An Accurate and Fast Animal Species Detection System for Embedded Devices

By

Mai Mahmoud Ibraheam
B.Sc., El-Mansoura University, Egypt, 2004
M.Sc., El-Mansoura University, Egypt, 2009

A Dissertation Submitted in Partial Fulfillment of the

Requirements for The Degree of

DOCTOR OF PHILOSOPHY

In the Department of Electrical and Computer Engineering

University of Victoria

We acknowledge with respect the Lekwungen peoples on whose traditional territory the university stands and the Songhees, Esquimalt, and W̱SÁNEĆ peoples whose historical relationships with the land continue to this day.

An Accurate and Fast Animal Species Detection System for Embedded Devices

By

Mai Mahmoud Ibraheam
B.Sc., El-Mansoura University, Egypt, 2004
M.Sc., El-Mansoura University, Egypt, 2009

Supervisory Committee

Prof. Kin Fun Li, Co-Supervisor
(Department of Electrical and Computer Engineering)

Prof. Fayez Gebali, Co-Supervisor
(Department of Electrical and Computer Engineering)

Prof. Rishi Gupta, Outside Member
(Department of Civil Engineering)

Abstract

Object detection is one of the vital and challenging tasks in the field of computer vision. It supports a wide range of applications in real life, such as surveillance, autonomous driving, and medical diagnostics. Object detection techniques aim to identify and localize objects of certain target classes within an image and assign each object to a corresponding class label. These techniques vary in their network architecture, training strategy, and optimization function.

In this dissertation, an investigation into object detection is presented, with a specific emphasis on animal species detection. The research aims to mitigate the negative impacts of wildlife-human conflicts (WHCs) and wildlife-vehicle collisions (WVCs), particularly in remote wilderness regions/trails, urban areas/backyards, and on highways. Our goal is to enhance the accuracy and speed of animal species detection to ensure safer environments for both humans and wildlife.

The research involves a comprehensive analysis of object detection techniques based on R-CNN models. Four different R-CNN models and a deformable convolutional neural network are applied on three wildlife datasets, and results are evaluated using four metrics. This comprehensive analysis informs the proposal of a novel animal species detection system. The results illustrate the system's high accuracy in distinguishing between different object categories such as animals, humans, and vehicles, as well as in identifying specific animal species. This work aims to develop an automated labelling and annotation system that eliminates the need for human intervention, thereby saving time and costs. Furthermore, it seeks to contribute to the development of robust and reliable systems which can be applied to various aspects of biological sciences, such as wildlife monitoring, conservation, and management.

A key proposal of the research is to develop WHCs and WVCs real-time mitigation systems based on a lightweight animal species detection model (M-YOLO) derived from YOLOv2. Multi-level features merging is employed by adding a new pass-through layer to improve the feature extraction ability and accuracy of YOLOv2. Moreover, the two repeated 3×3 convolutional layers in the seventh block of the YOLOv2 architecture are removed to reduce computational complexity, and thus increase detection speed without reducing accuracy. Animal species detection methods based on regular Convolutional Neural Networks (CNNs) have been widely applied; however,

these methods are difficult to adapt to geometric variations of animals in images. Thus, a modified YOLOv2 with the addition of deformable convolutional layers (DCLs) was proposed to resolve this issue. Our experimental results show that the proposed model outperforms the original YOLOv2 by 5.0% in accuracy and 12.0% in speed. Furthermore, our analysis shows that the proposed model is more suitable for deployment on embedded devices than YOLOv3 and YOLOv4.

To further enhance the M-YOLO model and achieve real-time alerts on low-power and resource-constrained devices, the research proposes the integration of two key ideas: the Motion-selective Control Frames (MCF) algorithm and a parallel processing technique. These enhancements aim to minimize the detection processing delay and power consumption, which are crucial for the efficient operation of low-power, computationally limited embedded devices. Importantly, these improvements are achieved while maintaining detection accuracy.

Table of Contents

List of Figures

List of Tables

List of Publications

1. M. Ibraheam, F. Gebali, K. F. Li, and L. E. Sielecki, "Animal species recognition using deep learning," in *International Conference on Advanced Information Networking and Applications*, pp. 523-532, Springer, Cham, 2020.
2. M. Ibraheam, K. F. Li, F. Gebali, and L. E. Sielecki, "A Performance Comparison and Enhancement of Animal Species Detection in Images with Various R-CNN Models." In *AI* 2, no. 4, pp. 552-577, 2021.
3. M. Ibraheam, K. F. Li, and F. Gebali, "An Accurate and Fast Animal Species Detection System for Embedded Devices," in *IEEE Access*, vol. 11, pp. 23462-23473, 2023.
4. M. Ibraheam, K. F. Li, and F. Gebali, "MCFP-YOLO Animal Species Detector for Embedded Systems" submitted to *Electronics*

Acronyms

Adam	Adaptive Moment Estimation
Acc	Accuracy
AI	Artificial Intelligence
ANNs	Artificial Neural Networks
AP	Average Precision
BC	British Columbia
BCCOS	BC Conservation Office Service
BCMoTI	British Columbia Ministry of Transportation and Infrastructure
BN	Bayesian Network
CNNs	Convolutional Neural Networks
CSPDarkNet-53	Cross-Stage Partial DarkNet-53
CPU	Central Processing Unit
D-CNN	Deformable Convolutional Neural Network
DCLs	Deformable Convolutional Layers
DFG	Dataflow Graph
DNNs	Deep Neural Networks
FB	FlannBased
FCLs	Fully Connected Layers
FCN	Fully Convolutional Network
FIFO	First-In-First-Out
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPGA	Field-Programmable Gate Array
FPS	Frame Per Second
GPU	Graphic Processing Unit
HOG	Histogram of Oriented Gradients
ICBC	Insurance Corporation of British Columbia
IoU	Intersection over Union
Inter-Process Communication	IPC
IR	Infrared
JSON	JavaScript Object Notation
K-NN	K-Nearest Neighbours
LAN	Local Area Network
LBP- Adaboost	Local Binary Pattern Adaptive boost
LCADS	Live Camera Animal Detection System
mAP	mean Average Precision
MCF	Motion-selective Control Frames
MCF-YOLO	Integration of MCF algorithm into M-YOLO
MCFP-YOLO	Integration of MCF algorithm and parallel processing into M-YOLO
MIMD	Multiple Instruction Multiple Data
ML	Machine Learning

Mutex	Mutual Exclusion
M-YOLO	Modified You Only Look Once
MCF-YOLO	Modified You Only Look Once Using MCF Algorithm
MCFP-YOLO	MCF-YOLO Model Using Parallel Processing Implementation
NB	Naïve Bayes
NMS	Non-Maximum Suppression
PIR	Passive Infra-Red
PRC	Precision-Recall Curve
R-CNN	Region-based Convolution Neural Network
ReLU	Rectifier Linear Unit
ResNet	Residual Network
RF	Random Forest
RoIs	Regions of Interest
RPN	Region Proposal Network
RP4B	Raspberry Pi 4 Model B
SIFT	Scale-Invariant Feature Transform
SIMD	Single Instruction Multiple Data
SURF	Speeded-Up Robust Features
SSD	Single Shot Detector
SVM	Support Vector Machines
TN	True Negative
TP	True Positive
UVic	University of Victoria
VIADS	Video Image Animal Detection System
VOC	Visual Object Classes
WAN	Wide Area Network
WHCs	Wildlife-human conflicts
WVCs	Wildlife-vehicle collisions
YOLO	You Only Look Once

Acknowledgements

First and foremost, I would like to express my deepest appreciation and prayers to the *Almighty Allah*, whose blessings have given me the strength and ability to think, learn, and accomplish this journey.

To my dear parents, *Mahmoud* and *Safenaz*, your boundless love, support, and guidance have been the foundation of my success. Your sacrifices and encouragement have not only shaped who I am but have also empowered me to reach this significant milestone in my academic journey.

To my loving husband, *Khalid*, your patience, understanding, and love have been an endless source of inspiration and resilience. Your unwavering support has been invaluable in the completion of my PhD. My adorable children, *Adham* and *Saleem*, you are my greatest motivation. Your joy and innocence have been my stress relief during my most challenging times.

More specially, I am profoundly indebted to my supervisors, *Prof./ Kin Fun Li* and *Prof./ Fayez Gebali*. Your invaluable time, effort, guidance, patience, and insights have significantly shaped this research. Your continual support, advice, and mentorship have been instrumental to my growth as a researcher, and for that, I am extremely grateful to be you student. In working with you, I've been exposed to an entirely new world of research and learning.

Further, I extend my gratitude to *Prof./ Rishi Gupta*, for his support and valuable advice, as well as for his contribution to my supervisory committee and his time spent reviewing my dissertation.

I am also deeply appreciative of BCMoTI's generosity in sharing a valuable dataset, which has significantly contributed to the success of our research.

In my pursuit of knowledge and in the journey of life, you have all been my pillars of strength. I sincerely thank each of you, as well as everyone else who made this dissertation possible. Whether it be conversations with my colleagues and friends and valuable feedback from peer reviewers have all inspired me to continue striving to learn beyond what I know.

Mai Mahmoud Ibraheam

Victoria, BC, Canada

Dedication

I dedicate this work to my supportive family. A special feeling of gratitude to my loving parents, Mahmoud and Safenaz, who have always loved me unconditionally.

This work is also dedicated to my beloved husband Khalid, who have been a constant source of support and encouragement during the challenges of graduate school and life.

Also, to my kids, Adham and Saleem who are the joy and happiness of my life.

Chapter 1

Introduction

1.1 Problem Statement and Motivation

Wildlife-vehicle and wildlife-human encounters cause a significant danger to both humans and wildlife, resulting in a wide range of adverse effects that extend from physical injuries to financial consequences. In addition to addressing these risks, there is a critical need to assist biologists, researchers, and conservationists in monitoring and analyzing wildlife behavior to develop effective strategies for wildlife management and conservation efforts to help in reducing the risks associated with wildlife encounters and ensuring the safety of both humans and wildlife.

Wildlife-vehicle collisions (WVCs) have been increasing in North America over the past few decades due to the increase in traffic volume and higher vehicle speed limits [1] [2]. Canada has numerous national and provincial parks teeming with wildlife, has highways passing through forests where animals movement are unpredictable; hence, the probabilities of WVCs are high. As shown in Fig. 1.1, between the years 2000 and 2014, 474 humans fatalities occurred on Canadian roads as a result of WVCs [3] [4].

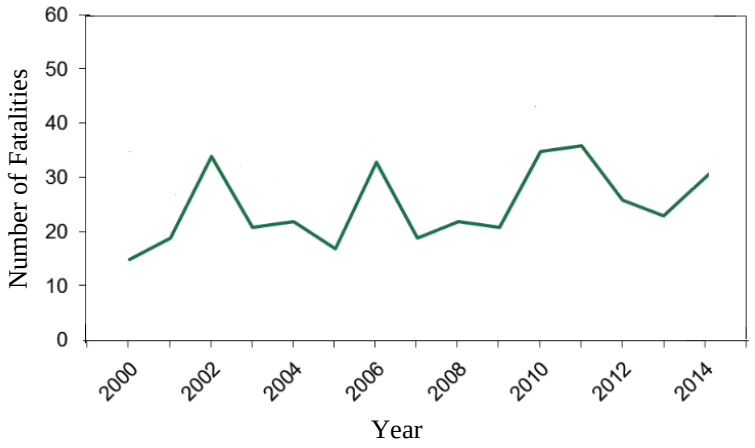


Figure 1.1: Number of humans died annually in Canada from 2000 to 2014 because of WVCs involving large animals such as bear, deer, elk, and moose [4].

In the Canadian province of British Columbia (BC), an average of 11,000 WVCs occur each year, as reported by the Insurance Corporation of British Columbia (ICBC) [5]. Moreover, on an annual basis, there are approximately 870 injuries, 4 fatalities, and 6,100 wildlife deaths resulting from WVCs in BC [5] [6]. According to a report by British Columbia Ministry of Transportation and Infrastructure (BCMoTI), the recorded number of wildlife deaths represents only 25% to 35% of the actual number, as some animals die away from the collision site [7] [8]. Furthermore, their statistics show that approximately 80% of WVCs involve bear, deer, elk, moose, and mountain goat [6] [9]. From monetary perspective, the total claim cost associated with WVCs in BC amounts to about $41 million annually [10], with BCMoTI spending approximately $700,000 every year for highway clean-up and carcass removal [6] [8]. Therefore, mitigating WVCs on highways is a significant research topic for humans, animals, the environment, and the economy.

Wildlife-human conflicts (WHCs) is becoming a serious problem in Canada over the past few decades [11] [12] [13]. Several factors have contributed to this escalation: (1) Canada has 48 national parks which covers 3% of its landmass [14], the more human activities take place within these parks, the more WHCs that are life-threating for both, (2) the expansion of agriculture, (3) the expansion of urban areas towards wildlife habitats or territories, (4) climate change, for example the increasing heat waves and wildfires, has forced wildlife to seek survival in urban areas [15], (5) the increasing of wildlife sightings in urban areas, with wildlife became urban dwellers as human activities decreased during the COVID-19 pandemic lockdown [16] [17] [18].

In North America, BC ranks third in terms of the number of wildlife-human encounters, following California and Colorado [19]. Many BC residents report sightings of wildlife inside or in front of their houses. WHCs pose a life-threating risk to both humans and wildlife. According to a report from the BC Conservation Officer Service (BCCOS), between 2011 and 2019, officers killed 3,314 black bears and 590 cougars across the province as a result of encounters with humans [20]. The BCCOS statistics further reveal that from January 2020 to August 2021, 748 black bears and 69 cougars were killed, indicating a significant increase in such incidents during that period [21-25].

There are several measures available to mitigate WVCs and WHCs in various settings, including highways, remote wilderness regions/trails, and urban areas/backyards. Some examples of these measures range from the traditional methods to more advanced wireless sensor-based methods.

The traditional methods employed for mitigation include the use of : wildlife warning reflectors [26] [27], wildlife warning signs, wildlife exclusion fencing, road lighting, wildlife crossings such as overpasses and underpasses, road planning considerations like road tunnels and elevated roadway [26] [28], wildlife translocation [29], and repellents such as chemical barrier [29].

In addition to these traditional measures, more advanced and effective approaches involve the use of wireless sensor-based systems, such as animal detection systems [30]. These systems utilize sensors to detect the presence of wildlife near roadways and provide real-time warnings to drivers, helping them take precautionary measures and reduce the risk of collisions.

Table 1.1 shows that road planning, which includes features like road layout, wildlife overpasses, and underpasses, has the potential to achieve a 100% reduction in WVCs. However, the cost is extremely high and applying such measure is infeasible and impractical in regions like BC due its challenging geographical features [31]. In contrast, animal detection systems offer several advantages over the other shown mitigation measures. These systems can be installed without any road constructions, allowing animals to move safely without any restrictions. They can also be relocated if animals change their crossing locations, making them portable systems. They have been proven to reduce WVCs by 87% at a reasonable cost [31] [32]. Based on these advantages, our objective is to improve the effectiveness and efficiency of animal detection systems, ensuring they provide accurate and timely detection of wildlife not only near highways but also along trails, and in urban areas such as backyards and school playgrounds.

Table 1.1: The estimated effectiveness of seven mitigation measures on reducing WVCs and WHCs.

#	Mitigation Measures	Effectiveness	Comments
1	Wildlife warning reflectors	1% [32]	No control on animals' movement directions, the reflected lights sometimes attract animals
2	Wildlife warning signs	26% [31]	Drivers and hikers habituate to them
3	Wildlife exclusion fencing	40% [31]	Isolate animals' population
4	Road lighting	65% [33]	Expensive and infeasible
5	Wildlife crossings with fences	86% [31]	Expensive, fences are needed to direct animals' movement
6	Animal detection systems	87% [31]	The effectiveness of these systems can be improved, animals are allowed to move across the landscape
7	Road planning	100% [31]	Expensive

1.2 Animal Detection Systems

Animal detection systems are designed to detect animals that are close to highways, remote wilderness regions/trails, and urban areas/backyards. Once an animal is detected, an appropriate action must be taken by a reliable mitigation system to warn drivers, hikers, residents, and even the animals themselves. Two main types of animal detection systems have been developed in different countries to assist in the detection of ground-based moving animals [30] [34] [35] [36]. The first type is area-cover animal detection systems, which can detect animals within a specific detection area and sensor range, such as: live video cameras [8] [37], and passive infra-red (PIR) motion detecting cameras which are designed to activate and start capturing images when motion is detected within their field of view [38]. The second type is break-the-beam animal detection systems, which can detect animals when their bodies reduce or block the used wireless sensor beam [39], such as: ultrasonic acoustic waves [40], laser [41], microwave radio [30], and doppler radar sensors [42].

There are several considerations that have to be taken to select a reliable animal detection system for mitigating WVCs and WHCs such as: animal size, environmental conditions, and landscape characteristics [30] [43]. In the case of BC, the landscape is not flat, as it is characterized

by a lot of rocky areas, curves, on and off ramps, and dense vegetation with abundant trees. Therefore, the effectiveness of break-the-beam animal detection systems will be challenging due to the increase in false alarms of animal detection, as these systems require free space between the sensors. As a result of that, our work will focus only on the area-cover animal detection systems.

PIR motion detector cameras [44] are popular tools in area-cover animal detection systems due to their: (1) ease of use, (2) reliability, (3) ability to detect moving objects, (4) high definition images, (5) capturing additional information such as timestamp (time and date), temperature and moon phase, and (6) invisible infrared flashes, eliminating disturbances caused by the white light flashes or the camera noises that may startle animals [45]. Many modern PIR cameras allow users to configure settings, for examples a user can set the camera to capture from one to ten images per second or to record a short video (e.g., 10-30 seconds) when activity is spotted [45]. These specifications make PIR camera a powerful tool for our research in mitigating WVCs and WHCs. Moreover, IR live video cameras are designed for real-time surveillance and use infrared to stream videos both day and night. These cameras offer continuous monitoring and do not depend on a triggering mechanism. Once an animal is detected at day/night, a reliable mitigation system is activated to warn drivers, hikers, or residents. Additionally, specific animal warnings can also be issued. However, it is important to note that any movements within the PIR camera's detection area such as moving grasses, or tree branches (no moving animals), will trigger image capture, resulting in many false alarms. Therefore, it is essential to propose an object detection model or algorithm capable of identifying and localizing objects in the images under various weather conditions, both day and night. This proposed model should demonstrate high accuracy in real-time with low power consumption, cost, and ease of implementation for effective WVCs and WHCs mitigation systems. These systems consist of two subsystems: a detection subsystem and a warning subsystem, as shown in Fig. 1.2.

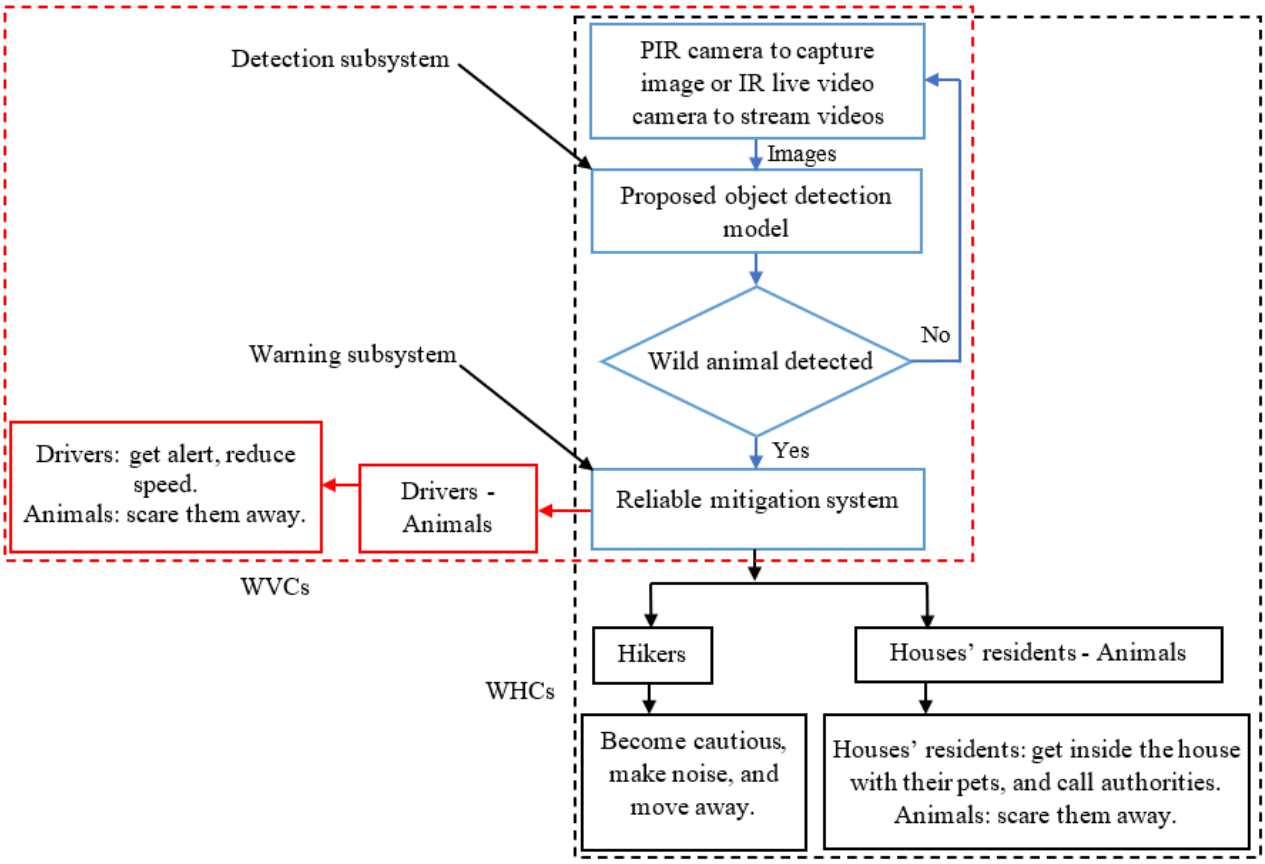


Figure 1.2: Block diagram of the mitigation systems. The red dashed box represents the WVCs mitigation system, and the black dashed box represents the WHCs mitigation system.

To develop effective mitigation systems based on animal detection model, a comparative study of managing WVCs and WHCs across three different environments: highways, trails, and urban areas is presented in Table 1.2. Each setting has challenges and opportunities for managing these collisions and conflicts. This comparison seeks to understand the requirements and equipments employed in each environment to improve WVCs and WHCs management, making our strategies more effective and suitable for each specific environment. The factors considered in this comparison include:

1. **Power Source**: Power requirements vary across various environments. For highway and urban area settings, we have access to the power grid and can use wired solutions. However, in remote wilderness trails, access to the power grid may not always be available, so the use of batteries or alternative energy sources such as solar panels is

required. Even in grid-accessible areas, employing alternative energy sources can be beneficial for reducing costs and lessening environmental impact.

2. **Camera**: The camera is an essential part of the mitigation system as it is the initial point of data collection. Camera type, resolution, and placement are critical factors to consider when managing WVCs and WHCs.
    - Highways: Given the high speed and frequency of events involving both vehicles and animals on highways, IR live video cameras/night vision cameras are effective in capturing the reflection of infrared light from wildlife, regardless of lighting conditions [46]. Their wide field of view enables continuous monitoring of a larger area, facilitating the earlier detection of animals, even those at significant distance from the camera's location. While PIR cameras, which also typically have a wide field of view, are triggered by movement within their detection range, they can sometimes miss far-off movement. Additionally, high-resolution cameras are essential on highways, particularly for preventing blurring of fast-moving objects, which is crucial for accurately identifying wildlife and triggering timely alerts. Camera placement is typically ad hoc, focused on capturing the road and the surrounding areas.
    - Trails: PIR cameras, which detect changes in IR radiation due to motion, capturing images when movement is detected, can effectively save power in trail settings. However, there's a chance of missing an event if the motion detection isn't triggered promptly. If the power source is not an issue, IR live video cameras can be used to enhance safety. In trail environments, high-resolution imaging can be beneficial for identifying, localizing, and counting wildlife. Camera placement aims to capture wildlife along the trail path.
    - Urban areas: IR live video or PIR cameras can be used depending on the specific needs. In urban areas, while low-resolution cameras may be acceptable for areas with limited distances between the camera and potential wildlife, it is generally preferred to use high-resolution cameras to ensure accurate identification, localization, and counting of wildlife. Camera placement focuses on capturing wildlife activities within the property.

3. **Weather Considerations**: Different environments are subject to a variety of weather conditions that can impact animal detection systems and camera performance. Therefore, it is crucial to consider these factors when deploying wildlife detection systems.
    - Highways: Highways are exposed to a range of weather conditions, such as rain, fog, snow, and high temperatures. These conditions reduce visibility and can affect the performance of the animal detection model. For instance, rain or snow can obstruct the view and fog can reduce the detection range. Hence, it is essential to train the detection model on these conditions and ensure that the mitigation systems are equipped with protective housings to maintain consistent performance.
    - Trails: In wilderness trails, in addition to the weather conditions faced on highways, there are other challenges. One such challenge is the high density of trees and bushes which, when moved by the wind, can generate many false alarms. Therefore, animal detection model in these areas should be designed to handle these challenges and maintain accuracy.
    - Urban areas: Weather conditions can be more controlled in urban areas since they are often partially shielded by buildings. However, cameras still need to withstand weather conditions. Factors such as rain, snow, and temperature changes are still relevant considerations for the animal detection models deployed in urban areas.
4. **Object Detection Model:** The detection model is a core component of the mitigation systems. It is trained to identify, localize, and count objects in the images or videos captured by the camera. The training focus depends on the deployment location: for highways, the model is specialized to detect and count a single class-either an animal or not, whereas for trails or urban areas, it is trained on multiple selected/predefined species that are commonly encountered. In our research, we conducted experiments to evaluate the model's accuracy, processing speed, and power consumption, taking into account various factors such as lighting conditions, weather conditions, and the presence of vegetation or other natural elements that may partially obscure animals. The accuracy of the model refers to its ability to correctly identify, localize, and count objects, while the processing speed represents the rate at which the model can analyze and process the captured data. Additionally, power consumption is a critical consideration as it influences the energy efficiency and sustainability of the system. The selection of animal species in

our research was motivated by the prevalence of encounters between these animals and vehicles, which often lead to severe crashes on highways. Also, these animals are sometimes involved in tragic direct encounters with humans, as previously mentioned. Accurate and fast identification, localization, and counting of wildlife in each environment are crucial for effective conflict management.

5. **Processing Hardware**: The mitigation systems require hardware to process and analyze the images or videos captured by the camera. This hardware is typically an embedded device capable of handling the computational needs of the detection model.
6. **Cloud Storage**: Cloud-based platforms, such as Firebase Real-time Database and AWS, are used to store processed data. This data can be accessed from anywhere, facilitating prompt and efficient alerting. While this may be feasible for urban areas, it might pose challenges on highways and trails due to connectivity coverage.
7. **Warning Mechanisms and Animal Deterrence:** The warning systems employed in each environment should align with the demands and characteristics of the setting, considering the speed of detection, alert dissemination, and their effectiveness in preventing or mitigating collisions and conflicts. They vary from visual signals, such as electronic roadside signs, to cellphone warning application and ultrasound or sound devices specifically designed for animal deterrence.
    - Highways: In highway settings, prompt alert dissemination are critical due to the high speeds at which vehicles travel. In such settings, electronic roadside signs or cellphone warning application that warn drivers about the presence of wildlife on or near the road can be effective. The cellphone warning application will be installed on the cellphone and should always be on in the background to provide continuous audio alerts. Additionally, ultrasound or sound devices can be used to scare away animals from the road, thereby mitigating potential conflicts.
    - Trails: On wilderness trails, warning systems can employ a cellphone warning application integrated with Google Maps to indicate the location of detected animals. However, recognizing that not all trails have cellular coverage, alternative alerting methods can be employed. These methods can include dispatching a radio signal from a transmitter to a pre-specified frequency or placing digital trail signs in areas where hikers frequently pass by. The focus here is on alerting hikers about

potential wildlife encounters, allowing for direction change and reasonable reaction time due to the lower speeds of travel. It is preferred not to scare animals to avoid causing undue stress to them.

- Urban areas: For urban area settings, alerts could take the form of cellphone warning application notifications sent to homeowners, accompanied by images of detected animals. The effectiveness of these alerts would depend on how quickly the homeowners react to the notifications and their ability to scare the animal away or secure their property. In some cases, movement-triggered sprinklers, ultrasound device, sound device, or flashing lights could be activated to frighten away the animals, preventing potential conflicts.

Table 1.2: Comparative study of managing real-time WVCs and WHCs mitigation systems.

Specification	**WVCs Mitigation System for Highways**	**WHCs Mitigation System for Trails**	**WHCs Mitigation System for Urban Areas**
Camera			
Type	IR Live video camera	PIR Camera/IR Live video camera	PIR Camera/IR Live video camera
Output	Video	Images/Video	Images/Video
Resolution	High	High	Low/High
Field of View	Wide angle	Wide angle	Standard angle
Detection Model			
Classes	Single class: animal	Predefined animal species classes	Predefined animal species classes
Warning System			
Network	Cellphone application	Cellphone application with Google Maps	Cellphone application
Type of Alert	Audio alert from the cellphone application	Text with the location of animal	Image with the detected animal
No-Network	Digital roadside sign	Radio signal sent from radio transmitter to a pre-specified frequency or a digital trail sign	Sound signal
Animal Deterrence			
Type	Ultrasound or sound devices	Nothing	Movement-triggered sprinklers, ultrasound, or sound device

1.3 Challenge Statement

Dataset preparation plays a vital role in improving the performance of object detection tasks. To the best of our knowledge, while there are several publicly available labelled datasets of animal species from different countries such as Russia, South Africa, India, and Australia [47] [48] [49] [50], there is currently a lack of labelled datasets specifically focusing on North American large animals in various poses. The availability of accurately labelled datasets is crucial for training effective animal detection models. However, acquiring such datasets can be time-consuming and costly, posing a challenge in terms of data collection and annotation.

In our research, we encountered several challenges related to the identification and localization of animal species in images and the development of a reliable and feasible warning system for WVCs and WHCs mitigation systems. Among the complexities to consider are the need for animal species detector to:

- be deployed on embedded devices for the development of wildlife mitigation systems.
- detect objects during both day and night under various weather conditions, such as snow and fog, as shown in Fig. 1.3(a, b).
- handle scenarios involving multiple occluded objects, as shown in Fig. 1.3(c), as well as variations in distance (both far and close to the camera) and posture of animals from the camera, as shown in Fig. 1.3(d, e, f) respectively.

These concerns will be discussed in Chapter 4. The trade-off between accuracy, detection time, power consumption, and cost considerations for object detector is a major challenge, especially when considering the limitations of embedded devices.

Addressing these challenges in our research requires careful consideration and the development of innovative solutions to improve the effectiveness and real-time capabilities of animal detection for WVCs and WHCs mitigation systems.

(a) (b)

(c) (d)

(e) (f)

Figure 1.3: Examples of the challenging images: (a) day image of moose, (b) snowy and foggy night image of moose, (c) cluttered and occluded image of moose, (d) image of three bears far from the camera, (e) night image of moose closes to the camera, and (f) image of two mountain goats with different poses.

1.4 Dissertation Goals and Contributions

The dissertation goals and contributions can be summarized as follows:

1. Build a new labelled dataset of North American/Canadian large wildlife species in various environments to address the scarcity of such datasets.
2. Propose a new detection model, Deformable Mask R-CNN, intended for developing an automated labelling and annotation system in different format.
3. Develop a fast and accurate M-YOLO animal species detection model, designed for deployment on embedded devices.
4. Introduce the MCF algorithm to control the number of frames to be processed based on the motion activity within them. To the best of our knowledge, this represents its first utilization in object detection systems.
5. Propose an algorithm that allocates and distributes object detection model layers across pipeline stages to enhance pipelining and improve throughput.
6. Propose a novel hybrid approach, integrating pipelining and dataflow techniques.
7. Propose the MCFP-YOLO detection model which is designed for embedded devices and is intended to be employed in live camera based WVCs and WHCs mitigation systems.
8. Develop and implement real-time LCDS and VIADS applications to assist biologists, researchers, and conservationists in identifying, counting, and labelling wildlife. These applications contribute to monitor and analyze wildlife behavior for both real-time and batch processing.

1.5 Dissertation Organization

This dissertation is organized into eight chapters detailing various aspects of the research. Chapter 1 discusses the problem statement, motivation behind the research, and research goals. Chapter 2 presents the background information relevant to the research topic. It provides a summary of existing knowledge and research related to object detection, with a specific focus on animal species detection. Chapter 3 proposes new variants of region-based Convolutional Neural Networks (R-CNN) models with deformable convolutional layers. Chapter 4 presents the proposed animal species detection model to be deployed on embedded devices for real-time applications. Chapter 5 explores the integration of Motion-selective Control Frames (MCF) algorithm into the proposed

detector/model to improve its detection speed and power consumption. Chapter 6 investigates the use of parallel processing technique in the proposed model to reduce its processing delay. Chapter 7 evaluates the integration of the MCF algorithm and parallel processing technique into the proposed detector to enhance real-world object detection in embedded systems. Chapter 8 summarizes the research, highlighting its accomplishments and contributions and outlining potential future directions.

Chapter 2

Background and Related Work

In this chapter, background information on both traditional machine learning and deep learning for object detection task, as well as on the fundamentals of parallel processing to improve object detection speed is provided in Section 2.1. Then, some related work to our research is discussed in Section 2.2.

2.1 Background

2.1.1 Traditional Machine Learning and Deep Learning

Artificial Intelligence (AI) systems have the capability to learn patterns from the environment (training data) and make efficient and accurate decisions or predictions on new input data relevant to the same environment. These systems rely on machine learning (ML) algorithms [51] to solve complex problems in a way similar to humans, using three types of analytics: (i) descriptive analytics: it uses data aggregations to explain what happened (what we know or what has happened), such as image captioning [52], (ii) predictive analytics: it uses statistical models or algorithms to analyze the input data and predict future outcomes (what is likely to happen), such as object detection [53], (iii) prescriptive analytics: it uses the output of predictive analysis to recommend or suggest actions to take for any predefined outcomes (what should happen), such as medical image diagnosis [54] [55].

Currently, research in computer vision tasks is divided into two directions: traditional ML [56] [57] [58] and deep learning [59] [60] [61]. The choice of the right direction depends on several factors, such as: the size of the data used, the type of task that needs to be accomplished, the platform (computer hardware) used, and the desired accuracy. For example, it is found that deep learning provides poor performance with tens or hundreds of examples for each class [62] [63] [64]. However, traditional ML can adapt to a limited amount of data and provide better performance [64] [65] [66] [67].

The traditional ML algorithms have a long history since 1959, when Arthur Samuel defined the concept of ML [68]. Traditional ML models involve a process that takes data and tries to extract the significant information (handcrafted features) from it using feature descriptors such as Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), Histogram of Oriented Gradients (HOG), etc. [69]. These extracted features are fed into one of the following traditional ML algorithms [70]: Linear Classifier, Logistic Regression, Naïve Bayes (NB), Bayesian Network (BN), Support Vector Machines (SVM), Decision Tree, Random Forest (RF), AdaBoost, Bootstrapped Aggregation (Bagging), K-Nearest Neighbour (K-NN), and Artificial Neural Networks (ANNs). The choice of the suitable algorithm depends on the type of the problem being addressed, such as classification, regression, clustering, or detection.

The traditional ML models often struggle to achieve high accuracy in object detection tasks, as demonstrated by their performance on commonly used datasets such as ImageNet [71] and MS COCO [72], due to several reasons such as: (i) the difficulty of selecting or designing a robust feature descriptor that can handle challenges like cluttered backgrounds, illumination changes, and diverse object appearances, (ii) the difficulty of building and training traditional ML models, as they require the integration of multiple algorithms or the combination with deep learning techniques to address complex computer vision tasks like object detection, (iii) the difficulty of dealing with large datasets, as the limitations of traditional ML models become noted with the growth of data. Therefore, efforts have been made to overcome these limitations by using deep learning techniques which offer better feature extraction capabilities to improve the performance of object detection tasks [73] [74].

Deep learning or deep neural networks (DNNs) gained popularity in the mid-2000s [75], when they were found to be effective in reducing the dimensionality of data. DNNs are a type of ML based on ANNs that consist of multiple stacking connected hidden or intermediate layers, such as convolutional neural networks (CNNs) [76] [77], whereas traditional ANNs have only one layer referred to as shallow neural networks.

As shown in Fig. 2.1, each layer in DNNs structure contains multiple neurons (N). Each neuron performs mathematical processing unit by taking input values (X), multiplying them by corresponding weights (W), adding bias values (b), and applying an activation function (f) [77]

to the weighted sum of the inputs to be passed to the next layer's neurons as shown in the following equation:

$$y = f(\sum_{i=1}^{N} W_i X_i + b_i) \quad (2.1)$$

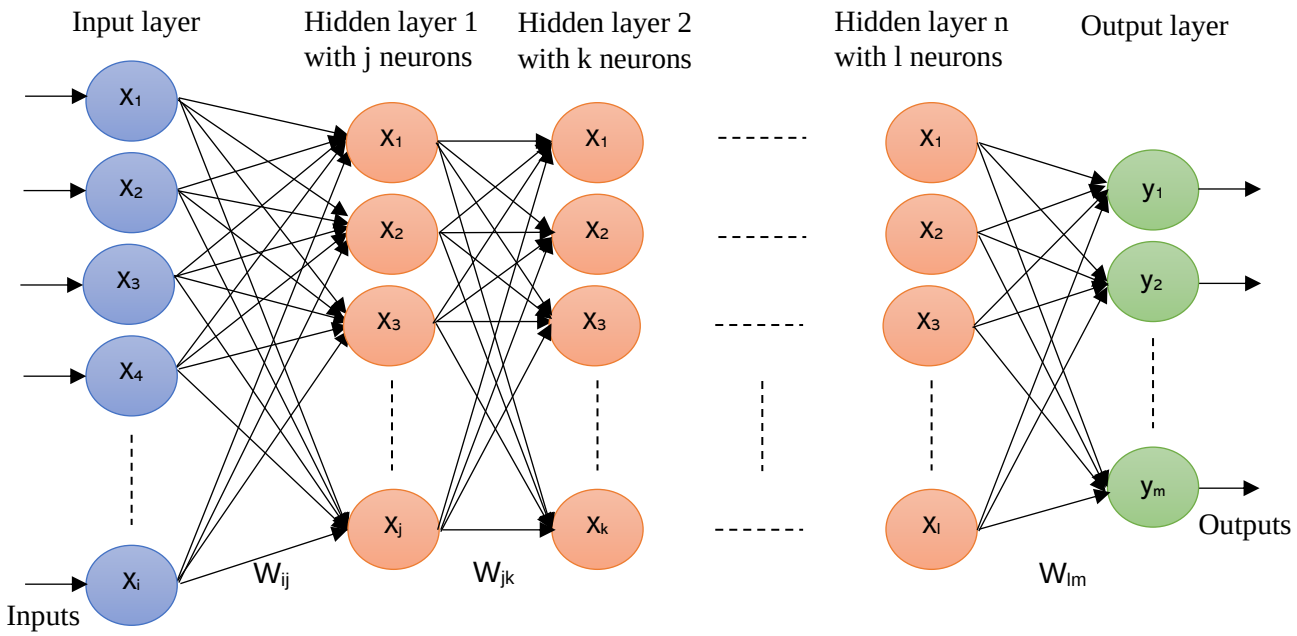


Figure 2.1: DNN structure which consists of one input layer, n hidden layers, and one output layer. Each layer contains multiple neurons or nodes (i, j, k, l, m).

DNNs have proven their capability to solve complex problems in real-time applications, such as object detection, semantic image segmentation [78], speech recognition [79], and many other tasks, surpassing the capabilities of traditional ML algorithms [80] [81]. This is because DNNs have a hierarchical structure, as shown in Fig. 2.1, which allows them to extract and learn low- and mid-level features in the initial layers and progressively build up more complex (high-level) features in the subsequent layers. Unlike traditional ML algorithms that rely on feature descriptors, DNNs can directly extract features at different levels, including low-level features such as edges, mid-level features such as corners and textures, and high-level features such as parts of objects [82]. The depth of the network, achieved by increasing the number of hidden layers, enhances the extracted features depending on the specific application or task [83]. The increased depth allows DNNs to achieve high accuracy in complex tasks. Several notable DNN architectures are

developed, including AlexNet [77], GoogLeNet [84], VGGNet [83], and ResNet [85]. These networks are commonly trained on large publicly available labelled datasets, such as ImageNet [71], and PASCAL Visual Object Classes (VOC) [86], which cover a wide range of object classes. These datasets do not contain all the classes, so for new applications or new datasets, the above pre-trained DNNs (backbone networks) can be reused as a starting point. Through transfer learning [87], the learned features are transferred from the pre-trained networks to fine-tune the network weights to adapt to the new task or dataset, saving significant training time and resources.

In the context of image classification, which involves identifying the contents of an image, CNNs exhibit different levels of accuracy performance. The number of computational layers used for learning features from input images varies depending on the specific visual task. The following subsections will provide an overview of regular CNNs and Deformable CNNs (D-CNN), highlighting their role and significance in the field of image classification.

CNN

CNN is widely recognized as the most popular deep learning algorithm, which was originally developed to analyze input images and videos for computer vision tasks such as image recognition and object detection [88]. However, the applicability of CNNs have been proven for almost any type of data, such as text analysis [89] and audio processing [90]. The name of CNN comes from the mathematical linear convolutional operation between matrices. The primary objective of CNNs is to extract the significant features from the input data, making it well-suited for image classification tasks [77]. As shown in Fig. 2.2, the structure of a CNN is essentially a sequence of layers which can be divided into two linked main parts: the feature learning part, and the classification part.

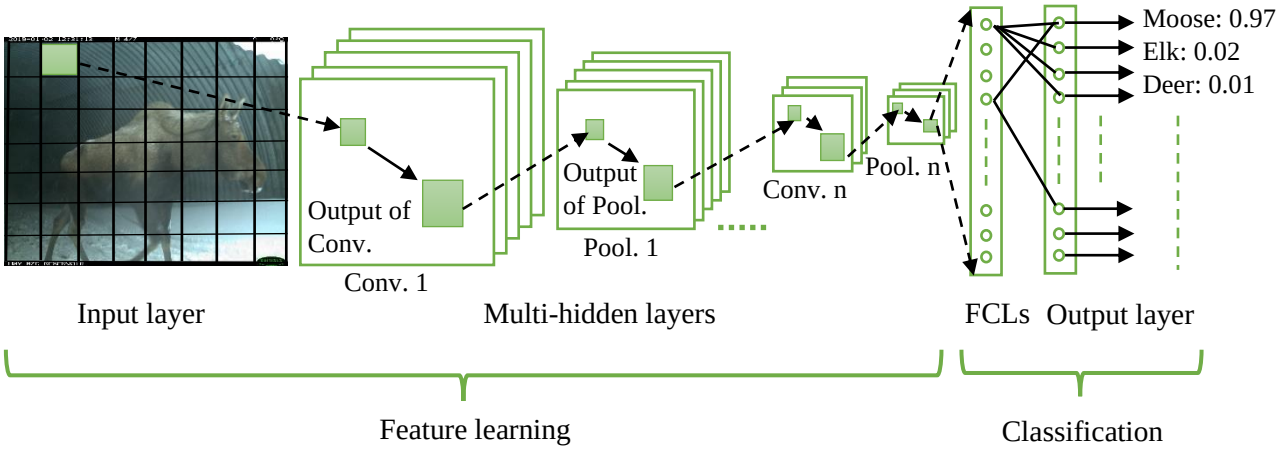


Figure 2.2: Illustration of an example of CNN architecture in animal classification. Convolution layers are denoted as Conv. and pooling layers are denoted as Pool. Multi-hidden layers consist of n hidden layers (Conv. n + Pool. n), depending on the input image and the visual task. The Fully Connected Layers (FCLs) flatten the output of the previous layers, which is called feature maps, and output them to the output layer with Softmax activation function to classify the object to different probabilities such as 0.97 for moose, 0.02 for elk and 0.01 for deer.

For feature learning, the input layer of a CNN converts input image into a feature map, which is a matrix representing the pixel intensities for the whole image (a matrix of extracted features). Then each layer in the multi-hidden layers, including the convolution layers and pooling layers, performs convolution and pooling operations on its input feature map. These operations help the network learn spatial hierarchies of features from low- to high-level patterns [91] [92].

For classification, the Fully Connected Layers (FCLs) [93] shown in Fig. 2.2 are the output layers which flatten the outputs of the previous layers, the feature maps, into a single vector, which serves as input for the Softmax layer [94]. Each input in the vector is connected to all neurons, represented as circles in Fig. 2.2, in order to predict the class of the object in the input image. The Softmax activation function is used to convert the output values into conditional probabilities, representing normalized classification scores for prediction. Each probability value ranges between 0 and 1, and the sum of all values is equal to one [77] [94]. The architecture of CNN is capable of learning and extracting object features and can handle multiple tasks simultaneously, such as object detection and segmentation [94-103].

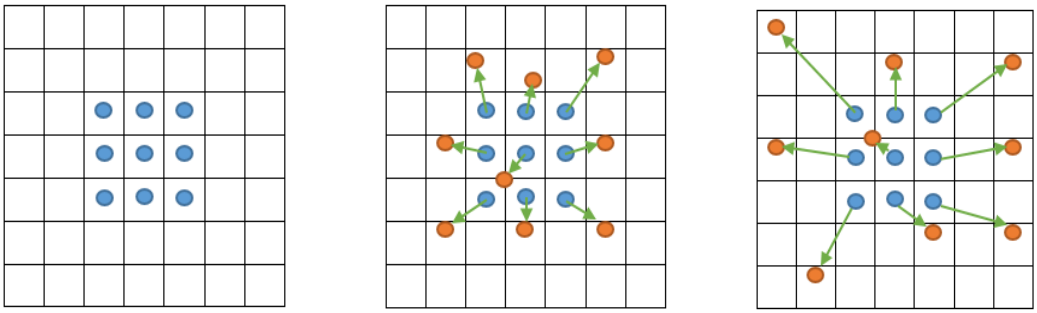
CNNs are built on fixed and known geometric structures, which limits their ability to handle geometric variations in objects, such as: pose, scale, viewpoint, and deformation parts [95], as illustrated in Fig. 2.3. To address this limitation, CNNs are typically trained on datasets that exhibit

sufficient variation, or on augmented data by changing the size, shape, and rotation angle of objects, to attain high detection accuracy. Although, these approaches have proven effective, the training process can be complex and therefore expensive. To enhance the capability of CNN to handle geometric variations and object deformations without relying on data-augmentation, D-CNNs were introduced [96] [97].

