DEVELOPMENT OF AN ARTIFICIAL INTELLIGENCE DEEP NEURAL NETWORK FOR THE IDENTIFICATION OF INDIVIDUAL ANIMALS

PIETER STEFANUS VELDTSMAN

Thesis submitted in fulfilment of the requirements for the Degree

DOCTOR OF ENGINEERING IN ELECTRICAL ENGINEERING

in the

Department of Electrical, Electronics and Computer Engineering

Faculty of Engineering, Built Environment and Information Technology

at the

Central University of Technology, Free State

Promoter: Prof BJ Kotze (DTECH: Engineering: Electrical)

BLOEMFONTEIN

January 2026

DECLARATION OF INDEPENDENT WORK

DECLARATION WITH REGARD
TO INDEPENDENT WORK

I, PIETER STEFANUS VELDTSMAN, identity number _____ and student number _____, do hereby declare that this research project submitted to the Central University of Technology, Free State for the Degree DOCTOR OF ENGINEERING IN ELECTRICAL ENGINEERING, is my own independent work; and complies with the Code of Academic Integrity, as well as other relevant policies, procedures, rules and regulations of the Central University of Technology, Free State; and has not been submitted before to any institution by myself of any other person in fulfilment (or partial fulfilment) of the requirements for the attainment of any qualification.

2026/01/28

SIGNATURE OF STUDENT **DATE**

LIST OF PUBLICATIONS

1. P.S. Veldtsman and B.J. Kotze, " Development of an Artificial Intelligence Deep Neural Network for the Identification of Individual Animals", Tuijin Jishu/Journal of Propulsion Technology, Vol. 44 No. 3 (2023), pp. 67-76, ISSN: 1001-4055, doi: 10.52783/tjjpt.v44.i3.241 Link:https://www.propulsiontechjournal.com/index.php/journal/article/view/241
   Status: Published

DEDICATION

This thesis is dedicated to my beloved mother, whose love and encouragement have been a constant source of strength; to the memory of my late father, whose wisdom and guidance continue to inspire me; to my dear wife and children, for their unwavering support and understanding throughout this journey; and to my sister, for always believing in me and standing by my side.

ACKNOWLEDGEMENTS

I am deeply grateful to Almighty God for granting me the strength to endure and for surrounding me with supportive individuals during challenging times.

I also want to extend my heartfelt thanks to my supervisor, Prof. B.J. Kotze, for his invaluable guidance and for inspiring hope and optimism when it was most needed.

Furthermore, I would like to thank Prof. P.H. Wessels for the opportunity to gather animal data on his farm.

Lastly, I am profoundly thankful to my wife and family for their unwavering support and patience throughout the course of my studies.

ABSTRACT

The literature review highlighted a clear need for a system capable of identifying individual animals. Such a system will enable a farmer to feed specific livestock animals, such as cows, to prevent under- and overfeeding, and to track cattle for stock theft prevention and identification if found. Other current identification methods, such as ear tags and RFID tags, were investigated and considered, but not used.

This research further explored the application of deep learning, specifically the You Only Look Once (YOLO) family of models, for real-time animal identification and monitoring in automated feeding systems. Traditional animal recognition methods, such as RFID tags and manual observation, are limited by accuracy, maintenance costs, and scalability. To address these challenges, this study investigated the potential of YOLOv3 and YOLOv4 architectures on portable hardware for detecting and identifying individual cattle using image and video data.

A comprehensive review of object detection frameworks, including R-CNN and SSD, was conducted to establish YOLO's advantages in terms of speed, accuracy, and ease of deployment. The methodology involved dataset creation, preprocessing, augmentation, and training of YOLO models using Python and GPU-accelerated platforms such as the NVIDIA® Jetson Nano. Model performance was evaluated using precision, recall, and mean average precision (mAP) metrics, with results demonstrating YOLOv4's superior real-time detection capability and robustness under varying environmental conditions.

The YOLO-based detection system demonstrates strong overall performance across most classes, with the majority achieving near-perfect detection accuracy. Fourteen classes attain an Average Precision (AP) of 100% alongside consistent Area Under the Precision–Recall Curve (AUC) values of 0.75, indicating highly reliable and stable detection behaviour. Several additional classes also perform well, recording AP values above 90% with only a modest reduction in AUC, which suggests robust generalisation with minimal precision–recall degradation. However, performance declines noticeably for a small number of classes, where

AP falls to 50% and below, accompanied by significantly reduced AUC values. Three classes show no successful detections, yielding zero AP and AUC scores. Overall, these results highlight the effectiveness of the YOLO system for most target classes, while indicating that limited or challenging classes may benefit from additional training data, improved class balance, or further model optimisation.

The findings confirmed that YOLO-based vision systems can significantly enhance the automation and efficiency of livestock management by enabling accurate, non-invasive identification and behavioural monitoring. The study contributes to the advancement of intelligent feeding systems and provides a foundation for future research in precision agriculture and AI-based animal welfare monitoring.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS, ABBREVIATIONS

Artificial Intelligence (AI)

Automated Learning (AL)

Complete Intersection Over Union (CIoU)

Computer vision (CV)

Convolutional neural network (CNN)

Cross-Stage Partial Network (CSPNet)

Deep Learning (DL)

Detection transformer (DETR)

K-Nearest Neighbours (KNN)

Long short-term memory (LSTMs)

Mean average precision (mAP)

Natural language processing (NLP)

Histogram of oriented gradients (HOG)

Intersection over Union (IOU)

K-Nearest Neighbours (KNN)

Long short-term memory (LSTMs)

Machine Learning (ML)

Natural language processing (NLP)

Non-Maximum Suppression (NMS)

Open Neural Network Exchange (ONNX)

Region-based Convolutional Neural Network (Faster R-CNN)

Recurrent Neural Networks (RNNs)

Self-adversarial Training (SAT)

Single Shot Detector (SSD)

Spatial Pyramid Pooling (SPP)

Support Vector Machines (SVM)

Speeded-Up Robust features (SURF)

You Look Only Once (YOLO)

CHAPTER 1: INTRODUCTION TO THE RESEARCH

1.1 Context

South Africa is a very dry country, and often a need arises to feed farm animals. All animals will not come to feed at the same time; thus, the farmer will need to keep human observers at the feeder on a 24-hour basis to monitor the movement and feeding habits of the animals. The use of people introduces its own problems, as people might not always see the animals at night; the animals may wander off, and the observers may fall asleep. All these factors will influence the quality of the gathered data. Existing research demonstrates that numerous studies employ images and neural networks to identify animal species. However, there is currently no system designed to recognise individual animals and apply this information to the management of farm animals.

An electronic system can reduce the errors introduced into the gathered data significantly. Any system that works with animals should not disturb or cause harm to the animals while gathering the data. If the animals are disturbed, they will become stressed and may change their behaviour.

1.2 Machine vision and animal detection

According to Wäldchen and Mäder [1], "*machine learning is the fastest growing field in computer science pervading fields as diverse as marketing, health care, manufacturing, information security and transportation.*"

Identifying species can be difficult, especially when dealing with animals that live underground and in crowded areas. However, a variety of methods are being used to take pictures and videos, such as camera traps, autonomous underwater vehicles (AUVs), and low-resolution airborne thermal imaging. These methods offer important new information about the behaviours of species that were previously difficult to study. Among the various techniques, camera traps are

emerging as particularly popular and widely embraced by both professionals and amateurs, largely due to their affordability.

In essence, a camera trap encompasses a digital camera along with an infrared sensor. An animal emitting heat that is detected by the sensor will trigger the camera, activating a video recording until the object is no longer detectable. These traps may be set out in the wilderness for several weeks to months, collecting valuable data concerning the behavioural patterns of the animals. With little adverse impact on wildlife, these traps are deemed “wildlife friendly.” Additionally, the data collected is permanent and can be verified. Information gathered includes species locations, populations, and interactions. Networked camera traps are even capable of transmitting data through phone or satellite networks in near real-time, making them highly effective in combating poaching and illegal hunting.

The vast amount of data collected by camera traps can be used to train artificial intelligence systems, particularly machine learning models, to recognise species from images. In species detection, algorithms and statistical models in computer vision, machine learning, and deep learning enable systems to classify and identify images. Once trained, these systems can sort and recognise thousands of images in a matter of seconds, a task that would be impossible for humans to achieve manually [2].

The researcher believed that a vision system might be a more suitable solution to track the habits of the individual animals, by using the unique properties or characteristics of the individual animals. When collecting your own images, it is preferable to use or gather images that were taken under the same conditions as where the system will be implemented. Diverse images will also enhance accuracy [3].

1.3 You Only Look Once

The three prominent object detection models used today are You Only Look Once (YOLO), Single Shot Detector (SSD), and Region-based Convolutional Neural Network (Faster R-CNN). Faster R-CNN first proposes regions of interest and then

classifies them [4]. Faster R-CNNs give accurate outputs or results, but the two-stage process of region proposal and classification reduces speed. YOLO predicts probabilities and bounding boxes in a single pass, which makes it a one-stage object detection model. YOLO is extremely fast because it does not deal with complex pipelines [5] and can perform real-time inference, but due to the high speed, accuracy is sacrificed. SSD is also a one-stage detection and finds a balance between the speed of YOLO and the accuracy of Faster R-CNNs.

Using YOLO for identifying animals is often preferred over other object detection methods due to a combination of speed, accuracy, and real-time capabilities. Table 1-1 shows a comparison between YOLO and other object detection models.

Table 1-1: Animal Detection: Object Detection Method Comparison

Feature / Method	YOLO (v4–v8)	SSD	Faster R-CNN	DETR (Transformer)	Traditional CV (HOG, Haar, etc.)
Speed (Inference Time)	Very Fast (real-time)	Fast	Slow	Very Slow	Fast
Accuracy (mAP)	High	Moderate to High	High	Very High	Low
Real-Time Suitability	Yes	Yes	No	No	Yes (limited)
Training Time	Short to Moderate	Moderate	Long	Very Long	Very Fast
Multi-Class Detection	Yes	Yes	Yes	Yes	No or limited
Complex Backgrounds	Handles well	Decent	Good	Excellent	Poor

Feature / Method	YOLO (v4–v8)	SSD	Faster R-CNN	DETR (Transformer)	Traditional CV (HOG, Haar, etc.)
Custom Animal Detection	Easy to fine-tune	Easy to fine-tune	Requires more setup	Complex setup	Hard to generalise
Edge Device Support	Excellent (e.g. Jetson, Raspberry Pi)	Good	Limited	Poor	Good
Model Size	Small to Medium	Small to Medium	Large	Large	Small
Community Support	Large, active	Active	Active	Growing	Outdated

Shown below are reasons for choosing YOLO over other methods like R-CNN, SSD, or traditional image processing techniques for animal detection.

1.3.1. Real-Time Performance

As the name suggests ("You Only Look Once"), YOLO processes images in a single pass and is thus extremely fast. The high operation speed makes it ideal for real-time animal detection in videos or camera feeds, which is useful for wildlife monitoring, anti-poaching surveillance, smart farming or livestock management and pet detection in home security [6].

1.3.2. High Accuracy and Generalisation

YOLO is a deep learning-based object detector, which learns robust and generalisable features. It performs well in detecting animals in complex environments (e.g., forests, farms) where traditional methods struggle [7].

1.3.3. Simultaneous Detection and Classification

YOLO can detect multiple animals on a single video frame and classify them in a single step. It returns bounding boxes, class labels and confidence scores for objects like animals, in one output, making it efficient and simple to use [8].

1.3.4. Ease of Integration and Deployment

YOLO models are lightweight, easy to train or fine-tune on custom animal datasets and are supported on various platforms like Python, ONNX, and TensorRT. This makes them ideal for deploying on devices like drones, Raspberry Pi's, or mobile phones. The biggest problem when training the dataset is the time it takes to identify and find the bounding boxes for each frame of training data [9].

1.3.5. Community and Ecosystem

YOLO has a huge open-source community support, making it easier to find pre-trained models of animals, tutorials, documentation, and tools for annotation and training [10].

1.4 Why Not Use Other Methods?

Table 1-2 compares the disadvantages or drawbacks of other methods of animal identification compared to YOLO.

Table 1-2: Drawbacks of other methods of animal identification

Method	Drawbacks Compared to YOLO
R-CNN/ Fast R-CNN	Slower, requires multiple passes over image, harder to deploy
SSD	Faster than R-CNN but usually less accurate than YOLO
Traditional CV (e.g., Haar cascades, HOG+SVM)	Poor performance in natural, varied environments
Transformer-based (e.g., DETR)	More accurate but slower and harder to run on edge devices

1.5 Summary Introduction to the YOLO family

When Joseph Redmon made YOLO available in 2016, it changed the way object detection is done. As mentioned earlier, YOLO does not pick one object at a time, but all objects in a single video frame and gives results for all detected objects. It is fast and accurate. It is thus useful in applications like self-driving cars, robots, and even to keep tabs on animals.

1.5.1. Key Concept of YOLO

Unlike traditional object detectors, like R-CNN, that use region proposals, YOLO treats object detection as a single regression problem. It divides an image into a grid and directly predicts bounding boxes and class probabilities from the entire image in one evaluation.

Table 1-3 shows the evolution of the YOLO family from its inception in 2016. Over the years, many improvements in speed and accuracy were made to YOLO.

Table 1-3: Evolution of the YOLO Family

Version	Year	Highlights
YOLOv1	2015	First release: fast but less accurate.
YOLOv2 / YOLO9000	2017	Better accuracy, support for 9000+ classes.
YOLOv3	2018	Multi-scale detection, better performance.
YOLOv4	2020	Significant accuracy and speed improvements, widely used in industry.
YOLOv5	2020	Not officially from original authors, but extremely popular; easy to use, fast, and flexible.
YOLOv6 / v7	2022	Optimised for edge devices and faster inference.
YOLOv8	2023	The most advanced version; supports instance segmentation, classification,

		and detection in one model; streamlined training and deployment.

1.5.2. Why YOLO Is So Widely Used

Real-time speed makes it ideal for video feeds, drones, and other mobile applications, whilst end-to-end simplicity means detection and classification are done in one go. Transfer learning makes it easy to retrain on custom datasets and will assist in adding animals to the current trained data. Broad community support ensures that there are pretrained models, tutorials and tools available.

1.5.3. YOLO in Action: Animal Detection Example

YOLO is a sheer groundbreaking innovation [11] if you want to spot dogs, cats, deer or elephants or any animal captured on camera, in a variety of different environments. It is fast, with accurate detections and adapts to any scene you expose it to. YOLO is thus popular with wildlife enthusiasts, researchers and programmers alike. YOLO combines real-time speed, high accuracy, and ease of use, which makes it a good choice for animal detection. Applications where quick decisions or insights are critical, are well-suited for YOLO.

1.6 Problem statement

“Animal production is one of the major sectors in South African agriculture and contributes substantially to the economy of the country.” [12]. Therefore, it is crucial for farmers to cultivate at an optimal profit. According to Meisner et al. [13], seventy percent of agricultural land in South Africa can only be utilised by livestock, and game and other species are found in all provinces with high concentrations in the eastern higher rainfall regions. Livestock production in South Africa is a significant contributor to food security and clothing, as well as to the country's social and economic levels [13].

Industrial-scale production of animal feeds commenced in the late 19th century [14]. During this time, the benefits of a balanced diet and nutrition in human and animal diets were identified. The benefits and role processing of certain raw materials were also identified. Today, the nutritional needs of farm animals are

known and understood and might be satisfied through natural grazing alone, but it might be necessary to supplement the nutrients in concentrated and controlled form [15]. The quality of the feed is influenced by the nutrient content, but also by other factors like feed presentation, hygiene, digestibility, and effect on intestinal health [15], [16], [17].

Due to the differences in the digestive systems of the different livestock species, like cattle, horses as well as baby animals, it is important to provide the animals with the correct type of feed. Since the feeding of animals can be the most expensive part of farming, farmers can supplement the normal feed with other industry by-products, like distiller's grains and soybean [18], [19]. This is not only environmentally sustainable but also economically viable for the farmers.

Most stock farmers buy their feedstuffs in bulk because it is more cost-effective. Agriseta [20] identified the most important aspects of good quality control as effective record keeping. Some of the items that should be included in the record system are daily intake and usage [20]. Feeding behaviour contains important information that can enable producers to better manage livestock; similarly, researchers can benefit by better understanding the factors that influence feed intake [21].

A survey carried out on 18 farms in Switzerland, Germany, Denmark and the Netherlands indicates that an increasing number of farms are relying on automatic feeding to ease their workload, save time and achieve flexibility [22]. Various systems are available which make automated feeders and feeding possible. Rail-guided feed wagons are currently the most established in practice, but conveyor belts and self-propelled feeders are also utilised [22].

Livestock farmers suffer losses in the region of R2 billion each year due to predators, such as black-backed jackal, caracal, leopard, cheetah, brown hyena and even crows and stray dogs [23]. To investigate the reasons for disappearing animals, i.e., effective predator control, it is thus very important to know when a specific animal stops feeding.

There is currently also a need to identify animals for other reasons as well. Identification might be required for tracking of animals, to prevent the spread of disease between different regions and even cross-border movement of animals [24]. Stock theft is also a large problem in South Africa [25]. The identification of stolen animals is very important for the successful prosecution of thieves as well as the recovery of the animals.

There are different marks that can be applied to animals for identification. Some are permanent, while others are non-permanent. Non-permanent methods will include methods like cutting off the brush of the tail or colour markers, such as paint. Permanent methods include ear notching, ear tags, tattoo, hot iron branding, freeze branding and electronic markers [26]. The Draft Document for a Livestock Identification and Traceability system in South Africa [24] suggests identification methods such as ear tags and RFID tags.

