

**IOT BASED DISASTER DETECTION  
USING MULTI-OUTPUT CLASSIFICATION**

**WONG YI JIE**

**UNIVERSITI TUNKU ABDUL RAHMAN**

**IOT BASED DISASTER DETECTION  
USING MULTI-OUTPUT CLASSIFICATION**

**WONG YI JIE**

**A project report submitted in partial fulfilment of the  
requirements for the award of Bachelor of Engineering  
(Honours) Biomedical Engineering**

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

**April 2022**

**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 : *yj wong*

Name : Wong Yi Jie


ID No. : 1703758

Date : 24<sup>th</sup> April 2022

**APPROVAL FOR SUBMISSION**

I certify that this project report entitled “**IOT BASED DISASTER DETECTION USING MULTI-OUTPUT CLASSIFICATION**” was prepared by **WONG YI JIE** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Biomedical Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature :   
\_\_\_\_\_

Supervisor : Dr. Tham Mau Luen

Date : 24<sup>th</sup> April 2022

Signature :   
\_\_\_\_\_

Co-Supervisor : Dr. Kwan Ban Hoe

Date : 24<sup>th</sup> April 2022

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2022, Wong Yi Jie. All right reserved.

## 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 and co-supervisor, Ir. Ts. Dr. Tham Mau Luen and Dr. Kwan Ban Hoe, for their invaluable advice, guidance and 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 full support in this project. I appreciate the encouragement given by my supervisors, loving parents, and friends.

## ABSTRACT

Deep learning (DL) can learn useful insights from disaster events and detect the number of victims and activity of disaster for an efficient and timely rescue operation. Monitoring these disasters at large-scale coverage, however, requires a plethora of Internet of things (IoT) devices, which often have limited processing capacity. Furthermore, centralized training which demands the collection of multiple local datasets contained in each IoT node, is impractical for resource-constrained IoT networks. To realize the full potential of IoT, this project proposes a holistic IoT-based disaster detection framework which optimizes the performance at both training and inference levels. The starting point is the design of a YOLO-based multi-task model that could jointly perform disaster classification and victim detection. This eliminates the straightforward approach of running multiple individual DL models. Next, federated learning (FL) in combination with active learning (AL), is leveraged to enable collaborative training of a global model among IoT devices, without sharing the bandwidth-hungry data. Lastly, at the inference stage, Open Visual Inference and Neural Network Optimization (OpenVINO) toolkit is utilized to optimize the trained model for real-time implementation. Experiment results show that the multi-task model can achieve up to 0.7933 F1 score and 0.6938 average precision (AP) for disaster classification and victim detection tasks, respectively. For the OpenVINO model, the frames per second (FPS) is 16.46, resulting in more than doubled the speed achieved by the original model before model optimization and compression.

## TABLE OF CONTENTS

<b>DECLARATION</b>	<b>i</b>
<b>APPROVAL FOR SUBMISSION</b>	<b>ii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>iv</b>
<b>ABSTRACT</b>	<b>v</b>
<b>TABLE OF CONTENTS</b>	<b>vi</b>
<b>LIST OF TABLES</b>	<b>viii</b>
<b>LIST OF FIGURES</b>	<b>ix</b>
<b>LIST OF SYMBOLS / ABBREVIATIONS</b>	<b>xi</b>
<b>CHAPTER</b>	
<b>INTRODUCTION</b>	<b>1</b>
1.1 General Introduction	1
1.2 Importance of the Study	2
1.3 Problem Statement	2
1.4 Aim and Objectives	3
1.5 Scope and Limitation of the Study	3
1.6 Contribution of the Study	4
1.7 Outline of the Report	4
<b>LITERATURE REVIEW</b>	<b>6</b>
2.1 Introduction	6
2.2 Multi-Task Learning in Deep Learning	6
2.2.1 General Approaches of Multi-Task Learning in Deep Learning	6
2.2.2 Why does Multi-Task Learning work?	8
2.3 Disaster Classification using Deep Learning	9
2.4 Multi-Task Learning Involving Object Detection	16
2.5 Edge Computing	23
2.6 Federated Learning	24
2.7 Active Learning	25
2.8 Summary	28
<b>METHODOLOGY AND WORK PLAN</b>	<b>29</b>



3.1	Introduction	29
3.2	Work Plan	29
3.3	Tools to Use	30
	3.3.1 Training Platform	30
	3.3.2 Deep Learning Framework	31
	3.3.3 Federated Learning Framework	33
3.4	Data Preparation	34
3.5	Model Architecture	35
	3.5.1 Object Detector Selection for Victim Detection	35
	3.5.2 Head Model for Disaster Classification	39
	3.5.3 The Architecture of the Unified MTL Model	43
3.6	Training the MTL Model	44
	3.6.1 Training the Victim Detection Head Model	44
	3.6.2 Training the Disaster Classification Head Model	48
3.7	Conversion of the Multi-Task Model into Intermediate Representation (IR) to load in Inference Engine (IE)	52
3.8	Summary	52
<b>RESULTS AND DISCUSSION</b>		<b>54</b>
4.1	Introduction	54
4.2	Victim Detection	54
4.3	Disaster Classification	56
4.4	Model Optimization using OpenVINO Toolkit	61
4.5	Summary	64
<b>CONCLUSIONS AND RECOMMENDATIONS</b>		<b>65</b>
5.1	Conclusions	65
5.2	Recommendations for future work	65
<b>REFERENCES</b>		<b>66</b>
<b>LIST OF PUBLICATIONS</b>		<b>72</b>

## LIST OF TABLES

Table 2.1:	Why does Multi-Task Learning Work (Ruder, 2017)?	8
Table 2.2:	Research Work Related to Disaster Classification.	9
Table 2.3:	Multi-Task Model that adds Additional Head Model(s) for Other Task(s).	16
Table 2.4:	Multi-Task Model that Makes Minimal Modification to the Original Object Detector Model.	19
Table 2.5:	Object Detection Model that Implements MTL to Improve Object Detection.	21
Table 3.1:	Comparison between Colab and the Workstation Used in This Project.	31
Table 3.2:	Comparison between TensorFlow and PyTorch (Boesch, 2021).	32
Table 3.3:	Class Distribution for the Training, Validation and Test for Disaster Type Sub-Dataset.	35
Table 4.1:	Performance of the CL-Trained Head Model with the Provided Benchmarks. Bolded Values indicate the Best Score in the Particular Metrics, while Gray-Shaded Values are the Score for the Proposed Solution.	57
Table 4.2:	Comparison of the Performance of the Disaster Classification Head Models trained via CL, FL, and AL-FL. Methods labelled with an asterisk (*) are trained using 3 FL clients.	58
Table 4.3:	Examples of Images in the Crisis Benchmark Dataset that Confused the Model.	60
Table 4.4:	Comparison of Model's Inference Speed (FPS) Before and After Optimization on Different Processing Units.	62
Table 4.5:	Comparison of the Accuracy of the Victim Detection Head Model Before and After Compression.	63
Table 4.6:	Comparison of the Accuracy of the Disaster Classification Head Model Before and After Compression.	63

## LIST OF FIGURES

Figure 2.1:	Hard Parameter Sharing for Multi-Task Learning (Ruder, 2017).	7
Figure 2.2:	Soft Parameter Sharing for Multi-Task Learning (Ruder, 2017).	8
Figure 2.3:	Multi-Task Model in Research Work 3 (Wen et al., 2020) in <b>Table 2.3</b> . Head Models are added for each Additional Task.	18
Figure 2.4:	Multi-Task Model in Research Work 4 (Zhang et al., 2021) in <b>Table 2.3</b> . Hard Parameter Sharing is used in the form of Shared Backbone, while Soft Parameter Sharing is applied to the two Head Models in the form of TAM.	19
Figure 2.5:	Multi-Task Model in Research Work 2 (Chen et al., 2018) in <b>Table 2.4</b> . Instead of Modifying the SSD Model, the Authors Introduced Variation on the Original Classes/Labels for Additional Tasks. Originally, there were M numbers of Labels. N Variations are Introduced for Each Label, resulting in M x N Labels.	21
Figure 2.6:	Multi-Task Model in Research Work 1 (Lee, Na and Kim, 2019) in <b>Table 2.5</b> . Additional Tasks are Auxiliary Tasks, which are meant to Improve the Learning of the Main Task.	22
Figure 2.7:	The Flowchart of a typical FL Pipeline. Component 1 represents the Global Model. Component 2 represents the Copy of the Global Model in each client, each trained on the Local Dataset. Finally, Component 3 is the Aggregator Function to Merge all the Local Models into one Global Model for the next round of FL.	25
Figure 2.8:	Decision Boundary of a Given Dataset.	26
Figure 2.9:	Red-Circled Data Points are Trivial for Model Training.	26
Figure 2.10:	Red-Circled Data Points are the Informative Data Points, which are essential for Model Training.	27
Figure 3.1:	Gantt Chart for Final Year Project Part 1.	29
Figure 3.2:	Gantt Chart for Final Year Project Part 2.	30
Figure 3.3:	Illustration of Bounding Boxes Removal via Non-Max Suppression (Redmon et al., 2015).	39

Figure 3.4:	The Modified Architecture of YOLOv3.	39
Figure 3.5:	Illustration of the Working Principle of Regular Convolution. The Spatial Information in the $D_k \times D_k \times M$ Region is Convolved at Once. Since the Output Has $N$ Channels, the Region will be Convolved $N$ Times. The Entire Process will be Repeated $D_f \times D_f$ Times, since the Output has Spatial Width of $D_f \times D_f$ .	40
Figure 3.6:	Illustration of Depthwise Separable Convolution. The Spatial Information in $D_k \times D_k \times M$ Region will be Convolved Separately $M$ Times, Resulting in the $1 \times 1 \times M$ Region in the Intermediate Output. The $1 \times 1 \times M$ Region will be Convolved at Once via Regular Convolution. Since the Output has $N$ Channels, the $1 \times 1 \times M$ Region will be Convolved $N$ Times. The Entire Process will be Repeated $D_f \times D_f$ Times, because the Output has Spatial Width of $D_f \times D_f$ .	41
Figure 3.7:	The Architecture for the Disaster Classification Head Model.	43
Figure 3.8:	The Architecture of the Unified MTL Model for Disaster Classification and Victim Detection.	43
Figure 3.9:	Illustration of Data-Parallelism.	45
Figure 3.10:	Illustration of Gradient Accumulation.	45
Figure 3.11:	Pseudocode to Train the Victim Detection Head Models using Centralised Learning.	47
Figure 3.12:	Pseudocode to Train the Victim Detection Head Models using Federated Learning.	48
Figure 3.13:	Pseudocode of the Proposed Active Learning Process.	51
Figure 3.14:	The Methodology of the Proposed System.	53
Figure 4.1:	The Precision-Recall Curve for Victim Detection Head Model.	55
Figure 4.2:	Victim Detection at Different Areas: (a) Flood, (b) Landslide, (c) Earthquake, (d) Hurricane, (e) Fire, and (f) Other Disaster.	56
Figure 4.3:	The Embeddings of the Top 33% Hardest Images in Test Dataset.	59

## LIST OF SYMBOLS / ABBREVIATIONS

AL	Active Learning
AL-FL	Active Learning based Federated Learning
AP	Average Precision
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformer
CL	Centralized Learning
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CSP	Cross Stage Partial Network
DL	Deep Learning
FAIR	Facebook's AI Research Lab
FedAvg	Federated Averaging
FL	Federated Learning
FPN	Feature Pyramid network
FPGA	Field Programmable Gate Array
FPS	Frame Per Second
GB	Gigabytes
GPU	Graphics Processing Units
IE	Inference Engine
IoT	Internet of Things
IR	Intermediate Representation
INT8	8-bit Integer Type
mAP	Mean Accuracy Precision
MB	Megabytes
MO	Model Optimizer
MPSO	Modified Particle Swarm Optimization
MTL	Multi-Task Learning
NCS-2	Neural Compute Stick
NMS	Non-Max Suppression
OpenFL	Open Federated Learning
OpenVINO	Open Visual Inference and Neural Network Optimization

PA-Net	Path Aggregation Network
R-CNN	Region Based Convolutional Neural Networks
ResNet	Residual Neural Network
SSD	Single Shot Detector
TAM	Task-related Attention Module
TFF	TensorFlow Federated
TPU	Tensor Processing Units
t-SNE	t-distributed Stochastic Neighbour Embedding
VOC	Visual Object Classes
YOLO	You Only Look Once

## CHAPTER 1

### INTRODUCTION

#### 1.1 General Introduction

Natural disasters such as hurricanes and wildfires frequently occur around the world. Every year, natural disasters cause substantial amounts of damages, monetary loss, as well as injuries and deaths. For example, the recent floods in China have killed at least 99 people and caused damages estimated to be 348 million United States dollars (Siqi, 2021). To make things worse, climate change has caused natural disasters to increase in recent years. Thus, efforts must be focused on improving disaster response systems.

The first 72 hours after a disaster are critical for rescuing survivors (United Nations Office for the Coordination of Humanitarian Affairs, 2017). Thus, disaster detection is extremely important so that search and rescue efforts can be conducted immediately to save as many survivors as possible. For efficient manpower planning, understanding the total number of victims in a particular region is also essential. With this information, manpower can be planned and coordinated properly. Thus, information such as the type of disaster and the number of victims in the disaster area is crucial for disaster response.

The lacking of communication capabilities will diminish the performance of the rescue actions. Traditionally, rescue actions usually comprise three components: command post, scout teams, and rescue teams (Boukerche and Coutinho, 2018). A command post is responsible for centralising information and coordinating the rescue missions. Scout teams are responsible for surveying the impacted area and search for victims. They will report their findings to the command post. Lastly, rescue teams will coordinate rescue efforts based on the information provided by the command post. There are two limitations to the traditional approach. Firstly, the delay in information sharing between the teams and the command post may delay the timely rescue. Secondly, some manpower has to be allocated for surveying instead of rescuing people.

In recent years, Internet of Things (IoT) is actively adopted to improve situational awareness for disaster response. Besides, the increased adoption of smart surveillance cameras allows various real-time video analytics. These smart surveillance cameras can be integrated into an IoT environment for disaster response. Smart cameras can perform image analytics, and the analytics information can be shared via the IoT framework. With such systems, more manpower can be allocated to the rescue teams instead of scout teams.

## **1.2 Importance of the Study**

Image analytics for disaster response has great potential. Firstly, there are many existing smart surveillance cameras in urbanised cities (Khan et al., 2019; Chaudhuri and Bose, 2019). These cameras can be leveraged for situational awareness for disaster response. Besides, cameras can be used to detect multiple types of disasters. Most conventional disaster monitoring system uses specialised sensors for different disasters. Camera as a sensor can assist the existing disaster monitoring sensor, if not replacing them. For instance, it is very hard to detect smoke using a smoke sensor in an outdoor environment. It is more effective to use a convolutional neural network (CNN) for smoke detection and wildfire detection (Khan et al., 2019). Besides detecting disasters, smart cameras can also be used for victim detection. CNN is proven to be useful for object detection.

Thus, a CNN can then be deployed in an IoT framework so that cataclysmic events can be detected immediately and timely rescue actions can be conducted. Meanwhile, the CNN model can be fine-tuned or retrained for higher accuracy with more training data collected in each IoT node. To achieve this, federated learning can train the CNN model in a decentralized manner, using data in each IoT node.

## **1.3 Problem Statement**

Currently, there is a lack of multi-task models that could perform multiple image analytic tasks for disaster response. For instance, separate models are used for disaster classification and person detection. A camera device has to run an N number of CNN for N tasks. This approach is complicated and not



feasible since most smart edge devices have limited computational power. Thus, a unified multi-task model can be designed to perform the two tasks together. A multi-task CNN model can be used in an IoT framework for efficient disaster response. Since there is always a bias in the training dataset, data collection and model retraining are some of the important phases in a machine learning operation (ML Ops) cycle. However, existing work on disaster response systems often neglected these phases. Based on this intuition, active learning (AL) and federated learning (FL) can be integrated into the overall system, making the system sustainable. Besides that, past works on disaster response systems often neglected the prohibitive inference workload of an edge device to run a deep learning model, which may prohibit the widespread deployment of such systems. Thus, this project proposes integrating the Open Visual Inference and Neural Network Optimization (OpenVINO) into the system, to optimize the model for faster inference speed with little performance trade-off.

#### **1.4 Aim and Objectives**

This project aims to develop a multi-task model for disaster response. The objectives of this study are:

- i. To design a multi-task model for disaster classification, as well as victim detection and counting.
- ii. To deploy the multi-task model in a sustainable machine learning operation system.
- iii. To evaluate the performance in terms of accuracy and speed.

#### **1.5 Scope and Limitation of the Study**

Past research works on disaster response and multi-task models are explored in the literature review. A multi-task model has been developed for disaster classification and victim detection. The multi-task model was trained using several strategies, including centralized learning, federated learning, and active learning-based federated learning. The performance of the multi-task model trained via these methods were compared.

For disaster classification, the dataset used in this study is the sub-dataset for disaster types from the “Crisis Image Benchmarks Dataset”. The

dataset is meant for benchmarking for disaster-related tasks. However, a large portion of the images from the dataset are collected from social media or the internet. Thus, the training images may not resemble the actual images acquired from a smart surveillance camera. Besides, there is no victim detection dataset available on the internet. Hence, the dataset for disaster classification is annotated for victim detection. The ground truth label for the localisation of victims in the images is done using an auto-annotation tool.

## **1.6 Contribution of the Study**

This project will develop a working system to train a multi-task model in a real-world setup, where the dataset is distributed around the edge devices. The contributions of this project are listed below:

- i. Discuss past works on disaster response systems.
- ii. Propose a multi-task model for disaster classification and victim detection.
- iii. Propose a federated learning pipeline to train the multi-task model.
- iv. Propose an active learning strategy to assist the data labelling.

## **1.7 Outline of the Report**

This report consists of five chapters. Each chapter will provide the readers with an adequate amount of information. The outline of each chapter is described below.

Chapter 1 discusses the project background, aims and objectives, scope and limitation, as well as the contribution of the study, to allow the reader to have a basic understanding of the project.

Chapter 2 reviews the past works on the related topic, including disaster classification system, deep learning multi-task model, federated learning and active learning.