Figure 2.3: Examples of images that contain geometric variations in the object (moose) which make it difficult to be identified by using regular CNN.

D-CNN

The concept behind D-CNNs is to replace the fixed locations of the regular sampling matrix, (convolution kernel), represented by the 3 × 3 blue points in Fig. 2.4(a), with the deformable sampling matrix that has movable locations, represented by the orange points in Fig. 2.4(b, c). These orange points are redistributed to different locations based on the shape of the object using learned augmented offsets (the green arrows). The structure of the deformable sampling matrix can be obtained through a convolution algorithm that calculates the offset of the sampling position to learn the objects' geometrical properties [96] [97]. Each point in the regular sampling matrix is moved by adding the learnable offset to each of them, resulting in a deformable sampling matrix.



(a) (b) (c)

Figure 2.4: Illustration of the sampling locations in 3 × 3 regular and deformable sampling matrices. (a) Regular sampling matrix (blue points). (b) Deformable sampling matrix (orange points) with offsets (green arrows). (c) Example of how the positions of the deformable sampling matrix are changed from the original 3 × 3 squared positions according to the objects shape to identify deformed or occluded objects in the image.

As shown in Fig. 2.5, a D-CNN consists of: (i) a regular convolution layers with a fixed 3 × 3 sampling matrix, as in depicted in Fig. 2.5(a), to generate a feature map for the whole input image, shown in Fig. 2.5(b), and (ii) an additional convolution layer with a 3 × 3 regular sampling matrix, as presented in Fig. 2.5(c), for the learned augmented offsets, displayed in Fig. 2.5(d) to be learned from the feature map. These offsets are easily trained through end-to-end backpropagation, to generate a deformable 3 × 3 sampling matrix, as depicted in Fig. 2.5(e). Fig. 2.5(f) shows the output feature map obtained by applying the regular sampling matrix. The positions of the sampling points in the sampling matrix (represented by orange circles) are in a fixed 3 × 3 square shape. On the contrary, Fig. 2.5(g) shows the output feature map after applying the deformable sampling matrix. The positions of the sampling points are changed from the original 3 × 3 square shape to another shape according to the object's scale and shape, leading to a more precise bounding box. By comparing the two outputs in Fig. 2.5(f) and Fig. 2.5(g), it becomes evident that the deformable convolutional layer enhances the detection accuracy of the network at the cost of an amount of computations for the offset learning, as elaborated later in Chapter 3 and 4.

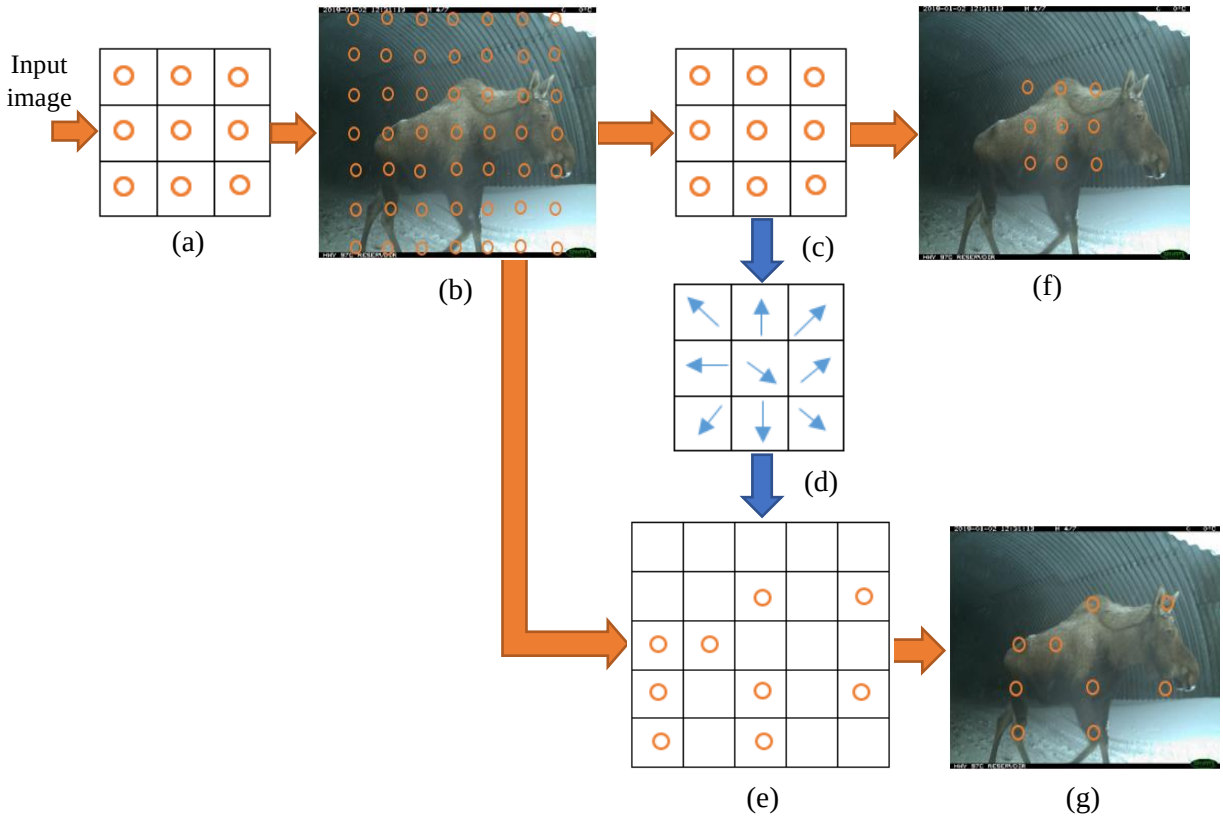


Figure 2.5: Illustration of the difference between applying 3 × 3 regular and deformable sampling matrices on input feature maps. (a) Regular sampling matrix (orange circles) of regular convolution layers. (b) Feature map of the whole image after convolution layer. (c) Regular sampling matrix. (d) Learned augmented offsets (blue arrows) from the preceding feature map via an additional convolutional layer to redistribute the sampling locations of the regular sampling matrix to focus on the objects. (e) Deformable sampling matrix after adding offsets to the regular sampling matrix. (f) Output feature map after applying regular convolution. (g) Output feature map after applying deformable convolution.

The advantages of D-CNN compared with the regular CNN can be summarized as:

1. D-CNN can be integrated into any CNN architectures, giving them the ability to deform its sampling matrix to accommodate the object's structure.
2. The offsets in D-CNN are dynamic model outputs which vary according to the object's location in the image.
3. D-CNN has the capability to learn adaptive receptive fields.
4. D-CNN learns sampling locations instead of filter weights, resulting in more precise bounding box generation.

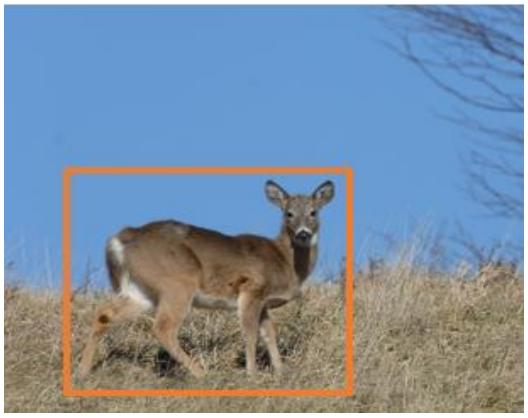
5. D-CNN provides improvements in the object detection accuracy [96] [97].

2.1.2 Object Detection

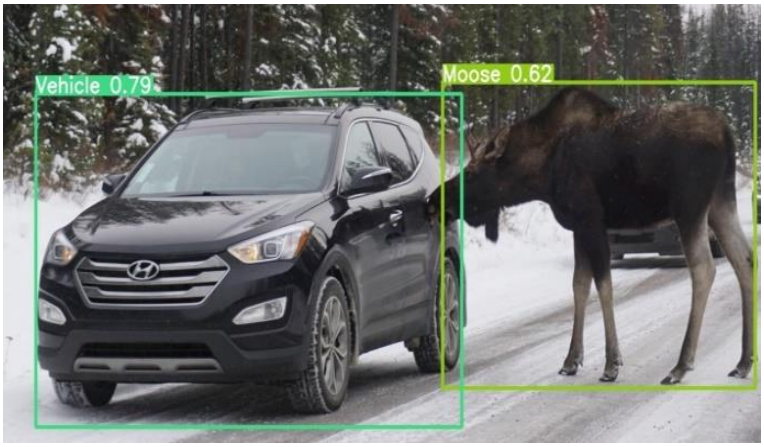
Object detection, also known as generic object detection, has been widely studied to identify objects within a digital image from a predefined set of object classes (object identification), as shown in Fig. 2.6(a). It aims not only determine the presence of objects but also localize them in the image using bounding boxes (object localization) [98], as shown in Fig. 2.6(b). In summary, object detection techniques produce bounding boxes with four sets of coordinates around each object within the image, along with the corresponding probability that each box belongs to a specific class, as shown in Fig. 2.6(c).

Elk

(a)



(b)



(c)

Figure 2.6: Computer vision tasks. (a) Object identification technique. (b) Object localization technique. (c) Object detection technique (generic).

Object detection is a basic step for image/video understanding and other computer vision tasks, such as object tracking [61] [99], counting [100] [101], instance segmentation [102] [103] [104], etc. Therefore, object detection techniques are considered as learning systems [105] that output valuable needed information for numerous applications in different aspects of our lives. From an application perspective, object detection can be classified into two main types: salient object detection [106] and generic object detection [107]. Salient object detection, also known as salient object segmentation, aims to identify and segment the most noticeable/salient objects [108] from their surrounding areas or less attractive background in an image, as shown in Fig. 2.7. This is achieved through techniques such as local contrast enhancement [109] and pixel-based segmentation [110]. Salient object detection has been used for many applications such as image segmentation [111] [112], image cropping [113], image retrieval [114], etc. On the other hand, generic object detection or object class detection as previously defined, aims to detect instances of different predefined classes and provide their spatial location and extent using bounding boxes, as shown before in Fig. 2.6(c). This is accomplished using bounding box regression technique [115]. The most outstanding applications of generic object detection are surveillance [116], autonomous driving [117], vehicle [118] and pedestrian detection [119] [120], medical diagnosis [121] [122], robotic vision [123] and so on. In our work, the primary focus will be on generic object detection.

(a) (b)

Figure 2.7: Example of salient object detection technique. (a) Input image. (b) Output image of salient object detection technique.

The standard pipeline of object detection techniques based on deep learning typically consists of three stages, as shown in Fig. 2.8: (i) regions of interest (RoIs) selection, also known as region proposals, (ii) feature extraction for each region proposal, and (iii) a detection network that includes region identification and object localization.

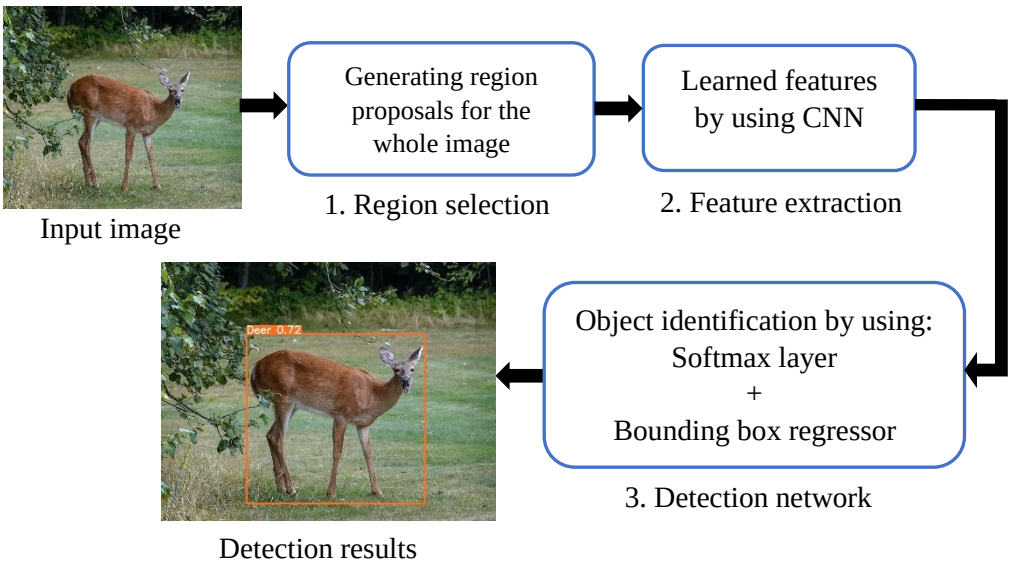


Figure 2.8: General architecture pipeline of object detection techniques based on deep learning approaches.

The RoIs selection stage is used to find out and extract all potential object locations in the input image. In traditional techniques, the entire image needs to be scanned by sliding multiple windows at various scales to locate objects [98], as different objects with varying sizes and scales may appear at different locations in the image. Although, this strategy can detect most possible object locations of objects by applying the classifier at every location and scale in the image, it produces many redundant windows, increases computational complexity, and consumes time. Therefore, deep learning techniques adopt an alternative strategy by using a region proposal algorithm. This algorithm generates different candidate regions within the input image that are highly likely to contain objects of interest [124] [125]. By utilizing this algorithm, detection time is reduced, computational costs are decreased, and detection accuracy is improved by reducing false positives [126] [127].

The most important stage in the object detection task is the extraction of significant features to accurately identify and localize objects within the image. There is a variety of features that can be used, ranging from handcrafted features which can be extracted by using SIFT, HOG, or Haar feature descriptors to learned or deep features extracted by CNNs. Since CNNs have the ability to extract complex and detailed features that can represent the image in more detail and learn invariant features, deep features have become popular over handcrafted features for most object detection applications [128] [129], as shown in Fig. 2.9 [130]. Theses features are fed into the

detection network to identify objects in each proposal (region identification) and to localize these objects by combining overlapped region proposals into a single bounding box around each detected object using bounding box regression [131].

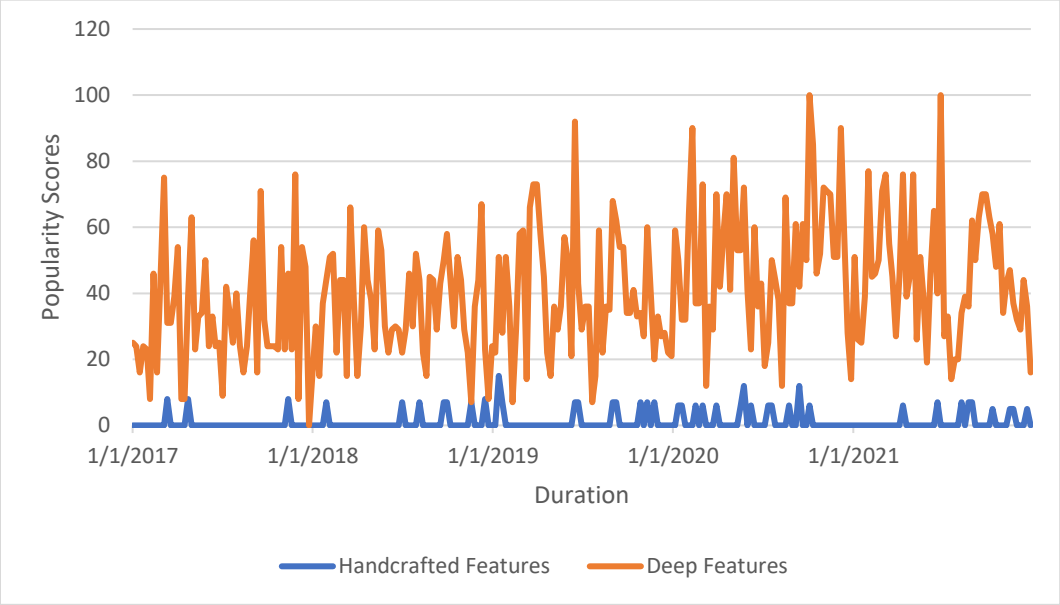


Figure 2.9: The worldwide popularity scores of handcrafted and deep features over the last five years based on the data collected from Google Trends [130].

The object detection techniques based on deep learning are divided into two categories: (i) two-stage detectors, and (ii) one-stage detectors [132]. Two-stage detectors generate region proposals for the whole image using various algorithms, and then use CNNs to identify objects within each region. On the other hand, one-stage detectors directly treat object detection as a regression problem, predicting bounding box coordinates and class probabilities without generating region proposals. Thus, finding the best object detector that achieves the enhanced accuracy and speed is a non-trivial task.

2.1.2.1 Two-stage Detectors

Object detection algorithms based on region proposals were introduced and developed as part of the PASCAL VOC challenge [133]. One notable algorithm that emerged from this challenge was the Region-based CNN (R-CNN) detector, proposed by Girshick et al. [134]. R-CNN merged region proposals with CNNs to achieve accurate object detection. As a result of the success of the region proposal method, the Fast R-CNN [135] was proposed to address the computational

complexity of the R-CNN. Fast R-CNN improved both the speed and accuracy of object detection. Further advancements in object detection came with the introduction of Faster R-CNN by Ren et al. [136]. Faster R-CNN combined the Region Proposal Network (RPN) and Fast R-CNN into a single network "Faster R-CNN" to accomplish further speed-up and higher object detection accuracy. Another notable extension of Faster R-CNN is Mask R-CNN [104], which introduced the concept of instance segmentation. In addition to detecting bounding boxes, Mask R-CNN computes object masks in parallel, allowing for more precise instance segmentation. All these improvements in object detection are significant and can be applied to animal species detection.

R-CNN

The R-CNN architecture is divided into five stages, as shown in Fig. 2.10. It starts by using a selective search algorithm to generate hundreds to thousands of region proposals for an input image. These region proposals are cropped and resized using region pooling layer [98] [137]. Then, each resized region proposal is fed into CNN to extract object features. The output of each CNN is the input of a linear SVM to identify the regions of objects in image [138]. Finally, these identified regions are adjusted by using the linear bounding box regressor, to tighten and to refine the final bounding boxes of the detected objects [131].

Selective search algorithm [137] generates regions based on a segmentation approach. It combines both object search and segmentation to detect all the possible locations of objects. In terms of segmentation of object and non-object, the image structures including object size, color similarities, and texture similarities, are used to obtain many segmented areas. Then, a bottom-up approach is typically used as part of the selective search algorithm to merge all the similar areas to get more accurate and larger segmented areas to produce the final candidate region proposals.

The R-CNN model cannot be applied to real-time applications because:

- Network processing is expensive and slow due to the use of selective search algorithm, where hundreds to thousands of region proposals need to be classified for each image.
- R-CNN sometimes generates bad candidate region proposals as the selective search is a fixed algorithm which has no learning capabilities.

At the same time, the training of the R-CNN model is complex and requires a big memory space, since R-CNN has to train three different models separately: CNN, SVM, and bounding box regressor.

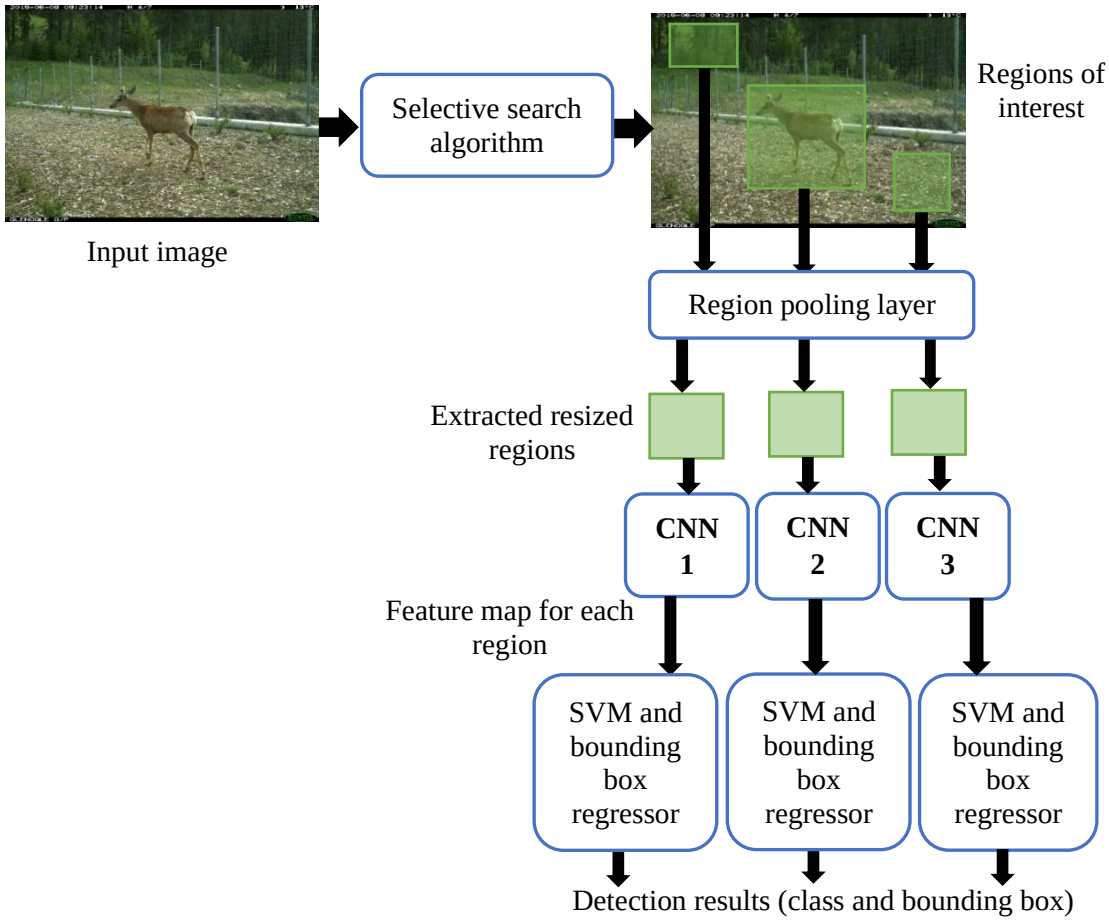


Figure 2.10: Basic architecture of R-CNN model. The number of SVM classifiers varies depending on the number of classes.

Fast R-CNN

The same developer who introduced R-CNN also proposed a modified model called Fast R-CNN [135] to address some of limitations of R-CNN. In Fast R-CNN, as shown in Fig. 2.11, CNN is used to extract features and produce feature maps for the entire input image instead of processing each region proposal individually, as done in R-CNN. Thereby, Fast R-CNN can save time and memory compared to R-CNN. From the feature maps of the whole image and the RoIs identified by the selective search algorithm, regions are cropped out and resized to a fixed size feature map

for each region proposal by using the region pooling layer. Then, these feature maps for each region are flattened into vectors through FCLs and fed into Softmax classification and bounding box regressor to predict the class and bounding box locations for each object in the image.

Despite the advantages of Fast R-CNN in terms of reduced memory usage, processing time, and improved detection accuracy, the selective search algorithm that generates region proposals still poses a bottleneck in terms of processing time for the model.

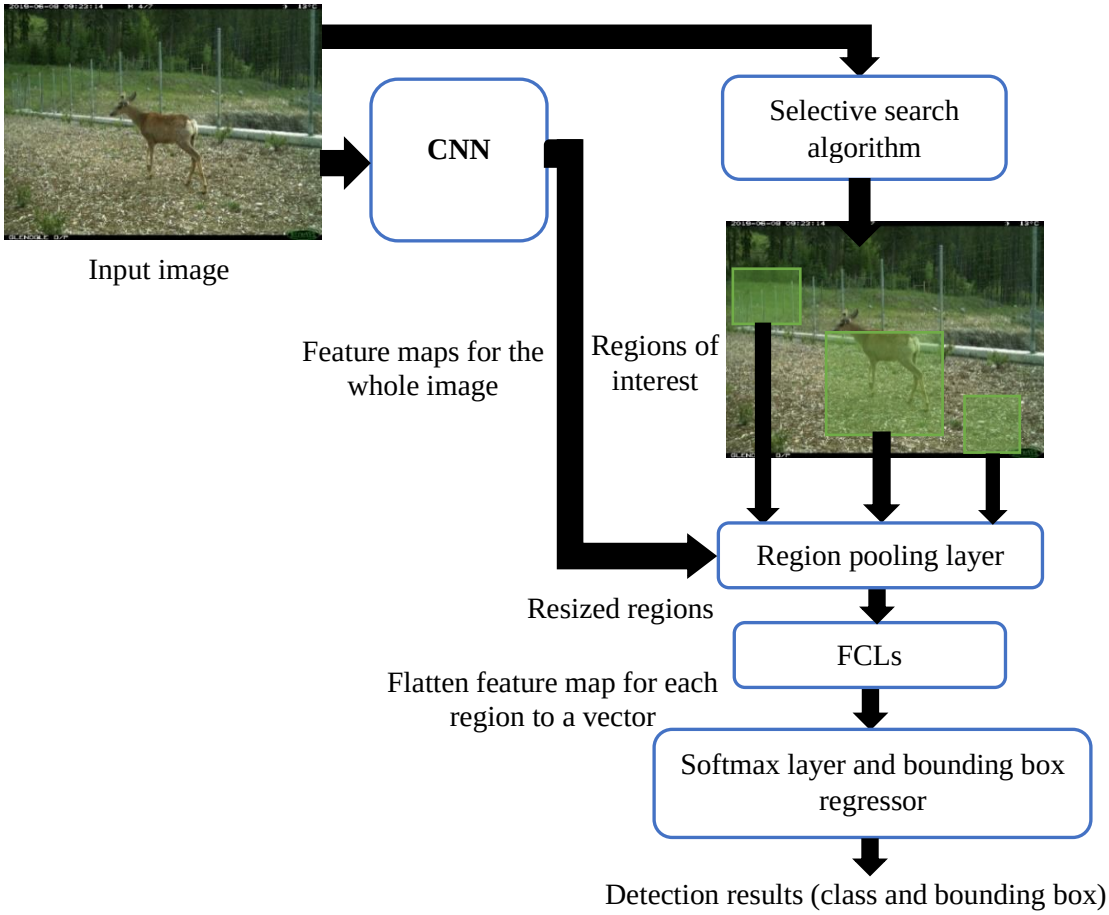


Figure 2.11: Basic architecture of Fast R-CNN Model.

Faster R-CNN

In this improved model, the selective search algorithm used in Fast R-CNN has been replaced by RPN. As shown in Fig. 2.12, RPN is more efficient in generating region proposals compared to the selective search algorithm. This efficiency is due to RPN sharing most computations with Fast R-CNN, as both networks have the same convolution layers and feature maps.

As shown in Fig. 2.13, RPN is used to generate a set of various size anchor boxes across the image [136]. Anchor boxes are predefined bounding boxes with different sizes and aspect ratios, which have been selected based on object size and are used as a reference in the testing process for the prediction of object class and localization. These anchor boxes are then passed through a binary classifier to determine the probability of containing object or not, and a regressor to create the bounding boxes of these proposals. After that, a Non-Maximum Suppression (NMS) filter [139] is used to remove overlapping anchor boxes, by (i) selecting the anchor box that has the highest confidence score, (ii) computing the overlap between this anchor box and other anchor boxes by calculating the intersection over union (IoU), (iii) removing anchor boxes that have higher overlap than a predefined overlap threshold, and (iv) repeating steps (ii), and (iii) until all overlapping anchor boxes are removed.

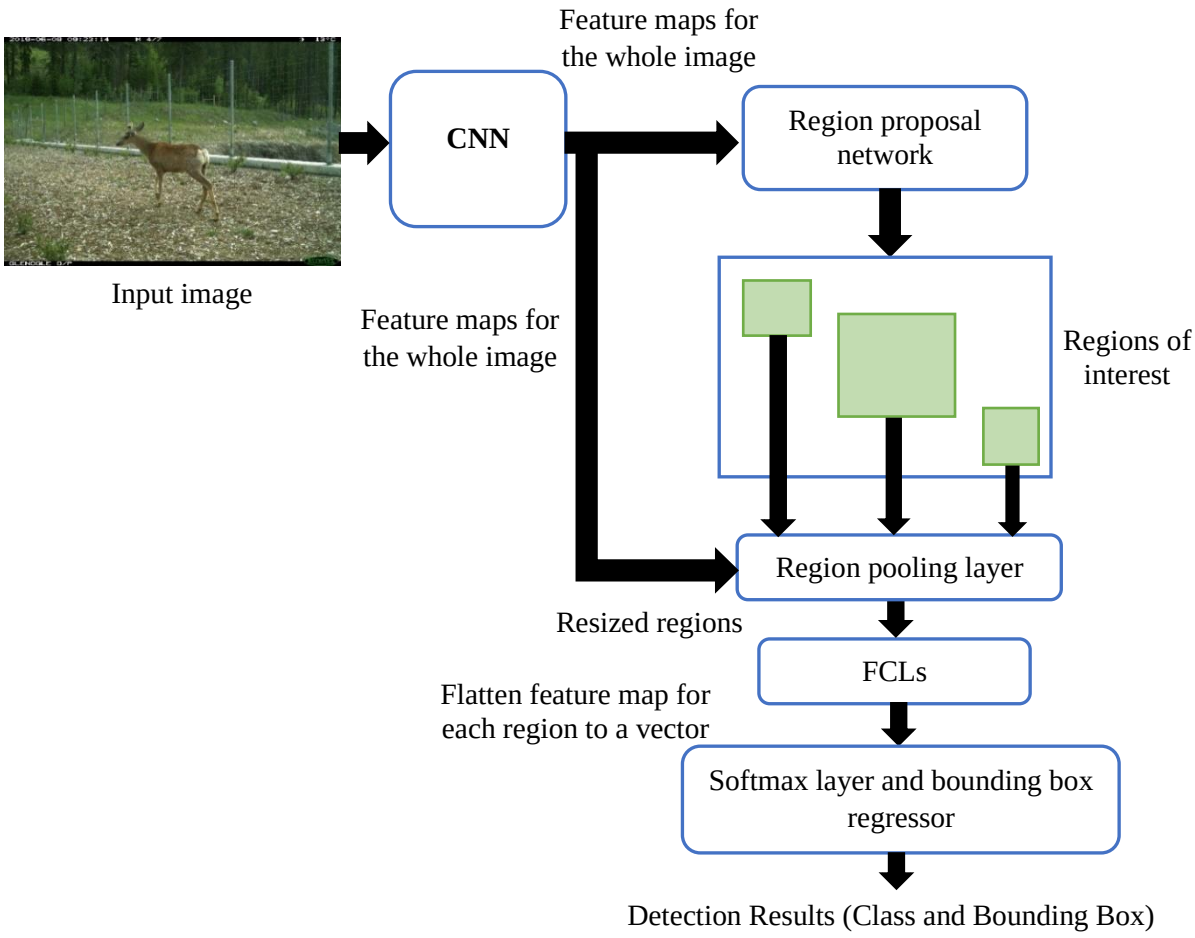


Figure 2.12: Basic architecture of Faster R-CNN Model.

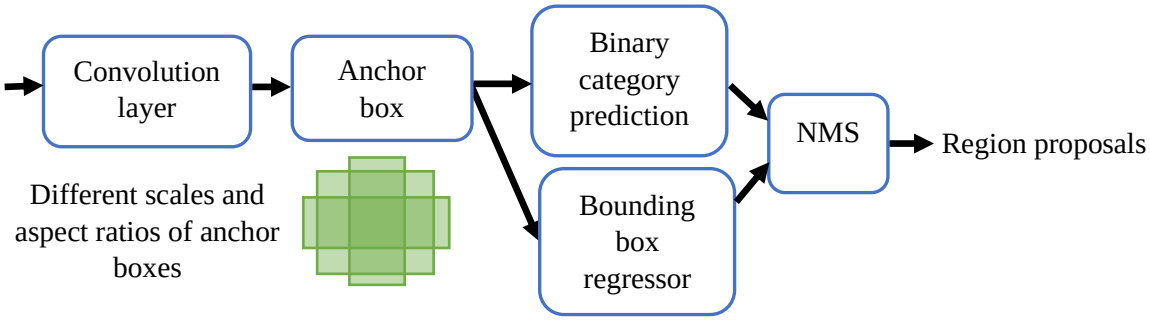


Figure 2.13: Region Proposal Network.

Mask R-CNN

Mask R-CNN is an extension of Faster R-CNN especially used for instance segmentation, which involves identifying and labelling each pixel in an image with its corresponding object class [104]. Two types of segmentations have been applied on the image in Fig. 2.14(a). Semantic segmentation, as shown in Fig. 2.14(b), assigns a single bounding box to objects of the same class without distinguishing between individual instances (e.g., there is one bounding box for the two bears). On the other hand, instance segmentation using Mask R-CNN, as shown in Fig. 2.14(c), provided a more detailed analysis by segmenting and distinguishing between objects of the same class individually in an image and localizing each object instance with a separate bounding box (e.g., there is a bounding box for each bear).

(a) (b) (c)

Figure 2.14: Image segmentation techniques. (a) Original image of two bears. (b) Semantic segmentation. (c) Instance segmentation using Mask R-CNN.

As shown in Fig. 2.15, Mask R-CNN consists of two parts: (i) Faster R-CNN for object detection, and (ii) Fully Convolutional Network (FCN) for providing segmentation masks on each object (object mask) [140]. In Faster R-CNN, the regions which have been resized by RoI pooling layer are slightly misaligned from the original input image. It is not important in bounding boxes; however, it has a negative effect on instance segmentation. So, Mask R-CNN uses the RoI Align layer to overcome this problem and to align more precisely by removing any quantization operations.

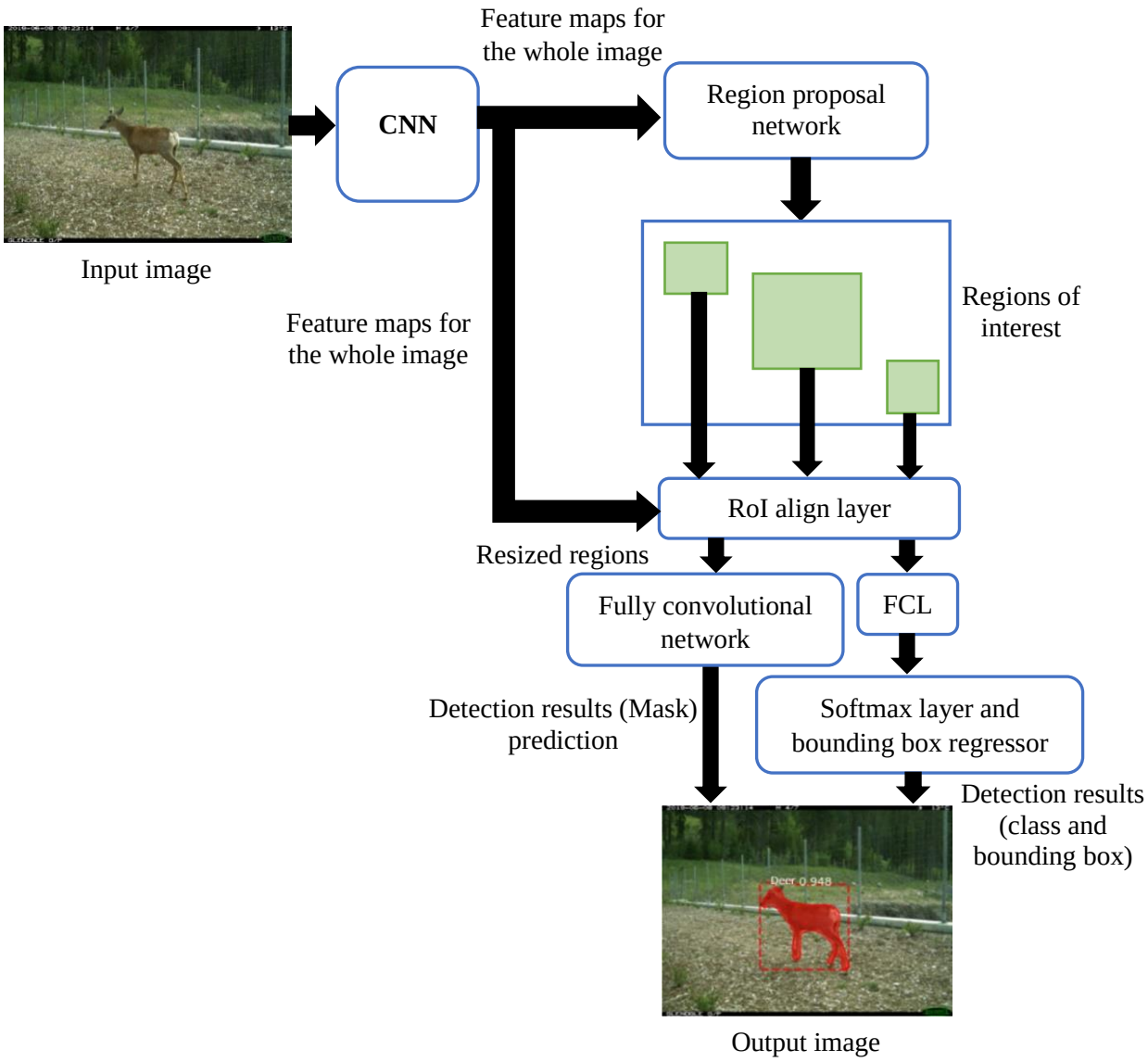


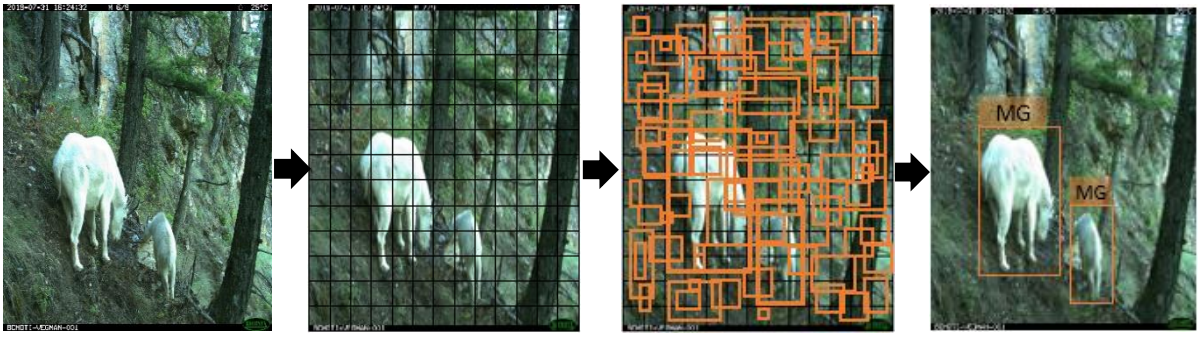
Figure 2.15: Basic architecture of Mask R-CNN Model.

2.1.2.2 One-stage Detectors

Two-stage object detectors go through the image several times to identify and localize the objects in the image. This process is slow as each step is trained separately. In 2016, Redmon et al. [141] addressed this issue by dealing with the object detection task as a regression problem directly from the image to bounding box coordinates and class probabilities instead of a classification problem. They proposed You Only Look Once (YOLO) in an image as a one-stage detector to identify and localize objects. YOLO is significantly faster than two-stage detectors, but less accurate compared

to Faster R-CNN due to the static nature of anchor boxes [142]. Later, Redmon et al. [143] proposed an improved version called YOLOv2, which utilized a new network called DarkNet-19 to improve the accuracy and speed of object detection. However, YOLOv2 struggled with detecting small objects, as discussed later in Chapter 4. In 2018, YOLOv3 was introduced as the last version of the YOLO model by Redmon et al. [144]. DarkNet-53 [144] was used as a backbone network to extract features in YOLOv3, instead of DarkNet-19 [145] as in YOLOv2, to improve the detection accuracy of small objects. This makes YOLOv3 slower than YOLOv2 because the computational complexity and the number of model parameters increase. Moreover, YOLOv3's ability to detect small objects is still limited because it does not effectively utilize low-level features. All these restrictions limited the use of YOLOv3 in industrial applications [146] [147] [148]. YOLOv4 was proposed by Bochkovskiy et al. in 2020 [149], with the goal of enhancing both the accuracy and the detection speed. The architecture of YOLOv4 uses Cross-Stage Partial DarkNet-53 (CSPDarkNet-53) as its backbone, which is a combination of DarkNet-53 and CSP network [149]. However, deploying YOLOv4 on embedded systems may still pose challenges, due to the device's limited computational capabilities.

The main steps of YOLO are shown in Fig. 2.16. In the initial step, the input image in Fig. 2.16(a) is divided into S × S grid cells, as in Fig. 2.16(b), where S is an integer. As shown in Fig. 2.16(c), each cell predicts: (i) a fixed number of bounding boxes with different aspect ratios and scales (orange boxes), to cover a wide range of object shapes and sizes, each bounding box has four coordinates and one confidence score, which measures the probability of an object in the bounding box, and (ii) object class probabilities, these probabilities indicate the likelihood of the detected object belonging to different predefined classes. Once these predictions are made for all grid cells, the subsequent step involves filtering out bounding boxes with low confidence scores, as shown in Fig. 2.16(d). This is achieved by using the Non-Maximum Suppression (NMS) algorithm [141], which aims to eliminate redundant and overlapping bounding boxes while retaining the most confident and accurate ones.



(a) (b) (c) (d)

Figure 2.16: The framework of YOLO. (a) Input image of two animals (Mountain Goat: MG). (b) Divide up input image with S × S grid cells. (c) Each cell predicts bounding boxes and confidence probabilities: P(object). If there is an object in the grid cell then P(object) =1, otherwise P(object) =0. Also, each cell predicts class conditional probability: P(class: MG/object)). (d) Output image with the detected animal species (MG).

2.1.3 Parallel Processing

Parallel processing is a method of computing where multiple computations or tasks are executed simultaneously to increase processing speed. It involves breaking down a problem/task into M sub-tasks ($M \geq 2$) and solving them simultaneously using P processors/cores. In order to implement parallel processing in our work, it is essential to determine whether the object detection task can be broken down in this way or not.

Parallel Processing Techniques/Architectures

Parallelism can be implemented in a computing system by using various techniques or architecture, such as [150] [151]:

1. Multiprocessing or multicore processing: It is a technique which allows multiple processors or cores to run simultaneously within a CPU, enabling parallel execution of multiple tasks simultaneously. As shown in Fig. 2.17, each processor/core runs independently and has its own local memory space known as individual or cache memory for temporary data storage during processing. For data exchange and communication between processors/cores, a shared memory is provided. The cache memory is smaller and faster compared to the shared memory. Additionally, there is the main memory, which is directly connected to the CPU and is responsible for storing the program code and data that can be accessed by all cores.

2. SIMD (Single Instruction Multiple Data) stream: It is a technique which allows multiple processors to execute the same instructions/programs on multiple data sets in parallel, as shown in Fig. 2.18. To improve the performance of SIMD systems, instruction cache memory is used. Instruction cache is a type of cache memory that stores frequently used instructions within the processor, reducing the time spent waiting for instructions to be fetched from the main memory.
3. MIMD (Multiple Instruction Multiple Data) stream: It is a technique which allows multiple processors or cores to execute their own individual instructions/programs on their own data and resources, including instruction cache, in parallel, as shown in Fig. 2.19.

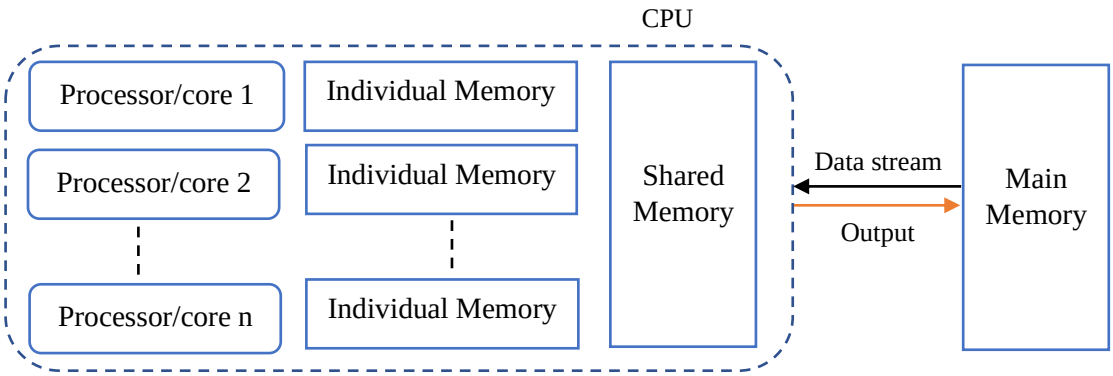


Figure 2.17: Multicore processing architecture.

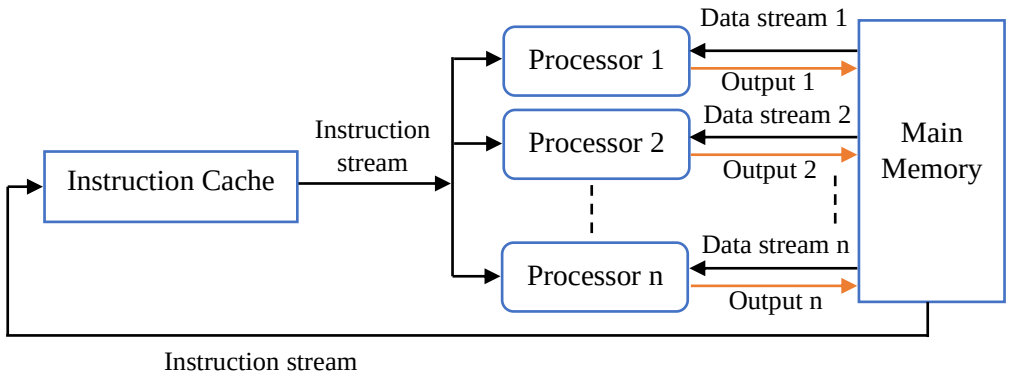


Figure 2.18: SIMD stream architecture.