“It is noteworthy that with certain breeds of cattle e.g., Nguni and Friesland, the colour markings of the animals are as unique as a thumb print” [26]. It is thus possible to record colour marking and patterns for the identification of these animals. Photos can be taken and kept for the identification of individual animals. Due to the distinct markings on animals, it should therefore be possible to identify individual animals using image processing techniques. According to Yu et al. [27], image sensors are increasingly being used in biodiversity monitoring. Every study produces thousands or millions of pictures. They also state that efficiently identifying the species captured by each image is a critical challenge for the advancement of this field. Matcher [28] believes that Artificial Intelligence (AI) can successfully be used to identify specimens to make science more accessible to everybody. AI can therefore be used to enhance the success rate as well as the reliability of image recognition.

Existing research shows that many studies make use of images and neural networks to identify species. However, there is currently no video-based system for identifying individual animals to use this information to manage farm animals.

1.7 Aim and Objectives

The study aimed to develop a deep learning artificial intelligence vision system for the identification of individual animals. If such a system can successfully be implemented, there are many applications where it can be utilised. An example would be an automatic animal feeder system. The feeding of the animals allows a farmer to track the feeding habits and observe the condition of the animals. Different quantities of food can be given to each animal. An electronic system can significantly reduce the errors introduced into the data gathered on animals, compared to traditional methods, where people are used. Figure 1.1 depicts the three objectives of the research in the development of such an electronic system.

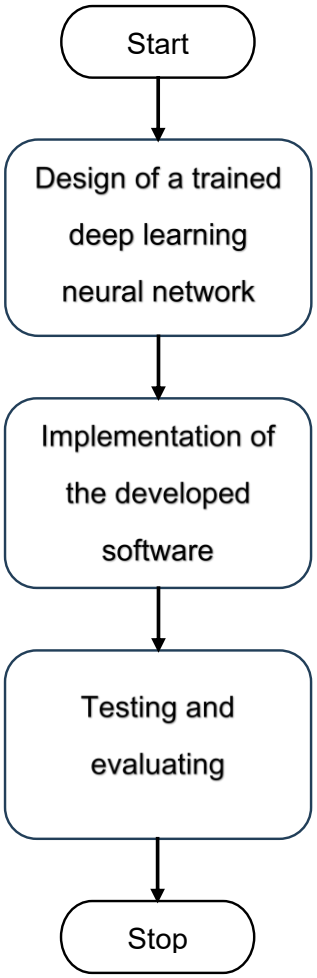


Figure 1.1: The objectives of the research

A deep learning neural network will be trained to perform “facial” recognition of animals by using computer vision and Machine Learning techniques. The second

phase is the implementation of the developed software on a suitable hardware system, such as the NVIDIA® Jetson Nano, whereafter it will be tested and the concept of individual animal identification evaluated.

1.8 Research question

This study is guided by three interrelated research questions. The first explores how a deep learning artificial intelligence vision system can be designed and implemented to accurately identify individual farm animals, focusing on the technical development and the requirements for achieving reliable recognition performance. A second research question considers the benefits that such an AI based identification system may offer when compared to traditional livestock identification methods, including potential improvements in accuracy, efficiency, and reductions in human error. The third question examines the practical applications of an AI driven vision system within livestock management, with particular attention to its possible use in areas such as feeding optimisation, behavioural monitoring, and broader herd management processes.

1.9 Limitations of the study

The limitations of this study included the following:

- Challenges in data collection: Operationally, it was difficult to implement the system; therefore, different methods of data collection were used.
- Environmental durability of equipment: The equipment was exposed to the elements, or the animals could damage the equipment.
- Hardware limitations, e.g., achieving real-time detection in field conditions where computational resources may be restricted.
- A limited training data set which narrowed the scope of the study and could affect the system's scalability and generalisability.

1.10 Contributions

The research addressed questions like:

- Is it possible to successfully implement deep learning on a portable device like a NVIDIA® Jetson Nano?

- Can individual animals be correctly identified with such a limited resource system?

In both these instances, the researcher was successful in implementing these targets.

1.11 Thesis structure overview

The structure of the dissertation is as follows:

Chapter 1 introduces the need for a vision-based system for monitoring individual animals on South African farms, where traditional monitoring methods are challenged by high labour costs and requirements, and potential inaccuracies. South Africa's dry climate often requires supplementary feeding for livestock, requiring constant supervision to ensure each animal's feeding habits are adequately tracked. However, human observers may miss important data, especially at night, and their presence can inadvertently disturb the animals. The chapter discusses how an electronic, automated system could enhance monitoring accuracy without disturbing the animals.

The chapter then explores machine vision in animal detection applications. Tools like camera traps and other image-based techniques are being used to gather wildlife data. When integrated with artificial intelligence (AI) and machine learning, these tools allow for automated species identification from images and video streams, significantly aiding researchers in analysing extensive datasets with speed and accuracy. Such systems, especially when these systems are applied in controlled environments like farms, allow for managing individual animals by their unique identifiers, such as colour patterns and horn profiles which could be used for facial recognition.

The problem statement highlights the importance of optimal livestock farming in South Africa, which is a major sector of the country's agriculture. Proper animal identification and monitoring can help prevent issues like diseases, cross-border livestock movement, and theft. Current identification techniques range from ear tags to more advanced RFID tags, but these methods are often labour-intensive and prone to errors.

The research aimed to develop a deep learning-based vision system that could identify individual animals through AI-enhanced image processing. The objectives included designing and training a deep learning neural network, implementing it on a suitable hardware platform like the NVIDIA® Jetson Nano, and evaluating its performance for individual animal identification. Such a system could automate feeding management, track individual health conditions, and reduce dependency on human management, providing economic and operational benefits to farmers.

Lastly, the research question addressed how an AI-driven vision system could be developed for identifying individual animals and explored its potential applications in livestock management. Study limitations included challenges in data collection, environmental durability of equipment, and focusing on animal identification rather than livestock theft prevention. Although implementation hurdles remained, projects like "Snapshot Serengeti" provided valuable information, indicating the feasibility of species recognition through neural networks and large-scale datasets.

In Chapter 2, the research discusses the classical methods for animal identification, whereafter it compares the innovations in feeding systems and data collection methods in animal farming, particularly to improve feed efficiency and animal monitoring. Self-feeders are presented as an effective alternative to feeding animals directly on the ground, which can result in significant feed wastage. Feed costs contribute considerably to farming expenses. Automated and self-feeding systems offer economic advantages by minimising feed losses. The chapter emphasises the role of livestock in South African agriculture, where balanced feeding is critical due to the sector’s impact on food security and rural economies.

The need for precise feeding and tracking is highlighted due to the differences in nutritional requirements across livestock species and growth stages. Automatic feeders are growing in popularity, especially in parts of Europe, where they streamline feeding routines and improve flexibility for farmers. Additionally, large-scale automated feeding systems reduce manual labour and improve record-keeping, which can enhance livestock management practices and overall farm productivity.

Animal identification is vital not only for effective feeding and health monitoring but also for managing predator threats, disease control, and reducing stock theft, which is a significant issue in South Africa. Traditional identification methods, including ear tags and branding, have limitations in durability and reliability, and advanced methods such as RFID tagging or biometric markers are discussed as alternatives. Unique features such as colour patterns can be utilised for visual identification, supporting individual monitoring through AI and image processing.

The current data collection methods are analysed, covering both traditional identification techniques and modern technologies like RFID tags, GPS tracking, and camera traps. Camera traps are widely used for wildlife monitoring but pose challenges in manual data processing, especially in poor lighting or when animals are partially visible. Biometric methods offer a promising solution, leveraging unique traits like iris patterns for individual recognition, achieving high accuracy with AI models like Convolutional Neural Networks (CNNs) in trials with Arabian horses.

The chapter concludes with insights into neural networks and their role in automating image analysis. While ANNs have successfully classified animals in images, ongoing efforts are focusing on achieving reliable individual identification within categories, which is crucial for precise monitoring and targeted interventions in smart farming. New technologies like drones and IoT devices further support data collection, though they require integration with machine learning models for meaningful application in automated livestock management. This foundational understanding of modern feeding systems and data collection tools sets the stage for deeper exploration into neural networks and machine learning in subsequent chapters.

Chapter 3 expounds upon the comprehensive design concept methodology that underpins the proposed study, providing a holistic overview of the study's architectural framework and methodology.

Chapter 4 is dedicated to presenting the study's results and conducting an in-depth discussion of the model employed. The focus was to align the research objectives through rigorous experimental procedures and meticulous analysis of the obtained results, following the research methodology.

Chapter 5 outlines foundational steps taken in the development of an animal recognition system, focusing on challenges, methods of data collection, and preliminary testing. Animal facial recognition remains underdeveloped compared to human recognition, with enhancements needed to improve accuracy, such as adding side profiles. The chapter highlights that cattle often damage cameras and other monitoring equipment due to curiosity and physical contact, which can lead to mechanical, electrical, and data loss issues, especially in outdoor environments. A key requirement for animal identification is capturing, analysing images and cataloguing high- and low-quality images, which allows the building of a database of images to distinguish individual animals. This involves capturing and utilising a deep neural network to facilitate accurate recognition. To balance data capture and recognition accuracy, the project aimed to gather comprehensive image datasets from various sources. Image processing began with extracting every twentieth frame from video footage to reduce redundancy, as adjacent frames are often similar. These frames were manually cropped to highlight areas of interest for each animal. The chapter provides an overview of the tools and techniques used, including MATLAB®, to manage and crop the images. Recording video instead of individual photos proved more efficient for capturing moving animals. The footage was initially recorded in high resolution, then resized to various lower resolutions (960x540 and 480x270) for analysis, balancing image quality and file manageability. The process involved reducing file sizes for effective handling while preserving essential visual details.

Initial Feature Detection with MATLAB® and a developed Speeded-Up Robust Features (SURF) algorithm was tested to identify unique points in cattle images, though initial results showed limited success for individual identification, as it matched only a single feature in almost identical images. Consequently, the research shifted to testing the You Only Look Once (YOLO) deep learning model, renowned for real-time object detection. YOLO’s capacity to detect objects within bounding boxes and label them with confidence scores makes it suitable for

dynamic applications like animal identification in farming environments. This choice of model allows for efficient, single-pass detection across entire images, facilitating high-speed and accurate data processing essential for automated monitoring.

In Chapter 6, the methodology for using YOLO-based object detection, particularly YOLOv3 and YOLOv4, is described for the identification of cattle and other objects within images. The chapter begins by explaining the architecture of YOLOv3, which includes the backbone (for feature extraction), neck (for multi-scale feature integration), and head (for object location and classification). YOLOv3 detects objects at three stages in its network—layers 82, 94, and 106, enabling detection across different object scales using anchor boxes and non-max suppression to filter overlapping bounding boxes.

The chapter also provides a detailed walkthrough of implementing a YOLOv3 model in Python using Keras, with a flowchart outlining the steps for model creation, saving, and object detection for cattle. A Python program identifies cattle heads, with a flowchart visualising the steps of loading the model, performing detection, and presenting results. The study's initial experiments processed images in batches (with size n) of dimensions 416x416, and MATLAB® was used to execute object detection and visualise results, showing successful identification of cattle heads with high confidence scores.

For further accuracy, YOLOv4 was trained with a dataset of cattle images, capturing various poses and conditions. The training process included data augmentation, bounding box initialisation, and loss function optimisation to improve the robustness of the model. The training was conducted on a desktop, due to technical limitations with Google Colab. Results were evaluated by tracking metrics such as training loss and mean average precision (mAP), which improved over time, indicating better object detection performance.

In conclusion, YOLOv3 and YOLOv4 demonstrated successful cattle head identification. However, future improvements are necessary to achieve more accurate individual animal recognition, particularly by enhancing dataset diversity

and refining algorithms. These efforts in automation are expected to support more effective training of advanced object detection models.

Chapter 7 discusses the hardware implementation. This includes cameras, the Jetson Nano implementation, an intelligent feeder and a prototype for proof of concept of the animal feeder.

Chapter 8 presents the results of the YOLO-based animal detection system, outlining the experimental setup, image processing, and performance evaluation. The findings show that YOLOv4 achieves higher accuracy and faster detection than YOLOv3, demonstrating strong robustness under varying environmental conditions. Common issues such as false detections, motion blur, and occlusions were identified, with potential improvements proposed. Comparative analysis confirms YOLO's effectiveness for real-time animal identification, while remaining limitations highlight the need for expanded datasets and further model optimisation.

Chapter 9 concludes the study by summarising the evolution and performance of the YOLO models, highlighting the progression from YOLOv1 to YOLOv4 and their respective speed and accuracy improvements. The research objectives and problem statement are revisited, confirming the success of YOLO-based approaches for real-time animal identification. Key findings emphasise YOLOv4's superior performance and practical applicability in precision livestock monitoring. The chapter outlines the study's contributions, discusses limitations such as dataset diversity and model generalisation, and proposes directions for future research, including advanced deep learning models, expanded datasets, and edge-based implementations. Final remarks reinforce the potential of YOLO in advancing intelligent and efficient animal monitoring systems.

CHAPTER 2: ADVANCEMENTS IN ANIMAL FEEDING SYSTEMS AND REAL-TIME DATA COLLECTION

2.1 Classical methods for animal identification

According to a study done by Taha et al. [29] on the identification of Arabian horses, traditional identification methods such as freeze-branding, hot-iron branding, hoof marking, lip tattooing, ear tags and neckbands can be used. The traditional methods can cause painful infections, duplication and fraud [18], [29].

Over time various new technological methods of tracking animals have been developed and used. These technologies include microchips inserted into the neck of an animal, RFID tags, radio tracking, wireless sensor network tracking, satellite and global positioning systems (GPS) tracking and motion-sensitive camera traps [18], [30]. Disadvantages of the Microchips and RFID tags include that they can be lost, or they can malfunction [29]. RFID tags offer a method of individual animal identification, but according to Yukun et al. [31] RFID tags have the disadvantages of misreading and damage to animal body parts [18].

Automatic camera traps can be used to collect high-resolution images or video of animals, and information such as temperature, can be included in the image data [30]. Due to the lower cost of the traps with advanced digital technology, camera traps are widely used to monitor wildlife, but the data must still be collected from the traps and analysed manually. Another drawback of these camera traps is the fact that even for humans, the processing of these images is challenging. This is because many images will only include partial images of animals, long distances between the camera and animal and poor lighting conditions.

A biometric solution to the identification of the animals offers a solution to the above-mentioned problems. Biometric methods depend on features extracted from horses and not on any external devices [29]. These features can come from the iris, retina and DNA. Salama et al. used pictures of the iris of Arabian horses

with CNNs to identify individual horses [32]. Their tests achieved a 97.6% accuracy in identifying the Arabian horses. They believe that it will be possible to implement this system on other horses, but that horse eyes are different from other eyes, which might not make it suitable for different types of animals.

To automate the data analysis of collected camera images, Artificial Neural Networks (ANN) can be used. Nasser et al. [33], used ANNs with a 100% accuracy to predict animal categories, but did not identify individual animals of the same category or species.

To obtain quality products from farming, the production areas need to be visited frequently, but when farmers spend time and resources on these visits, the cost of farming and consequently the products is increased [34]. Monitoring of animals and crops is possible through Internet-of-things (IoT) technologies, but monitoring only is not enough to make farming smart. Processes such as observation, diagnosis, decision and action are necessary to make farming smart and thus also affordable [34].

Rivas et al. [35] made use of drones to collect images from the air to identify cattle in the fields. These images were later analysed by Convolutional Neural Networks (CNNs) to identify the objects in the gathered images. Using these CNNs, they identified the objects as cattle, but individual animals were not identified [18].

Due to the ability of Neural Networks to analyse captured images, the next section discusses some points associated with neural networks and machine learning.

2.2 Computer Vision Approaches: Traditional image processing vs Deep Learning

Deep learning has significantly transformed image processing, enabling neural networks, particularly convolutional neural networks (CNNs), to automatically learn hierarchical features from data, reducing the need for manual feature engineering [36], [37]. Tasks that were previously considered difficult or infeasible, such as object detection, classification, semantic segmentation, and simultaneous

localisation and mapping (SLAM), can now be performed with high accuracy using deep learning models [38], [39]. While traditional image-processing and classical computer vision techniques remain valuable for simpler tasks or resource-constrained environments, they often require manual design and tuning of features, making them less flexible and scalable than deep learning approaches [40], [41].

The advantages of deep learning are further amplified by advancements in hardware, including increased memory, enhanced computing power, and improved imaging sensors, which have made real-time deployment feasible even on portable devices [36], [38]. Unlike traditional algorithms, CNNs can be retrained with custom datasets to adapt to new domains, offering greater flexibility and reducing the need for extensive domain-specific programming [39], [42]. Nevertheless, classical methods continue to have a role in cases where computational cost, interpretability, or simplicity is critical, and hybrid approaches combining both deep learning and traditional techniques are increasingly being explored for practical applications [41], [43].

This study applied a YOLOv4-based deep learning vision system for individual livestock identification, demonstrating that accurate recognition of animals is feasible on low-power embedded hardware, such as the NVIDIA® Jetson Nano. The system achieved high Average Precision (AP) and Area Under the Precision–Recall Curve (AUC) scores for the majority of classes, confirming that individual recognition can be reliably implemented using CNN-based object detection. While fourteen classes achieved perfect AP of 100% and AUC values of 0.75, a few classes showed lower performance, and three classes yielded no successful detections, highlighting the importance of sufficient training data and balanced class representation.

Compared to traditional identification methods, such as ear tags or branding, the YOLO-based approach offers clear advantages. It enables non-invasive, automated, and repeatable identification, reducing human intervention and the risk of error [41], [42]. The system also supports practical livestock management applications, including targeted feeding optimisation, behavioural and health

monitoring, and theft prevention, thereby demonstrating the broader applicability of AI-driven vision systems for operational efficiency in farming [4], [8]. By moving beyond species-level detection to individual-level recognition, the proposed approach addresses limitations of classical techniques and provides a foundation for data-driven herd management.