Chapter 3 explains the methodology of the project. The required tools and pipeline for each component of the project will be listed in this chapter. The readers will be able to replicate the project following the procedures.

Chapter 4 demonstrates the findings of the project, mainly on the results and analysis of the performance of the multi-task model under different training strategies.

Chapter 5 concludes this project and provide suggestions for future work.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Introduction

This project aims to develop a multi-task Convolutional Neural Network (CNN) for two major tasks, which are (i) disaster classification as well as (ii) victim detection and counting. Therefore, this chapter will first cover some literature review on Multi-Task Learning (MTL), as well as its application involving either disaster prediction and/or object detection. Besides, this chapter will cover edge computing as the computing paradigm for machine learning deployment. Lastly, federated learning and active learning will be introduced later in the chapter as tools to train the multi-task model in a real-world deployment.

#### 2.2 Multi-Task Learning in Deep Learning

One of the intuitions of MTL is to perform more tasks using one model, without the need of using multiple models for each task. This section will discuss the general approaches of MTL and some possible explanations on why it works.

##### 2.2.1 General Approaches of Multi-Task Learning in Deep Learning

In Deep Learning (DL), the general approaches of MTL can be categorized into two main methods, which are hard and soft parameter sharing.

##### Hard Parameter Sharing

Hard parameter sharing is the most frequently used approach to MTL in DL (Ruder, 2017). As illustrated in Figure 2.1, the general idea of hard parameter sharing is to share multiple hidden layers for all tasks, which then branched out into several task-specific output layers. In the vision domain, the shared hidden layers are usually the modern CNN architectures such as VGG16, AlexNet, ResNet50, ResNet101, InceptionNet, MobileNet and EfficientNet. According to Baxter (1996), hard parameter sharing could lead to better generalisation of the neural network. Supposing we have  $N$  number of tasks,

the risk of overfitting the shared parameters is an order  $N$  smaller than overfitting the task-specific parameters (Baxter, 1997). This is because a model that learns several tasks simultaneously needs to find a representation that fits all tasks (Ruder, 2017). However, the exact correlation between the number of tasks and the impact on generalisation depends on the task domain as well as the neural network architecture. Generally, it results in improved generalisation if the tasks are related. Although hard parameter sharing is useful in many scenarios, it could break down easily if the tasks are not closely related or require reasoning on different levels.

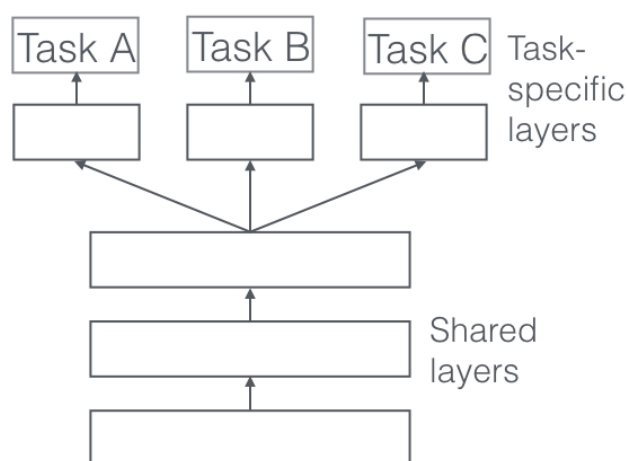


Figure 2.1: Hard Parameter Sharing for Multi-Task Learning (Ruder, 2017).

### Soft Parameter Sharing

For soft parameter sharing, each task has its own model with its own parameters. Specifically, each task has its own backbone, where the parameters of each backbone are then regularised to encourage them to be similar. These layers are often referred to as the constrained layers. After that, each backbone is connected to the task-specific output layers. Figure 2.2 shows an example of MTL using the soft parameter sharing approach.

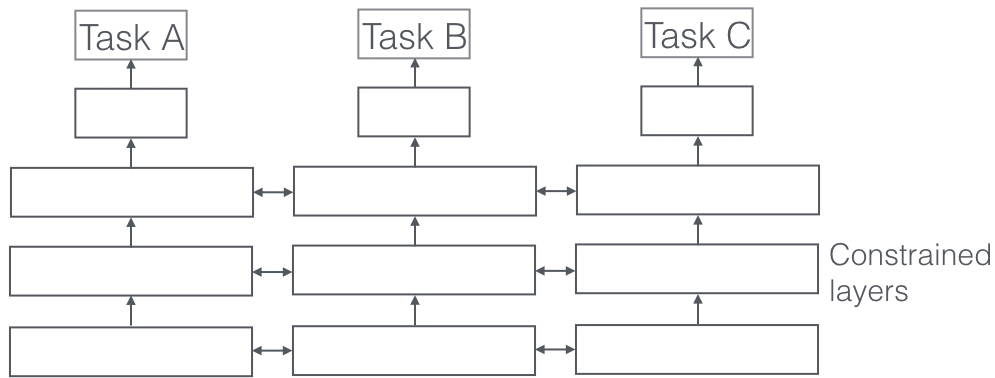


Figure 2.2: Soft Parameter Sharing for Multi-Task Learning (Ruder, 2017).

### 2.2.2 Why does Multi-Task Learning work?

Table 2.1 shows a summary of how MTL works according to Ruder (2017). For all examples, it is assumed that there are two related tasks A and B, which rely on a common hidden layer representation F.

Table 2.1: Why does Multi-Task Learning Work (Ruder, 2017)?

<b>Implicit data augmentation</b>	<ul style="list-style-type: none"> <li>- Different tasks have different noise patterns</li> <li>- A model that only learns a single task has a higher risk of overfitting to that task</li> <li>- A model that learns several tasks simultaneously could learn a more general representation F through averaging the noise patterns</li> </ul>
<b>Attention focusing</b>	<ul style="list-style-type: none"> <li>- It can be difficult for a model to differentiate between relevant and irrelevant features if (i) a task is very noisy, or (ii) data is limited and high-dimensional</li> <li>- If a model has difficulties in learning features for Task A, the features learnt from Task B could help the model to better differentiate between relevant and irrelevant features for Task A</li> </ul>
<b>Eavesdropping</b>	<ul style="list-style-type: none"> <li>- Some features G are easy to learn for task A, but difficult to learn for another task B</li> <li>- This could be either (i) task B interacts with the</li> </ul>

	<p>features in a more complex way or (ii) other features are impeding the model's ability to learn features G</p> <ul style="list-style-type: none"> <li>- Therefore, the model could learn features G from task A, which will help to improve the performance of the model on task B</li> </ul>
<b>Representation bias</b>	<ul style="list-style-type: none"> <li>- MTL biases the model to prefer representations that other tasks also prefer</li> <li>- This will help the model to generalise to new tasks in the future because a model that performs well for a large number of training tasks could also perform well for learning novel tasks as long as they are from the same environment</li> </ul>
<b>Regularization</b>	<ul style="list-style-type: none"> <li>- MTL could introduce inductive bias, acting as a regularise</li> <li>- A common form of inductive bias is L1 regularisation, which biases the model for sparse solutions</li> </ul>

### 2.3 Disaster Classification using Deep Learning

Disaster classification is one of the main tasks in this project. However, there are limited works on MTL in the disaster classification domain. Therefore, this section will cover relevant works involving disaster classification, which are not limited to MTL. Table 2.2 shows a summary of the research work related to disaster classification.

Table 2.2: Research Work Related to Disaster Classification.

	<b>Title</b>	<b>Year</b>	<b>Dataset</b>	<b>Main Highlights</b>
1	Application of Image Analytics for	2019	- Focused on <b>Earthquake</b> images	- Proposed a novel application of big data (particularly the

	Disaster Response in Smart Cities (Chaudhuri and Bose, 2019)		<ul style="list-style-type: none"> <li>- Earthquake images are acquired from (i) National Hazards Image Dataset of the National Centres for Environmental Information, and (ii) images from Google</li> </ul>	<ul style="list-style-type: none"> <li>- visual content by surveillance camera) in smart cities</li> <li>- Used CNN to detect the presence of victim (binary classification of victim presence)</li> </ul>
2	Disaster Prediction System using Convolution Neural Network (Padmawar et al., 2019)	2019	<ul style="list-style-type: none"> <li>- Focused on <b>Flood</b> images</li> <li>- Images obtained from Google</li> <li>- 137 training images, 500 testing images</li> </ul>	<ul style="list-style-type: none"> <li>- Input images are passed to Modified Particle Swarm Optimization (MPSO) before passing to CNN</li> <li>- MPSO is used to search for the optimal parameter values from the inputs for the CNN training process</li> </ul>
3	Flood Disaster Identification and Decision Support System using Crowdsourced Data Based on	2020	<ul style="list-style-type: none"> <li>- Focused on <b>Flood</b> images</li> <li>- Crowdsourced images from flood hotspots in Thailand</li> <li>- The number of</li> </ul>	<ul style="list-style-type: none"> <li>- Proposed a web application for flood detection analysis</li> <li>- Used CNN to identify flood events based on the uploaded images</li> </ul>



	Convolutional Neural Network and 3S Technology (Puttinaovarat et al., 2020)		training images is not mentioned, but 200 images are used for testing	
4	CrisisMMD: Multimodal Twitter Datasets from Natural Disasters (Alam, Ofli and Imran, 2018)	2018	<ul style="list-style-type: none"> <li>- Focused on several major natural disasters, including <b>earthquakes, hurricanes, wildfires, and floods</b></li> <li>- The images and texts are labelled for three tasks, which are (i) informativeness, (ii) humanitarian categories and (iii) damage severity assessment</li> <li>- A total of 18082 Twitter images and 16058 Twitter</li> </ul>	<ul style="list-style-type: none"> <li>- The first public multimodal dataset for natural disasters</li> <li>- The dataset is multi-label as well</li> <li>- The CrisisMMD dataset is actively used in ongoing disaster response research</li> </ul>

			texts are collected	
5	Multimodal Analysis of Disaster Tweets (Gautam et al., 2019)	2019	<ul style="list-style-type: none"> <li>- Used a custom subset of <b>CrisisMMD</b> dataset</li> <li>- Only focused on the informativeness labels of the images and texts</li> <li>- A total of 12762 images (along with tweeter text) are used</li> <li>- The dataset is split into 70, 10 and 20 rations for training, validation and testing set, respectively</li> </ul>	<ul style="list-style-type: none"> <li>- The multimodal analysis is done on the decision level instead of the features level</li> <li>- Separate models are used to extract text and visual features/representation.</li> <li>- A policy system is used to evaluate which systems are able to best filter the required information</li> </ul>
6	Detecting Disaster-Related Tweets Via Multimodal Adversarial	2020	<ul style="list-style-type: none"> <li>- A total of 4000 texts and 4562 images are extracted from the <b>CrisisMMD</b></li> </ul>	<ul style="list-style-type: none"> <li>- CNN and Bidirectional Encoder Representations from Transformer (BERT) models are used to</li> </ul>

	Neural Network (Gao et al., 2020)		<p>dataset as the positive examples</p> <ul style="list-style-type: none"> <li>- 8415 English tweets (with pictures) irrelevant to disasters are collected as negative examples</li> <li>- 80 and 20 rations are used for training and testing dataset, respectively</li> </ul>	<p>extract visual and text features from the input images and texts</p> <ul style="list-style-type: none"> <li>- The visual and text features are concatenated into a single multimodal feature</li> <li>- The multimodal feature is then passed to two different multilayer perceptrons for disaster discrimination and tweet detection</li> </ul>
7	Deep Learning Benchmarks and Datasets for Social Media Image Classification for Disaster Response (Alam et al., 2020)	2020	<ul style="list-style-type: none"> <li>- A consolidated dataset comprised of multiple public and in-house datasets, including (i) Damage Assessment Dataset (DAD), (ii) CrisisMMD, (iii) AIDR Dataset and (iv) Damage</li> </ul>	<ul style="list-style-type: none"> <li>- The dataset aims to provide proper baselines for disaster prediction related tasks</li> </ul>

			<p>Multimodal Dataset (DMD).</p> <ul style="list-style-type: none"> <li>- The dataset is labelled for four tasks, including (i) disaster type classification, (ii) informativeness, (iii) humanitarian category and (iv) damage severity</li> </ul>	
8	<p>Social Media Images Classification Models for Real-time Disaster Response (Alam et al., 2021)</p>	2021	<ul style="list-style-type: none"> <li>- The consolidated dataset created by research work (7) by Alam et al. (2020)</li> </ul>	<ul style="list-style-type: none"> <li>- An MTL model is used for the four tasks labelled in the consolidated dataset.</li> <li>- MTL can be an ideal solution for the real-time system as it can potentially provide speed-ups of multiple factors during inference.</li> <li>- However, some tasks may perform worse than their single task settings (in the presence of</li> </ul>

				incomplete labels).
--	--	--	--	---------------------

One of the notable highlights in the summary is the dataset used, which are the CrisisMMD and a consolidated dataset for disaster response. CrisisMMD dataset was released in 2018. It was the first public multimodal dataset for natural disasters, which is multi-labelled as well. The dataset was labelled for informativeness, humanitarian categories and damage severity. It is used in research work by Gautam, et al. (2019) and Gao, et al. (2020). In 2020, a consolidated dataset consisting of several datasets such as DAD, AIDR, DMD and CrisisMMD was released. There are four sub-datasets in this dataset where each of them is labelled for the disaster types, informativeness, humanitarian categories and damage severity, respectively. The consolidated dataset is meant for benchmarking in deep learning tasks related to disaster response.

Another notable highlight is research work (8) by Alam, et al. (2021) in Table 2.2, which contributed to MTL involving disaster classification. The consolidated dataset was used in this research, where a multi-task CNN is developed for the four tasks: (i) disaster type classification, (ii) informativeness, (iii) humanitarian categories, and (iv) damage severity.

Out of all research works reviewed, there are limited works involving victim detection and counting for disaster response. There is room for improvements for such tasks in the disaster response domain.

In short, the main takeaway for this section is: (i) there exists a consolidated dataset comprised of multiple reliable public datasets, which is split into non-overlapping training, validation and testing set for benchmarking, and (ii) there are limited research works on victim detection and counting for disaster response. An object detection model for victim detection can be incorporated with disaster classification for better disaster response. Thus, the next section will discuss MTL applications involving object detection.

## 2.4 Multi-Task Learning Involving Object Detection

Applications of MTL in computer vision come in various forms. For the scope of this project, only MTL involving object detection will be discussed. Object detection is a prevalent task in the computer vision domain. An object detection task is to localize objects in the image using bounding boxes, and to predict the classes of the detected objects. Thus, object detection itself is a form of MTL application in computer vision. Table 2.3, Table 2.4 and Table 2.5 show several applications of MTL in computer vision that involve object detection as one of the main tasks. The object detectors used include variants of Single Shot Detector (SSD) and Faster Region Based Convolutional Neural Networks (R-CNN).

Table 2.3: Multi-Task Model that adds Additional Head Model(s) for Other Task(s).

	Title	Year	Types of Object Detector	Main Highlights
1	BlitzNet: A Real-Time Deep Network for Scene Understanding (Dvornik et al., 2017)	2017	Single-stage Detector (SSD Variant)	<ul style="list-style-type: none"> <li>- A multi-task model for (i) object detection and (ii) semantic segmentation of the detected object</li> <li>- Model architecture:               <ul style="list-style-type: none"> <li>a) Backbone: ResNet-50</li> <li>b) Neck: Feature Pyramid Network (FPN)</li> <li>c) Head: Two independent head models for the two tasks.</li> </ul> </li> </ul>
2	DLT-Net: Joint Detection of Drivable Areas,	2020	Single-stage Detector	<ul style="list-style-type: none"> <li>- A multi-task model for (i) traffic object detection, (ii) road segmentation and (iii)</li> </ul>

	Lane Lines, and Traffic Objects (Qian, Dolan and Yang, 2020)		(SSD Variant)	<p>lane line detection</p> <ul style="list-style-type: none"> <li>- Model architecture: <ul style="list-style-type: none"> <li>a) Backbone: VGG16</li> <li>b) Neck: FPN</li> <li>c) Head: Three head models for the three tasks. The road segmentation head model is shared and combined with the other two head models.</li> </ul> </li> </ul>
3	Multi-scene citrus detection based on multi-task deep learning network (Wen et al., 2020)	2020	Two-stage Detector (Faster R-CNN)	<ul style="list-style-type: none"> <li>- A multi-task model for (i) object detection and (ii) instance segmentation of citrus in a given image, as well as (iii) maturity and (iv) quality of the detected citrus</li> <li>- Two head models for maturity and quality prediction are added to the original RCNN head.</li> <li>- A segmentation head model is added into the architecture, parallel to the RCNN head.</li> <li>- Refer to Figure 2.3 for a more detailed model definition.</li> </ul>
4	A loss-balanced multi-task model for simultaneous	2021	Single-stage Detector	<ul style="list-style-type: none"> <li>- A multi-task model for (i) object detection and (ii) semantic segmentation</li> </ul>

<p>detection and segmentation (Zhang et al., 2021)</p>		<p>(SSD Variant)</p>	<ul style="list-style-type: none"> <li>- Model architecture:             <ul style="list-style-type: none"> <li>a) Backbone: VGG16</li> <li>b) Head: Two head models for each task</li> </ul> </li> <li>- Hard parameter sharing is used in the form of a shared backbone, while soft parameter sharing is applied to the two head models using Task-related Attention Module (TAM)</li> <li>- TAM allows the two tasks to share related information that can aid each other to select feature maps effectively</li> <li>- Refer Figure 2.4 for a more detailed model definition.</li> </ul>
--	--	----------------------	--

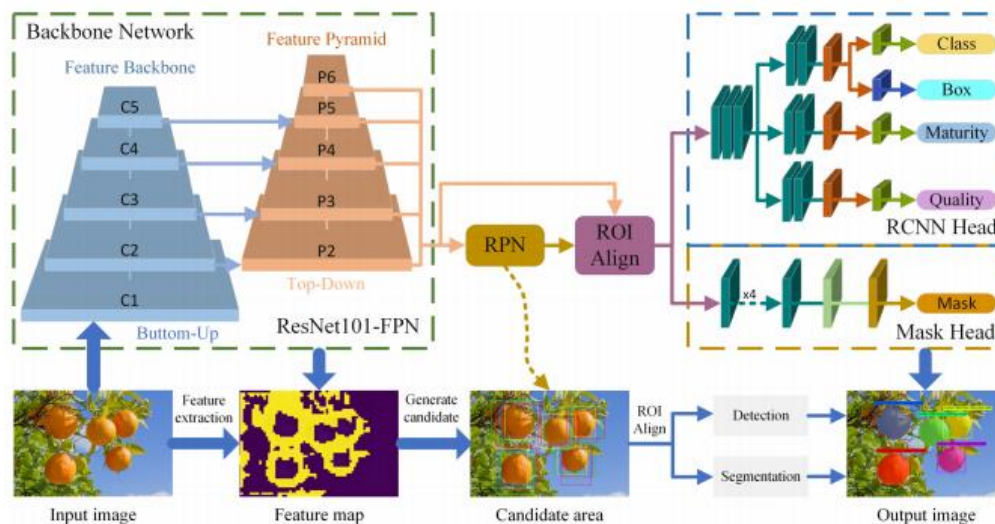


Figure 2.3: Multi-Task Model in Research Work 3 (Wen et al., 2020) in

Table 2.3. Head Models are added for each Additional Task.



