



**UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA**



**DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE**

**MASTER DEGREE IN  
ICT FOR INTERNET AND MULTIMEDIA**

**Multi Configuration Object Detection Framework for Camera Surveillance  
System**

**Supervisor: Prof. Michele Zorzi**

**Students' name: MEMEN HAMZAH OBAID SALIHI**

**ACADEMIC YEAR 2021 – 2022**

**Date of graduation 24/03/2022**



# Abstract

The high delay produced by the two-stage Object detection methods is limiting their use in video surveillance scenario. Therefore, most of the existing works use only light object detection method to be applied on IoT devices.

The problem of these detectors is the low accuracy. To find a trade off between the latency and the accuracy we proposed multi configuration framework where each group of cameras are set under one configuration. After solving the optimization problem, the system will be able to decide where to process the tasks based on the minimum cost for all the configurations. The computation decision can be done locally (Edge) or remotely (Cloud).

The experiment result demonstrates that the proposed multi-configuration object detection framework outperforms the existing uni configuration system in terms of lower latency and higher accuracy.

**Keywords:** Surveillance system, Edge-Cloud computing, Object detection, IoT



# Acknowledgements

I would like to thank my family, my dad and mum, my siblings Muhammed, Nihad and Estabraq, my second family, Roberta, Stefano, Alessandro and Elisa for their love, patience and support through my academic life. I thank professors Simone Milani and Leonardo Badia for their main contribution to my studying at the University of Padova and National Taiwan University.

I thank all the lecturers and course mates at both universities, that taught me and gave me the necessary knowledge through my study and while doing this thesis. I am grateful for my supervisor, Professor Hung-Yu Wei for his infinite support and expert advise since the beginning of my study at NTU. I am glad and thankful to professor Michele Zorzi for his valuable time and advice being my supervisor in UNIPD.

I say thank you to my lab mates Ann, Louis, Howard, Zac, Zan Lun, Diego,Hao, Kuang Hsun Lin , 楊雅婷,Jacob Hsieh and Jason. Their friendship and bonds that we created during my stay in NTU are cherished and greatly appreciated.

Lastly, thanks to all the administration staff in both universities, they are behind all the long process of the graduation and administrative work.



# Contents

	<b>Page</b>
<b>Abstract</b>	<b>3</b>
<b>Acknowledgements</b>	<b>5</b>
<b>Contents</b>	<b>7</b>
<b>List of Figures</b>	<b>11</b>
<b>List of Tables</b>	<b>13</b>
<b>Denotation</b>	<b>15</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Proposed framework . . . . .	2
1.2 Related work . . . . .	3
1.3 Outline . . . . .	6
<b>Chapter 2 Object detection in camera surveillance</b>	<b>7</b>
2.1 Object detection methods . . . . .	8
2.1.1 One Stage Light Detector . . . . .	8
2.1.2 One Stage Heavy Detector . . . . .	10
2.1.3 Two Stage Detector . . . . .	10
<b>Chapter 3 Methodology</b>	<b>13</b>
3.1 Materials . . . . .	13

3.2	Tools . . . . .	13
3.3	Methods . . . . .	15
3.3.1	Object detection inference . . . . .	15
3.3.2	Model selection . . . . .	16
3.3.3	Evaluation metrics . . . . .	17
<b>Chapter 4</b>	<b>System model</b>	<b>23</b>
4.1	Latency optimization problem . . . . .	24
4.1.1	Computing decision parameters . . . . .	24
4.1.2	Problem formulation . . . . .	27
4.1.3	Optimal solution characteristics . . . . .	28
4.2	Cost optimization . . . . .	29
4.2.1	System model for cost minimization . . . . .	29
4.2.2	Decision model . . . . .	30
4.2.3	Computation model . . . . .	30
4.2.4	Cost minimization problem . . . . .	31
4.2.5	Problem formulation . . . . .	33
<b>Chapter 5</b>	<b>Result and evaluation</b>	<b>35</b>
5.1	Scenario1: The number of configurations is equal to the number of cameras . . . . .	36
5.2	Scenario2: Each group of cameras under one configuration . . . . .	38
5.2.1	Case1: $Cam_1$ and $Cam_4$ are set to configuration one ( $C_1$ ) . . . . .	38
5.2.2	Case2 : $Cam_2$ and $Cam_4$ are set to configuration two ( $C_2$ ) . . . . .	41
5.2.3	Case3: $Cam_3$ and $Cam_4$ are set to configuration three ( $C_3$ ) . . . . .	42



5.3	Performance evaluation . . . . .	43
5.3.1	Overall cost comparison of the three cases of scenario2. . . . .	44
5.3.2	Overall cost comparison of multi and uni configuration system . . .	46
5.3.3	Performance comparison between uni (baseline) and multi configuration framework (proposed) in terms of accuracy (mAP) and cost of each camera . . . . .	47
<b>Chapter 6</b>	<b>Conclusions</b>	<b>53</b>
6.1	Conclusions . . . . .	53
6.2	Future work . . . . .	55
	<b>References</b>	<b>57</b>



# List of Figures

1.1	The proposed framework (Hybrid).	2
2.1	One stage detector.	9
2.2	Two stage detector.	11
3.1	TensorFlow Hub.	14
3.2	Resources locally characteristics.	14
3.3	Resources remotely characteristics.	15
3.4	Example of mean Average Precision.	19
3.5	Example of ground truth bounding box and the predicted one.	20
4.1	Decision parameters.	26
5.1	Multi configuration framework (scenario1).	37
5.2	Multi configuration framework: Scenario2-Case1.	39
5.3	The total cost performance for multi configuration framework (scenario2).	45
5.4	Uni configuration framework (baseline)	46
5.5	The total minimum cost for each solution in Uni configuration framework(baseline).	47
5.6	Performance comparison between Uni configuration (Baseline) build on Configuration1 and Case1 from Multi configuration (Proposed) in terms (a) Accuracy (mAP) and (b) Cost of each camera.	48
5.7	Performance comparison between Uni configuration (Baseline) build on Configuration2 and Case2 from Multi configuration (Proposed) in terms (a) Accuracy (mAP) and (b) Cost of each camera.	49

5.8	Performance comparison between Uni configuration (Baseline) build on Configuration3 and Case3 from Multi configuration (Proposed) in terms (a) Accuracy (mAP) and (b) Cost of each camera. . . . .	50
5.9	Performance comparison between (a) Case1, (b) Case2 and (c) Case3 in terms of minimizing the cost and maximizing the mAP. . . . .	51

# List of Tables

3.1	Object detection models . . . . .	16
3.2	Selected Object detection models . . . . .	17
3.3	Selected OD models with mean Average Precision and image scale . . . . .	17
4.1	The system notations and description. . . . .	23
4.2	Configurations offloading solutions. . . . .	29
4.3	Parameters and description for general case. . . . .	31
5.1	Simulation parameters setting. . . . .	36
5.2	Object detection configurations for scenario1 . . . . .	37
5.3	The cost for each configuration in scenario1 . . . . .	37
5.4	The total cost solutions for scenario1. . . . .	38
5.5	Object detection configurations for scenario2 - Case1 . . . . .	39
5.6	The cost for each camera in the scenario2 - Case1 . . . . .	40
5.7	The total cost solutions for scenario2 - Case1 . . . . .	40
5.8	Object detection configurations for scenario2 - Case2 . . . . .	41
5.9	The cost for each camera in the scenario2 - Case2 . . . . .	41
5.10	The total cost solutions for scenario2- Case2 . . . . .	42
5.11	Object detection configurations for scenario2- Case3 . . . . .	42
5.12	The cost for each camera in the scenario2 - Case3 . . . . .	43
5.13	The total cost solutions for scenario2 - Case3 . . . . .	44
5.14	Performance comparison between the cost and accuracy(mAP) of configuration1 from uni configuration framework and Case1 from multi configuration framework. . . . .	48

5.15 Performance comparison between the cost and accuracy of uni configura- tion2 system and multi configuration system. . . . .	49
5.16 Performance comparison between the cost and accuracy of uni configura- tion3 system and multi configuration system. . . . .	50

# Denotation

ML	Machine Learning
DL	Deep Learning
mAP	mean Average Precision
OD	Object Detection
CPU	Central Processing Unit
GPU	Graphic Processing Unit
CM	Confusion Matrix
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
SSD	Single Shot MultiBox Detector

OD	Object detection
IoU	Intersection over Union
AI	Artificial Intelligent
DNN	Deep Neural Network
CNN	Convolutional Neural Network
IoT	Internet of Things
VOC	Visual Object Classes
FPN	Feature Pyramid Networks
RPN	Regional Proposal Network
EU	End Users
MEC	Multi-access Edge Computing



# Chapter 1 Introduction

Object detection in general is one of the important challenges in the field of Deep Learning (DL) because of the wide use in everyone's daily life such as security, military, medical and transportation (Autonomous driving) fields. In specific, this work focus on Object Detection(OD) in security field as a camera surveillance scenario.

Object detection accuracy and latency in the surveillance system is an important concern. The existing camera surveillance frameworks stand only on one configuration. Those configurations are object detection methods that detect some specific objects according to the target of each system such as face recognition. The challenge in this field to build an efficient framework that can find trade-off between the accuracy of the configuration and its latency.

With the purpose of building a framework that combine high accuracy detection and low latency, many configurations has been used such as one-stage heavy and light detectors. Most of the state-of-the-art detection work tend to use light backbone detector to deploy object detection task at the end devices. Regarding two-stage detectors, because of their heavy, complex computing requirements and resource constraints of the end devices most of the proposed work use division strategies such as distribution, layer partitioning and pruning between fog, edge and cloud nodes in order to reduce their computing load.

Although, the aforementioned division strategies reduce the latency somehow but still not enough to be deployed on the end devices. Thus, in the proposed framework, it is suggested to set some of those cameras on high accuracy configurations in order to reduce the over all latency for the system. More on the proposed framework in 1.1

## 1.1 Proposed framework

To find a trade-off between the accuracy and the latency of two stage detectors in the surveillance system, the proposed framework (Hybrid) is able to set a surveillance system on more than one configuration. So instead of having a system where both accuracy and latency are high or low, in our framework we will have a system with more than one configuration for the cameras.

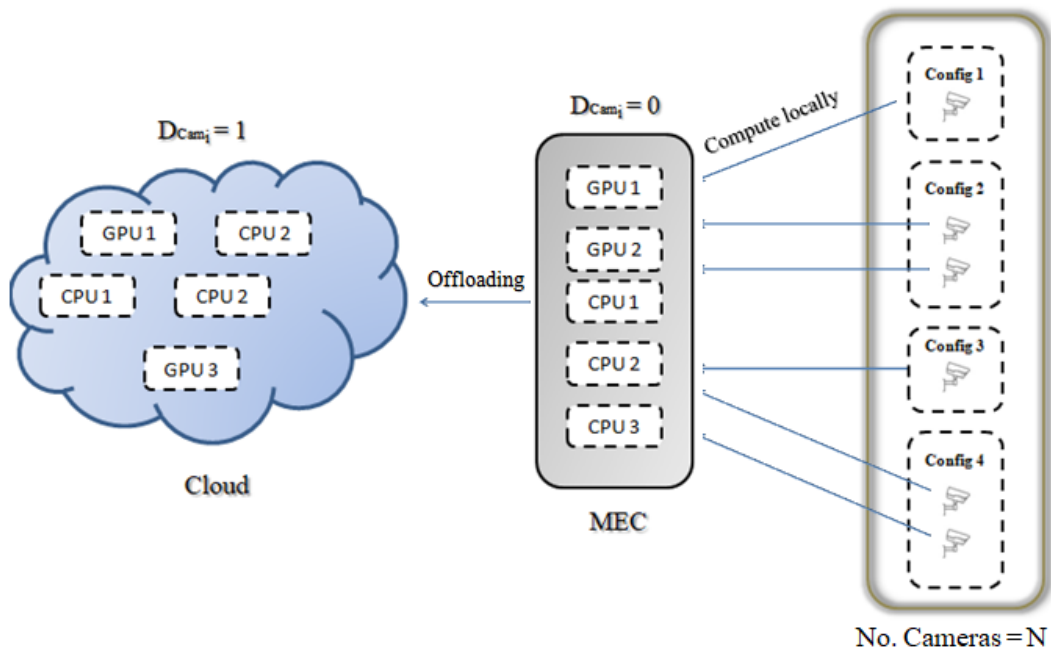


Figure 1.1: The proposed framework (Hybrid).

The proposed framework consists on many cameras that each or some of the them are set to one configuration. Those configurations are object detection models that differ

in accuracy and latency. In this framework, each or group of cameras are set under one configuration according to the operating system requirements. After that, each camera can choose whether to process their task locally or by offloading based on the available resources at each node and total minimum cost.

In this paper, object detection inference performed on ten Object detection models. Those models or configurations are selected according to their characteristics such as one stage light detectors, one stage heavy detectors and two stage detectors. In the model selection phase, It has been selected the models with the highest accuracy and lowest latency from each category configuration. Also, the inference time was measured and recorded for those configurations locally and by offloading in order to compute the overall minimum cost.