In summary, the integration of deep learning into livestock identification illustrates both the technical feasibility and the practical benefits of AI-driven vision systems. While challenges remain for underrepresented classes and extreme scenarios, the study confirms that CNN-based object detection is a powerful tool for modern livestock management, complementing or replacing traditional methods where appropriate.

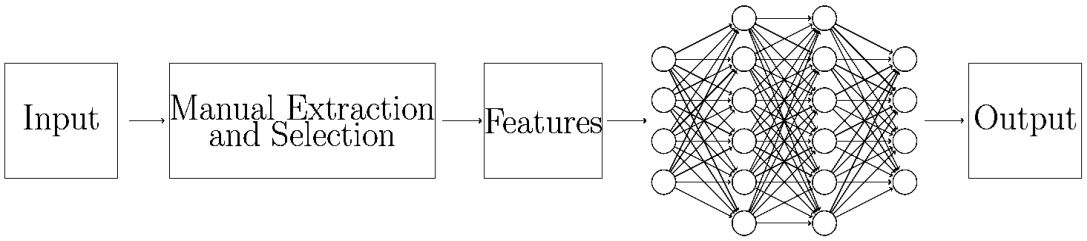


Figure 2.1: Traditional Image Processing workflow

In Figure 2.1 [44], the NN performs the classification of the images using traditional image processing. It uses manual (human) input to help with the extraction and the selection of features.

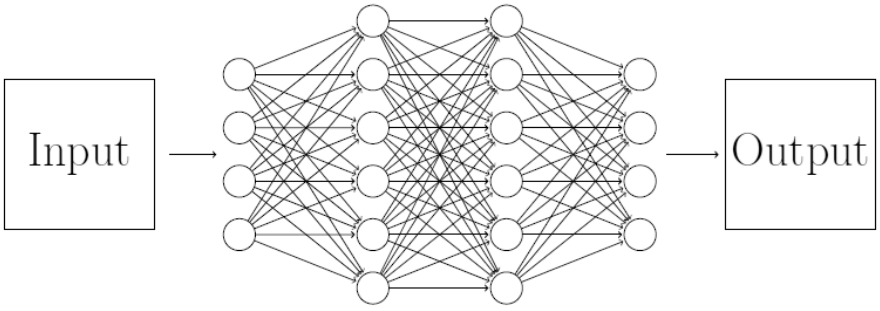


Figure 2.2: Deep Learning workflow

In Figure 2.2 [44], the deep learning NN performs the feature extraction and classification of the images.

2.2.1. Guidelines for making a suitable choice

Andrew Ng, a renowned AI practitioner, once said: *"The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms"* [45]. This highlights the importance of the model, architecture and the data used to train it. Just like the fuel quality will influence the performance of the engine, the quality of the data will influence the quality of the models. Even though DL is powerful, it is not suitable for all applications. Applications where large-scale labelled data is not available or where the problem goes beyond the training distribution pose challenges to DL systems. Common applications which are according to literature, not suitable or as effective as DL, are the following: augmented and virtual reality (AR/VR), 3D modelling, video stabilisation, motion capture and calculation, noise reduction, image registration, stereo vision processing, data compression and encoding.

According to Lin et al. and Deng et al., object detection datasets like COCO and ImageNet explain the massive data requirements necessary for achieving good performance in deep learning systems [46], [47]. If large-scale data and diversity are not available, the robustness and accuracy of DL models significantly diminish. DL performs well with tasks where the outcome is fixed and known beforehand, for instance, classifying animals like a cat, dog or bird. The models choose from a predefined list, and the result will always be one of the options from the list. It is therefore important to map a wide variety of inputs according to a limited and well-defined set of categories, and to have high-quality training data and minimal deviation between the training- and testing data. DL models might suffer in environments with high variability or little data [48].

2.2.2. Deep Learning

To learn and train a NN, a significant amount of information or data is required for DL to work. A powerful computer, with a fast processor or special chips like GPUs, is required for the large amount of data that needs to be processed. DL requires a lot of computing power, which usually makes DL models unsuitable for tiny devices like sensors and microcontrollers. They need more computing power and memory. If a certain field or how to select the right features is unknown, DL can help fill in the gap.

2.2.3. Traditional Methods like basic image processing

In the absence of substantial labelled data for training, employing simpler techniques such as fundamental image processing may prove to be a more effective alternative. Furthermore, if you have limited computer capacity, traditional methods will be more efficient. GPUs necessary for DL are very expensive, therefore if cost is a concern, traditional methods will be preferable. Traditional methods are more flexible and are easier to adapt to different hardware platforms. If you are knowledgeable about the data and know what features of the data matter, traditional methods can be implemented with great success.

Table 2.1 depicts a comparison between DL and traditional image processing.

Table 2.1: Comparison between deep learning and traditional image processing [49]

Selection criteria	**Traditional Image processing**	**Deep learning**
Training dataset	Small	Large
Computing power	Low	High
Feature engineering	Required	Unnecessary
Training time	Short	Long
Annotation time	Short	Long
Algorithm Transparency	High	Low
Domain expertise	High	Low
Priors (Assumptions)	Few	Many
Proprietary material - Risk of exposure	DLLs (Risk negligible)	Model files, DLLs (Risk high)
Deployment flexibility	High	Low
Expenditure (BOM)	Low	High

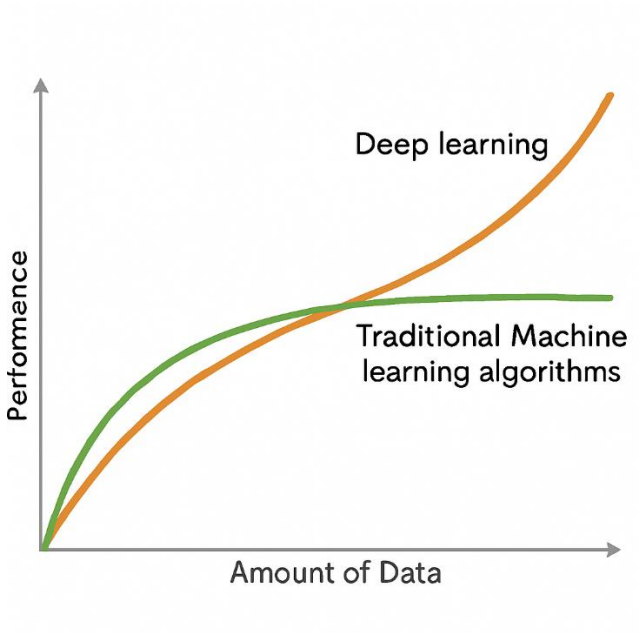


Figure 2.3: Traditional Machine Learning vs. Deep Learning

Figure 2.3 [49], [50] depicts the relationship between the amount of available data and performance for both DL and traditional ML. It can be noted that as the amount of data increases for DL, the performance also improves, but the performance of Traditional ML does not increase once it reaches a certain amount of data.

Combining evaluation tools from classical machine learning with deep learning models (deep networks) has grown in importance and popularity over time due to evidence that it produces superior models. Hybrid implementation requires significantly lower CPU resources and about half of the memory bandwidth.

Table 2-2 [49] shows a comparison between applications suitable for Traditional Image Processing and Deep Learning.

Table 2-2: Typical applications of DL and traditional image processing

Traditional Image Processing	Deep Learning
Image transformation (Lens distortion correction, view changes)	Image classification (OCR and Handwritten character recognition
Image Signal Processing (ISP)	Object detection/ identification
Camera calibration	Semantic segmentation

Industrial inspection – Defect detection	Instance segmentation
Stereo image processing	Image synthesis
Automatic panorama stitching	Image colourisation
3D data processing	Image Super-resolution
Calculating geometries	Scene understanding

2.2.4. Traditional Machine Learning

Traditional machine learning relies on human-engineered features which implies that, before training a model, experts need to decide which characteristics (features) of the data are important. For example, in image classification, they might extract colour histograms, edges, or textures manually and then use those as inputs for algorithms like decision trees, Support Vector Machines (SVM), K-Nearest Neighbours (KNN), and Logistic Regression. These models usually perform well with well-chosen features and datasets that are not too complex. Traditional ML often struggles when raw data is multi-dimensional or vague (like images, videos, or audio).

2.2.5. Deep Learning

Deep learning is a subset of machine learning that automatically learns features from raw data using neural networks, especially deep neural networks with many layers. In the case of image recognition, for instance, a deep learning model like a Convolutional Neural Network (CNN) can process the raw pixels and gradually learn complex features like edges, shapes, and objects, without needing manual or human feature engineering. Deep learning thrives with large datasets, high computational power and complex data types (images, sound, language).

Some popular deep learning models include:

- CNNs (for image tasks)
- Recurrent Neural Networks (RNNs) and long short-term memory (LSTMs) (for sequential data)
- Transformers (for Natural Language Processing (NLP) and vision)

Table 2-3 highlights the key differences between traditional ML and DL.

Table 2-3: Key Differences: Traditional ML and DL

Aspect	Traditional Machine Learning	Deep Learning
Feature Engineering	Manual	Automatic
Data Requirement	Works with small to medium datasets	Requires large datasets
Computation	Relatively lightweight	Computationally intensive
Interpretability	Often easier to interpret	Often considered a "black box"
Performance on Complex Tasks	Limited	Superior on complex/unstructured data

2.2.6. Summary

Traditional machine learning needs humans to tell the model what to look for. If DL has enough data and computing power, it can identify what is important on its own.

2.3 Object Detection Models: R-CNN, SSD, YOLO series (v1 to v4)

Object detection has significantly advanced with the development of various DL models. Among the most influential are R-CNN, SSD, and the YOLO series. Each of these models follows different strategies to identify and localise objects within images, offering trade-offs in terms of accuracy, speed, and complexity. Table 2-4 shows a comparison between the different detection models and also lists the strengths and weaknesses of each one.

Table 2-4: Different detection models with strengths and weaknesses

Object Detection Model	Description	Strengths	Weaknesses
The Region-based Convolutional Neural Network (R-CNN)	(R-CNN) was one of the earliest deep learning-based object detectors. It works by first generating a set of region proposals using a selective search, thereafter a CNN is used to extract features from each proposed region. These features are passed to a classifier (such as SVM) to label the objects.	High accuracy due to the use of region proposals and powerful CNN features.	Very slow during both training and inference, as it processes each proposal individually. It also requires a multi-stage pipeline, making implementation and optimisation more complex.
The Single Shot Multibox Detector (SSD)	Improves detection speed by removing the need for region proposals. It divides the input image into a grid and predicts bounding boxes and class	Much faster than R-CNN and suitable for real-time applications. It balances speed and accuracy effectively.	Struggles with detecting small objects and can be less accurate than more complex models.

	probabilities directly from feature maps in a single pass.		
YOLOv1	Introduced the unified detection approach. It was extremely fast but suffered from localisation errors and poor performance with small objects.	Excellent for real-time applications. Later versions offer a great compromise between speed and accuracy.	Earlier versions had issues with small object detection. Although v3 and v4 improved this, they still trail behind some models like Faster R-CNN in terms of precision.
YOLOv2 ***(YOLO9000)***	Enhanced accuracy with batch normalisation, anchor boxes, and higher resolution training. It improved significantly compared to v1, especially for smaller objects.		

YOLOv3	Introduced multi-scale predictions and a deeper architecture (Darknet-53). This version balanced speed and accuracy better, making it one of the most widely adopted models at the time.		
YOLOv4	Integrated various advancements from the broader deep learning community (e.g., Mish Activation, Cross-Stage Partial connections). It offered superior performance with competitive speed, making it suitable for both research and production.		

2.3.1. Comparison of different models

Each model has played a crucial role in advancing the field of object detection. While R-CNN prioritised accuracy, SSD and the YOLO series paved the way for real-time applications by optimising speed and simplicity.

Table 2-5: Comparing different models

Model	Accuracy	Speed	Key Feature	Use Case Suitability
R-CNN	High	Very Slow	Region proposals + CNN	Offline, high-accuracy tasks
SSD	Moderate-High	Fast	Single-shot detection	Real-time applications
YOLOv1	Moderate	Very Fast	Unified architecture	Fast detection, fewer objects
YOLOv2	Improved	Fast	Anchor boxes, better backbone	General object detection
YOLOv3	High	Fast	Multi-scale predictions	Versatile, real-time use
YOLOv4	Very High	Fast	SOTA tricks + optimised design	Industrial-grade solutions

2.4 YOLOv4 in Detail: Key advancements and performance in other domains

You Only Look Once version 4 (YOLOv4) marked a major leap in the YOLO family of object detection models. Released in 2020, it was designed to find a balance between high accuracy and real-time performance, making it suitable for both research and production environments. One of the standout features of YOLOv4 was its integration of several cutting-edge techniques from the broader deep learning community. Techniques used include the addition of the Bag of Freebies (BoF) as well as training techniques that improve accuracy without affecting inference speed, like Data augmentation methods such as Mosaic augmentation, CutMix, DropBlock, and CIoU Loss for better bounding box regression. The Bag of Specials (BoS) includes architectural enhancements that improve accuracy with a slight cost in speed, such as Mish activation function, Cross-Stage Partial Connections (CSP) and Spatial Pyramid Pooling (SPP). Better backbone and neck design adopted CSPDarknet53 as its backbone for feature extraction and used structures like PANet to enhance feature fusion. These techniques allow it to

perform competitively without the need for extremely powerful hardware. YOLOv4 introduced a number of improvements across the model's architecture, training strategies, and performance optimisation. These upgrades were the result of a modular design philosophy, where the best components from different research studies were combined to create a high-performing system. Some of the key advancements include:

Table 2-6: Key improvements introduced to YOLOv4

Improvement component	Description
Backbone: CSPDarknet53	YOLOv4 adopted CSPDarknet53 as its backbone network, which is an enhanced version of Darknet53 used in YOLOv3. The introduction of Cross Stage Partial (CSP) connections reduces computational complexity while maintaining effective gradient flow and preserving model accuracy.
Neck: PANet and SPP	To improve multi-scale feature aggregation, YOLOv4 employed a combination of Path Aggregation Network (PANet) and Spatial Pyramid Pooling (SPP). This design enhances the detection of objects at varying scales and improves localisation accuracy.
Bag of Freebies (BoF)	Bag of Freebies refers to training-time enhancements that improve detection accuracy without increasing inference cost. YOLOv4 incorporated techniques such as mosaic data augmentation, DropBlock regularisation, and class label smoothing to improve model generalisation.

Bag of Specials (BoS)	Bag of Specials includes architectural and training modifications that introduce a minor increase in computational cost in exchange for improved accuracy. YOLOv4 integrated Mish activation functions, Complete Intersection over Union (CIoU) loss for bounding box regression, and Self-Adversarial Training (SAT) to enhance overall performance.

2.4.1. Efficiency and Real-Time Performance

One of YOLOv4's most impressive achievements is its ability to deliver state-of-the-art results on standard object detection benchmarks while running at real-time speeds on a single GPU (such as NVIDIA® GTX 1080 Ti), making it more accessible than earlier models requiring TPUs or multi-GPU setups.

2.4.2. Performance in Other Domains

YOLOv4's versatility has made it popular across various application domains, often with minimal adjustments. Its fast inference speed and reliable accuracy make it particularly valuable in real-world settings.

2.5 YOLOv4 applications

2.5.1. Autonomous Driving

YOLOv4 has been applied to tasks like vehicle detection, pedestrian tracking, and traffic sign recognition. Its ability to process frames in real time makes it suitable for on-board vision systems in self-driving cars and driver-assistance technologies.

2.5.2. Medical Imaging

Researchers have used YOLOv4 in analysing X-rays, CT scans, and MRI images. Tasks such as detecting tumours, counting cells, or identifying anomalies benefit from the model's accurate localisation abilities.

2.5.3. Surveillance and Security

For video surveillance, YOLOv4 helps in detecting suspicious behaviours, monitoring people in restricted zones, or tracking individuals across multiple camera feeds—all while maintaining real-time performance.

2.5.4. Agriculture

In smart farming, YOLOv4 has been used for detecting pests, classifying plant diseases, and identifying crops at different growth stages. Its robustness to variations in lighting and weather conditions makes it reliable in outdoor environments.

2.5.5. Industrial Automation

In manufacturing, YOLOv4 supports quality control tasks such as detecting product defects, sorting items on conveyor belts, and monitoring assembly lines.

2.6 Conclusion

YOLOv4 represents a well-balanced object detection model that combines high performance with practical efficiency. By integrating innovations in architecture and training strategies, it pushes the boundaries of what could be achieved with a single GPU. It's strong generalisation, speed, and modular design have enabled it to succeed in a wide range of fields. YOLOv4 demonstrates the power of modern computer vision in solving complex real-world applications, like autonomous vehicles, healthcare and agriculture.

2.7 Applications in Animal Monitoring: Review of similar works using CNNs for animal detection and feeding

The easiest way to feed animals is to simply put the feed on the ground, but this method can lead to losses of up to 30% [51]. Due to the price of the animal feed,

this can be very costly mistake. Feeders and self-feeders vary widely in cost, from inexpensive to highly priced, but the potential savings from preventing animals from trampling feed into the ground can offset the initial investment [52].

“Animal production is one of the major sectors in South African agriculture and contributes substantially to the economy of the country” [12]. It is thus of utmost importance to farmers to farm at an optimal profit. According to Meisner et al. [13], 70% of agricultural land in South Africa can only be utilised by livestock, and game and other species are found in all provinces, with high concentrations in the eastern higher rainfall regions. Livestock production in South Africa is a significant contributor to food security and clothing, as well as the country's social and economic levels [13].

Industrial-scale production of animal feeds commenced in the late 19th century [14]. During this time the benefits of a balanced diet and nutrition in human and animal diets were identified. The benefits and role processing of certain raw materials were identified. Today, the nutritional needs of farm animals are known and understood and can be satisfied through natural grazing alone, but it may be necessary to supplement the nutrients in concentrated and controlled form [15]. The quality of the feed is influenced by the nutrient content, as well as other factors like feed presentation, hygiene, digestibility, and effect on intestinal health [15], [16], [17].