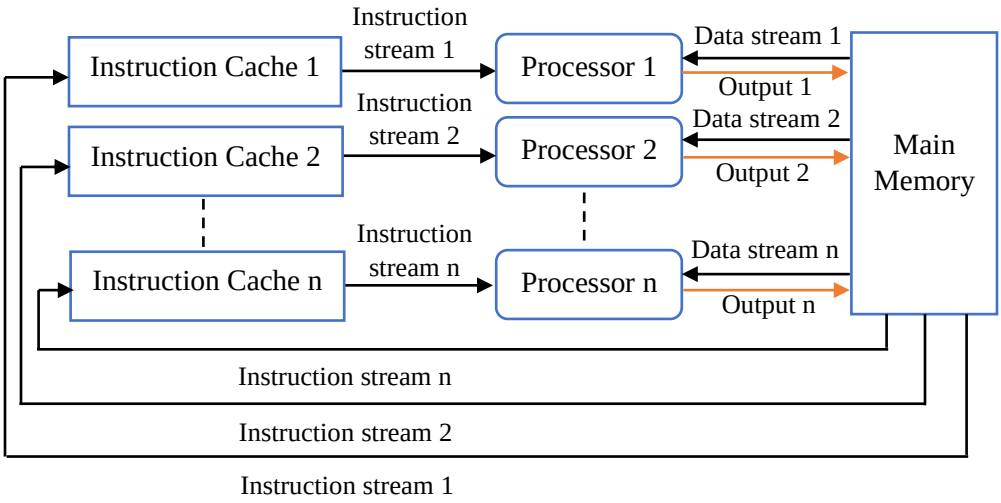


Figure 2.19: MIMD stream architecture.

Each of these techniques or architectures has its advantages and limitations. The choice of which one to use depends on the problem's requirements that needs to be solved, and the available hardware resources. For example, multiprocessing or multicore processing is appropriate for problems requiring coordination between processors or tasks, as it distribute the workload across multiple cores and allows them to work together. However, the scalability (ability to expand data and resources) of multicore processing may be limited by the number of available cores and memory bandwidth. While SIMD is suitable for problems that involve processing large amounts of data in parallel within a single instruction. However, utilizing SIMD requires specialized hardware. In addition, MIMD works well for problems where each processor/core operates on its own data and executes its own instructions, without any shared memory or coordination between processors.

2.2 Related Work

2.2.1 Animal Species Detection

Recent studies have attempted to identify animal species in camera-trap images like [152] [153] [154] [155] [156] [157] [158] [159]. However, only few studies have focused on animal species detection to identify and localize them [100] [160] [161] [162]. Some of them are specialized to detect single animal species, such as cattle [100] [163], zebras [160], and pigs [161].

Yu et al. [164] manually cropped and selected images, that only contained the entire animal body. They used a dataset which consists of over 7,000 Infrared (IR) images captured by motion

detection camera, called camera-trap, from two different field sites. This technique of cropping images allowed them to obtain 82% accuracy by using linear SVM to classify 18 animal species. Chen et al. [165] used 6-layers CNN to classify 20 animal species in their own dataset of 23,876 images with an accuracy of 38.32%. The authors used a segmentation algorithm for cropping the animals from the images and used these cropped images to train and test their system. Gomez et al. [166] used deep CNNs to identify animal species in the Snapshot Serengeti dataset. They reached 88.9% of accuracy in Top-1 (the highest probability prediction matches the actual class) and 98.1% in Top-5 (one of the five highest probability prediction matches the actual class). Willi et al. [153] identified animal species by using CNNs. They achieved an accuracy of 92.5% in Snapshot Serengeti dataset, and an accuracy of 91.4% in Snapshot Wisconsin dataset. Norouzzadeh et al. [154] used a human labelling process to train a deep active learning system to classify and count animals in to reduce images. Their system achieved an accuracy of 92.9% on cropped animal images from the Snapshot Serengeti dataset, by using ResNet-50 as a backbone network for their model. Furthermore, Norouzzadeh et al. [152] used CNN and reported an accuracy of 93.8% in classifying images that contain only a single animal in the Snapshot Serengeti dataset. The performance matched human accuracy in their experiments. However, though this work showed promising results for classifying images with only a single animal, it could not handle the challenge of localizing several animals.

Parham et al. [160] used YOLO detector to detect zebras from a dataset of 2,500 images and created bounding boxes of Plains Zebras with an accuracy of 55.6% and Grevy's Zebras with an accuracy of 56.6%. Zhang et al. [167] created a dataset of 23 different species in both daytime color and nighttime grayscale formats using 800 camera-traps. They compared Fast R-CNN and Faster R-CNN with their proposed method, the spatiotemporal object proposal and patch verification framework, which achieved an average F-measure of 82.1% in animal species detection. Xu et al. [100] evaluated the Mask R-CNN model for the detection and counting of cattle (single class) from quadcopter imagery. The authors achieved accuracy of 94%. Gupta et al. [168] used the Mask R-CNN model with a pre-trained network, ResNet-101, to detect two animal species (cows and dogs). They achieved an average precision of 79.47% and 81.09%, for detecting cows and dogs, respectively. Schneider et al. [142] compared the accuracy of Faster R-CNN and YOLOv2 models in detecting animal species within camera trap images. Their results showed that Faster R-CNN outperformed YOLOv2 by 33.4%.

2.2.2 Animal Detection Systems

Current research on animal detection systems can be divided into two directions: using traditional ML algorithms, which rely on feature extraction descriptors to detect animals [56] [57] [58], and using deep learning algorithms, which rely on CNNs [59] [60] [61].

Parikh et al. [169] proposed an animal detection system with an audio and visual alarm signals to help reduce WVCs using a template matching algorithm [170]. The authors stated that their system produces many false predictions under various lighting conditions, particularly at night. Mammeri et al. [171] proposed a moose detection system, which used a roadside camera, to warn drivers. A Two-stage strategy was used: first, the Local Binary Pattern Adaptive boost (LBP-Adaboost) algorithm [172] was applied to generate RoIs, and second, each generated RoI was processed by an adapted version of the HOG-SVM detector. Their system achieved an accuracy of more than 83% for moose detection with an inference time of 52.8 ms. Sharma et al. [173] used the HOG descriptor and boosted cascade classifier in an animal detection system on highways to alert drivers. They focused only on cows and dogs. The accuracy of their proposed system was almost 82.5% with an average detection time of 100 ms. Matuska et al. [174] presented an animal classification system to monitor animal migration. Their system can identify five animal species: brown bear, deer, fox, wolf, and wild boar with an accuracy of 94%, using a combination of SURF, SIFT, and FlannBased (FB) as feature descriptors, and SVM as a classifier. Antonio et al. [175] presented an animal detection system to detect the presence of animals on roads and to warn drivers. Synthetic animal images were used to train two supervised ML algorithms: K-Nearest Neighbors (KNN) [176], and Random Forest (RF) [176]. The KNN model outperformed RF in identifying animals on roads and was implemented and tested on a Raspberry Pi 3 B+ [177].

In several applications, deep learning algorithms outperform traditional ML algorithm, particularly in domains with large and diverse training datasets [59]. Saxena et al. [178] assembled a dataset of 31,774 images for various animals and proposed an animal vehicle collision avoidance system. They evaluated the performance of these two object detectors for animals: single shot detector (SSD) [179] and Faster R-CNN. The Faster R-CNN model achieved a mean average precision (mAP) of 82.11% with a detection speed of 90 ms, whereas the SSD model achieved a mAP of 80.5% with a detection speed of 10 ms. Adami et al. [180] developed a smart agriculture application to protect crops from animals by repelling them through generated ultrasounds. They

deployed and evaluated the performance of YOLOv3 and YOLOv3-tiny (a light version of YOLOv3) on different edge computing devices to detect deer and wild boar. The authors achieved an 82.5% mAP by YOLOv3 and a 62.4% mAP by YOLOv3-tiny. Sato et al. [181] presented an animal detection system to prevent accidents involving animals on highways. They used YOLOv4 and YOLOv4-tiny (a light version of YOLOv4) and compared their performance in detecting horses and donkeys. These models were trained on 2,000 downloaded images from Google and tested on 147 images. The accuracy was 84.87% with a detection time of 0.035 ms and 79.87% with a detection time of 0.025 ms, using YOLOv4 and YOLOv4-tiny, respectively. A desktop computer with GPU GTX 1050Ti was used to evaluate the model's speed. Similar to our work, the authors deployed their proposed animal detection system on an embedded system, as a proof of the capability of the system and as an initial step to build highway detection systems.

Deploying deep learning-based animal detection models on embedded devices and reducing the processing delay, which is crucial for real-time applications, poses challenges due to their power constraints. Some studies have applied techniques such as pruning and quantization to compress the CNN models and reduce the computational workload [182] [183], making it suitable for deployment on embedded devices. Although these techniques improve the throughput and reduce the detection delay, they negatively impact the detection accuracy [184] [185]. Field-Programmable Gate Arrays (FPGAs) have been used for hardware acceleration [186] [187]. However, programming them can be complex as it requires knowledge and expertise in hardware programming and optimization. Minakova et al. [188] proposed the concurrent use of using task-level and data-level parallelism to run CNN inference on embedded devices. This approach ensures high-throughput execution of CNN inference, allowing for utilization of the NVIDIA Jetson, which achieves 20% higher throughput compared to standalone CNN inference.

All the above-mentioned approaches have limitations. Some of them use two-stage detectors which are not suitable for real-time applications, as they are slow. Others can only detect one or two animal species. Moreover, there remains room for detection accuracy and speed improvements across all these models.

Chapter 3

Proposed R-CNN Models for Animal Species Detection

In this chapter, our objective is to develop an efficient and accurate animal species detector that can effectively identify and localize various species. To achieve this, new variants of R-CNN models with deformable convolutional layers are proposed. These models are evaluated and compared using three diverse animal datasets to assess their performance in terms of both accuracy and speed. The key idea behind our proposed models is to enhance the extracted features by replacing the regular convolution with deformable convolution, which in return improves the models' capability in detecting animal species. Our goal is to develop an accurate automated labelling and annotation system that eliminates the need for human intervention, saving time and costs. Furthermore, we aim to contribute to the development of robust and reliable systems which can be applied to various purposes in the field of biological sciences, such as wildlife monitoring, conservation, and management.

3.1 Animal Datasets

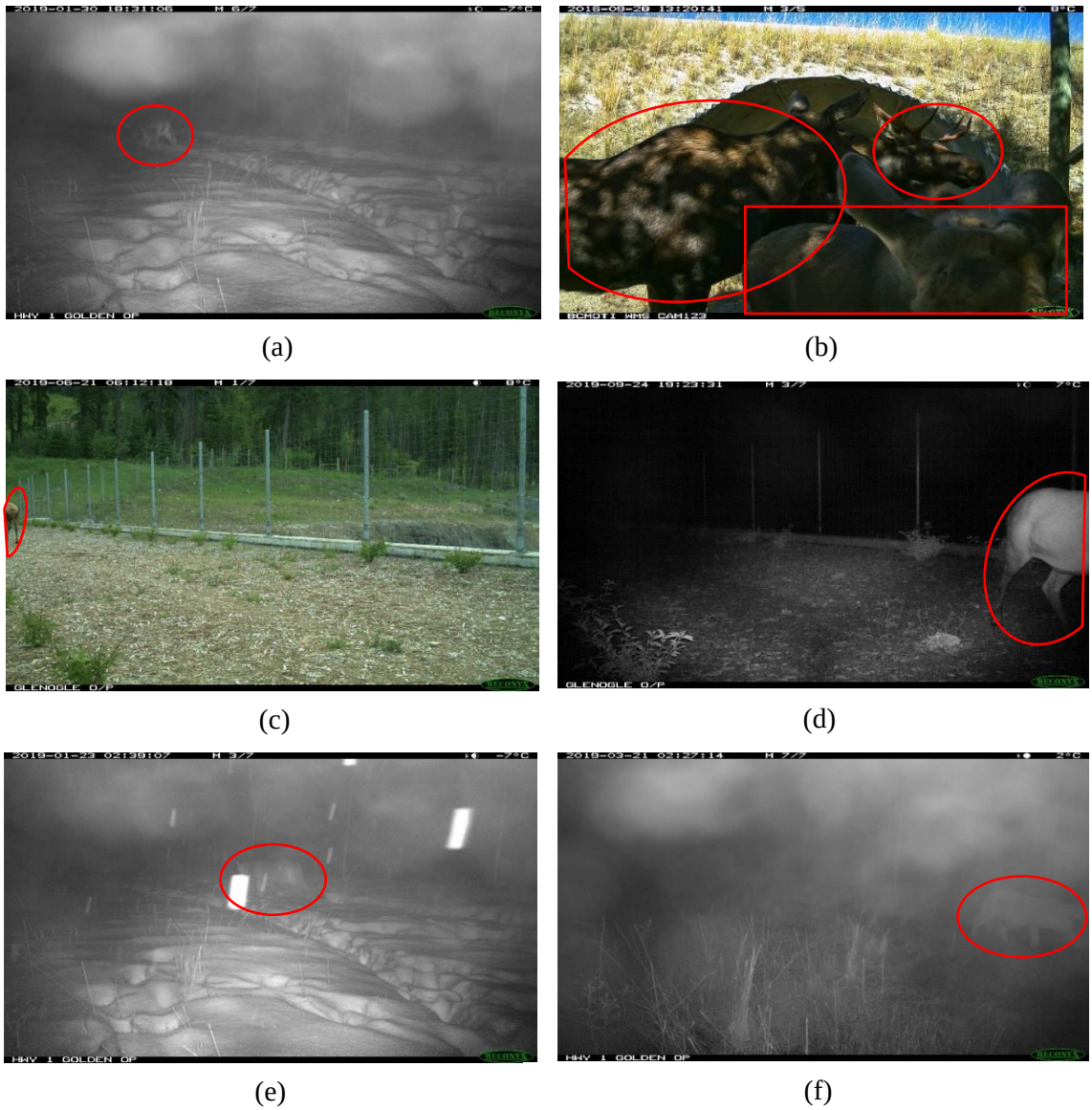
3.1.1 Datasets Used in Our Study

In this chapter, we used three datasets: (1) the Snapshot Serengeti dataset [189], (2) the dataset furnished by BCMoTI, and (3) the Snapshot Wisconsin dataset [190]. The Snapshot Serengeti is the dataset for the animal species in Africa (Serengeti National Park in Tanzania). A total of 712,158 images for seven species (buffalo, deer, elephant, fox, giraffe, lion, and zebra) were selected. The BCMoTI dataset has 53,000 images for eight species (bear, cougar, deer, elk, fox, moose, mountain goat, and wolf) as they are commonly seen in highways and remote areas in Canada. The Snapshot Wisconsin dataset was collected in North America by using 1,037 camera-traps placed in a forest in Wisconsin. It contains 0.5 million images for different animal species, six types of animals were chosen (bears, deer, elk, moose, wolf, and fox) since encounters between these animals and vehicles typically lead to severe crashes on highways. These animals are sometimes involved in tragic direct encounters with humans as well.

The images were labelled by manually as empty or as the name of animal species. The images in the datasets have resolutions ranging between 512 × 384 and 2048 × 1536 pixels. Snapshot Serengeti, BCMoTI, and Snapshot Wisconsin differ in many aspects such as dataset size, camera placement, camera configuration, and species coverage, thus allowing one to draw more general conclusions.

3.1.2 Limitations of Datasets

Detection of animal species in images is challenging due to images' conditions. In some instances, the whole animal covers only a small area of the field of view, as shown in Fig. 3.1(a). In other instances, two or more animals are too close from the field of view and combined with each other, as shown in Fig. 3.1(b). Sometimes, only part of the animal is visible in the field of view, as shown in Fig. 3.1(c, d). Furthermore, different lighting conditions, shadows, and weather, as shown in Fig. 3.1(e, f), can make the feature extraction task even harder.



(a) (b)

(c) (d)

(e) (f)

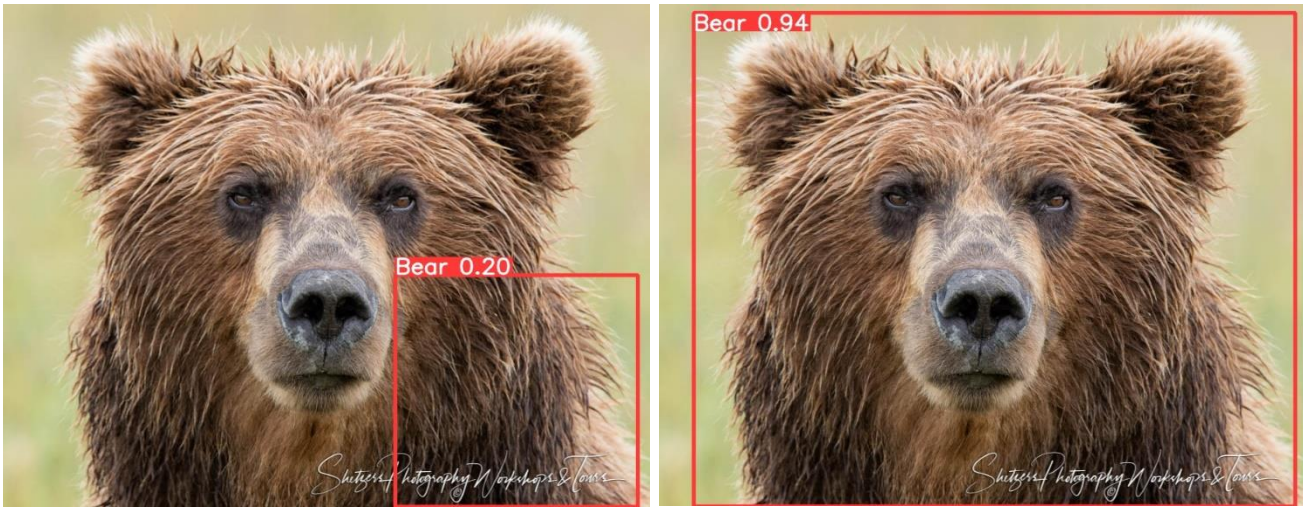
Figure 3.1: Image samples from the dataset used. (a) Low resolution image. (b) An image of three moose close to camera and merge to each other. (c, d) A part of the animal. (e) A night image of cougar with falling snow. (f) A night image of cougar with mist.

3.2 Methodology of Animal Species Detection

This section is dedicated to a comprehensive analysis of various R-CNN models' performance when applied to three animal datasets. To enhance the extracted features and improve the models' capabilities in detecting animals, D-CNN has been integrated into the R-CNN models architecture.

3.2.1 Features Enhancement

Regular R-CNNs extract the features from the image by using a fixed size square kernel. This kernel does not cover properly all the pixels of the target object to precisely represent it. The predicted bounding box using regular R-CNN does not cover the whole animal, as shown in Fig. 3.2(a). As a result, a novel technique is required to enhance the extracted features. By adding deformable convolutional layers to the regular R-CNN animal detectors, the learning of the geometric transformation of animals is possible. These layers can produce adaptive deformable kernel and offset according to the object's scale and shape by augmenting the spatial sampling locations in convolution layers, as explained earlier in Section 2.1.1.2. Therefore, the predicted bounding box using deformable R-CNN covers the whole animal, as shown in Fig. 3.2(b). After experimental tries, three deformable convolutional layers are used to learn offsets, these offsets are added to the regular grid sampling locations in the regular convolution. The detection capability and accuracy are enhanced as reported later in Fig. 3.7, Fig 3.8, and Fig. 3.9 for the three used datasets.



(a) (b)

Figure 3.2: Animal species detection by using: (a) regular convolution, and (b) deformable convolution.

3.2.2 Training

Each of the three datasets has been split into 70% for training, 15% for validation, and 15% for testing, which are the commonly used percentages in similar research. In the training of deep learning models, it is important to find the significant values of hyper-parameters such as: learning

rate, batch size, number of iterations, etc. Reaching the optimum performance of a model is achieved by experiment using various values for these hyper-parameters [191]. A validation dataset is used as well to fine tune the model to prevent overfitting and adjust these hyper-parameters.

The eight R-CNN models, both with and without deformable convolutional layer, were trained using backpropagation. They were then fine-tuned on the validation set to prevent overfitting, utilizing a learning rate of 0.0025 and a batch size of 32. These models' networks were initialized with the ResNet-101 [85] pre-trained model and were fine-tuned end-to-end to enhance training time and improve detection accuracy. All training images were annotated using the MATLAB 2021a's Image Labeler app [192], which provided labelled bounding boxes for the animals present in the images, known as ground truth boxes. These annotations were then exported from MATLAB and converted from the groundTruth format to JavaScript Object Notation (JSON) format to be compatible for training in Python.

To identify animal species, seven pre-trained models are experimented including: AlexNet, GoogleNet, VGG-16, VGG-19, ResNet-18, ResNet-50, and ResNet-101, as shown in Table 3.1. Finally, ResNet-101 was selected as a backbone network for the R-CNN models to detect animals in the training process. The main reason for that selection is the ability to balance between computational complexity and the animal species detection accuracy. ResNet-101 introduces shortcut/skip connections to speed up the convergence of the network and to avoid vanishing gradient problems during the training process, as these problems could stop the network from further learning during training [85] [193]. Furthermore, ResNet-101 achieves competitive accuracy and speed performance in scale-invariant feature extraction.

Table 3.1: Evaluation of animal species identification by using seven pre-trained models on the three datasets.

Pre-trained Models	Accuracy of animal identification
AlexNet	93.1%
GoogleNet	95.9%
VGG-16	96.8%
VGG-19	96.3%
ResNet-18	96.8%
ResNet-50	97.1%
ResNet-101	97.6%

As shown in Fig. 3.3, ResNet-101 consists of five regularized residual convolution blocks (Rconv.1, Rconv.2, Rconv.3, Rconv.4, and Rconv.5) with shortcut connections. These connections prevent overfitting and allow data flow from the input layer to the output layer of each block. The five blocks use 101 hidden layers to extract the image features and to produce feature maps by using 3 × 3 and 1 × 1 filter windows [85]. The output of the last block (Rconv.5) will be the input of max-pooling layer with a stride of 2 pixels to reduce the number of feature maps. FCL flattens these maps to a single vector that can be used as an input of the Softmax classification layer to deal with the thirteen classes of animal species.

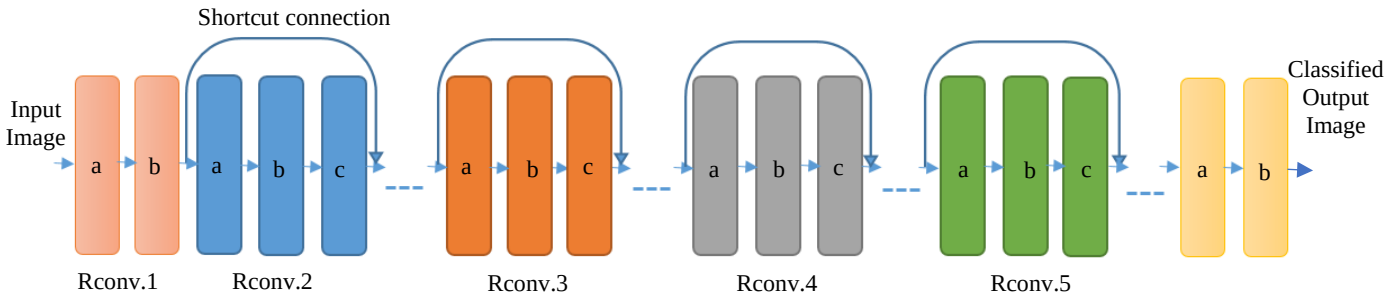


Figure 3.3: Architecture of ResNet-101. Rconv.1 has two layers: a. convolution layer with kernel size (7 × 7) and 64 filters, and b. max pooling layer of size (3 × 3). Rconv.2 has 9 convolution layers with kernel sizes (1 × 1) and (3 × 3) and with different number of filters (64 and 256). Similarly, Rconv.3 has 12 convolution layers, Rconv.4 has 69 convolution layers, and Rconv.5 has 9 convolution layers.

Fig. 3.4 shows the animal species detection procedure for the regular and deformable R-CNN models. The training of the system was applied by using the pre-trained residual network (ResNet-101). First, four regular region-based object detection models (R-CNN, Fast R-CNN, Faster R-CNN, and Mask R-CNN) were trained. Then, four new deformable region-based object detection models were trained after adding three deformable convolutional layers to the last three convolutional layers with kernel size (3 × 3) in the last block of ResNet-101 (Rconv.5).

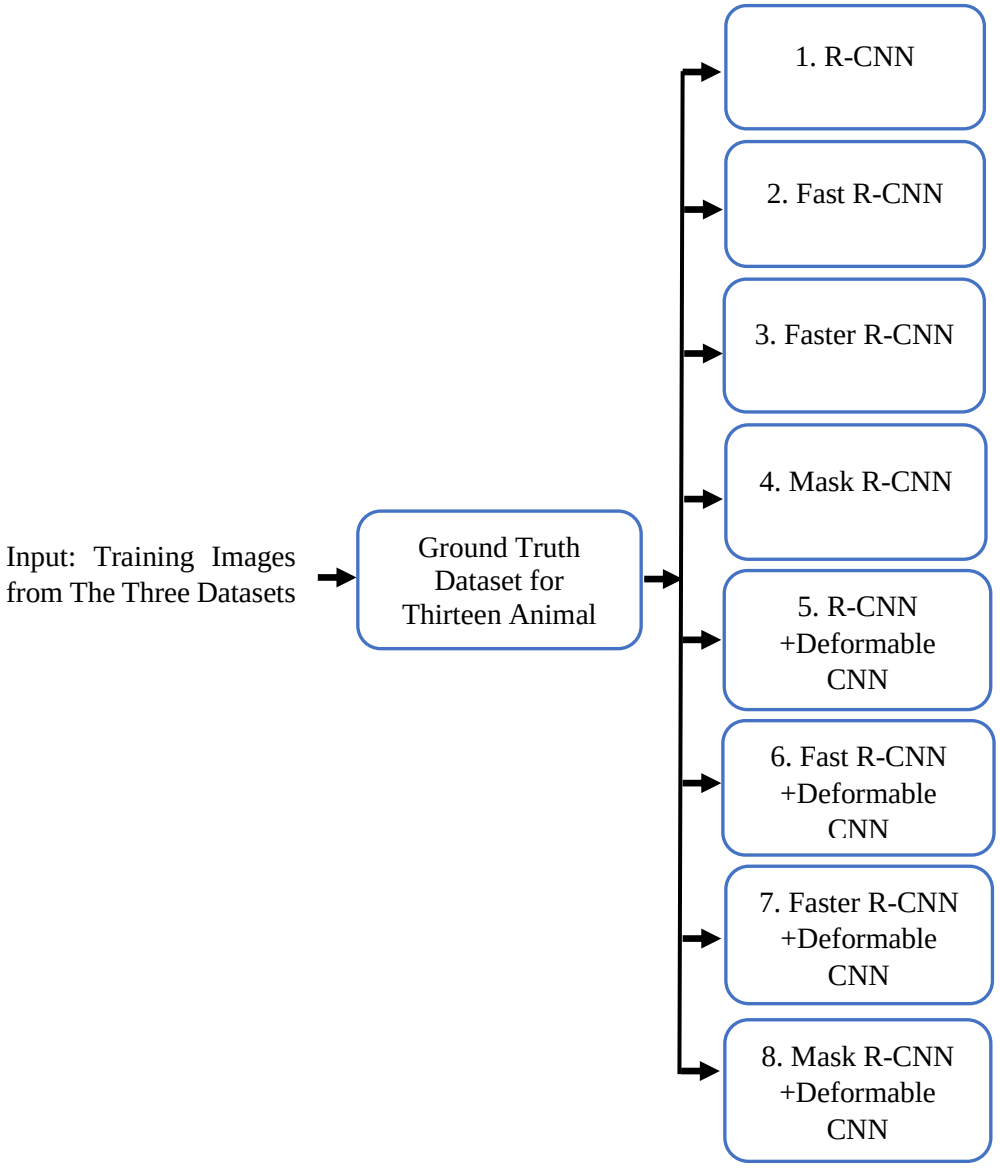


Figure 3.4: Animal species detection training model with eight detectors.

Our work was carried out using Python 3.7 under the PyCharm integrated development environment (deep learning framework) [194], and implemented on a system with Core i7-10750H

Processor, NVIDIA GeForce RTX 2070 graphics accelerator, 32 GB RAM memory, and Windows 10 Professional x64 operating system. Further specifications can be found in Section 4.4.3.

3.3 Experimental Results of Animal Species Detection

3.3.1 Performance Evaluation Metrics

To compare and evaluate the performance of animal species detectors, four metrics are used: False Negative Rate (FNR), accuracy, mean Average Precision (mAP), and inference-time.

IoU measures the overlap "intersection" between the actual bounding box (ground truth) and the predicted bounding box divided by their union. The resulting value shows how close is the predicted bounding box to the ground truth box. To determine if the detection is positive or negative, a predefined IoU threshold value is used. It is important for this threshold value to be calibrated; neither too small nor too large. In object detection research, threshold values ranging from 0.4 to 0.7 are commonly adopted [128] [133]. Fig. 3.5 shows the effect of IoU threshold on the performance of deformable Mask R-CNN. As shown in Fig. 3.5(a), a higher threshold (greater than 0.5) successfully detected two animals, creating two bounding boxes for each animal. Conversely, as shown in Fig. 3.5(b), a lower threshold (lower than 0.5) failed to detect two animals, producing a bounding box for only one detected animal. Thereby, FNR, accuracy, and mAP are measured using an IoU threshold value of 0.5.

(a) (b)

Figure 3.5: Effect of IoU threshold on the animal detection using the deformable Mask R-CNN. (a) High threshold scenario (two bounding boxes generated for each detected bear). (b) Low threshold scenario (only one bear detected, with a single bounding box).

FNR is an essential metric in our evaluation, as it quantifies the number of images that contain animals (positive) but are incorrectly classified as empty images (negative). Thereby, FNR does not consider the specific animal class, and only focuses on the performance of binary classification. In this context, a true positive (*TP*) is defined as the number of correctly classified images with animals, while a false negative (*FN*) represents the number of images containing animals that are falsely classified as empty images. Therefore, the FNR is calculated as:

$$FNR = \frac{FN}{TP+FN} \tag{3.1}$$

Accuracy is an evaluation metric, which is calculated by dividing the total number of correctly predicted objects by the total number of input images, as presented in Equation (3.2). Within this context, *TP* is defined as the accurate detection of a ground truth box, occurring when the IoU is greater than or equal to 0.5, *FN* as the false detection of a ground truth box, which happens when the IoU is less than 0.5, false positive (*FP*) as the false detection of an object that does not exist, and true negative (*TN*) as the number of bounding boxes that are supposed not to be detected inside any image.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \tag{3.2}$$

The mAP is a widely used metric in literature to evaluate the accuracy of object detectors. It is a single number metric that combines both precision and recall by averaging precision across varying recall values from 0. This is essentially calculating the area under the precision-recall curve (PRC) [195] for the detections of each animal class [196]. Then, the result is divided by the number of classes *N* in the dataset as shown in the following equation:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{3.3}$$

where AP_i is the average precision (*AP*) for each animal species class (*i*). It is measured with the Riemann sum as the true area under the PRC, which computes the average value of precision over the recall interval from 0 to 1, as shown in Fig. 3.6.

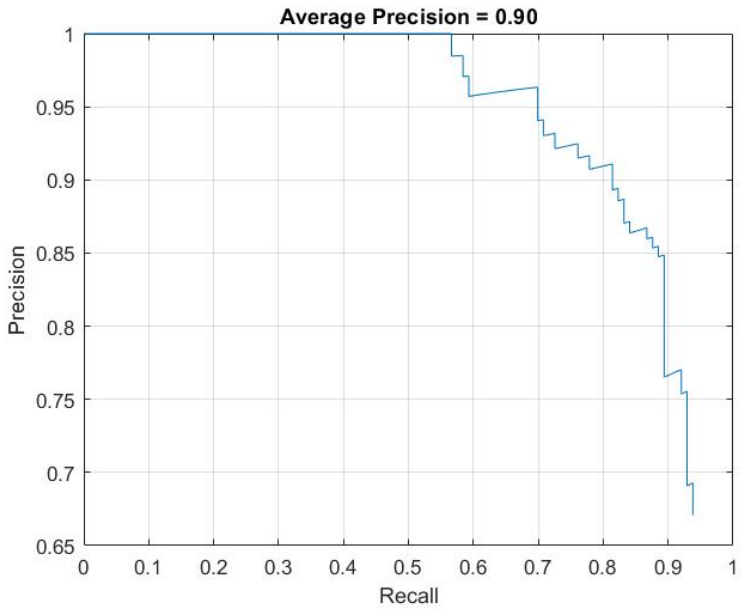


Figure 3.6: Precision-recall curve of detecting deer using Mask R-CNN model.

Precision measures how accurate the object detection model is, as shown in Equation (3.4), so high precision means low false positive rate.

$$Precision = \frac{TP}{TP+FP} \tag{3.4}$$

Recall measures how many correct detections are found by the object detection model, as shown in Equation (3.5), so high recall means a low false negative rate.

$$Recall = \frac{TP}{TP+FN} \tag{3.5}$$

Inference-time, also referred to as elapsed time, is an important evaluation metric which measures the amount of time Python takes to detect animals in a single input image using a given object detector model. This is calculated from the moment the detection process starts until it finishes.

3.3.2 Performance Comparison Results and Discussion

The results in Figures 3.7, 3.8, and 3.9 present the performance of the eight R-CNN models (four regular and four deformable) with FNR, Accuracy (Acc.), and mAP on Snapshot Serengeti, BCMoTI, and Snapshot Wisconsin datasets, respectively. Moreover, Fig. 3.10 presents the

inference-time per image (second) for each of these three datasets. These figures show that deformable Mask R-CNN outperforms the other R-CNNs models, demonstrating superior performance in both detection and instance segmentation of animal species within images. Overall, the results show that the inclusion of deformable convolution layers can significantly improve the model ‘s detection performance.

In Fig. 3.7, according to the evaluation metrics (FNR, Acc., and mAP), deformable Mask R-CNN achieves the highest performance among both regular CNNs and D-CNNs. Deformable Mask R-CNN stands out by providing superior results with an accuracy of 98.4% and a mAP 89.2%. However, it is noted that it incorrectly identified 427 images, which contain animals in the test set as empty images.

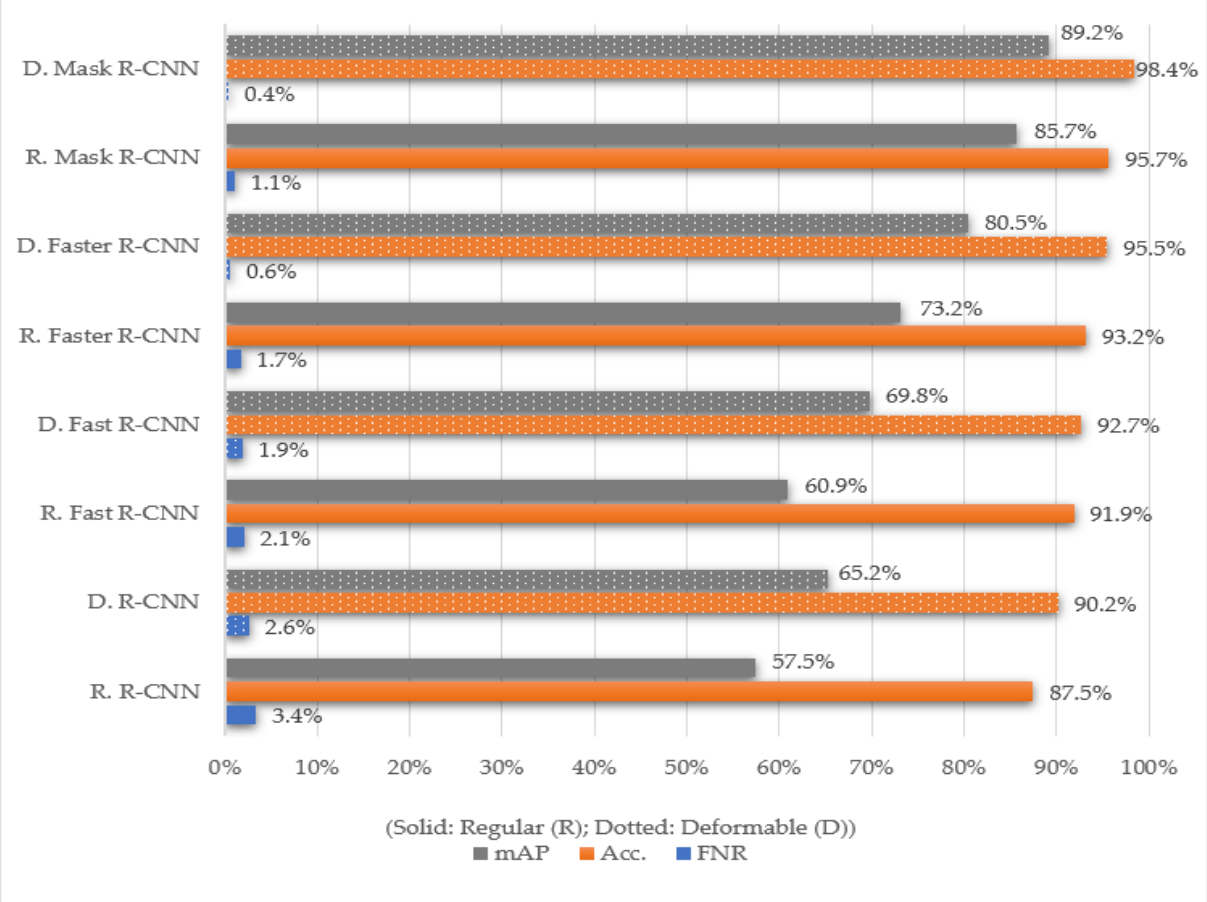


Figure 3.7: Evaluation of object detection models by using both Regular (R.) and Deformable (D.) versions in terms of FNR, Acc., and mAP on the Snapshot Serengeti dataset.

In Fig. 3.8, which represents the performance evaluation on the BCMoTI dataset, the smallest dataset used in this work, the performance of the deformable Mask R-CNN decreases to an accuracy of 93.3%, a mAP of 82.9%, and a 1.7% increase in FNR. This decrease in performance can be attributed to the fact that most of the images in this dataset were captured at night, often with poor resolution and from the backside of the animals, as shown earlier in Fig. 3.1.

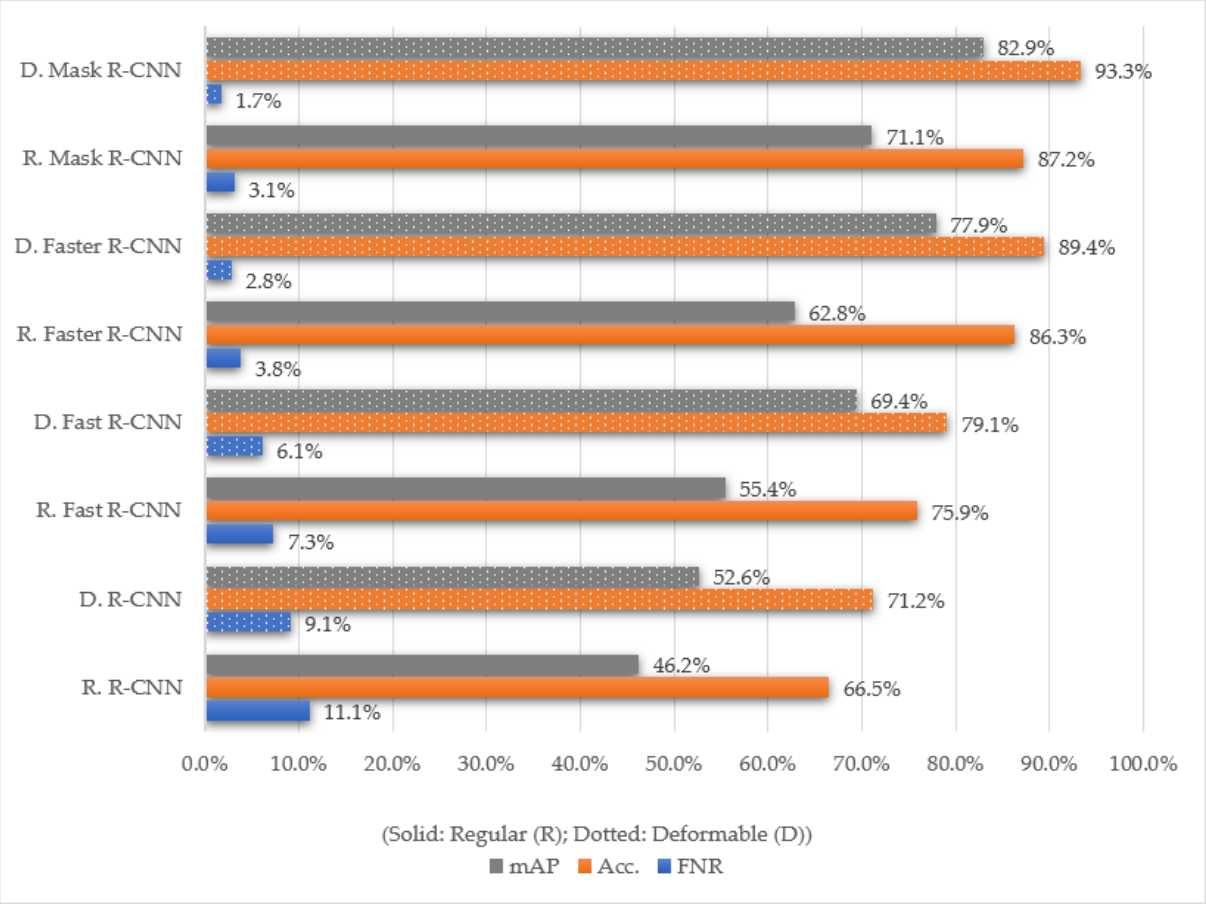


Figure 3.8: Evaluation of object detection models by using both Regular (R.) and Deformable (D.) versions in terms of FNR, Acc. and mAP on the BCMoTI dataset.

Fig. 3.9 shows that by using deformable Mask R-CNN, the accuracy and mAP of detection achieve 97.6% and 87.6%, respectively, on the Snapshot Wisconsin dataset, with a FNR of 0.6%. In the Snapshot Serengeti dataset, the system was trained on a larger training set compared to that of Snapshot Wisconsin and BCMoTI. This resulted in an increase in accuracy by up to 5.1% compared to BCMoTI, and up to 0.8% compared to Snapshot Wisconsin. This indicates the importance of having a large training set with a significant number of instances for each class.

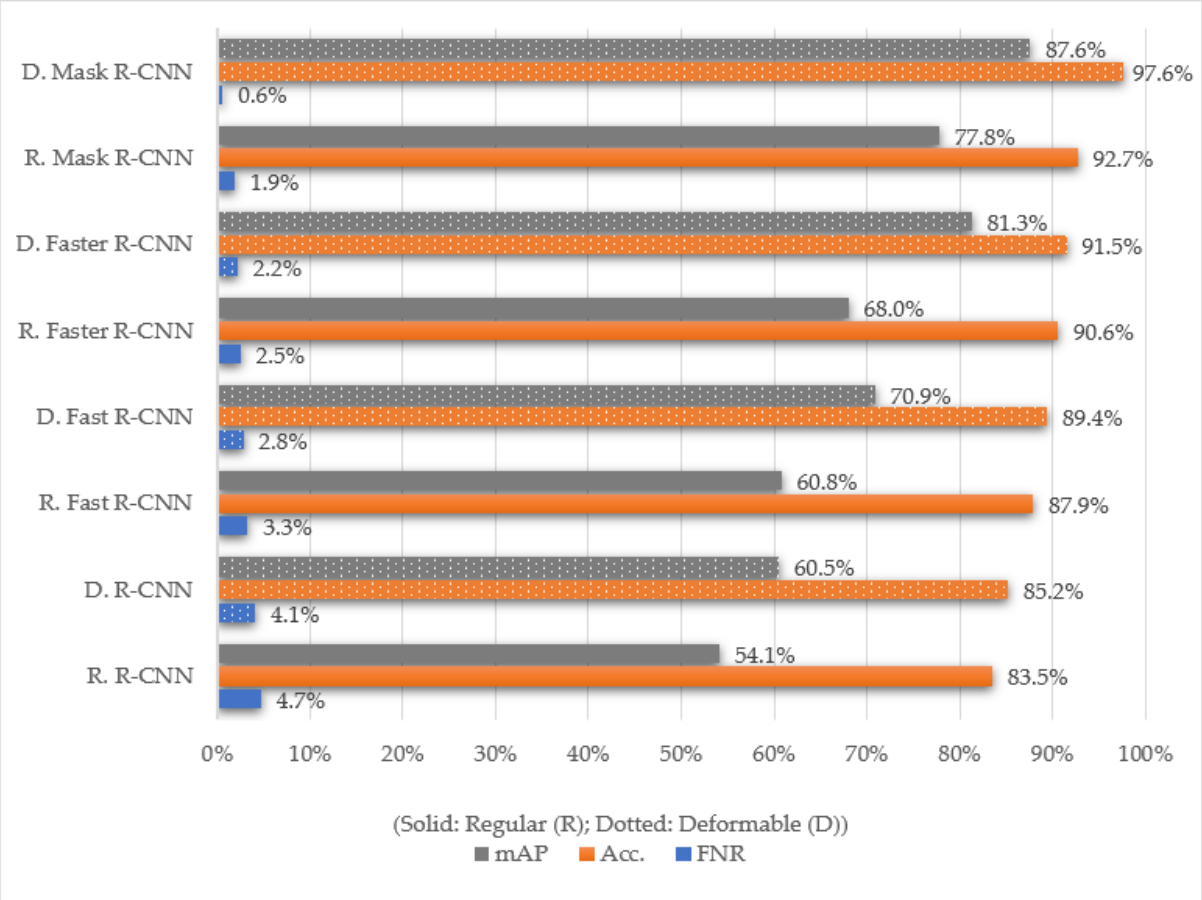


Figure 3.9: Evaluation of object detection models by using both Regular (R.) and Deformable (D.) versions in terms of FNR, Acc. and mAP on the Snapshot Wisconsin dataset.

Dataset		Regular CNN			Deformable CNN		
		Acc.	**mAP**	**Test-time**	**Acc.**	**mAP**	**Test-time**
Snapshot Wisconsin	**R-CNN**	83.5%	54.1%	48 sec	85.2%	60.5%	57 sec
	Fast R-CNN	87.9%	60.8%	2.3 sec	89.4%	69.9%	2.56 sec
	Faster R-CNN	90.6%	68.0%	1.2 sec	91.5%	71.3%	1.25 sec
	Mask R-CNN	92.7%	77.8%	0.78 sec	97.6%	87.6%	0.70 sec

As shown in Fig. 3.10, deformable Mask R-CNN is able to detect objects in about 0.78 second per image on all three datasets. That makes deformable Mask R-CNN, though slightly slower than the regular version, suitable for some real-time applications, depending on their specific requirements, such as automated wildlife monitoring.