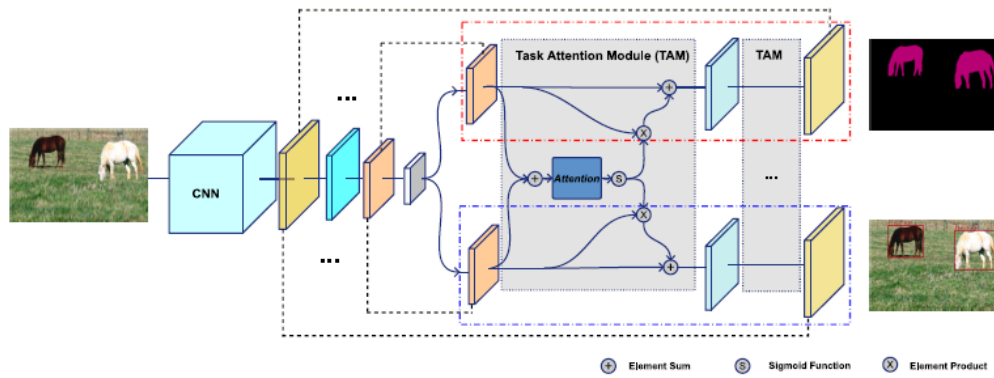


Figure 2.4: Multi-Task Model in Research Work 4 (Zhang et al., 2021) in Table 2.3. Hard Parameter Sharing is used in the form of Shared Backbone, while Soft Parameter Sharing is applied to the two Head Models in the form of TAM.

Table 2.4: Multi-Task Model that Makes Minimal Modification to the Original Object Detector Model.

	Title	Year	Types of Object Detector	Main Highlights
1	Helmet Use Detection of Tracked Motorcycles Using CNN-Based Multi-Task Learning (Lin et al., 2020)	2020	Single-stage Detector (SSD Variant, specifically RetinaNet)	<ul style="list-style-type: none"> <li>- A multi-task model for (i) motorcycle detection and (ii) helmet use detection/classification</li> <li>- No modification was done to the original object detector model. The additional tasks were performed after the object detector architecture.</li> <li>- Model architecture: <ul style="list-style-type: none"> <li>a) Motorcycle detection: RetinaNet</li> <li>b) Helmet use classification: InceptionV3 is used to</li> </ul> </li> </ul>

				classify the helmet use of each detected motorcyclist
2	Multi-task learning for dangerous object detection in autonomous driving (Chen et al., 2018)	2018	Single-stage Detector (SSD Variant)	<ul style="list-style-type: none"> <li>- A multi-task model for two tasks - object detection and distance prediction.</li> <li>- Instead of using linear multi-task combination strategy (which is often used in MTL models), the authors proposed a Cartesian product-based multi-task combination strategy.</li> <li>- Unlike traditional hard parameter sharing method (which has separate head models for each task), the authors proposed to share the head models as well.</li> <li>- To do so, object classification part of the object detection task now has two labels, which are (i) the class of the object and (ii) the distance of the object (which has been discretized).</li> <li>- All possible combination of labels is created. For instance, given M classes and N types of distance to be predicted, there will be a total</li> </ul>

				<p>of <math>M \times N</math> new labels to be formed.</p> <ul style="list-style-type: none"> <li>- Refer Figure 2.5 for detailed model definition.</li> </ul>
--	--	--	--	--

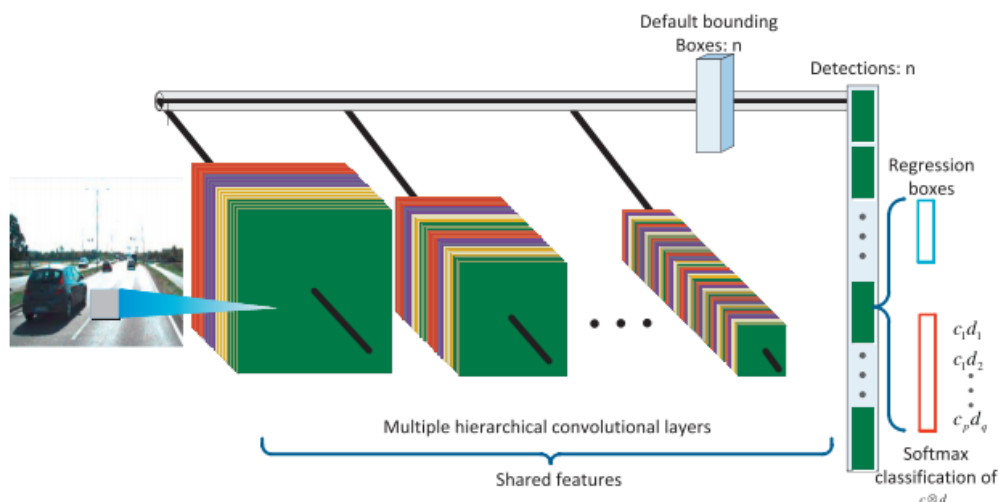


Figure 2.5: Multi-Task Model in Research Work 2 (Chen et al., 2018) in Table 2.4. Instead of Modifying the SSD Model, the Authors Introduced Variation on the Original Classes/Labels for Additional Tasks. Originally, there were M numbers of Labels. N Variations are Introduced for Each Label, resulting in  $M \times N$  Labels.

Table 2.5: Object Detection Model that Implements MTL to Improve Object Detection.

	Title	Year	Types of Object Detector	Main Highlights
1	Multi-task Self-supervised Object Detection via Recycling of Bounding Box	2019	Two-stage Detector	<ul style="list-style-type: none"> <li>- MTL and self-supervised learning are used to improve the performance of objection detection.</li> <li>- The feature maps generated</li> </ul>

	<p>Annotations (Lee, Na and Kim, 2019)</p>		<p>by the region proposal network will be used by both the main and auxiliary tasks.</p> <ul style="list-style-type: none"> <li>- The main task is object detection, while three auxiliary tasks are (i) multi-object soft labels, (ii) closeness labels and (iii) foreground labels.</li> <li>- The output vectors of each auxiliary task will serve as a “refinement” for the class prediction in the main task.</li> <li>- Refer Figure 2.6 for a detailed model definition.</li> </ul>
--	--	--	--

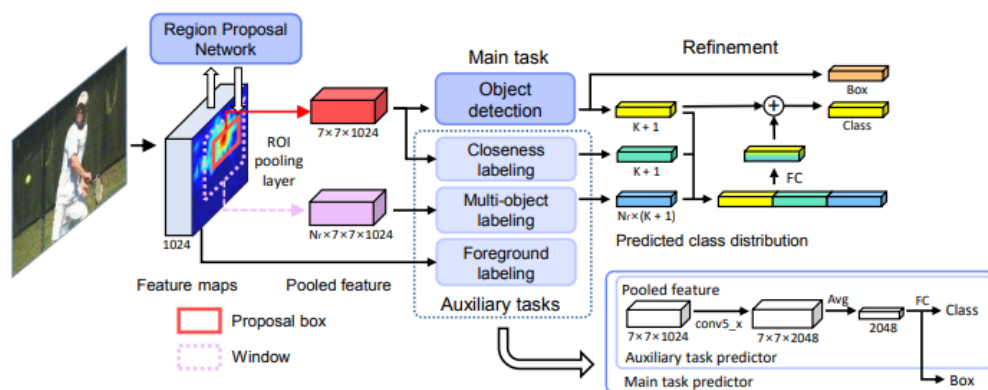


Figure 2.6: Multi-Task Model in Research Work 1 (Lee, Na and Kim, 2019) in Table 2.5. Additional Tasks are Auxiliary Tasks, which are meant to Improve the Learning of the Main Task.

MTL related to object detection comes in three forms: (i) additional head model for the additional task, (ii) minimal modification to the object detection model, and (iii) auxiliary tasks to improve the performance of the object detection model. For type (i), usually, the additional head model will branch out from the backbone or the neck. Type (ii) can be achieved by

varying the labels, or by adding additional layers after the object detection head model. For type (iii), the additional tasks are the auxiliary tasks, which have no actual purpose other than aiding the training of object detection.

Most of the discussed research works apply some forms of hard parameter sharing with slight modification as their main approach to MTL. It is easier to implement the hard parameter sharing approach, as it is analogous to a “plug and play” mechanism. In most examples, a new head model can be added for a new task. On the other hand, soft parameter sharing requires a more complex architecture to share the parameters between the two tasks. The key takeaway in this section is that hard parameter sharing is still prevalent in MTL for computer vision. Thus, the multi-task CNN model for this project will be based on the hard parameter sharing setup.

## **2.5 Edge Computing**

Cloud computing has been one of the emerging computing trends in recent years. According to Patidar, Rane and Jain (2012), “clouds” refer to a large pool of resources, such as hardware and services, which are virtualized so that end-user can easily access these resources without the need to own them physically. To achieve this, cloud service providers own a massive amount of resources, where they rent the resources to the clients as a service. In recent years, multiple cloud service providers have provided platforms for machine learning developers to train and/or deploy the application via the cloud. Machine learning deployment using cloud computing requires the end devices to send the data to the cloud, where the cloud server will run the prediction model.

However, cloud computing is unsuitable for applications that demand low latency. Therefore, a new computing framework called edge computing is introduced to address such application demands. Edge computing is a distributed computing framework that brings computation and data storage closer to data sources, such as IoT devices or local edge servers (Xia et al., 2021). Edge computing coupled with machine learning is a promising technology that has a wide field of applications. For a machine learning application, edge devices could deploy a machine learning model locally without sending the data to the centralized server and waiting for the

result. This could significantly reduce the latency dramatically as data does not need to travel, making edge computing a suitable computing paradigm for real-time and time-sensitive applications. However, edge devices generally do not have enough computing power to match the supercomputers in the central server. Thus, recent efforts in machine learning deployment on the edge focus on optimizing the machine learning models via methods such as model compression and knowledge distillation (Buciluundefined, Caruana and Niculescu-Mizil, 2006; Gou et al., 2020).

Intel OpenVINO (Open Visual Inference and Neural Network Optimization) is a free toolkit developed to facilitate the (i) optimization of deep learning models, and to (ii) deploy the optimized model using an inference engine onto an Intel Hardware (Intel, 2022). For users who are non-machine learning researchers, the OpenVINO toolkit also offers various pretrained deep learning models that are optimized for inference in their Open Model Zoo platform (Intel, 2022). Examples of OpenVINO application includes license plate recognition (Castro-Zunti, Yépez and Ko, 2020), person tracking (Yrjänäinen et al., 2020) and person reidentification system (Izutov, 2018) using CNN. OpenVINO is a mature toolkit that has become one of the go-to toolkits for deep learning model optimization and inferencing.

## **2.6 Federated Learning**

Multiple data can be collected throughout the service of an edge device. Naturally, the next step is to use the data to train or fine-tune the machine learning model. However, the data collected on each edge device are limited and may be biased. It is widely known that larger and diverse datasets are required to generate a reliable machine learning model (Kaushal, Altman and Langlotz, 2020). Although there is no guarantee that more data translates to better performance, it is without a doubt the right direction toward improving accuracy and reducing bias in a machine learning model (Reina et al., 2021). However, collecting data from multiple data sources to a centralized server is time-consuming and may raise privacy concerns. Federated Learning (FL) is a technique that aims to overcome this issue by training machine learning models in a decentralized form. It is a collaborative machine learning

framework that allows devices from different private datasets to work together to train a global model. In this setting, a single global model is stored in a central server, while the data are stored locally at where they are collected. During training, only the model weights have to be transferred across the network, which is faster and easier compared to transferring the whole dataset across the network. Figure 2.7 shows the flowchart for a typical FL pipeline.

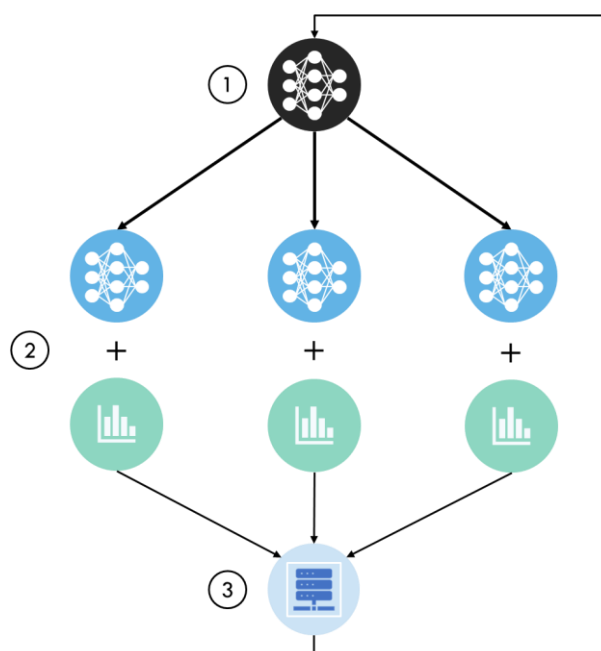


Figure 2.7: The Flowchart of a typical FL Pipeline. Component 1 represents the Global Model. Component 2 represents the Copy of the Global Model in each client, each trained on the Local Dataset. Finally, Component 3 is the Aggregator Function to Merge all the Local Models into one Global Model for the next round of FL.

## 2.7 Active Learning

Active learning (AL) is the process of prioritizing the data which needs to be labelled in order to have the highest impact on training a supervised model. AL is applied in situations where there is a large amount of unlabeled data, where some priority needs to be made to label the data in an intelligent way. Figure 2.8 shows an example of the decision boundary of a given dataset with two features, which are feature 1 and feature 2. Most of the data points in the dataset are trivial for the model, where the model can make a prediction

easily, as shown in Figure 2.9. On the other hand, data points closer to the decision boundary are usually more challenging for the model to accurately make the correct prediction, as illustrated in Figure 2.10. These data points are more informative for model training, as the model has to be robust enough to generate a more accurate decision boundary to predict them. These data points are also referred to as informative data, as they are more informative than those trivial data points.

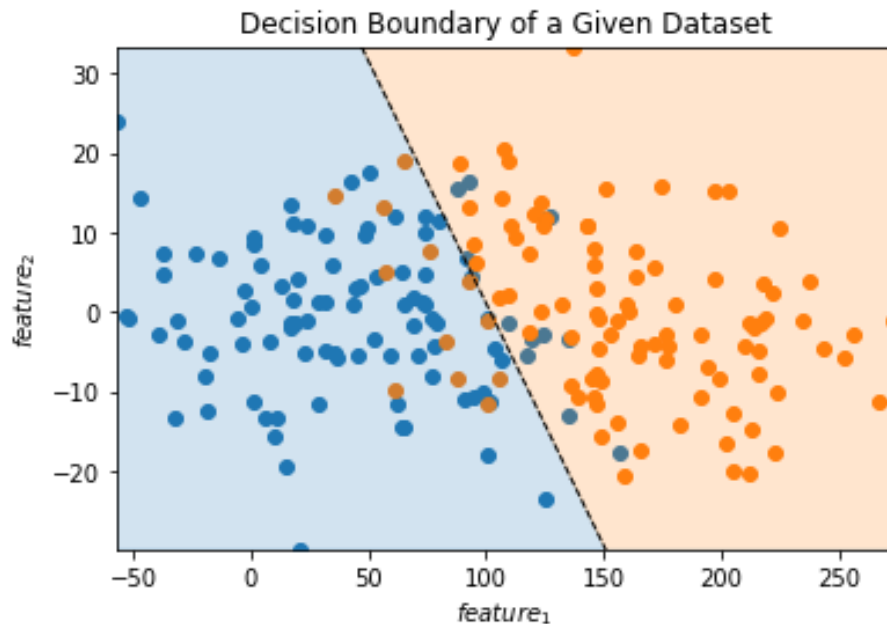


Figure 2.8: Decision Boundary of a Given Dataset.

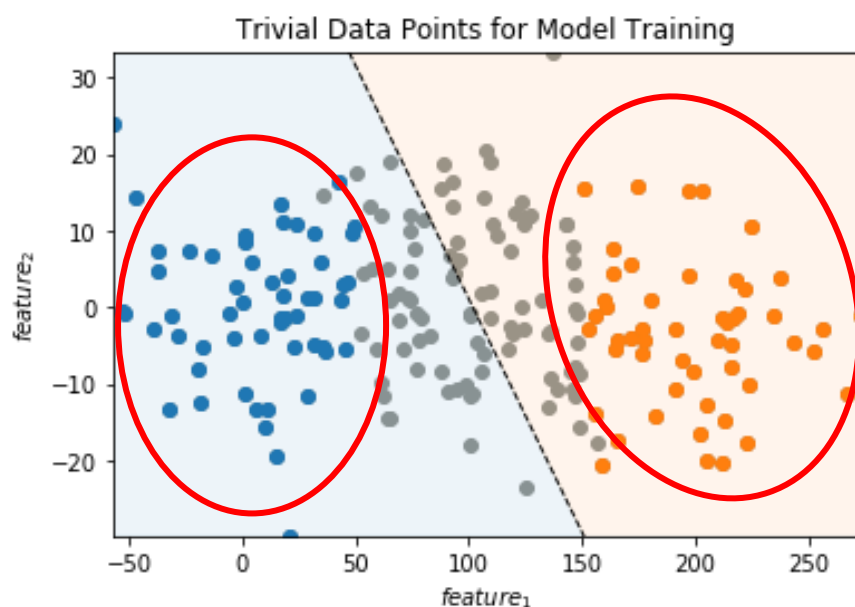


Figure 2.9: Red-Circled Data Points are Trivial for Model Training.