Due to the differences in the digestive systems of the different species of livestock, like cattle, horses, as well as baby animals, it is important to feed the animals the correct type of feed. Since the feeding of animals can be the most expensive part of farming, farmers can supplement the normal feed, with other industry by-products, like distiller’s grains and soybean [18], [19]. This is not only environmentally sustainable but also economically viable for the farmers.

Most stock farmers buy their feedstuffs in bulk because this is more cost-effective. Agriseta [20] identified the most important aspects of good quality control as effective record keeping by the farmer. Some of the items that should be included in the record system are daily intake and usage [20]. Feeding behaviour contains

important information that can enable producers to better manage livestock; similarly, researchers can benefit by better understanding the factors that influence feed intake [21].

A survey carried out on 18 farms in Switzerland, Germany, Denmark and the Netherlands indicates that an increasing number of farms are relying on automatic feeding to ease their workload, save time and achieve flexibility [22]. Various systems are available which make automated feeders and feeding possible. Rail-guided feed wagons are currently the most established in practice, but conveyor belts and self-propelled feeders are also utilised [22].

Livestock farmers suffer losses in the region of R2 billion each year due to predators, such as black-backed jackal, caracal, leopard, cheetah, brown hyena, and even crows and stray dogs [23]. To investigate the reasons for disappearing animals, it is very important to know when a specific animal stops feeding. This allows for effective predator control.

There is currently also a need to identify animals for other reasons as well. Identification might be required for tracking of animals, to prevent the spread of disease between different regions and cross-border movement of animals [24]. Stock theft is also a large problem in South Africa [25]. The identification of stolen animals is very important for the successful prosecution of thieves as well as the recovery of the animals.

There are different marks that can be applied to animals for identification. Some are permanent, while others are non-permanent. Non-permanent methods include cutting the brush of the tail or colour markers, such as paint. Permanent methods include ear notching, ear tags, tattoo, hot iron branding, freeze branding and electronic markers [26]. The Draft Document for a Livestock Identification and Traceability system in South Africa [24] suggests identification methods such as ear tags and RFID tags.

“It is noteworthy that with certain breeds of cattle e.g., Nguni and Friesland, the colour markings of the animals are as unique as a thumb print” [26]. It is thus

possible to record colour marking and patterns for the identification of these animals. Photos can be taken and kept for the identification of individual animals. Due to the distinct markings on animals, it should therefore be possible to identify individual animals using image processing techniques. According to Yu et al [27], image sensors are increasingly being used in biodiversity monitoring. Every study produces thousands or millions of pictures. They also state that efficiently identifying the species captured by each image is a critical challenge for the advancement of this field. Matcher [28] believes that Artificial Intelligence (AI) can successfully be used to identify specimens to make science more accessible to everybody. AI can therefore be used to enhance the success rate as well as the reliability of image recognition.

2.8 Research Gaps Identified

Current vision research identifies species of animals and not individual animals by using portable equipment. When individual animals are identified, it increases the possible applications on a farm and will benefit the farmers day-to-day operations and decision-making. Deploying such a system on a small portable computer like the NVIDIA® Jetson Nano, enables the farmer to use the system in different places, like in the kraal, as well as in the veld. Not only will it make the care and control of animals easier, but it can also lead to cost savings on the farm.

CHAPTER 3: FOUNDATIONS OF ARTIFICIAL INTELLIGENCE: AN OVERVIEW

3.1 Background

In this chapter, we embark on a journey through the fascinating world of artificial intelligence (AI), exploring its diverse interpretations and its fundamental role in simulating intelligent behaviour. We will begin by unravelling the principles of machine learning, highlighting how algorithms empower computers to analyse data and generate predictions.

Our exploration will then shift to deep learning, where we will uncover how multi-layered algorithms emulate human learning processes to tackle complex tasks. We will examine the intricate relationship between AI, machine learning, and deep learning, clarifying how each concept interconnects within the broader AI landscape.

Delving deeper, we'll provide a thorough examination of neural networks, focusing on their architecture and their critical function in deep learning. This will include a discussion on various activation functions and training mechanisms such as backpropagation, gradient descent, and dropout. We will also explore convolutional neural networks (CNNs), detailing their convolutional and pooling layers and their pivotal role in image processing.

Through this chapter, you will gain a comprehensive understanding of these core concepts and their applications, setting the stage for more advanced topics in AI and machine learning.

3.2 Artificial Intelligence

Although there is debate across a wide range of fields and disciplines, including psychology, philosophy, sociology as well as computer science, a generally accepted definition of intelligence does not exist. People interpret intelligence in many ways [53]. Science tried to mimic intelligence for decades, which led to the

development of AI systems [54]. Even though the term AI does not have a specific, clear meaning [53], one can argue that what occurs is artificial. AI is accepted to be a field in Computer Science that researches “intelligent behaviour” in machines, that produces systems that produce slightly predictable outputs [54].

AI can include activities such as learning, reasoning and understanding. However, a computer can often mimic intelligence as part of a simulation. This simulation process will include the following steps, as shown in Figure 3.1 below [53]:

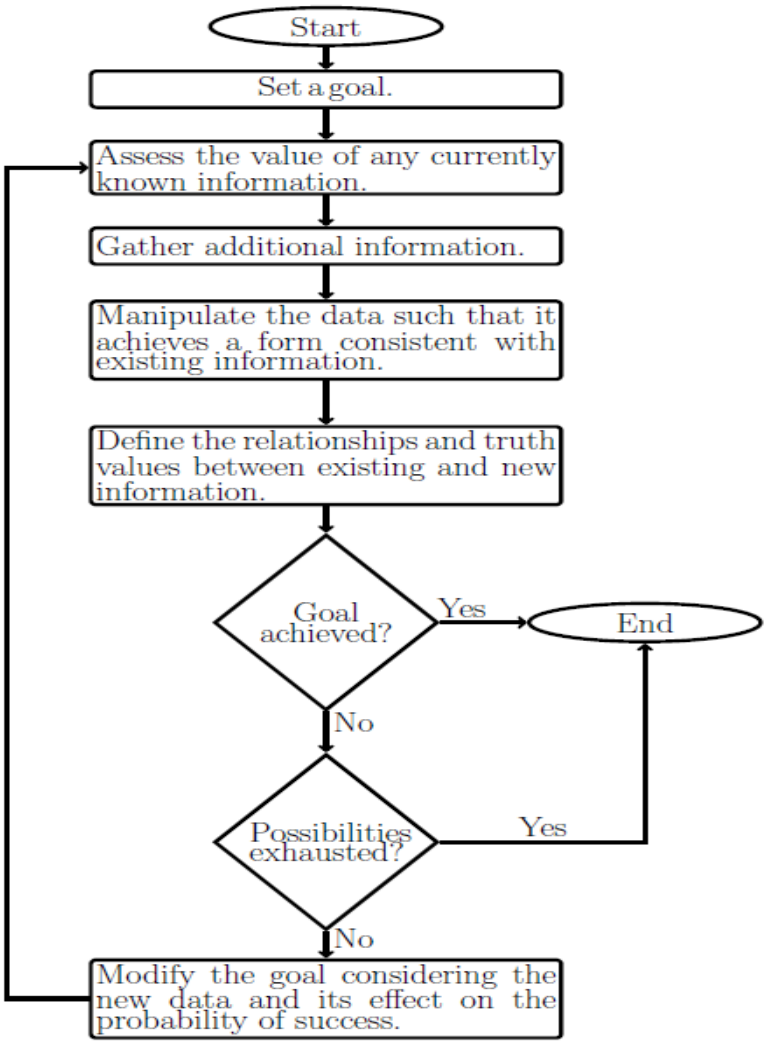


Figure 3.1: Simulation process of AI

3.3 Machine Learning: Automating Knowledge Acquisition

Machine learning (ML) is automated learning [55]. In machine learning, we program computers so that they “learn” from inputs available to them and use an algorithm to solve problems [54], [55]. An algorithm that recognises cattle and sheep can be created by showing it hundreds of photos of farm animals. This is a process of converting experience into expertise or knowledge [55].

A popular approach to ML is Deep Learning [54].

3.4 Deep Learning: Layers of Complexity

Deep learning (DL) is the use of many learning algorithms in layers to solve complex tasks. For humans to learn to read, young children will first be taught the alphabet, and when this is mastered, they will read simple words and later sentences. With practice they can read books, form opinions and comprehend the content of a book. Algorithms used in deep learning, break down tasks into sequential problems. Just like the child who learns to read, each task will help to solve the problem [54].

In practice, DL and ML algorithms are frequently the tools of choice for Data Science (DS), Data Engineering, Data Analytics, Data Analysis and Cognitive Computing [54].

3.5 The Differences between AI, ML, and DL: A Hierarchical Approach

The relationship between AI, ML and DL can be explained in the following way: Machine learning is a kind of artificial intelligence and deep learning is a kind of machine learning. ML develops a model or algorithm out of “data” [56]. Training refers to the process of using a deep-learning framework and a training dataset to create a machine learning algorithm [18], [57], [58]. Figure 3.2: Machine Learning Process , shows the process that takes place when machine learning or training occurs.

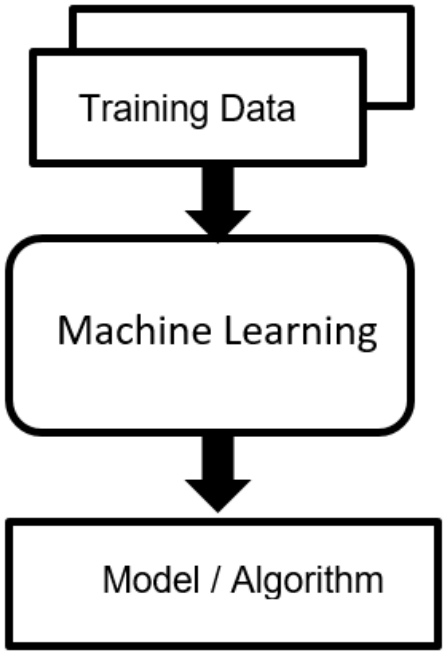


Figure 3.2: Machine Learning Process [18], [56]

Once the algorithm has been developed or trained, it can be implemented with real-world data [56]. Figure 3.3 shows how the developed algorithm is implemented to achieve a predictable output.

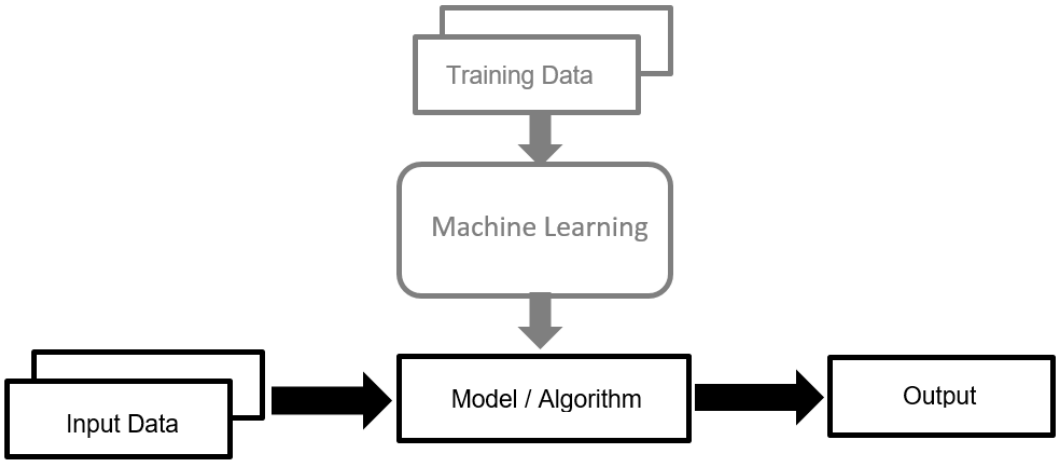


Figure 3.3: Implementation of Developed Algorithm [18], [56]

Inference time refers to the amount of time it takes to use a trained machine learning algorithm to make a prediction [58, 57] or to get an output from real-world data, such as images.

Due to the amount of processing required when implementing an algorithm, the size and performance of the hardware platform are determined by the amount of work, i.e., the decision-making you expect the AI platform to perform [53].

Imran Bangash [59] benchmarked three of the most popular AI hardware accelerators: Intel Movidius NCS stick, Google Coral USB stick, and NVIDIA® Jetson Nano [18].

Table 3-1: Benchmark results [18]

Parameters	Nvidia® Jetson Nano	Google Coral® USB	Intel Movidius® NCS
Inference time	~38 ms	~70 – 92.32 ms	~225-227 ms
Fps	~25	~9-7	~4.43 – 4.39
CPU usage	47-50%	135%	87-90%
Memory usage	32%	8.7%	7%

From the results in Table 3-1 above, the NVIDIA® Jetson Nano, with its low inference time, will be a good choice for implementation of an AI system. Specifications published on NVIDIA's website [60], show that in comparison with the available systems, the NVIDIA Jetson Nano performs very well with different neural networks and AI tasks [18]. In May 2020, NVIDIA® released the NVIDIA® Xavier NX, which is more directed towards the professional and commercial market than the NVIDIA® Jetson Nano. Although the NVIDIA® Xavier NX offers higher performance, it also comes at a much higher price of 399 USD compared to the NVIDIA® Jetson Nano price of 99 USD [61].

3.6 Artificial Neural Networks: Building Blocks of Deep Learning

Neural Networks serve as the foundation of deep learning, aiming to approximate unknown functions through interconnected neurons. These neurons possess weights and biases that are updated during training based on errors. An activation function introduces nonlinearity to the linear combination, producing the network's output. Liping Yang defines neural networks as systems comprised of

interconnected artificial neurons, passing data and adjusting weights based on the network's experience [62]. Neurons with activation thresholds are fired when certain weight and data combinations are met, contributing to the network's learning process.

A neuron serves as the fundamental unit in both the brain (Figure 3.4) and a neural network (Figure 3.5). Analogous to how we process information to produce an output, a neural network's neuron receives input, processes it, and generates an output. This output may either be sent to other neurons for additional processing or represent the final output of the neural network [62].

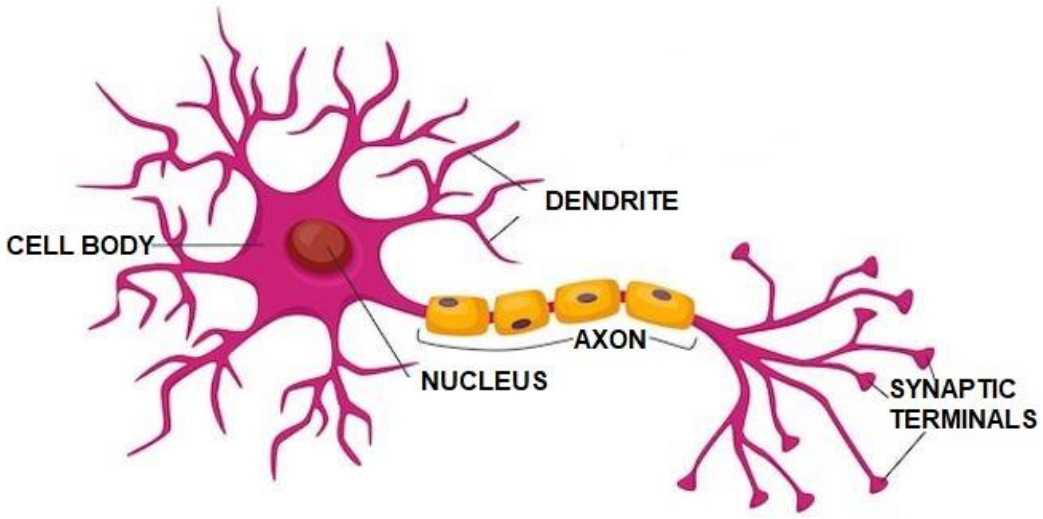


Figure 3.4: Biological Brain Neuron [63]

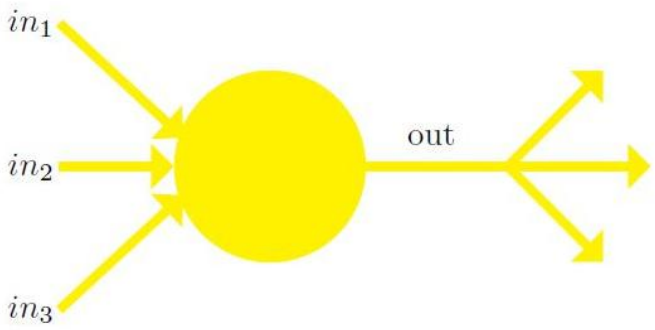


Figure 3.5: Neuron in a neural network

In neural networks, input values entering a neuron undergo multiplication by associated weights. In machine learning, models are characterised by operations guided by weights. Altering these weights allows the model to perform different tasks. For example, a network can be trained to recognise various objects like cats and dogs or birds and fish, with the changes occurring in the weights rather than the network structure itself [54].