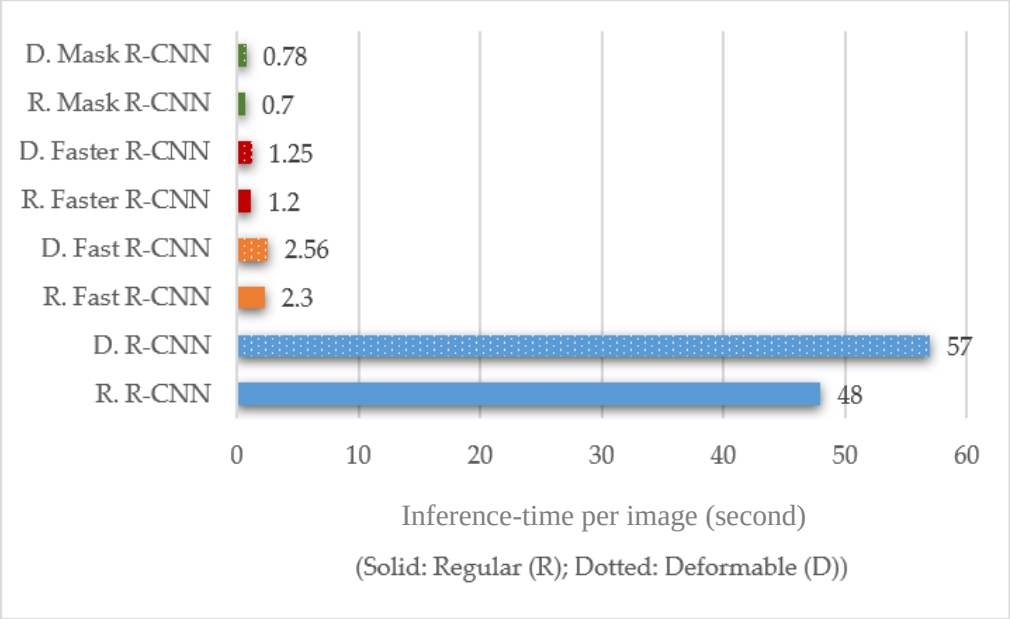


Figure 3.10: Evaluation of object detection models by using both Regular (R.) and Deformable (D.) versions in terms of inference-time on the three datasets.

The visual results in Fig. 3.11 show that deformable Mask R-CNN is capable of detecting and segmenting single and multiple animal species with a confidence score for each class. The deformable Mask R-CNN detects animal species with higher accuracy and speed in comparison to other regular and deformable R-CNN models. Therefore, deformable Mask R-CNN not only detects single and multiple animal species, but it also can produce a mask over each detected animal in the image. This feature is useful for counting the number of occluded and overlapping animal species.

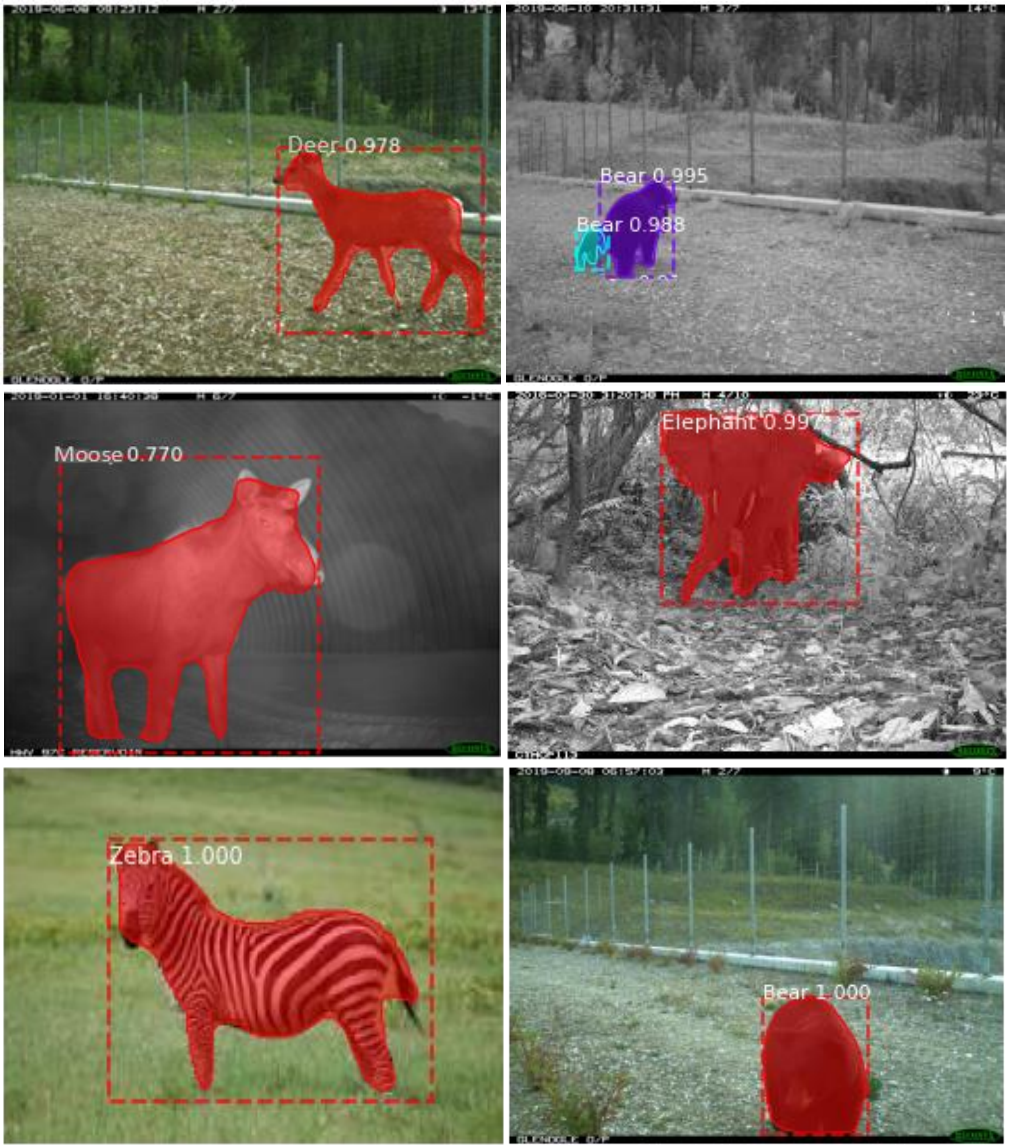


Figure 3.11: Some examples of animal species detection after deformable Mask R-CNN (output mask size is the object size).

In general, our results indicate that the deformable Mask R-CNN using ResNet-101 can detect and segment animals with remarkable accuracy, exceeding the performance of similar work, as shown in Table 3.2. This table summarizes the datasets, performance metrics, and techniques employed in our research and other relevant studies on animal species detection. The integration of D-CNN into Mask R-CNN improves the performance of animal species detection. Our research has an improvement over other related works due to the following reasons:

1. Three datasets of different characteristics have been used for training and testing.

2. Deformable convolutional layers have been added to the R-CNN detectors, which have a great effect on enhancing the extracted features, which in turn improve the performance of animal species detection.

Table 3.2: A comparison of related work in animal species detection.

References	Year	Dataset	Performance	Technique
Parham et al. **[160]**	2016	2,500 images of Plain and Grevy zebras	mAP of zebra detection: 55.6% for Plain and 56.6% for Grevy	YOLOv1 detector
Norouzzadeh et al. **[154]**	2019	Snapshot Serengeti	Accuracy of animal species detection 92.9%	Deep active learning
Xu et al. **[100]**	2020	750 images of cattle	Accuracy of cattle detection: 94%	Mask R-CNN
Gupta et al. **[168]**	2020	MS COCO dataset [72]	AP of detection: 79.47% for cows, and 81.09% for dogs	Mask R-CNN
Saxena et al. **[178]**	2021	31,774 images of various animals	mAP of animal species detection: 82.11%	Faster R-CNN
Yılmaz et al. **[197]**	2021	1,500 images of cattle	Accuracy of cattle detection: 92.85%	YOLOv4
Sato et al. **[198]**	2021	2,000 images of donkeys and horses	Accuracy of donkeys and horses' detection: 84.87%	YOLOv4
Our work	2021	Snapshot Serengeti, BCMoTI, and Snapshot Wisconsin	Accuracy, and mAP of animal species detection respectively: 98.4% and 89.2% in Snapshot Serengeti, 93.3% and 82.9% in BCMoTI, 97.6% and 87.6% in Snapshot Wisconsin	Deformable Mask R-CNN

3.4 Case Study: Automated Image Labelling and Annotation System

Both the labelling and annotation process of images are time-consuming, labor-intensive process, and expensive, considering the costs for wages and infrastructure. Therefore, an automated image labelling and annotation system is needed to save time and money without requiring human intervention.

As discussed previously, the proposed deformable Mask R-CNN model was employed to build an automated labelling and annotation system, as shown in Fig. 3.12. This system is capable of labelling, segmenting, and annotating six North American wildlife species (bear, cougar, deer, elk, moose, and mountain goat). To expand this model's ability to detect new two classes (human and vehicle), a set of images for each class was collected from various sources such as the UA-DETRAC [199], Kaggle [200] [201], and MPII [202] websites. This has extended the model's capability to accurately detect all the eight classes. The whole images will be fed to the automated labelling and annotation system, to label them into eight classes, and save each image with its predicted identification probability (score) in the designated labelled folders. If the image has two different object classes, the system creates two labelled folders for each. Each folder contains the image with an annotated bounding box around the object which belongs to this folder. Furthermore, this system is capable of converting bounding box annotations from JSON format to various other formats, including text format, XML format, and groundTruth format. This ensures the system's adaptability to meet the requirements of different object detection algorithms and frameworks. For instance, the YOLO model, which will be discussed in detail in Chapter 4, uses text format annotations for training.

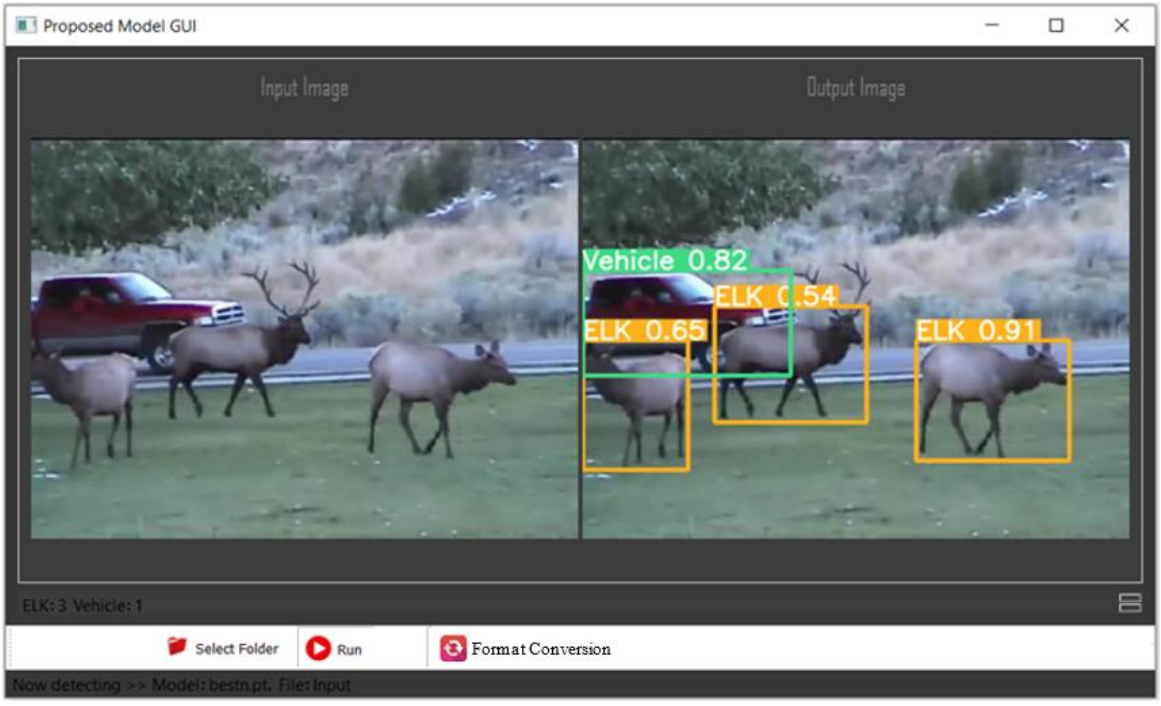


Figure 3.12: An automated labelling and annotation system.

This system was utilized to detect and label 224,509 unlabelled images provided by BCMoTI, achieving an overall accuracy of 96.8%. Upon completion of our work, we plan to make the automated labelling and annotation system publicly available to the research community.

3.5 Conclusion

In this chapter, a new idea of integrating deformable convolutional layers into the regular R-CNN models was proposed for high accuracy animal species detection. The performance of four regular R-CNN models was evaluated on animal images from three distinct datasets. Subsequently, the accuracy and speed performance of animal species detection were provided after enhancing the extracted features by using four new deformable R-CNN models. The results show that deformable Mask R-CNN is the suitable choice in animal species detection, and it can achieve the best performance in terms of FNR, accuracy and mAP, as shown in Table 3.2.

Deformable Mask R-CNN is capable of handling geometric variations or deformations of an object without the need for further training on datasets, which reduces validation time and costs. Moreover, as shown in Fig. 3.11, deformable Mask R-CNN provides promising results for detecting animal species under a wide range of lighting conditions, shadows, and weather scenarios.

In conclusion, the use of deformable Mask R-CNN not only improves the detection accuracy of animal species, but also contributes to the development of an accurate automated image labelling and annotation system. Traditional image annotation for object segmentation or detection is labor-intensive and time-consuming. However, with the automated image annotation system based on deformable Mask R-CNN, this task becomes simpler, more accurate, and efficient, enhancing the overall quality and performance of the resulting object detection and segmentation models. Image annotation is an essential component in the training process of supervised object detection models.

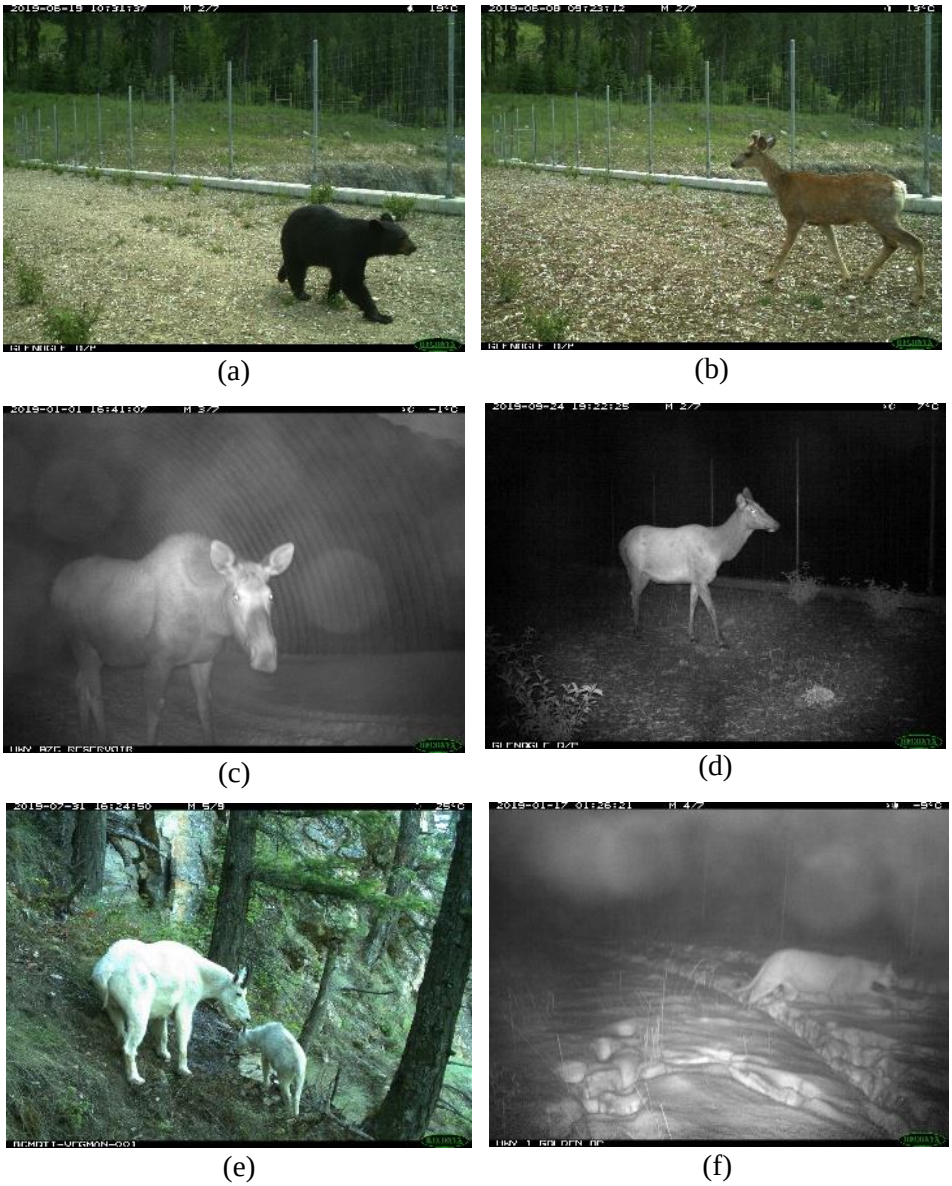
Chapter 4

M-YOLO Animal Species Detector

YOLO object detection models have complex architectures as mentioned before in Section 2.1.2.2. Subsequently, they require a large number of parameters or weights to make predictions, which are learned from the training dataset. Therefore, these models require a powerful platform, such as Graphic Processing Unit (GPU) [203] and a large memory, to be efficient and effective in real-time applications. However, most real-world applications utilizing embedded GPU or Central Processing Unit (CPU) devices are constrained by limited storage space and low computational power. To overcome this, many studies proposed lightweight YOLO object detection models that are more suitable for less powerful platforms. These include Fast YOLO [204], YOLOv3-Tiny [205] [206] [207], and YOLOv4-Tiny [208] [209]. These models have simpler architectures with fewer parameters than regular YOLO object detection models. Although, the detection speed of these lightweight models has improved with limited hardware platforms, the detection accuracy decreased [210] [211] [212] [213]. Therefore, accurate, fast, and lightweight object detector that can be implemented on portable embedded platforms with limited resources remains a challenge.

4.1 Our Dataset

The dataset used in this chapter was provided by the BCMoTI. This dataset has 138,482 unlabelled images from various locations and angles of six North American animal species: bear, cougar, deer, elk, moose, and mountain goat, and 20,114 images without animals. These images have resolutions ranging between 512 × 384 to 2,048 × 1,536 pixels, which were captured using Reconyx camera trap [214]. Fig. 4.1 shows example images of the animal species in the dataset. Due to the hazy and snowy night images of cougars, as shown in Fig. 4.1(f), a video of cougar was downloaded from YouTube, transferred to a sequence of images, and used for training to improve the identification performance. Table 4.1 shows the percentage of each animal species and images with no animals in the dataset.



(a) (b) (c) (d) (e) (f)

Figure 4.1: Example of images from our dataset. (a) Bear, (b) deer, (c) moose, (d) elk, (e) mountain goat, and (f) cougar.

Table 4.1: Percentage of the animal species in our dataset.

Animal Species	**Percentage**
Bear	16%
Cougar	13%
Deer	16%
Elk	15%
Moose	14%
Mountain Goat	13%
No Animal	13%

4.2 YOLOv2 Model

YOLOv2 architecture was selected in this work to be modified mainly due to: (i) the detection capability of YOLOv2 has been proved when there is a variety of classes with big differences [143], and (ii) the limited computational resources, YOLOv2 is still one of the most used detectors in many applications in industry [161] [215] [216] [217] [218]. Therefore, a modified YOLOv2 model is proposed in this work to be deployed to embedded system. The superiority of adding a pass-through layer, adding three Deformable Convolutional Layers (DCLs), and removing two high convolutional layers in the YOLOv2 architecture is proven through comparative analysis between YOLOv2, modified YOLOv2, YOLOv3, and YOLOv4 models, as will be shown later in Section 4.5.

A deep learning architecture consists of: (i) a deep, fully convolutional network to generate feature maps for the input image, and (ii) a shallow network for a specific task to generate results from the feature maps [95]. As shown in Fig. 4.2, DarkNet-19 was used as a backbone network for YOLOv2 for feature extraction [143]. It consists of 19 successive convolutional layers with kernel size of 3 × 3, 1 × 1 and five 2 × 2 max. pooling layers with stride = 2 for feature extraction on top of the Softmax classification layer.

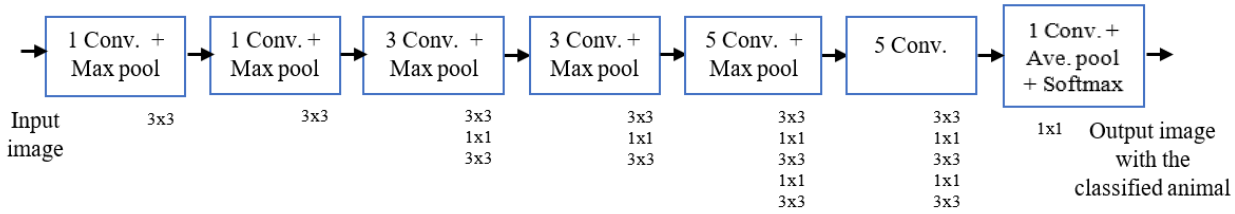


Figure 4.2: Architecture of DarkNet-19.

Fig. 4.3 shows the original architecture of YOLOv2, the last block of DarkNet-19 architecture has been replaced by two blocks (blue dashed blocks) with three 3 × 3 convolutional layer and one 1 × 1 convolutional layer for detection. In addition, YOLOv2 has a pass-through layer named Reorganisation layer (Reorg. layer) (black dashed block), which is used to reduce the size of mid-level features to match the size of the high-level features by using a down-sampling factor equal to 2. Thereby, these two feature maps can be combined to improve network performance.

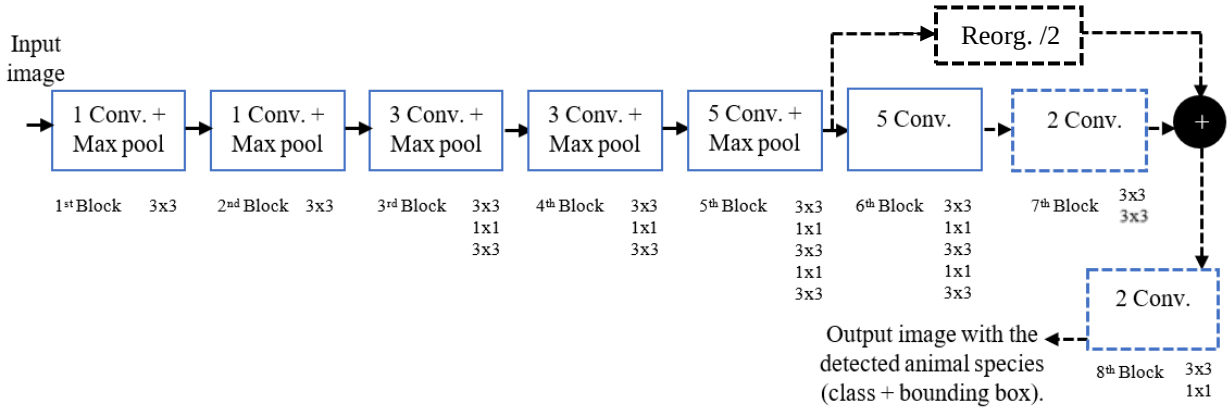


Figure 4.3: Architecture of YOLOv2.

4.3 Proposed Animal Species Detector (Modified YOLOv2: M-YOLO)

The YOLOv2 model has shown impressive results in many generic object detection applications which have different degrees of variation in object classes, such as humans, animals, traffic signs, and vehicles [219] [220]. This motivated us to propose a lightweight animal species detector based on YOLOv2 to detect animal species. However, there are a lot of challenges in this work to improve detection accuracy and speed to be deployed on portable platforms with limited resources (embedded device).

4.3.1 Multi-Level Features Merging

The differences between the six animal species are related to edges, intensity, contour, texture, color, etc., which exist in low-level features. To improve animal species identification and localization accuracy of the original YOLOv2, the low-level features are merged with high-level features by adding a new pass-through layer (Reorg. layer) with a down-sampling factor of 4; as shown by the black dashed line in Fig. 4.4.

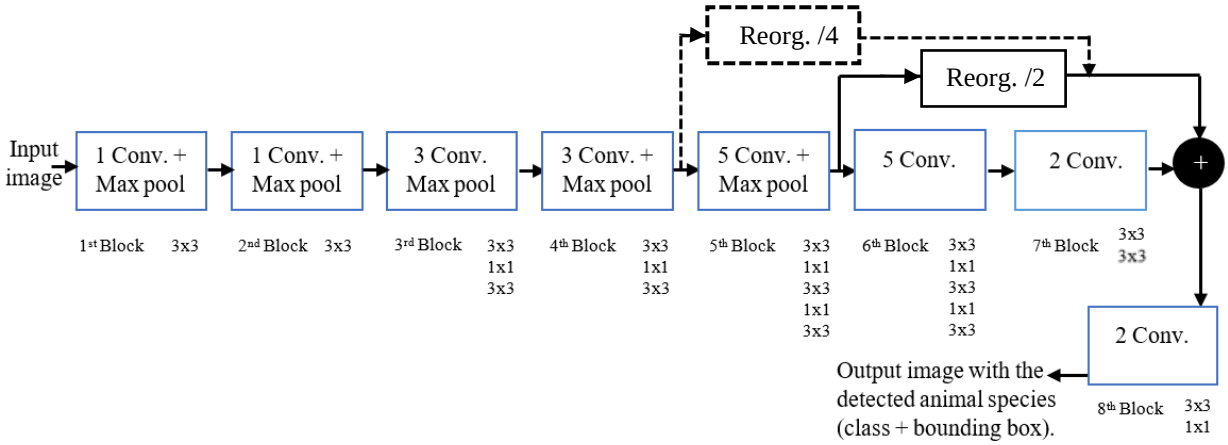


Figure 4.4: Architecture of YOLOv2 after adding a passthrough layer (Reorg./4).

As shown in Fig. 4.5, after the low- and high-level features merging, the capability of YOLOv2 to recognize the small differences between animal species increased by 1.8%. The biggest improvement was achieved with the deer images as most of these images were captured during daylight, as shown in Fig. 4.1(b), and the smallest improvement was with the moose images, as most of them were captured at night as in Fig. 4.1(c).

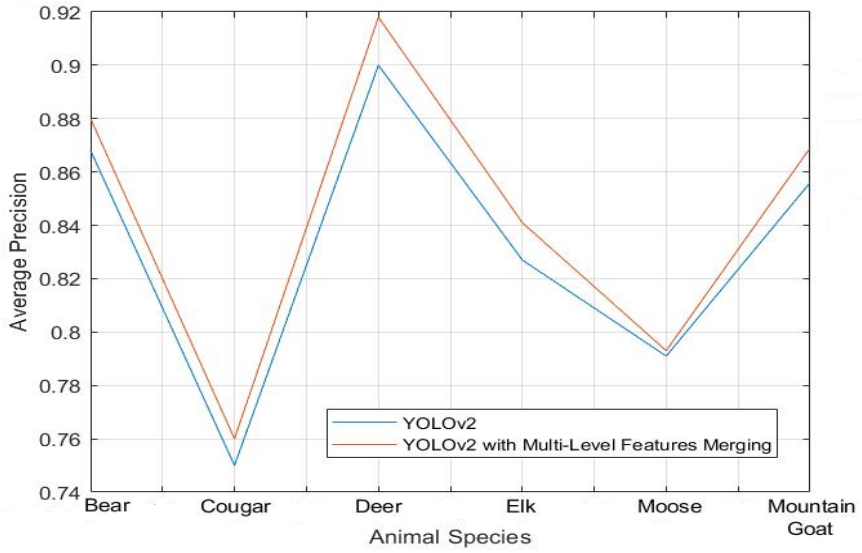


Figure 4.5: Comparison between YOLOv2 and M-YOLO with multi-level features merging to detect six animal species in terms of AP on the BCMoTI dataset.

4.3.2 Convolutional Layers Removal

The architecture of YOLOv2 was designed for generic object detection applications. Generic objects imply that the number of classes is large. Thereby, there were five repeated 3 × 3 convolutional layers with 1024 filters in the sixth and seventh blocks of the YOLOv2 architecture. However, only six species of animals need to be detected in this study. Fig. 4.6 shows the loss curve (sum of errors for each iteration) of detecting animal species in the validation dataset with and without two 3 × 3 convolutional layers in the seventh block of YOLOv2 architecture. The two models, with and without two convolutional layers, were trained with the same number of epochs, and both reached the same accuracy. Thus, removing the two 3 × 3 convolutional layers from the seventh block of YOLOv2 does not reduce the accuracy of detecting the six selected animal species, however it speeds up the detection process. It can be concluded that, YOLOv2 without the two 3 × 3 convolutional layers, has sufficient features to distinguish between the six animal species.

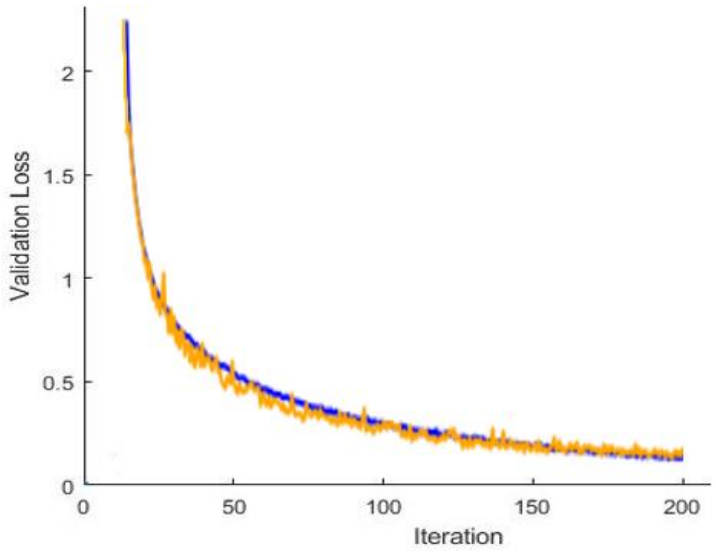


Figure 4.6: The loss curve of the validation dataset with (blue line) and without (orange line) two 3 × 3 convolutional layers in the seventh block of YOLOv2 architecture.

4.3.3 Adding DCLs

For object classification, regular CNNs extract features from images by using a fixed kernel. If any changes occur in the shape, size, and posture of objects in the image due to the object's motion, or the locations of the cameras, the neural network may have difficulties in classifying these objects

correctly. Therefore, in this study, the idea of adding DCLs to the YOLOv2 model was proposed to learn the geometric transformation of animals. It can produce an adaptive deformable kernel and offsets according to the scale and shape of the animal, by augmenting the spatial sampling locations in the convolution layers. This is used as our solution instead of augmenting the dataset in advance, which increases the cost of data pre-processing, and the augmented data never cover all the images in real applications [96]. As shown by the blue dashed box, in Fig. 4.7, three DCLs are used to learn the offsets, which are added to the regular grid 3 × 3 sampling locations in the sixth block. The detection capability and accuracy were enhanced as reported in Table 4.2.

Table 4.2: Evaluation of animal species detection by using regular and deformable YOLOv2 in terms of AP, mAP, and Inference-time on BCMoTI dataset.

Animal Species	**YOLOv2 (AP)**	**Deformable YOLOv2 (AP)**
Bear	0.860	0.893
Cougar	0.750	0.764
Deer	0.900	0.911
Elk	0.823	0.851
Moose	0.791	0.836
Mountain Goat	0.856	0.875
mAP	0.830	0.855
Inference-time (second)	0.025	0.027

Table 4.2 shows the AP, mAP, and inference-time, which were elaborated on earlier in Section 3.3. The deformable YOLOv2 reaches higher accuracy in detecting the six animal species as compared to the regular YOLOv2, as expected. This adaptability results in more accurate feature mapping and object localization, leading to the observed increases in AP for each species. While the percentage increases in accuracy might appear small, they are significant in the context of object detection, especially in wildlife monitoring. This improvement is crucial in applications where accurate wildlife detection is paramount, such as in reducing wildlife-vehicle collisions. In the BCMoTI dataset, the deformable YOLOv2 provides an enhanced result for the six animal species with a mAP of 85.5% in about 0.027 second per image on a Core i7 system, will be specified in Section 4.4.3, which equates to 37 images in one second.

To summarize the above discussion, the proposed M-YOLO, as shown in Fig. 4.7, differs from the original YOLOv2 by: (i) adding a pass-through layer with a down-sampling factor equal to 4, which increases the mAP of detecting six animal species by 1.8%, (ii) removing two repeated

3 × 3 convolution layers from the seventh block of YOLOv2 speeds up the detection process while maintaining the accuracy of detecting the six animal species, and (iii) adding three DCLs to the sixth block (blue dashed box) to deal with the geometric transformation of animals in images, which increases the animal species detection mAP by 3.0%.

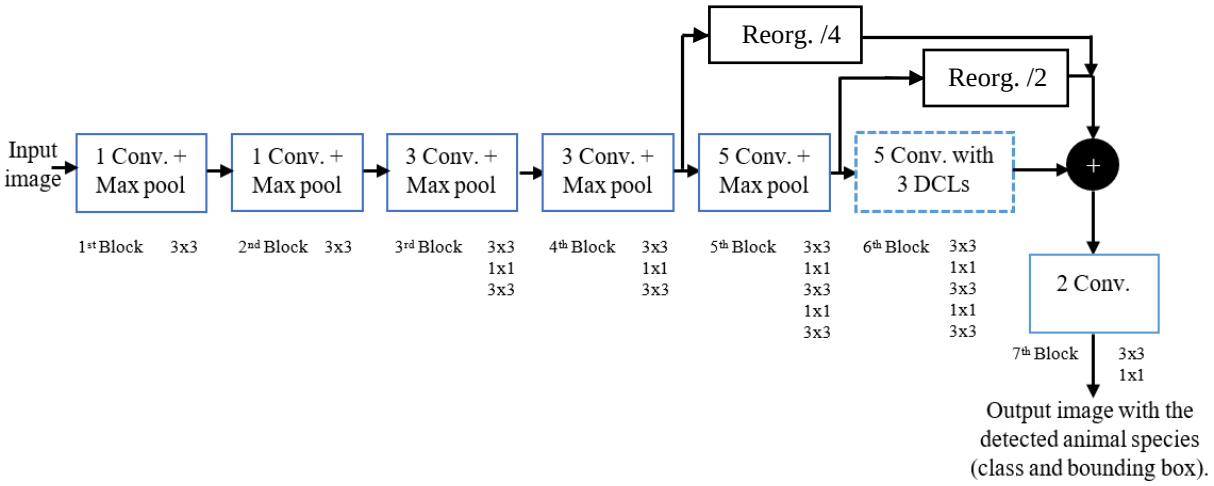


Figure 4.7: Architecture of the proposed M-YOLO animal species detector. The detector has a new pass-through layer with a down-sampling factor of 4 to merge low- with high-level features. The sixth block of detector (blue dashed block) has five convolution layers with three added DCLs.

4.4 Implementation and Experimental Setup

In this section, the training settings and the workstation and experimental setup configurations for the YOLO models are introduced.

4.4.1 RoI Labelling

Image annotation was conducted using our automated image labelling and annotation system, as described in Section 3.4. This system accepts a sequence of images, allowing for the annotation of objects within these images using rectangular bounding boxes with associated class labels. These annotations are then converted from JSON format to YOLO format (image and corresponding text file) to prepare them for training.

4.4.2 Training

The dataset was divided into 70% training, 15% validation, and 15% testing. The validation dataset is important for adjusting and finding the significant values of the hyper-parameters through trial-

and-error [221]. These hyper-parameters have a vital effect on the performance of the YOLOv2, M-YOLO, YOLOv3, and YOLOv4 models. In this work, all four models used the same hyper-parameters. The Adaptive Moment Estimation (Adam) optimizer [222] was used with an empirically determined momentum of 0.9 and a weight decay of 0.0005, to accelerate the convergence to improve the detector performance. The models were also trained and fine-tuned by using a learning rate of 0.001.

The training was performed after resizing the input images to 416 × 416 pixels by a pre-trained ImageNet DarkNet network to extract the objects' features from the input images. All the models were initialized with the ImageNet weights [71] and trained with the same number of epochs.

4.4.3 Experimental Setup

The training and testing of YOLO and M-YOLO models were performed using Python 3.7 under the PyCharm development environment and implemented on a system with Core i7-10750H Processor, NVIDIA GeForce RTX 2070, 32 GB Dual-channel DDR4 RAM memory with 2666 MHz, and Windows 10 Professional x64 operating system. For the effective use of the system's GPU to accelerate the training process, CUDA 10 [223] was installed.

For in-field use, there are many embedded devices [224] [225]; however, in our work Raspberry Pi [226] was chosen as the embedded platform for the following reasons: (1) low price, (2) less power consumption, (3) wide availability from many international suppliers, (4) compatibility with many operating systems and open-source software, (5) big community of users, and (6) strong technical support. In our work, a Raspberry Pi 4 Model B (RP4B) with 1.5GHz 64-bit quad-core CPU, 128 GB, and 8 GB RAM memory [227] was used for deployment.

4.5 Results and Discussion

This section shows the accuracy and detection speed comparisons between YOLOv2, YOLOv3, YOLOv4, and M-YOLO models on the BCMoTI test dataset, which contains 20,772 images of six animal species.

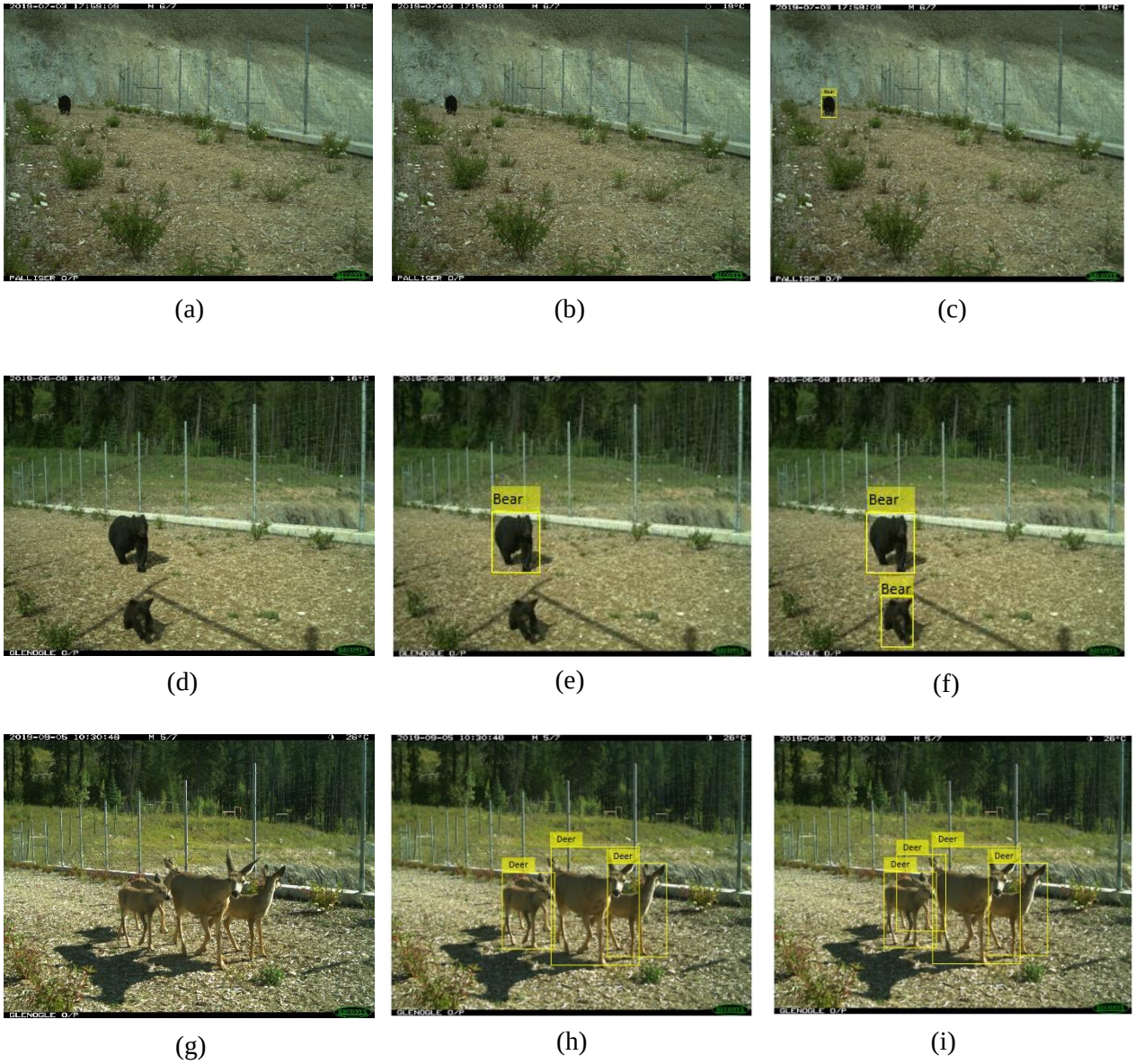
As shown in Table 4.3, the proposed M-YOLO model was evaluated and compared with the other models. Two machines with different computation power were used to measure the inference-time. For the high-power machine (Core i7 system), M-YOLO model has a higher

accuracy (mAP) than the regular YOLOv2 by 5.0% and faster detection speed by 3 ms. The accuracy of YOLOv3 is 4.5% higher than the regular YOLOv2 model; however, accuracy of the M-YOLO model is higher than YOLOv3 by 0.4%. In addition, YOLOv2, even after modifications, is still faster than YOLOv3. However, YOLOv4 model is more accurate than M-YOLO model by 2.5% and faster by 2 ms. The YOLOv4 model has complex architecture and many parameters. Therefore, YOLOv4 requires a powerful platform, which restricts its deployment on embedded devices for portable real-time applications. On the other hand, for the low-power machine (RP4B), the M-YOLO model (lightweight model) is faster than YOLOv4 by 70 ms. Although, accuracy of the proposed model is lower than YOLOv4, this difference in accuracy can almost be ignored when compared with the improvement in the detection speed, which makes M-YOLO suitable for deployment on embedded devices.

Table 4.3: Evaluation of animal species detection by using YOLOv2, M-YOLO, YOLOv3, and YOLOv4 models in terms of AP, mAP, and Inference-time on BCMoTI dataset.

Animal Species	**YOLOv2 (AP)**	**M-YOLO (AP)**	**YOLOv3 (AP)**	**YOLOv4 (AP)**
Bear	0.86	0.915	0.907	0.920
Cougar	0.750	0.778	0.762	0.831
Deer	0.900	0.936	0.925	0.915
Elk	0.823	0.865	0.854	0.871
Moose	0.791	0.838	0.860	0.898
Mountain Goat	0.856	0.904	0.903	0.930
mAP	0.830	0.872	0.868	0.894
Inference-time (second) on Core i7 system	0.025	0.022	0.049	0.020
Inference-time (second) on RP4B	0.13	0.16	1.54	0.23

In Fig. 4.8, the YOLOv2 and M-YOLO models were applied to some images for comparison. This is to evaluate whether the M-YOLO model can solve the limitations of the original YOLOv2. The left-column images Fig. 4.8(a, d, and g) are the original images, the middle-column images Fig. 4.8(b, e, and h) show the detection results with the original YOLOv2 model, and the right-column images Fig. 4.8(c, f, and i) show the detection results with the M-YOLO model. From Fig. 4.8(b, e), it is noted that YOLOv2 struggles to detect small animals while M-YOLO can detect them, as shown in Fig. 4.8(c, f). Also, it can be seen that YOLOv2 has difficulty with occluded animals, as shown in Fig. 4.8(h), but is resolved by adding three DCLs to YOLOv2, as in Fig. 4.8(i).



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 4.8: Animal species detection results of two models. (a) Original image of a far bear, (b) no detected animal (FN) using YOLOv2, (c) detected far bear after using M-YOLO, (d) original image of two bears, (e) detected only one bear (big) using YOLOv2, (f) detected two bears (big and small) using M-YOLO, (g) original image of four deer, (h) detected only three deer using YOLOv2, and (i) detected four deer using M-YOLO.

The results in Fig. 4.8 show that the proposed M-YOLO model is more reliable than YOLOv2 in detecting small and occluded animals. Moreover, it reduces the false negative when there is an animal in the image which is not detected by YOLOv2. These enhancements can be attributed to the multi-level features merge and DCLs in our M-YOLO model.

4.6 Case Study: WHCs Mitigation system Prototype

The goal of this work is to propose a prototype for a novel, robust, and real-time WHCs mitigation system. This system, as shown in Fig. 4.9, can be installed in urban areas such as front and backyards, or school playgrounds. As illustrated in Fig. 4.10, the WHCs mitigation system consists of two main subsystems: a detection subsystem and a warning subsystem.

The detection subsystem, represented by the black dashed box, contains a PIR or IR live camera and a RP4B processor. These components work together to process our proposed detection model (M-YOLO) to detect moving objects and identify them as either humans or common North American wildlife species, including bear, cougar, deer, elk, moose, and mountain goat, with high accuracy and near real-time responsiveness.

The warning subsystem, represented by the blue dashed box, features a custom-developed Android application named “Be Safe”, as shown in Fig. 4.11. This application alerts humans to be cautious, thereby contributing to the mitigation of WHCs. The data sent to the application can vary in format. It can be the full image with the detected object, a cropped image of the detected object, or simply data that includes the object’s name, number, and timestamp.

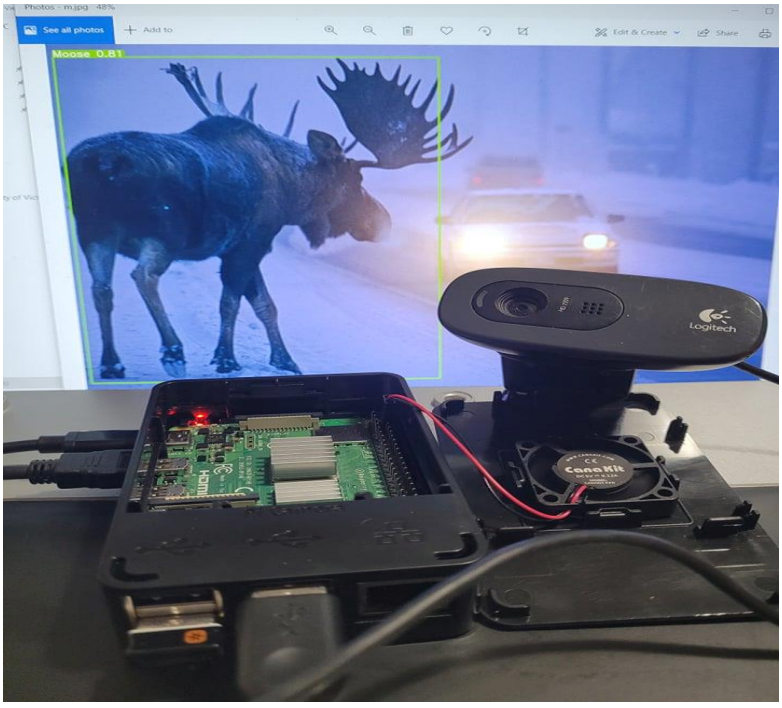


Figure 4.9: A mock-up using RP4B and a web camera to illustrate the capability of the proposed animal detection model.