The configurations (Object detection models) were selected from Tensorflow hub, an open source software library for Machine Learning(ML) and Artificial Intelligent(AI). It is used from across ranges of object detection tasks but it focus mostly on training and inference of Deep Neural Network (DNN).Figure 1.1 describes the aspired state of the proposed framework.

## **1.2 Related work**

Due to the heavy backbone of two stage object detectors, it is difficult to deploy them in IoT devices such as web cameras. Therefore, there is the need to design a framework that handle the division of tasks between the end devices considering the available resources.

In this section, we have three parts of relevant work concerning the surveillance sys-

tem, they are as follow:

1. Surveillance system or video analytics framework:

A Decomposable intelligence inference approach to video analytics on Edge-Cloud IoT is presented in [38], they provide a framework to support joint latency and accuracy aware live video analytics services. They used a game theory approach for the negotiation between the MEC stratum and the cloud to distribute the computing load. YOLO3 [24] as one stage detector was used for face recognition purpose. Authors in [15] introduced a DeePar which is a hybrid Device-Edge-Cloud framework that consists on layer level partitioning strategy for Deep Neural Network (DNN). Their framework distribute the computing load between the device, edge and cloud in order to reduce the overall execution delay. [39] presented intelligent surveillance system by transferring only the required frame to precise detection when it detect a potential risk in their construction field. Therefore, it reduces the network traffic as well the computational burden for the server. Moreover, they used Tiny-YOLO [17] as one-stage detector as a first step detection. While in [9] they use the autoencoder machine learning strategy to reduce the dimensions of the input data. First the data encoded at the edge and then decoded on the cloud to extract the feature. Finally, authors in [6] presented a surveillance intelligent system where the deep learning models distributed on edge computing architecture layers.

2. Object detection's division strategies in IoT.

Due to the resource constraints on the edge and the high latency produced by the object detection methods on the cloud , there are some proposed work regarding this concept to reduce this latency and deploy those complex deep learning models on

end devices. ,[28] proposed an approach to partition a deep neural network models and apply progressive transmission of intermediate convolutions filter maps. Authors in [14] presents a novel approach to split machine learning models between IoT devices and Cloud in a progressive manner. Also their approach could be combined with model compression and adaptive model partitioning to create an integrated system on IoT-Cloud partitioning. In [36] suggest an efficient energy consumption framework by pruning the layers of object detection models aggressively. Finally,[33] present Edgelens which is a framework to deploy deep learning applications in fog-Cloud environment to harness edge and cloud resources to provide better service quality.

### 3. Computation offloading:

In terms of computation offloading there are several papers have been presented, but in this section we mention the most related to our work. In[22] authors introduced an algorithm that do partial offloading from end users(EU) to MEC under a comprehensive optimization of EU's energy consumption and performance. Another approach was presented by [10] to enable the mobile devices to offload their task to fog computing nodes by a module placement method. [35] proposed a dynamic offloading framework, uplink Non-Orthogonal Multiple Access (NOMA) is used to enable the end devices to offload their task via the same frequency. Lastly, in [4] they improved the response time for offloading by Decision tree, Random Forest and Ada boost classifier in mobile fog computing.

All the frameworks described above, they use light object detection configuration to be deployed on the end devices or they use partitioning, slicing or punning to reduce the computing load. those frameworks has the same configuration for all the cameras.

In other words, all the cameras in the system or IoT devices have the same accuracy and cost. So if the system implemented on a configuration with low accuracy and low latency they are compromising one important factor in the system which is the accuracy. Also, if they implement a system with high accuracy, the latency will be high as well. To find a trade-off between those two important objectives, the proposed framework utilize more than one configuration in a surveillance system to have a balanced trade-off between the two important factors in the surveillance system.

### 1.3 Outline

The rest of the paper is structured as follow:

1. **Chapter two** describes Object Detection concept in camera surveillance system.
2. **Chapter three** introduces the theory,tools, methods and materials used in this thesis work.
3. **Chapter four** presents the model system for both latency and cost minimization problem.
4. **Chapter five** evaluates the proposed framework and compare it with the uni configuration system.
5. **Chapter six** concludes the work on this research and gives some suggestion for future works.

# Chapter 2 Object detection in camera surveillance

Surveillance system is a camera based monitoring that observe activities and collect information to be used in video analytics or for security purpose such as the surveillance systems in the bank, airport and military fields. In the recent year, Unmanned Ariel Vehicles (UAVs) such as drones started to be deployed for surveillance in military and civil purposes[37].

The cameras in surveillance system usually capture point of interest according to the purpose of the system that are designed for, in this case the selection of the detector is very important task when it comes to sensitive scenarios such as face recognition while wearing a mask as the situation now a days in the COVID 19 outbreak. In some scenarios of surveillance framework the accuracy is more important than the latency therefore the choice of the object detection models falls on two stage detectors such as Faster R-CNN [26] and Mask R-CNN [11]. In order to understand better what is actually one stage heavy detectors, light detectors and two stage detectors , in this chapter we provide some of the literature review of the aforementioned object detectors categories.

## 2.1 Object detection methods

Object detection methods are Deep Learning Convolutional Neural Networks (CNN) trained on very huge data such as ImageNet [7], MS-COCO [19] and VOC [8]. Object detection methods are usually divided to two main parts: one-stage detector such as the most notable one is YOLO [23] and SSD [20], while the second category is the two-stage detectors such as Faster R-CNN. One stage methods detectors provide very fast inference ,therefore it is used for real time application. On the other hand, those detectors have lower accuracy than two stage detectors. In the following subsections, we introduce more details about one stage light, heavy and two stage detectors as well in more comprehensive way and also we list the detectors that have been used to perform inference in this research.

Generally, object detection methods have Backbone or so called feature extractor network. These backbones are Convolutional Neural Network (CNN) that takes the input (image) extract the features and pass it to the fully connected for classification and localization. Based on the number of the convolutional layer we classify the backbone of the detector whether its heavy or light. Example of those backbones are VGG16 [30], MobileNet [29], Hourgalss [21], ResNet [12], MobileNetV2 [29].

In the following subsection we give a brief descriptions of these three kinds of Object Detection methods.

### 2.1.1 One Stage Light Detector

As the name suggest, one stage detector predict the categories from the first stage with the bounding boxes. Figure 2.1 shows the architecture of this kind of detectors, in



our experiment we selected three of those as follow:

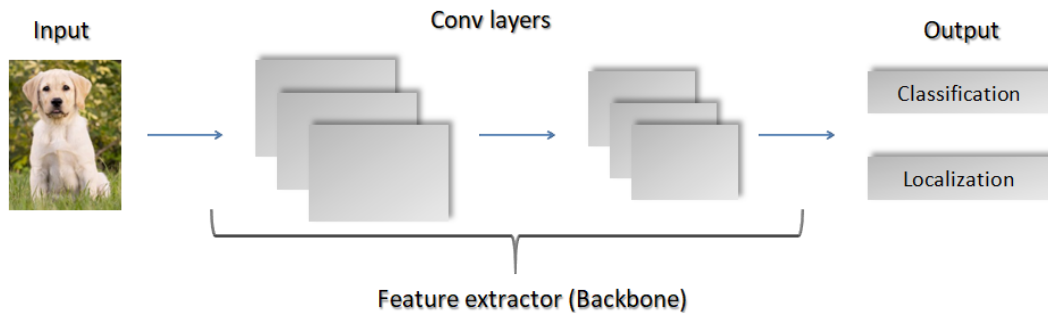


Figure 2.1: One stage detector.

- **Single Shot Detector (SSD MobileNet v2):**

SSD originally was presented in 2016 and it is a simple object detection method based on VGG16 network as a backbone. While SSD MobileNetV2 has the architecture of SSD but the backbone of MobileNetV2 [29] and the last uses lightweight convolutional layers to extract the feature. In specific this method with 320x320 image resolution give 20.2 as mean Average Precision (mAP).

- **SSD MobileNet V1 FPN**

This detector is Single Shot Detector architecture but with MobileNetV1 [13] as the backbone. The difference between the first model and this, is the Feature Pyramid Network (FPN) [18] that predicts the objects on different scales. Therefore, the accuracy have been increased from 20 to 29.1.

- **Faster R-CNN ResNet50 V1 640x640**

Faster R-CNN [25] is a Region Proposal Network (RPN) that detect the objects in each position of the image. The architecture of this method based on Faster R-CNN but the backbone is ResNet50. The number of layers for this method is 50 convolutional layers. Therefore, it is considered as one light object detection method but with quite high accuracy as 29.3 comparing to SSD.

### 2.1.2 One Stage Heavy Detector

Heavy detectors are similar to the light but with heavy backbone. In other words, the number of convolutional layers is higher and as a result the accuracy of this kind of detectors is higher. The three detectors that were used from this type are the following:

- **Faster R-CNN ResNet50 V1 1024x1024 :**

An OD model based on Regional Proposal Network (RPN) with the ResNet50 backbone that detect the images on 1024x1024 scale. The mean Average Precision for this method is 31 which is higher from the same method with image scale lower.

- **EfficientDet D0 512x512:**

EfficientDet D0 [32] is an object detection method proposed in 2019 for prediction on image with 512x512 scale. This version of the detector gives an accuracy of 33.6, also this accuracy starts increasing when the image scale is higher.

- **Faster R-CNN ResNet101 V1 640x640**

This is the same method in One stage light detector but with heavy ResNet as a feature extractor network that contains 101 Convolution layers. Although, the image scale is 640x640 but the accuracy is higher by 31.8.

### 2.1.3 Two Stage Detector

Figure 2.2 shows the structure of two stage detectors that consist of two parts: the first part is the convolutional layer or so called backbone network that extract the features from the input image and then it pass it to the next part. The second part is the object

proposal network that extracts the Region Of Interest (ROI) from the passed classification and then it classifies it and localize the bounding box as the output of the detector.

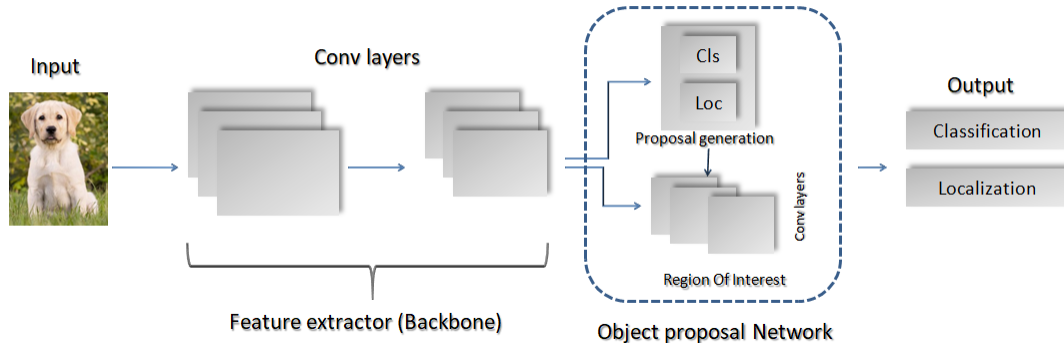


Figure 2.2: Two stage detector.

Four detectors from this type were selected with different image scales.

- **CenterNet Hourglass104**

CenterNet [40] alone is not a traditional two stage detectors but because it has two steps to detect the objects is set under the category of two stage detectors. The two steps for detection are: Firstly, it estimate the probabilities of the classes. Secondly, it classifies the class. Another reason for CenterNet to be under this classification is that CenterNet is build on Hourglass104 [21] as a backbone, which means there are 104 convolutional layers, which is considered a heavy backbone.

- **Faster R-CNN Inception ResNet V2 (640x640)**

It is Faster R-CNN with the Inception [31] family backbone, it predict the objects as points with 640x640 image scale. The mean Average Precision (mAP) for this detector is 37.7 .

- **Faster R-CNN Inception ResNet V2 (1024x1024)**

This method is similar to the once mentioned before but the image scale. It predicts

with mean Average Precision higher than the previous method with 38.7 due to the image resolution that is 1024x1024.

- **Mask R-CNN Inception ResNet V2:**

It is an object detection model trained on COCO dataset as all the other methods mentioned in this section. It is based on Mask R-CNN [11] architecture with Inception ResNetV2 [31] as a backbone. Due to the behavior of Mask R-CNN that generate high quality mask instance for each object, the mAP is higher by 39.0.

# Chapter 3 Methodology

This thesis is conducted based on the Object Detection models in TensorFlow hub machine learning library. This is an original work inspired firstly by the decomposable approach [38] mainly and all the papers that have been mentioned in the related work section 1.2.

## 3.1 Materials

In this research it has been used articles, books and ML libraries, Python language for the simulation part and matplotlib [16] for visualization.

## 3.2 Tools

In the whole thesis, Python [34] has been used as one of the most used language in the field of Machine Learning and data science. Beside python the following tools has been also used :

- TensorFlow Hub :

In the first step of the experiment, TensorFlow [2] Hub was used to the Object

Detection models. It is an open source library for large scale of Machine Learning environment. While, Tensorflow Hub is a repository with trained Object detection models. We have used it to run inference on ten models that was described in 2.1.