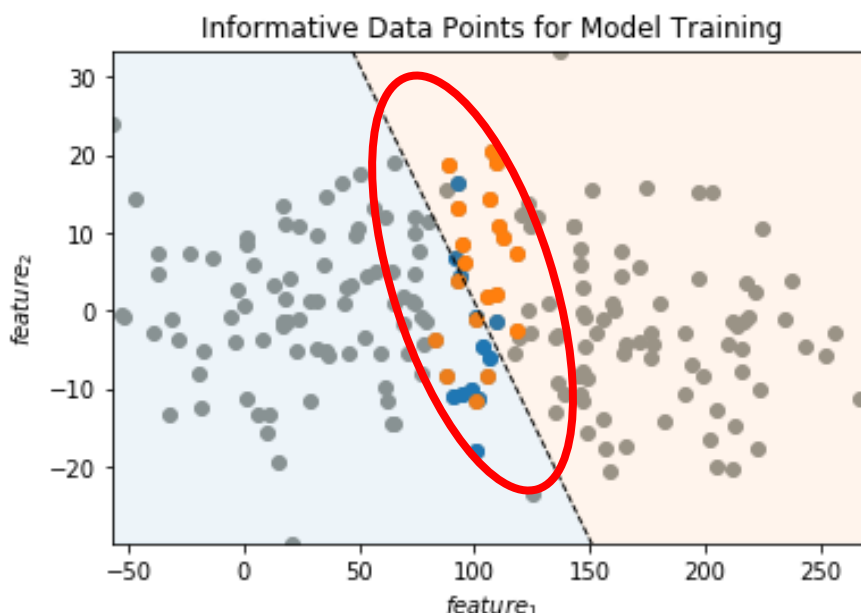


Figure 2.10: Red-Circled Data Points are the Informative Data Points, which are essential for Model Training.

In AL, there are usually three major components which are (i) the seed dataset (the initial training dataset for the model, which is usually small in size), (ii) a pool of unlabeled dataset, and (iii) a learner (the model trained on the seed dataset). There are typically three types of active learning querying strategies, which are (i) membership query synthesis, (ii) stream-based selective sampling, and (iii) pool-based sampling (Settles, 2010). In membership query synthesis, the learner constructs an instance from the underlying natural distribution, and requests a human annotator to label the instance. However, this could be a problem for image datasets, as the instance generated by the model could be unrecognizable for humans. On the other hand, both stream-based selective and pool-based sampling expect a large pool of unlabeled data. The former draws unlabeled instances from the data source one by one and lets the learner decide if it wants to query the image for human annotation based on its informativeness. On the other hand, the latter draws multiple queries from the unlabeled data in a greedy fashion, according to the informativeness of the data. Among the three methods, the pool-based sampling technique has been deployed in waste and natural disaster classification applications (Ahmed et al., 2020). Thus, this sampling technique will be used in this project for the AL phase.

## 2.8 Summary

The general approaches of MTL can be categorized into soft and hard parameter sharing. While there are various methods used to improve MTL, hard parameter sharing is still widely used in MTL for computer vision. In the disaster response domain, there are limited works that involve both disaster classification and victim detection. Thus, a multi-task model for the two tasks will be developed to contribute to disaster response.

Besides, the multi-task model has to be optimized before deploying it in the edge devices in an IoT setup. Thus, the project will adopt the OpenVINO toolkit to optimize the model to be computationally less expensive while retaining its performance. In addition, since each edge device can collect the dataset during its deployment, naturally, the next step is to use it for training. However, it is not practical to collect local datasets from all clients to a centralized server for model training. Thus, we can use federated learning to train a global model using all datasets from each client, without collecting and centralizing the dataset in a data center. Lastly, active learning can aid human annotators in selecting informative data for the model training, where the pool-based sampling technique will be used in this project.

## CHAPTER 3

### METHODOLOGY AND WORK PLAN

#### 3.1 Introduction

This chapter discusses the methodology and work plan of the project. The considerations for tools selection, including the training platform, deep learning and FL framework, will be covered as well. Then, the architecture of the MTL model and the training pipeline will be explained. Finally, the trained MTL model is then optimised using the OpenVINO toolkit for faster inference speed.

#### 3.2 Work Plan

Figure 3.1 and Figure 3.2 show the Gantt chart for Part One and Part Two of this project. Part One of this project will focus mainly on data preparation and the development of the multi-output model. On the other hand, Part Two will focus on integrating FL and AL to train the system in a real-world IoT setup.

No.	Project Activities	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
M1	Problem Formulation and Project Planning	■	■												
M2	Literature Review		■	■	■	■	■								
M3	Data Preparation			■	■	■	■	■							
M4	Development of the Multi-Output Model				■	■	■	■	■	■	■	■	■		
M5	Report Submission												■	■	■

Figure 3.1: Gantt Chart for Final Year Project Part 1.

No.	Project Activities	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
M1	Problem Formulation and Project Planning	■	■												
M2	Development of the IoT setup			■	■	■	■	■	■	■	■	■			
M3	Deployment of the Multi-Output Model in IoT setup									■	■	■			
M4	Report Submission												■	■	■

Figure 3.2: Gantt Chart for Final Year Project Part 2.

### 3.3 Tools to Use

#### 3.3.1 Training Platform

The prototyping and training of the model will be conducted in two platforms, which are (i) Google Colaboratory (hereinafter referred to as Colab) and (ii) a workstation.

Colab is a product from Google Research, where it is a hosted Jupyter notebook service that provides free access to computing resources, such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs). The GPUs available in Colab include Nvidia K80s, T4s, P4s and P100s. However, Colab resources are not guaranteed and are nowhere unlimited. In order to provide its service for free, Colab needs to maintain the flexibility to adjust hardware availability to provide the computational resources for more users (Google, 2021b). Thus, users cannot choose the type of GPU to connect to, and the maximum runtime is only 12 hours.

Google also provides the paid version of Colab, which are the Colab Pro and Colab Pro+. The main benefits of the paid versions include (i) access to faster GPUs and TPUs, (ii) more memory, as well as (iii) longer runtimes (Google, 2021a). Although better GPUs will be reserved for paid users, there is still no guarantee on which types of GPU is provided. Besides, the maximum runtime for paid versions is up to 24 hours, roughly twice longer than the free version. The non-continuous session in any Colab versions will cause issues in storing the dataset. Colab can access the dataset in Drive after initializing the session if the dataset can be stored in Drive. However, the

dataset cannot be downloaded to Drive if the size is too big. The remaining option is to download the dataset whenever using Colab.

On the other hand, a workstation has the benefit of clear hardware specifications and continuous runtime. The workstation used in this project utilizes the GeForce RTX 2070 SUPER Graphic Cards as its GPU. The memory size for the GPU is 8 gigabytes (GB), which is considerably large. Besides, the disk space for the workstation is 514.4 GB. The disk space is sufficient to store all the required datasets and software dependencies for the project. After downloading the required dataset and software dependencies, the remaining disk space still has 432.4 GB.

Due to the limited runtime, Colab is not suitable for continuous training of our model. However, most popular Python packages and dependencies are ready to be used in Colab. Thus, fast prototyping will be performed in Colab, while actual training will be conducted in the workstation since there is no time limit constraint. Table 3.1 shows the comparison between Colab and the workstation used in this project.

Table 3.1: Comparison between Colab and the Workstation Used in This Project.

	Colab	Workstation
<b>GPU Memory Size</b>	- 12 GB, 16 GB or 24 GB (depending on the GPU provided)	- 8 GB
<b>Available Disk Space</b>	- Roughly 77 GB	- Roughly 514.4GB
<b>Runtime Limit</b>	- Up to 12 hours for free version - Up to 24 hours for paid versions	- No limit

### 3.3.2 Deep Learning Framework

In 2021, TensorFlow and PyTorch are the most popular deep learning frameworks available. TensorFlow is developed by Google Brain, while PyTorch is developed by Facebook's AI Research Lab (FAIR). Both

frameworks are considerably matured and have a large community supporting the frameworks. Table 3.2 shows the comparison between TensorFlow and PyTorch as of 2021 (Boesch, 2021).

Table 3.2: Comparison between TensorFlow and PyTorch (Boesch, 2021).

	<b>TensorFlow</b>	<b>PyTorch</b>
<b>Adoption</b>	<ul style="list-style-type: none"> <li>- Considered the to-go tool by many researchers and industry professionals.</li> </ul>	<ul style="list-style-type: none"> <li>- Lack of model serving in production.</li> <li>- Currently used more in the research domain.</li> </ul>
<b>Scalability</b>	<ul style="list-style-type: none"> <li>- Very scalable.</li> <li>- Can be deployed on every machine, including servers and mobile devices.</li> </ul>	<ul style="list-style-type: none"> <li>- Less scalable.</li> <li>- Requires conversion of the PyTorch code into another framework (such as Caffe2) to deploy applications to servers and mobile devices.</li> </ul>
<b>Learning Curve</b>	<ul style="list-style-type: none"> <li>- Due to the low-level implementations of neural network structure, it is harder to learn.</li> <li>- However, it is compatible with Keras, which allow users to code high-level functionality.</li> </ul>	<ul style="list-style-type: none"> <li>- The syntax is similar to conventional Programming languages, making it easier to learn.</li> </ul>

TensorFlow is preferred if the application is meant for real production work. This is because TensorFlow has been widely used for production work compared to PyTorch. Hence, the TensorFlow developer communities are larger than the latter. For instance, the 2020 Stack Overflow Developer Survey reports that 10.4 % of professional developers use TensorFlow, while 7.6 % use PyTorch (Stack Overflow, 2020). However, if the application is not for real production work, the decision between the two frameworks is subject to a preference matter. For this project, TensorFlow is

used because it has more lower-level functionalities that may be useful for the design and training of the MTL model.

TensorFlow can be executed in two modes, which are (i) graph mode and (ii) eager mode. In graph mode, the tensor (high dimensional array) computations are executed as a TensorFlow graph. In a TensorFlow graph, all the operations are defined and connected into a graph data structure. The graph can be optimized, allowing the compiler to run the operations more efficiently (TensorFlow, 2021a).

In eager mode, TensorFlow operations are executed by Python operation by operation, without building graphs. After executing each operation, the results will be returned immediately to Python, which is then used for other operations (TensorFlow, 2021a). Eager mode is better for debugging since users can use standard Python debugging tools to identify errors. However, it is slower than graph mode since the operations are not optimized.

Graph mode is generally recommended for training, while the eager mode is often used for debugging. Both modes will be used in this project, as mentioned in Section 3.6.

### **3.3.3 Federated Learning Framework**

Currently, there are multiple open-source FL frameworks available. This section will only discuss FL frameworks that support Python 3 and TensorFlow 2.0, since these are the tools used in this project. TensorFlow Federated (TFF) and Intel Open Federated Learning (OpenFL) are the more popular choices among these frameworks.

TFF is an FL framework developed by Google. TFF consists of two main application programming interface (API) layers, which are (i) Federated Learning (FL) API and (ii) Federated Core (FC) API. FL API offers a set of high-level interfaces with implementations of the existing federated algorithm, which developers can directly apply to their existing TensorFlow models (TensorFlow, 2022c). On the other hand, FC API serves as the foundation for the FL API, where it includes a set of lower-level interfaces to build a novel federated algorithm (TensorFlow, 2022c). Although TFF is

designed with deployment to real devices in mind, it is currently only ready for simulations as of the time of writing (TensorFlow, 2022).

OpenFL is another Python 3 open-source federated learning framework that supports TensorFlow 2.0. It was originally developed by Intel Labs and the University of Pennsylvania for healthcare applications, and was then further developed for general-purpose real-world applications. OpenFL works with training pipelines built with TensorFlow and PyTorch, and can be extended to other machine learning and deep learning frameworks. It is a production-ready platform that is widely adopted in various industries, including medical image segmentation (Sheller et al., 2019), medical image classification (Baid et al., 2022) and prediction of physiological effects of radiation exposure on astronauts (Intel, 2021). The library consists of two components: (i) collaborator and (ii) aggregator. Collaborators are the devices that use their local dataset to train the global model, and aggregators will receive the model updates from each collaborator and combine them to create the new global model.

TFF currently only allows developers to simulate custom federated training plans and design novel federated algorithms. On the other hand, OpenFL is a production-ready framework that has been actively deployed in real-life applications. Therefore, since this project aims to develop a working prototype using FL, OpenFL is chosen as the FL framework instead of TFF.

### **3.4 Data Preparation**

The images used in the dataset are extracted from the “Crisis Image Benchmarks Dataset” – the consolidated dataset for benchmarking mentioned earlier. The sub-dataset for disaster types is used to train both the disaster classification task and victim detection task. The class label for disaster type in the sub-dataset include (i) fire, (ii) hurricane, (iii) flood, (iv) earthquake, (v) landslide, (vi) other disasters (to cover the remaining disaster types, such as plane crashes) and (vii) not disaster. The distribution of the dataset across the classes is listed in Table 3.3.



Table 3.3: Class Distribution for the Training, Validation and Test for Disaster Type Sub-Dataset.

<b>Class Labels</b>	<b>Train</b>	<b>Validation</b>	<b>Test</b>
<b>Fire</b>	1270	121	280
<b>Hurricane</b>	1444	175	352
<b>Flood</b>	2336	266	599
<b>Earthquake</b>	2058	207	404
<b>Landslide</b>	940	123	268
<b>Other Disaster</b>	1132	143	302
<b>Not Disaster</b>	3666	435	990
<b>Total</b>	12846	1470	3195

There is a lack of open-sourced victim detection datasets available on the internet. Thus, the disaster type dataset will be annotated for victim detection. Due to time constraints, the dataset is labelled using the auto-annotation method. The tool used is Auto-Annotate, which is built on top of a Mask R-CNN model. The pre-trained Mask R-CNN model has been forked 1400 times in GitHub, making it a reliable model. In total, 5994, 634 and 1448 images from the training, validation and testing dataset, respectively, are labelled. The remaining images are not used because there is no victim in the images.

### 3.5 Model Architecture

An MTL model is required for the two tasks for this project, which are (i) disaster classification as well as (ii) victim detection and counting. The MTL model used in this project will use the hard parameter sharing approach, where the backbone network is shared for all tasks. A suitable object detector model will be selected. Then, an additional head model will be added for disaster classification.

#### 3.5.1 Object Detector Selection for Victim Detection

An object detection model is required for the victim detection and counting task. There are two types of object detection models, which are two-stage detectors and one-stage detectors. For two-stage detectors, the detection

process is separated into the region proposal and the classification stage. These models will first propose several object candidates, which is known as regions of interest (RoI). In the second step, each proposal is classified, and its localization is regressed. Examples of two-stage detectors are the Region-Based Convolutional Neural Networks (R-CNN) family. On the other hand, one-stage detectors contain a single feed-forward fully convolutional network that directly provides the bounding boxes and the object classification. One-stage detectors perform classification and regression on dense anchor boxes without generating the RoI. Notable one-stage detectors include the You Only Look Once (YOLO) and Single-Shot Detector (SSD) variants.

Generally, two-stage detectors tend to have higher accuracy than one-stage detectors due to the higher computational cost. However, one-stage detectors are usually faster because they have a lower computational cost. The model developed for this project is meant for real-time applications on edge devices, which are computationally constrained. Thus, a one-stage object detection model is preferred for this application. There are various forms of one-stage object detection models, such as Single Shot Detector (SSD) variants and also YOLO variants. Generally, YOLO variants are much faster compared to other forms of one-stage detectors. Thus, YOLO is the preferred object detection model for the scope of the project.

YOLO variants are usually named with YOLOvX, which stands for YOLO version X. The original YOLO (hereinafter referred to as YOLOv1) was introduced by Redmon et al. in 2015. It was the first object detection network to combine the bounding boxes and class labels in one evaluation (Redmon et al., 2015). YOLOv1 is the first object detector to process images in real-time, with an inference rate of 45 frames per second (FPS) on a graphic processing unit (GPU). YOLOv1 is faster than Faster R-CNN with Zeiler-and-Fergus (ZF) backbone, which is the fastest detector at the time with 18 FPS. However, it is less accurate than the most accurate detector at the time, which is Faster R-CNN with a VGG-16 backbone. The mean accuracy precision (mAP) of the two models on the Visual Object Classes (VOC) 2007 dataset are 63.4 and 73.2, respectively.

YOLO family has continued to receive updates ever since its initial release in 2015. Redmon and Farhadi published YOLOv2 in 2016, where

several modifications were built on top of the original YOLOv1. Concepts including batch normalization and anchor boxes are adopted into the architecture of YOLOv2 (Redmon and Farhadi, 2016). YOLOv2 outperforms all R-CNN models and the original SSD. At the time, YOLOv2 with a resolution of 480 x 480 is the fastest and most accurate object detector, with 78.6 mAP on the VOC 2007 dataset and an inference rate of 59 FPS.

YOLOv3 is the last YOLO contributed by Redmon, which is released in 2018. YOLOv3's backbone adopts the residual network ideas from the Residual Neural Network (ResNet), which adds residual connections between the layers. YOLOv3 also adopts the FPN as its neck model to extract features at three different scales. Three head models are used to predict boxes at the three different scales. Previous YOLO incarnations often struggled with small objects. By adopting multi-scale prediction, YOLOv3 can detect small objects more accurately. Although YOLOv3 is not the most accurate detector, it is still faster than any detectors present at the time (Redmon and Farhadi, 2018). YOLOv3 with resolution 608 x 608 has an accuracy of 57.9 mAP, following closely behind Faster R-CNN (with FPN) with 59.1 mAP. The same YOLOv3 slightly outperforms RetinaNet-101 with a resolution of 800 x 800, which has 57.5 mAP.

