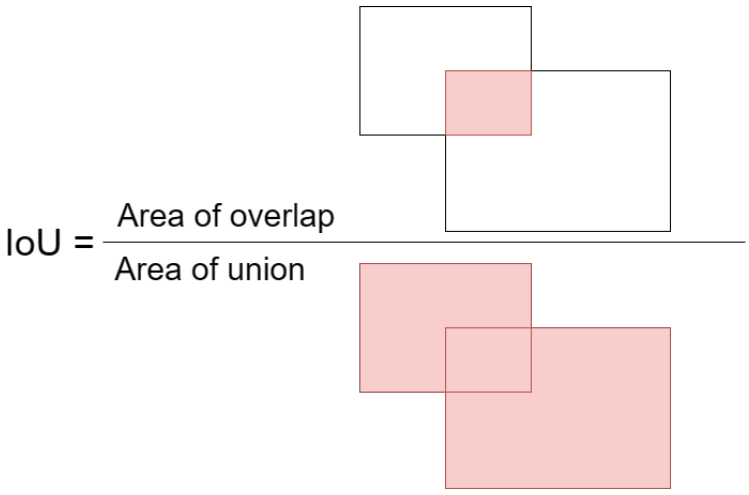

Figure D.1. Intersection over Union

Intersection over Union (IoU) is a crucial evaluation metric in object detection, which quantifies the degree of overlap between the predicted bounding box and the

ground truth bounding box. It is defined as

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

The metrics is use for as a threshold for determining the quality of a prediction, where 0 means no overlap between the predicted bounding box and the ground truth, while 1 equal the complete overlap of predicted bounding box on ground-truth bounding box.

D.3 Mean Average Precision (mAP)

mAP is the average of the maximum precisions at different recall values. It is calculated as follows:

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} \frac{TP(c)}{TP(c) + FP(c)}$$

where $TP(c)$ is the number of true positives and $FP(c)$ is the number of false positives for class c. In object detection, mAP is often associated with IoU threshold to provide better insight on the performance of object detection model. mAP@0.5 is the mAP calculated at an Intersection over Union (IoU) threshold of 0.5. If the IoU of a predicted bounding box with the ground truth bounding box is greater than 0.5, it is considered a true positive. mAP@0.5-0.95, the primary challenge metrics for MSCOCO dataset [45], is the average mAP over different IoU thresholds, from 0.5 to 0.95, with a step size of 0.05. It is calculated as follows:

$$mAP@0.5:0.95 = \frac{1}{10} \sum_{i=[0.5:0.05:0.95]} mAP@i$$