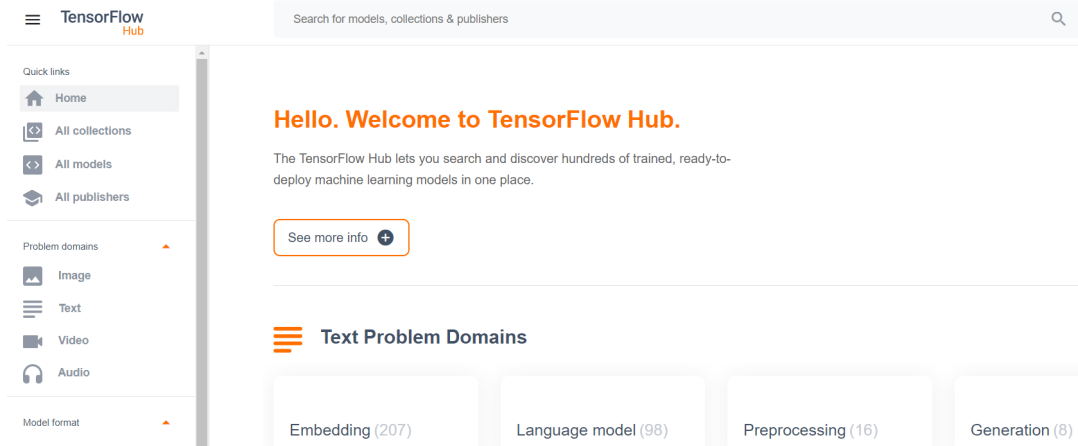


Figure 3.1: TensorFlow Hub.

- Resources locally :

For running the inference locally we used Anaconda3 [1],python distribution platform for data and machine learning processing. Locally, we performed inference on Intel(R) Core(TM) i5-8365U with frequency of 1.60GHz as ( $CPU_{loc}$ ). On the GPU we have GeForce MX110 with memory of 2048MiB as ( $GPU_{loc}$ ). Figure 3.2 shows the properties of the available resources locally.

```

In [4]: import tensorflow as tf

In [5]: from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())

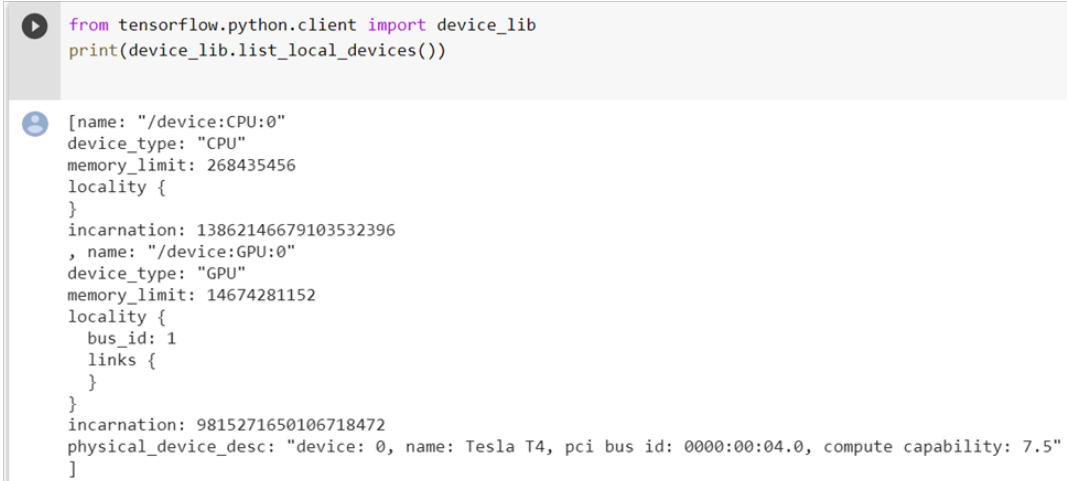
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 10508657310786861730
, name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 1440274841
 locality {
   bus_id: 1
   links {
 }
 }
 incarnation: 156754912993624017
 physical_device_desc: "device: 0, name: GeForce MX110, pci bus id: 0000:01:00.0, compute capability: 5.0"
]

```

Figure 3.2: Resources locally characteristics.

- Resources offloading :

For remote offloading, Google Colab [5] has been used, it is an interactive jupyter notebook that provide free access to the cloud GPUs and CPUs. We performed inference on Intel(R) Xeon(R) CPU @ 2.20GHz as ( $CPU_{off}$ ) and Tesla T4 as ( $GPU_{off}$ ). Figure 3.3 shows the properties of the computing resources remotely we used to perform the inference.



```

from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())

[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 13862146679103532396
 , name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 14674281152
 locality {
   bus_id: 1
   links {
 }
 }
 incarnation: 9815271650106718472
 physical_device_desc: "device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5"
 ]

```

Figure 3.3: Resources remotely characteristics.

## 3.3 Methods

In the following section we describes the steps to conduct our experiment:

### 3.3.1 Object detection inference

We performed Object Detection inference on ten models according to the classification in section 2.1. Three of those models are under one stage light detectors where the accuracy ranges from 20 to 30. Also, we selected three more models as one stage heavy detector with the mean Average Precision ranges from 30 to 35. Finally, the last four models are two stage detectors where their mAP between 35 and 40. Table 3.1 shows the

ten object detection models.

According to the categories of object detectors that was explained in 2.1, the three configurations used in this work named as Configuration1 ( $C_1$ ) that is the One stage light detectors, Configuration2 ( $C_2$ ) is the One stage heavy detectors and lastly Configuration3 ( $C_3$ ) which is the Two stage detectors.

Configuration	Model	$Lc_i$	$Lc_i$	$Lc_i$	$Lc_i$
		( $CPU_{loc}$ )	( $GPU_{loc}$ )	( $CPU_{off}$ )	( $GPU_{off}$ )
C1	Faster R-CNN ResNet101 V1 640x640	14.21	-	13.63	10.33
<b>C1</b>	<b>SSD MobileNet v2 320x320</b>	<b>5.57</b>	<b>9.40</b>	<b>8.55</b>	<b>8.04</b>
C1	SSD MobileNet V1 FPN 640x640	6.58	10.91	11.20	8.82
C2	Faster R-CNN ResNet50 V1 1024x1024	7.515	-	15.23	10.40
<b>C2</b>	<b>EfficientDet D0 512x512</b>	<b>6.88</b>	<b>11.16</b>	<b>10.54</b>	<b>10.01</b>
C2	Faster R-CNN ResNet50 V1 640x640	7.386	-	11.69	11.46
<b>C3</b>	<b>CenterNet HourGlass104 Keypoints 512x512</b>	<b>7.32</b>	<b>23.01</b>	<b>17.16</b>	<b>9.79</b>
C3	Faster R-CNN Inception ResNet V2 640x640	15.583	-	38.88	15.19
C3	Faster R-CNN Inception ResNet V2 1024x1024	16.82	-	46.81	16.89
C3	Mask R-CNN Inception ResNet V2 1024x1024	32.84	-	69.35	20.87

Table 3.1: Object detection models

### 3.3.2 Model selection

After performing object detection inference on the models that were mentioned in section 2.1, in this step we have selected one model from each category Based on the lowest latency and highest accuracy. Table 3.2 shows the selected three models that will be used in the next performance evaluation and simulations. Each of which is named as Configuration1 ( $C_1$ ), Configuration2 ( $C_2$ ), Configuration3 ( $C_3$ ) as one stage light, one-stage heavy and two stage detector, respectively. We selected those three models because :

1. Our purpose in this thesis to build a framework with highest accuracy and lowest latency, Therefore the selected models are the best between other in the same category.



2. Although, their image scale is low, they predict well with high accuracy comparing to others as we can see in Table 3.3.
3. Another reason is that we were able to run inference of the three of them on our limited resources locally. As we can see in Table 3.1 not all of them we were able to run them on the edge GPU, and the last represents the available resources locally in our work.

Configuration	Model	$Lc_i$	$Lc_i$	$Lc_i$	$Lc_i$
		( $CPU_{loc}$ )	( $GPU_{loc}$ )	( $CPU_{off}$ )	( $GPU_{off}$ )
C1	SSD MobileNet v2 320x320	5.57	9.40	8.55	8.04
C2	EfficientDet D0 512x512	6.88	11.16	10.54	10.01
C3	CenterNet HourGlass104 Keypoints 512x512	7.32	23.01	17.16	9.79

Table 3.2: Selected Object detection models

Configuration	Model	Image scale	$mAP$
$C1$	SSD MobileNet v2	320x320	20.2
$C2$	EfficientDet D0	512x512	33.6
$C3$	CenterNet HourGlass104 Keypoints	512x512	40.0

Table 3.3: Selected OD models with mean Average Precision and image scale

### 3.3.3 Evaluation metrics

In this section, we present the metrics that have been used to evaluate the proposed framework. For the accuracy of the object detection methods, mean Average precision ( $mAP$ ) [3] has been used to measure the accuracy of the detector. Alongside with it, the Intersection over Union(IoU) [27] for the precision of the bounding boxes. Also, the latency of the each model has been taken into consideration as it is explained in the latency minimization problem in section 4.1. Moreover, the total cost for the system has been measured and evaluated in section 4.2.

In the following we list the metrics that have been used in this work to evaluate the proposed framework:

- True Positive (TP):

Is the correct detection for the ground truth bounding box.

- False Negative (FN):

It is undetected ground truth bounding box.

- True Negative (TN):

They are instances for given class and they are correctly classified negative.

- False Positive (FP):

They are instances incorrectly classified or misplacing for the predicted bounding box comparing to the ground truth.

- Precision:

Is the ratio between the true positive instances and the predicted result (True Positive and False Positive), also defined as following:

$$Precision = \frac{TP}{TP + FP} \quad (3.1)$$

- Recall:

Is the True Positive predicted instances over the actual result (True Positive and False negative) or as:

$$Recall = \frac{TP}{TP + FN} \quad (3.2)$$

- mean Average Precision (mAP)

Figure 3.4 shows the area under the Precision-Recall curve, measured as follow:

$$mAP = \frac{1}{n} \sum_{i=1}^n AP_i \quad (3.3)$$

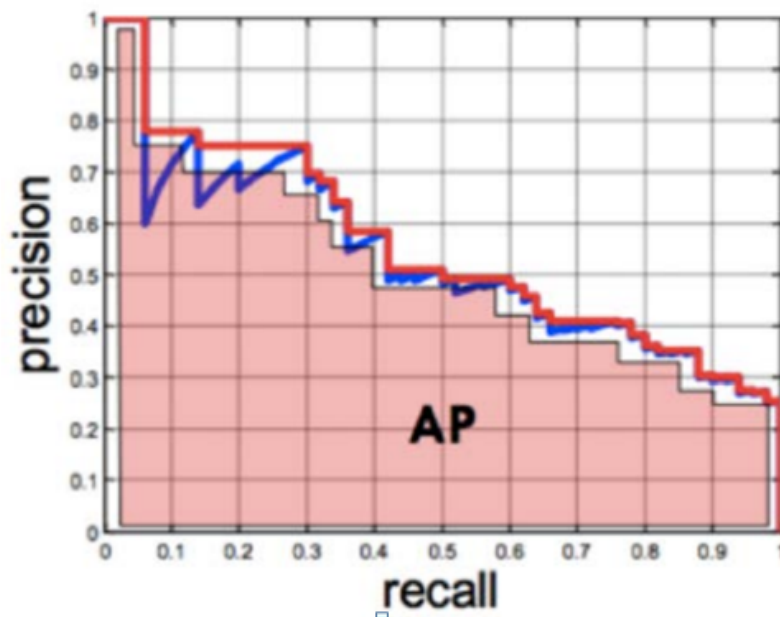


Figure 3.4: Example of mean Average Precision.