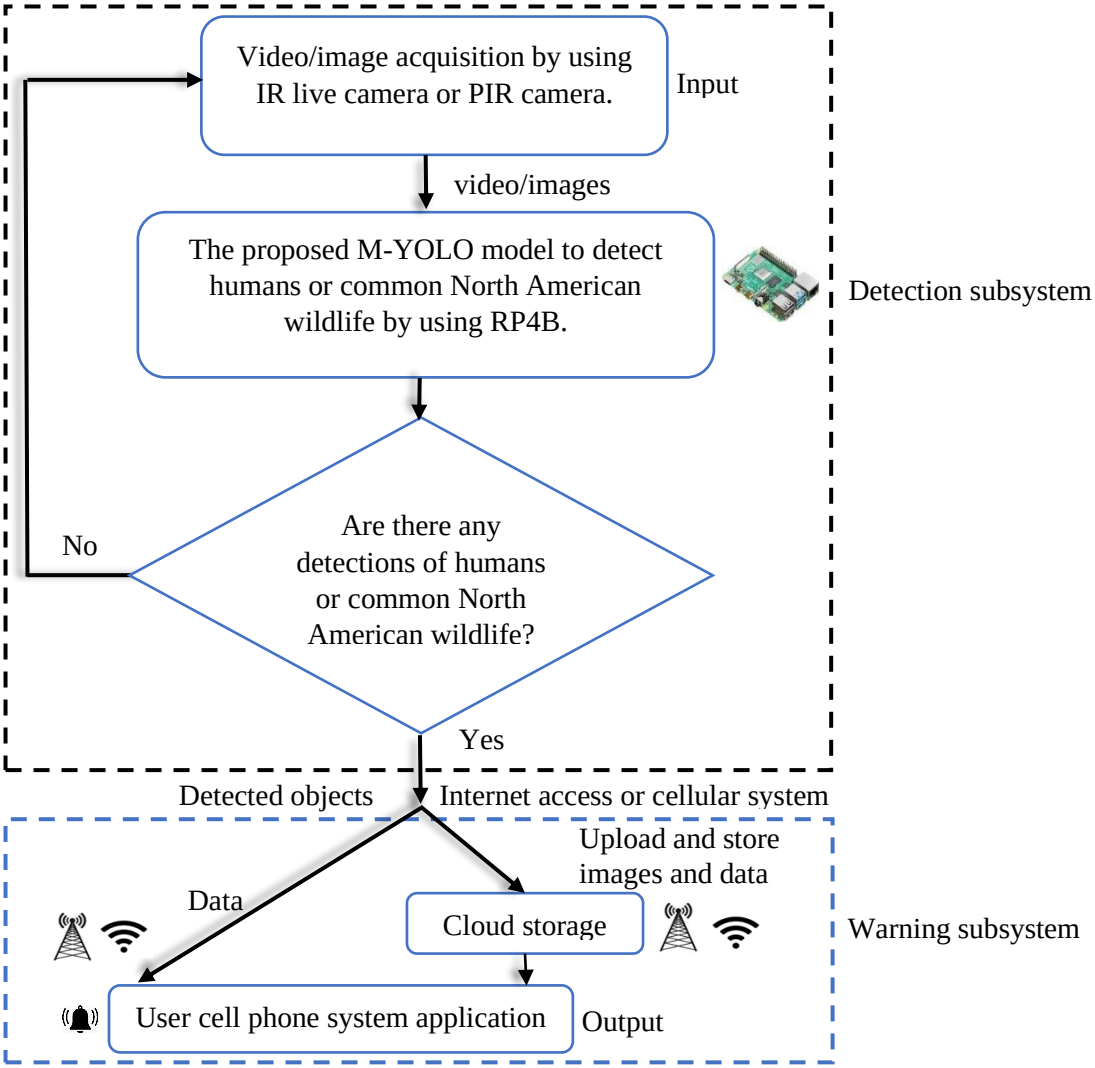


Figure 4.10: A block diagram of WHCs mitigation system.

Most commercial security cameras use PIR sensors to detect moving objects and send the captured images to the user via email, text message, or cellphone software application. However, these security cameras often generate a large number of false alarms. Occasionally, they are triggered to send unwanted empty images (i.e., images with no moving object within the camera's detection zone), due to factors such as weather conditions or movement of grass or tree branches. These false alarms consume battery life and storage capacity, reducing the camera's efficiency and diminishing user trust. To address these issues, our proposed M-YOLO model is integrated into WHCs mitigation system.

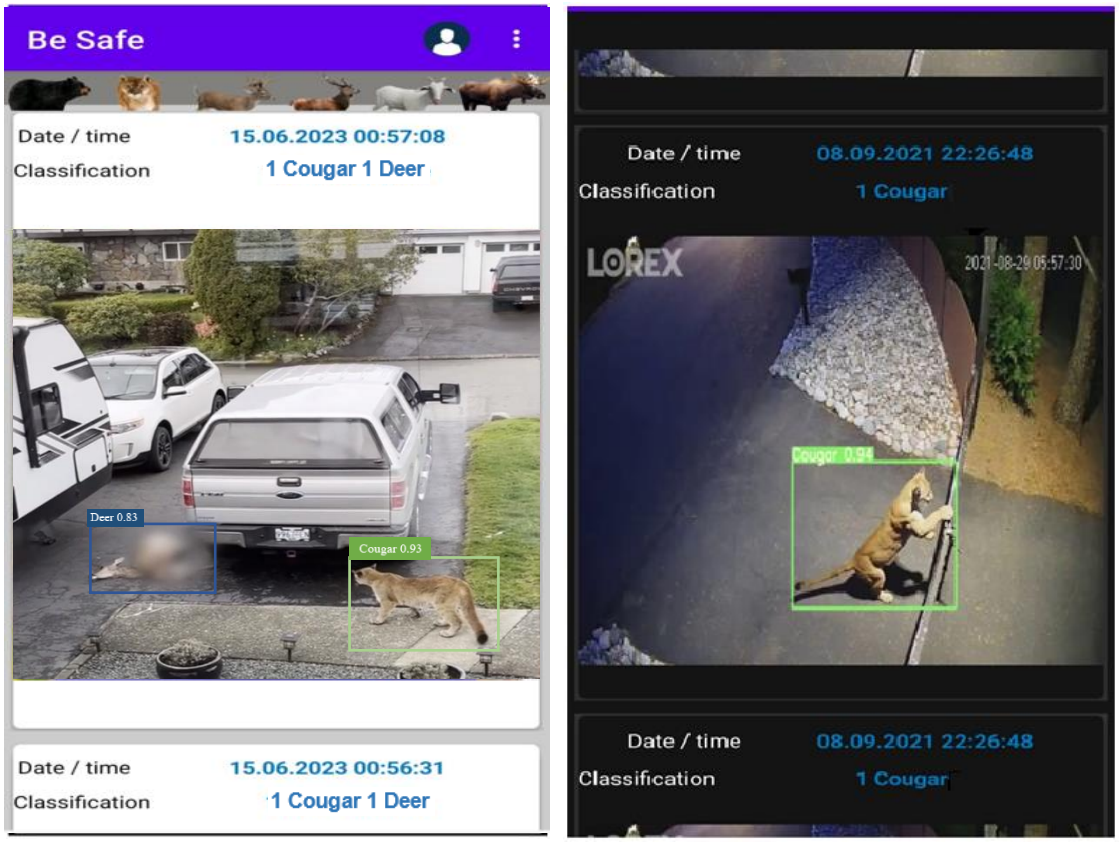


Figure 4.11: Two examples of notifications sent to the user's cell phone application. Two cougar images sourced from [24].

Our proposed novel WHCs mitigation system, has several advantages:

1. The system can accurately detect humans, and common North American/Canadian wildlife species (currently bear, cougar, deer, elk, moose, and mountain goat) in near real-time, under varying environmental conditions, and during both day and night. This capability applies to both PIR motion sensor cameras and live cameras.
2. The system is also able to identify species other than the above mentioned as animals and it can be upgraded to identify and localize additional animal species in the future.
3. Upon detecting an object, the system sends user-selectable alerts with images of the detected objects to the user's cellphone application, as shown in Fig. 4.11. The alert sent when detecting any animal will be different when detecting dangerous animals such as bears or cougars. The user can customize the nature and type of alerts received.

4. Users can select what they want to be notified about. They can choose to be alerted about any combination of humans, common North American wildlife species, or all of them, depending on their safety requirements.
5. The user can mount multiple cameras at different locations and our detection model can process the received images or videos from all of them in parallel and send notifications with images of the detected objects to the user's cellphone application, indicating the locations of the cameras.

Despite these advantages, there are further challenges to be addressed in order to enhance the efficiency of the WHCs mitigation system. These include increasing detection speed and reducing power consumption, as will be discussed in Chapter 5.

4.7. Conclusion

In this chapter, a modified YOLOv2 model (M-YOLO) is proposed that can be deployed on embedded devices to detect animal species with high performance as an initial step for WVCs and WHCs mitigation systems to reduce the negative impacts of these encounters. Our model was trained and tested on the BCMoTI dataset with three modifications to the original YOLOv2. First, to enhance the feature extraction ability, multi-level features merging was performed by adding low-level features, which improved YOLOv2's capability to identify animal species by 1.8%. Second, to reduce complexity and speed up the detection process without sacrificing the accuracy, the two repeated 3 × 3 convolutional layers with 1024 filters in the seventh block of the YOLOv2 architecture were removed. Third, to handle geometric transformations of animals in the images, three DCLs were added to the sixth block of YOLOv2, resulting in an increase of 3.0% in the animal species detection mAP. Our results show that the mAP of M-YOLO model was increased by 5.0% and 0.4% over YOLOv2 and YOLOv3, respectively. Moreover, the M-YOLO model can detect small, far, and occluded animals better than the regular YOLOv2 model. Compared with YOLOv4, the proposed M-YOLO model has faster detection speed which make it feasible to be deployed on embedded devices. To summarize, our proposed model is able to enhance detection speed without trading off accuracy in our animal species detection system.

Chapter 5

Motion-Selective Control Frames (MCF) for Improving Efficiency of M-YOLO Detector for Embedded Systems

For the safety of wildlife and humans, it is crucial that WVCs and WHCs mitigation systems detect wildlife in real-time. Therefore, the delay between when the wildlife first appears in the image and its detection should be enhanced, particularly on low power machines with limited computational capabilities, such as embedded systems. While most researchers propose lightweight object detectors or hardware acceleration techniques to reduce the detection delay and maximize throughput, our work takes a different approach. We aim to enhance the detection delay or processing speed by integrating two ideas into the M-YOLO model while maintaining the detection accuracy. The first is the Motion-Selective Control Frames (MCF) algorithm, which enhances the processing speed and reduce the power consumption of the object detection system by selectively processing only those frames where motion is detected. This selectivity reduces the amount of data that the M-YOLO model has to process, thus optimizing the overall system's efficiency. This is the focus of this chapter. The second idea is a parallel processing technique, which directly minimizes the processing delay of the M-YOLO model. This will be discussed in more detail in Chapter 6.

5.1 The Proposed MCF Algorithm

In real-time processing, especially when the detection model is deployed on an embedded device with limited computational capabilities, it is essential to focus computational resources only on frames that have motion. Continuously processing every captured frame can be both resource-intensive and power-draining. To address this challenge, the MCF algorithm is introduced.

The architecture of the object detection system (M-YOLO), as shown in Fig. 5.1, consists of: (i) the frame capture subsystem, which captures data from various sources, such as live camera feeds, video stream for real-time processing, or memory/video file for batch processing, and (ii) the detection model subsystem, which makes predictions through sequential tasks:

1. Pre-processing task: where the basic operations are applied to each captured frame, such as resizing and scaling, to ensure the frame size matches the CNN's requirements for a fixed input size.
2. CNN feature extraction task: where convolution and pooling operations are applied to extract and process significant features from the preprocessed frame to make predictions regarding identification probabilities and bounding box coordinates, as explained in detail in Section 4.3.
3. Post-processing task: where NMS algorithm is applied to eliminate overlapping bounding boxes and display the final detection results, as explained before in Section 2.1.2.

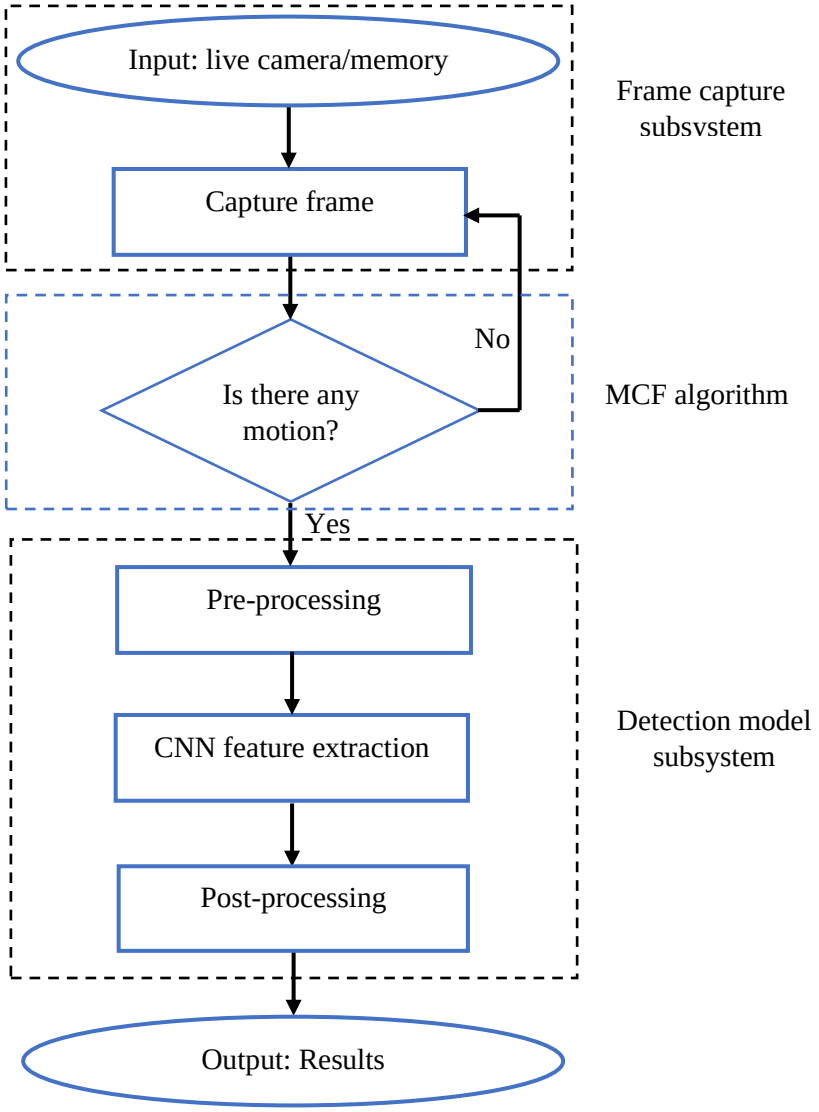


Figure 5.1: Sequential tasks of the object detection system, incorporating the frame capture subsystem and detection model subsystem (black dashed boxes), supplemented by the integration of the MCF algorithm (blue dashed box).

Following the frame capture subsystem, the MCF algorithm is proposed and integrated into the M-YOLO animal species detector, as shown in Fig. 5.1. This integration aims to minimize the animal detection processing delay and decrease both computational complexity and power consumption, making it suitable for embedded devices with restricted memory and computational power. This algorithm exploits the fact that not all captured frames have unique and crucial information, thus M-YOLO detector does not need to process all frames.

The main goals of the proposed MCF algorithm are:

1. Identifying the frames which contain animal motion.
2. Extracting those frames and ignoring or dropping all the others.
3. Passing those frames to M-YOLO for animal identification and localization.

5.2 Absolute Subtraction Technique

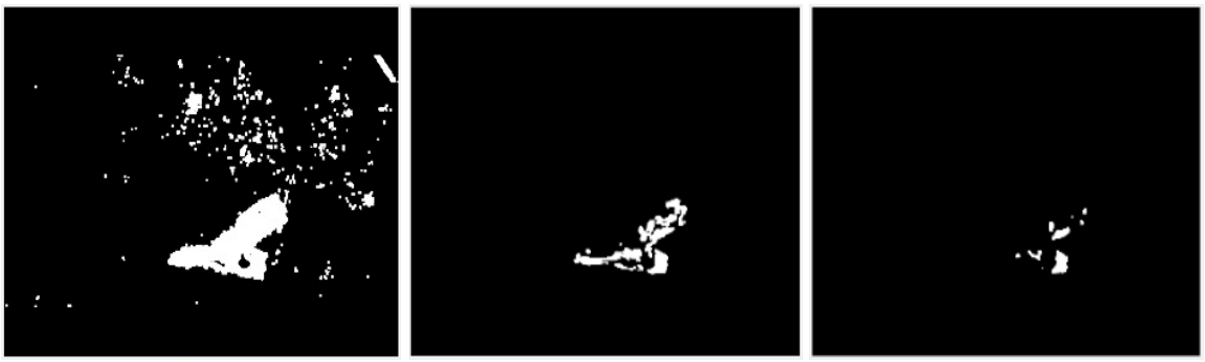
MCF is a pixel-based algorithm that relied on the absolute subtraction technique which compares the pixel values or intensities between two input frames from live camera or video sequence. The frame differences will decide if there is any moved object in the frame or not. After trial-and-error processing, four consecutive frames from the input are considered at a time and the first frame of each is assumed as a reference frame $F(x, y)$. This strategy of selecting four consecutive frames can be changed depending on the specific requirements of the application, laying the foundation for further enhancement. Equation (5.1) represents the absolute subtraction of pixel values $\Delta_i(x, y)$ on the frame $F_i(x, y)$ with the reference frame $F(x, y)$ where i varies from 2 to 4.

$$\Delta_i(x, y) = \left| F(x, y) - F_i(x, y) \right| \tag{5.1}$$

where $\Delta_i(x, y)$ determine if the frame $F_i(x, y)$ has a unique information compared to the reference frame $F(x, y)$. If $\Delta_i(x, y)$ differ from zero, this is an indication of moving objects, while when $\Delta_i(x, y)$ *is* equal to zero this means stationary objects. However, in reality it is impossible that $\Delta_i(x, y)$ will be equal to zero due to the presence of noise. In order to decide if these obtained non-zero values are caused by noise or motion, $\Delta_i(x, y)$ is compared with a predefined threshold (T), as shown in Equation (5.2).

$$m_i(x, y) = \begin{cases} \Delta i\ (x, y) & if\ \Delta i\ (x, y) \geq T \\ 0 & if\ \Delta i\ (x, y) < T \end{cases} \quad (5.2)$$

The selection of threshold value is important. The low threshold value can result in false motion detection because of the presence of noise, where the noisy pixels can be detected as motion pixels as shown in Fig. 5.2(a). The high threshold value can lead to missing motion detection as the small changes in movement will be ignored as shown in Fig. 5.2(c). In our work the significant value of the threshold is 55 after several trial-and-error processing as shown in Fig. 5.2(b). The chosen threshold ensures that the system is sensitive enough to detect meaningful motion while being robust against noise and minor variations, which is crucial for the reliability and practicality of the motion detection algorithm in varied environments.



(a) (b) (c)

Figure 5.2: The visualization comparison after applying different value of threshold: (a) at low threshold, (b) at threshold = 55, and (c) at high threshold.

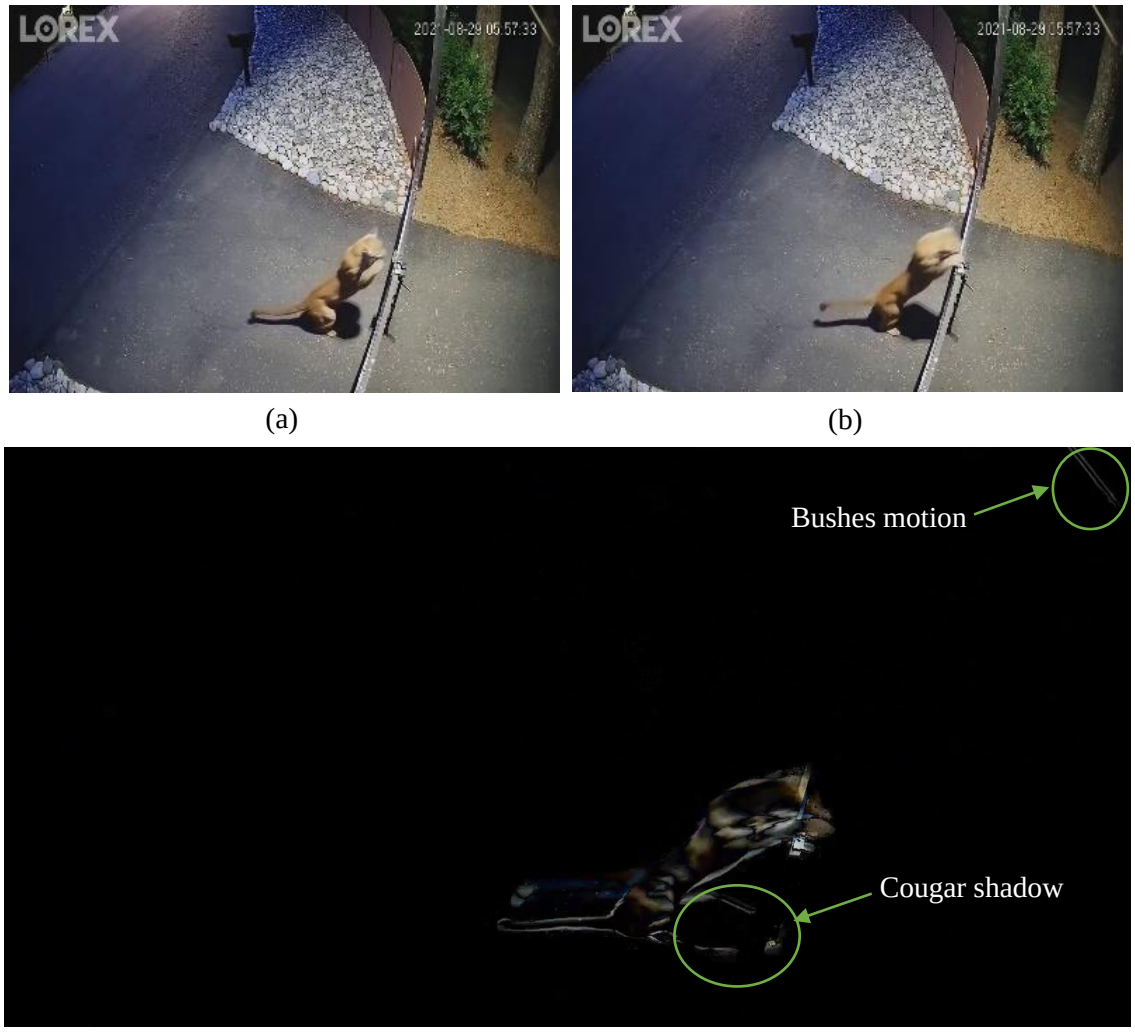
By taking advantage of this simple technique, only frames that have a unique information or animal motion are processed by M-YOLO detector, leading to decrease in processing time and power consumption.

5.2.1 Absolute Subtraction Technique Challenges

There are some challenges that can affect the pixel values, and hence subsequentially having a negative impact on identifying any changes in the frame as motion within a real system. Most of these challenges come from the environmental conditions within the scene, with fewer attributed to the used camera such as:

1. Frames noise which produced during the acquisition [228].
2. Illumination or light intensity variation of the frame's background.

3. Dynamic movement of static objects in the background like wind blowing to bushes and trees, as shown in Fig. 5.3(c), obtained by applying the absolute subtraction technique between the first frame in Fig. 5.3(a) and the third frame in Fig. 5.3(b).
4. Shadows of moving objects that can be detected as another moving object, as shown in Fig. 5.3(c).



(a) (b)

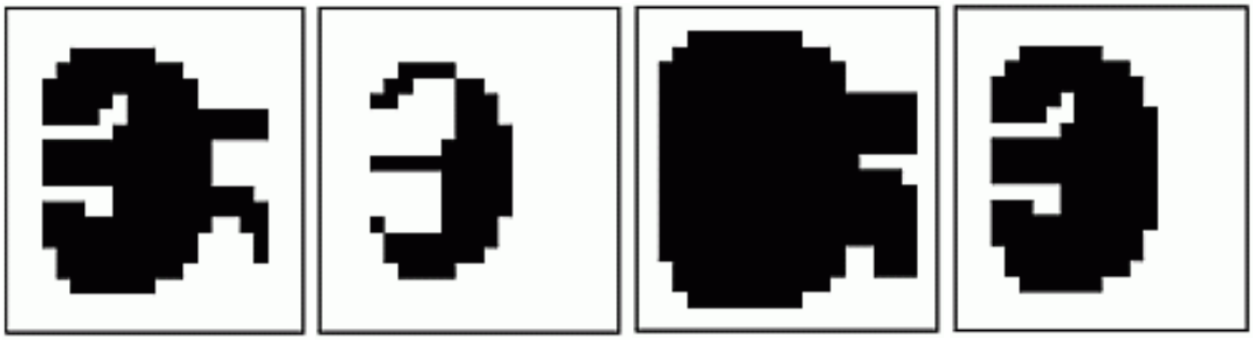
(c)

Figure 5.3: The absolute subtraction technique between: (a) the first frame "Reference frame", and (b) the third frame of the four consecutive frames. (c) The resulting output of the absolute subtraction between the two frames with bushes motion and cougar shadow.

5.2.2 Improvements to the Absolute Subtraction Technique

To overcome the effects of the above factors, the absolute subtraction technique can be improved by adding a morphological opening operation, which consists of an erosion operation followed by

a dilation operation [229]. As shown in Fig. 5.4, the erosion operation shrinks the area of detected objects or remove pixels from the object boundaries, otherwise the dilation operation expands the area of detected objects or add pixels to the object boundaries [230]. Therefore, the main objective of applying the opening operation is to smoothen contours, remove any isolated bright pixels, and break slight connections between objects in the output image of absolute subtraction technique. This process helps maintain the shape and size of large objects, as each pixel in the image is adjusted based on the value of its neighboring pixels.



(a) (b) (c) (d)

Figure 5.4: Morphological operations. (a) Original binary image, (b) output image after applying erosion operator, (c) output image after applying dilation operator, and (d) output image after applying opening operator [230].

After applying opening operation, the effect of noise is decreased; however, there is still noise in the frame that is always detected as a motion. Gaussian filter [231] is widely used by many researchers to reduce undesired fluctuation (high frequency components) in the video stream sequences which are caused by surrounding environment such as low illumination and high temperature. Due to its simplicity and capability of smoothing and softening video stream sequences [232], Gaussian filter is utilized in our work for noise reduction by applying image convolution technique with a Gaussian kernel size 5×5.

The MCF algorithm can be simplified into few steps as shown in Algorithm 5.1 and as shown in Fig. 5.5.

Algorithm 5.1 MCF Algorithm.

Input: Consecutive i frames from the live camera or video sequence: $F_i(x, y)$, and T.

Output: Decision of moving animal.

Set: Reference frame = $F_1(x, y)$;

Pass: $F_1(x, y)$ to detection model subsystem;

For $i \leftarrow 2$ **to** 4 **do**

    $\Delta_i(x, y) = |F_1(x, y) - F_i(x, y)|$;

    If $\Delta_i(x, y) \geq T$ **then**

        $m_i(x, y) = \Delta_i(x, y)$;

    Else

        $m_i(x, y) = 0$;

    End if

End for

Convert: Convert subtracted RGB frames to grayscale frames.

Morphological opening operation: Apply opening operation on the grayscale frames to remove any small objects, single pixels, or slight connections between regions.

Filtering: Apply Gaussian filter to the grayscale images.

Decision: Pass original frame to detection model subsystem if motion is detected.

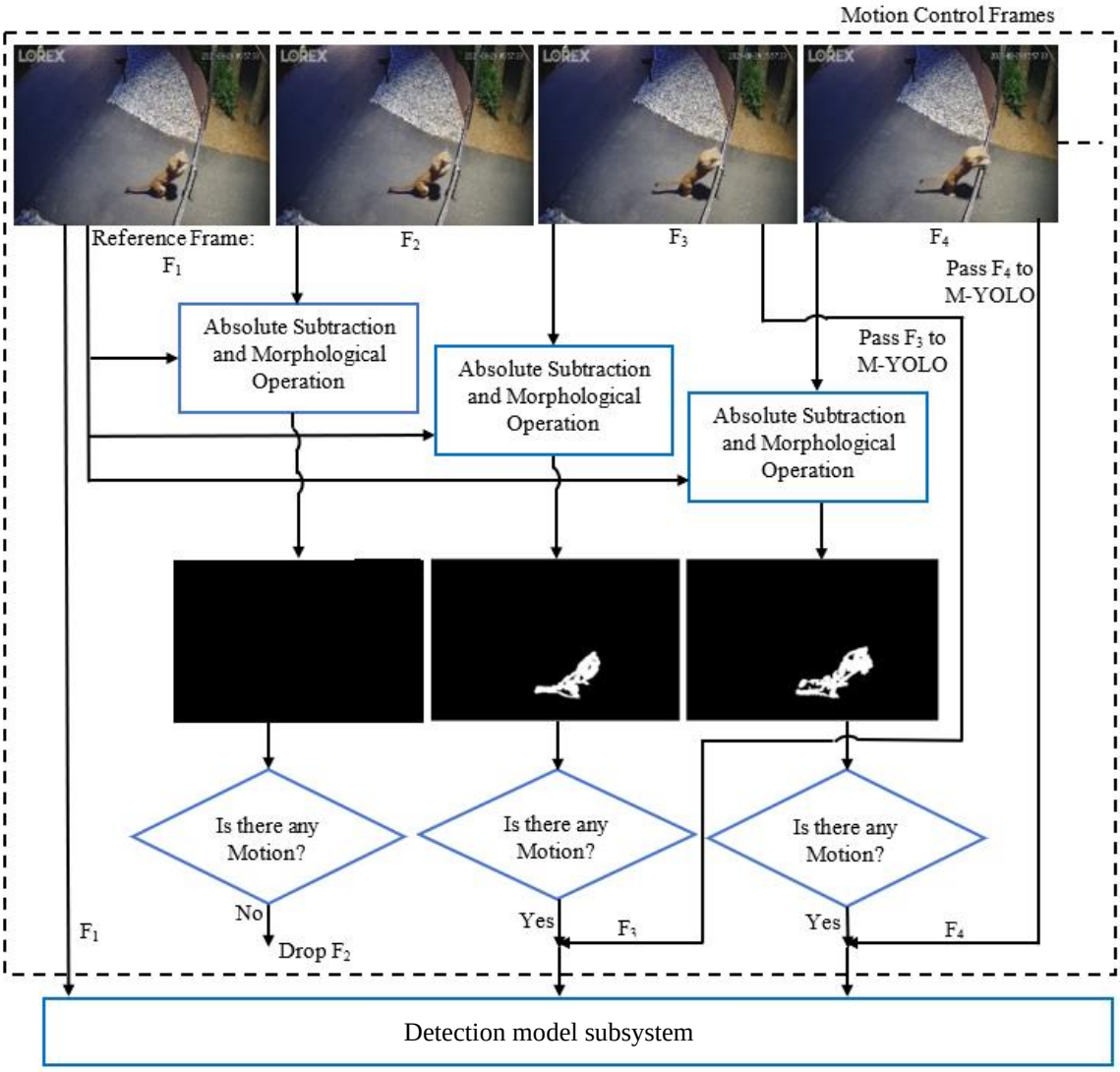


Figure 5.5: The proposed M-YOLO detector with the MCF algorithm (MCF-YOLO).

5.3 Experimental Results: MCF Algorithm Integration into M-YOLO Detector

M-YOLO animal species detector is evaluated before and after integrating the proposed MCF algorithm to prove the efficiency and feasibility of the integrated algorithm using the same workstation as mentioned before in Section 4.4.3. In this context, efficiency refers to the reduction of the processing delay and power consumption. Four fundamental evaluation metrics are used for the evaluation: Elapsed time, CPU time, Frame Per Second (FPS), and CPU utilization.

Elapsed time is the wall clock time which measures the total runtime from the moment that the program is launched to the moment it finishes running. Elapsed time is defined as:

$$\text{Elapsed time} = \text{End time} - \text{Start time} \tag{5.3}$$

CPU time is the total amount of time for which the CPU is busy executing the program.

FPS is used to judge how fast the object detector model in processing the input, whether it is live stream or video file, to produce the desired output every unit time. The FPS can be calculated through the following equation:

$$\text{FPS} = \text{NPF} / (\text{Current time} - \text{Start time}) \tag{5.4}$$

where NPF represents the Number of Processed Frames in a specified time during the execution of the detector.

CPU utilization measures the proportion of active CPU cycles utilized by a running program during a specific time interval, thus estimating its power consumption [233]. For example, a CPU utilization of 100% means that all CPU cycles are dedicated to computations of the program throughout its entire runtime, with no idle time. High CPU utilization for executing a program indicates a high demand for processing power. CPU utilization is defined as:

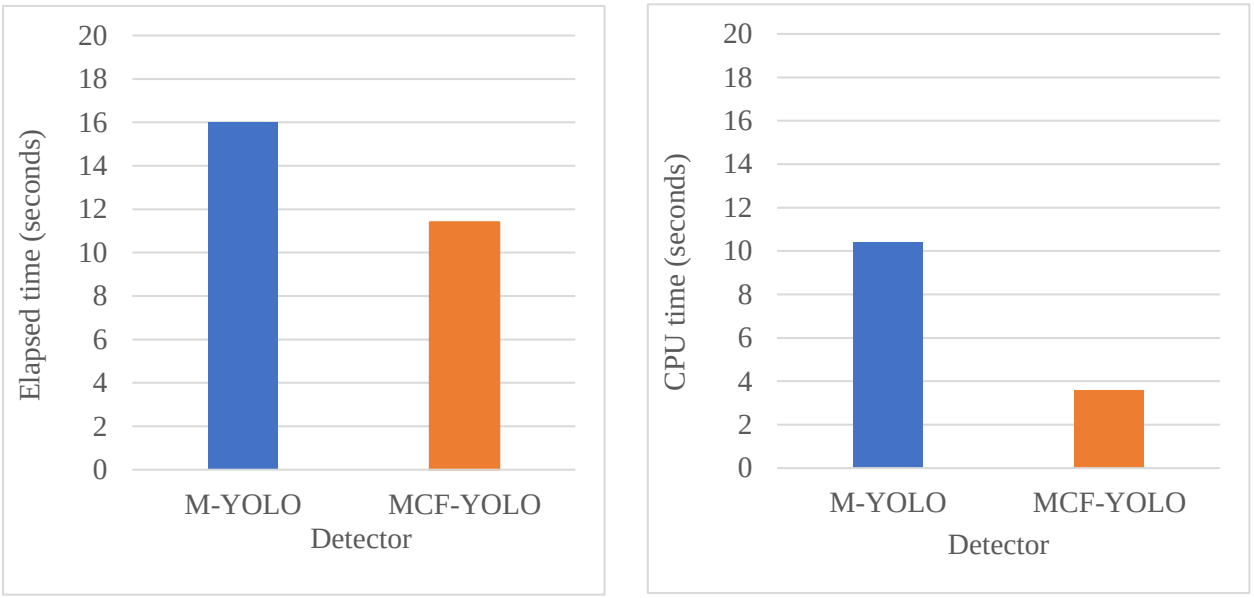
$$\%\text{CPU utilization} = (\text{Active CPU cycles} / \text{Total CPU cycles}) \times 100\% \tag{5.5}$$

5.3.1 Processing Speed Comparisons: Batch and Real-Time Processing

The evaluation metrics: Elapsed time, CPU time, and FPS, are applied to measure and compare the processing speed of the M-YOLO and MCF-YOLO (M-YOLO with MCF algorithm) detectors. These metrics are calculated by using Python's "time" module/library, with "time.time ()" function for measuring Elapsed time and FPS, while with "time.process_time ()" function for CPU time. The evaluation is performed using two types of inputs: (i) batch processing, which involves the analysis of recorded and stored videos in a non-real-time manner, allowing for the processing of multiple frames in a sequential manner, and (ii) real-time processing, which includes the analysis of streaming videos from the internet or capturing video in real-time from a web camera, requiring immediate processing and response to each frame as it becomes available.

For batch processing, the results in Fig. 5.6 show the processing speed comparison between the M-YOLO and MCF-YOLO detectors on a 727-frame video from [25]. According to the evaluation metrics: Elapsed time as in Fig. 5.6(a) and CPU time as in Fig. 5.6(b), the experimental

results prove that the integration of MCF algorithm to M-YOLO detector speed up the detection process by 28.7% and 65.3%, respectively.



(a) (b)

Figure 5.6: Processing speed evaluation of M-YOLO and MCF-YOLO models for batch processing sourced from [25] in terms of: (a) Elapsed time, and (b) CPU time.

For real-time processing, the system manages a queue to buffer captured frames from a live camera feed to smooth out any variations between the frame rate of the camera and the processing speed of the detection system. This queue is implemented using Python's queue module. The web camera being used in our work operates at a frame rate of 30 FPS and the detection processing time is 0.022 second per frame for single animal detection, as evaluated and reported in Table 4.3. In this case, the processing time per frame is shorter than the time interval between frames based on the camera's frame rate, therefore the queue used to buffer the captured frames is typically empty, as the detection system is ready for the next frame before it arrives. On the other hand, if the processing time per frame is consistently longer than the time interval between frames, the queue size is initialized to hold 4 frames. This capacity was determined through trial-and-error testing, taking into account factors such as reducing the queueing delay/waiting time, the amount of time a frame spends in the queue waiting to be processed, and memory limitations of the embedded devices. The frame capture and queue management algorithm can be simplified into few steps, as shown in Algorithm 5.2.

Algorithm 5.2 Frame Capture and Queue Management Algorithm.

Input: *u*: A unique identifier for the live camera device we are capturing frames from;

Q: The queue used for holding the frames captured from the live camera;

S: The size of the queue;

Output: Stream of frames.

1 **Set**: *u* to the live camera device.

2 **Set**: $S = 4$;

3 **For** $i \leftarrow 0$ **to** *S* **do**

4 **Enqueue**: Call *u* to capture a frame and enqueue it into *Q*;

10 **End for**

11 **While** True **do**

12 **Wait**: Call *u* to wait until *Q* is ready for dequeuing a frame;

13 **Send**: Dequeue a frame from the front of *Q* and send it for processing;

14 **Requeue**: Call *u* to capture a new frame and enqueue it to the end of *Q*;

15 **End While**

When the queue reaches capacity, two strategies can be considered:

1. Dropping frames: This strategy involves dropping incoming frames when the queue is full. This is suitable when real-time detection is critical, as in our system which is designed to alert drivers or hikers to the presence of wildlife on roads or trails ahead. In these scenarios, it is important to process the most recent frames to provide real-time alerts. This strategy could miss some animals, especially fast-moving ones, but it ensures the system operates as close to real-time as possible.
2. Reducing frame rate: This strategy involves decreasing the frame rate of the camera to prevent queue overflow. This is suitable for systems where capturing every animal sighting is more important than real-time processing, such as in wildlife research applications. However, this strategy is not recommended for video feeds in low-light conditions, where reducing the frame rate may cause motion blur for moving objects, affecting the detection accuracy.

In our work, animals typically move slowly or remain stationary unless they are scared or in pursuit. Therefore, dropping frames strategy has a minimal impact on the detection accuracy.

In the context of real-time processing, where there is no time limit, Elapsed time and CPU time are not suitable for evaluating processing speed; therefore, the FPS evaluation metric is used instead. The results in Fig. 5.7 show that the MCF-YOLO detector achieves a 39.2% higher processing speed compared to the M-YOLO detector. It is essential to note that this percentage can vary based on the number of detected objects per frame and the level of activity in the surrounding environment.

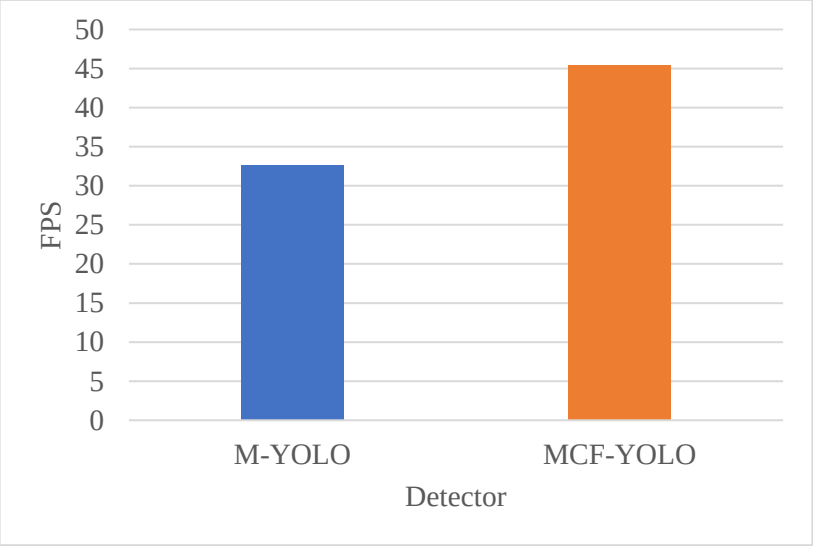


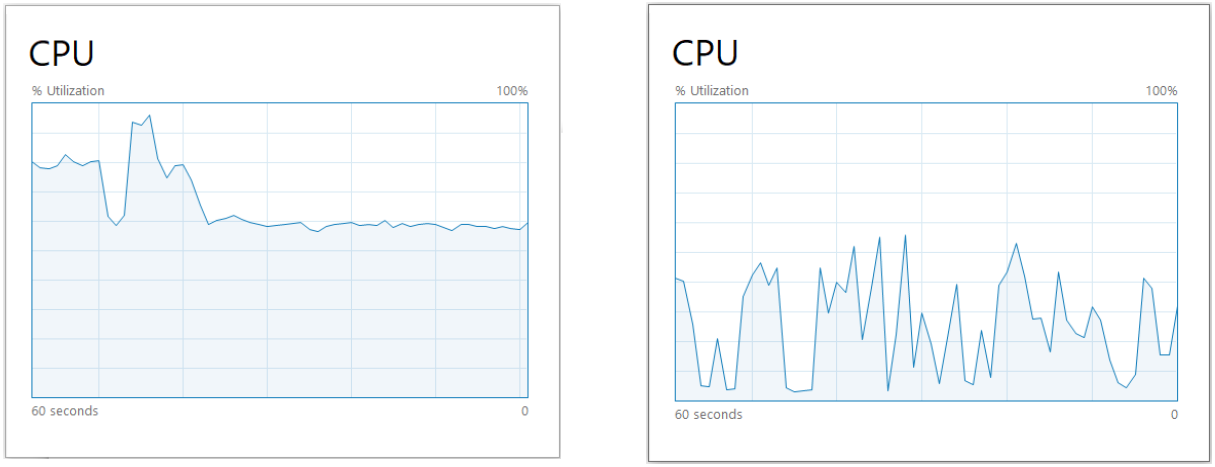
Figure 5.7: Processing speed evaluation of M-YOLO and MCF-YOLO models for real-time processing in terms of FPS.

5.3.2 Power Consumption Comparisons: Batch and Real-Time Processing

The CPU utilization metric is applied to compare how much power is consumed by the CPU to process the M-YOLO and MCF-YOLO detectors in both batch and real-time processing scenarios. To measure this metric, Python library “psutil” is used. The results of the CPU utilization are presented in Figures 5.8, and 5.9. In these figures, the vertical axis represents the percentage of CPU utilization, ranging from 0% to 100%. A value of 0% would indicate no CPU usage, while 100% would mean that the CPU is operating at full capacity. The figures show the variation of CPU utilization over a 60 second time duration, providing insights into how the CPU handles the workload of processing two different input sources with our models.

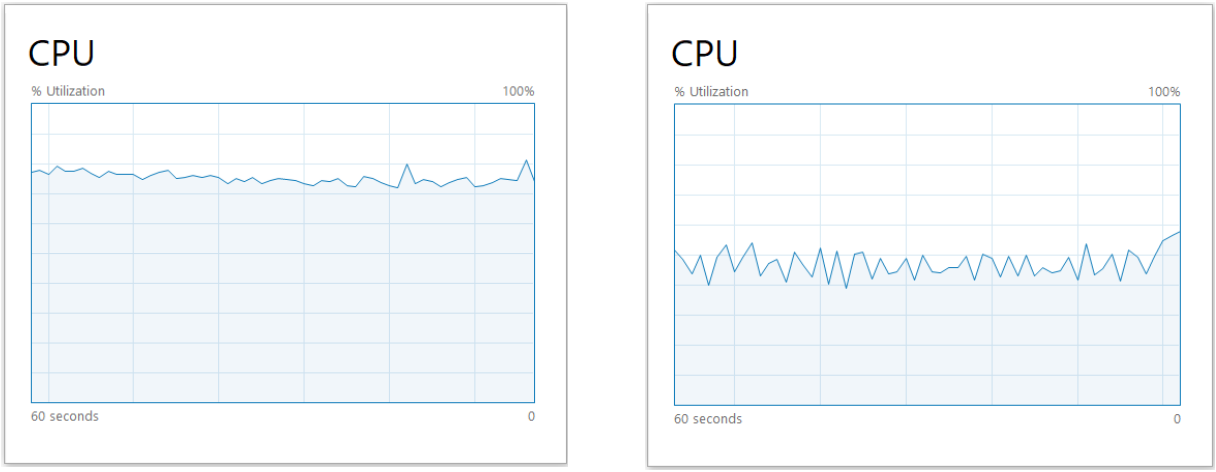
For batch processing, as shown in Fig. 5.8, the CPU is active only when there is work assigned to it. Therefore, the integration of the MCF algorithm into the M-YOLO model, as shown in Fig. 5.8(b), significantly reduces power consumption compared to the original M-YOLO model. This reduction in power consumption is due to the CPU having many idle states during intervals of dropped unnecessary frames when no animal motion is detected. These results indicate that the MCF-YOLO model efficiently utilizes CPU resources, leading to power savings.