If a neuron has multiple inputs, each input is assigned a specific weight, initially set randomly. Throughout the model training, these weights are adjusted. Post-training, the neural network assigns higher weights to inputs deemed more crucial, while a weight of zero signifies insignificance for a particular feature [62]. Assume the input to be x_1, and the weight associated to be ω_1. After passing through the node the input becomes:

$$\boldsymbol{Input_n = x_n * \omega_n} \quad \textbf{(3.1)}$$

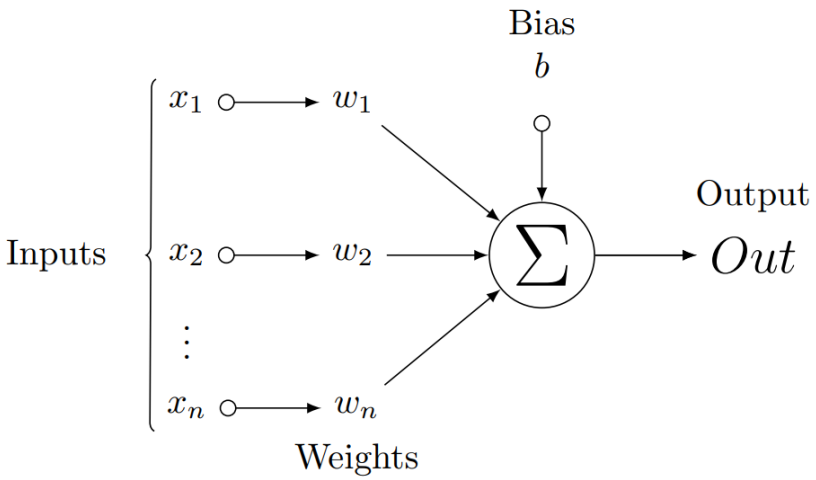


Figure 3.6: Neuron weights and bias

In addition to weights, a linear component called bias is applied to the input in neural networks. The bias is added to the result of the weight multiplication, altering the range of the weighted input. The final linear transformation is expressed as:

$$\boldsymbol{Out = \sum_{n=1}^{n}(x_n * \omega_n) + b} \quad \textbf{(3.2)}$$

Following this, an activation function is applied, introducing non-linearity to the process. The activation function, f(), translates input signals to output signals, resulting in an expression like:

$$\boldsymbol{f(x_1 * \omega_1 + Bias)} \tag{3.3}$$

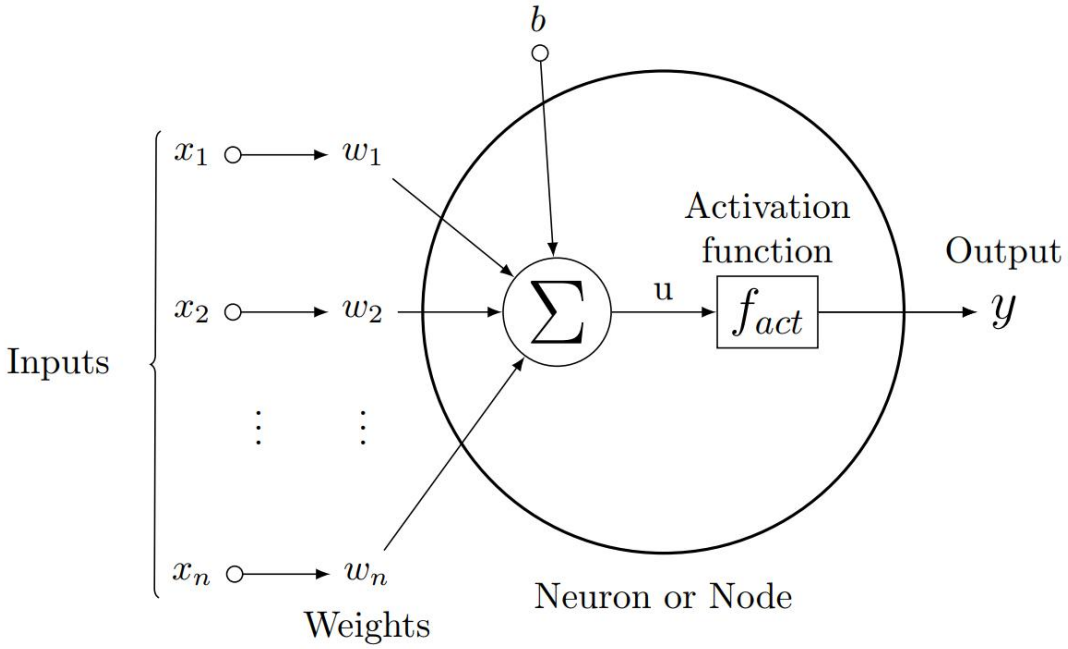


Figure 3.7: Neural Network unit [62]

Figure 3.7 illustrates "n" inputs (x_1 to x_n) with corresponding weights (W_1 to W_n) and a bias (b), showcasing the multiplication of weights with inputs, their summation, and the addition of the bias.

$$\boldsymbol{u = \sum_{n=1}^{n}(w_n * x_n) + b} \tag{3.4}$$

The activation function is applied to u or f(u) and we receive the final output from the neuron as [62]:

$$\boldsymbol{y = f(u)} \tag{3.5}$$

The Multi-Layer Perceptron (MLP) utilises stacks of neurons to handle complex tasks, as a single neuron is insufficient for such purposes. A basic MLP configuration comprises an input layer, a hidden layer, and an output layer. Each layer contains multiple neurons, and all neurons within a layer are interconnected with those in the subsequent layer, forming fully connected networks [62].

Artificial Neural Networks (ANN) are a widely adopted approach in implementing deep learning. They consist of stacked layers of simple learning algorithms that sequentially process input data to generate an output. This sequential processing reflects the essence of deep learning, where each layer progressively refines its understanding of the input [62]. ANNs consist of layers, as depicted in Figure 3.8. Each layer performs an operation taking inputs, parameters, and generating outputs. The structure involves processing input data through various layers to produce the final output.

Neural networks consist of three types of layers: the input layer, which receives the input and is the network's initial layer; the output layer, responsible for generating the final output; and hidden layers, which process data and pass their outputs to the next layer. While the input and output layers are visible, the intermediate hidden layers perform specific tasks on the input data and remain concealed in the network structure.

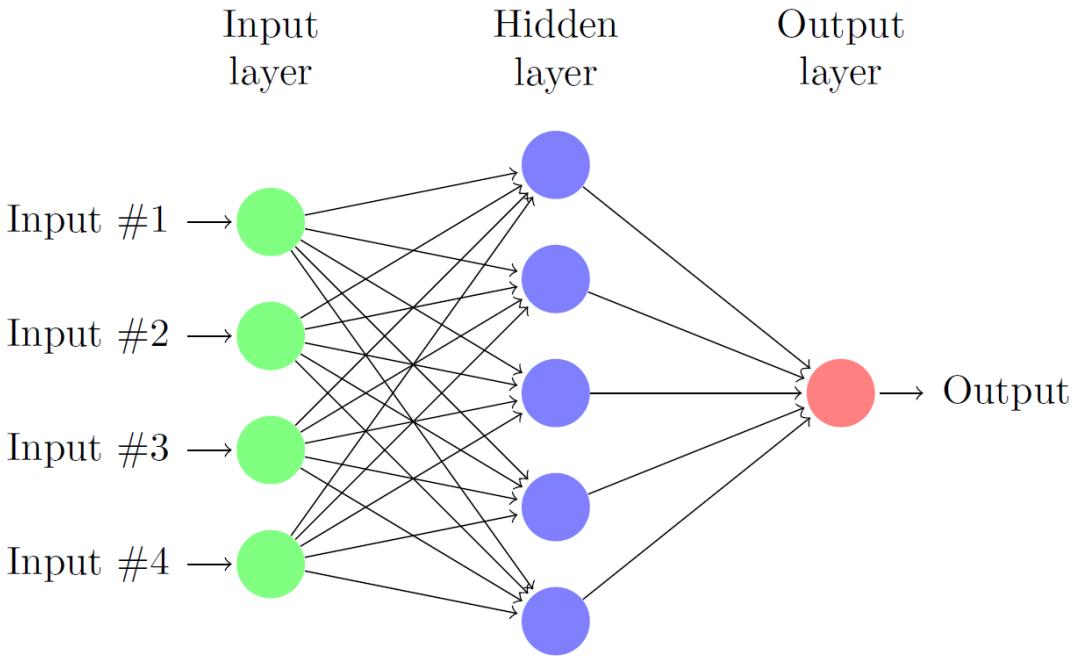


Figure 3.8: Single hidden layer ANN

The number of hidden layers is not limited to only one (see Figure 3.9: Two hidden layers ANN, Figure 3.9 and Figure 3.10), but most problems can be solved using a single hidden layer [64].

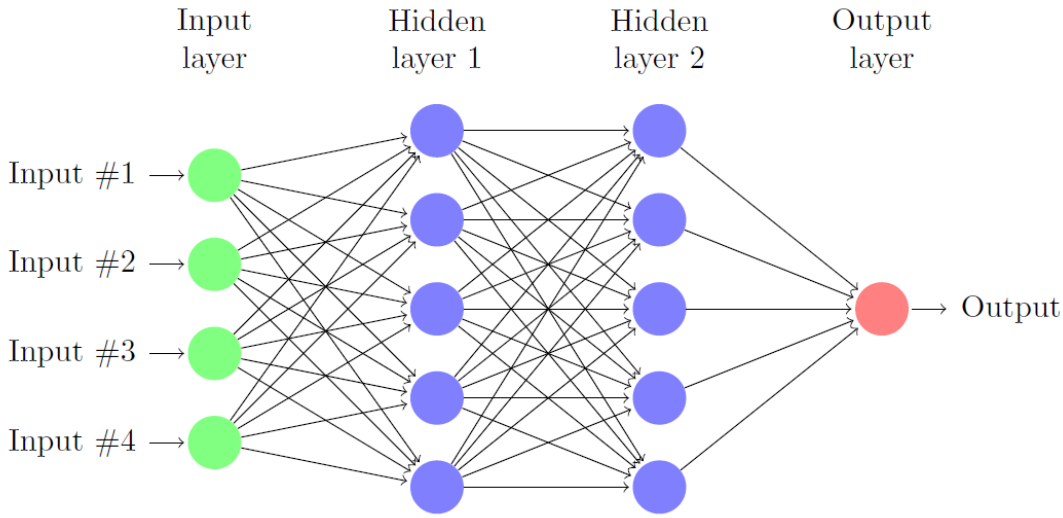


Figure 3.9: Two hidden layers ANN

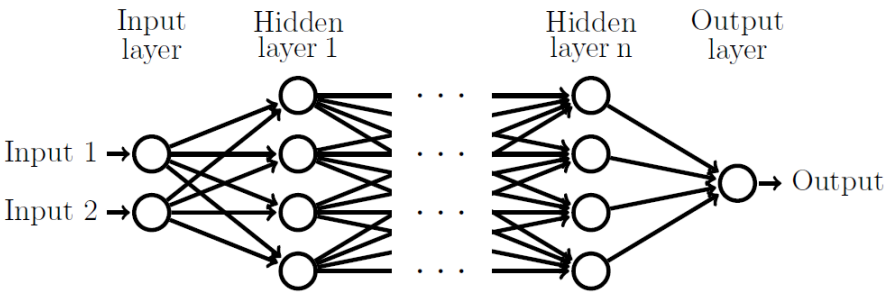


Figure 3.10: Input, Output and Multiple Hidden Layers

Forward Propagation is the input moving forward through the hidden layers towards the output layers. Data or inputs are passed in a single direction through the hidden layers and then to the outputs [62].

Dense Network, Fully Connected Network, and Multi-Layer Perceptron are interchangeable terms referring to the fundamental neural network structure. This configuration consists of dense layers intertwined with activation functions. Emphasising the significance of activation functions, it is noted that without them, the network would lack complexity, resembling a single dense layer. The inclusion of non-linearities through activation functions enhances the model's power by transforming ordinary components into a more sophisticated system [54].

The Dense Layer in a neural network is a crucial element where each neuron is intricately connected to every neuron in its preceding layer. Figure 3.11 depicts an example of Dense Layers.

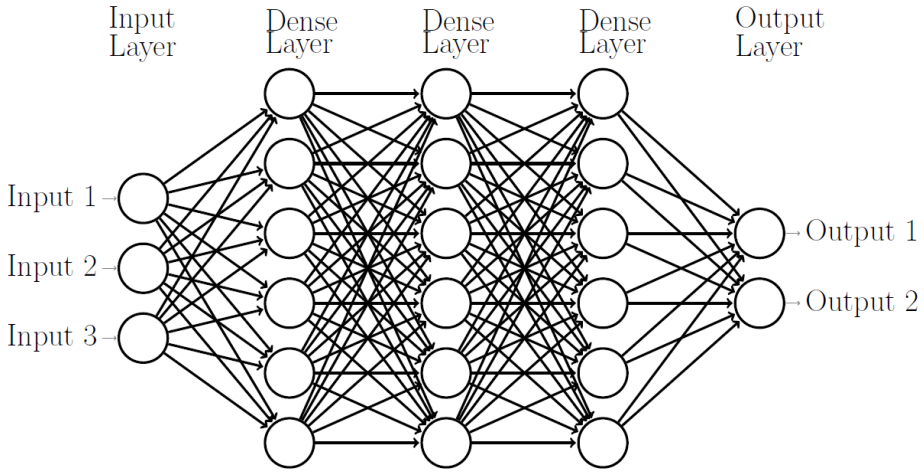


Figure 3.11: Dense Neural Network

This layer commonly employs matrix-vector multiplication, wherein the output from the preceding layer, represented as a row vector, is multiplied by the parameters of the dense layer, arranged as a column vector. The general formula for matrix-vector multiplication dictates that the row vector must have the same number of columns as the column vector.

The general formula for a matrix-vector product is:

$$\mathrm{Ax} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}x_1 + & a_{12}x_2 + & \dots + & a_{1n}x_n \\ a_{21}x_1 + & a_{22}x_2 + & \dots + & a_{2n}x_n \\ & & \vdots & \\ a_{m1}x_1 + & a_{m2}x_2 + & \dots + & a_{mn}x_n \end{bmatrix} \quad \textbf{(3.6)}$$

The dense layer stands out as the most frequently utilised layer in various models [65]. The dense layer in a neural network is a key component involved in dimensionality reduction. Represented by a matrix A (M x N) and a vector x (1 x N), the layer's trained parameters, which may be updated through backpropagation, contribute to this process. Backpropagation, a commonly used algorithm for training feedforward neural networks, computes the gradient of the loss function concerning the network's weights for a single input or output. The output from the dense layer is an N-dimensional vector, indicating a reduction in vector dimensionality [66].

Activation Functions and Training Mechanisms in Neural Networks

Activation functions are common non-linear functions employed in deep learning with the primary objective of introducing non-linearity to the input [54].

The most common activation functions are [62]:

- Sigmoid
- Rectified Linear Units (ReLU)
- Softmax

3.7.1. Sigmoid

Sigmoid (σ) is one of the most common activation functions used. It is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.7}$$

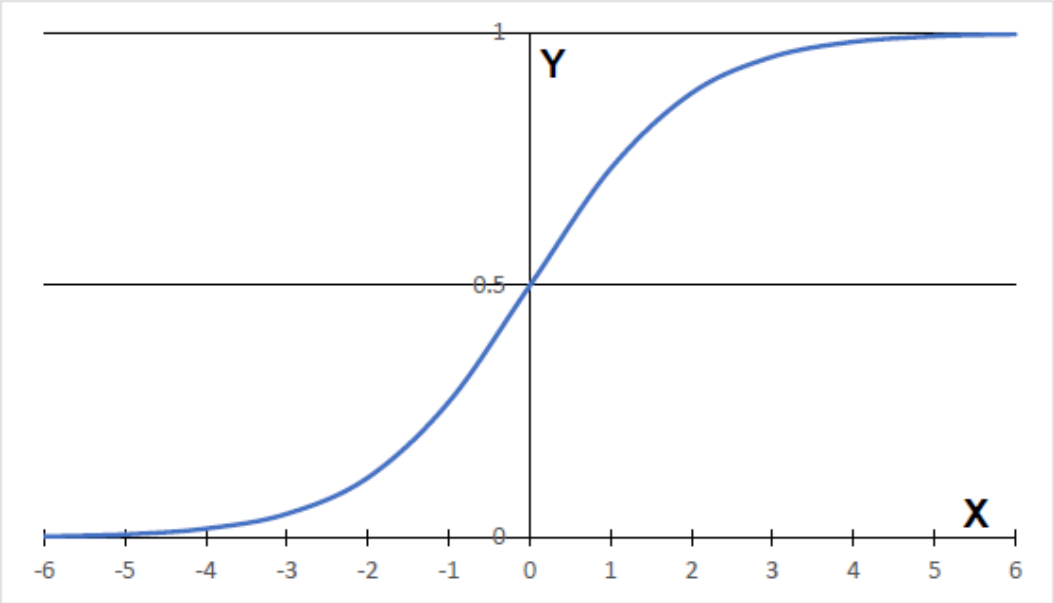


Figure 3.12: Sigmoid [62]

One of the main advantages of the sigmoid function is that it generates a smoother range of output values between 0 and 1 compared to step functions. Ideally, the output should change gradually in response to small changes in the input values, which the sigmoid function achieves effectively [62].

3.7.2. Rectified Linear Units

While sigmoid functions were historically popular, Rectified Linear Units (ReLU) have become the preferred activation functions for hidden layers in modern

neural networks. This shift is largely due to their computational efficiency and training advantages.

The function is defined as [62]:

$$f(x) = max\ (x, 0) \tag{3.8}$$

As a result, the output of the function is equal to x when x > 0 and 0 when x ≤ 0.

The graphical representation is illustrated in Figure 3.13 [62].