YOLOv4 is developed by Bochkovskiy, Wang and Liao (2020). The research work tested various combinations of features that could improve the performance of object detectors. For instance, the authors experimented with different architectures for the (i) backbone, (ii) neck and (iii) head models for YOLO. After practical testing, YOLOv4 uses Cross Stage Partial Networks (CSP), which is based on DenseNet, as its backbone. It also replaces the FPN neck in YOLOv3 with Path Aggregation Network (PA-Net). YOLOv4 is much of the YOLOv3 architecture in general. YOLOv4 is faster and more accurate than all detectors at the time of publication (Bochkovskiy, Wang and Liao, 2020). It has an inference rate of ~65 FPS on Tesla V100, and an mAP of 65.7.

On the other hand, YOLOv5 was released by a company called Ultralytics in its GitHub repository. Ultralytics claimed YOLOv5 to be state of the art among all YOLO variants. However, its legitimacy remained questionable in the computer vision community (Kanjee, 2020; Meel, 2021).

Firstly, YOLOv5 does not introduce any novel improvements to the YOLO architecture. YOLOv5 still uses the same backbone and neck used in YOLOv4, which are the CSP and PA-Net, respectively. Thus, YOLOv5 is not considered a novel implementation to deserve the “version 5” title. In addition, YOLOv5 was not published in peer-reviewed research papers that supported its architecture and performance. At the time of writing, YOLOv5 is still yet to be published in any formal research paper. Furthermore, YOLOv5 has unvalidated data backing its improvement over YOLOv4. It is not tested on the generally accepted Microsoft Common Objects in Context (COCO) dataset for benchmark tests (Meel, 2021).

YOLOv1 and YOLOv2 are very much outdated. Meanwhile, YOLOv5 is not a novel implementation compared to YOLOv4, and is yet to be supported by any peer-reviewed paper. While YOLOv4 is often considered the state-of-the-art model, its general architecture remains similar compared to YOLOv3. In 2021, there is still much research focusing on designing a lightweight and faster YOLOv3 (Zhang and Fan, 2021; Shi et al., 2021; Zhang, Li and Zhang, 2021; Zheng, Zhao and Li, 2021). On the contrary, there is limited research work on lightweight YOLOv4. A possible explanation is that most lightweight YOLOv3s are designed by modifying the backbone of the model. However, the main unique feature of YOLOv4 is its backbone, while the rest of the model architecture resembles YOLOv3. If the backbone for YOLOv4 is changed, then the modified architecture would essentially be a YOLOv3 variant. For this project, a YOLOv3 will be developed for victim detection.

Originally in YOLOv3, the outputs of the three head models are the feature maps with the spatial information and class labels of the detected objects. Ideally, the model should only predict one bounding box for each detected object. However, an object detector will likely predict more than one bounding box for each object. Thus, non-max suppression (NMS) is applied to remove the redundant bounding boxes. After applying NMS, the remaining bounding boxes are theoretically the best bounding boxes for the detected object. Figure 3.3 illustrates the removal of duplicated bounding boxes using NMS.

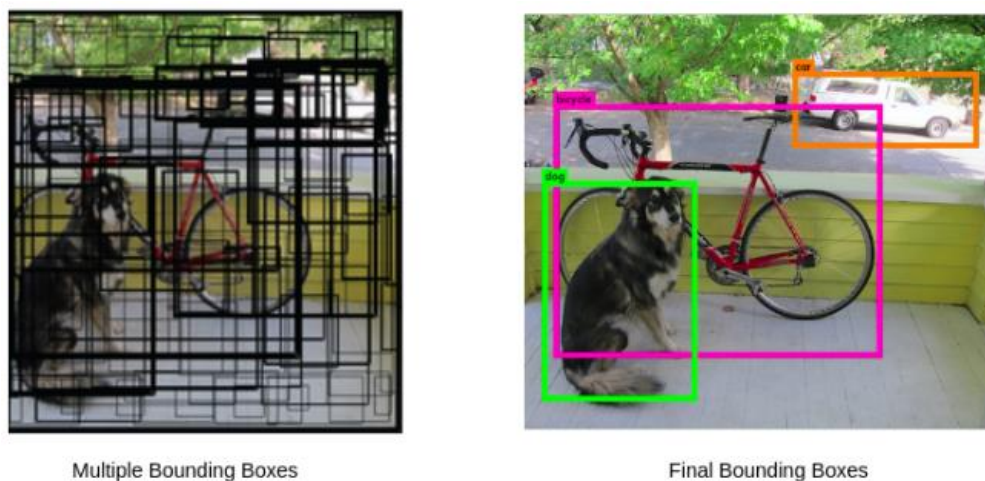


Figure 3.3: Illustration of Bounding Boxes Removal via Non-Max Suppression (Redmon et al., 2015).

Since the object detector for this project is only for victim detection, all the bounding boxes generated are for the detected victim. Therefore, the victim counts can be predicted based on the number of bounding boxes that remained after NMS. Victim counts will be returned as a tensor from the modified YOLOv3. When deployed in an IoT framework, only the victim counts have to be returned by the smart camera to the IoT server. Figure 3.4 shows the general architecture for the modified YOLOv3.

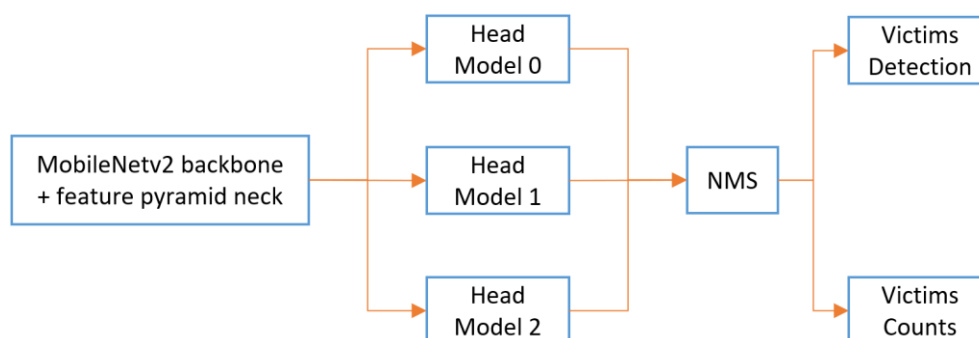


Figure 3.4: The Modified Architecture of YOLOv3.

### 3.5.2 Head Model for Disaster Classification

A head model for disaster classification will be added to the victim detection model. In order to minimise the computation cost, the head model will be based on a MobileNet-like architecture. The main highlight of

MobileNet architecture is its adoption of depthwise separable convolutions as the feature extraction layers. The regular convolution layer is four-dimensional, where the convolution kernel ( $K$ ) has a size of  $D_k \times D_k \times M \times N$ , where (i)  $D_k$  is the spatial dimension of the kernel, (ii)  $M$  is the number of input channel, and (iii)  $N$  is the number of output channels defined previously. The computation cost of the regular convolution will be  $D_k \times D_k \times M \times N \times D_f \times D_f$ , where  $D_f$  is the spatial width of the convolution output. Figure 3.5 shows the illustration of regular convolution.

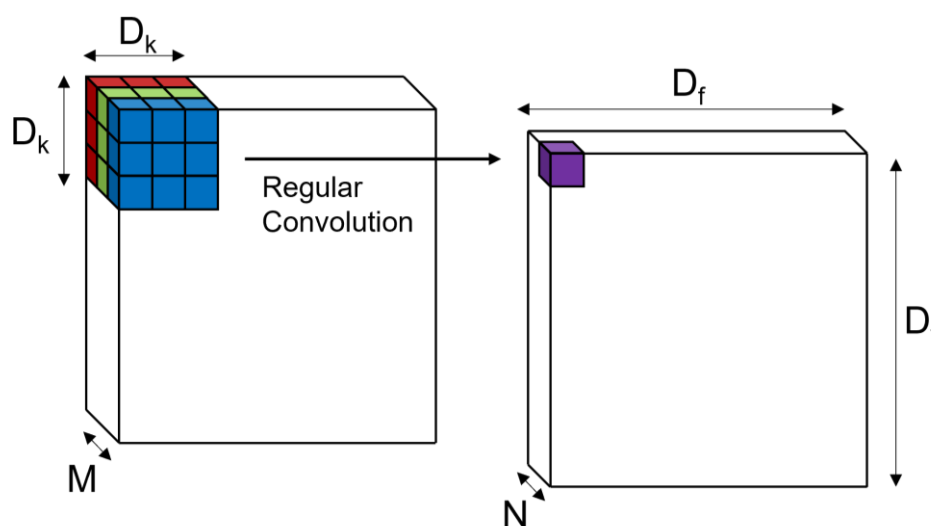


Figure 3.5: Illustration of the Working Principle of Regular Convolution. The Spatial Information in the  $D_k \times D_k \times M$  Region is Convolved at Once. Since the Output Has  $N$  Channels, the Region will be Convolved  $N$  Times. The Entire Process will be Repeated  $D_f \times D_f$  Times, since the Output has Spatial Width of  $D_f \times D_f$ .

On the other hand, MobileNet adopts depthwise separable convolutions in its architecture. Depthwise separable convolution comprises two layers, which are the (i) depthwise convolution and (ii) pointwise convolution (Howard et al., 2017). In depthwise convolution, the convolution is performed layer by layer, instead of the whole volume at once. Thus, the kernel size is reduced to  $D_k \times D_k \times 1 \times M$ . Depthwise convolution is then followed by pointwise convolution, which is a simple  $1 \times 1$  regular convolution. A pointwise convolution is applied to create a linear combination of the output of the depthwise layer. Together, the two layers

mimic the traditional convolution layer with lesser parameters. The total computation cost of a depthwise separable convolution is  $(D_k \times D_k \times M + M \times N) \times D_f \times D_f$ . Figure 3.6 shows the illustration of depthwise separable convolution.

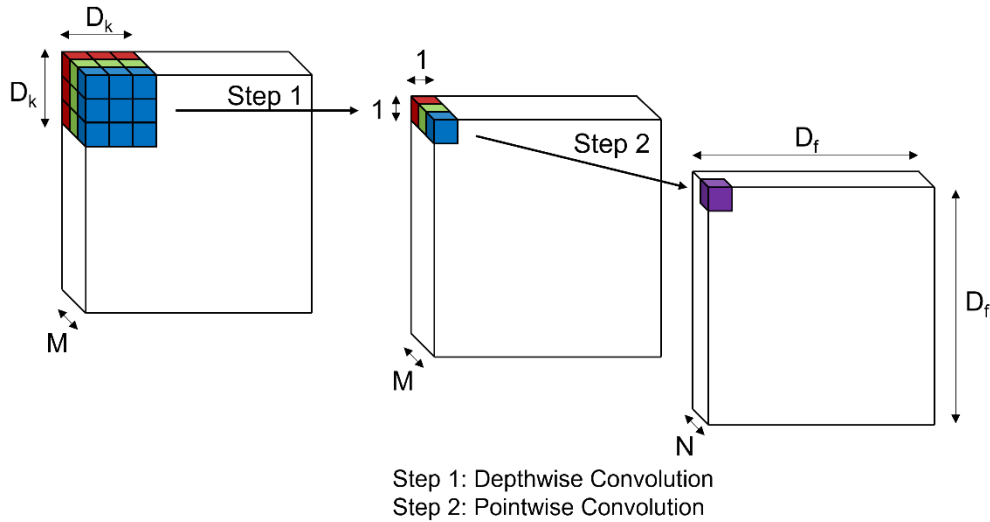


Figure 3.6: Illustration of Depthwise Separable Convolution. The Spatial Information in  $D_k \times D_k \times M$  Region will be Convolved Separately  $M$  Times, Resulting in the  $1 \times 1 \times M$  Region in the Intermediate Output. The  $1 \times 1 \times M$  Region will be Convolved at Once via Regular Convolution. Since the Output has  $N$  Channels, the  $1 \times 1 \times M$  Region will be Convolved  $N$  Times. The Entire Process will be Repeated  $D_f \times D_f$  Times, because the Output has Spatial Width of  $D_f \times D_f$ .

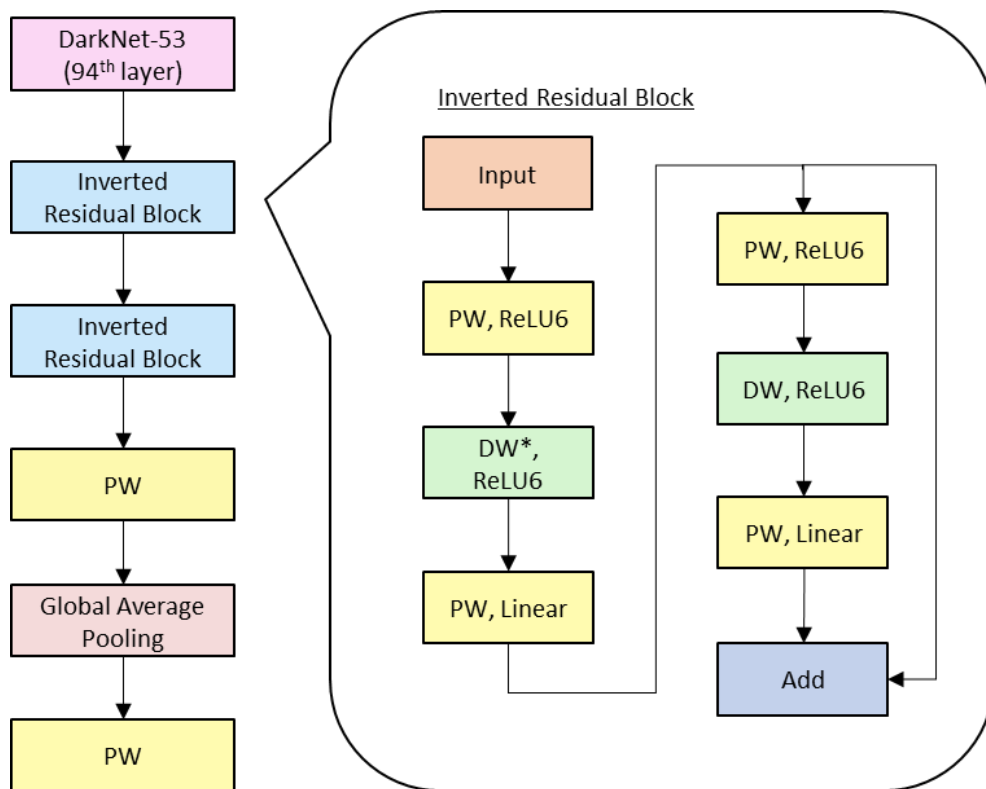
The total reduction in the computing cost using depthwise separable convolution can be expressed in Equation 3.1. MobileNet uses  $3 \times 3$  depthwise separable convolutions, which means the  $D_k$  value in Equation 3.1 is three. In general, the total computation cost is between one eighth to one-ninth of the regular convolution, with minimal sacrifice of accuracy (Howard et al., 2017). Thus, MobileNet is the go-to network when computation cost is a concern.

*Computation Reduction (in Ratio)*

$$\begin{aligned}
&= \frac{\text{Computation Cost of Depthwise Separable Convolution}}{\text{Computation Cost of Regular Convolution}} \\
&= \frac{(D_k \cdot D_k \cdot M + M \cdot N) \cdot D_f \cdot D_f}{D_k \cdot D_k \cdot M \cdot N \cdot D_f \cdot D_f} \\
&= \frac{D_k \cdot D_k \cdot M + M \cdot N}{D_k \cdot D_k \cdot M \cdot N} \\
&= \frac{(D_k \cdot D_k + N) \cdot M}{D_k \cdot D_k \cdot M \cdot N} \\
&= \frac{D_k \cdot D_k + N}{D_k \cdot D_k \cdot N} \\
&= \frac{1}{N} + \frac{1}{(D_k)^2} \tag{3.1}
\end{aligned}$$

For disaster classification task, the head model comprises a feature extraction layer and output classification layer. The feature extraction layer is based on the depthwise-separable convolution block. It is used to extract higher-level features for disaster classification. Lastly, the output classification layer is a pointwise layer with seven output nodes since there are seven label classes. The activation function of the last layer is softmax for multiclass classification. Figure 3.7 illustrates the architecture of the disaster classification head model.





All convolution layers have stride=1, except \* (where stride=2 is used)

Figure 3.7: The Architecture for the Disaster Classification Head Model.

### 3.5.3 The Architecture of the Unified MTL Model

Together, the YOLOv3 and disaster classification head model form an MTL model for victim detection/counting and disaster classification. Figure 3.8 shows the illustration of the architecture of the MTL model.

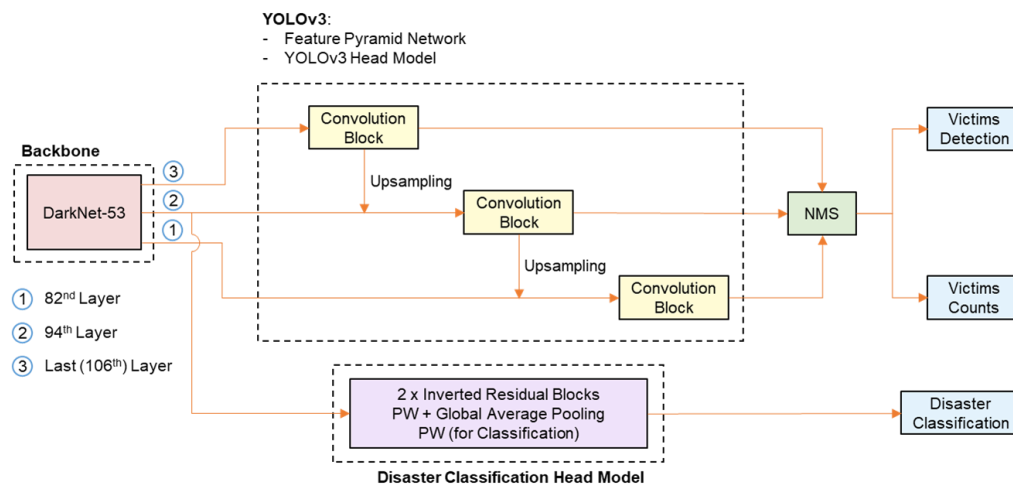


Figure 3.8: The Architecture of the Unified MTL Model for Disaster Classification and Victim Detection.

### 3.6 Training the MTL Model

The training of the MTL model can be divided into two parts: the (i) training of the victim detection head model and the (ii) training of the head model for disaster classification.