For real-time processing, as shown in Fig. 5.9, the CPU is responsive and active all times. However, with the integration of the MCF algorithm, as shown in Fig. 5.9(b), the CPU does not require full performance as in Fig. 5.9(a), which affects the power consumption. The MCF-YOLO model, as shown in Fig. 5.9(b), demonstrates lower CPU utilization compared to the M-YOLO model, indicating more efficient power utilization during real-time processing.



(a) (b)

Figure 5.8: CPU utilization percentage comparison for batch processing sourced from [25] of: (a) M-YOLO, and (b) MCF-YOLO models.



(a) (b)

Figure 5.9: CPU utilization percentage comparison for real-time processing from a web camera of: (a) M-YOLO, and (b) MCF-YOLO models.

The above results prove the robustness and the simplicity of the proposed MCF algorithm. The integration of this algorithm into the M-YOLO detector (MCF-YOLO) achieves faster processing time compared to M-YOLO and reduces power consumption and unnecessary computations while maintaining detection accuracy during both batch and real-time processing. These advantages make the MCF-YOLO detector suitable for deployment on embedded devices such as RP4B to build WHCs and WVCs mitigation systems that rely on live cameras. For these systems, the MCF algorithm operating until motion is detected and M-YOLO identifies this as an animal. At this point, the MCF algorithm pauses temporarily, allowing the M-YOLO detector to process subsequent frames. The MCF algorithm resumes its operation only when M-YOLO no longer detects any animal.

5.4 Conclusion

Through our evaluation, we have illustrated the advantages of integrating the MCF algorithm into the M-YOLO detector. This MCF-YOLO detector achieves faster detection speed for both batch processing and real-time processing. Additionally, it shows enhanced power consumption. Notably, the real-time processing speed is influenced by the frame rate of the utilized live camera, the number of detected objects per frame, and the level of activity in the surrounding environment. These improvements are critical for the successful deployment of wildlife mitigation systems on low-power embedded devices. Overall, our results highlight the robustness and practicality of the MCF algorithm, positioning it as a valuable enhancement to the M-YOLO detector for efficient animal species detection in real-world scenarios.

However, although the successful integration of the MCF algorithm and the promising results it has yielded, there may be still a need for further improvements for several reasons:

- **Real-time applications:** While achieving a fast detection speed, certain real-time scenarios, such as dynamic environments (different lighting conditions, weather patterns or seasons) or moving animals, may require even higher detection speed to accurately identify, localize, and count them. Enhancements in detection speed is crucial to maintain accurate and up-to-date detections in such scenarios.
- **Scalability:** MCF-YOLO performs well at its current FPS, but as the number of animals increases, the processing speed may decrease. This is because MCF-YOLO needs to process and analyze more regions of interest when there are more objects in a frame. Each

detected object requires the evaluation of multiple bounding box predictions, calculation of confidence scores, and the application of NMS to filter redundant detections. To address these challenges, further improvements can be implemented to maintain or improve the processing speed of the detection system.

- **Hardware limitations:** The achieved FPS depends on the hardware configuration/setup and available computational resources. Thus, it is essential to investigate additional optimizations that increase the detection speed without requiring significant hardware upgrades. This can allow the MCF-YOLO detector to perform efficiently on a wide range of hardware setups, including those with limited computational capabilities.

Considering these factors, further research is needed to address these areas of improvement. By enhancing the MCF-YOLO detector, as will be shown in Chapter 7, we can strive to meet the demands of real-time applications, improve scalability, and accommodate various hardware configurations. These efforts will contribute to the continued advancement and applicability of wildlife mitigation systems in diverse real-world scenarios.

Chapter 6

Parallel Processing Implementation for Reducing Processing Delay of M-YOLO Detector

This chapter focuses on reducing the processing delay in the detection model subsystem of the proposed M-YOLO detector, which is caused by its sequential implementation on a single core. To address this challenge, the concept of parallel processing is introduced as a tool to enhance the processing speed using pipeline parallelism implementation. The performance of this approach is evaluated in terms of processing speed for both batch and real-time processing scenarios.

6.1 Sequential Processing Implementation of M-YOLO Detector

The detection model subsystem, as shown in Fig. 6.1, consists of three sequential tasks: pre-processing, CNN feature extraction, and post-processing, as explained in Section 5.1. Each of these tasks is implemented as a distinct function coded in Python and is executed one after the other on a single core.

The detection model subsystem is based on CNNs algorithm. The architecture of CNNs is a multi-layered feed forward neural network stacked on top of each other in a sequential design to learn hierarchical attributes. Each layer takes input from the previous layer and produces output to the subsequent layer, as shown in Fig. 2.2 in Section 2.1.1.2.

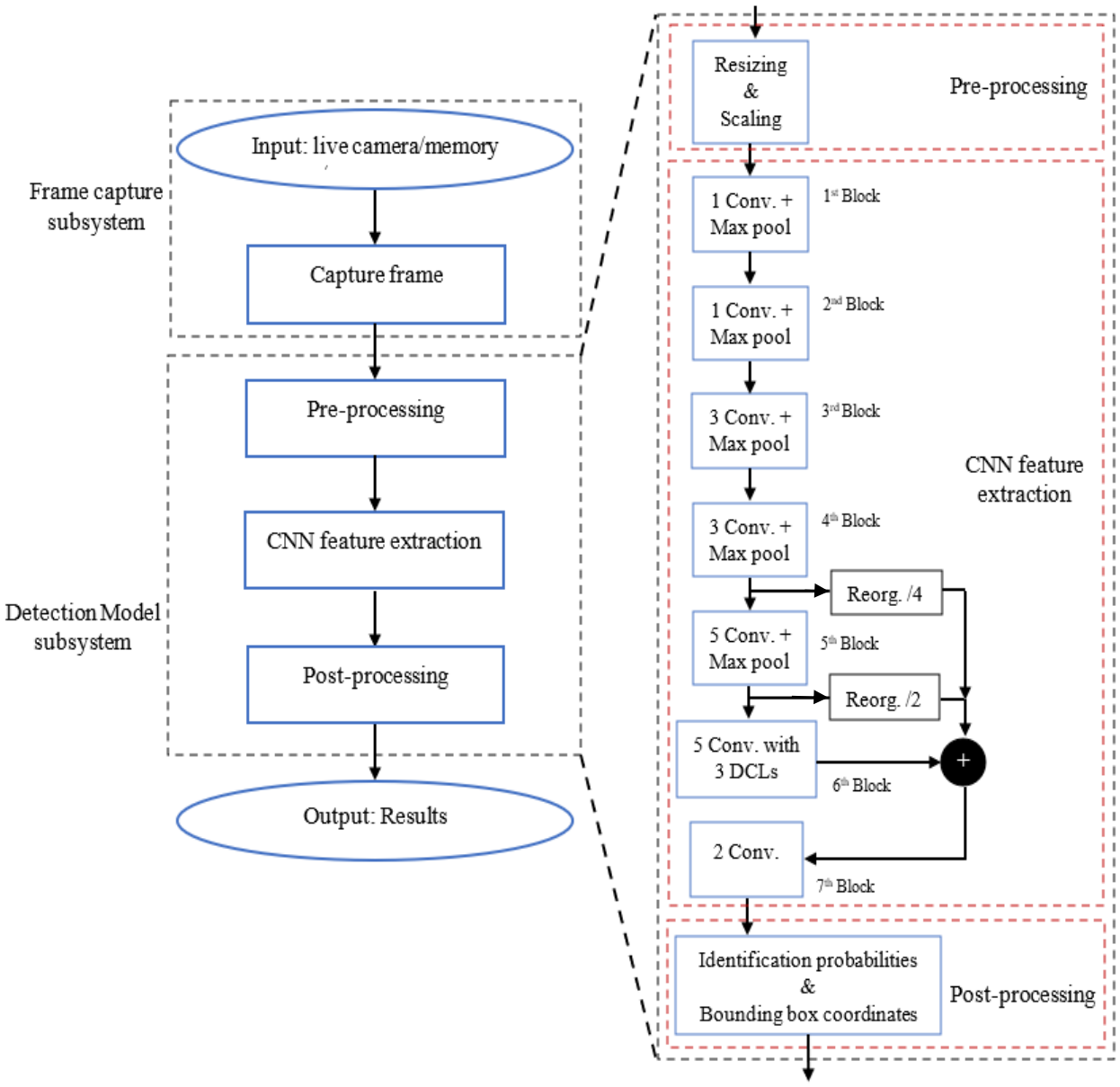


Figure 6.1: Sequential processing flow of the detection model subsystem.

Measuring the processing time of each task in the detection model subsystem is crucial for reducing its processing delay. Python's "time" module/library with "time.time ()" function is used to measure this time on a Core i7 system, as specified in Section 4.4.3. The processing times of pre-processing (T_1), CNN feature extraction (T_2), and post-processing (T_3) tasks take 1.67 ms, 20.04 ms, and 0.28 ms, respectively. Therefore, the total sequential detection processing time (T_{sp}) for each frame can be calculated as:

$$T_{sp} = T_1 + T_2 + T_3 \tag{6.1}$$

Achieving a detection processing time of 21.99 ms per frame is impressive and demonstrates enhanced detection speed. Considering the requirements of our wildlife mitigation system, especially in real-time scenarios with multiple presence of wildlife, as explained in Chapter 5, faster processing speed is crucial. It ensures timely and accurate detection, allowing the system to respond to sudden wildlife activity and facilitate proactive mitigation. Moreover, this system is intended for deployment on a low-power embedded device. As discussed in Section 4.5, the processing time on the RP4B is 160 ms per frame for a single animal detection. This highlights the need for further improvements to balance detection speed and power constraints within the deployment hardware. Therefore, a parallel processing technique is proposed for the M-YOLO detector to distribute the computational workload across multiple processors or cores, enabling faster processing of input frames and increased throughput.

6.2 Implementation of Parallel Multicore processing for Detection Model Using Python

In the case of parallelizing the three tasks of the detection model subsystem, utilizing multicore processing can be a cost-effective choice, as it only requires standard hardware to assign processes to separate cores and does not require any specialized hardware. This allows for simultaneous execution of these processes, which can improve the detection speed of the algorithm. However, it is important to select the appropriate communication mechanism between the cores to ensure consistency of the data shared between them.

Communication Mechanisms

In Python, the "multiprocessing" module/ library [234] provides a way to write parallel code, enabling efficient use of the available cores in a system. This module provides a number of functions for managing processes and implementing inter-process communication (IPC) primitives using lock and unlock operations for sending and receiving data between processes through mechanisms such as [151]: pipes, queues, and shared memory. These IPC mechanisms allow different processes on different processors or cores to communicate and exchange information with each other to accomplish their tasks.

As shown in Fig. 6.2(a), pipes allow for unidirectional communication between two related processes (i.e., parent process and its child process). With a pipe, one process sends data by writing to one end of the pipe (write-end), while the other process receives this data by reading from the

opposite end (read-end). As shown in Fig. 6.2(b), queues enable communication between two unrelated processes to put and get data. Queues provide additional synchronization and locking mechanism to ensure that only one process accesses the queue at a time in a synchronized and orderly manner, by using First-In-First-Out (FIFO) principle. As shown in Fig. 6.2(c), shared memory allows two or more processes to read and write to a common memory space. This shared memory acts as a shared buffer for exchanging data between these processes, making it useful when large amounts of data need frequent access and updates by multiple processes. However, synchronization mechanisms are required to avoid race conditions or data inconsistencies.

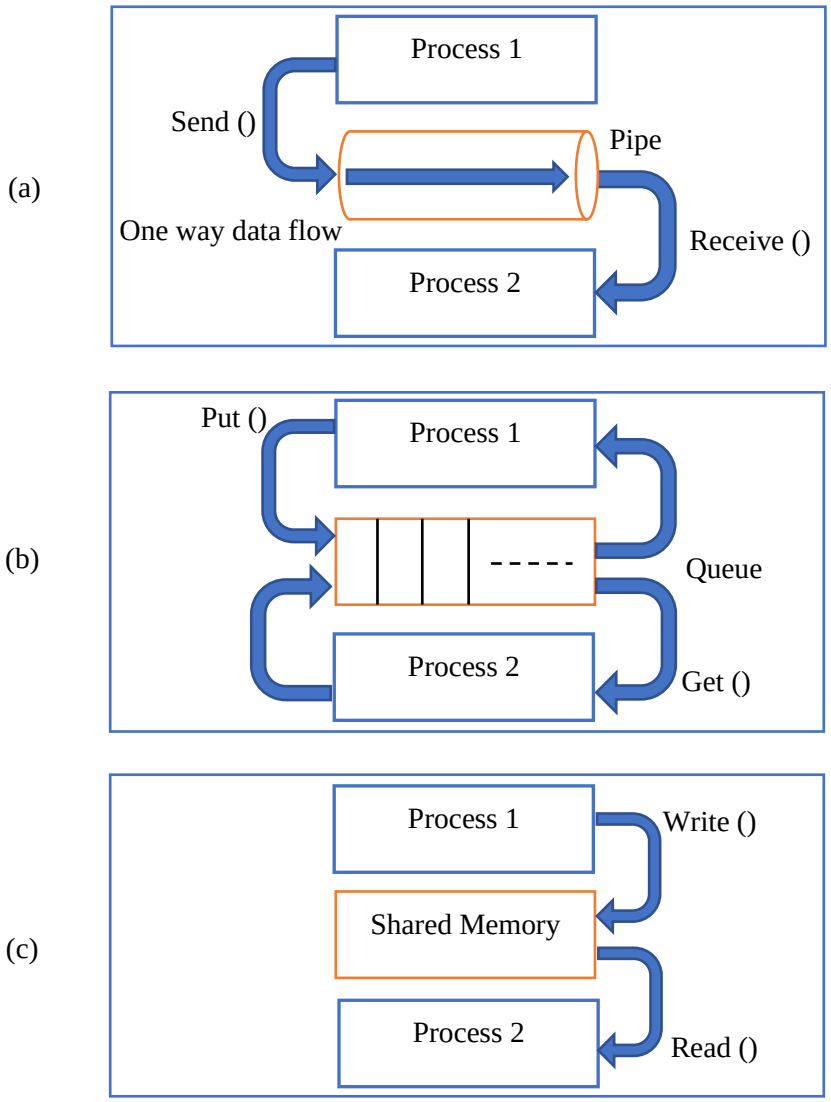


Figure 6.2: IPC mechanisms between two processes: (a) pipe which enables direct communication between two processes with one acting as a sender and the other as a receiver, (b) queue which allows synchronized access to data in a FIFO ordering, and (c) shared memory which provides a shared block of memory for multiple processes to exchange information.

The choice of which IPC mechanism to use depends on the specific requirements of the application such as:

- Latency: The time delay experienced in transferring data between two processes.
- Synchronization: The coordination of data access when multiple processes require concurrent and orderly interaction with shared resources.

For example, if the primary concern is the efficient transfer of data, especially in scenarios requiring real-time processing or low latency, shared memory may be a suitable choice compared to pipes and queues. This is because there is no data copying involved; once data is written to the shared memory, another process can directly access it without the need for data to be copied or transferred. In contrast, pipes and queues can have higher latency due to the overhead of copying and transferring data between processes. On the other hand, if the data needs to be accessed in a synchronized and ordered way by multiple processes, queues can be used as they provide synchronized access to data with FIFO ordering, or shared memory with proper synchronization mechanisms/primitives [151] [234] [235], such as semaphore, mutex (Mutual Exclusion), barrier, or condition variable.

Synchronization Primitives

Synchronization primitives are used to prevent interference between processes. They ensure that data is neither overwritten nor read before it is ready, thereby allowing for the safe and efficient sharing of data among the different processes.

Semaphore is an integer value, that is initialized with the number of available shared resources in the system. It works like a guard or lock on these resources. In Python, processes use two functions: “acquire ()” and “release ()”, to decrease and increase the semaphore value, respectively. If the semaphore value is greater than zero, the process acquires the semaphore and access a shared resource. If the semaphore value is zero, indicating no available resource, the process is blocked until another process completes its processing the shared resource and releases the semaphore.

Mutex is a binary semaphore, also known as a lock, that allows only one process to access a shared resource at a time. When a process wants to access a shared resource, it must “lock ()” the mutex before accessing the resource and “unlock ()” it after finishing. If the mutex is currently

held by another process, the calling process is blocked and put to sleep until the mutex becomes available.

Barrier is used to synchronize the execution of multiple processes that are performing a certain task and depend on each other, where each process sets a signal or a flag when it reaches a certain point in the execution (barrier point) and then waits for all other processes to reach this barrier point. Once all processes have reached the barrier point, they are all released to continue their execution with the next stage of their work.

Condition variable is used to synchronize the execution of multiple processes that are depending on each other by allowing them to wait for a specific condition to be met before proceeding. It is often used in conjunction with a mutex or lock to ensure that only one process can access shared resources at a time while the others wait for the required condition to be true before proceeding, preventing race conditions and other concurrency-related issues.

For the purpose of reducing the processing delay, implementation approach can be considered and evaluated in our work to parallelize the three tasks of the M-YOLO detection model subsystem using shared memory blocks and appropriate synchronization mechanisms: pipeline parallelism [236].

6.3 Pipeline Parallelism Approach for the Detection Model

Pipeline parallelism is a computational model that allows tasks to be processed in a fixed sequence of stages, where each stage performs a specific function and passes its output to the next stage in the pipeline [236]. This approach helps in enhancing the processing time as multiple tasks can be processed concurrently at different stages of the pipeline.

The goal of this approach, in the context of the M-YOLO model, is to balance the processing load across all pipeline stages by matching the stage with the shortest processing time to the one with the longest processing time, and then executing them simultaneously [236] [237]. To facilitate communication between cores/stages, shared memory is positioned between adjacent ones. A combination of mutex and barrier synchronization mechanisms is employed to ensure that only one stage can access the shared memory at a time and synchronize the stages at the end of each task by reaching an appropriate synchronization point, identified as the longest processing time,

before proceeding to the next frame. This helps to avoid data races and corruption while ensuring orderly access to the shared memory.

In order to implement pipeline parallelism in our proposed M-YOLO model, it is essential to go through the following process.

6.3.1 Indexing and Labelling the M-YOLO Layers

The M-YOLO model is composed of several blocks operating in a sequential fashion. Each block contains a certain number of layers, leading to a cumulative total of L layers. We merge the pre-processing and post-processing tasks with the first and last convolution layers of the CNN feature extraction task, respectively.

The execution order of these L sequential layers imposes the task execution sequence. The first input layer must be evaluated based on the M-YOLO model input. Subsequently, the second layer can be evaluated once the first layer has completed executing. The model's layers are labelled with an index i, ranging from 1 to L. The correct execution order of the L layers is:

$$L_1 \rightarrow L_2 \rightarrow \cdots L_{L-1} \rightarrow L_L$$

6.3.2 M-YOLO Model Assumptions

There are some assumptions needed to pipeline M-YOLO model, as follows:

- As mentioned before in Section 6.1, the processing time of the pre- and post-processing tasks of the M-YOLO model is small compared to that of the CNN feature extraction task. Therefore, we merge the pre-processing and post-processing tasks with the first and last convolution layers of the CNN feature extraction task, respectively, as represented by the following equations:

$$t_1 \leftarrow t_1 + T_{pre} \tag{6.2}$$

$$t_L \leftarrow t_L + T_{post} \tag{6.3}$$

  where:

  T_{pre}: processing time of the pre-processing task,

  T_{post}: processing time of the post-processing task,

  t_1: processing time of the first convolution layer of the CNN feature extraction task,

t_L: processing time of the last convolution layer of the CNN feature extraction task.

Subsequently, for the M-YOLO model comprising *L* layers, the processing time of each layer of the M-YOLO model is represented by the vector *T*, defined as:

$$T = [t_1, t_2, \cdots, t_L] \tag{6.4}$$

The total processing time of the M-YOLO model is given by:

$$T_{total} = \sum_{i=1}^{L} t_i \tag{6.5}$$

The maximum processing time of the M-YOLO model is defined as:

$$T_{max} = \max(T) \tag{6.6}$$

- Merge the 5th, 6th blocks of the M-YOLO model, in Fig. 6.1, along with the two passthrough Reorg. layers into one block, name it the 'passthrough block', and treat it as a single layer. This merging is done because the passthrough layers cannot be broken down or separated without affecting the model's performance.

6.3.3 Pipeline Throughput

The performance of the M-YOLO model is governed by the maximum delay of the M-YOLO layers. The pipeline clock duration/period for the fastest pipeline is given by:

$$T_{clk} \geq T_{max} \tag{6.7}$$

Based on that, the throughput *th* of the pipeline is given by:

$$th = \frac{1}{T_{clk}} \tag{6.8}$$

6.3.4 Number of Pipeline Stages in Multicore Systems

Pipeline stages can be created with as many cores as are available on the used platform to achieve higher throughput for CNN inference. However, determining the maximum number of pipeline stages is crucial, as it allows for suitable resource allocation and balances the computational load, thereby enhancing the model's processing speed.

There are several restrictions on determining the number of pipeline stages *S*:

- Minimum number of pipeline stages satisfies the inequality:

$$S > 1 \tag{6.9}$$

- Maximum limit on the number of pipeline stages is imposed by the hardware support of the multicore system:

$$S \leq C \tag{6.10}$$

  where C is the number of cores in the utilized hardware multicore system.

- The number of pipeline stages, S, should be determined in a way that the total processing time across all pipeline stages is at least as long as the total processing time of the M-YOLO model, as expressed by:

$$S \times T_{clk} \geq T_{total} \tag{6.11}$$

After calculating the number of pipeline stages, it is vital to design and distribute the M-YOLO layers across these pipeline stages. Moreover, if the utilized platform cannot accommodate the calculated maximum number of pipeline stages, reducing this number becomes necessary; however, maintaining enhancement pipeline performance remains a priority. To address this, we propose an algorithm for designing pipeline stages in the subsequent section.

6.3.5 Proposed Algorithm for Allocating Layers to Pipeline Stages

This algorithm aims to achieve a balanced distribution of computational workload by allocating the M-YOLO model layers to pipeline stages. The goal is to ensure that all stages have processing times close to a given maximum processing time, T_{max}, while adhering to the specified number of pipeline stages, thereby preventing any stage from becoming a bottleneck and slowing down the overall processing speed.

The primary inputs for this algorithm are the vector T, which represents the processing time of each layer; T_{total}, which denotes the total processing time of the model; T_{clk}, representing the pipeline clock period; and S, the desired number of pipeline stages.

Algorithm 6.1 Proposed Algorithm for Allocating Layers to Pipeline Stages.

Require: T, T_{total}, T_{clk}, S.

```
1  L = length(T);
2  i = 1;   //start index of T
3  For s = 1: S do
4      t = 0;
5      Layers_for_current_stage = [ ];
6      While t + T (i) < T_clk do
7          t = t + T (i);
8          Layers_for_current_stage = [Layers_for_current_stage, i];
9          i = i + 1;
10         If i > L then
11             break
12         End if
13     End while
14     pipeline_delay(s) = t;
15     pipeline_layers(s) = Layers_for_current_stage;
16 End for
```

The step-by-step explanation of the algorithm, as shown in Algorithm 6.1, is as follows:

- Initialization:

  Algorithm 6.1 Line 1: *L*: The total number of layers in the M-YOLO model, determined by the length of vector *T*, which contains the processing times for each layer.

  Algorithm 6.1 Line 2: *i*: A counter initialized to 1, representing the starting index of vector *T*. It acts as a pointer to the current layer in vector *T* that is being considered for allocation to a pipeline stage.

- Procedure:

Algorithm 6.1 Line 3: A for-loop begins, iterating variable *s* from 1 to *S* (the desired number of pipeline stages).

Algorithm 6.1 Line 4: *t*: A variable initialized to 0, representing the accumulated processing time for the current pipeline stage *s*.

Algorithm 6.1 Line 5: Layers_for_current_stage: An empty array initialized to keep track of the layers allocated to the current pipeline stage *s*.

Algorithm 6.1 Line 6-13: A while-loop begins, with the condition '$t + T(i) < T_{clk}$'. This loop continues allocating layers to the current pipeline stage *s* as long as the accumulated processing time *t* plus the processing time of the next layer $T(i)$ does not exceed the specified pipeline clock period T_{clk}.

Algorithm 6.1 Line 7: Update *t* by adding the processing time of the new current layer $T(i)$ to the accumulated processing time *t* for the current pipeline stage *s*.

Algorithm 6.1 Line 8: Append the index *i* to the array 'Layers_for_current_stage', indicating that this layer is allocated to the current pipeline stage *s*.

Algorithm 6.1 Line 9: Increment the counter *i* by 1, moving to the next layer.

Algorithm 6.1 Line 10-13: Conditional statement to check if *i* exceeds *L*, and if so, breaks out of the while-loop as there are no more layers to process.

Algorithm 6.1 Line 14: pipeline_delay(*s*): Record the total processing time *t* accumulated for the current pipeline stage *s*.

Algorithm 6.1 Line 15: pipeline_layers(*s*): Record the indices of layers allocated to the current pipeline stage *s*.

Algorithm 6.1 Line 16: Ends the for-loop, moving on to the next pipeline stage *s* until all layers have been allocated.

This algorithm attempts to distribute the layers across the specified number of pipeline stages, ensuring that the total processing time for each stage is within the specified pipeline clock period, T_{clk}. By doing so, it enhances resource utilization and thereby the processing speed of the detection

model during inference, leading to a well-distributed computational load across different pipeline stages.

6.3.6 Proving the Correctness of the Proposed Algorithm

The following theorem proves that the Algorithm 6.1 maintains the correctness of execution of the M-YOLO model.

Theorem 6.1: Pipelining the M-YOLO model following the Algorithm 6.1 maintains the correctness of the detection model.

Proof: The FOR loop spanning Lines 3-16 in Algorithm 6.1 iterates through the M-YOLO model layers in increasing index order. Moreover, The WHILE loop spanning Lines 6-13 in Algorithm 6.1 ensures that each selected layer has an index that is one unit greater than the index of previously selected layer. Therefore, we ensured the correct execution of the M-YOLO model through pipelining.

6.4 Pipelining Implementation of M-YOLO Model Using Python

The processing time of each layer of the M-YOLO model was measured using the Python module mentioned in Section 6.1, on a Core i7 system. Although the 7th block seems independent, the processing times of its convolution layers are 0.13 ms and 0.02 ms, respectively. Given these small processing times, treating it separately would make it challenging to achieve a balanced pipeline. Therefore, merging the 7th block with the 5th and 6th blocks emerges as an efficient strategy for pipeline processing. Fig. 6.3 represents the processing time of the first 12 layers of the CNN feature extraction task and the passthrough block, which is treated as a single layer, within the M-YOLO model.

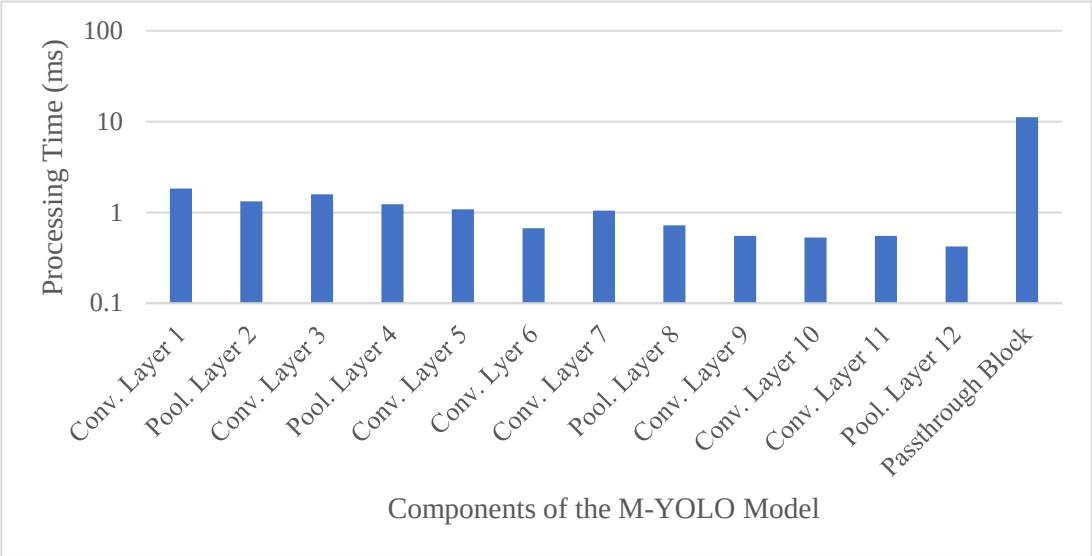


Figure 6.3: Processing time for the components of the M-YOLO model as measured on a Core i7 system.

Given the structure of the M-YOLO model, the passthrough block is treated as a single layer, which provides the maximum processing time, T_{max} = 11.20 ms. As mentioned in Section 6.1, the total processing time of the M-YOLO model is T_{total} = 21.99 ms. From Equation (6.11), and based on the restrictions outlined in Section 6.3.4, the number of pipeline stages, S, is set to 2. Therefore, a two-stage implementation is adopted for the M-YOLO model.

6.4.1 Two-Stage Pipelining Implementation

In the two-stage pipelining, after merging the pre-processing task to the first convolution layer and the post-processing task to the last convolution layer of the M-YOLO model, the allocation of layers to pipeline stages was performed following the proposed Algorithm 6.1. As shown in Fig. 6.4, the first 12 layers of the M-YOLO model are assigned to the first stage, which takes a processing time of T_{1s} = 10.79 ms. Meanwhile, the passthrough block, which is treated as a single layer, is assigned to the second stage, which takes a processing time of T_{2s} = 11.20 ms. The constraint imposed by the passthrough layers restricts the allocation from extending beyond the first 12 layers, ensuring the integrity and functionality of the model while adhering to the pipelining paradigm.

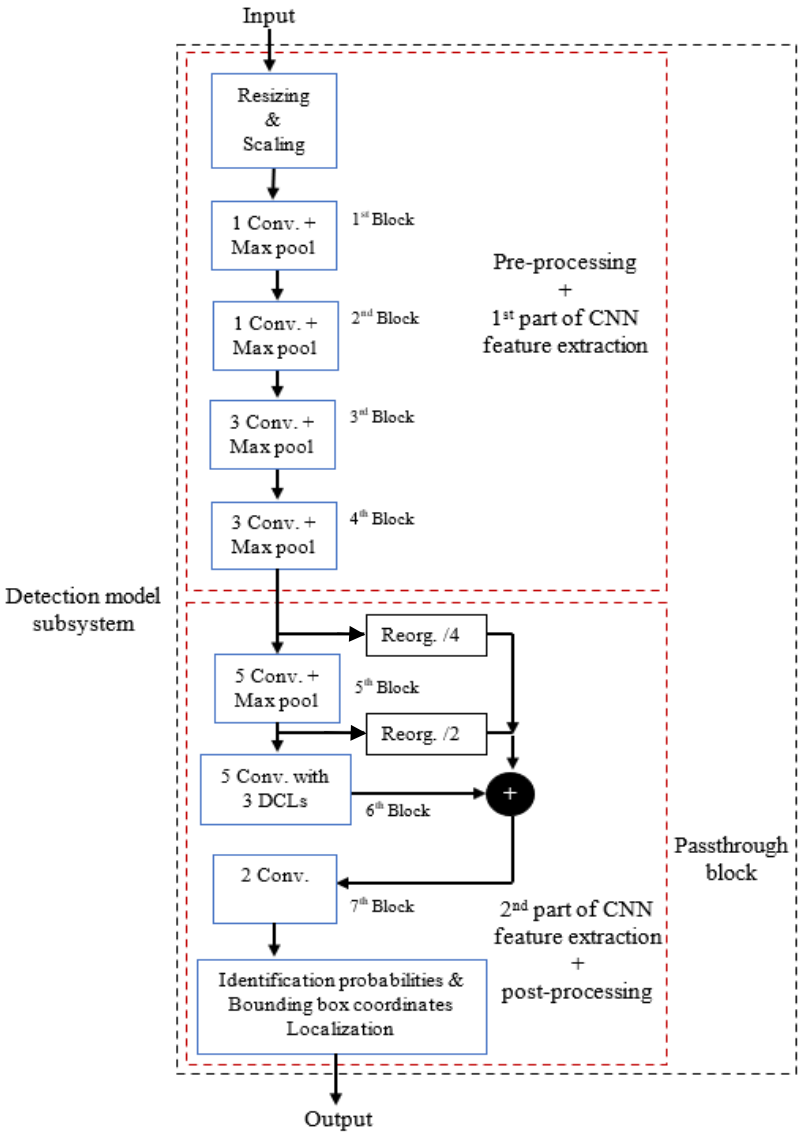


Figure 6.4: Distribution of M-YOLO model layers across the two-stage pipeline. Passthrough block is treated as a single layer.

A single shared memory is utilized to facilitates communication between the two cores/stages. As shown in Fig. 6.5, the first stage writes its output to the shared memory location, which is then read by the second stage as its input. Because of these dependencies between the stages, a combination of mutex and barrier synchronization mechanisms is employed to ensure that each stage of the model is executed in the correct order, maintaining the integrity of the pipeline parallelism approach. This way, the mutex not only prevents simultaneous access to the shared memory (preventing data races and other concurrency issues), but it also serves as a signal to each

stage that the previous stage is done processing the current frame and the processed frame is ready to be read.

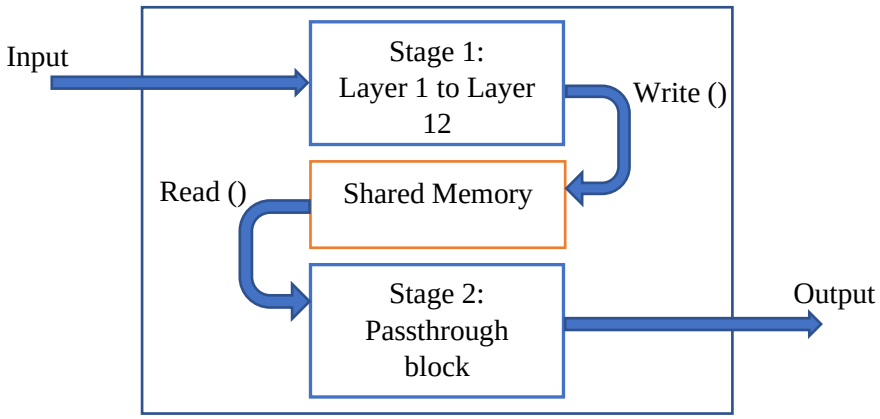


Figure 6.5: Pipeline parallelism implementation block diagram of the detection model.

The implementation of a two-stage pipelining in the proposed M-YOLO model has the following steps:

1. Allocate a shared memory block between the two stages to hold the intermediate results produced by the first stage.
2. The first stage processes the first 12 layers of the M-YOLO model, which takes a processing time of T_{1s} = 10.79 ms.
3. Once the first stage completes the processing of a frame, it acquires a mutex to place the intermediate results into the shared memory block to prevent the other stage from accessing it, then releases the mutex before reaching the barrier.
4. The second stage then acquire a mutex and access the shared memory and processes the passthrough block, which takes a processing time of T_{2s} = 11.20 ms, while the first stage begins processing the next frame.
5. The two stages work concurrently, utilizing shared memory with the mutex ensuring exclusive access to the shared memory block and the barrier ensuring that both stages are synchronized at the end of each frame's processing, thereby maintaining the correct processing order.

Fig. 6.6 shows an example of the pipeline parallelism implementation of the detection model subsystem using two-stage pipelining. The pipeline clock period is determined by the maximum processing time:

$$T_{clk} = T_{max} = T_{2s} = T^* \tag{6.12}$$

Therefore, the total parallel processing time for each frame can be calculated as:

$$T_{ppt} = 2T^* \tag{6.13}$$

The throughput can then be calculated as:

$$th = 1 / T_{clk} = 1 / T^* \tag{6.14}$$

The total elapsed time for N input frames can be calculated as:

$$\text{Elapsed time} = T^* + (N \times T^*) \tag{6.15}$$

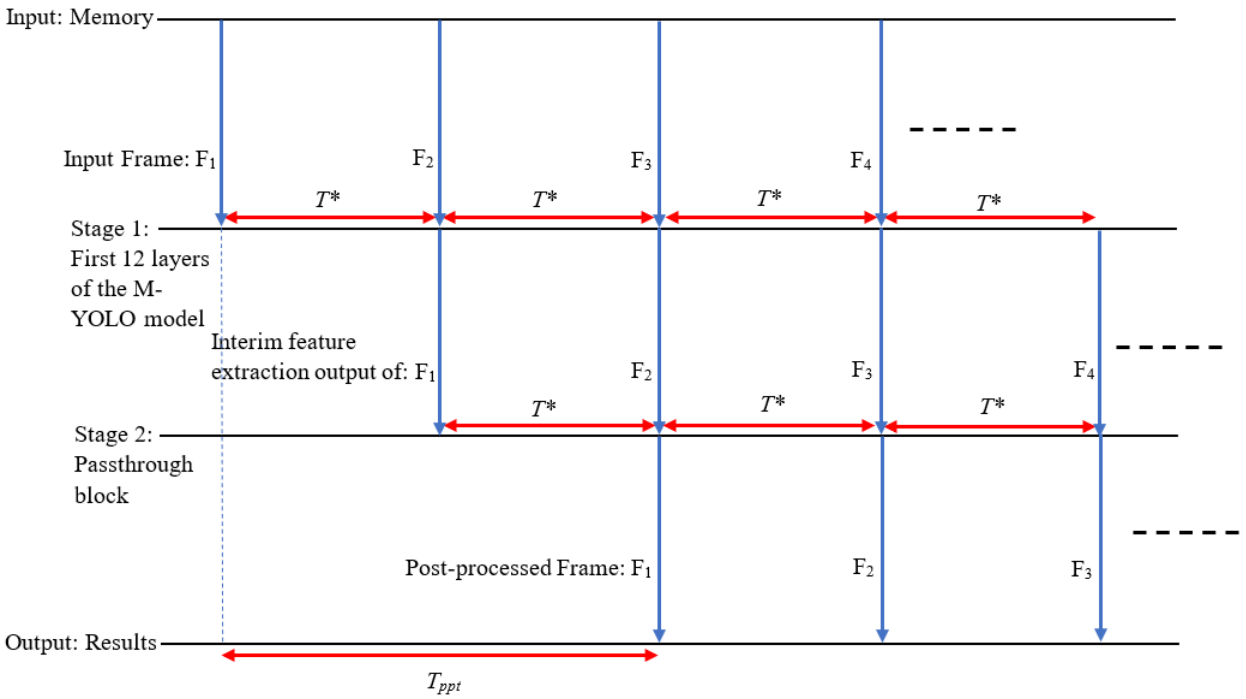
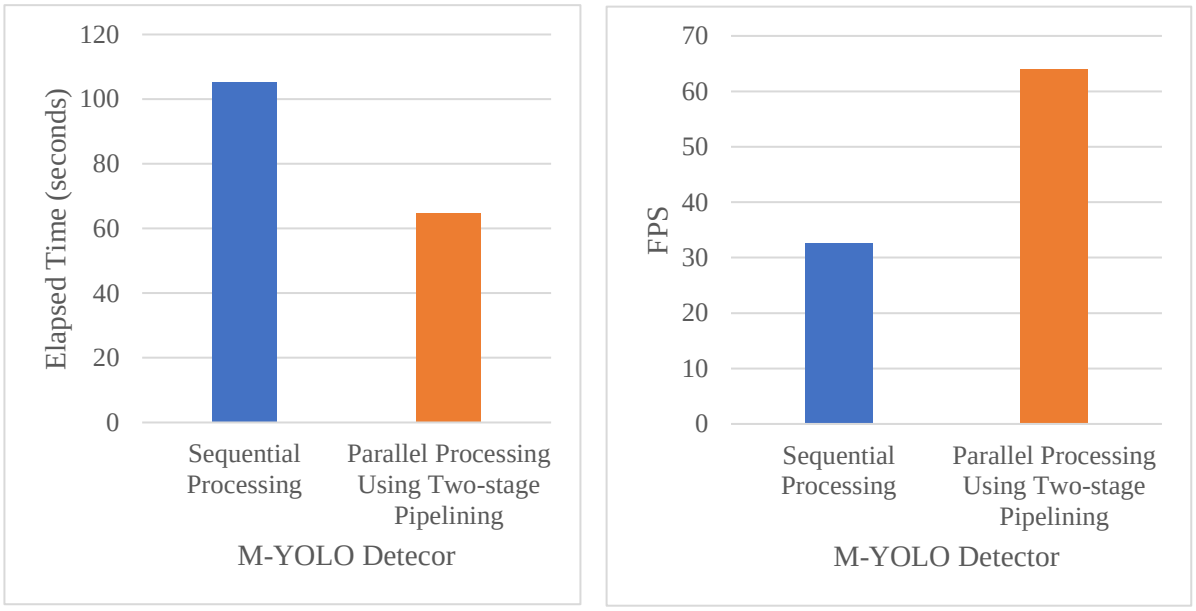


Figure 6.6: Pipeline parallelism implementation of the detection model subsystem using two-stage pipelining. Data transfer time is insignificant.

6.4.2 Processing Speed Evaluation of Two-Stage Pipelining in M-YOLO Detector

In this section, the processing speed of the two-stage pipelining for the detection model subsystem is assessed using two key metrics: elapsed time for batch processing and FPS for real-time processing, as defined in Section 5.3.

Fig. 6.7 provides a comparison between the M-YOLO detector utilizing sequential/single-core processing and two-stage pipelining. For our evaluation, we used a 4160-frame video with a resolution of 1280 × 720 pixels, sourced from [238], along with real-time footage from a 30 FPS web camera with a resolution of 1280 × 720 pixels. As shown in Fig. 6.7(a), for batch processing, the two-stage pipelining of the M-YOLO model achieves a significant reduction of 38.3% in elapsed time compared to sequential processing. In Fig. 6.7(b), for real-time processing, the FPS using the two-stage pipelining of the M-YOLO model outperforms the sequential processing by a factor of 1.9 times.



(a) (b)

Figure 6.7: Processing speed evaluation of the original M-YOLO detector and its two-stage pipelining implementation in terms of: (a) elapsed time for batch processing sourced from [238], and (b) FPS for a web camera feed.

Based on the results obtained from our experiments, it was found that the two-stage pipelining implementation effectively enhances the processing speed of the animal species detector compared to the sequential processing, across both batch and real-time processing scenarios.

6.5 Conclusion

This chapter explained the strategic shift from a sequential processing framework to a parallel multicore processing paradigm in our proposed M-YOLO detector, to reduce the processing delay inherent in sequential implementation. By employing a pipeline parallelism approach, the processing layers were distributed across different stages, thus ensuring concurrent execution and significantly enhancing the throughput. The proposed algorithm for layer allocation to pipeline stages was vital in achieving a balanced distribution of computational workload, thus preventing any stage from becoming a bottleneck. The two-stage pipelining implementation, as exhibited through the processing speed evaluation, outperformed the sequential processing of the M-YOLO model in both batch and real-time processing scenarios, reducing the elapsed time by 38.3% and doubling the frame rate. Pipelining indeed shows promise in our current context, but its effectiveness may vary with different models, largely depending on their architecture.

Chapter 7

Incorporating MCF Algorithm and Parallel Processing Technique into M-YOLO Detector for Enhanced Real-World Detection

In this chapter, we propose the integration of the MCF algorithm, discussed in Chapter 5, with the two-stage pipelining parallel approach, discussed in Chapter 6, into the proposed M-YOLO detector, as shown in Fig. 7.1. This integration employs both pipelining and dataflow [237] techniques, combining elements from both paradigms. This combination is expected to enhance the processing speed and power consumption of the M-YOLO model, leading to faster animal species detection in real-world scenarios.

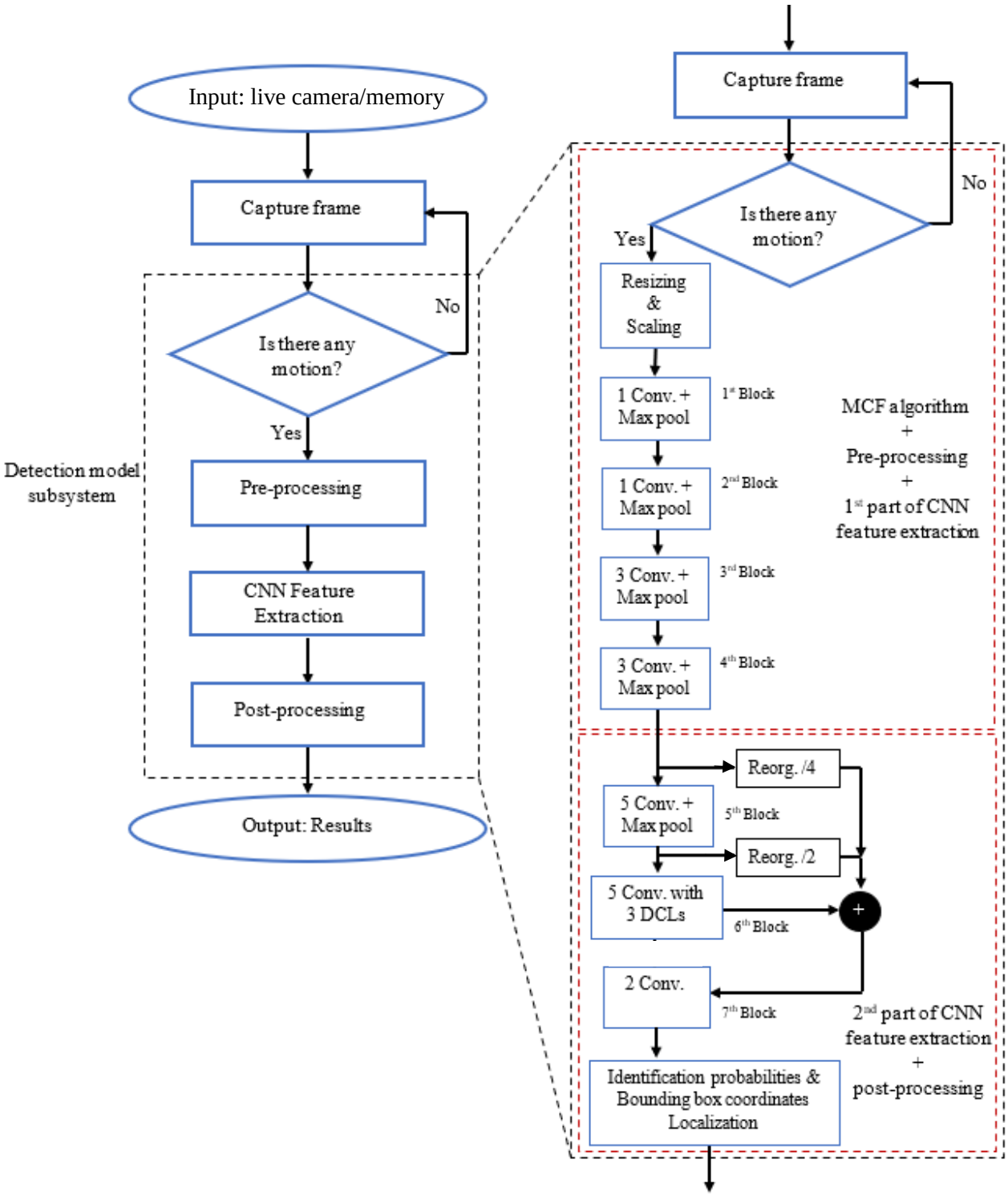


Figure 7.1: Processing flow of the detection model subsystem after integrating the MCF algorithm and two-stage pipelining parallel approach.