- Intersection over Union (IoU)

It is the ratio between the Ground truth bounding box and the predicted bounding boxes as it shows in Figure 3.5, it is also defined as following:

$$IoU = \frac{B_p \cap B_{gt}}{B_p \cup B_{gt}} \quad (3.4)$$

- Accuracy:

It is the summation of the correct prediction and the True Negative to the Total, also defined as follow:

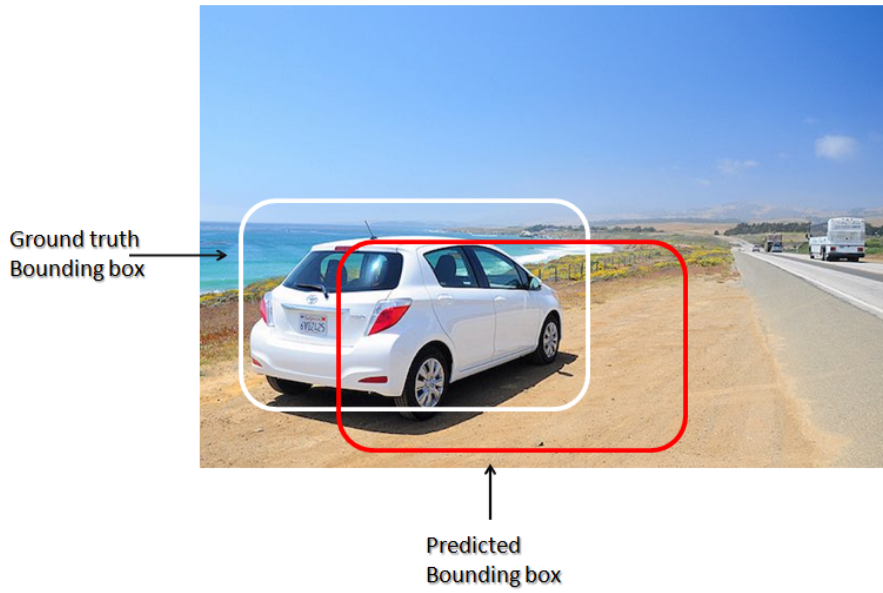


Figure 3.5: Example of ground truth bounding box and the predicted one.

$$Accuracy = \frac{TP + TN}{Total} \quad (3.5)$$

- Total latency ( $T_l$ ):

Is the total minimum latencies for all the cameras in the system that is defined as follow:

$$T_l = \sum_{i=1}^N Lc_i \quad (3.6)$$

More on that in section 4.1

- Total cost ( $Cost_{all-C_i}$ ):

Finally the total minimum cost for the camera surveillance system, it is calculated as follow:

$$Cost_{all-C_i} = \sum_{i=1}^{N_{C_i}} Cost_{C_i} = 2^{N_{C_i}} \quad (3.7)$$

More details in subsection 4.2.4.



## Chapter 4 System model

In this chapter, there are two main sections that are related to the system model. The first part is the latency minimization problem while the second part is the cost minimization problem for general camera surveillance system.

Before going to the next section , Table 4.1 shows the system notation and their description that have been used for the latency optimization problem.

Table 4.1: The system notations and description.

Notation	Description
$N$	Number of cameras for the whole system. $i = \{1, 2, 3\}$ $N = N_{c_1} + N_{c_2} + N_{c_3}$
$N_{c_1}$	Number of cameras under configuration 1.
$N_{c_2}$	Number of cameras under configuration 2.
$N_{c_3}$	Number of cameras under configuration 3.
$C_1$	Configuration 1: One stage light object detection model
$C_2$	Configuration 2: One stage heavy object detection model
$C_3$	Configuration 3: Two stage object detection model
$L_{c_i-e}$	The latency of configuration $i$ on the edge.
$L_{c_i-cl}$	The latency of configuration $i$ on the cloud.
$T_{l-e}, T_{l-cl}$	Total latency on the edge and cloud respectively
$L_{c_i}$	The minimum latency between edge and cloud for configuration $i$
$T_l$	The total minimum latency between edge and cloud
$mAP$	mean Average Precision.
$Dc_i$	Computing decision for configuration $i$ .
$T_R$	Total resources.
$U_{max}$	Maximum utilization.
$R_{av-e}$	Available resources at the edge.
$R_{av-cl}$	Available resources at the cloud.

## 4.1 Latency optimization problem

In the beginning of the optimizing problem, we consider minimizing the total latency for all the configurations in the system. The configurations that are considered in this section are the three selected models in Table 3.2. Therefore, latency minimizing problem is applied on limited scale system where the number of cameras are equal to the number of configurations and the last is equal to three.

### 4.1.1 Computing decision parameters

The computing decision ( $Dc_i$ ) for each configuration task is done either locally on the edge or to be offloaded remotely to the cloud server. The decision depends on the system parameters such as the total minimum latency ( $T_l$ ) and the total available resources ( $T_R$ ) of each node.

The decision parameters are a vector that represent the characteristics of the system environment. they are as follow:

#### 1. The latency on the edge

The latency on the Edge for *Configuration<sub>i</sub>* is calculated as multiplying the number of cameras under *Configuration<sub>i</sub>* by the latency of that configuration on the Edge.

$$Lc_{i-e} = Nc_i Lc_{i-e} \quad (4.1)$$

#### 2. The latency on the cloud



The latency on the Cloud for  $Configuration_i$  is calculated as multiplying the number of cameras under  $Configuration_i$  by the latency of that configuration on the Cloud.

$$Lc_{i-cl} = Nc_i Lc_{i-cl} \quad (4.2)$$

### 3. Total latency on the edge

The total latency for all the configurations at the edge ( $T_{l-e}$ ) expressed as the summation of all the latencies of all the configurations on the edge.

$$T_{l-e} = \sum_{i=1}^3 Lc_{i-e} \quad (4.3)$$

### 4. Total latency on the cloud

The total latency on the cloud ( $T_{l-cl}$ ) is the sum of computing all the configuration on the cloud.

$$T_{l-cl} = \sum_{i=1}^3 Lc_{i-cl} \quad (4.4)$$

### 5. The minimum total latency

Since the total delay on the edge or the cloud is not always the minimum total latency, it was necessary to compute another parameter which is the minimum latency ( $Lc_i$ ) that takes the minimum latency for each configuration between edge and cloud.

$$Lc_i = \min(Lc_{i-e}, Lc_{i-cl}) \quad (4.5)$$

And thus, we have the minimum total latency as

$$T_l = \sum_{i=1}^3 Lc_i \quad (4.6)$$

## 6. Maximum utilization of each configuration

The maximum utilization of each configuration ( $U_{max-C_i}$ ) is one of the system parameters that is shown in Figure 4.1. 34.8 % represent the maximum resource utilization for *Configuration<sub>1</sub>* and 46.9% for *Configuration<sub>2</sub>*. And this is correspond to the possibility of computing configuration 1 and 2 on the edge because they utilize less than the total available resources on the edge. while for *Configuration<sub>3</sub>* must be computed on the cloud. By doing so, *Configuration<sub>3</sub>* will not need to wait for configuration 1 and 2 to be done and results in high latency.

Models	Configuration	Le	Lcl	mAP	TI <sub>e</sub>	TI <sub>cl</sub>	Min(TI)	Umax <sub>ci</sub>	Rav <sub>e</sub>	Rav <sub>cl</sub>	Dc
SSD MobileNet v2	C1	5.57	8.04	20.2	27.85	40.20	27.85	34.8	100.0	100.0	Edge
EfficientDet D0	C2	6.88	10.01	33.6	34.40	50.05	34.40	46.9	65.2	100.0	Edge
CenterNet HourGlass104	C3	7.32	9.79	40.0	36.60	48.95	36.60	96.7	18.3	100.0	Cloud

Figure 4.1: Decision parameters.

## 7. Total available resources

The total available resources in the system expressed as the summation of the available resources on the edge and the cloud in this case.

$$T_R = R_{av-e} + R_{av-cl} \quad (4.7)$$

## 8. The available resources on the edge

The available resources on the edge ( $R_{av-e}$ ) in our system is able to allocate only configuration 1 and 2 or configuration 3 but the solution that result in lower latency for all the configurations is to allocate configuration 1 and 2 on the edge and 3 on

the cloud according to the following equation:

$$R_{av-e} = 100 - (U_{max-c1} + U_{max-c2}) \quad (4.8)$$

## 9. The available resources on the cloud

Since, the remaining available resources on the edge is 18.3% which is not sufficient enough to execute configuration 3 on, as a result it will be offload to the cloud.

$$R_{av-cl} = 100 - (U_{max-c3}) \quad (4.9)$$

### 4.1.2 Problem formulation

The objective function is to minimize the total latency of all the configurations in the system. The minimization problem demonstrated as follow:

$$\mathbf{Problem:} \min(T_l) \quad (4.10a)$$

$$s.t. \quad C1 : D_{c_i} \in \{0, 1\} \quad (4.10b)$$

$$C2 : 0 \leq L_{c_i} \leq T_l \quad (4.10c)$$

$$C3 : \sum_{i=1}^3 L_{c_i} \leq T_l \quad (4.10d)$$

$$C4 : 0 \leq U_{max-c_i} \leq T_R \quad (4.10e)$$

$$C5 : \sum_{i=1}^3 U_{max-c_i} \leq T_R \quad (4.10f)$$

In the optimization problem, there are constraints are related to the latency of the

configurations and others related to the available resources in the system.

The first constraint (C1) is the computing decision (Dc) where each task must be processed locally (Dc = 0) or offloaded to the cloud (Dc = 1). The second constraint (C2) guarantees that the latency of any configuration must be between 0 and the total latency for all the configurations in the system. The third constraint (C3) makes sure that the total latency for all the configurations is less or equal to the minimum total latency.

Moreover, other constraints are related to the available resources in the system such as CPUs and GPUs. C3 and C4 assure that the maximum utilization for each configuration and the total maximum utilization for all the configurations must be less or equal than the total available resources in the system.

### 4.1.3 Optimal solution characteristics

According to the optimization problem, each configuration has two choices that is either to process on the edge or to be offloaded to the cloud. As a result we can get eight combination of solutions showed in Table 4.2.

After solving the optimization problem, we can get the lowest minimum latency which is **19.77** which is coming from processing all the configurations on the edge but this is not possible solution because constraint five (C5) will not be fulfilled. At this point, we need to look for another minimum total latency that is **22.24** which is the result of two solutions. The first is processing *Configuration<sub>1</sub>* on the cloud and *Configuration<sub>2</sub>*, *Configuration<sub>3</sub>* on the edge. The second minimum latency that shares the same value is from processing *Configuration<sub>1</sub>*, *Configuration<sub>2</sub>* on the edge and *Configuration<sub>3</sub>* on the cloud.

Allocating *Configuration<sub>2</sub>* and *Configuration<sub>3</sub>* locally on the edge does not satisfies constraint5 because it will exceed the total available resources on the edge. Therefore , the only possible minimum total latency is processing *Configuration<sub>1</sub>* and *Configuration<sub>2</sub>* on the edge and offloading *Configuration<sub>3</sub>* to the cloud as its highlighted in Table 4.2 .

Edge	Cloud	$T_l$
$C_1$	$C_2 + C_3$	25.37
$C_2$	$C_1 + C_3$	24.71
$C_3$	$C_1 + C_2$	25.87
$C_2 + C_3$	$C_1$	22.24
$C_1 + C_3$	$C_2$	22.9
$C_1 + C_2$	$C_3$	22.24
$C_1 + C_2 + C_3$	—	19.77
—	$C_1 + C_2 + C_3$	27.84

Table 4.2: Configurations offloading solutions.

## 4.2 Cost optimization

To extend our experiment on large scale system, we consider a system with N configurations and N cameras.

### 4.2.1 System model for cost minimization

In general model for camera surveillance system, we consider a number of camera web  $N = \{Cam_1, Cam_2, \dots, Cam_n\}$ . Each group of cameras are set under one configuration  $C_i = \{C_1, C_2, \dots, C_n\}$

Each configuration is an Object Detection model that is different from the other in terms of accuracy and whether its one or two stage, heavy or light detector. Those detectors are chosen according to the operating system requirements priorities. Object Detection task for each web camera can choose to be processed locally or to be offloaded.

## 4.2.2 Decision model

The processing decision for each task must be done locally or to be offload to the cloud as it was explained in subsection 4.1.1. The decision parameter  $D_{C_i} \in \{0, 1\}$  indicates whether to process locally where  $D_{C_i} = 0$ , otherwise  $D_{C_i} = 1$  where the decision is to offload the task.

The offloading decision is taken according to the overall minimum cost and the available resources in the system.

## 4.2.3 Computation model

The computation decision for each configuration is performed either locally or by offloading remotely. Thus, we denote  $Cost_{C_i, Cam_i}, \forall i \in [0, n]$  as the cost to process each task. We calculate the cost of each decision as the following:

### 1. Computing locally

There are some parameters to be considered when processing locally such as the latency of processing the task itself locally ( $L_{C_i-loc}$ ) and the power consumption for the device ( $\rho$ ). Therefore, the cost function for processing locally calculated as

$$Cost_{loc-C_i, Cam_i} = \rho + L_{C_i-loc} \quad (4.11)$$

### 2. Offloading

The second decision in our problem is to offload the task, by doing so we need to take into consideration the following parameters:

- Transmission data latency:

We denote  $L_{tran}$  the time takes to transfer the data to the cloud and back (propagation delay).

- Processing latency:

The delay produced by the object detection model for each task to be processed as  $LC_{i-off}$ .

As a result we can calculate the cost of offloading the task as follow

$$Cost_{off-C_i,Cam_i} = L_{tran} + LC_{i-off} \quad (4.12)$$

#### 4.2.4 Cost minimization problem

Before starting the cost minimization problem, Table 4.3 describes the notations and its description that have been used for the cost optimization problem for general case.