#### 3.6.1 Training the Victim Detection Head Model

YOLOv3 has a considerably large size. In total, the YOLOv3 used in this project has 61,576,342 parameters. Therefore, a significant amount of memory has to be allocated to the model. The remaining memory will determine the maximum batch size that could be used for training. The CPU and GPU memory in Colab and the workstation used in this project are insufficient to train the YOLOv3 variants in a batch size of more than eight. However, a batch size of eight is not optimal, as most training of CNN usually requires at least a batch size of 32.

To overcome this problem, each sample batch can be split into smaller mini-batches. Each mini-batch should be small enough, where the GPU memory can be satisfied. These mini-batches will be run independently, and their gradients should be averaged or summed before updating the model parameters. There are two main ways to implement this approach, which are (i) data-parallelism and (ii) gradient accumulation.

Data-parallelism requires multiple processing units (either CPUs or GPUs, although GPUs are usually preferred) in a device, where each processing unit is used to train all mini-batches in parallel. The gradients for all mini-batches are computed at once, and will be averaged and used to update the model parameters. Figure 3.9 illustrates the working principle of data-parallelism. On the other hand, gradient accumulation requires only one processing unit. Each mini-batches are trained sequentially by the same processing unit, where the computed gradients will be accumulated. After accumulating all gradients, the accumulated gradients will be averaged. Figure 3.10 shows the working principle of gradient accumulation.

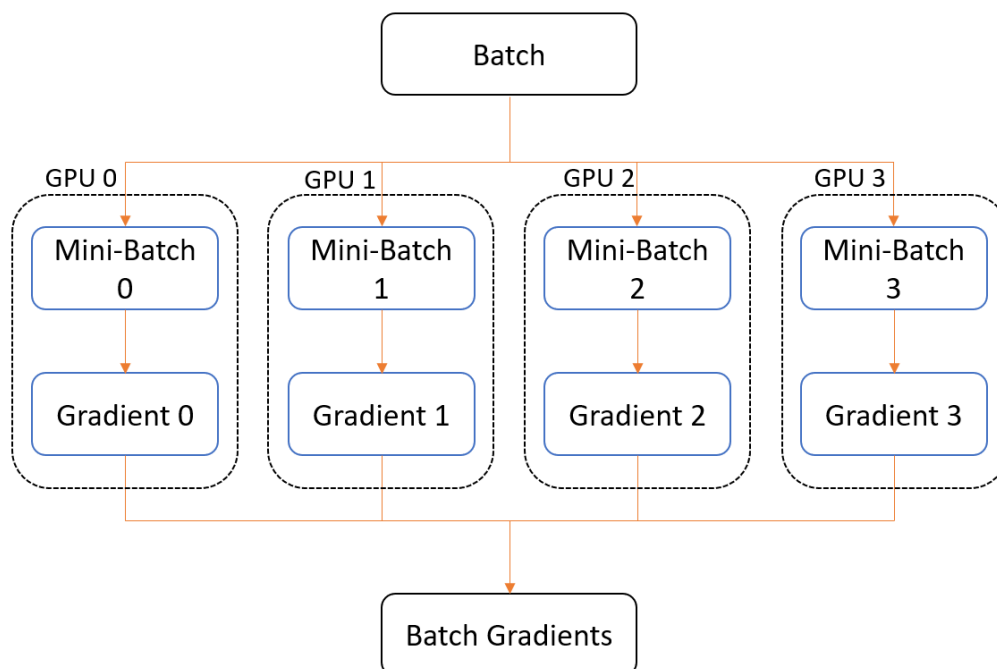


Figure 3.9: Illustration of Data-Parallelism.

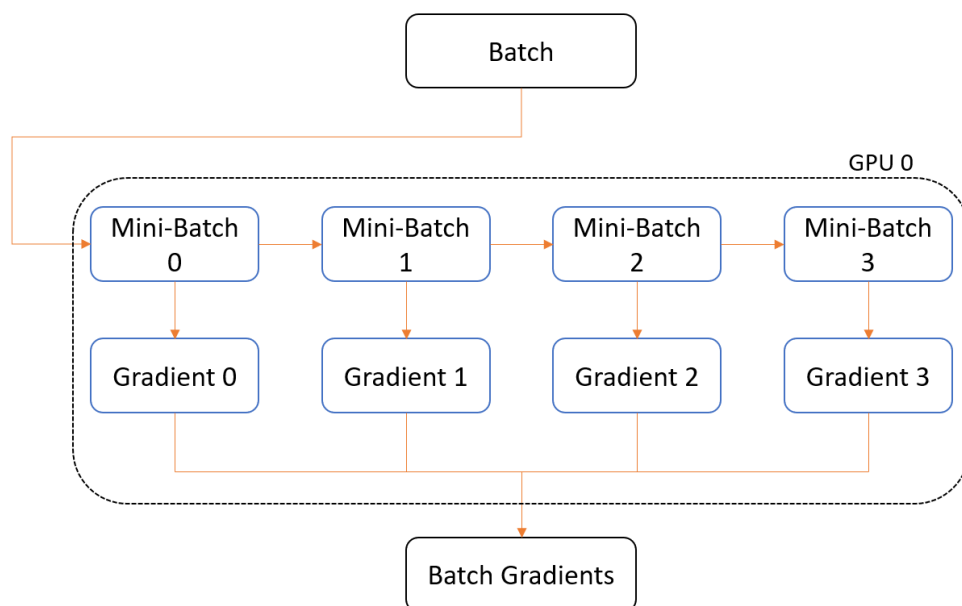


Figure 3.10: Illustration of Gradient Accumulation.

Data-parallelism cannot be implemented in this project, since both Colab and the workstation used only have one GPU. Thus, gradient accumulation will be used in the training of the model. The batch size used is 64, where each batch will be divided into eight mini-batches. Each mini-batches then consists of eight training samples. Since TensorFlow does not support gradient accumulation, custom training codes have to be written.

Since a high degree of customisation is required for training, the codes are written for TensorFlow eager mode execution (refer to Section 3.3.2) for detailed explanation). To update the model parameters manually, the GradientTape object in TensorFlow framework will be used. For each mini-batch, the gradients will be computed using the gradient() method in the GradientTape object. After accumulating the gradients for eight mini-batches, the averaged gradient resembles the gradient using a batch size of 64. The averaged gradient is used to update the model's parameter via apply\_gradient() method in the GradientTape object.

The victim detection model will be trained via two methods, which are CL and FL, both using gradient descent to overcome the memory limitation problem.

#### Training Victim Detection Head Models using CL

The pretrained YOLOv3 weights by Redmon and Farhadi (2018) will be used for transfer learning. The Darknet-51 backbone and FPN neck will be reused to build the YOLOv3 for victim detection. The weights for the three head models for object localisation and classification will be initialised randomly. The head models will initially be trained for 10 epochs with 64 batch sizes and a learning rate of  $1 \times 10^{-5}$ . This preliminary training is to warm up the head models before intensive training. After that, the head models will be trained for another 10 epochs with a learning rate of  $1 \times 10^{-4}$ . Figure 3.11 shows the pseudocode to train the victim detection head model using CL.

---

**Algorithm 1** Train the YOLOv3 using Centralized Learning
 

---

**Input:** Labeled dataset,  $\mathcal{L}$ ;  
 Number of Epoch,  $\mathcal{N}$ ;  
 Mini batch gradient accumulate round,  $\mathcal{B}$ ;  
 Learning rate,  $\alpha$ ;  
 Victim Detection Model,  $\theta$ ;  
 YOLOv3 Loss Function,  $\mathcal{J}$ ;

**Output:** Trained Victim Detection Model,  $\theta$ ;

```

 $\mathcal{L}_0$  is divided into mini batches of data,  $\mathbf{l}$ 
Freeze the Backbone and FPN of YOLOv3
for t=1:  $\mathcal{N}$  do
  Counter:  $c \leftarrow 0$ 
  Accumulated Gradients:  $\nabla_{accumulate} \leftarrow 0$ 
  for  $\mathbf{l}$  in  $\mathcal{L}$  do
    // Compute & accumulate gradients
     $\nabla \leftarrow \frac{\partial}{\partial x} \mathcal{J}(\theta_t, \mathbf{l})$ 
     $\nabla_{accumulate} \leftarrow \nabla_{accumulate} + \nabla$ 
     $c \leftarrow c + 1$ 
    // Update model
    if  $c \bmod \mathcal{B} = 0$  then
       $\nabla_{accumulate} \leftarrow \nabla_{accumulate} / \mathcal{B}$ 
       $\theta_{t+1} \leftarrow \theta_t - \alpha \nabla$ 
       $\nabla_{accumulate} \leftarrow 0$ 
    end if
    // Decrease Learning Rate by 10
    if  $\mathcal{N} \bmod 10 = 0$  then
       $\alpha \leftarrow \alpha / 10$ 
    end if
  end for
end for
end for

```

---

Figure 3.11: Pseudocode to Train the Victim Detection Head Models using Centralised Learning.

### Training the Victim Detection Head Models using FL

The victim detection head model will be trained using FL on 2-device and 3-device setups. Firstly, the training dataset will be divided equally on each training device. Then, an FL server is initialised, and each training device will connect to the server as a client. The server will instantiate a global copy of the victim detection model, and send it to each client. In each communication round, each client will train their copy of the model using the training strategy similar to CL. Then, the FL server will collect the model weights from each client, aggregating them into a new global model using the

Federated Averaging (FedAvg) algorithm proposed by McMahan et al. (2016). The entire process will be repeated for 20 rounds. Figure 3.12 shows the pseudocode to train the victim detection head model using FL.

---

**Algorithm 2** Train the YOLOv3 using Federated Learning

---

**Input:** Victim Detection Model,  $\theta$ ;  
 Number of Epoch,  $\mathcal{N}$ ;  
 Total Number of Clients,  $K$

**Output:** Trained Victim Detection Model,  $\theta$ ;

**Server executes:**  
 Initialize a global model,  $\theta^{global}$   
 for  $t=1: \mathcal{N}$  do  
   **Operations on the server side:**  
   // Select a fraction of Clients,  $C$   
    $m \leftarrow \max(C \cdot K, 1)$   
    $S_t \leftarrow \{\text{random set of } m \text{ clients}\}$   
   // Train each selected client,  $\theta^k$   
   for each client  $k \in S_t$  in parallel do  
      $\theta_{t+1}^k \leftarrow \text{ClientUpdate}(\theta_t^{global})$   
   end for  
   // Update the global model  
    $\theta_{t+1}^{global} \leftarrow \sum_{k=1}^K \frac{n_k}{n} \theta_{t+1}^k$   
 end for

**ClientUpdate( $\theta^{global}$ ):**  
 $\theta \leftarrow \theta^{global}$   
 Update  $\theta$  using training strategy shows in Algorithm 1  
 return  $\theta$

---

Figure 3.12: Pseudocode to Train the Victim Detection Head Models using Federated Learning.

### 3.6.2 Training the Disaster Classification Head Model

The head model for disaster classification is relatively small compared to the FPN neck and head models of YOLOv3. Thus, there are no GPU memory constraints, and a larger batch size can be used for training. There is no customization required for the training loop, and the model can be trained by the existing fit() method in TensorFlow. Hence, the training is conducted in TensorFlow graph mode, which is more efficient than the eager mode.

The training of the disaster classification head model will be performed via CL, FL, and AL-based FL (hereinafter referred to as AL-FL) methods.

#### Training Disaster Classification Head Model using CL

For the CL method, the head model will be trained using the Cosine Decay strategy. The initial learning rate is set as  $5 \times 10^{-3}$ , which will be decayed by a factor of 100 after 40 epochs of training. Meanwhile, the batch size is set to 32. Theoretically, a larger batch size can be used to train the disaster classification head model, since we have sufficient GPU memory. However, the batch size is kept at 32 (which is usually the default batch size) since we intend to train the model in edge devices which may have computation limitations.

#### Training Disaster Classification Head Model using FL

The FL strategy used to train the disaster classification head model is similar to the one used to train the victim detection model. The disaster classification head model is trained on 2-device and 3-device setups, where the training dataset is divided equally among the training devices. OpenFL is used as the FL framework, and the training strategy used to train the model for each client is similar to the CL method.

#### Training Disaster Classification Head Model using AL-FL

There are two types of AL-FL strategy, which are the online and offline AL-FL. In the former strategy, the AL task is conducted in each communication round of FL, meaning that both AL and FL are performed simultaneously. The latter strategy would keep the AL task offline, where the FL phase will start only after all data in each client has been labelled after the AL phase.

The AL-FL strategy used in this project is the offline AL method. By doing so, the communication rounds in FL could be reduced since the query phase usually requires a large amount of AL rounds (Ahmed et al., 2020). Besides, the FL phase could be deployed smoother when AL is separated from FL, since all the required data in each client have been labelled beforehand. On the other hand, it would be user-unfriendly to wait

for human annotators to label the samples query by AL strategy between each communication round in an online AL-FL strategy. Thus, it is decided that offline AL will be more suitable for the scope of the project.

Given a pool of unlabelled datasets, a sampling technique is required to query samples from the pool for labelling. The sampling technique used in this project is margin sampling. It aims to pick samples with the smallest difference between the probabilities of the two most probable classes. The intuition of this strategy is that: if the classification model is confident in its prediction, it should assign most of the probability to the correct class. On the other hand, if the classification model is not confident, the probability would be distributed among a few classes. Thus, a large difference between the probabilities of the two most probable classes would infer that the model is struggling to make the correct prediction. This input data would be treated as a hard sample that the model will query using margin sampling, where the human annotator could label it.

In this project, several variations of the margin sampling technique are proposed to improve the performance. Instead of sampling only the hard samples, samples that are trivial (easy) and moderately hard to the model can also be sampled. The proposed method will be analysed in Section 4.3. In the proposed AL-FL strategy, each client initialises a victim detection model, and trains the model using a small labelled dataset. Then, 20 rounds of AL cycle are deployed to each client. Each client will sample 32 samples in each round comprised of easy, moderately-hard and hard images. After labelling, the labelled queries are added to the initial training dataset, where the model will be fine-tuned using the expanded dataset for one epoch. Figure 3.13 shows the flowchart for the proposed active learning process.



---

**Algorithm 3** The Proposed Active Learning Process
 

---

**Input:** Disaster Classification Model,  $\theta_0$ ;  
 Number of Active Learning Round,  $\mathcal{N}$ ;  
 Small Labeled dataset,  $\mathcal{L}_0$ ;  
 Unlabeled dataset,  $\mathcal{U}_0$ ;  
 Uncertainty Function,  $\mathcal{F}_{unc}$ ;  
 Query Batch Size,  $\mathcal{K}_{easy}, \mathcal{K}_{mod}, \mathcal{K}_{hard}$ ;

**Output:** Labeled dataset,  $\mathcal{L}_t$ ;

**Each Client executes:**

```
// Phase 1: Warm Up the Model
Initialize model,  $\theta$ 
 $\theta_0 \leftarrow$  train  $\theta$  for 5 epochs using  $\mathcal{L}_0$ 
for t=0:  $\mathcal{N}$  do
  // Phase 2: Query Selection
   $\mathcal{Q} \leftarrow \mathcal{F}_{unc}(\mathcal{U}_t, \theta_t)$ 
   $\mathcal{Q}_{t+1}^{easy} \leftarrow \{\mathcal{U}_{t,i} \mid i \in \underset{unc}{\text{argbtm}}K(\mathcal{Q}, \mathcal{K}_{easy})\}$ 
   $\mathcal{Q}_{t+1}^{mod} \leftarrow \{\mathcal{U}_{t,i} \mid i \in \underset{unc}{\text{argmid}}K(\mathcal{Q}, \mathcal{K}_{mod})\}$ 
   $\mathcal{Q}_{t+1}^{hard} \leftarrow \{\mathcal{U}_{t,i} \mid i \in \underset{unc}{\text{argtop}}K(\mathcal{Q}, \mathcal{K}_{hard})\}$ 
   $\mathcal{Q}_{t+1} = \mathcal{Q}_{t+1}^{easy} \cup \mathcal{Q}_{t+1}^{mod} \cup \mathcal{Q}_{t+1}^{hard}$ 
  // Phase 3: Sample Annotation
   $\mathcal{Y}_{t+1} \leftarrow \text{annotate } \mathcal{Q}_{t+1}$ 
  // Phase 4: Update Model
   $\mathcal{L}_{t+1} \leftarrow \mathcal{L}_t \cup \{(X, \mathcal{Y}) \mid X \in \mathcal{Q}_{t+1}, \mathcal{Y} \in \mathcal{Y}_{t+1}\}$ 
   $\theta_{t+1} \leftarrow$  fine-tuning  $\theta_t$  using  $\mathcal{L}_{t+1}$ 
end for
return  $\mathcal{L}_{t+1}$ 
```

---

Figure 3.13: Pseudocode of the Proposed Active Learning Process.

After 20 rounds of AL cycle, each device would have labelled two-thirds of the entire dataset in each client. Then, a global model will be instantiated, and the FL training will begin using the new global model and the expanded dataset for each client.

### **3.7 Conversion of the Multi-Task Model into Intermediate Representation (IR) to load in Inference Engine (IE)**

Model Optimiser (MO) is a tool provided in OpenVINO that facilitates the transition between training and deployment environments. It takes a deep learning model trained using some famous frameworks, including TensorFlow, to convert it to an Intermediate Representation (IR). Besides, MO performs static model analysis, adjusting the models for optimal execution on the target devices. In this project, the MO is used to (i) add pre-processing layers as sub-graphs into the converted model, and (ii) compress the model from single-precision floating-point (FP32) to half-precision floating-point (FP16) and 8-bit integer type (INT8) format. The former model adjustment allows the pre-processing layers of the input data as part of the multi-task model itself. The latter adjustment decreases the model size by half, and allows higher inference speed.

After converting the multi-task model into IR, the IR files are loaded into the Inference Engine (IE). Like the MO, which optimises the model based on the complexity of the models to improve memory and computation times, the IE on the other hand provides hardware-based optimisations. The supported devices for the IE includes all Intel hardware, including CPU, GPU, neural compute stick (NCS-2), and field-programmable gate array (FPGA). In this project, the IE will optimise the models for Intel CPU.