7.1 Background of Dataflow Approach

In this section, we explore how data dependencies between different tasks can be represented by a dataflow graph (DFG). This DFG consists of three sets: variables (*V*), functions (*F*), and directed arcs (*A*). Variables are classified into three types based on their locations in the algorithm: input, intermediate, and output. Functions represent the tasks and operations of the algorithm, and they are allocated to processors. Directed arcs depict the communication and dependencies between variables and functions, associating them with memories or registers. Fig. 7.2 provides an example of a DFG of an algorithm composed of ten variables and seven functions [237]. Variables could have multiple outputs but only one input, while functions might have multiple inputs but only one output.

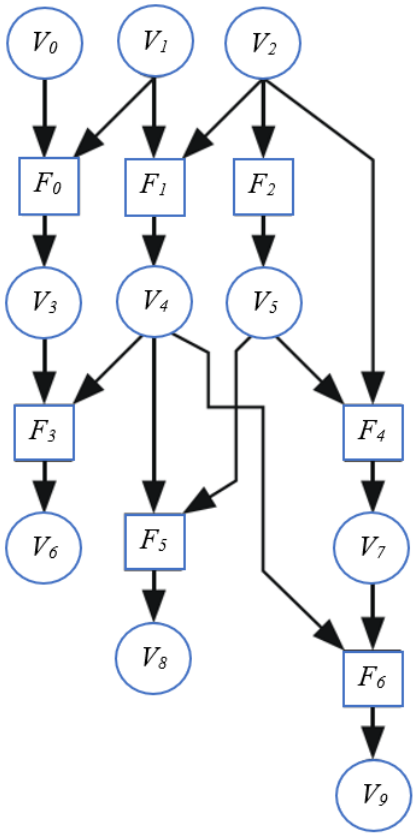


Figure 7.2: A DFG for an algorithm. A variable is denoted by a circle and a function is denoted by a square.

As shown in Fig. 7.3, the timing of the availability of variables, marking the initiation of function execution, is represented by blue circles on the graph, termed events/tokens. These tokens

are assigned to the variables when they are valid and available for use by the function. Variables V_0 and V_1 have tokens, indicating that function F_0 and F_1 will be triggered. Consequently, when a function is finished, a token is placed at the variable node associated with its output to signify the availability of this variable, as can be observed with the internal variable V_3.

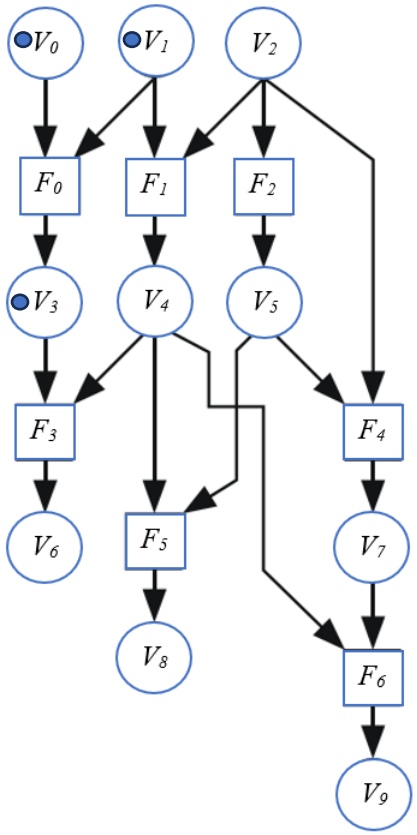


Figure 7.3: State of DFG for an algorithm at a given time.

In Python, to ensure that data flows in the correct order and resources are accessed in a safe manner, synchronization mechanisms are implemented. A combination of condition variable and mutex synchronization mechanisms is well-suited for scenarios that require a flexible synchronization strategy, where individual tasks either proceed or wait based on a specific condition, rather than synchronizing all cores at a common point. This condition revolves around the availability of new data in the shared memory block.

7.2 Proposed Pipelining-Dataflow Hybrid Approach for the Detection Model

This hybrid approach aims to leverage the throughput benefits of pipelining while incorporating the data-driven execution characteristic of dataflow paradigms to manage conditional processing based on the MCF algorithm. In a pure pipelining paradigm, each pipeline stage would proceed to process data in a fixed sequence without conditional decision-making based on data content. On the other hand, a pure dataflow paradigm would have tasks executing more independently based on data availability without a fixed sequence of stages.

Following our experiments from Section 6.4, we employed a two-stage strategy to implement the pipelining-dataflow hybrid approach in the proposed MCF-YOLO model. As shown in Fig. 7.4, the pipelining-dataflow hybrid approach of our proposed MCF-YOLO model is composed of two parts:

- Pipelining part: it consists of two pipeline stages, S_1 and S_2, with two registers/memories R_1 and R_2.
- Dataflow part: it manages the data-driven conditional processing. When the MCF block identifies motion within frames, the event/token D_1 signals and prompting R_1 to load the data (*LD*) to the first processing stage, S_1. After a designated pipeline clock period, D_2 signals, instructing R_2 to load the data (*LD*) to the second processing stage, S_2, which then initiates its operations.

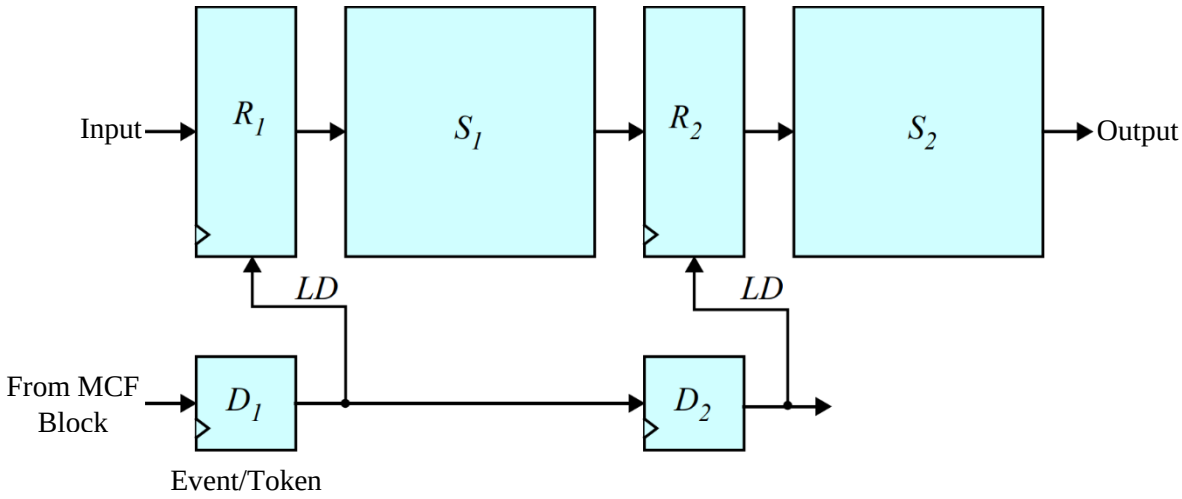


Figure 7.4: Logical block diagram of a two-stage pipelining-dataflow hybrid approach of the MCF-YOLO model.

Algorithm 7.1 represents the implementation of a two-stage pipelining-dataflow hybrid approach in the proposed MCF-YOLO model.

Algorithm 7.1 Two-Stage Pipelining-Dataflow Hybrid Paradigm for the MCF-YOLO Model.

```
1  Create: shared memory space: shared_memory;
           condition variable: condition_variable;
           mutex: mutex_lock;
2  Function MCF:
3       While True:       // Continuously check for motion in frames
4            Frame = CaptureFrameData ();
5            Motion_detected = MCFAlgorithm (Frame);
6            If  Motion_Detected:
7                 Call Function S1;
8            Else:
9                 Wait until the next frame is available;
10           End If;
11 Function S1:
12      Preprocess_data = Preprocess (Frame);
13      Feature_data_twelve_Layers = CNNFEU (Preprocess_data);
14      Acquire mutex_lock;
15       shared_memory ['Frame'] = Feature_data_twelve_Layers;
16       shared_memory ['token'] = True;
17       Signal condition_variable;  // Notify S2 that data is ready
18       Release mutex_lock;
19 Function S2:
20      Acquire mutex_lock;
21      Wait until shared_memory['token'] is True;
22      Feature_data_passblock = CNNFEL (shared_memory ['Frame']);
23      Postprocess_data = Postprocess (Feature_data_passblock);
24      shared_memory ['token'] = False;
```

25 **Release** mutex_lock;

26 Allocate S_1 to Core 1

27 Allocate S_2 to Core 2

28 Start executing S_1 and S_2 concurrently

Algorithm 7.1 illustrates a hybrid approach, combining aspects of both pipelining and dataflow paradigms. Here's how:

1. Pipelining:
   The arrangement of Stage 1, S_1, and Stage 2, S_2, in a fixed sequence where Stage 2 depends on the output of Stage 1 represents a pipelining paradigm. Pipelining helps in maximizing the throughput of the processing by allowing Stage2 to process one frame while Stage1 is processing the next.
2. Dataflow:
   The use of a condition variable to synchronize the processing of pipeline stages based on the availability of data (in this case, motion detected, and data readiness signaled by a token) represents a dataflow paradigm. In dataflow, the execution of pipeline stages is driven by data availability.
3. Conditional Processing:
   The conditional processing based on motion detection is a characteristic of data-driven processing, which is a key aspect of dataflow paradigms.
4. Shared Memory:
   The use of shared memory for inter-stage communication is a common technique in both pipelined and dataflow architectures to enable data sharing between different stages or processes. The mutex is used to prevent data corruption during concurrent access by the two stages.

The integration of a decision node (motion detection/ MCF algorithm) to direct the flow of data, coupled with synchronization based on data availability (signaled by a condition variable), blends elements of both pipelining and dataflow paradigms. This hybrid approach aims to leverage the throughput benefits of pipelining while incorporating the data-driven execution characteristic of dataflow paradigms to manage conditional processing based on motion detection.

Fig. 7.5 shows an example of the two-stage pipelining-dataflow hybrid approach of the MCF-YOLO model. When a frame exhibits motion, an event/token triggers the first stage, S_1, to process the first 12 layers of the CNN feature extraction task. Subsequently, the second stage processes the passthrough block of the MCF-YOLO model. Among the frames, F_1 and F_4 exhibit motion, while F_2, F_3, F_5, and F_6 do not.

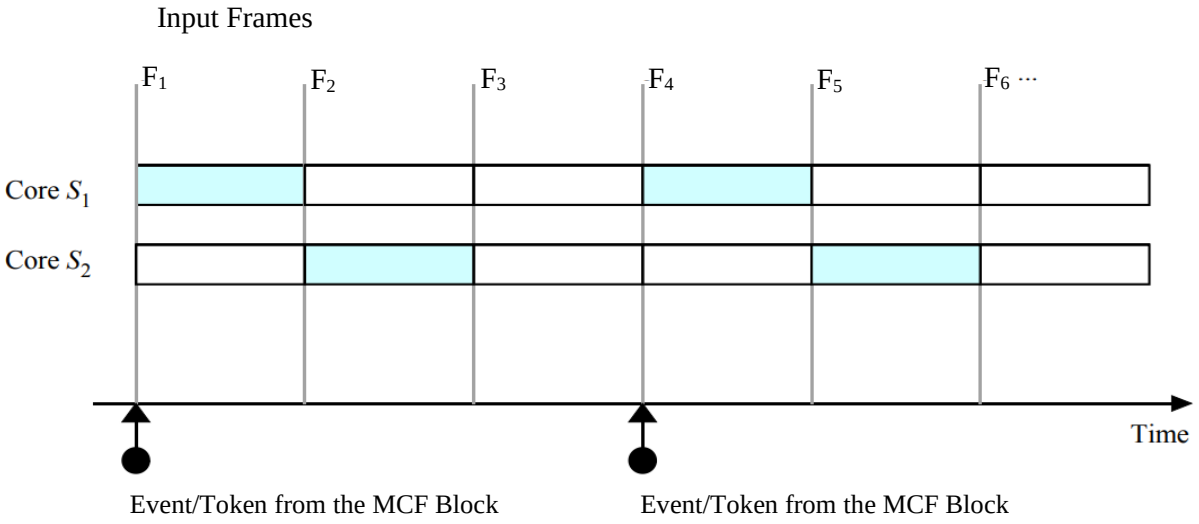


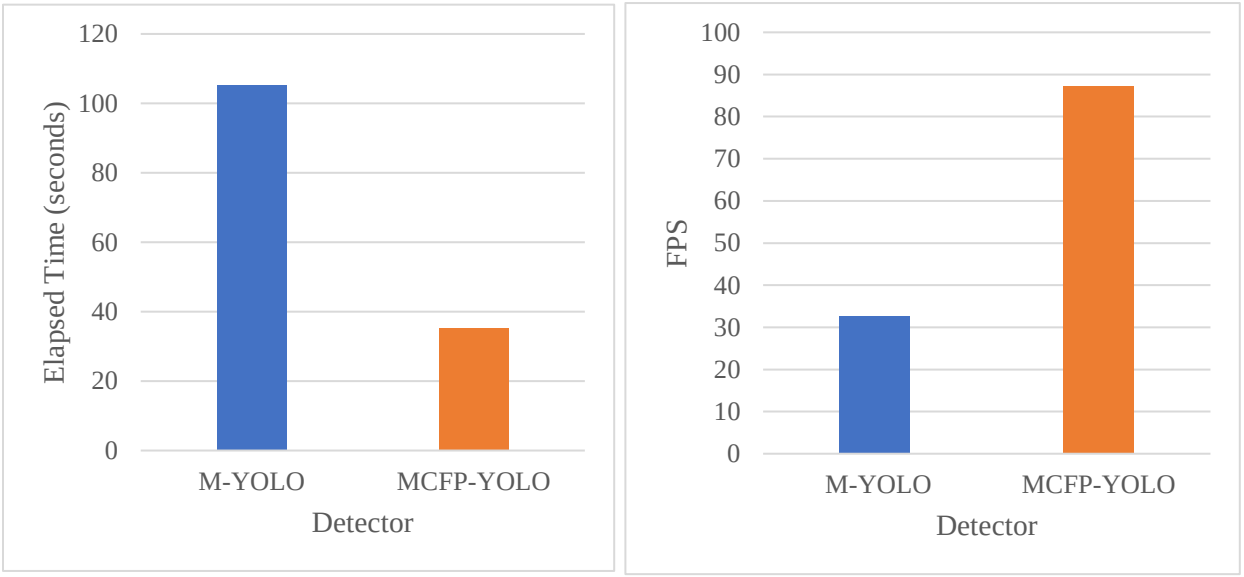
Figure 7.5: Parallel implementation of the MCF-YOLO model using two-stage pipelining-dataflow hybrid approach. Data transfer time is insignificant.

7.3 Efficiency Evaluation of the MCFP-YOLO Detector: Processing Speed and Power Consumption

By parallelizing the MCF-YOLO detector using the two-stage pipelining-dataflow hybrid approach, we produced a new detector named MCFP-YOLO. The implementation of this proposed detector is structurally similar for both batch and real-time processing. However, for real-time processing, an additional component, a queue, is used to manage incoming frames from a live web camera feed, as mentioned earlier in Section 5.3.1.

The evaluation metrics provide valuable information on the efficiency and effectiveness of the proposed MCFP-YOLO animal species detector in real-world scenarios. Efficiency refers to achieving our goals with minimal processing time and power consumption; while effectiveness refers to the detector's ability to accurately identify and localize objects in images, as well as its proficiency in distinguishing between various classes, even with subtle differences between them.

Fig. 7.6 presents a comparison on a Core i7 system, as specified in Section 4.4.3, between two detectors: the proposed MCFP-YOLO (MCF-YOLO with parallel processing) and the original M-YOLO detector in terms of the elapsed time and FPS for batch processing and real-time processing, respectively. As shown in Fig. 7.6(a), for batch processing of a video (detailed in Section 6.4.2) in which 69% of the frames exhibit motion, the elapsed time of the proposed MCFP-YOLO detector is reduced by 66.5% compared to the original M-YOLO detector. Similarly, as shown in Fig. 7.6(b), for real-time processing using footage from a web camera (also detailed in Section 6.4.2), the FPS of the proposed MCFP-YOLO detector is enhanced by 2.6 times compared to the original M-YOLO detector.

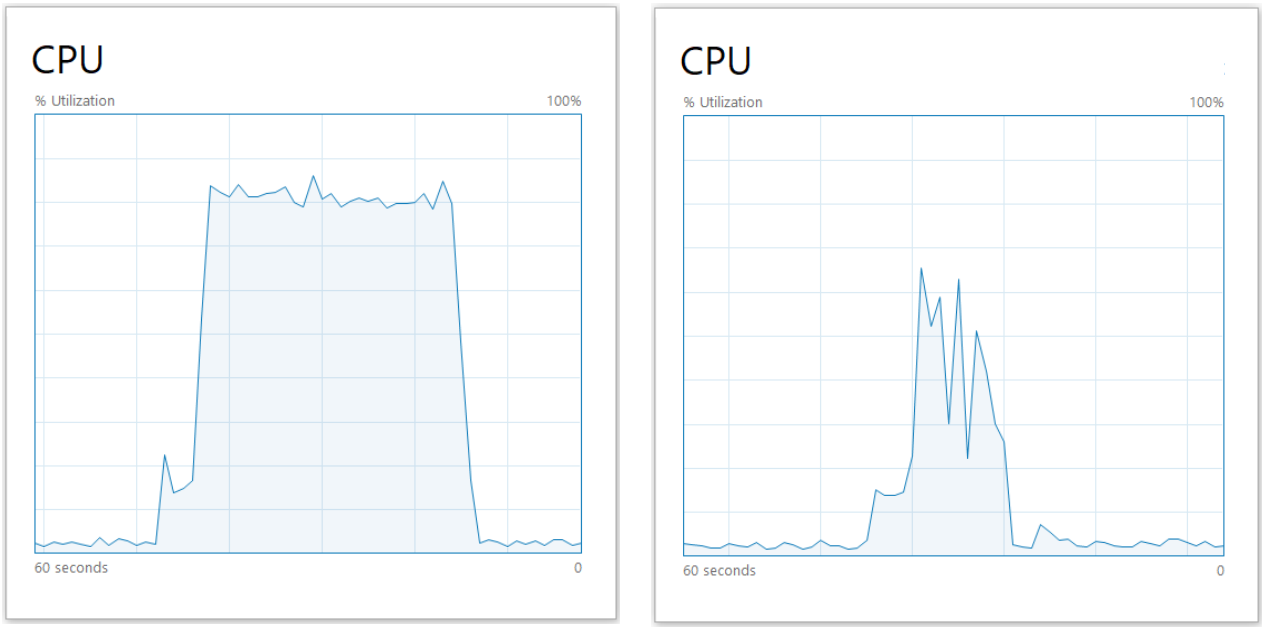


(a) (b)

Figure 7.6: Processing speed comparison of the original M-YOLO detector and the proposed MCFP-YOLO animal species detector in terms of: (a) elapsed time for batch processing sourced from [238], and (b) FPS for real-time processing from a web camera feed.

Following the demonstration of the enhanced processing speed with the proposed MCFP-YOLO, the CPU utilization metric is used to evaluate and compare the power consumption of the CPU during the operation of both the original M-YOLO and the proposed MCFP-YOLO detectors in batch and real-time processing. The results in Fig. 7.7 illustrate the variations in CPU utilization over a 60 second time duration while processing a video sourced from [238] using both detectors. As shown in Fig. 7.7(a), with the M-YOLO detector, the CPU is only active when tasks are assigned. Consequently, the integration of the MCF algorithm and two-stage pipelining-dataflow

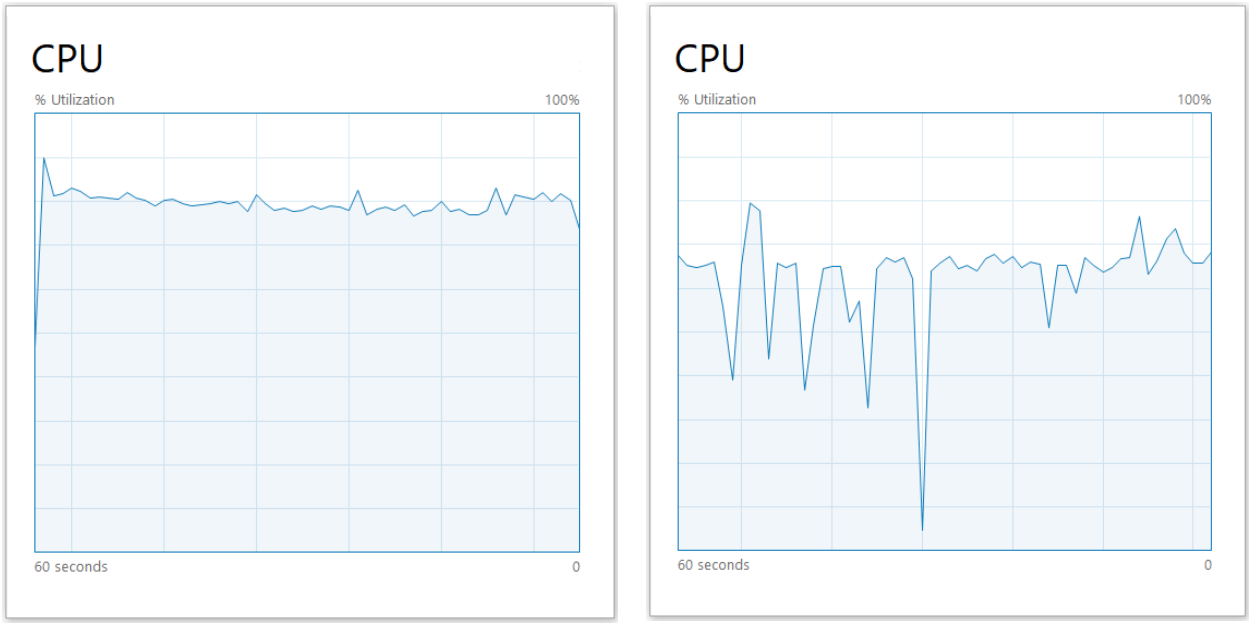
hybrid implementation into the M-YOLO model, as demonstrated in Fig. 7.7(b), leads to a significant reduction in power consumption compared to the original M-YOLO detector. This reduction is achieved because the CPU enters idle states during periods when frames are dropped due to the absence of detected animal motion. Furthermore, the two-stage pipelining-dataflow hybrid approach contributes to power savings, by allowing for more efficient use of cores and reducing idle time.



(a) (b)

Figure 7.7: Comparison of CPU utilization percentage for batch processing sourced from [238] using: (a) the original M-YOLO detector, (b) the proposed MCFP-YOLO detector.

Fig. 7.8 provides a visual comparison of how the CPU responds during real-time processing when using the two detectors. Fig. 7.8(a) shows the use of the original M-YOLO detector, where the CPU operates at a high-performance level, leading to elevated power consumption. On the other hand, Fig. 7.8(b) shows the situation when the proposed MCFP-YOLO detector is used. In this case, the CPU manages to perform the required tasks without needing to operate at this high-performance level, thereby resulting in reduced power consumption.



(a) (b)

Figure 7.8: Comparison of CPU utilization percentage for real-time processing from a web camera feed using: (a) the original M-YOLO detector, and (b) the proposed MCFP-YOLO detector.

The above results emphasize the advantages of the proposed MCFP-YOLO detector, making it suitable for deployment on embedded systems for WHCs and WVCs mitigation systems.

7.4 The MCFP-YOLO Detector on Embedded Systems

The proposed MCFP-YOLO detector was deployed on both the RP4B and NVIDIA Jetson Nano devices [239], to evaluate its efficiency in terms of processing speed and power consumption. This evaluation was conducted using FPS, current consumption (mA), and CPU utilization (%) metrics for real-time processing from a web camera feed.

The RP4B features a Quad-core Cortex-A72 64-bit, 1.5 GHz CPU, and a VideoCore VI 3D 500 MHz GPU. It can support up to 8 GB of SDRAM and uses a microSD for storage. RP4B has a two-lane camera port, HDMI and Display Port, and 4 USB ports. Additionally, the RP4B supports Gigabit Ethernet, Bluetooth 5.0, and 2.4/5 GHz Wi-Fi for connectivity [177]. Similarly, the NVIDIA Jetson Nano, another embedded device, includes a Quad-core Cortex A-57 64-bit, 1.43 GHz CPU, and an NVIDIA 128-core GPU. Jetson Nano has 4 GB memory with a RAM speed of 25.6 GB/s and a microSD for storage. It also has a 2-lane camera port, HDMI and Display Port, and 5 USB ports [239].

As presented in Table 7.1, the results show that the Jetson Nano outperforms the RP4B by approximately 1.2 times in terms of processing speed, but it also consumes approximately 4.7% more current and exhibits 33.3% higher CPU utilization. Moreover, in terms of cost, the Jetson Nano device is significantly pricier than the RP4B.

Table 7.1: Comparison between RP4B and Jetson Nano devices in terms of FPS, current consumption, CPU utilization percentage for real-time processing from a web camera feed, and cost.

	RP4B	**Jetson Nano**
FPS	19.7	23.3
Current (mA)	876.9	918.3
CPU Utilization (%)	42.6	56.8
Cost (C$)	195 [240]	321 [239]

Despite these differences, the RP4B device proves to be a suitable choice for our work as it offers a desirable balance between cost and efficiency, including processing speed and power consumption. Compared to other embedded devices such as those referenced in [241] [242], the RP4B provides a relatively powerful processing capability at a lower cost and with less power consumption. This makes it an affordable and power-efficient option for real-time wildlife detection systems. These features enable efficient real-time processing while maintaining budget and power constraints, which are crucial for the widespread and successful deployment of WHCs and WVCs mitigation systems.

7.5 Applications Development

In an effort to assist biologists, researchers, and conservationists in monitoring wildlife, we have developed two executable applications leveraging our proposed MCFP-YOLO animal species detector: Live Camera Animal Detection System (LCADS), as shown in Fig. 7.9(a), and Video Image Animal Detection System (VIADS), as shown in Fig. 7.9(b).

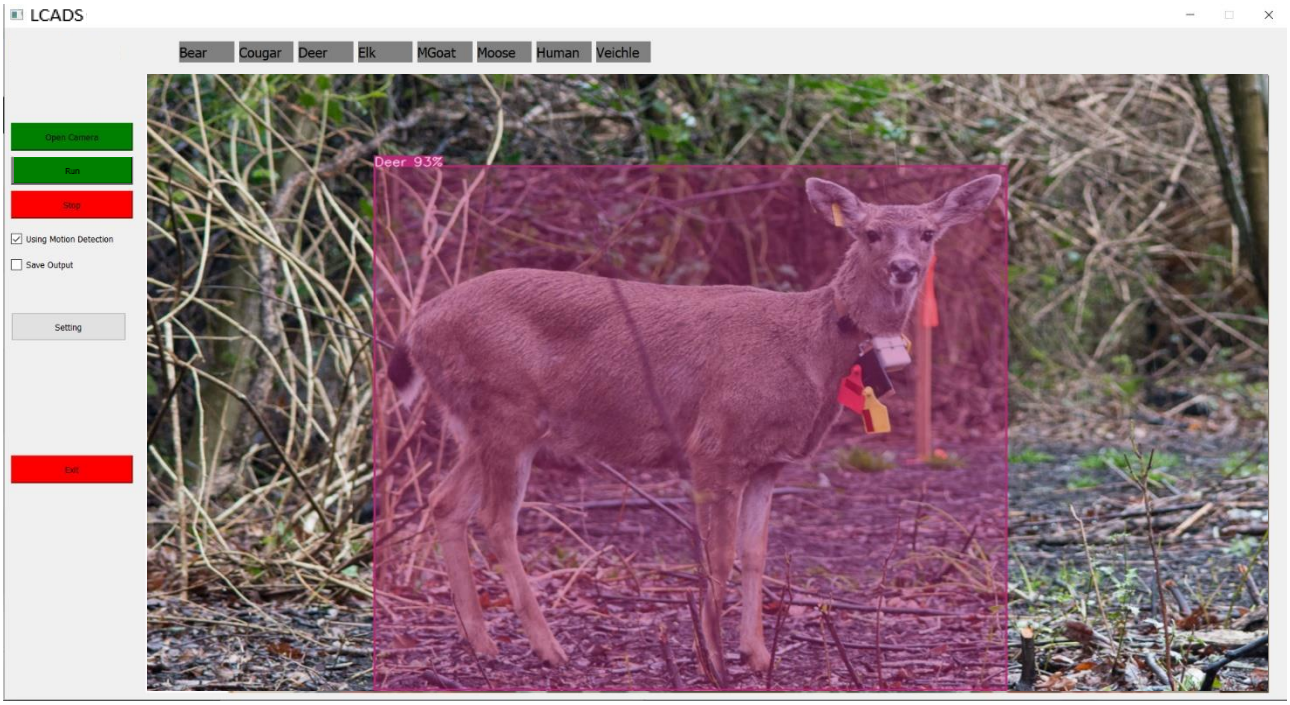
LCADS is designed for real-time wildlife monitoring using live camera feeds, enabling instant detection and response to wildlife presence. This system is particularly useful in areas with high wildlife traffic, where quick responses are crucial for preventing accidents, reducing WHCs, and

preserving habitats. To extend its functionality, we have included two additional classes (human and vehicle), allowing LCADS to also serve as a security camera for residential areas. This enhancement makes the system adaptable to various use cases. LCADS can be easily integrated into existing infrastructure, such as traffic monitoring systems, park management, or residential security systems to enhance situational awareness and enable more informed decision-making.

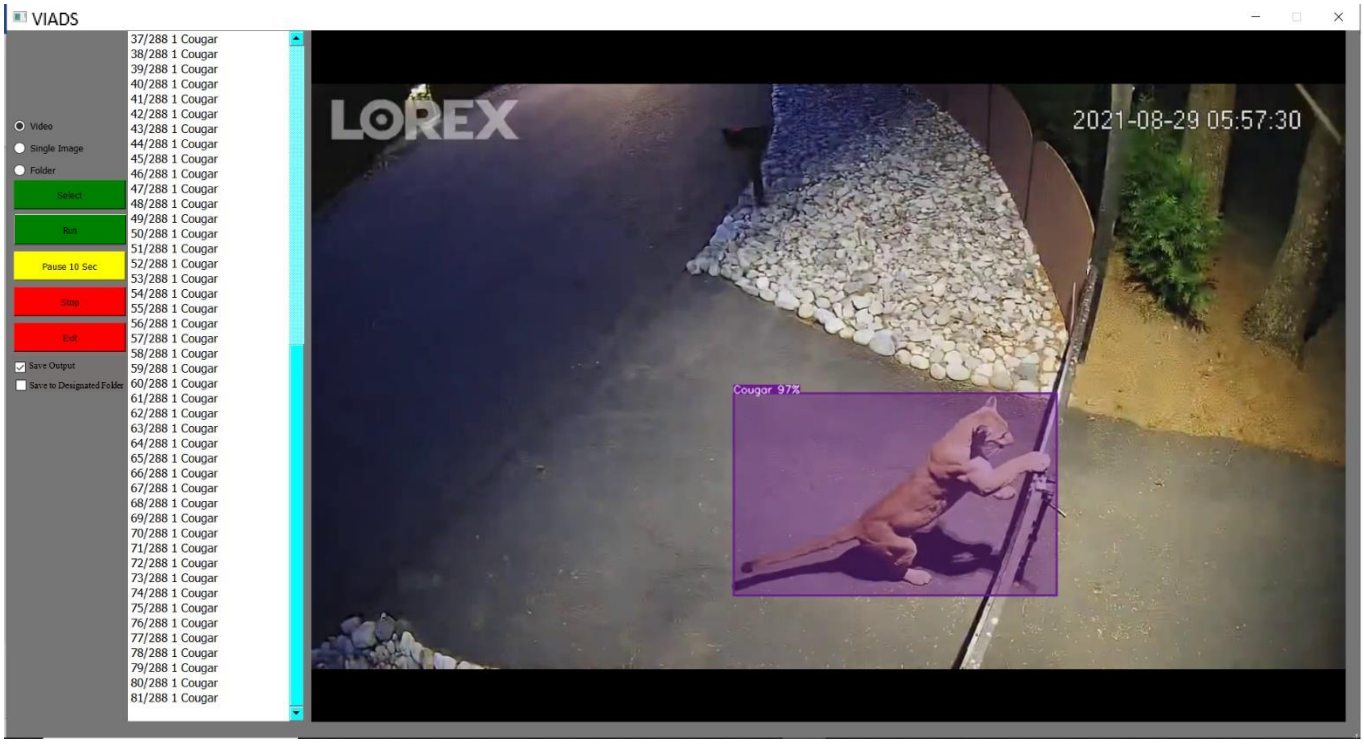
On the other hand, VIADS processes pre-recorded video footage or folder of images, allowing for more effective analysis of large volumes of data to study wildlife behavior and patterns in-depth over time. VIADS is a valuable tool for researchers and conservationists seeking to understand wildlife habits and develop targeted strategies to protect threatened species or manage habitats. These applications demonstrate the capability of our proposed MCFP-YOLO animal species detector in addressing various wildlife-related challenges. By developing these applications, we aim to support ongoing efforts to conserve wildlife and promote coexistence between humans and animals.

These applications were developed using the “PyQt” Python library for the user interface. Additionally, the “PyInstaller” library was employed to create standalone executables for multi-core processors on the Windows 10 platform.

Several projects in Western Canada [243], have deployed hundreds of cameras to monitor animal species such as bears, cougars, deer, elk, moose, mountain goats, etc. These projects could benefit from our applications to enhance real-time monitoring capabilities, automate species identification and data collection, and streamline the data analysis process for improved wildlife management and conservation efforts.



(a)



(b)

Figure 7.9: Animal species detection applications: (a) LCADS, and (b) VIADS.

7.6 Conclusion

In this chapter, we explored the integration of two ideas into the M-YOLO animal species detector: the MCF algorithm and the utilization of parallel processing technique. The MCF algorithm is designed to minimize the processing delay of the frame capture subsystem by controlling the number of frames to be processed by the detection model subsystem, based on detected motion activity in the frames. On the other hand, the parallel processing technique aims to minimize the processing delay of the detection model subsystem by processing the three tasks (pre-processing, CNN feature extraction, and post-processing) in parallel using two-stage pipelining-dataflow hybrid approach. Experimental results were provided to validate these proposed ideas and their effectiveness in improving the efficiency (detection speed and power consumption) of the proposed MCFP-YOLO detector, particularly for embedded systems with limited resources.

The proposed MCFP-YOLO detector was then deployed on both RP4B and Jetson Nano devices to assess their performance in real-life scenarios. The results indicate that while Jetson Nano outperforms the RP4B by approximately 1.2 times in terms of speed, it also consumes around 4.7% more current and utilizes 33.3% more CPU. The RP4B is advantageous for our system, especially when it operates for extended durations on limited power resources, such as in remote areas or relying on battery power. Therefore, the RP4B remains a suitable choice for wildlife detection applications due to its balance between cost and performance, making it an affordable and power-efficient option for real-time WHCs and WVCs mitigation systems.

Chapter 8

Summary, Contributions and Future Work

8.1 Summary

This dissertation explores the application of deep learning-based object detection models for animal species detection, with a particular focus on mitigating wildlife-human conflicts (WHCs) and wildlife-vehicle collisions (WVCs). A comprehensive review and comparative analysis of various models, including regular and deformable R-CNN, Fast R-CNN, Faster R-CNN, and Mask R-CNN were conducted. The proposed deformable Mask R-CNN emerged as an suitable choice for accurate animal species detection due to its superior performance and capability to handle geometric variations of objects. An automated image labelling and annotation system was developed, utilizing the proposed deformable Mask R-CNN model to label, segment, and annotate six North American wildlife species (bear, cougar, deer, elk, moose, and mountain goat), along with two additional classes (human and vehicle) in various formats, thereby saving time and cost by reducing the need for human intervention. Despite the high accuracy of the deformable Mask R-CNN model, a faster model is needed to develop WHCs and WVCs mitigation systems on embedded devices.

Building on this need, we proposed a modified version of the YOLOv2 model (M-YOLO), specifically designed for deployment on embedded devices. This model was designed to enhance the feature extraction capabilities of YOLOv2, reduce its computational complexity, and improve its ability to handle geometric transformations of animals in images. These modifications led to a 5.0% improvement in accuracy and enhanced detection of small, far, and occluded animals compared to the original YOLOv2. The modified model also outperformed YOLOv3 in accuracy and exceeded YOLOv4 in detection speed on low-power machine, highlighting its suitability for real-time animal detection on embedded devices. A prototype for a WHCs mitigation system was developed, comprising two main subsystems: a detection subsystem and a warning subsystem. The detection subsystem, based on the M-YOLO model, was deployed on a RP4B device with a web camera. The warning subsystem features a custom-developed Android application named “Be Safe”, designed to alert humans when animals are present.

Further enhancements to the M-YOLO model/detector were achieved through the integration of the MCF algorithm and two-stage pipelining-dataflow hybrid parallel processing approach. These modifications significantly reduced the processing delay and power consumption of the proposed detector, particularly on embedded systems with limited resources, without trading off the accuracy of our animal species detection system. For field applications, the proposed MCFP-YOLO model was deployed and tested on two embedded devices, the RP4B and the Jetson Nano devices. While the Jetson Nano provided faster processing, the RP4B was selected due to its lower power consumption and a balanced cost-performance ratio, making it particularly suitable for extended use in remote areas.

8.2 Contributions

The contributions of this dissertation can be summarized as follows:

1. **Dataset creation**: A new dataset consisting of 201,759 labelled day/night images of six North American animal species (bear, cougar, deer, elk, moose, and mountain goat) captured from various locations and angles was curated to address the scarcity of such datasets. This dataset has been collected from several sources, including Google, YouTube, and the BCMoTI. We plan to make it publicly available, allowing researchers and biologists to use it in various applications within the field. By providing this labelled dataset, our aim is to contribute to advancements in computer vision research and facilitate the development of more effective methods for wildlife monitoring, mitigation, and conservation efforts.
2. **Deformable Mask R-CNN**: A new detection model, named the deformable Mask R-CNN, was proposed, utilizing ResNet-101 as its backbone network. This model achieved a high accuracy rate of 93.3% in identifying, localizing, and segmenting animal species. The proposed model was expanded to include two additional classes (human and vehicle) and used to develop an automated labelling and annotation system. This system simplified and improved the process of image annotation, enhancing the overall quality and accuracy of object detection and segmentation models.
3. **Effective and efficient embedded detection model**: A novel detection model, M-YOLO, was developed specifically for deployment on embedded devices to be used in wildlife mitigation systems. This model underwent three architecture modifications:

i. Enhancement of extracted features: the proposed M-YOLO model addresses the geometric variations and part deformations of animals by integrating three deformable convolutional layers. This allows the model to capture more accurate representations of distinct animal features.

ii. Improvement in species differentiation: the model's capability to identify animal species with small differences is improved by incorporating low-level features for multi-level features merging. This approach enhances the model's ability to distinguish between closely related species.

iii. Complexity reduction and process acceleration: the complexity of the proposed M-YOLO model is reduced, and the detection process is accelerated without sacrificing the accuracy. This is achieved by removing two repeated 3 × 3 convolutional layers, each with 1024 filters, from the seventh block of the model's architecture.

4. **MCFP-YOLO detection model**: In addition to enhance the processing speed and ensure efficient power usage of the object detection system, two ideas were proposed and integrated into the M-YOLO model:

   i. The proposed MCF algorithm: this algorithm is integrated into the M-YOLO model to control the number of frames to be processed based on the motion activity detected within them. This integration led to a new model variant: the MCF-YOLO model. This algorithm enhances detection speed and reduces power consumption during processing.

   ii. Parallel processing technique: the MCF-YOLO model was implemented using a novel two-stage pipelining-dataflow hybrid approach, aiming to leverage the throughput advantages of pipelining while integrating the data-driven execution characteristic of dataflow paradigms, especially for conditional processing based on the MCF algorithm. In traditional pipelining, each stage processes data in a fixed sequence without any conditional decision-making based on data content. In contrast, a pure dataflow approach would allow tasks to execute more independently, relying solely on data availability without a strict sequence. Our proposed method combines the best of both, underlined by a specifically designed algorithm to allocate the model layers. The integration of both the MCF algorithm

and parallel processing approach reduces end-to-end detection delay and ensures that power is used efficiently based on the content of the frames. As a result, the proposed MCFP-YOLO model achieves a 66.5% reduction in elapsed time compared to the M-YOLO model for batch processing. Moreover, it demonstrated a 2.6 times improvement in FPS for real-time processing, making it suitable for real-time WHCs and WVCs systems on embedded devices.

5. **Applications development**: Two executable applications, LCADS and VIADS, were developed based on the proposed MCFP-YOLO animal species detector. LCADS enables real-time wildlife monitoring using live camera feeds, facilitating instant detection and response to the presence of wildlife. It can also function as a security camera for residential areas by including human and vehicle classes. LCADS can be easily integrated into existing infrastructure, whether for traffic monitoring, park management, or residential security systems to enhance situational awareness and enable more informed decision-making. On the other hand, VIADS processes pre-recorded video footage or image folders, allowing an analysis of wildlife behavior and patterns over time. This application is valuable for researchers and conservationists seeking to monitor wildlife and develop targeted strategies for conservation efforts.

In conclusion, this research successfully addressed challenges in wildlife detection and mitigation by creating a dataset, proposing an effective and efficient object detection model for embedded devices, and developing a cell phone warning and two executable applications for real-time wildlife detection and monitoring. Analyzing the proposed models' applicability across the three environments, MCF-YOLO emerged as the most fitting choice for trails due to its motion detection aligning with sporadic animal activities. For highways, the rapid detection requirements make MCFP-YOLO, with its fusion of motion detection and parallel processing, the most promising. Meanwhile, in the intricate urban scenarios, MCFP-YOLO holds potential as the top performer, though empirical testing is recommended for final validation.

8.3 Future Work

The results of this dissertation provide an encouraging foundation and a steppingstone for further exploration and advancement for real-time animal species detection systems. Several opportunities for future work have been identified.

1. **Expansion of animal species and datasets**: Although the current model has shown promising results, it was trained and evaluated on a limited number of animal species. Future work could extend the application of the model to a wider range of animal species. This may involve collecting and curating additional datasets that include more diverse animal species, in various settings and conditions.
2. **Handling of low-quality images**: The proposed MCFP-YOLO model could be further enhanced to improve the detection accuracy in images with poor texture information and low resolution. This would be particularly beneficial for real-world applications where high-quality images might not always be available.
3. **Optimization techniques**: The integration of additional optimization techniques could potentially enhance the model's performance. Future studies could investigate other feature extraction methods, advanced training strategies, and efficient computational techniques to further improve detection speed and accuracy.
4. **Performance on various embedded devices**: While this dissertation evaluated the proposed MCFP-YOLO model's performance on RP4B and Jetson Nano devices, it would be valuable to test it on other embedded devices, including those with newer AI chips. The emergence of these advanced AI chips could significantly impact both current and future research work. They may offer enhanced processing capabilities, improved power efficiency, and better support for complex AI models like MCFP-YOLO. Evaluating the model on a variety of hardware platforms, especially those equipped with the latest AI-focused hardware, would ensure the model's effectiveness and efficiency across a broader range of technologies. This exploration is crucial for understanding how newer technological advancements can be leveraged to optimize performance and expand the applicability of our research in real-world scenarios.
5. **Integration with real-time mitigation systems**: As our ultimate goal is to mitigate WHCs and WVCs, future research should focus on the integration of the developed model into real-time mitigation systems. This could involve the development of warning systems, automatic control systems for vehicles, or other innovative solutions to minimize the impact of these encounters. One key research objective is to propose and implement robust real-time mitigation systems, complete with a cell phone warning application. Essential methodologies will include:

- **Direction tracking:** For both WVCs and WHCs mitigation systems, the direction of detected animals will be tracked to assess the likelihood of encounters. This information will be crucial to alert users via a reliable warning system.
- **Real-time distance estimation**: An algorithm for real-time distance estimation between detected objects and the camera, or between various detected objects, will be proposed. This will enable more precise warnings to users about potential wildlife interactions on highways and trails.

6. **Power Consumption Enhancement**: As the proposed MCFP-YOLO model is designed for use in low-power, computationally limited devices, additional work could focus on enhancing the model's power consumption. This could include investigating lower-power architectures or developing more efficient processing techniques.

In summary, the path forward is geared towards enhancing the current model and expanding its capabilities, leading to safer human-animal interactions, and contributing to wildlife conservation efforts. By achieving these future research directions, we are committed to facilitating harmonious coexistence between humans and wildlife.

As part of our future work, we plan to address the privacy issues in wildlife monitoring and other surveillance applications, inherently carries the risk of infringing on privacy, especially when the technology is applied in areas beyond remote wildlife habitats. This will involve developing and implementing stringent data handling protocols and possibly incorporating privacy-preserving technologies, such as anonymization techniques or secure data processing methods. It is essential to ensure that the deployment of such advanced detection models complies with privacy regulations and ethical guidelines, especially in scenarios where there is a potential overlap with human activities.

References

[1] D. C. Wilkins, K. M. Kockelman and N. Jiang, "Animal-vehicle collisions in Texas: How to protect travelers and animals on roadways," Accident Analysis and Prevention, pp. 157-170, 2019.