Parameter	Description
$C_i$	Configuration i.
$N$	The total number of cameras in the system.
$N_{C_i}$	Total number of configurations in the system.
$Cost_{C_i,Cam_i}$	The cost of configuration i or camera i between to process locally or to be offloaded.
$LC_{i-loc,off}$	Latency to process configuration i locally or by offloading.
$D_{C_i,Cam_i}$	The computation decision for each configuration i or camera i
$\rho$	Power consumption
$Cost_{loc-C_i,Cam_i}$	Cost to process configuration i or camera i locally
$L_{tran}$	Data transmission latency
$T_R$	The total available resources (Locally and remotely)
$U_{max-C_i}$	Maximum utilization of configuration i.
$Cost_{off-C_i,Cam_i}$	The cost of configuration i or camera i to offload the task
$Cost_{all-C_i,Cam_i}$	Minimum total cost for all the configurations or the cameras in the system .

Table 4.3: Parameters and description for general case.

Since our target is to minimize the overall cost for the system, we need to consider two scenarios:

1. The number of cameras are equal to the number of configurations:

In this scenario, to get the overall minimum cost for the system we need to enumerate through all the possible solutions for each configuration. As a result each configuration will have two choices, either to process locally or to offload their task remotely. So the overall cost for all the configurations in the system ( $Cost_{all-C_i}$ ) is expressed as follow

$$Cost_{all-C_i} = \sum_{i=1}^{N_{c_i}} Cost_{C_i} = 2^{N_{c_i}} \quad (4.13)$$

2. The number of cameras in the system are bigger than the number of configurations in the system.

For this scenario where the number of cameras are more than the number of configuration means there are more than one camera under each configuration. To find the overall minimum cost we need to go through all the solutions for each camera in this case, as a result each camera will have the choice to process their task locally or remotely. As a result, the overall cost for this scenario is expressed in 4.14.

After that we can select the minimum possible solution that minimize the overall cost and fulfil all the constraints in the optimization problem 4.16

$$Cost_{all-Cam_i} = \sum_{i=1}^N Cost_{Cam_i} = 2^N \quad (4.14)$$



## 4.2.5 Problem formulation

The main objective function is to maximize the mean Average Precision (mAP) and minimize the cost for each camera. the objective function is expressed as follow:

$$Max(\alpha mAP - \beta Cost) \quad (4.15)$$

Where  $\alpha$  and  $\beta$  are scaling coefficients. Therefore they are set according to each operating system requirement.

Since the mAP is set beforehand for each camera, thus the optimization problem is to minimize the overall cost for all the configurations or all the cameras in the system. The minimization problem demonstrated as follow

$$\mathbf{Problem:} \min(Cost_{all-C_i, Cam_i}) \quad (4.16a)$$

$$s.t. \quad C1 : D_{C_i, Cam_i} \in \{0, 1\} \quad (4.16b)$$

$$C2 : 0 \leq Cost_{C_i, Cam_i} \leq Cost_{all-C_i, Cam_i} \quad (4.16c)$$

$$C3 : \sum_{i=1}^N Cost_{C_i, Cam_i} \leq Cost_{all-C_i, Cam_i} \quad (4.16d)$$

$$C4 : 0 \leq U_{max-C_i} \leq T_R \quad (4.16e)$$

$$C5 : \sum_{i=1}^N U_{max-C_i} \leq T_R \quad (4.16f)$$

In the optimization problem, we need to select the minimum overall cost for all the configurations in the system by going through all the combination of solutions in 4.13 and

choose the combination that fulfill all the constraints in our problem 4.16.

The first constraint (C1) in the optimization problem ensure that the computing decision must be either locally when ( $D_c = 0$ ) or remotely when ( $D_c = 1$ ). The second (C2) and the third constraint (C3) guarantee that the cost for each configuration or camera and the sum of all costs in system must be less or equal than the overall minimum cost in the system. And finally Constraints C4 and C5 make sure that the maximum and the summation of the maximum utilization of each configuration is less or equal to the available total resources in the system.

## Chapter 5 Result and evaluation

To verify and evaluate our proposed framework, this chapter will present the simulation result according to Algorithm 1.

---

**Algorithm 1** An Algorithm Describing Multi Configuration System

---

**Require:**

- 1:  $C_i \leftarrow \{C_1, C_2, \dots, C_n\}$
- 2:  $N \leftarrow \{Cam_1, Cam_2, \dots, Cam_n\}$
- 3:  $T_R \leftarrow \{R_{loc}, R_{off}\}$
- 4:  $U_{max-C_i} \leftarrow \{U_{max-cl}, \dots, U_{max-cn}\}$

**Ensure:**

- 5: The best computation decision ( $D_{C_i, Cam_i}$ )
  - 6: The optimal minimum total cost for the system ( $Cost_{all-C_i, Cam_i}$ )
  - 7: **if**  $N = N_{C_i}$  **then**
  - 8:     **for**  $i \leftarrow 1$  to  $N_{C_i}$  **do**
  - 9:         Calculate  $Cost_{loc-C_i}$  by equation 4.11
  - 10:         Calculate  $Cost_{off-C_i}$  by equation 4.12
  - 11:     **end for**
  - 12:     Calculate  $Cost_{all-C_i}$  by equation 4.13.
  - 13:     Choose the minimum overall cost ( $Cost_{all-C_i}$ ) that fulfill all the constraints in the optimization problem 4.16.
  - 14:     **return**( $D_{C_i}, Cost_{all-C_i}$ )
  - 15: **else if**  $N > N_{C_i}$  **then**
  - 16:     **for**  $i \leftarrow 1$  to  $N$  **do**
  - 17:         Calculate  $Cost_{loc-Cam_i}$  by equation 4.11
  - 18:         Calculate  $Cost_{off-Cam_i}$  by equation 4.12
  - 19:     **end for**
  - 20:     Calculate  $Cost_{all-Cam_i}$  by equation 4.14.
  - 21:     Choose the minimum overall cost ( $Cost_{all-Cam_i}$ ) that fulfill all the constraints in the optimization problem 4.16.
  - 22:     **return**( $D_{Cam_i}, Cost_{all-Cam_i}$ )
  - 23: **end if**
- 

In our simulation, we used some parameters such the transmission delay and power

consumption that is chosen from their ranges in Table 5.1. Moreover, CPU and GPU are used as local resources and remote resources. While, other parameters are given with in each scenario.

Parameter	Value
$N$	$\{3, 4\}$
$\rho, L_{tran}$	$[0, 1]$
$CPU_{loc}$	1.60GHz
$GPU_{loc}$	GeForce MX110, 2048MiB
$CPU_{off}$	2.20GHz, 39424 KB
$GPU_{off}$	Tesla T4

Table 5.1: Simulation parameters setting.

The simulation is conducted in two main scenarios:

## 5.1 Scenario1: The number of configurations is equal to the number of cameras

Figure 5.1 shows scenario1 from the proposed multi configuration framework, we consider the number of configurations is equal to the number of cameras as 3 where  $C_i = \{C_1 = SSD, C_2 = EfficientDet, C_3 = CenterNet\}$  which is equal to the number of cameras  $N = \{Cam_1, Cam_2, Cam_3\}$ . Those configurations are object detection models are selected from ten other models that are considered in this research. The reason of choosing those models is mentioned in chapter 3.

The processing latency for each configuration locally and by offloading for each model has been recorded in Table 3.1 in subsection 3.3.1. The maximum utilization for those methods are  $U_{max-C_i} = \{34.8, 46.9, 96.7\}$  for each configuration respectively. The power consumption for each device is randomly chosen between 0 and 1 as  $\rho = \{0.70, 0.80, 0.98\}$  while the transmission latency as  $L_{tran} = \{0.16, 0.23, 0.4\}$ . In addi-

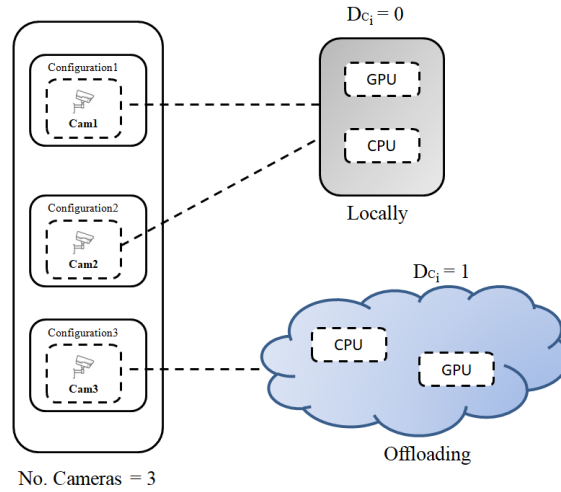


Figure 5.1: Multi configuration framework (scenario1).

tion, Table 5.1 contains the total available resources to process the tasks locally or by offloading .

We set the given information in Table 5.2 and after that we can follow algorithm 1 to calculate the cost for each configuration in this scenario to get Table 5.3.

$N$	Config.	$Lc_i$ ( $CPU_{loc}$ )	$Lc_i$ ( $GPU_{loc}$ )	$Lc_i$ ( $CPU_{off}$ )	$Lc_i$ ( $GPU_{off}$ )	$\rho$	$L_{trans}$
$Cam_1$	C1	5.57	9.40	8.55	8.04	0.70	0.16
$Cam_2$	C2	6.88	11.16	10.54	10.01	0.80	0.23
$Cam_3$	C3	7.32	23.01	17.16	9.79	0.97	0.04

Table 5.2: Object detection configurations for scenario1

$N$	$Cost_{C_i}$ ( $CPU_{loc}$ )	$Cost_{C_i}$ ( $GPU_{loc}$ )	$Cost_{C_i}$ ( $CPU_{off}$ )	$Cost_{C_i}$ ( $GPU_{off}$ )
$Cam_1$	6.27	10.10	8.71	8.20
$Cam_2$	7.68	11.96	10.77	10.24
$Cam_3$	8.29	23.98	17.20	9.83

Table 5.3: The cost for each configuration in scenario1

After that we calculate the overall cost ( $Cost_{all-C_i}$ ) that has eight solutions, they are set in Table 5.4 with all the combinations of solution. But we need to select only the minimum total cost that fulfill all the optimization problem constraints.

As a result we get only the highlighted solution in Table 5.4 where is the total minimum cost and possible solution is 23.44 that is coming from processing *Configuration<sub>1</sub>* and *Configuration<sub>2</sub>* locally and *Configuration<sub>3</sub>* remotely.

Locally ( $Dc_i = 0$ )	Offloading ( $Dc_i = 1$ )	$Cost_{all-C_i}$
$C_1$	$C_2 + C_3$	26.78
$C_2$	$C_1 + C_3$	26.53
$C_3$	$C_1 + C_2$	27.92
$C_2 + C_3$	$C_1$	24.58
$C_1 + C_3$	$C_2$	24.83
$C_1 + C_2$	$C_3$	23.44
$C_1 + C_2 + C_3$	–	21.49
–	$C_1 + C_2 + C_3$	29.87

Table 5.4: The total cost solutions for scenario1.

## 5.2 Scenario2: Each group of cameras under one configuration

The second scenario in this research is setting group of cameras under one configuration. We consider having four cameras and three configurations as it shows in Figure 5.2. The first, second and third are set on configuration 1, 2 and 3, respectively. The fourth camera is set under each configuration every time in order to calculate the overall cost for the system under each case.

### 5.2.1 Case1: $Cam_1$ and $Cam_4$ are set to configuration one ( $C_1$ )

The first case has different scenario where each group of cameras is set to one configuration. By assuming that  $C_i = \{C_1 = SSD, C_2 = Efficient, C_3 = CenterNet\}$  and the number of cameras are four  $N = \{Cam_1, Cam_2, Cam_3, Cam_4\}$ . Where  $Cam_1$  and  $Cam_4$  are set to configuration one ( $C_1$ ).

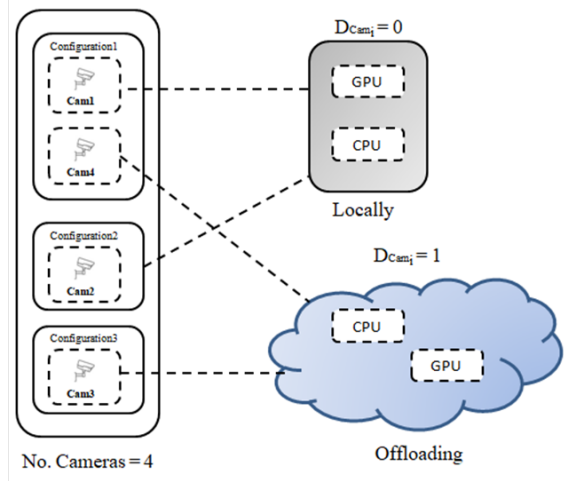


Figure 5.2: Multi configuration framework: Scenario2-Case1.

The maximum utilization for those configurations in this scenario are the same that set in the first scenario as  $U_{max-C_i} = \{34.8, 46.9, 96.7, 34.8\}$  for each configuration. Also, the available resources considered in this case are all set in Table 5.1.

After having all the parameters we set them in Table 5.5.