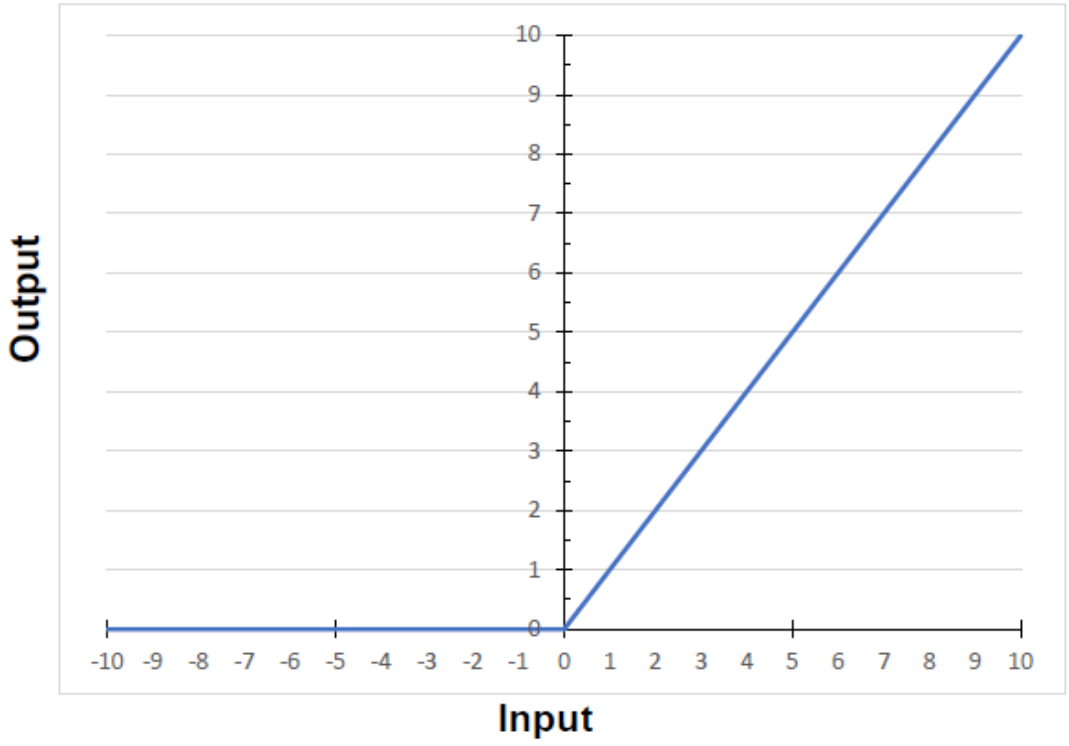


Figure 3.13: Rectified Linear Units

The key advantage of utilising ReLU lies in its constant derivative value for all inputs greater than 0. This property significantly facilitates faster training of the network by reducing the likelihood of vanishing gradients [62].

3.7.3. Softmax

In contrast to sigmoid and ReLU functions, which are commonly used in hidden layers, Softmax activation functions are typically employed in the output layer for classification tasks, particularly in multiclass scenarios. Similar to the sigmoid function, Softmax converts outputs into probabilities by normalising them so that their total equals 1. However, while sigmoid is suitable for binary classification, Softmax is better suited for multiclass problems, as it simplifies the assignment of probabilities to each class and provides easily interpretable [62].

For example, identifying numbers, such as distinguishing between 6 and 8, can be simplified by assigning probabilities to each possible outcome. In this scenario, the Softmax function assigns the highest probability to the most likely number (e.g., 6), followed by lower probabilities for alternatives such as 8, making classification straightforward [62].

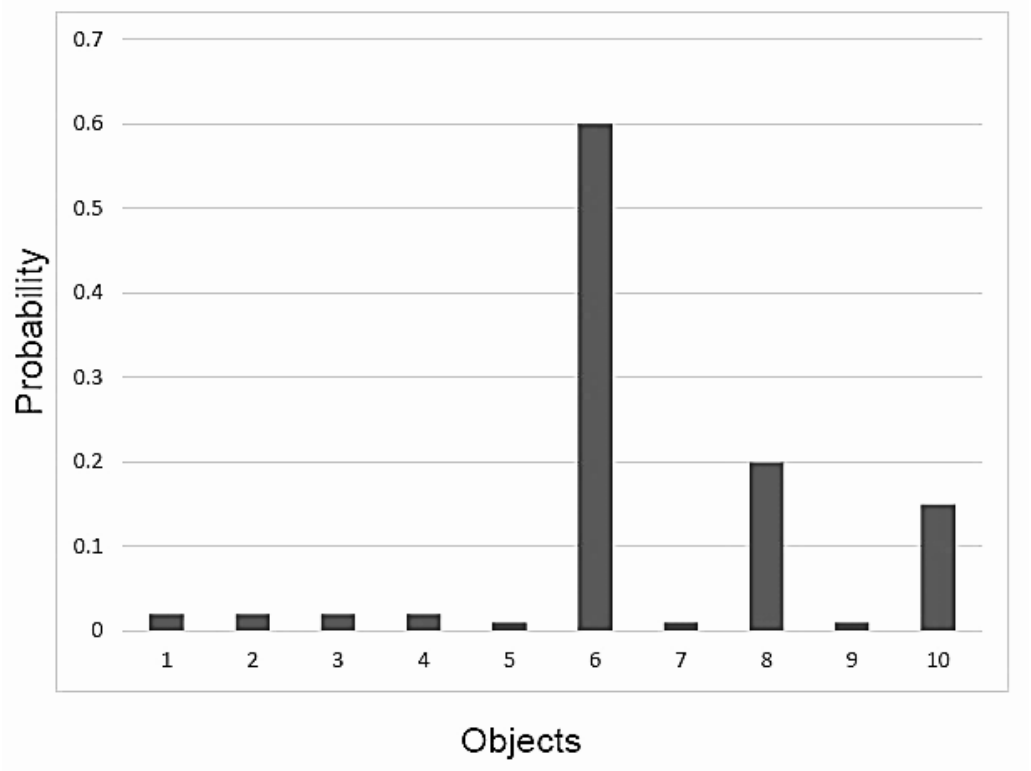


Figure 3.14: Probabilities for each object (animal) using Softmax

Training in the context of machine learning involves the learning process executed by the training loop (see Figure 3.15). In basic terms, it entails providing inputs to

a model, gathering its outputs, comparing them with the expected results, and adjusting the weights to ensure accurate outputs. The key element in this process is the unspecified component identified as:

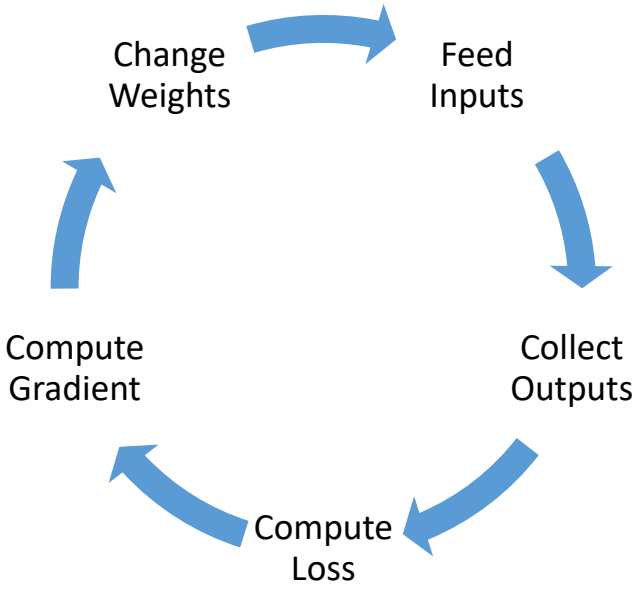


Figure 3.15: The training loop [54]

The loss function in machine learning evaluates the model's error by measuring its deviation from the correct outcomes. The goal is to minimise the loss, indicating increased accuracy in the model's predictions [54].

Epochs represent the completion of a full training cycle, where the model is trained with every available data example. The completion of a full circle through your dataset makes an epoch. Typically, models undergo tens to hundreds of epochs until their losses are minimised.

The training process involves a mathematical technique called backpropagation, which is a more complex version of the simplified procedure described. Backpropagation includes the concept of optimisers like SGD, RMSprop, and Adam, among others, which enhance the efficiency of backpropagation.

Learning Rate, a parameter of optimisers, determines the size of weight updates during training, with common values being 0.01, 0.001, and 0.0001.

Batches are employed when the dataset is too large to fit into memory or a GPU, with subsets (batches) used to compute gradients, enhancing computational efficiency [54].

The Cost Function in building a network assesses the accuracy of the model's predictions by comparing them to the actual values. It serves to penalise the network for errors, aiming to minimise the disparity between predicted and actual outputs during the training process [67].

During the network operation, the goal is to enhance prediction accuracy and decrease errors, ultimately minimising the cost function. The most optimised output is achieved when the cost or loss function attains the lowest value. If the cost function is defined as the mean squared error, it can be expressed as:

$$C = \frac{1}{m}\sum(y - a)^2 \quad \textbf{(3.9)}$$

where C is the cost function, m is the number of training inputs, a is the predicted value, and y is the actual value for a specific example.
The learning process is centred around the objective of minimising the cost.

Gradient Descent is an optimisation algorithm employed to minimise the cost in a process analogous to descending a hill. Intuitively, the approach involves taking small steps downward, akin to walking down a hill rather than making a single large jump. Starting from a point x, the algorithm moves down by a small amount (delta h), updating the position to x - delta h. This process is iteratively repeated until reaching the bottom, representing the point of minimum cost.

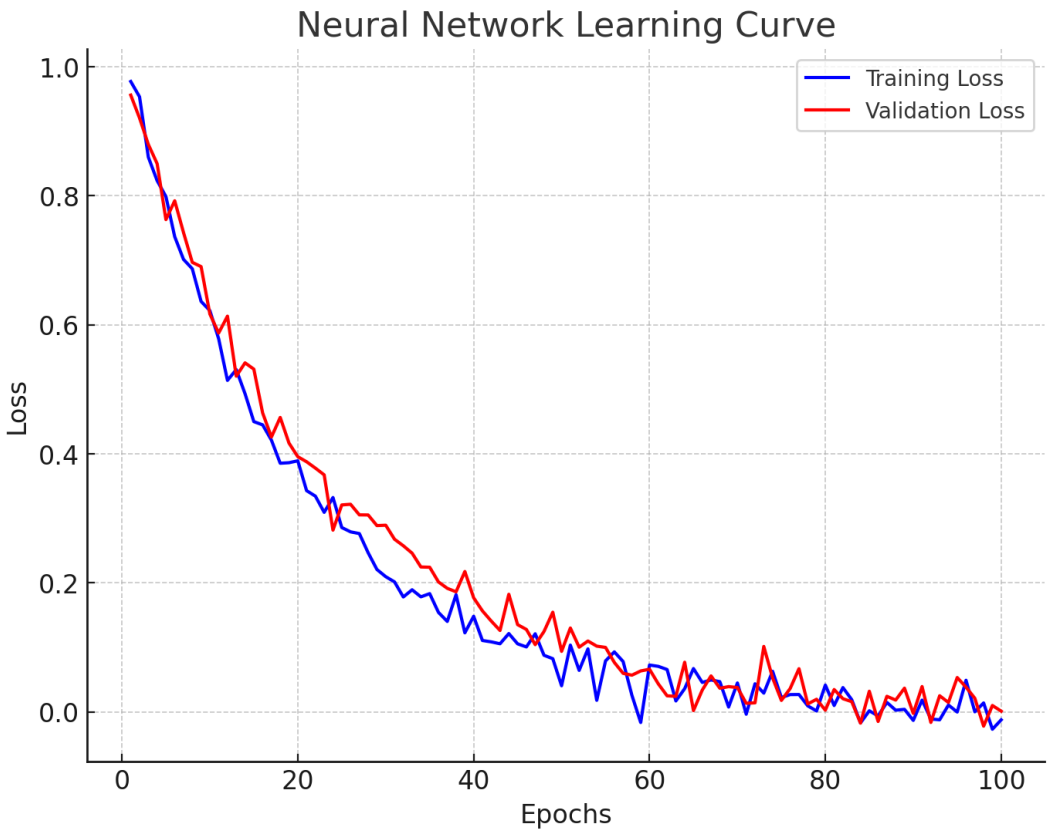


Figure 3.16: Gradient descent

In mathematical terms, to locate the local minimum of a function, one follows steps that are proportional to the negative of the function's gradient. The learning rate is the degree of reduction in the cost function at each iteration. It represents the pace at which the descent toward the minima of the cost function occurs. Selecting an appropriate learning rate is crucial, as it should not be too large, risking the oversight of the optimal solution, nor too low, causing prolonged convergence time for the network.

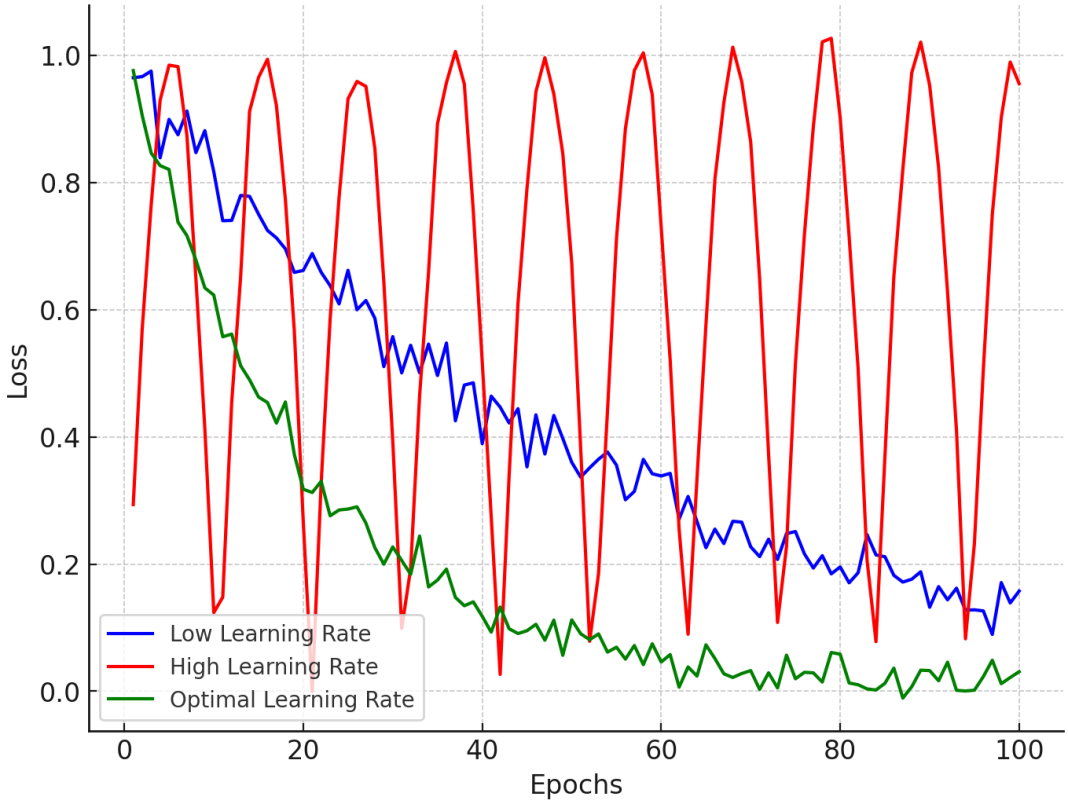


Figure 3.17: Learning rate comparison in neural network training

Backpropagation is a process in neural networks where initially assigned random weights and biases are adjusted based on the calculated error during an iteration. The error, along with the gradient of the cost function, is propagated backwards through the network—from the output layer through the hidden layers. This backward movement enables the update of weights in such a way that errors in subsequent iterations are minimised. The key concept is to iteratively refine the network by adjusting weights using the gradient of the cost function, ultimately improving the model's performance.

During neural network training, the input data is divided into batches rather than being processed all at once. This batch-wise training enhances model generalisation compared to training on the entire dataset simultaneously.

3.8 Epoch

An epoch represents a single training iteration involving both forward and backward passes through all batches. The number of epochs chosen for network training is a user-defined parameter. While a higher number of epochs may lead to increased accuracy, it also extends the time for convergence. However, caution is needed to avoid overfitting if the number of epochs is excessively high. [68]

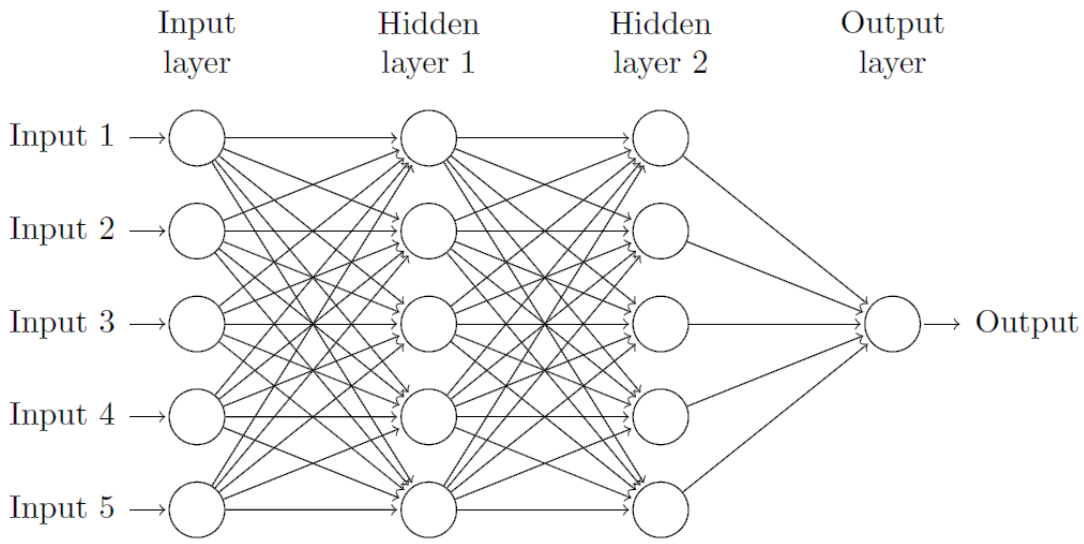


Figure 3.18: Standard Neural Network

3.9 Dropout

Dropout is a regularisation technique aimed at preventing overfitting in neural networks. During training, a random set of neurons in the hidden layer is deactivated, leading to training on various neural network architectures with different combinations of neurons. Dropout functions as an ensemble technique, utilising the outputs from multiple networks to produce the final output. Figure 3.18 depicts a standard Neural Network, and Figure 3.19 demonstrates a Neural Network where dropout has been applied.