### **3.8 Summary**

A YOLO-based multi-task model for joint disaster classification and victim detection is designed. TensorFlow and OpenFL will be used as the deep learning and FL framework for this project. Gradient accumulation will be used to solve the memory insufficiency problem. The victim detection head model will be trained using CL and FL, while the disaster classification head model will be trained CL, FL, and AL-FL. Lastly, the trained multi-task model will be optimized using the OpenVINO toolkit, and then deployed in Intel CPU hardware. Figure 3.14 illustrate the complete methodology of the proposed system.

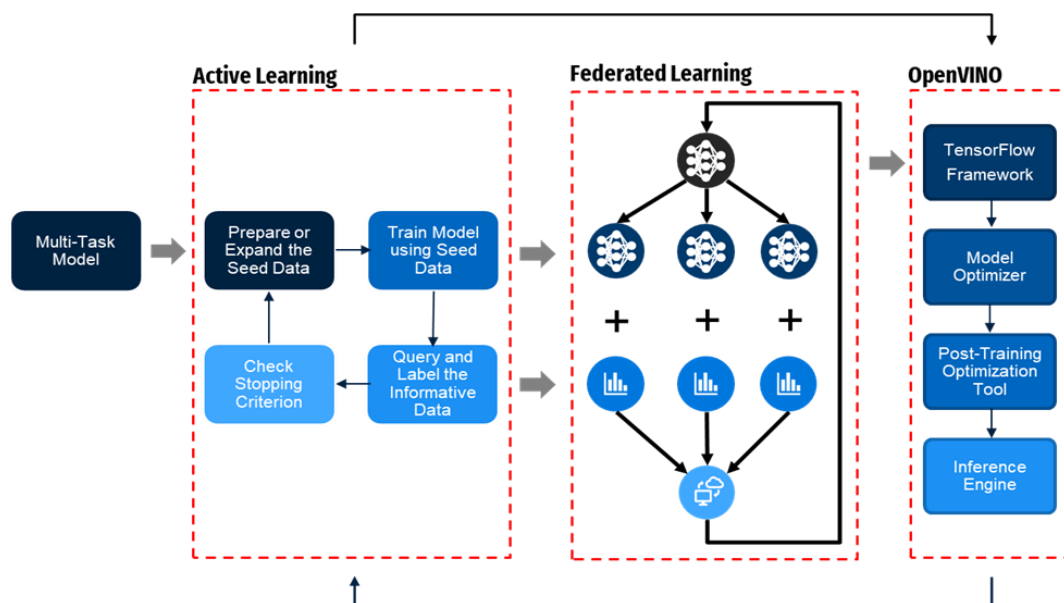


Figure 3.14: The Methodology of the Proposed System.

## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Introduction

The performance of the multi-task model will be evaluated in two parts. Section 4.2 will discuss the model's performance on the victim detection task, while Section 4.3 will cover the model's performance on the disaster classification task. The overall designed pipeline for this project will be covered.

#### 4.2 Victim Detection

Since the victim detection models are trained with a custom dataset, there is no benchmark to compare the performance of the trained models. The result analysis for this task will mainly compare the CL-trained and FL-trained models.

The performance of a single-class object detection model can be evaluated using accuracy precision (AP). Figure 4.1 shows the PR curve for the victim detection head model. AP is derived from precision and recall. The definition of precision (P) and recall (R) are as shown in Equation 4.1 and Equation 4.2, where TP, FP and FN are true-positive counts, false-positive counts, and false-negative counts, respectively. Based on P and R, the AP can be expressed as the integral of function P of R, as shown in Equation 4.3. In other words, AP is the area under the PR curve.

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

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

$$AP = \int_0^1 P(R) dR \quad (4.3)$$

The CL-trained victim detection head model achieves train, validation and test AP of 0.7814, 0.6907 and 0.6938, respectively. The PR

curve of the CL-trained victim detection head model is shown in Figure 4.1. The model is considered robust as it could reach a high test AP of 0.6938, considering that a large portion of the detected victim is small or partially occluded. Besides, the model has a high inference speed on NVIDIA RTX 2070 SUPER, with a 20.31 frame per second (FPS) score. This high inference speed is expected, as YOLO models are famous for being fast.

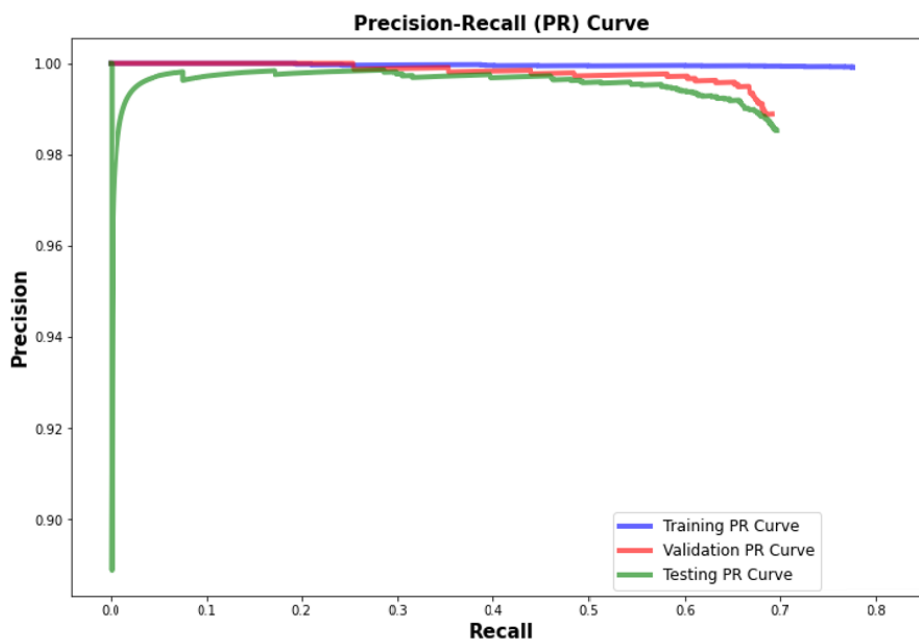


Figure 4.1: The Precision-Recall Curve for Victim Detection Head Model.

For the victim detection task, the FL models trained using 2-client and 3-client have a test AP score of 0.5902 and 0.5417. Both FL-trained models do not outperform the CL-trained model. However, both models outperform models trained solely on local datasets on any of the clients. Under a 3-client setup, any model trained using any local dataset could only achieve an average test AP of 0.3998. This test AP score is 0.1419 lower than the test AP of an FL-trained model using 3 clients. Thus, FL is still highly recommended if centralising data to a data centre for CL is not possible.

Besides, the lower test AP score for FL-trained models may be due to the limited training dataset. Ahmed et al. (2020) showed that to fully exploit FL's potential, each client should have enough training dataset. The exact number of the required local dataset is highly dependent on applications. But their study shows that there is a significant increase in performance once

the amount of local training samples exceeds a certain threshold. In this project, the training dataset (before splitting into multiple local datasets at each client) is 5994. After splitting, the number of local datasets for 2-client and 3-client setups are 2997 and 1998, respectively. It is possible that more training data for each client is required to reach the threshold, to witness a performance boost. Figure 4.2 shows victim detection by the multi-task model in different scenarios.



Figure 4.2: Victim Detection at Different Areas: (a) Flood, (b) Landslide, (c) Earthquake, (d) Hurricane, (e) Fire, and (f) Other Disaster.

### 4.3 Disaster Classification

Firstly, the performance of the disaster classification head model trained via CL will be compared to the benchmarks provided by Alam et al. (2021). Then, this CL-trained head model will serve as a benchmark for the models

trained via FL and AL-FL. Table 4.1 shows the comparison of our model with the provided benchmarks.

Table 4.1: Performance of the CL-Trained Head Model with the Provided Benchmarks. Bolded Values indicate the Best Score in the Particular Metrics, while Gray-Shaded Values are the Score for the Proposed Solution.

<b>Backbone</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1 Score</b>
ResNet18	0.812	0.807	0.809	0.809
ResNet50	0.817	0.81	0.812	0.812
ResNet101	0.819	0.815	0.816	0.816
AlexNet	0.755	0.753	0.753	0.753
VGG16	0.803	0.797	0.798	0.798
DenseNet (121)	0.817	0.811	0.813	0.813
SqueezeNet	0.726	0.719	0.717	0.717
InceptionNet (v2)	0.808	0.801	0.802	0.802
MobileNet (v2)	0.793	0.788	0.793	0.789
EfficientNet (b1)	<b>0.838</b>	<b>0.834</b>	<b>0.838</b>	<b>0.835</b>
Proposed Solution	0.792	0.827	0.769	0.766

Interestingly, the proposed solution is comparable with most models that are trained specifically for disaster classification. The ability to classify disasters on top of a victim detection model comes at the cost of only a 4.6% and 1.786% accuracy drop compared to the best model (EfficientNet) and other models, respectively. As for precision, our model has the second-highest precision compared to the other models and approximates the best model within a 0.7% gap. As for recall and F1 score, the proposed solution performs slightly worse than the other models. However, it outperforms AlexNet and SqueezeNet, while approximating other models by 4.325% on average. Overall, our solution is considered robust given that it has to handle both tasks.

Since the CL-trained head model is comparable with the provided benchmarks, we will use its performance as the benchmark for the models

trained via FL and AL-FL. Table 4.2 compares the performance of the disaster classification head models trained via CL, FL and AL-FL.

Table 4.2: Comparison of the Performance of the Disaster Classification Head Models trained via CL, FL, and AL-FL. Methods labelled with an asterisk (\*) are trained using 3 FL clients.

<b>Method</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1 Score</b>
CL (all data)	0.792	0.827	0.769	0.766
CL (half data)	0.7542	0.7566	0.7524	0.7427
CL (1/3 data)	0.7318	0.739	0.7267	0.7213
FL (2 clients)	<b>0.8001</b>	<b>0.805</b>	<b>0.7941</b>	<b>0.7933</b>
FL (3 clients)	0.7964	0.8004	0.7896	0.7879
AL-FL hard*	0.7189	0.7202	0.7198	0.7201
AL-FL mod/hard*	0.7222	0.7314	0.7146	0.7396
AL-FL easy/mod/hard*	0.7673	0.7741	0.7592	0.7578

The analysis below will mainly focus on the F1 score of each model, as it is a better metric than the other three. Firstly, it is noticed that FL outperform CL in the disaster classification task in both 2-client and 3-client setups. FL with 2-client and 3-client outperform CL by 2.73% and 2.19% in F1 score, respectively. This performance increment is considered a huge improvement, as both methods used the same training strategy. Although such a performance boost is not anticipated, several research studies have shown that FL could outperform CL depending on the applications. For instance, Xiong et al. (2020) tested the FL framework on 7 drug solubility datasets and showed that the FL-trained model can always outperform models built on individual datasets. Asad, Moustafa and Ito (2020) also showed that FL outperformed CL for image classification tasks using MNIST and CIFAR-10 datasets.

The best AL-FL trained model is trained using the proposed AL strategy (which queries easy, moderately-hard and hard samples). Its performance is comparable with the CL-trained model, using only two-thirds of the local dataset in each client. It approximates the CL-trained model within 0.82%. Besides, it outperforms the original strategy that queries only



the hardest samples. As shown in Table 4.2, the F1 score experiences a 1.95% boost by querying moderately-hard samples alongside hard samples. By adding easy samples, the proposed method could improve the original strategy by 3.77%. The performance gain is a worth investigating aspect of the research, as sampling using hard samples is usually better. Based on some analysis, it is suggested that the proposed AL strategy works for this task because the decision boundary for the task is highly ambiguous. To illustrate this, t-distributed stochastic neighbour embedding (t-SNE) can be used to perform dimension reduction, mapping the features of the images in a two-dimension plot. The tensors generated before the last classification layer in the head model can be treated as the condensed features of an image, extracted by all the previous layers. Thus, all images in the test dataset are passed into the best head model (trained via FL) to extract the beforementioned tensors. Then, t-SNE is used to reduce the dimension of the tensors into a two-dimensional matrix, which can be plotted in a two-dimensional plot. Figure 4.3 shows the embeddings of the top 33% of the hardest images from the test dataset.

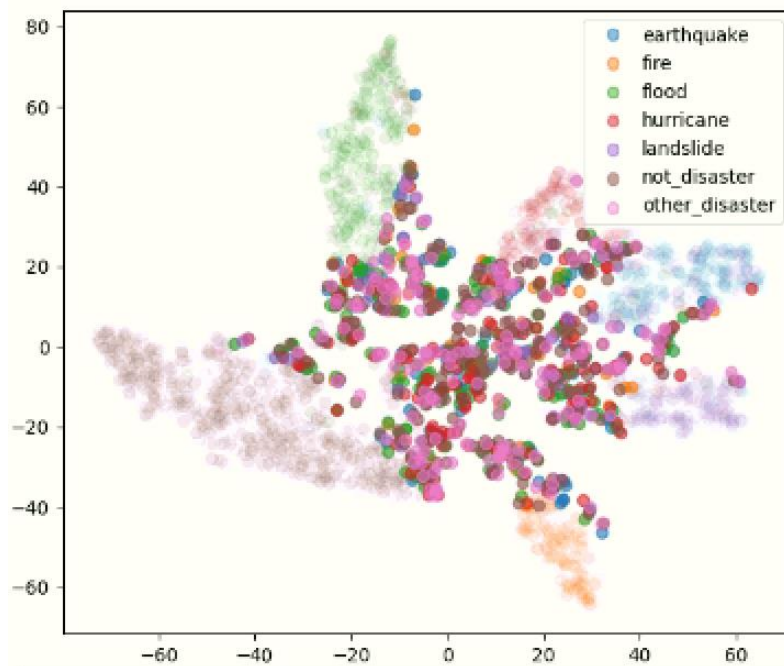






Figure 4.3: The Embeddings of the Top 33% Hardest Images in Test Dataset.

From Figure 4.3, it is observed that most of the hardest images, regardless of the classes, are almost distributed randomly in this centre of the

plot. There are no clear decision boundaries, as if the data is scattered all around vector space. However, most of the easier data are clustered nicely according to their classes, indicating an underlying distribution of the data instead of randomness. It is suggested that if the model is trained solely on the hardest data, it may not perform well on normal data. This hypothesis is backed by our result in Table 4.2, where we can see querying easier data points can boost the performance of an AL-FI-trained model. However, it is worth mentioning that this is dependent on the underlying distribution of the data. The conventional approach of querying only the hardest data may work better for applications with less ambiguous classification. The proposed method may only benefit applications with high ambiguity in labelling the data, even for human annotators. Table 4.3 shows some examples in the Crisis Benchmark Dataset to illustrate the ambiguity of the dataset. For instance, a hurricane may cause flooding and collapse some buildings, confusing the model to predict the scene as a flood or earthquake.

Table 4.3: Examples of Images in the Crisis Benchmark Dataset that Confused the Model.

Sample Image	Predicted Class	Actual Class
	Flood	Hurricane
	Earthquake	Fire

	Not Disaster	Hurricane
	Fire	Not Disaster

The FL-trained and AL-FL trained models are also superior to models trained solely on fractions of the dataset. If centralising data is not a feasible option, FL or AL-FL is the go-to method (regardless of if they are better than CL or not). Training the model solely on any local dataset could not outperform collaborative training using FL or AL-FL. Table 4.3 shows that if the model is trained solely on any of the local datasets in the three clients, the performance is inferior to training the model using all three clients via FL and AL-FL.

In short, all results show that FL and AL-FL is the go-to method if centralising the dataset for training is not feasible. The results also suggest that FL is better than CL in the disaster classification domain. Another observation is that querying easier samples during the AL phase is essential for disaster classification due to the noisy data distribution among the hardest samples.

#### 4.4 Model Optimization using OpenVINO Toolkit

In Section 4.2 and Section 4.3, the CL-trained model is used as the benchmark to compare the performance of various training methods. In this

section, the CL-trained model will be optimized using the OpenVINO toolkit, and the performance before and after optimization will be compared.

The multi-task CL-trained model will be compressed from FP32 to FP16 and INT8, as mentioned in Section 3.7. Firstly, Table 4.4 discuss the improvement in the model’s inference speed after compression. Device A, the NVIDIA RTX 2070 SUPER servers as the benchmark for the comparison. On the other hand, device B serves as the edge device which will deploy the model in the actual scenario.

Table 4.4: Comparison of Model’s Inference Speed (FPS) Before and After Optimization on Different Processing Units.

Device	Framework	Data Format	FPS
A	TensorFlow GPU (tf-gpu)	FP32	20.31
B	TensorFlow-CPU (tf)	FP32	6.55
B	OpenVINO IR Format	FP16	9.37
B	OpenVINO IR Format	INT8	16.46

Device A: NVIDIA RTX 2070 SUPER

Device B: Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz

When the multi-task model is deployed in Device A, the inference speed is as high as 20.31 FPS. If deployed in Device B, the inference drops to 6.55 FPS. Based on this, it is observed that without a powerful GPU accelerator, the inference speed will drop significantly (67.75 % in this case). Thus, the OpenVINO toolkit plays a big role in improving the model's inference speed without a powerful GPU accelerator. Based on the result in Table 4.4, it is observed that the multi-task model with FP16 and INT8 data format have an inference speed of 9.37 FPS and 16.46 FPS, respectively. The inference speed of the FP16 and INT8 models is 43.05% and 151.30% faster than the original inference speed of 6.55 FPS. This shows that model compression using OpenVINO can greatly improve the inference speed of the model.

Due to the intrinsic trade-off between inference speed and accuracy, an increase in inference speed may decrease accuracy. Thus, the next part of

this section is to analyse the trade-off of accuracy and inference speed. The evaluation metric used for the victim detection task is AP, as mentioned in Section 4.2. The evaluation metric used for the disaster classification task is accuracy only, as the OpenVINO toolkit only supports this metric as of the time of writing. The results comparison for victim detection and disaster classification task is shown in Table 4.5 and Table 4.6.

Table 4.5: Comparison of the Accuracy of the Victim Detection Head Model Before and After Compression.