$N$	Config.	$Lc_i$ ( $CPU_{loc}$ )	$Lc_i$ ( $GPU_{loc}$ )	$Lc_i$ ( $CPU_{off}$ )	$Lc_i$ ( $GPU_{off}$ )	$\rho$	$L_{trans}$
$Cam_1$	C1	5.57	9.40	8.55	8.04	0.70	0.16
$Cam_2$	C2	6.88	11.16	10.54	10.01	0.80	0.23
$Cam_3$	C3	7.32	23.01	17.16	9.79	0.97	0.04
$Cam_4$	C1	5.57	9.40	8.55	8.04	0.95	0.89

Table 5.5: Object detection configurations for scenario2 - Case1

Based on the inference delay for each configuration, power consumption and transmission delay in Table 5.5, we computed the cost not for each configuration but for each camera because the transmission delay and power consumption for the cameras that under one configuration which are  $Cam_1$  and  $Cam_4$  are different from each other. Therefore, the cost of each camera differ from one camera to another. By doing so, we get the cost for each camera recorded in Table 5.6.

After having the cost for each camera, we need to compute the total minimum cost

$N$	$Cost_{Cam_i}$ ( $CPU_{loc}$ )	$Cost_{Cam_i}$ ( $GPU_{loc}$ )	$Cost_{Cam_i}$ ( $CPU_{off}$ )	$Cost_{Cam_i}$ ( $GPU_{off}$ )
$Cam_1$	6.27	10.10	8.71	8.20
$Cam_2$	7.68	11.96	10.77	10.24
$Cam_3$	8.29	23.98	17.20	9.83
$Cam_4$	6.52	10.35	9.44	8.93

Table 5.6: The cost for each camera in the scenario2 - Case1

for the cameras in the system by going through all the solutions that are 16 combinations of all the cameras, that are set in Table 5.7.

Locally ( $D_{Cam_i} = 0$ )	Offloading ( $D_{Cam_i} = 1$ )	$Cost_{all-Cam_i}$
$Cam_1$	$Cam_2 + Cam_3 + Cam_4$	42.64
$Cam_2$	$Cam_1 + Cam_3 + Cam_4$	35.66
$Cam_3$	$Cam_1 + Cam_2 + Cam_4$	36.68
$Cam_4$	$Cam_1 + Cam_2 + Cam_3$	35.83
$Cam_2 + Cam_3 + Cam_4$	$Cam_1$	38.80
$Cam_1 + Cam_3 + Cam_4$	$Cam_2$	38.98
$Cam_1 + Cam_2 + Cam_4$	$Cam_3$	37.96
$Cam_1 + Cam_2 + Cam_3$	$Cam_4$	39.28
$Cam_1 + Cam_2$	$Cam_3 + Cam_4$	33.22
$Cam_1 + Cam_3$	$Cam_2 + Cam_4$	38.07
$Cam_2 + Cam_3$	$Cam_1 + Cam_4$	37.38
$Cam_1 + Cam_4$	$Cam_2 + Cam_3$	33.39
$Cam_1 + Cam_2 + Cam_3 + Cam_4$	-	45.16
-	$Cam_1 + Cam_2 + Cam_3 + Cam_4$	48.07
$Cam_3 + Cam_4$	$Cam_1 + Cam_2$	37.59
$Cam_2 + Cam_4$	$Cam_1 + Cam_3$	36.57

Table 5.7: The total cost solutions for scenario2 - Case1

Table 5.7 shows all the solutions for the overall cost. Some of those combinations are not possible such as 45.16 and 48.07 as an overall cost for all the cameras because the total available resources locally or remotely are not sufficient enough to process all the cameras without waiting time. Therefore, they result in high total cost and not possible solution.

The possible and minimum total cost for the cameras in the system where the total



cost is 33.22 which is the result of processing the first and the second cameras locally and the third and fourth remotely.

### 5.2.2 Case2 : $Cam_2$ and $Cam_4$ are set to configuration two ( $C_2$ )

In this case, the fourth and the second cameras are set to *configuration<sub>2</sub>*, the latency for each configuration are set in Table 5.8.

$N$	Config.	$L_{C_i}$ ( $CPU_{loc}$ )	$L_{C_i}$ ( $GPU_{loc}$ )	$L_{C_i}$ ( $CPU_{off}$ )	$L_{C_i}$ ( $GPU_{off}$ )	$\rho$	$L_{trans}$
$Cam_1$	C1	5.57	9.40	8.55	8.04	0.70	0.16
$Cam_2$	C2	6.88	11.16	10.54	10.01	0.80	0.23
$Cam_3$	C3	7.32	23.01	17.16	9.79	0.97	0.04
$Cam_4$	C2	6.88	11.16	10.54	10.01	0.95	0.89

Table 5.8: Object detection configurations for scenario2 - Case2

After having all the parameters in Table 5.8 for each camera, we calculated the cost for each of them and we set them in Table 5.9.

$N$	$Cost_{Cam_i}$ ( $CPU_{loc}$ )	$Cost_{Cam_i}$ ( $GPU_{loc}$ )	$Cost_{Cam_i}$ ( $CPU_{off}$ )	$Cost_{Cam_i}$ ( $GPU_{off}$ )
$Cam_1$	6.27	10.10	8.71	8.20
$Cam_2$	7.68	11.96	10.77	10.24
$Cam_3$	8.29	23.98	17.20	9.83
$Cam_4$	7.83	12.11	11.43	10.90

Table 5.9: The cost for each camera in the scenario2 - Case2

Table 5.10 with the total cost shows the highlighted lowest possible solution is 35.44 where First, second and fourth cameras are processed locally and the third is offloaded. While, two other not possible solutions between the 16 combinations are 46.47 and 64.27 from processing all the cameras locally or offloading remotely.

Locally ( $D_{Cam_i} = 0$ )	Offloading ( $D_{Cam_i} = 1$ )	$Cost_{all-Cam_i}$
$Cam_1$	$Cam_2 + Cam_3 + Cam_4$	38.30
$Cam_2$	$Cam_1 + Cam_3 + Cam_4$	37.65
$Cam_3$	$Cam_1 + Cam_2 + Cam_4$	38.14
$Cam_4$	$Cam_1 + Cam_2 + Cam_3$	37.14
$Cam_2 + Cam_3 + Cam_4$	$Cam_1$	40.56
$Cam_1 + Cam_3 + Cam_4$	$Cam_2$	40.74
$Cam_1 + Cam_2 + Cam_4$	$Cam_3$	35.44
$Cam_1 + Cam_2 + Cam_3$	$Cam_4$	41.25
$Cam_1 + Cam_2$	$Cam_3 + Cam_4$	39.04
$Cam_1 + Cam_3$	$Cam_2 + Cam_4$	40.06
$Cam_2 + Cam_3$	$Cam_1 + Cam_4$	39.86
$Cam_1 + Cam_4$	$Cam_2 + Cam_3$	38.53
$Cam_1 + Cam_2 + Cam_3 + Cam_4$	-	46.47
-	$Cam_1 + Cam_2 + Cam_3 + Cam_4$	64.27
$Cam_3 + Cam_4$	$Cam_1 + Cam_2$	39.35
$Cam_2 + Cam_4$	$Cam_1 + Cam_3$	38.33

Table 5.10: The total cost solutions for scenario2- Case2

### 5.2.3 Case3: $Cam_3$ and $Cam_4$ are set to configuration three ( $C_3$ )

In the third case of scenario2, camera 4 and 3 are set to *Configuration<sub>3</sub>*. While other cameras are set similar to all the other cases as camera 1 and 2 under *Configuration<sub>1</sub>* and *Configuration<sub>2</sub>*, respectively. Table 5.11 shows the latency, power consumption and transmission delay for each camera.

$N$	Config.	$Lc_i$ ( $CPU_{loc}$ )	$Lc_i$ ( $GPU_{loc}$ )	$Lc_i$ ( $CPU_{off}$ )	$Lc_i$ ( $GPU_{off}$ )	$\rho$	$L_{trans}$
$Cam_1$	C1	5.57	9.40	8.55	8.04	0.70	0.16
$Cam_2$	C2	6.88	11.16	10.54	10.01	0.80	0.23
$Cam_3$	C3	7.32	23.01	17.16	9.79	0.97	0.04
$Cam_4$	C3	7.32	23.01	17.16	9.79	0.95	0.89

Table 5.11: Object detection configurations for scenario2- Case3

The cost of each camera has been calculated and recorded in Table 5.12. After that, we are able to compute the total cost for all the cameras in this case.

Table 5.13 shows all the combinations of solutions for all the cameras but not all of them are possible. Some solutions produce high latency when the available resources locally or remotely are not enough such as the first and the second rows where the total cost is 55.69 and 52.98, respectively. These combinations are not possible because processing camera 1 or 2 locally and the rest of cameras by offloading means one of the cameras needs to wait until the other finishes.

$N$	$Cost_{Cam_i}$ ( $CPU_{loc}$ )	$Cost_{Cam_i}$ ( $GPU_{loc}$ )	$Cost_{Cam_i}$ ( $CPU_{off}$ )	$Cost_{Cam_i}$ ( $GPU_{off}$ )
$Cam_1$	6.27	10.10	8.71	8.20
$Cam_2$	7.68	11.96	10.77	10.24
$Cam_3$	8.29	23.98	17.20	9.83
$Cam_4$	8.27	23.96	18.05	10.68

Table 5.12: The cost for each camera in the scenario2 - Case3

Other two not possible solutions found where the total cost for the cameras is 44.89 and 49.82 from processing all the cameras either locally or by offloading their tasks, respectively.

The minimum overall cost is found as 37.58 which is the result of processing camera 4 locally and camera 1,2 and 3 remotely as its highlighted in Table 5.13.

### 5.3 Performance evaluation

In this section, we evaluate and compare the three cases of scenario2 and state the differences between uni and multi configuration frameworks in terms of accuracy and overall cost for the system.

Locally ( $D_{Cam_i} = 0$ )	Offloading ( $D_{Cam_i} = 1$ )	$Cost_{all-Cam_i}$
$Cam_1$	$Cam_2 + Cam_3 + Cam_4$	55.69
$Cam_2$	$Cam_1 + Cam_3 + Cam_4$	52.98
$Cam_3$	$Cam_1 + Cam_2 + Cam_4$	38.45
$Cam_4$	$Cam_1 + Cam_2 + Cam_3$	37.58
$Cam_2 + Cam_3 + Cam_4$	$Cam_1$	56.42
$Cam_1 + Cam_3 + Cam_4$	$Cam_2$	62.69
$Cam_1 + Cam_2 + Cam_4$	$Cam_3$	40.16
$Cam_1 + Cam_2 + Cam_3$	$Cam_4$	41.03
$Cam_1 + Cam_2$	$Cam_3 + Cam_4$	45.66
$Cam_1 + Cam_3$	$Cam_2 + Cam_4$	39.84
$Cam_2 + Cam_3$	$Cam_1 + Cam_4$	39.64
$Cam_1 + Cam_4$	$Cam_2 + Cam_3$	38.97
$Cam_1 + Cam_2 + Cam_3 + Cam_4$	-	44.89
-	$Cam_1 + Cam_2 + Cam_3 + Cam_4$	49.82
$Cam_3 + Cam_4$	$Cam_1 + Cam_2$	51.20
$Cam_2 + Cam_4$	$Cam_1 + Cam_3$	38.77

Table 5.13: The total cost solutions for scenario2 - Case3

### 5.3.1 Overall cost comparison of the three cases of scenario2.

Figure 5.3 shows the comparison in the total cost for the three cases in scenario2 that have been explained in section 5.2.

Only in Figure 5.3 and Figure 5.5 the  $C_1, C_2, C_3$  and  $C_4$  represent  $Cam_1, Cam_2, Cam_3$  and  $Cam_4$ , respectively. The green dashed line shows the total cost of each solution in Scenario2-Case1 where we have four cameras and three configurations. The first and the forth are set to  $Configuration_1$ , while the second and third are set to  $Configuration_2$  and  $Configuration_3$ .

The solid purple line represent Case2 from scenario2 where the the second and the forth cameras are set to  $Configuration_2$ , while the first and third are set to  $Configuration_2$  and  $Configuration_3$ , respectively.

Finally, the red solid line demonstrates Case3 from scenario2. In this case we have the third and the fourth cameras are set to  $Configuration_3$ , while the first and second are set to  $Configuration_1$  and  $Configuration_2$ .