[2] A. L. W. Schwartz, F. M. Shilling and S. E. Perkins , "The value of monitoring wildlife roadkill," European Journal of Wildlife Research, vol. 66, no. 18, pp. 1-12, 2020.

[3] S. R. Meister, M. M. Hing, W. G. Vanlaar and R. D. Robertson, "Road safety monitor 2014: Driver behaviour and wildlife on the road in Canada," Traffic Injury Research Foundation, Ottawa, Ontario, Canada, 2016.

[4] G. Vanlaar, H. Barrett, M. a Hing, S. Brown and R. Robertson, "Canadian wildlife-vehicle collisions: An examination of knowledge and behavior for collision prevention," Journal of Safety Research, vol. 68, pp. 181-186, 2019.

[5] "ICBC Statistics," 3 April 2021. [Online]. Available: https://public.tableau.com/app/profile/icbc/viz/QuickStatistics Crashesinvolving/CrashesInvolving. [Accessed 17 September 2021].

[6] "WILDLIFE COLLISION PREVENTION PROGRAM," [Online]. Available: https://www.wildlifecollisions.ca/wildlife-vehicle-collision-facts.htm. [Accessed 17 09 2021].

[7] L. Seilecki, "Wildlife accident reporting and mitigation in British Columbia: Special annual report," Enviromental Management Section, Engineering Branch, British Columbia Ministry of Transportation and Infrastructure, Victoria, BC, Canada, 2010.

[8] P. Engineering, "WILDLIFE DETECTION SYSTEM," ACEC Canada Awards 2017, Vancouver, 2017.

[9] A. P. Clevenger, M. A. Sawaya and E. L. Landguth, "Mitigating multi-species mortality and fragmentation on the Trans-Canada Highway through Mount Revelstoke and Glacier National Parks," Parks Canada Agency, Revelstoke, British Columbia, 2014.

[10] "Trail Times," 13 03 2019. [Online]. Available: https://www.trailtimes.ca/news/animals-involved-in-11000-vehicle-collisions-annually-across-b-c/. [Accessed 17 09 2021].

[11] V. Penteriani, M. d. M. Delgado, F. Pinchera, J. Naves, A. Fernández-Gil, I. Kojola, S. Härkönen, H. Norberg, J. Frank, J. M. Fedriani, V. Sahlén, O.-G. Støen, J. E. Swenson, P. Wabakken, M. Pellegrini, S. Herrero and J. V. López-Bao, "Human behaviour can trigger large carnivore attacks in developed countries," scientific reports, vol. 6, no. 20552, pp. 1-4, 2016.

[12] "Wikipedia," List of fatal bear attacks in North America, 07 2021. [Online]. Available: https://en.wikipedia.org/wiki/List_of_fatal_bear_attacks_in_North_America#2020s. [Accessed 21 09 2021].

[13] "Wikipedia," List of fatal cougar attacks in North America, [Online]. Available: https://en.wikipedia.org/wiki/List_of_fatal_cougar_attacks_in_North_America#2010s. [Accessed 21 09 2021].

[14] K. McNamee and M. W. Finkelste, "National Parks of Canada," The Canadian Encyclopedian, 17 01 2021. [Online]. Available: https://www.thecanadianencyclopedia.ca/en/article/national-parks-of-canada. [Accessed 21 09 2021].

[15] B. Abrahms, "Human-wildlife conflict under climate change," Science, vol. 373, no. 6554, pp. 484-485, 2021.

[16] "Wildlife expert says animal encounters a risk as visitors return to Alberta parks," CBC News, 18 05 2020. [Online]. Available: https://www.cbc.ca/news/canada/calgary/alberta-parks-animal-wildlife-return-1.5574698. [Accessed 21 09 2021].

[17] E. A. Silva-Rodríguez, N. Gálvez, G. J. F. Swan, J. J. Cusack and D. Moreira-Arce, "Urban wildlife in times of COVID-19: What can we infer from novel carnivore records in urban areas?," Science of The Total Environment, vol. 765, pp. 1-7, 2021.

[18] L. SAHAGÚN, "Coyotes, falcons, deer and other wildlife are reclaiming L.A. territory as humans stay at home," Los Angeles Times, 21 04 2020. [Online]. Available: https://www.latimes.com/environment/story/2020-04-21/wildlife-thrives-amid-coronavirus-lockdown. [Accessed 30 09 2021].

[19] G. Bombieri, M. d. M. Delgado, L. F. Russo, P. J. Garrote, J. V. López-Bao, J. M. Fedriani and V. Penteriani, "Patterns of wild carnivore attacks on humans in urban areas," Scientific Reports, vol. 8, pp. 1-9, 2018.

[20] K. Firth, E. Gray and C. Sandborn, "Reform Proposals for Managing Human-Wildlife Conflict in British Columbia," Raincoast Conservation Foundation, Victoria, 2019.

[21] "CTV NEWS Vancouver," CTV NEWS Vancouver, 09 09 2021. [Online]. Available: https://bc.ctvnews.ca/unforgivable-average-of-2-4-black-bears-killed-daily-by-b-c-conservation-officers-in-august-1.5578532. [Accessed 28 09 2021].

[22] "British Columbia," Human-Wildlife Conflict, 08 2021. [Online]. Available: https://www2.gov.bc.ca/gov/content/environment/plants-animals-ecosystems/wildlife/human-wildlife-conflict. [Accessed 28 09 2021].

[23] J. Boivin, "Bold bear breaks into West Kootenay family home," Kamloops News, 21 10 2021. [Online]. Available: https://infotel.ca/newsitem/bold-bear-breaks-into-west-kootenay-family-home/it86563?fbclid=IwAR0vOp1PmUbGKIkY_k2lL_NfTUdSX-FxdCUWZ25ufhWGBrcnmxRAizEy4os. [Accessed 25 10 2021].

[24] "TERRACE STANDARD," TERRACE STANDARD, 29 08 2021. [Online]. Available: https://www.terracestandard.com/video/video-homeowner-not-concerned-after-cougar-spotted-jumping-gate-of-b-c-home/. [Accessed 27 09 2021]. With Permission.

[25] "CTV NEWS Vancouver," CTV NEWS Vancouver, 31 08 2021. [Online]. Available: https://www.ctvnews.ca/video?playlistId=1.5570997. [Accessed 27 09 2021]. With Permission.

[26] "Mitigations," WILDLIFE COLLISION PREVENTION PROGRAM, 2021. [Online]. Available: https://www.wildlifecollisions.ca/prevention/mitigation.htm#. [Accessed 01 10 2021].

[27] A. Benten, T. Hothorn, T. Vor and C. Ammer, "Wildlife warning reflectors do not mitigate wildlife–vehicle collisions on roads," Accident Analysis & Prevention, vol. 120, pp. 64-73, 2018.

[28] G. Hessa, "British Columbia Urban Ungulate Conflict Analysis," Btitish Columbia Ministry of Enviroment, Kamloops, 2010.

[29] P. J. Nyhus, "Human–Wildlife Conflict and Coexistence," Annual Review of Environment and Resources, vol. 41, pp. 143-171, 2016.

[30] M. P. Huijser, T. D. Holland, M. Blank, M. C. Greenwood, P. T. McGowen, B. Hubbard and S. Wang, "The Comparison of Animal Detection Systems in a Test-Bed: A Quantitative Comparison of System Reliability and Experiences with Operation and Maintenance," Western Transportation Institute, Helena, Montana State, 2009.

[31] M. Huijser, J. Duffield, A. Clevenger, R. Ament and P. McGowen, "Cost–Benefit Analyses of Mitigation Measures Aimed at Reducing Collisions with Large Ungulates in the United States and Canada: a Decision Support Tool," Ecology and Society, vol. 14, no. 2, 2009.

[32] T. Rytwinski, K. Soanes, J. A. G. Jaege, L. Fahrig, C. S. Findlay, J. Houlahan, R. v. d. Ree and E. A. v. d. Grift, "How Effective Is Road Mitigation at Reducing Road-Kill? A Meta-Analysis," PLos one, vol. 11, no. 11, pp. 1-25, 2016.

[33] E.-K. I. t. t. M. o. Transportation, "Environmental Guide for Mitigating Road Impacts to Wildlife," Ministry of Transportation, Catharines, Ontario, 2016.

[34] T. Clevenger, B. L. Cypher, A. Ford, M. Huijser and B. F. Leeson, "Wildlife vehicle collision reduction study," US Department of Transportation Federal Highway Adminstration, Montana state, 2008.

[35] M. P. Huijser, P. T. McGowen, W. Camel, A. Hardy, P. Wrigh and A. P. Clevenger, "Animal Vehicle Crash Mitigation Using Advanced Technology. Phase I: Review, Design And Implementation," SPR, 2006.

[36] M. P. Huijser, T. D. Holland, A. V. Kociolek, A. M. Barkdoll and J. D. Schwalm, "Animal-Vehicle Crash Mitigation Using Advanced Technology. Phase Ii: System Effectiveness And System Acceptance," SPR, 2009

[37] M. Gonzalez-de-Soto, R. Mora, J. A. Martín-Jiménez and D. Gonzalez-Aguilera, "A New Roadway Eventual Obstacle Detection System Based on Computer Vision," Sensors, vol. 20, no. 18, pp. 1-27, 2020.

[38] R. Kays, S. Tilak, B. Kranstauber, P. A. Jansen, C. Carbone, M. J. Rowcliffe, T. Fountain, J. Eggert and Z. He, "Monitoring wild animal communities with arrays of motion sensitive camera traps," arXiv:1009.5718v1 [cs.NI], pp. 1-22, 2010.

[39] A.-J. Garcia-Sanchez, F. Garcia-Sanchez, F. Losilla, P. Kulakowski, J. Garcia-Haro, A. Rodríguez, J.-V. López-Bao and F. Palomares , "Wireless Sensor Network Deployment for Monitoring Wildlife Passages," Sensors, vol. 10, no. 8, pp. 7236-7262, 2010.

[40] R. J. Lennox, K. Aarestrup, S. J. Cooke, P. D. Cowley, Z. D. Deng and e. a. , "Envisioning the Future of Aquatic Animal Tracking: Technology, Science, and Application," BioScience, vol. 67, no. 10, pp. 884-896, 2017.

[41] M. P. Huijser and P. T. McGowen, "Overview of animal detection and animal warning systems in North America and Europe," Road Ecology Center, pp. 368-382, 2003.

[42] A. Patrovsky and E. M. Biebl, "Microwave sensors for detection of wild animals during pasture mowing," Advances in Radio Science, vol. 3, pp. 211-217, 2005.

[43] M. P. Huijser and P. T. McGowen, "Overview of animal detection and animal warning systems in north america and europe," Western Transportation Institute, Montana State University, 2003.

[44] D. J. Welbourne, A. W. Claridge, D. J. Paull and A. Lambert, "How do passive infrared triggered camera traps operate and why does it matter? Breaking down common misconceptions," Remote Sensing in Ecology and Conservation, vol. 2, no. 2, pp. 77-83, 2016.

[45] D. L. Hughson, N. W. Darby and J. D. Dungan, "Comparison of motion-activated cameras for wildlife investigations," California Fish and Game, vol. 96, no. 2, pp. 101-109, 2010.

[46] N. Sundaram and S. D. Meena, "Integrated animal monitoring system with animal detection and classification capabilities: a review on image modality, techniques, applications, and challenges," Artificial Intelligence Review, pp. 1-51, 2023.

[47] S. Li, J. Li, H. Tang, R. Qian and W. Lin, "ATRW: A Benchmark for Amur Tiger Re-identification in the Wild," in In Proceedings of the 28th ACM International Conference on Multimedia, 2020.

[48] A. Khosla, N. Jayadevaprakash, B. Yao and F.-F. Li, "Novel Dataset for Fine-Grained Image Categorization: Stanford Dogs," in In Processing CVPR Workshop on Fine-Grained Visual Categorization (FGVC), 2011.

[49] O. M. Parkhi, A. Vedaldi, A. Zisserman and C. V. Jawahar, "Cats and Dogs," in IEEE Conference on Computer Vision and Pattern Recognition, 2012.

[50] J. Parham, C. Stewart, J. Crall, D. Rubenstein and J. Holmberg, "An Animal Detection Pipeline for Identification," in IEEE Winter Conference on Applications of Computer Vision (WACV), 2018.

[51] B. Mahesh, "Machine Learning Algorithms - A Review," International Journal of Science and Research (IJSR), vol. 9, pp. 381-386, 2020.

[52] P. Dognin, I. Melnyk, Y. Mroueh, I. Padhi, M. Rigotti, J. Ross, Y. Schiff, R. A. Young and B. Belgodere, "Image Captioning as an Assistive Technology: Lessons Learned from VizWiz 2020 Challenge," Journal of Artificial Intelligence Research, vol. 1, pp. 1-15, 2021.

[53] B. Wilson, J. Hoffman and J. Morgenstern, "Predictive Inequity in Object Detection," arXiv:1902.11097 [cs.CV], pp. 1-13, 2019.

[54] J. Lopes, T. Guimarães and M. F. Santos, "Predictive and Prescriptive Analytics in Healthcare: A Survey," Procedia Computer Science, vol. 170, pp. 1029-1034, 2020.

[55] S. Poornima and M. Pushpalatha, "A survey on various applications of prescriptive analytics," International Journal of Intelligent Networks, vol. 1, pp. 76-84, 2020.

[56] S. Z. Mishu and S. M. Rafiuddin, "Performance analysis of supervised machine learning algorithms for text classification," in International Conference on Computer and Information Technology (ICCIT), 2016.

[57] D. Bazazeh and R. Shubair, "Comparative Study of machine learning algorithms for breast cancer detection and diagnosis," in International Conference on Electronic Devices, Systems and Applications (ICEDSA), 2016.

[58] D. Li, L. Deng and Z. Cai , "Design of traffic object recognition system based on machine learning," Neural Computing and Applications, vol. 33, p. 8143–8156, 2020.

[59] Q. Zhang, L. T. Yang, Z. Chen and P. Li, "A survey on deep learning for big data," Information Fusion, vol. 42, pp. 146-157, 2018.

[60] F. Xing, Y. Xie, H. Su, F. Liu and L. Yang, "Deep Learning in Microscopy Image Analysis: A Survey," IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 10, pp. 4550-4568, 2018.

[61] G. Ciaparrone, F. L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri and F. Herrera, "Deep learning in video multi-object tracking: A survey," Neurocomputing, vol. 381, pp. 61-88, 2020.

[62] Q. Zhou and X. He, "Broad Learning Model Based on Enhanced Features Learning," IEEE ACCESS , vol. 7, p. 42536-42550, 2019.

[63] D. Zhang, W. Ding, C. Liu, H. Wang and B. Zhang, "Modulated Autocorrelation Convolution Networks for Automatic Modulation Classification Based on Small Sample Set," IEEE Access, vol. 8, pp. 27097-27105, 2020.

[64] D. S. a. T. Laboratory, Machine Learning with Limited Data, UK: Dstl biscuit books, 2020.

[65] S. Bonte, I. Goethals and R. V. Holen, "Machine learning based brain tumour segmentation on limited data using local texture and abnormality," Computers in Biology and Medicine, vol. 98, pp. 39-47, 2018.

[66] A. Vabalas, E. Gowen, E. Poliakoff and A. J. Casson, "Machine learning algorithm validation with a limited sample size," PLOS one, vol. 14, no. 11, pp. 1-20, 2019.

[67] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald and E. Muharemagic, "Deep learning applications and challenges in big data analytics," Journal of Big Data, vol. 2, no. 1, pp. 1-20, 2015.

[68] A. L. Samuel, "Some studies in machine learning using the game of checkers," IBM Journal of research and development, vol. 3, no. 3, pp. 210-229, 1959.

[69] T. Georgiou, Y. Liu, W. Chen and M. Lew , "A survey of traditional and deep learning-based feature descriptors for high dimensional data in computer vision," International Journal of Multimedia Information Retrieval, vol. 9, pp. 135-170, 2020.

[70] D. Pandey, K. Niwaria and B. Chourasia, "Machine Learning Algorithms: A Review," International Research Journal of Engineering and Technology, vol. 6, no. 2, pp. 916-922, 2019.

[71] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in IEEE Conference on Computer Vision and Pattern Recognition, 2009.

[72] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, "Microsoft COCO: common objects in context," in European Conference on Computer Vision (ECCV), 2014.

[73] Y. Lai, "A Comparison of Traditional Machine Learning and Deep Learning in Image Recognition," Journal of Physics: Conference Series, 2019.

[74] C. Janiesch, P. Zschech and K. Heinrich , "Machine learning and deep learning," Electronic Markets, vol. 31, p. 685-695, 2021.

[75] G. E. HINTON and R. R. SALAKHUTDINOV, "Reducing the Dimensionality of Data with Neural Networks," Science, vol. 313, no. 5786, pp. 504-507, 2006.

[76] J. Ahmad, H. Farman and Z. Jan, "Deep learning methods and applications," in In Deep Learning: Convergence to Big Data Analytics, Singapore, 2019.

[77] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Advances in neural information processing systems , vol. 25, pp. 1097-1105, 2012.

[78] X. Liu, Z. Deng and Y. Yang , "Recent progress in semantic image segmentation," Artificial Intelligence Review , vol. 52, pp. 1089-1106, 2018.

[79] D. Palaz, M. M. Doss and R. Collobert, "Convolutional Neural Networks-based continuous speech recognition using raw speech signal," in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015.

[80] Y. LeCun, Y. Bengio and G. Hinton , "Deep learning," Nature, vol. 521, pp. 436-444, 2015.

[81] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," Computers and Electronics in Agriculture, vol. 147, pp. 70-90, 2018.

[82] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in European Conference on Computer Vision (ECCV), Lecture Notes in Computer Science, 2014.

[83] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv:1409.1556 [cs.CV], pp. 1-14, 2014.

[84] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[85] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[86] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn and A. Zisserman, "The Pascal Visual Object Classes (VOC) challenge," International Journal of Computer Vision, vol. 88, no. 2, pp. 303-338, 2010.

[87] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang and C. Liu, "A Survey on Deep Transfer Learning," in International Conference on Artificial Neural Networks (ICANN), 2018.

[88] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," in International Conference on Engineering and Technology (ICET), 2017.

[89] T. Wang, D. J. Wu, A. Coates and A. Y. Ng, "End-to-End Text Recognition with Convolutional Neural Networks," in In Proceedings of the international conference on pattern recognition (ICPR), 2012.

[90] A. M. Badshah, J. Ahmad, N. Rahim and S. W. Baik, "Speech emotion recognition from spectrograms with deep convolutional neural network," in International Conference on Platform Technology and Service (PlatCon), 2017.

[91] M. Oquab, L. Bottou, I. Laptev and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

[92] R. Yamashita, M. Nishio, R. K. G. Do and K. Togashi , "Convolutional neural networks: an overview and application in radiology," Insights Imaging, vol. 9, p. 611-629, 2018.

[93] E. Göçeri, "Convolutional Neural Network Based Desktop Applications to Classify Dermatological Diseases," in IEEE 4th International Conference on Image Processing, Applications and Systems (IPAS), 2020.

[94] C. E. Nwankpa, W. Ijomah, n. Gachagan and S. Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning," arXiv:1811.03378v1 [cs.LG], pp. 1-20, 2018.

[95] C. M. Bishop, Pattern recognition, and machine learning, vol. 128, Springer, 2006, pp. 1-58.

[96] D. Jifeng, Q. Haozhi, X. Yuwen, L. Yi, Z. Guodong, H. Han and W. Yichen, " Deformable convolutional networks," arXiv:1703.06211v3 [cs.CV],pp. 1-12, 2017.

[97] X. Zhu, H. Hu, S. Lin and J. Dai, "Deformable ConvNets v2: More Deformable, Better Results," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[98] P. Felzenszwalb, R. Girshick, D. McAllester and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," IEEE transactions on pattern analysis and machine intelligence, vol. 32, no. 9, p. 1627-1645, 2010.

[99] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, X. Zhao and T.-K. Kim, "Multiple Object Tracking: A Literature Review," arXiv:1409.7618v4 [cs.CV], pp. 1-49, 2014.

[100] X. Beibei, W. Wensheng, F. Greg, K. Paul, G. Leifeng, C. Guipeng, T. Amy and S. Derek, "Automated cattle counting using Mask R-CNN in quadcopter vision system," In Computers and Electronics in Agriculture, vol. 171, pp. 1-12, 2020.

[101] H. Lin, X. Hong and Y. Wang, "Object Counting: You Only Need to Look at One," arXiv:2112.05993v1 [cs.CV], pp. 1-7, 2021.

[102] A. M. Hafiz and G. M. Bhat, "A Survey on Instance Segmentation: State of the art," arXiv:2007.00047v1 [cs.CV],pp. 1-28, 2020.

[103] D. Tian, Y. Han, B. Wang, T. Guan, H. Gu and W. Wei, "Review of object instance segmentation based on deep learning," Journal of Electronic Imaging, vol. 31, no. 4, pp. 1-18, 2021.

[104] K. He, G. Gkioxari, P. Dollar and R. Girshick, "Mask R-CNN," in Proceedings of the IEEE international conference on computer vision, 2017.

[105] R. J. Cintra, S. Duffner, C. Garcia and A. Leite, "Low-Complexity Approximate Convolutional Neural Networks," IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 12, pp. 5981-5992, 2018.

[106] A. K. Gupta, A. Seal , M. Prasad and P. Khanna, "Salient Object Detection Techniques in Computer Vision-A Survey," Entropy, vol. 22, no. 10, pp. 1-49, 2020.

[107] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu and M. Pietikäinen, "Deep Learning for Generic Object Detection: A Survey," International Journal of Computer Vision, vol. 128, p. 261-318, 2019.

[108] L. Wang, H. Lu, Y. Wang, M. Feng, D. Wang, B. Yin and X. Ruan, "Learning to Detect Salient Objects With Image-Level Supervision," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[109] S. S. Singh, T. T. Singh, N. G. Singh and H. M. Devi, "Global-Local Contrast Enhancement," International Journal of Computer Applications, vol. 54, no. 10, pp. 1-5, 2012.

[110] D. Das and S. Mukhopadhyay, "A Pixel Based Segmentation Scheme for Fingerprint Images," Information Systems Design and Intelligent Applications, vol. 340, pp. 439-448, 2015.

[111] Q. Li, Y. Zhou and J. Yang, "Saliency based image segmentation," in International Conference on Multimedia Technology, 2011.

[112] J. Han, L. Yang, D. Zhang, X. Chang and X. Liang, "Reinforcement Cutting-Agent Learning for Video Object Segmentation," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[113] H. Huang, L. Zhang and H.-C. Zhang, "Arcimboldo-like collage using internet images," ACM Transactions on Graphics, vol. 30, no. 6, pp. 1-8, 2011.

[114] J. Wang, S. Zhu, J. Xu and D. Cao, "The retrieval of the beautiful: Self supervised salient object detection for beauty product retrieval," in Proceedings of the 27th ACM International Conference on Multimedia, 2019.

[115] S. Lee, S. Kwak and M. Cho, "Universal Bounding Box Regression and Its Applications," arXiv:1904.06805v1 [cs.CV], pp. 1-14, 2019.

[116] J. Nascimento and J. Marques, "Performance evaluation of object detection algorithms for video surveillance," in IEEE Transactions on Multimedia, vol. 8, no. 4, pp. 761-774, 2006.

[117] X. Chen, H. Ma, J. Wan, B. Li and T. Xia, "Multi-View 3D Object Detection Network for Autonomous Driving," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[118] X. Du, M. H. Ang, S. Karaman and D. Rus, "A General Pipeline for 3D Detection of Vehicles," in in IEEE International Conference on Robotics and Automation (ICRA), 2018.

[119] A. Alahi, M. Bierlaire and P. Vandergheynst, "Robust real-time pedestrians detection in urban environments with low-resolution cameras," Transportation research part C: emerging technologies, vol. 39, pp. 113-128, 2014.

[120] L. Zhang, L. Lin, X. Liang and K. He, "Is faster r-cnn doing well for pedestrian detection?," in European Conference on Computer Vision (ECCV), 2016.

[121] R. Kuwana, Y. Ariji, M. Fukuda, Y. Kise, M. Nozawa, C. Kuwada, C. Muramatsu, A. Katsumata, H. Fujita and E. Ariji, "Performance of deep learning object detection technology in the detection and diagnosis of maxillary sinus lesions on panoramic radiographs," Dentomaxillofacial Radiology (DMFR), vol. 50, no. 1, pp. 1-6, 2020.

[122] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. d. Laak, B. v. Ginneken and C. I. Sánchez, "A survey on deep learning in medical image analysis," Medical Image Analysis, vol. 24, pp. 60-88, 2017.

[123] S. Wan and S. Goudos, "Faster R-CNN for multi-class fruit detection using a robotic vision system," Computer Networks, vol. 168, pp. 1-19, 2020.

[124] I. Endres and D. Hoiem, "Category Independent Object Proposals," in European Conference on Computer Vision (ECCV), 2010.

[125] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers and A. W. M. Smeulders , "Segmentation As Selective Search for Object Recognition," International Journal of Computer Vision, vol. 104, p. 154-171, 2013.

[126] B. Alexe, T. Deselaers and V. Ferrari, "Measuring the Objectness of Image Windows," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 11, pp. 2189-2202, 2012.

[127] X. Wang, M. Yang, S. Zhu and Y. Lin, "Regionlets for Generic Object Detection," in IEEE International Conference on Computer Vision, 2013.

[128] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei , "ImageNet Large Scale Visual Recognition challenge," International journal of computer vision, vol. 115, no. 3, pp. 211-252, 2015.

[129] I. H. Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions," SN Computer Science, vol. 2, no. 420, pp. 1-20, 2021.

[130] "Google Trends," Google, [Online]. Available: https://trends.google.com/trends/explore?date=2017-01-01%202021-12-30&q=handcrafted%20features,deep%20features. [Accessed 10 02 2022].

[131] Y. He, C. Zhu, J. Wang, M. Savvides and X. Zhang, "Bounding Box Regression with Uncertainty for Accurate Object Detection," in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[132] X. Zou, "A Review of Object Detection Techniques," in International Conference on Smart Grid and Electrical Automation (ICSGEA), 2019.

[133] M. Everingham, S. M. A. Eslami, L. V. Gool, C. K. I. Williams, J. Winn and A. Zisserman , "The Pascal Visual Object Classes Challenge: A Retrospective," International Journal of Computer Vision (IJCV), vol. 111, no. 1, pp. 98-136, 2015.

[134] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

[135] R. Girshick, "Fast R-CNN," in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2015.

[136] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in 28th Neural Information Processing Systems Conference (NIPS), vol. 28, pp. 91-99, 2015.

[137] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers and A. W. M. Smeulders , "Selective Search for Object Recognition," International Journal of Computer Vision, vol. 104, p. 154-171, 2013.

[138] S. Ding, X. Zhang, Y. An and Y. Xue, "Weighted linear loss multiple birth support vector machine based on information granulation for multi-class classification," Pattern Recognition, vol. 67, no. C, pp. 32-46, 2017.

[139] S. Prokudin, D. Kappler, S. Nowozin and P. Gehler, "Learning To Filter Object Detections," in German Conference Pattern Recognition (GCPR) , Basel, Switzerland, 2017.

[140] H. Wu, J. P. Siebert and X. Xu, "Fully Convolutional Networks for automatically generating image masks to train Mask R-CNN," arXiv:2003.01383 [cs.CV], pp. 1-7, 2020.

[141] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[142] S. Schneider, G. W. Taylor and S. Kremer, "Deep learning object detection methods for ecological camera trap data," in 15th Conference on Computer and Robot Vision (CRV), 2018.

[143] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017.

[144] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," in Computer Vision and Pattern Recognition, Berlin/Heidelberg, Germany, 2018.

[145] J. Redmon, "Darknet: Open Source Neural Networks in C (2013-2016)," [Online]. Available: https://pjreddie.com/darknet/. [Accessed 03 12 2021].

[146] Y.-Q. Huang, J.-C. Zheng, S.-D. Sun, C.-F. Yang and J. Liu, "Optimized YOLOv3 Algorithm and Its Application in Traffic Flow Detections," Applied Sciences, vol. 10, no. 9, pp. 1-15, 2020.

[147] M.-T. Pham, L. Courtrai, C. Friguet, S. Lefèvre and A. Baussard, "YOLO-Fine: One-Stage Detector of Small Objects Under Various Backgrounds in Remote Sensing Images," Remote Sensing, vol. 15, no. 12, pp. 1-26, 2020.

[148] L. Xiao, P. Zhou, K. Xu and X. Zhao, "Multi-Directional Scene Text Detection Based on Improved YOLOv3," Sensors, vol. 21, no. 14, pp. 1-12, 2021.

[149] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," in arXiv:2004.10934 [cs.CV], pp. 1-17, 2020.

[150] F. Gebali, Algorithms and parallel computing (1st edition), Wiley, 2011.

[151] A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, 10th Edition, John Wiley & Sons, Inc..

[152] M. S. Norouzzadeh, A. Nguyen, M. Kosmala, A. Swanson, M. S. Palmer, C. Packer and J. Clune, "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning," Proceedings of the National Academy of Sciences (PNAS), vol. 115, no. 25, pp. E5716-E5725, 2018.

[153] M. Willi, R. T. Pitman, A. W. Cardoso, C. Locke, A. Swanson, A. Boyer, M. Veldthuis and L. Fortson, "Identifying animal species in camera trap images using deep learning and citizen science," Methods in Ecology and Evolution, vol. 10, no. 1, pp. 80-91, 2018.

[154] M. S. Norouzzadeh, D. Morris, S. Beery, N. Joshi, N. Jojic and J. Clune, "A deep active learning system for species identification and counting in camera trap images," arXiv:1910.09716v1 [cs.LG], pp. 1-15, 2019.

[155] M. A. Tabak, M. S. Norouzzadeh, D. W. Wolfson, S. J. Sweeney, K. C. Vercauteren, N. P. Snow, J. M. Halseth, P. A. Di Salvo, J. S. Lewis and M. D. e. a. White, "Machine learning to classify animal species in camera trap images: Applications in ecology," Methods in Ecology and Evolution, vol. 10, no. 4, pp. 585-590, 2018.

[156] H. Yousif, J. Yuan, R. Kays and Z. He, "Fast human-animal detection from highly cluttered camera-trap images using joint background modeling and deep learning

classification," in IEEE International Symposium on Circuits and Systems (ISCAS), 2017.

[157] C. Zhu, T. H. Li and G. Li, " Towards automatic wild animal detection in low quality camera-trap images using two channeled perceiving residual pyramid networks," in IEEE International Conference on Computer Vision Workshops (ICCVW), 2017.

[158] S. Greenberg, "Automated Image Recognition for Wildlife Camera Traps: Making it Work for You," University of Calgary, Calgary, Alberta, Canada, pp. 1-15, 2020.

[159] M. Favorskaya and A. Pakhirka, "Animal species recognition in the wildlife based on muzzle and shape features using joint CNN," *Procedia Computer Science,* vol. 159, pp. 933-942, 2019.

[160] J. Parham and C. Stewart, "Detecting plains and grevy's zebras in the real-world," in IEEE Winter Applications of Computer Vision Workshops (WACVW), 2016.

[161] A. Khan and S. Khan, "YOLOv2 for Pigs Detection in Industrial Farming," Preprints 2020, 2020090088, 2020.

[162] S. Leorna and T. Brinkman, "Human vs. machine: Detecting wildlife in camera trap images," Ecological Informatics, vol. 72, pp. 1-12, 2022.

[163] A. Ter-Sarkisov, R. Ross, J. Kelleher, B. Earley and M. Keane, "Beef cattle instance segmentation using fully convolutional neural network," arXiv:1807.01972 [cs.CV], pp. 1-11, 2018.

[164] X. Yu, J. Wang, R. Kays, P. A. Jansen, T. Wang and T. Huang , "Automated identification of animal species in camera trap images," EURASIP Journal on Image and Video Processing, vol. 52, pp. 1-10, 2013.

[165] C. Guobin, H. Tony X., H. Zhihai, K. Roland and F. Tavis, "Deep convolutional neural network based species recognition for wild animal monitoring," in IEEE International Conference on Image Processing (ICIP), 2014.

[166] A. G. Villa, A. Salazar and F. Vargas, "Towards automatic wild animal monitoring: identification of animal species in camera-trap images using very deep convolutional neural networks," Ecological Informatics, vol. 41, pp. 24-32, 2017.

[167] Z. Zhang, Z. He, G. Cao and W. Cao, "Animal detection from highly cluttered natural scenes using spatiotemporal object region proposals and patch verification," IEEE Transactions on Multimedia, vol. 18, no. 10, pp. 2079-2092, 2016.

[168] S. Gupta, D. Chand and I. Kavati, "Computer Vision based Animal Collision Avoidance Framework for Autonomous Vehicles," arXiv:2012.10878 [cs.CV], pp. 237-248, 2020.

[169] M. Parikh, M. Patel and D. Bhatt, "Animal Detection Using Template Matching Algorithm," International Journal of Research in Modern Engineering and Emerging Technology, vol. 1, no. 3, pp. 26-32, 2013.

[170] F. Jurie and M. Dhome, "A simple and efficient template matching algorithm," in IEEE International Conference on Computer Vision (ICCV).

[171] A. Mammeri, D. Zhou, A. Boukerche and M. Almulla, "An efficient animal detection system for smart cars using cascaded classifiers," in IEEE International Conference on Communications (ICC), 2014.

[172] K.-B. Ge, J. Wen and B. Fang, "Adaboost algorithm based on MB-LBP features with skin color segmentation for face detection," in International Conference on Wavelet Analysis and Pattern Recognition, 2011.

[173] S. U. Sharma and D. J. Shah, "A Practical Animal Detection and Collision Avoidance System Using Computer Vision Technique," IEEE Access, vol. 5, pp. 347-358, 2017.

[174] S. Matuska, R. Hudec, M. Benco, P. Kamencay and M. Zachari, "A novel system for automatic detection and classification of animal," in ELEKTRO, pp. 76-80, 2014.

[175] W. H. S. Antônio, M. D. Silva, R. S. Miani and J. R. Souza, "A Proposal of an Animal Detection System Using Machine Learning," Applied Artificial Intelligence, vol. 33, no. 13, pp. 1093-1106, 2019.

[176] S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, 2014.

[177] "Raspberry Pi," [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. [Accessed 08 12 2021].

[178] A. Saxena, D. K. Gupta and S. Singh, "An Animal Detection and Collision Avoidance System Using Deep Learning," in Advances in Communication and Computational Technology. Lecture Notes in Electrical Engineering, Singapore, 2021.

[179] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: single shot multibox detector," in European Conference on Computer Vision (ECCV), 2016.

[180] D. Adami, M. O. Ojo and S. Giordano, "Design, Development and Evaluation of an Intelligent Animal Repelling System for Crop Protection Based on Embedded Eged-AI," IEEE Access, vol. 9, pp. 132125-132139, 2021.

[181] D. Sato, A. J. Zanella and E. X. Costa, "Computational classification of animals for a highway detection system," Brazilian Journal of Veterinary Research and Animal Science, vol. 58, pp. 1-10, 2021.

[182] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang and D. Shin, "Compression of Deep Convolutional Neural Networks For Fast And Low Power Mobile Applications," in ICLR, pp. 1-16, 2016.

[183] Z. Li, H. Li and L. Meng, "Model Compression for Deep Neural Networks: A Survey," Computers, vol. 12, no. 3, pp. 1-22, 2023.

[184] R.-T. Wu, A. Singla, M. R. Jahanshahi, E. Bertino, B. J. Ko and D. Verma, "Pruning deep convolutional neural networks for efficient edge computing in condition assessment of infrastructures," Computer-Aided Civil and Infrastructure Engineering, vol. 34, no. 9, pp. 774-789, 2019.

[185] N. Tonellotto, A. Gotta, F. M. Nardini, D. Gadler and F. Silvestri, "Neural network quantization in federated learning at the edge," Information Sciences, pp. 417-436, 2021.

[186] Y. Zhao, D. Wang and L. Wang, "Convolution Accelerator Designs Using Fast Algorithms," Algorithms, vol. 12, no. 5, pp. 1-14, 2019.

[187] L. Cambuim and E. Barros, "FPGA-Based Pedestrian Detection for Collision Prediction System," Sensors, vol. 22, no. 12, pp. 1-27, 2022.

[188] S. Minakova, E. Tang and T. Stefanov, "Combining Task- and Data-Level Parallelism for High-Throughput CNN Inference on Embedded CPUs-GPUs MPSoCs," in International Conference on Embedded Computer Systems, 2020.

[189] "Labeled Information Library of Alexandria: Biology and Conservation (LILA BC)," Snapshot Serengeti, [Online]. Available: http://lila.science/datasets/snapshot-serengeti. [Accessed 27 08 2020].

[190] "Snapshot Wisconsin, A volunteer-based project for wildlife monitoring," Snapshot Wisconsin, [Online]. Available: https://dnr.wisconsin.gov/topic/research/projects/snapshot. [Accessed 01 05 2020].

[191] A. Koutsoukas, K. J. Monaghan, X. Li and J. Huan, "Deep-learning: investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data," Journal of Cheminformatics, vol. 9, no. 42, pp. 1-13, 2017.

[192] MATLAB, [Online]. Available: https://www.mathworks.com/help/vision/ug/get-started-with-the-image-labeler.html . [Accessed 15 01 2020].

[193] A. Khan, A. Sohail, U. Zahoora and A. S. Qureshi, "A Survey of the Recent Architectures of Deep Convolutional Neural Networks," Artificial intelligence review, vol. 53, no. 8, pp. 5455-5516, 2020.

[194] "Pycharm," JETBRAINS, [Online]. Available: https://www.jetbrains.com/pycharm/. [Accessed 2022 01 15].

[195] P. Henderson and V. Ferrari, "End-to-end training of object class detectors for mean average precision," arXiv:1607.03476 [cs.CV], pp. 1-15, 2017.

[196] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto and E. A. B. da Silva, "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit," Electronics, vol. 10, no. 279, pp. 1-28, 2021.

[197] "Detection and Breed Classification of Cattle Using YOLO v4 Algorithm," in International Conference on INnovations in Intelligent SysTems and Applications (INISTA), Kocaeli, Turkey, 2021.

[198] D. Sato, A. J. Zanella and E. X. Costa , "Computational classification of animals for a highway detection system," Brazilian Journal of Veterinary Research and Animal Science, vol. 58, pp. 1-10, 2021.

[199] "UA-DETRAC," [Online]. Available: https://detrac-db.rit.albany.edu/download. [Accessed 03 03 2022].

[200] "Kaggle," Human Detection Dataset, [Online]. Available: https://www.kaggle.com/constantinwerner/human-detection-dataset. [Accessed 08 03 2022].

[201] "Kaggle," Cow-images, [Online]. Available: https://www.kaggle.com/afnanamin/cow-images. [Accessed 08 03 2022].

[202] "MPII Human Pose Dataset," [Online]. Available: https://human-poase.mpi-inf.mpg.de/#download. [Accessed 08 03 2022].

[203] "Wikipedia," Graphics processing unit, [Online]. Available: https://en.wikipedia.org/wiki/Graphics_processing_unit. [Accessed 12 12 2021].

[204] M. J. Shafiee, B. Chywl , F. Li and A. Wong, "Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video," arXiv:1709.05943v1 [cs.CV], pp. 1-3, 2017.

[205] S. Ding, H. Fan, L. Liu and Y. Wang, "A Novel YOLOv3-tiny Network for Unmanned Airship Obstacle Detection," in IEEE 8th Data Driven Control and Learning Systems Conference (DDCLS), 2019.

[206] S. Kumar, D. Yadav, H. Gupta, O. P. Verma, I. A. Ansari and C. W. Ahn, "A Novel YOLOv3 Algorithm-Based Deep Learning Approach for Waste Segregation: Towards Smart Waste Management," Electronics, vol. 10, no. 1, pp. 1-20, 2020.

[207] X. Wang, S. Wang, J. Cao and Y. S. Wang, "Data-driven based tiny-YOLOv3 method for front vehicle detection inducing SPP-net," IEEE Access, pp. 110227-110236, 2020.

[208] Z. JIANG, L. ZHAO, S. LI and Y. JIA, "Real-time object detection method for embedded devices," arXiv:2011.04244v2 [cs.CV], pp. 1-11, 2020.

[209] C. Guo, X.-l. Lv, Y. Zhang and M.-l. Zhang , "Improved YOLOv4-tiny network for real-time electronic component detection," Scientific Reports, vol. 11, no. 22744, pp. 1-13, 2021.

[210] Y. Zhang, Y. Shen and J. Zhang, "An improved tiny-yolov3 pedestrian detection algorithm," Optik, vol. 183, pp. 17-23, 2019.

[211] S. Zhang, Y. Wu, C. Men and X. Li, "Tiny YOLO Optimization Oriented Bus Passenger Object Detection," Chinese Journal of Electronics, vol. 29, no. 1, pp. 132-138, 2020.

[212] H. Ji, X. Zeng, H. Li, W. Ding, X. Nie, Y. Zhang and Z. Xiao, "Human Abnormal Behavior Detection Method based on TTINY-YOLO," in Proceedings of the 5th International Conference on Multimedia and Image Processing (ICMIP), 2020.

[213] R. Huang, J. Pedoeem and C. Chen, "YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers," arXiv:1811.05588 [cs.CV], pp. 1-8, 2018.

[214] "Reconyx camera traps," Reconyx, [Online]. Available: https://www.reconyx.com/product/Professional_Series. [Accessed 2021 12 21].

[215] K. Boudjit and N. Ramzan, " Human detection based on deep learning YOLO-v2 for real-time UAV applications," Journal of Experimental & Theoretical Artificial Intelligence, pp. 1-18, 2021.

[216] X. Huang, X. Wang, W. Lv, X. Bai, X. Long, K. Deng, Q. Dang, S. Han, Q. Liu, X. Hu, D. Yu, Y. Ma and O. Yoshie, "PP-YOLOv2: A Practical Object Detector," arXiv:2104.10419 [cs.CV], pp. 1-7, 2021.

[217] M. Loey, G. Manogaran, M. H. N. Taha and N. E. M. Khalifa, "Fighting against COVID-19: A novel deep learning model based on YOLO-v2 with ResNet-50 for medical face mask detection," Sustainable cities and society, vol. 65, pp 102600-102600, 2021.

[218] J. R. Terven and D. M. Córdova-Esparza, "A Comprehensive Review of YOLO: From YOLOv1 And Beyond," arXiv:2304.00501v2 [cs.CV], pp. 1-33, 2023.

[219] J. Zhang, M. Huang, X. Jin and X. Li, "A Real-Time Chinese Traffic Sign Detection Algorithm Based on Improved YOLOv2," Algorithms, vol. 10, no. 4, pp. 1-13, 2017.

[220] E. Dong, Y. Zhu, Y. Ji and S. Du, "An Improved Convolution Neural Network for Object Detection Using YOLOv2," in IEEE International Conference on Mechatronics and Automation (ICMA), 2018.

[221] Q. Fan, L. Brown and J. Smith, "A closer look at Faster R-CNN for vehicle detection," in IEEE Intelligent Vehicles Symposium (IV), 2016.

[222] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980v9 [cs.LG], pp. 1-15, 2017.

[223] "Nvidia Developer," Nvidia, [Online]. Available: https://developer.nvidia.com/cuda-toolkit. [Accessed 12 12 2021].

[224] J. Solawetz, "Roboflow," The Ultimate Guide to Object Detection, 01 02 2021. [Online]. Available: https://blog.roboflow.com/object-detection/. [Accessed 13 01 2022].

[225] J. Dean, "The Deep Learning Revolution and Its Implications for Computer Architecture and Chip Design," in IEEE International Solid- State Circuits Conference (ISSCC), 2020.

[226] "Raspberry Pi 4 Model B," Raspberry Pi, [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. [Accessed 08 12 2021].

[227] "Electronic Design," Embedded Revolution , 16 09 2019. [Online]. Available: https://www.electronicdesign.com/technologies/embedded-revolution/article/21134806/is-raspberry-pi-the-right-platform-for-embedded-product-development. [Accessed 13 01 2022].

[228] A. A. Yahya, J. Tan and M. Hu, "A novel video noise reduction method based on PDE, adaptive grouping, and thresholding techniques," The Journal of Engineering, pp. 605-620, 2021.

[229] K. A. M. Said, A. B. Jambek and N. Sulaiman, "A Study of Image Processing Using Morphological Opening and Closing Processes," International Journal of Control Theory and Applications, vol. 9, no. 31, pp. 15-21, 2017.

[230] P. Murray and S. Marshall, "A Review of Recent Advances in the Hit-or-Miss Transform," Advances in Imaging and Electron Physics, vol. 175, pp. 221-282, 2013.

[231] A. Bovik, Handbook of image and video processing, Academic press, 2010.

[232] P. Rawat and M. D. Sawale, " Gaussian kernel filtering for video stabilization," in International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE), 2017.

[233] A. G. Chekkilla and R. V. Kalidindi, Monitoring and Analysis of CPU Utilization, Disk Throughput and Latency in servers running Cassandra database, Karlskrona: Faculty of Computing, Blekinge Institute of Technology, 2016.

[234] "Python documentation," 12 09 2022. [Online]. Available: https://docs.python.org/3/library/multiprocessing.html#synchronization-primitives.

[235] J. A. Stuart and J. D. Owens, "Efficient Synchronization Primitives for GPUs," arXiv:1110.4623v1 [cs.OS], pp. 1-12, 2011.

[236] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania and T. Mitra, "High-Throughput CNN Inference on Embedded ARM Big.LITTLE Multicore Processors," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 10, pp. 2254-2267, 2020.

[237] A. Alazahrani and F. Gebali, "Multi-Core Dataflow Design and Implementation of Secure Hash Algorithm-3," IEEE Access, vol. 6, pp. 6092-6102, 2018.

[238] "WATCH: Trio of cougars spotted in British Columbia backyard," CTV News, 02 2023. [Online]. Available: https://www.youtube.com/watch?v=4sdWeiyWZ0w&t=6s. [Accessed 25 02 2023].

[239] "Getting Started with Jetson Nano Developer Kit," Nvidia Developer, [Online]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit . [Accessed 11 04 2023].

[240] "Raspberry Pi 4 Model B Specifications," Raspberry Pi, [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. [Accessed 11 04 2023].

[241] "Embedded Systems with Jeston," Nvidia, [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/. [Accessed 6 04 2023].

[242] "Intel Products," Intel, [Online]. Available: https://www.intel.com/content/www/us/en/products/overview.html. [Accessed 6 4 2023].

[243] “WILDCAM," [Online]. Available: https://wildcams.ca/projects/. [Accessed 29 08 2023].