Device	Framework	Data Format	Accuracy
A	TensorFlow GPU (tf-gpu)	FP32	0.6938
B	TensorFlow-CPU (tf)		
B	OpenVINO IR Format	FP16	0.6812
B	OpenVINO IR Format	INT8	0.6527

The head model has a similar AP for the victim detection task for both FP32 and FP16 formats. The slight decrease of 0.0126 in AP is expected. Although FP16 only has half the precision compared to FP32, various research projects show that it would only bring a minimal loss in accuracy (OpenVINO, 2022; TensorFlow, 2022b). On the other hand, the AP drops 0.0411 after being compressed to INT8 data format. A more significant accuracy drop for this compression is expected since it forcefully changed the data format from FP32 to INT8. However, OpenVINO will fine-tune the model while compressing to INT8 to ensure the performance drop is under control.

Table 4.6: Comparison of the Accuracy of the Disaster Classification Head Model Before and After Compression.

Device	Framework	Data Format	Accuracy
A	TensorFlow GPU (tf-gpu)	FP32	0.7920
B	TensorFlow-CPU (tf)		
B	OpenVINO IR Format	FP16	0.7931
B	OpenVINO IR Format	INT8	0.7963

For the disaster classification task, it is observed that there is no decrease in accuracy after compression. Instead, there is a slight increase in accuracy. The model's accuracy increases by 0.0011 and 0.0043 when compressed to FP16 and INT8 models. However, this is not an abnormal observation because OpenVINO uses the test dataset to tune the model during compression. This is an important step because direct compression from FP32 to FP16 and INT8 may ruin the model's performance. Only 20% of the test dataset is used for the model tuning to prevent overfitting in this setup. For this task, it is safe to conclude that the accuracy of the disaster classification head model retains after compression.

#### **4.5 Summary**

In short, the multi-task model is comparable with the provided benchmark regardless of the training method used. Both FL and AL-FL is proven to be the go-to method to train the multi-task model if centralizing the dataset for CL is not feasible. All results show that the two methods outperform a model trained solely on the local dataset from any client. Besides, an FL-trained model could outperform a CL-trained model for the disaster classification task. AL-FL is proven to be an effective method to train disaster classification head models without the need to label all data. Post-training quantization using OpenVINO could speed up the inference speed of the multi-task model without a GPU accelerator, with minimal accuracy trade-off. Together, the three components (AL, FL and OpenVINO) complete the training and deployment cycle.

## CHAPTER 5

### CONCLUSIONS AND RECOMMENDATIONS

#### 5.1 Conclusions

This project has successfully developed a multi-task model for joint disaster classification and victim detection. The multi-task model is trained using CL, FL, and AL-FL. Based on the results, it is shown that FL and AL-FL are recommended if collecting data in a centralized data centre is not possible. FL is also proven to be better than CL for the disaster classification task. While FL could not outperform CL in the victim detection task, it still provides a comparable result, given the small dataset provided for each client. It is suggested that with more data on each client, the performance of FL on victim detection task could experience a performance boost. The best model could achieve a 0.7993 F1 score and a 0.6938 AP for disaster classification and the victim detection task. OpenVINO is also shown to be capable of decreasing the computation workload of the multi-task model by half, with slight performance trade-off.

#### 5.2 Recommendations for future work

Several improvements can be made to this project. Firstly, the victim detection dataset could be expanded, since more data usually translates to better performance. A larger dataset can also guarantee that each client has enough training data for FL. Besides, an AL-FL strategy could be proposed for the victim detection task. Currently, this project only applies the AL-FL strategy to disaster classification task, and has shown its feasibility as a core part of ML Ops.

## REFERENCES

- Ahmed, L., Ahmad, K., Said, N., Qolomany, B., Qadir, J. and Al-Fuqaha, A., 2020. Active Learning Based Federated Learning for Waste and Natural Disaster Image Classification. *IEEE Access*, 8, pp.208518–208531.
- Alam, F., Alam, T., Ofli, F. and Imran, M., 2021. Social Media Images Classification Models for Real-time Disaster Response. *CoRR*, [online] abs/2104.0. Available at: <<https://arxiv.org/abs/2104.04184>>.
- Alam, F., Ofli, F. and Imran, M., 2018. CrisisMMD: Multimodal twitter datasets from natural disasters. In: *12th International AAAI Conference on Web and Social Media, ICWSM 2018*. [online] pp.465–473. Available at: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-85050637466&partnerID=40&md5=0fb528332fb3182d641214df5e854665>>.
- Alam, F., Ofli, F., Imran, M., Alam, T. and Qazi, U., 2020. Deep Learning Benchmarks and Datasets for Social Media Image Classification for Disaster Response. In: *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. pp.151–158.
- Asad, M., Moustafa, A. and Ito, T., 2020. *Federated Learning Versus Classical Machine Learning: A Convergence Comparison*.
- Baid, U., Pati, S., Kurc, T.M., Gupta, R., Bremer, E., Abousamra, S., Thakur, S.P., Saltz, J.H. and Bakas, S., 2022. *Federated Learning for the Classification of Tumor Infiltrating Lymphocytes*. Available at: <<https://arxiv.org/abs/2203.16622>>.
- Baxter, J., 1997. A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling. *Machine Learning*, [online] 28(1), pp.7–39. Available at: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-0031187873&doi=10.1023%2FA%3A1007327622663&partnerID=40&md5=2d2e6b9535edcb98cddf38bba9f8332b>>.
- Bochkovskiy, A., Wang, C.-Y. and Liao, H.-Y.M., 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. [online] Available at: <<https://arxiv.org/abs/2004.10934>> [Accessed 17 Aug. 2021].
- Boesch, G., 2021. *Pytorch vs Tensorflow: A Head-to-Head Comparison*. [online] viso.ai. Available at: <<https://viso.ai/deep-learning/pytorch-vs-tensorflow/>> [Accessed 15 Jul. 2021].
- Boukerche, A. and Coutinho, R.W.L., 2018. Smart Disaster Detection and Response System for Smart Cities. In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. pp.1102–1107.



Buciluundefined, C., Caruana, R. and Niculescu-Mizil, A., 2006. Model Compression. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06*. [online] New York, NY, USA: Association for Computing Machinery. pp.535–541. Available at: <<https://doi.org/10.1145/1150402.1150464>>.

Castro-Zunti, R.D., Yépez, J. and Ko, S.B., 2020. License plate segmentation and recognition system using deep learning and OpenVINO. *IET Intelligent Transport Systems*, 14(2), pp.119–126.

Chaudhuri, N. and Bose, I., 2019. Application of Image Analytics for Disaster Response in Smart Cities. In: *Proceedings of the 52nd Hawaii International Conference on System Sciences*.

Chen, Y., Zhao, D., Lv, L. and Zhang, Q., 2018. Multi-task learning for dangerous object detection in autonomous driving. *Information Sciences*, [online] 432, pp.559–571. Available at: <<https://www.sciencedirect.com/science/article/pii/S0020025517308848>>.

Dvornik, N., Shmelkov, K., Mairal, J. and Schmid, C., 2017. BlitzNet: A Real-Time Deep Network for Scene Understanding. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. pp.4174–4182.

Gao, W., Li, L., Zhu, X. and Wang, Y., 2020. Detecting Disaster-Related Tweets Via Multimodal Adversarial Neural Network. *IEEE MultiMedia*, 27(4), pp.28–37.

Gautam, A.K., Misra, L., Kumar, A., Misra, K., Aggarwal, S. and Shah, R.R., 2019. Multimodal Analysis of Disaster Tweets. In: *2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM)*. pp.94–103.

Google, 2021a. *Choose the Colab plan that's right for you*. [online] Google Research. Available at: <<https://colab.research.google.com/signup>> [Accessed 17 Jul. 2021].

Google, 2021b. *Colaboratory - Frequently Asked Questions*. [online] Google Research. Available at: <<https://research.google.com/colaboratory/faq.html>> [Accessed 14 Jul. 2021].

Gou, J., Yu, B., Maybank, S.J. and Tao, D., 2020. Knowledge Distillation: A Survey. *CoRR*, [online] abs/2006.0. Available at: <<https://arxiv.org/abs/2006.05525>>.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. [online] Available at: <<https://arxiv.org/abs/1704.04861>> [Accessed 22 Aug. 2021].

Intel, 2021. *Intel AI Mentors Seek to Improve Astronaut Health*. [online] [www.intel.com](http://www.intel.com). Available at: <https://www.intel.com/content/www/us/en/newsroom/news/intel-ai-mentors-seek-improve-astronaut-health.html#gs.xui7hw> [Accessed 29 Mar. 2022].

Intel, 2022. *Intel® Distribution of OpenVINO™ Toolkit*. [online] [intel.com](http://intel.com). Available at: <https://www.intel.com/content/www/us/en/developer/tools/opencvino-toolkit/overview.html> [Accessed 15 Mar. 2022].

Izutov, E., 2018. Fast and Accurate Person Re-Identification with RMNet. *CoRR*, [online] [abs/1812.0](https://arxiv.org/abs/1812.02465). Available at: <http://arxiv.org/abs/1812.02465>.

Kanjee, R., 2020. *YOLOv5 Controversy — Is YOLOv5 Real?* [online] [medium.com](https://medium.com). Available at: <https://medium.com/augmented-startups/yolov5-controversy-is-yolov5-real-20e048bebb08> [Accessed 17 Aug. 2021].

Kaushal, A., Altman, R. and Langlotz, C., 2020. *Health Care AI Systems Are Biased*. [online] *Scientific American*. Available at: <https://www.scientificamerican.com/article/health-care-ai-systems-are-biased/> [Accessed 3 Mar. 2022].

Khan, S., Muhammad, K., Mumtaz, S., Baik, S.W. and de Albuquerque, V.H.C., 2019. Energy-Efficient Deep CNN for Smoke Detection in Foggy IoT Environment. *IEEE Internet of Things Journal*, 6(6), pp.9237–9245.

Lee, W., Na, J. and Kim, G., 2019. Multi-Task Self-Supervised Object Detection via Recycling of Bounding Box Annotations. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. pp.4979–4988.

Lin, H., Deng, J.D., Albers, D. and Siebert, F.W., 2020. Helmet Use Detection of Tracked Motorcycles Using CNN-Based Multi-Task Learning. *IEEE Access*, 8, pp.162073–162084.

McMahan, H.B., Moore, E., Ramage, D. and y Arcas, B.A., 2016. Federated Learning of Deep Networks using Model Averaging. *CoRR*, [online] [abs/1602.0](https://arxiv.org/abs/1602.05629). Available at: <http://arxiv.org/abs/1602.05629>.

Meel, V., 2021. *YOLOv5 Is Here! Is It Real or a Fake?* [online] [viso.ai](https://viso.ai). Available at: <https://viso.ai/deep-learning/yolov5-controversy/> [Accessed 17 Aug. 2021].

OpenVINO, 2022. *Compression of a Model to FP16*. [online] [docs.openvino.org](https://docs.openvino.org). Available at:

<[https://docs.openvino.ai/latest/openvino\\_docs\\_MO\\_DG\\_FP16\\_Compression.html](https://docs.openvino.ai/latest/openvino_docs_MO_DG_FP16_Compression.html)> [Accessed 14 Mar. 2022].

Padmawar, P.M., Shinde, A.S., Sayyed, T.Z., Shinde, S.K. and Moholkar, K., 2019. Disaster Prediction System using Convolution Neural Network. In: *2019 International Conference on Communication and Electronics Systems (ICCES)*. pp.808–812.

Patidar, S., Rane, D. and Jain, P., 2012. A Survey Paper on Cloud Computing. In: *2012 Second International Conference on Advanced Computing Communication Technologies*. pp.394–398.

Puttinaovarat, S., Sriklın, T., Dangtia, S. and Khaimook, K., 2020. Flood Disaster Identification and Decision Support System using Crowdsourced Data Based on Convolutional Neural Network and 3S Technology. *International Journal of Interactive Mobile Technologies (iJIM)*, [online] 14(20), pp.117–134. Available at: <<https://online-journals.org/index.php/ijim/article/view/17243>>.

Qian, Y., Dolan, J.M. and Yang, M., 2020. DLT-Net: Joint Detection of Drivable Areas, Lane Lines, and Traffic Objects. *IEEE Transactions on Intelligent Transportation Systems*, 21(11), pp.4670–4679.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2015. You Only Look Once: Unified, Real-Time Object Detection. [online] Available at: <<https://arxiv.org/abs/1506.02640>> [Accessed 16 Aug. 2021].

Redmon, J. and Farhadi, A., 2016. YOLO9000: Better, Faster, Stronger. [online] Available at: <<http://arxiv.org/abs/1612.08242>> [Accessed 16 Aug. 2021].

Redmon, J. and Farhadi, A., 2018. YOLOv3: An Incremental Improvement. [online] Available at: <<https://arxiv.org/abs/1804.02767>> [Accessed 16 Aug. 2021].

Reina, G.A., Gruzdev, A., Foley, P., Perepelkina, O., Sharma, M., Davidyuk, I., Trushkin, I., Radionov, M., Mokrov, A., Agapov, D., Martin, J., Edwards, B., Sheller, M.J., Pati, S., Moorthy, P.N., Wang, H.S.-H., Shah, P. and Bakas, S., 2021. OpenFL: An open-source framework for Federated Learning. *CoRR*, [online] abs/2105.0. Available at: <<https://arxiv.org/abs/2105.06413>>.

Ruder, S., 2017. An Overview of Multi-Task Learning in Deep Neural Networks. *CoRR*, [online] abs/1706.0. Available at: <<http://arxiv.org/abs/1706.05098>>.

Settles, B., 2010. *Active Learning Literature Survey*. [online] Madison. Available at: <<https://burrsettles.com/pub/settles.activelearning.pdf>>.

Sheller, M.J., Reina, G.A., Edwards, B., Martin, J. and Bakas, S., 2019. Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11383 LNCS, pp.92–104.

Shi, J., Qu, X., Feng, Y. and Wang, C., 2021. A Vehicle Detection Method Based on Improved YOLOv3. In: *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. pp.2201–2207.

Siqi, J., 2021. *China floods: economic damage to livestock industry tops US\$348 million, 6.4 million chickens dead*. [online] China Macro Economy. Available at: <[https://www.scmp.com/economy/china-economy/article/3142889/chinese-floods-snarl-supply-chains-lead-and-aluminium-amid?module=perpetual\\_scroll&pgtype=article&campaign=3142889](https://www.scmp.com/economy/china-economy/article/3142889/chinese-floods-snarl-supply-chains-lead-and-aluminium-amid?module=perpetual_scroll&pgtype=article&campaign=3142889)> [Accessed 30 Aug. 2021].

Stack Overflow, 2020. *2020 Stack Overflow Developer Survey*. [online] Stack Overflow. Available at: <<https://insights.stackoverflow.com/survey/2020#technology-other-frameworks-libraries-and-tools-professional-developers3>> [Accessed 20 Aug. 2021].

TensorFlow, 2021a. *Introduction to graphs and tf.function*. [online] TensorFlow Core. Available at: <[https://www.tensorflow.org/guide/intro\\_to\\_graphs](https://www.tensorflow.org/guide/intro_to_graphs)> [Accessed 20 Jul. 2021].

TensorFlow, 2022. *Frequently Asked Questions*. [online] TensorFlow.org. Available at: <<https://www.tensorflow.org/federated/faq>> [Accessed 23 Feb. 2022].

TensorFlow, 2022b. *Post-training quantization*. [online] TensorFlow For Mobile & Edge. Available at: <[https://www.tensorflow.org/lite/performance/post\\_training\\_quantization](https://www.tensorflow.org/lite/performance/post_training_quantization)> [Accessed 12 Mar. 2022].

TensorFlow, 2022c. *TensorFlow Federated: Machine Learning on Decentralized Data*. [online] TensorFlow.org. Available at: <<https://www.tensorflow.org/federated>> [Accessed 25 Feb. 2022].

United Nations Office for the Coordination of Humanitarian Affairs, 2017. *Five essentials for the first 72 hours of disaster response*. [online] OCHA. Available at: <<https://www.unocha.org/story/five-essentials-first-72-hours-disaster-response>> [Accessed 20 Aug. 2021].

Wen, C., Zhang, H., Li, H., Li, H., Chen, J., Guo, H. and Cheng, S., 2020. Multi-scene citrus detection based on multi-task deep learning network. In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. pp.912–919.

Xia, Q., Ye, W., Tao, Z., Wu, J. and Li, Q., 2021. A survey of federated learning for edge computing: Research problems and solutions. *High-Confidence Computing*, [online] 1(1), p.100008. Available at: <<https://www.sciencedirect.com/science/article/pii/S266729522100009X>>.

Xiong, Z., Cheng, Z., Xu, C., Lin, X., Liu, X., Wang, D., Luo, X., Zhang, Y., Qiao, N., Zheng, M. and Jiang, H., 2020. Facing small and biased data dilemma in drug discovery with federated learning. *bioRxiv*, [online] p.2020.03.19.998898. Available at: <<http://biorxiv.org/content/early/2020/09/26/2020.03.19.998898.abstract>>.

Yrjänäinen, J., Ni, X., Adhikari, B. and Huttunen, H., 2020. Privacy-Aware Edge Computing System For People Tracking. In: *2020 IEEE International Conference on Image Processing (ICIP)*. pp.2096–2100.

Zhang, N. and Fan, J., 2021. A lightweight object detection algorithm based on YOLOv3 for vehicle and pedestrian detection. In: *2021 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*. pp.742–745.

Zhang, W., Wang, K., Wang, Y., Yan, L. and Wang, F.-Y., 2021. A loss-balanced multi-task model for simultaneous detection and segmentation. *Neurocomputing*, [online] 428, pp.65–78. Available at: <<https://www.sciencedirect.com/science/article/pii/S0925231220318105>>.

Zhang, X., Li, N. and Zhang, R., 2021. An Improved Lightweight Network MobileNetv3 Based YOLOv3 for Pedestrian Detection. In: *2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE)*. pp.114–118.

Zheng, Z., Zhao, J. and Li, Y., 2021. Research on Detecting Bearing-Cover Defects Based on Improved YOLOv3. *IEEE Access*, 9, pp.10304–10315.

## LIST OF PUBLICATIONS

Tham, M.-L., **Wong, Y.J.**, Kwan, B.H., Owada, Y., Sein, M.M. and Chang, Y.C., 2021. Joint Disaster Classification and Victim Detection using Multi-Task Learning. In: 2021 IEEE 12th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON). pp.407–412.