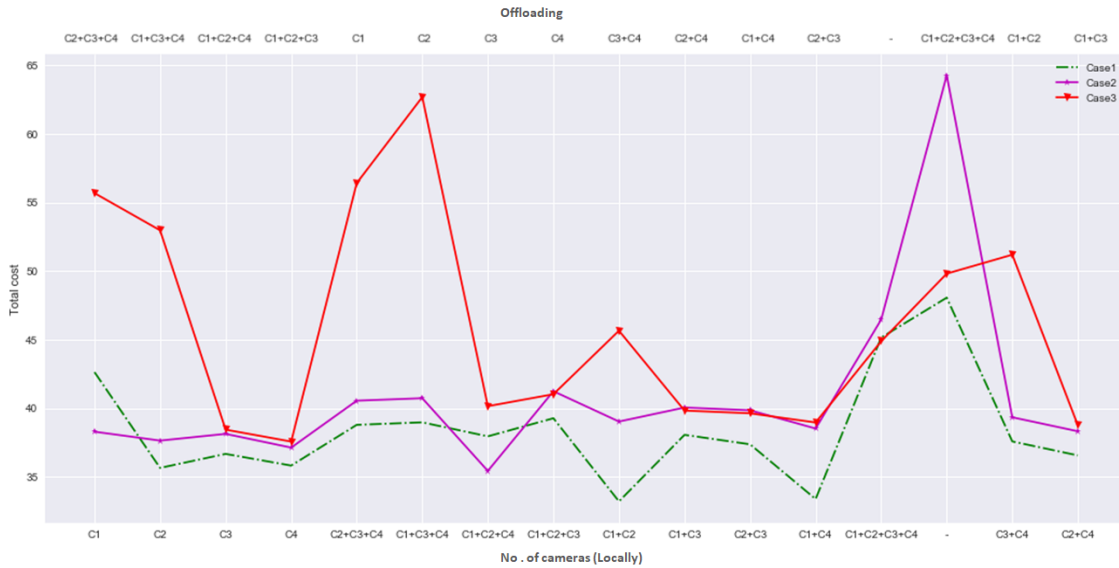


Figure 5.3: The total cost performance for multi configuration framework (scenario2).

In Figure 5.3 we can see the total cost increases when we set the fourth camera starting with  $Configuration_1$  to  $Configuration_3$ , but also we notice in some points of solutions the curve has been increased sharply due to the computing decision for all the cameras is locally or remotely. Some solutions are not possible due to the resource constraints therefore there is high total cost shown in the high points.

Comparing Figure 5.3 Case3 and  $Configuration_3$  from Figure 5.5, we can see that the optimal minimum cost for Case3 is 37.58 that belong to the multi configuration framework (proposed) but for  $Configuration_3$  that belong to the uni configuration framework(baseline) is 55.45, as a result we see a huge difference between the two frameworks in terms of total minimum cost. More on the differences between each case and configuration is explained in subsection 5.3.2.

### 5.3.2 Overall cost comparison of multi and uni configuration system

For comparative purpose, We draw Figure 5.4 to show a uni based configuration framework (baseline). In this framework, all the cameras in the system are set under one configuration. also in this specific figure we see that all the three cameras are set under configuration1 where the accuracy of the configuration is 20.

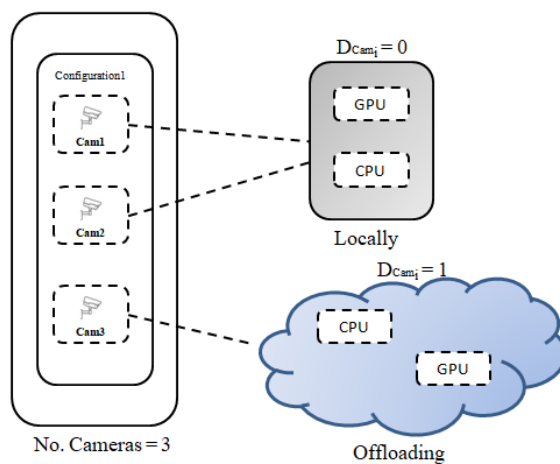


Figure 5.4: Uni configuration framework (baseline)

In this section, We consider a uni configuration system where firstly, all the cameras are set under  $Configuration_1$ , Secondly on  $Configuration_2$  and lastly  $Configuration_3$ .

Comparing Figure 5.5 and Figure 5.3 in terms of the total cost for each combination of solution. The curve in Figure 5.3 shows clearly the difference between each solution in all the three cases of scenario2. On the other hand, Figure 5.5 is showing the steady curve for the cost for each combination unless when the computing decision is done locally or by offloading for all the cameras. This steady behaviour reflects the fact that those cameras have the same configuration but different power consumption and transmission delay for each of which.

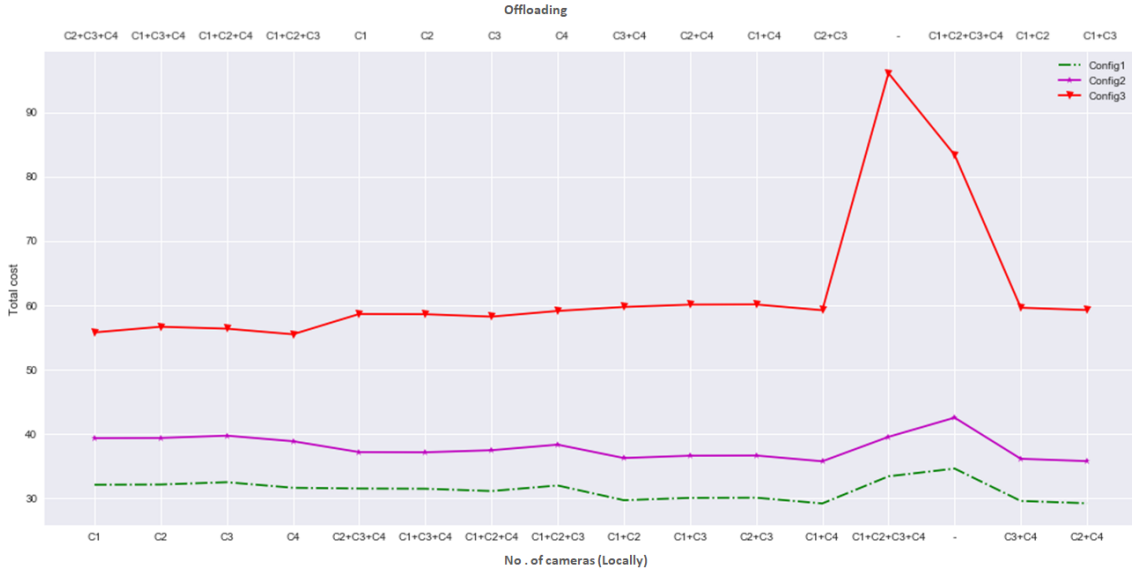


Figure 5.5: The total minimum cost for each solution in Uni configuration framework(baseline).

In all the possible solutions for multi configuration framework is achieving lower total cost in two cases than the uni configuration framework total cost.

### 5.3.3 Performance comparison between uni (baseline) and multi configuration framework (proposed) in terms of accuracy (mAP) and cost of each camera

In this sub section, we compare the uni and multi configuration system in terms of the accuracy and minimum cost of each camera in the system. The comparison is done on four cameras with different parameters such as power consumption and transmission delay for each camera.

In figure (a), the red line represents the accuracy(mAP) for multi configuration framework(proposed), while the green line indicates the accuracy(mAP) of each camera in uni configuration framework (baseline) sequentially. In figure (b) the blue line represents the cost of each camera in the Multi configuration framework(proposed), while the purple

line indicates the cost of each camera in the Uni configuration framework(proposed). The performance comparison is studied as follow:

1. Configuration1 and Case1 :

Figure 5.6 illustrates the difference between building the whole system on *Configuration<sub>1</sub>* as a uni configuration framework and building the system on Case1 from scenario2 where the accuracy of each camera is higher. We can see the mAP for all the cameras in Figure 5.6 (a) for uni configuration framework are the same as 20, therefore the cost for each camera in Figure 5.6 (b) ranges from 6 to 9. Table 5.14 shows clearly the differences between the mAP and Cost between the two frameworks.

$N$	$mAP_{Uni}$	$Cost_{Uni}$	$mAP_{Multi}$	$Cost_{Multi}$
$Cam_1$	20	6.27	20	6.27
$Cam_2$	20	8.27	33	7.68
$Cam_3$	20	8.08	40	9.83
$Cam_4$	20	6.52	20	9.44

Table 5.14: Performance comparison between the cost and accuracy(mAP) of configuration1 from uni configuration framework and Case1 from multi configuration framework.

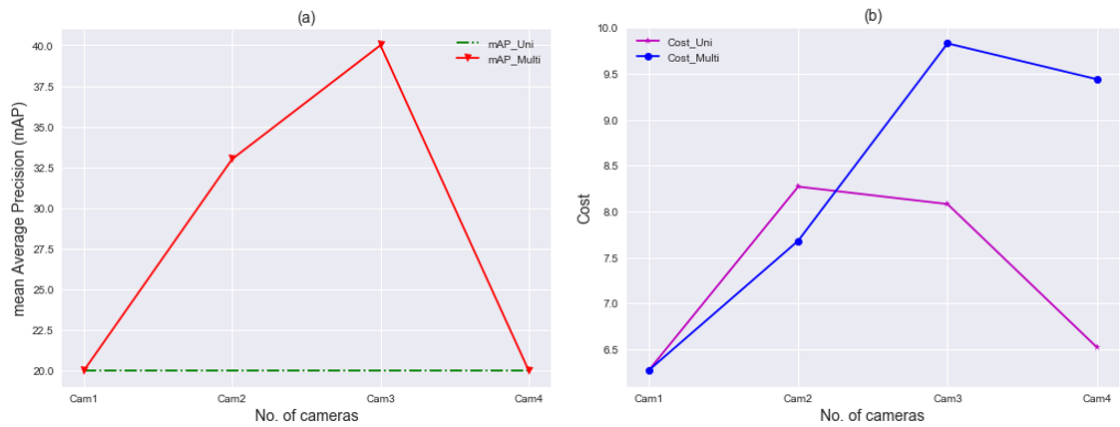


Figure 5.6: Performance comparison between Uni configuration (Baseline) build on Configuration1 and Case1 from Multi configuration (Proposed) in terms (a) Accuracy (mAP) and (b) Cost of each camera.

For multi configuration framework, although their accuracy is higher in two cameras which are cam2 and cam3 as 33 and 40 consecutively, the cost for those cameras are



slightly higher than the first and the fourth cameras that are set under *Configuration*<sub>1</sub>.

As a result, the difference in the total cost between those system in terms of the cost is very small but in terms of accuracy is very high. The total cost for uni configuration system is 29.14, while for the multi configuration is 33.22.

## 2. Configuration2 and Case2 :

The performance comparison in this case shows in Figure 5.7 (a) where all the cameras in the system are set to *Configuration*<sub>2</sub> with accuracy (mAP) of 33 for all the cameras in uni configuration framework (baseline). On the other hand, we have Case2 from scenario2 where the accuracy for two cameras are 33 and others as 20 and 40 that belongs to multi configuration framework(proposed). Table 5.15 shows the cost and the accuracy of each camera for the two frameworks.

$N$	$mAP_{Uni}$	$Cost_{Uni}$	$mAP_{Multi}$	$Cost_{Multi}$
$Cam_1$	33	7.58	20	10.10
$Cam_2$	33	7.83	33	7.68
$Cam_3$	33	10.24	40	7.83
$Cam_4$	33	10.05	33	9.83

Table 5.15: Performance comparison between the cost and accuracy of uni configuration2 system and multi configuration system.

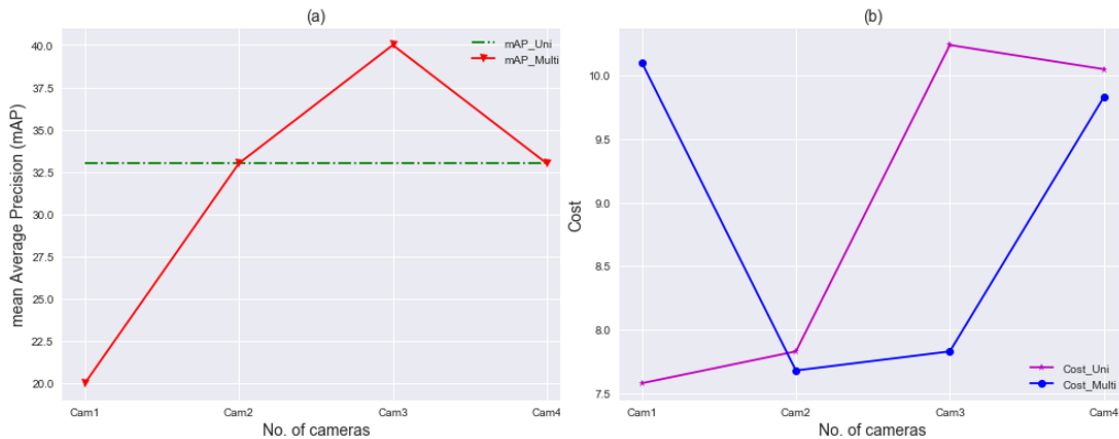


Figure 5.7: Performance comparison between Uni configuration (Baseline) build on Configuration2 and Case2 from Multi configuration (Proposed) in terms (a) Accuracy (mAP) and (b) Cost of each camera.

Figure 5.7 (b) shows clearly the cost of each camera in both frameworks ranges from 7 to 10. Also, it shows that the cost of three cameras that belongs to the multi configuration framework (proposed) is less than the the cost of the same cameras in the uni configuration framework (baseline).