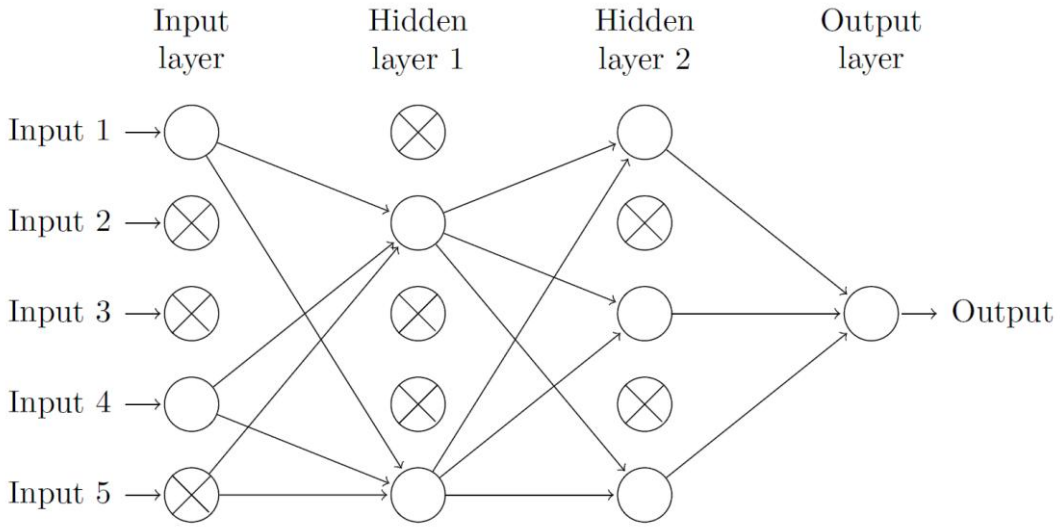


Figure 3.19: Dropout

3.10 Batch Normalisation

Batch Normalisation can be conceptualised as setting checkpoints in a river to ensure a consistent data distribution for the subsequent layer. During neural network training, weights change after each gradient descent step, influencing the shape of the data sent to the next layer. Batch Normalisation (BN) helps maintain a stable data distribution, contributing to more effective training. BN is a technique used to improve the training of deep neural networks by normalising the inputs of each layer. Listed are the key steps and calculations involved in Batch Normalisation:

3.10.1. Compute the Mean and Variance

For a mini-batch of data, compute the mean μ, and variance σ^2 for each feature:

$$\boldsymbol{\mu = \frac{1}{m}\sum_{i=1}^{m} x_i} \quad \textbf{(3.10)}$$

$$\boldsymbol{\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2} \quad \textbf{(3.11)}$$

where m is the batch size and x are the inputs in the batch.

3.10.2. Normalise the Inputs

Using the mean and variance, normalise the inputs:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad \textbf{(3.12)}$$

Here, ϵ is a small constant added for numerical stability.

3.10.3. Scale and Shift

After normalisation, the output is scaled and shifted using learnable parameters γ (scale) and β (shift):

$$\mathbf{y_i = \gamma \hat{x}_i + \beta} \quad \textbf{(3.13)}$$

3.10.4. During Training

During training, you update the running estimates of the mean and variance using momentum:

Running Mean ($\mu_{running}$)

$$\mu^2_{running} = (1 - momentum) * \mu_{running} + momentum * \sigma_{batch} \quad \textbf{(3.14)}$$

Running Variance ($\sigma^2{}_{running}$)

$$\boldsymbol{\sigma^2_{running} = (1 - momentum) * \sigma^2_{running} + momentum * \sigma^2_{batch}} \quad \textbf{(3.15)}$$

3.10.5. During Inference

At inference time, use the running mean and variance instead of the minibatch statistics for normalisation.

These steps help stabilise and accelerate training, leading to improved convergence and performance in deep learning models.

3.11 Convolutional Neural Networks: Extracting Features from Images

In a CNN, a filter functions as a weight matrix applied to portions of an input image to produce a convoluted output. For instance, in an image of size 28*28,* a randomly assigned 3x*3* filter is multiplied with various 3x3 sections of the image, generating a convoluted output. Typically, the filter size is smaller than the original image size. Like weights, filter values are updated during backpropagation to minimise the cost.

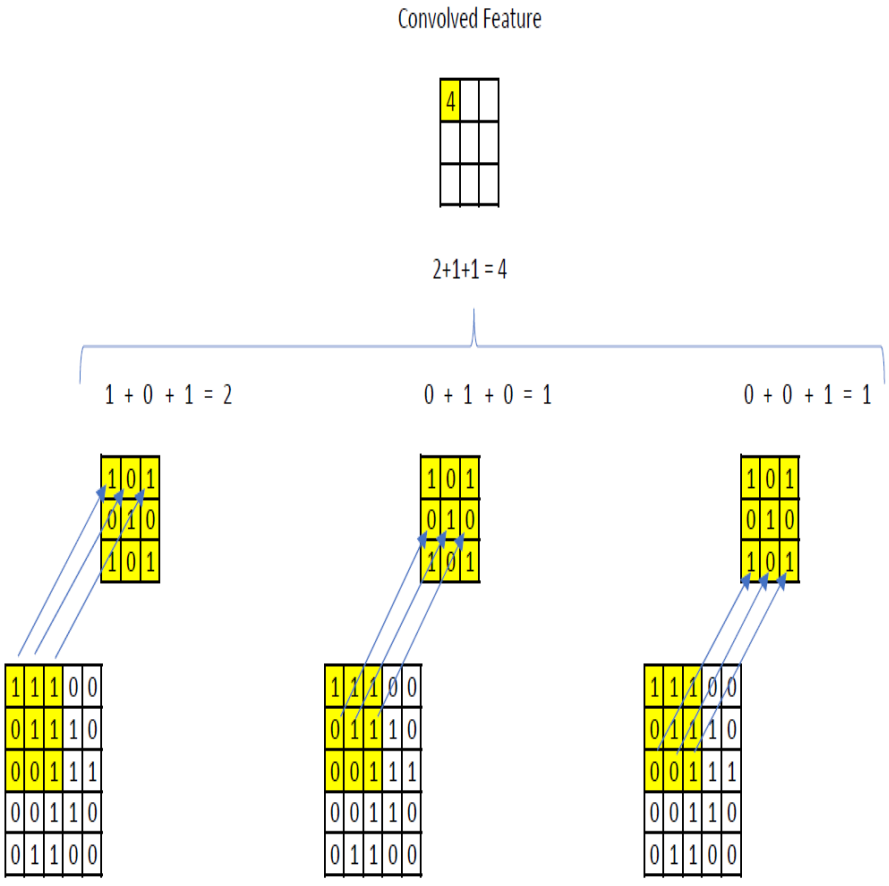


Figure 3.20: Filtering

Figure 3.20 shows how a filter is used to form the convoluted output.

Figure 3.21 shows how a 3 x 3 filter is multiplied with each 3 x 3 section to form a convolved feature.

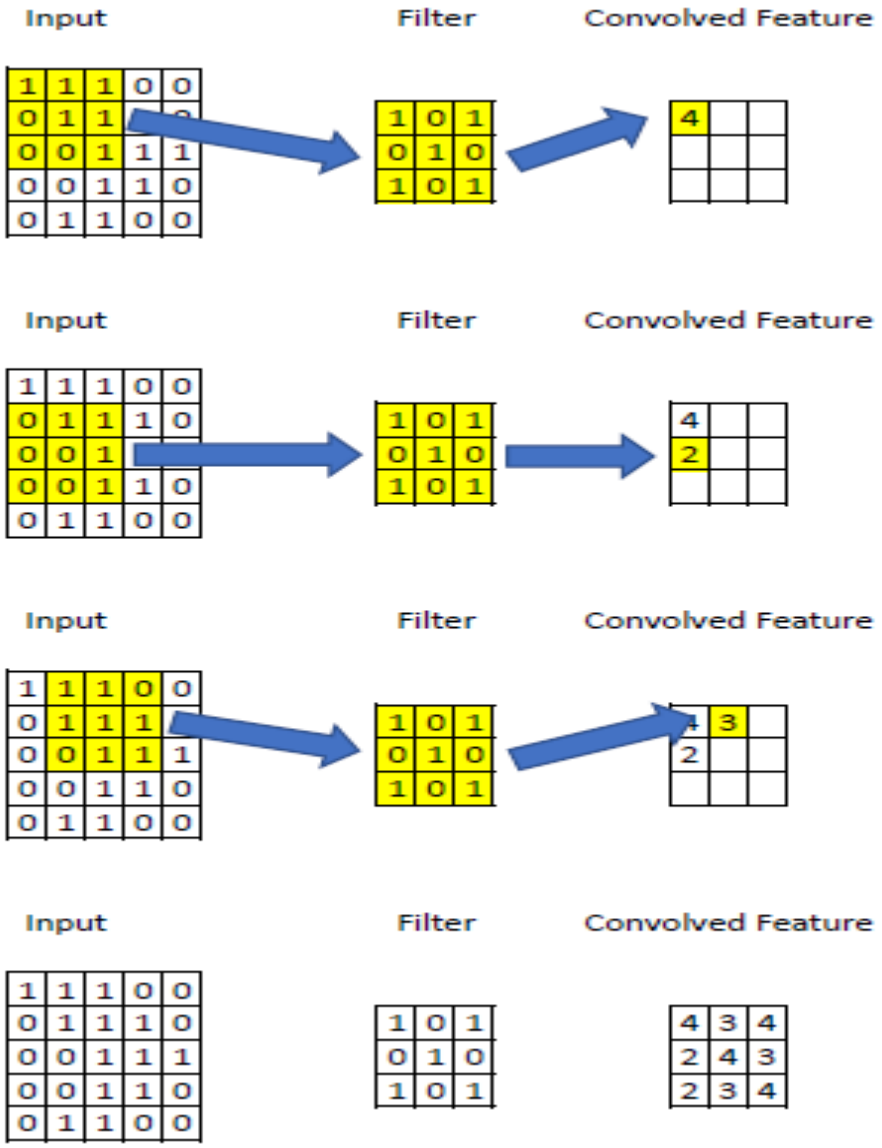


Figure 3.21: Three x three filter applied

3.12 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are primarily designed for image data. In a scenario where the input is size (28 x 28 x 3), using a regular neural network would result in many parameters, such as 2352 x (28 x 28 x 3). To mitigate this, CNNs employ a convolutional operation, reducing the parameter count by convolving the images with filters. These filters slide over the width and height of the input volume, producing a two-dimensional activation map at each position.

Stacking these activation maps along the depth dimension yields the final output volume.

3.13 Data Handling Techniques: Augmentation, Padding, and Model Design

Data handling techniques refer to practical techniques such as data augmentation, padding, and the overall design and development of machine learning models.

3.13.1. Pooling layers

Pooling layers are frequently inserted between convolution layers in neural networks to decrease the number of parameters and mitigate overfitting. The most popular form of pooling is using a 2 x 2 filter size and employing the MAX operation, which involves taking the maximum value from each 4 x 4 matrix of the original image.

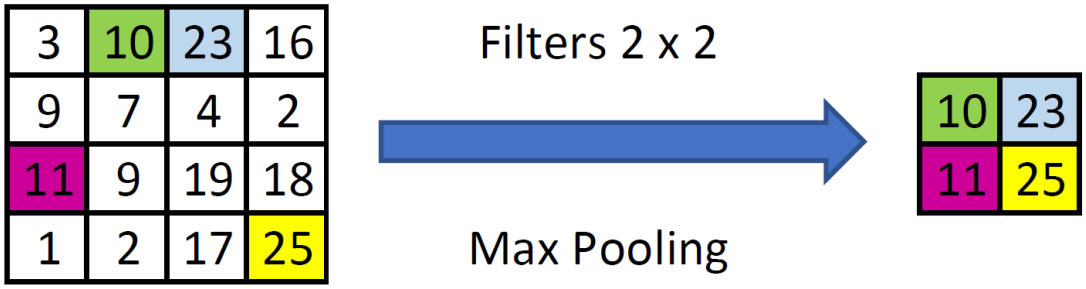


Figure 3.22: Max Pooling with 2 x 2 filter and a stride of two

3.13.2. Padding

Padding involves adding additional layers of zeros around images to ensure that the output image maintains the same size as the input. This technique is commonly referred to as "same padding."

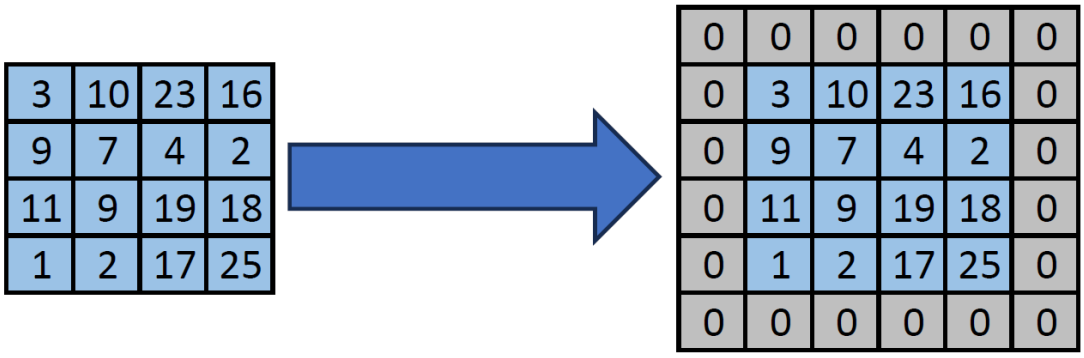


Figure 3.23: Padding of image data

In same padding, the convolved layer retains the same size as the original image after filter application. Valid padding, on the other hand, maintains only the valid pixels of the image, resulting in a reduction in the length and width of the output at each convolutional layer.

3.13.3. Data Augmentation

Data augmentation involves generating new data from existing data to potentially improve prediction accuracy. For instance, adjusting brightness in a dark image can enhance visibility, or rotating a slightly tilted digit can aid in digit recognition. These adjustments improve data quality and are examples of data augmentation techniques.

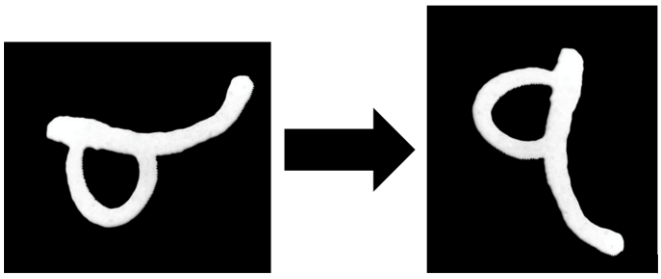


Figure 3.24: Rotating a digit to enhance recognition

3.14 Design and development of the neural network.

Machine learning (ML) is also a type of Artificial intelligence (AI) which can learn without explicit programming by feeding it data. By looking at patterns in the data, a program called a model or algorithm can be developed. The model can learn, change and grow when exposed to new data.

The development of an ML Model flowchart can be seen in Figure 3.25 [69].

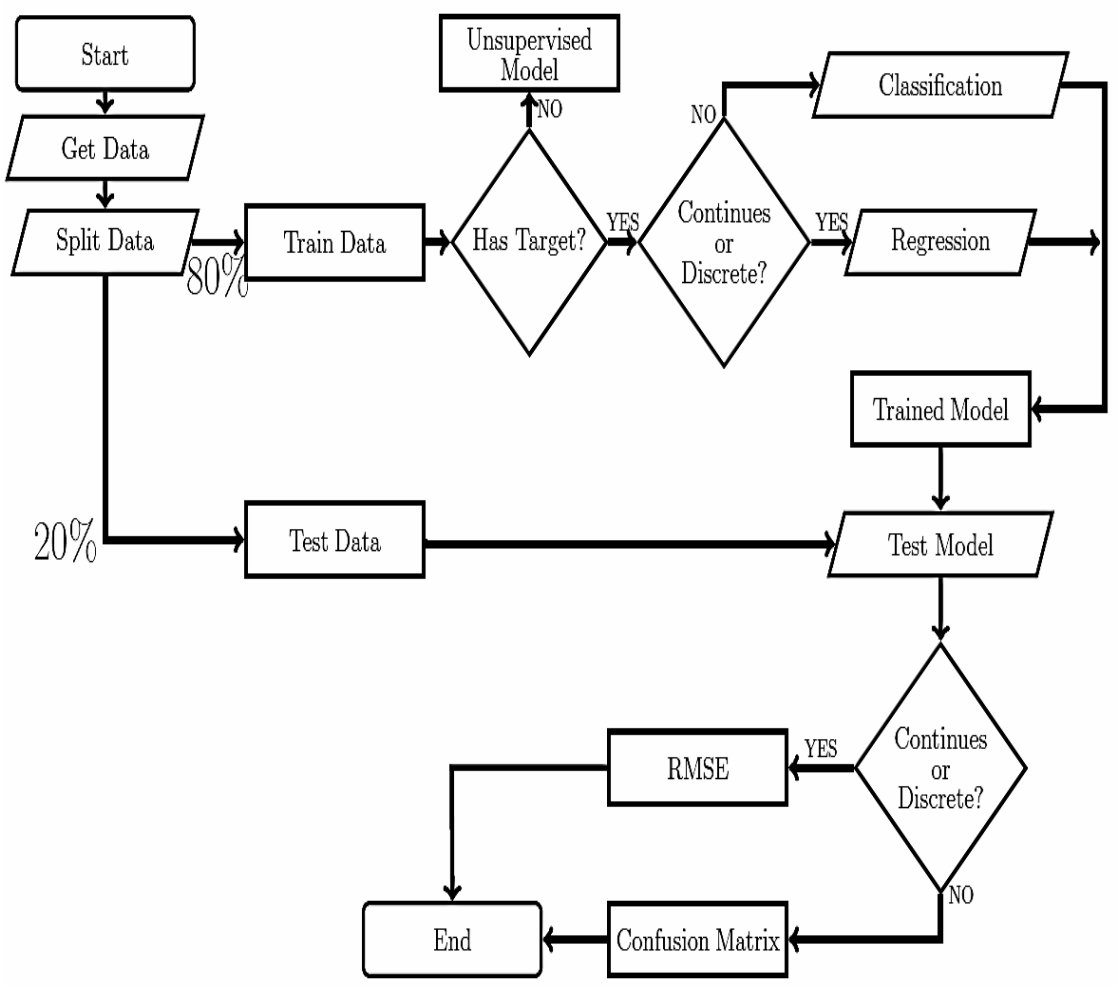


Figure 3.25: Development of an ML Model flowchart

The identification of the animals was done with the aid of a pre-trained deep neural network. Deep neural networks are popular for their use and success in computer vision. To train a neural network, a database of training data is required. In this study, it consisted of images of different animals. Furthermore, validation data was also required for the verification of the neural network performance.