### 3. Configuration3 and Case3 :

It has been compared a uni configuration framework (baseline) build with *Configuration3* and Case3 from the multi configuration framework (proposed) in terms of (a)the mAP and (b)the cost of camera. The mAP for all the cameras in the uni Configuration is 40, while for multi Configuration framework set within the range of 20 to 40. Table 5.16 shows the cost and the mAP of each camera for both uni and the multi configuration framework.

$N$	$mAP_{Uni}$	$Cost_{Uni}$	$mAP_{Multi}$	$Cost_{Multi}$
$Cam_1$	40	8.27	20	8.27
$Cam_2$	40	17.32	33	8.71
$Cam_3$	40	10.02	40	10.77
$Cam_4$	40	19.85	40	9.83

Table 5.16: Performance comparison between the cost and accuracy of uni configuration3 system and multi configuration system.

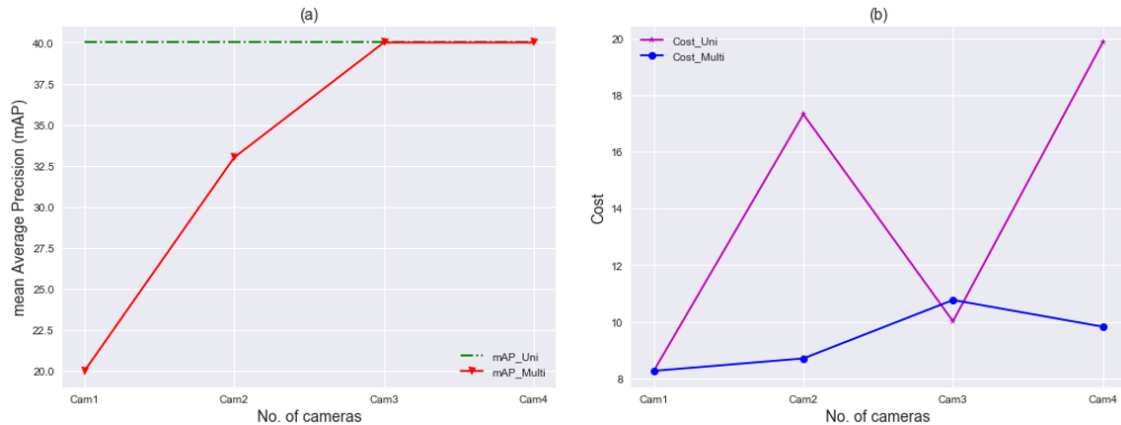


Figure 5.8: Performance comparison between Uni configuration (Baseline) build on Configuration3 and Case3 from Multi configuration (Proposed) in terms (a) Accuracy (mAP) and (b) Cost of each camera.

Figure 5.8 shows the cost of the cameras in uni configuration framework in the range of 8 to 20, while in the multi Configuration is between 8 and 10. In terms of total cost, for the uni framework is 55.46 which is very high comparing to the our proposed framework where the total cost is 37.58.

For better comparison we merged the two objectives (mAP and Cost) in Figure 5.9 where the red dots represent the cameras that belongs to the proposed framework (Multi configuration), while the blue once belongs to the baseline framework (Uni configuration). The figure shows the optimal and non dominant solutions are the cameras in the multi configuration framework (proposed) are more than the cameras that belongs to uni configuration framework(baseline) by 75%. Also second level optimal solutions are very close to be optimal comparing to the baseline cameras .Finally, for the baseline framework, it is shown that optimizing the mAP objective is worsen the other objective which is the cost,while in the proposed framework where we can see going from Case1 to Case3, we can optimize even the cost. Therefore, our proposed framework outperform the baseline by optimizing both the mean Average Precision (mAP) and the cost for each camera in the system.

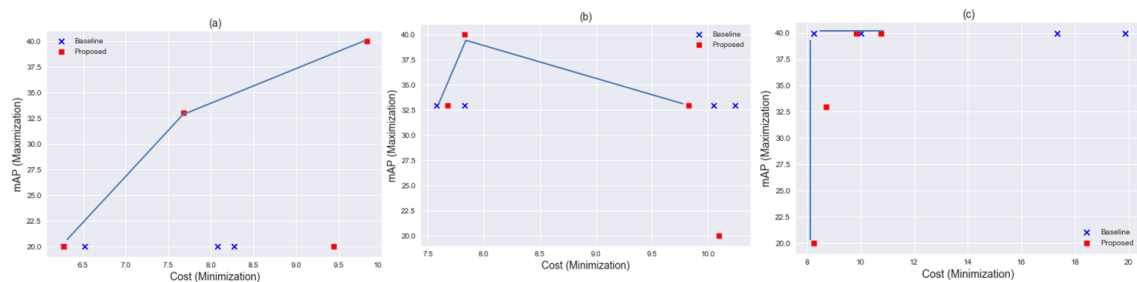


Figure 5.9: Performance comparison between (a) Case1, (b) Case2 and (c) Case3 in terms of minimizing the cost and maximizing the mAP.



# Chapter 6 Conclusions

## 6.1 Conclusions

In this thesis, in order to find a trade off between the accuracy and the latency of the detectors in the surveillance system, we proposed a multi configuration framework (Hybrid). The proposed framework sets each camera or a group of cameras in the system to a configuration that is different in accuracy and latency based on the operating system requirements. Each configuration is an object detection method, these methods are classified as light one stage detector, heavy one stage detector and two stage detector. We outlined the methods applied and we listed some examples of scenarios as cases. The scenarios and the cases were important to determine and evaluate the efficiency of our proposed framework. Then, we described the experiment that we have performed inference on each and we stated the findings.

In the first phase, we performed evaluation on three cases of scenario2 where each group of camera is set under a configuration. We performed on three cases of scenario2 where each group of cameras are set to one configuration. In this phase, for Case1 we set the first and fourth camera on configuration1 ( $C_1$ ) and the second and third cameras on configuration2 and 3, respectively. Results showed that the minimum overall cost is 33.22 as a result of processing the first and second camera locally, while for the second and third

remotely. After that we have Case2 where the second and the forth cameras are set under configuration2( $C_2$ ) and the first and forth cameras are set under configuration 1 and 3 consecutively. Findings showed a minimum overall cost as 35.44 where the first ,second and forth cameras process their task locally, while the third is offloaded remotely. Finally, Case3 where the first and forth camera are set under configuration3 while the second and third are set under configuration2 and 3,respectively. The findings showed the lowest total cost for this case is 37.58 as a result of processing the fourth camera locally and the first, second and third cameras remotely.

The second phase is the performance comparison between uni and multi configuration framework in terms of mean Average Precision (mAP) and total cost of the system. Firstly, comparison between uni configuration system with all the cameras set on configuration1 ( $C_1$ ) where the accuracy is 20 and multi configuration where only two cameras with the mAP is 20 and others are 33 and 40. The results showed the total cost is 29.14 while for multi configuration is 33.22. Secondly, Performance comparison between uni configuration system with configuration2 ( $C_2$ ) where the accuracy is 33 and case2 of scenario2. The findings showed the minimum cost all of uni configuration is 35.7, while for the multi configuration is 35.44 which is lower and higher accuracy. Finally, the comparison between uni configuration set on configuration3 ( $C_3$ ) where the accuracy is 40 and case3 of scenario2. The total minimum cost is 55.45 for uni configuration framework and 37.58 which is much lower in multi configuration system.

To conclude, with the proposed multi configuration framework (Hybrid), we could optimize the two objectives which are the mAP and the cost of each camera in the system. And the last proves the initial hypothesis of finding trade-off between very important factors of object detection in surveillance systems.In this regard, we were successful

in finding a framework that can balance the aforementioned factors. Further experiment with more real-life data and more configurations will be useful in finding the best configuration with each camera according to the operating system requirements.

## **6.2 Future work**

For the future work of this project, we would suggest using more configurations and more number of cameras with also setting the requirements of each camera to test the framework.

We also suggest using real time scenario to ensure the quality and authenticity of this work.





# References

- [1] Anaconda software distribution, 2020.
- [2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pages 265–283, 2016.
- [3] S. M. Beitzel, E. C. Jensen, and O. Frieder. MAP, pages 1691–1692. Springer US, Boston, MA, 2009.
- [4] S. Bi and Y. Zhang. Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. IEEE Transactions on Wireless Communications, 17(6):4177–4190, 2018.
- [5] E. Bisong. Google Colaboratory, pages 59–64. Apress, Berkeley, CA, 2019.
- [6] J. Chen, K. Li, Q. Deng, K. Li, and P. S. Yu. Distributed deep learning model for intelligent video surveillance systems with edge computing. IEEE Transactions on Industrial Informatics, pages 1–1, 2019.

- [7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, pages 248–255, 2009.
- [8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. International Journal of Computer Vision, 88(2):303–338, 2010.
- [9] A. M. Ghosh and K. Grolinger. Deep learning: Edge-cloud data analytics for iot. In 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE), pages 1–7, 2019.
- [10] Y. Gong, C. Lv, S. Cao, L. Yan, and H. Wang. Task offloading in mobile fog computing by classification and regression tree. Peer-to-Peer Networking and Applications, 2020(1):69, 2020.
- [11] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. CoRR, abs/1703.06870, 2017.
- [12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015.
- [13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861, 2017.
- [14] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon. Clio: enabling automatic compilation of deep learning pipelines across iot and cloud. pages 1–12, 09 2020.

- [15] Y. Huang, F. Wang, F. Wang, and J. Liu. Deepar: A hybrid device-edge-cloud execution framework for mobile deep learning applications. In IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pages 892–897, 2019.
- [16] J. D. Hunter. Matplotlib: A 2d graphics environment. Computing in Science & Engineering, 9(3):90–95, 2007.
- [17] I. Khokhlov, E. Davydenko, I. Osokin, I. Ryakin, A. Babaev, V. Litvinenko, and R. Gorbachev. Tiny-yolo object detection supplemented with geometrical data. CoRR, abs/2008.02170, 2020.
- [18] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. CoRR, abs/1612.03144, 2016.
- [19] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. CoRR, abs/1405.0312, 2014.
- [20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. CoRR, abs/1512.02325, 2015.
- [21] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. CoRR, abs/1603.06937, 2016.
- [22] D. Rahbari and M. Nickray. Deep learning-based computation offloading with energy and performance optimization. EURASIP Journal on Wireless Communications and Networking, 13(1):104–122, 2020.

- [23] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. CoRR, abs/1506.02640, 2015.
- [24] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. CoRR, abs/1804.02767, 2018.
- [25] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6):1137–1149, 2017.
- [26] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. CoRR, abs/1506.01497, 2015.
- [27] S. H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. D. Reid, and S. Savarese. Generalized intersection over union: A metric and A loss for bounding box regression. CoRR, abs/1902.09630, 2019.
- [28] C. Samplawski, J. Huang, D. Ganesan, and B. M. Marlin. Towards objection detection under iot resource constraints: Combining partitioning, slicing and compression. AIChallengeIoT '20, page 14–20, New York, NY, USA, 2020. Association for Computing Machinery.
- [29] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. CoRR, abs/1801.04381, 2018.
- [30] K. Simonyan and A. Zissermanl. Very deep convolutional networks for large-scale image recognition. 2014.

- [31] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. CoRR, abs/1602.07261, 2016.
- [32] M. Tan, R. Pang, and Q. V. Le. Efficientdet: Scalable and efficient object detection. CoRR, abs/1911.09070, 2019.
- [33] S. Tuli, N. Basumatary, and R. Buyya. Edgelens: Deep learning based object detection in integrated iot, fog and cloud computing environments. In 2019 4th International Conference on Information Systems and Computer Networks (ISCON), pages 496–502, 2019.
- [34] G. Van Rossum and F. L. Drake. Python 3 Reference Manual. CreateSpace, Scotts Valley, CA, 2009.
- [35] B. Yang, X. Cao, J. Basse, X. Li, T. Kroecker, and L. Qian. Computation offloading in multi-access edge computing networks: A multi-task learning approach. In ICC 2019 - 2019 IEEE International Conference on Communications (ICC), pages 1–6, 2019.
- [36] T.-J. Yang, Y.-H. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6071–6079, 2017.
- [37] Z. Zaheer, A. Usmani, E. Khan, and M. A. Qadeer. Aerial surveillance system using uav. In 2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN), pages 1–7, 2016.
- [38] Y. Zhang, J.-H. Liu, C.-Y. Wang, and H.-Y. Wei. Decomposable intelligence on cloud-edge iot framework for live video analytics. IEEE Internet of Things Journal, 7(9):8860–8873, 2020.

- [39] Y. Zhao, Q. Chen, W. Cao, W. Jiang, and G. Gui. Deep learning based couple-like cooperative computing method for iot-based intelligent surveillance systems. In 2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), pages 1–4, 2019.
- [40] X. Zhou, D. Wang, and P. Krähenbühl. Objects as points. CoRR, abs/1904.07850, 2019.