The possibility of using open-source images and pre-trained models for use with the neural network design was investigated and tested. By using these pre-trained neural networks, a deep learning neural network could be developed by using transfer learning. This could notably reduce the computational cost and training time of a new neural network.

CHAPTER 4: YOU LOOK ONLY ONCE

4.1 Introduction

"You Only Look Once" (YOLO) uses Convolutional Neural Networks (CNN) to perform object detection. YOLO, introduced by Joseph Redmon et al. in 2015, aimed to address issues present in the object recognition models of that period [70]. One notable model, Fast R-CNN, although state-of-the-art at the time, had significant limitations, such as slow prediction times of 2-3 seconds per image, making it unsuitable for real-time applications. In contrast, YOLO simplifies the process by requiring only a single forward pass through the network to generate final predictions, thus enabling real-time object detection. The network uses the full image to identify all objects in the image in one pass. It is thus possible for YOLO to detect multiple objects in an image [71], [72].

A single Neural Network is applied to the input image. The early convolutional layers extract image features, while the fully connected layers generate predictions for output probabilities and bounding box coordinates. The image is divided into grid cells, and the Neural Network produces cell probabilities. Features from the entire image are used in the prediction of each bounding box for all objects. At the same time, all bounding boxes for all classes in an image are predicted and the best ones are selected according to the predicted probabilities. Using non-max suppression, all recognised objects as well as bounding boxes are given as outputs. The location of these bounding boxes in the image is known. YOLO allows end-to-end training and real-time speeds with high average precision [73], [74] and [75].

The architecture of our network was inspired by the GoogLeNet model for image classification [34]. It consisted of 24 convolutional layers, followed by 2 fully connected layers. Unlike GoogLeNet's inception modules, we employed 1 × 1 reduction layers followed by 3 × 3 convolutional layers. A complete overview of the network is provided in Figure 4.1.

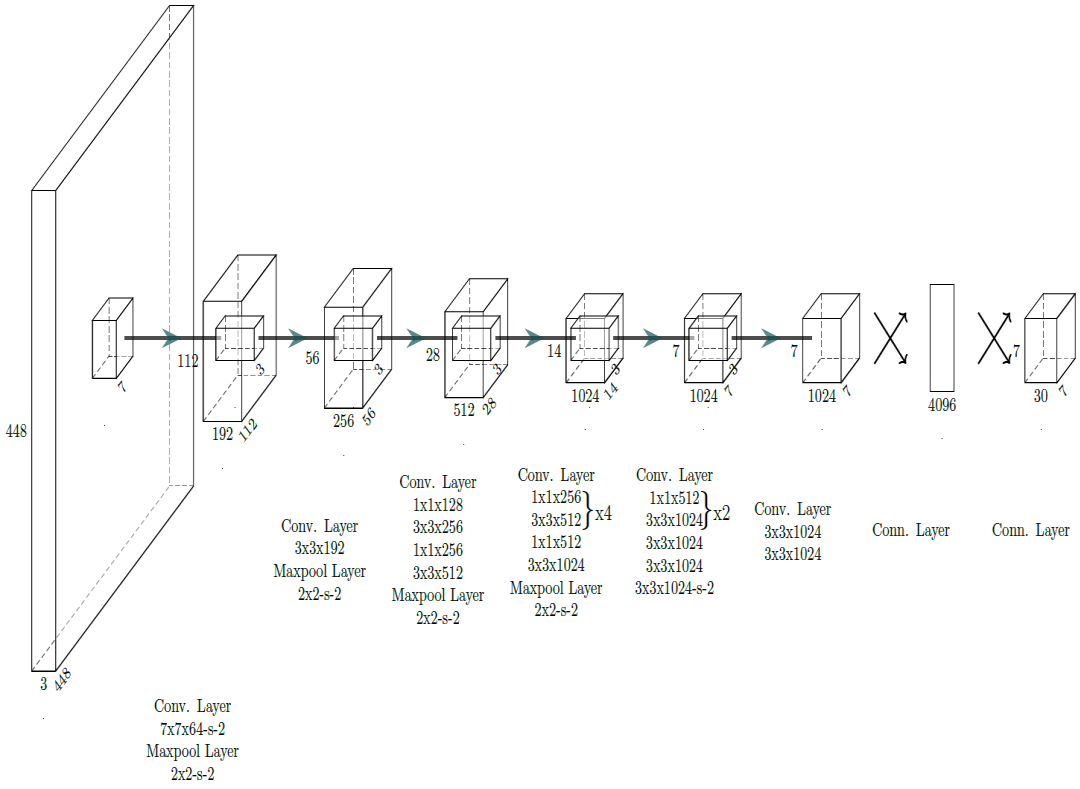


Figure 4.1: YOLO architecture

4.2 YOLOv3 Architecture

YOLOv3 uses the Darknet framework as a backbone for feature extraction, stacking 53 layers to create a fully convolutional network with 106 layers. Darknet consists of 53 layers, and detections are made at layers 82, 94 and 106. Each of the 53 layers is followed by a batch normalisation layer and uses the Leaky ReLU activation function. The architecture divides the image into S x S grid of cells, with each cell responsible for predicting one object [76], [77], using factors 32, 16 and 8. The inputs are downsampled by the network. These factors or numbers are called the network strides and are the outputs at layers 82, 94 and 106.

The YOLOv3 model takes a batch of n images with dimensions (n, 416, 416, 3), where 416 x 416 is the image size and 3 represents the RGB channels. If a network input or input image, in this case, is 416 x 416 pixels and a network stride of 32 is used, an output size of 13 x 13 will be realised. A 16 stride will have an output of 26 x 26, and eight will produce an output size of 52 x 52. The outputs are 13 x 13 for large objects, 26 x 26 for medium objects, and 52 x 52 for small objects, correlating to the respective strides.

The architecture uses no pooling layers and employs a stride of 2 in convolutional layers for downsampling feature maps. Each convolutional layer applies multiple filters to create various feature maps. Predictions occur at the 82nd, 94th, and 106th layers, with strides of 32, 16, and 8, respectively.

4.2.1. Detection Kernels

In YOLOv3, kernels refer to the small matrices of weights used in convolutional layers to process input data. These kernels, also known as filters, slide over the input image (or feature maps) to detect patterns, features, and structures. The kernel sizes are usually small, e.g. 3×3, 5×5, or 7×7. Kernel values are learned during training through backpropagation, and they are trained to detect certain patterns like edges, corners, texture and colour transition.

4.2.2. Convolutional Operation

Kernels are essential to the convolutional operation, where they are applied to the input data to produce feature maps. As the kernel progresses across the image, it performs element-wise multiplications and summations, effectively highlighting specific features such as edges, textures, and shapes.

4.2.3. Learnable Parameters

The values in the kernels are initially randomly set and are updated during the training process through backpropagation. As the model learns from the training data, these weights are adjusted to optimise the detection performance, allowing the kernels to become sensitive to relevant features for object detection.

4.2.4. Kernel Size and Stride

The size of a kernel determines how many pixels it covers during the convolution operation. Common sizes include 3x3 or 5x5. The stride indicates how far the kernel moves across the input data after each operation. Smaller strides result in more overlapping regions and finer detail, while larger strides reduce the computational load but may skip important features.

4.2.5. Multiple Kernels

YOLOv3 uses multiple kernels in each convolutional layer to extract a variety of features from the input image. Each kernel learns to detect different features, allowing the model to capture various object characteristics effectively.

4.2.6. Feature Extraction

By stacking multiple convolutional layers with various kernels, YOLOv3 can extract hierarchical features from the input image, facilitating the identification and localisation of objects at different scales and complexities.

In summary, kernels in YOLOv3 are essential components of the convolutional layers that help the model learn and extract meaningful features from images, playing a crucial role in the overall object detection process.

1 x 1 Detection Kernel Convolutions are applied to the down sampled images (13 x 13, 26 x 26 and 52 x 52 pixels) at the following layers: 82, 94 and 106, respectively, which allows the resultant feature maps to have same spatial dimensions.
Equation 4.1 is used to calculate the detection kernel's depth.

$$\boldsymbol{detection\ kernel\ depth} = (\boldsymbol{b} * (\mathbf{5} + \boldsymbol{c})) \quad \textbf{(4.1)}$$

where b = number of bounding boxes per cell of produced feature maps, and c is the number of classes.

4.2.7. Feature maps:

In YOLOv3, feature maps are the outputs generated by the convolutional layers of the network, representing different levels of information extracted from the input image. These maps are crucial for object detection, as they enable the model to identify and localise objects based on learned features. The following are key aspects of feature maps in YOLOv3:

- **Hierarchical Representation**: Feature maps capture hierarchical information about the input image. As the image passes through the layers of the network, the feature maps evolve from capturing basic features like edges and textures to more complex patterns and object shapes.
- **Multi-Scale Detection**: YOLOv3 employs a multi-scale detection strategy by generating feature maps at three distinct resolutions. This allows the model to effectively detect objects of varying sizes. Smaller objects are identified in finer resolution maps, while larger objects are better represented in coarser maps. The three resolutions correspond to feature maps sized 13 x 13, 26 x 26, and 52 x 52.
- **Object Predictions**: Each feature map is responsible for predicting bounding boxes, class probabilities, and objectness scores (OS). The objectness score indicates the probability for a cell to contain an object. Once the OS is assigned, Yolo will filter the cells based on an objectness threshold. Assuming that the threshold is set at 0.8, only those cells that have an OS larger than the threshold will be considered for further analysis for the objects it detects [78]. The information contained in these maps enables YOLOv3 and YOLOv4 to ascertain the presence and location of multiple objects within an image.
- **Integration of Features**: YOLOv3 utilises skip connections to combine features from earlier layers with those from deeper layers. This integration helps preserve spatial information while enhancing the model's ability to detect objects by leveraging both high-level abstract features and low-level detailed features.

In summary, feature maps in YOLOv3 are integral to the model's ability to perform accurate object detection by providing layered representations of the input image that capture essential details at multiple scales.

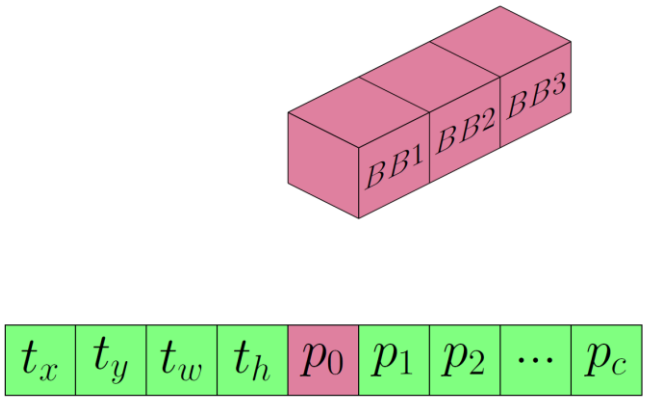


Figure 4.2: Resultant feature map

4.2.8. Bounding boxes

Each bounding box has the following attributes:

$$\textbf{Bounding box attributes} = (5 + c) \quad \textbf{(4.2)}$$

t_x = centre of bounding box x-axis.

t_y = centre of bounding box y-axis.

t_w = width of bounding box

t_h = height of bounding box

p_0 = objectness score

p_1 – p_c = list of confidences of every class that this object might belong to.

Common data sets used for training YOLOv3 networks are the COCO dataset, which has 80 classes. Therefore, c is 80, and the total number of attributes for each bounding box will thus be:

Total number of attributes for each bounding box $= (5 + c)$ **(4.3)**

$= (5 + 80)$

$= 85$

Each bounding box has attributes assigned across three layers in the YOLOv3 model. Therefore, the attribute count needs to be multiplied by 3 to cover all layers. The total attributes for all bounding boxes are then:

Attributes per bounding box at the *three* layers $= (3 * (5 + 80))$ **(4.4)**

$= 255$

The shape of the feature maps produced at the *three* different layers is: (13 x 13 x 255), (26 x 26 x 255) and (52 x 52 x 255).

4.2.9. Grid Cells (Detection Cells)

YOLOv3 produces three bounding boxes for every cell of the feature map. If the centre of one of the bounding boxes is in the cell's receptive field, it predicts an object. Training requires the identification of the cell that falls in the centre of the object. Each grid cell predicts five parameters to define a bounding box: x and y (coordinates of the centre), w and h (width and height), and confidence (indicating the presence or absence of an object). Additionally, each grid cell has class probabilities. To prevent the algorithm from predicting multiple bounding boxes for the same object, Non-Max Suppression is employed [79].

Figure 4.3: Original image

Figure 4.3 shows the resized image of 416 x 416 pixels before the YOLOv3 detection is done.

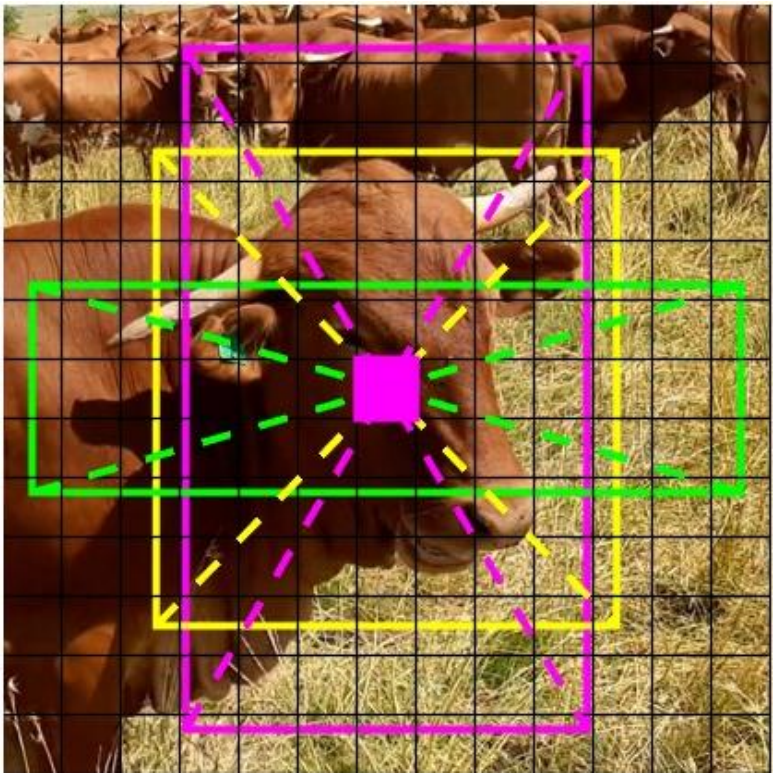


Figure 4.4: Resized image with 13 x 13 grid

Typical anchor boxes can be seen in Figure 4.4; however, the anchor boxes used for some of the tests can be seen in Figure 4.5.

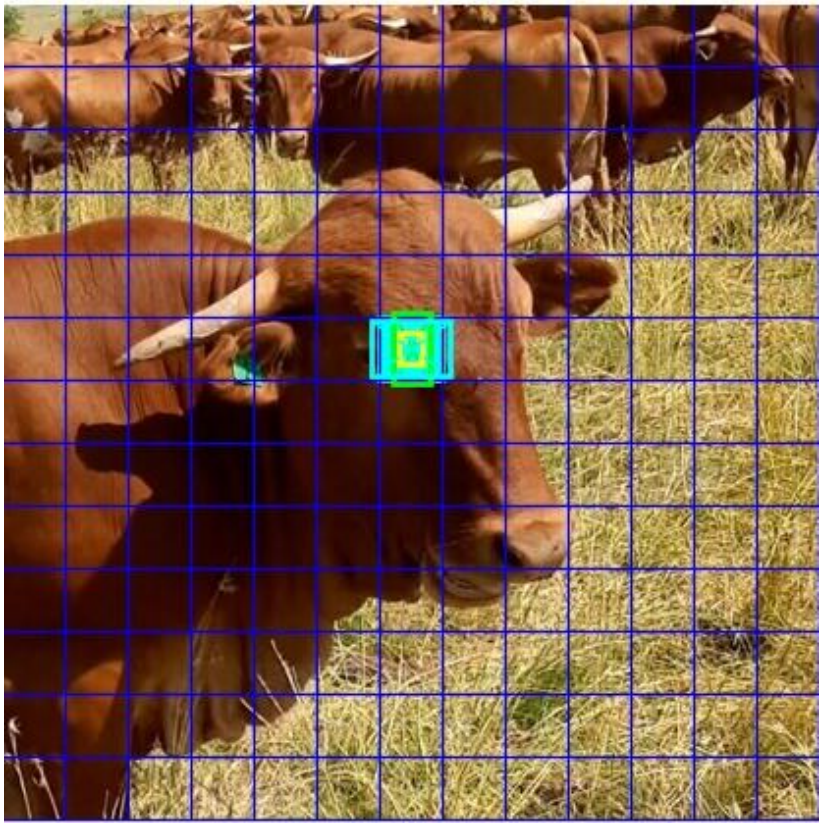


Figure 4.5: Resized image with a 13 x 13 grid with a bounding box

Figure 4.5 depicts the resized image with the 13 x 13 grid. The yellow, green and cyan boxes are the anchor boxes used for this grid, and the red rectangle is the determined bounding box with a probability score of 0.9999 for cattle head. The anchor boxes for this grid are 12 x 16, 19 x 36 and 40 x 28 pixels, respectively.

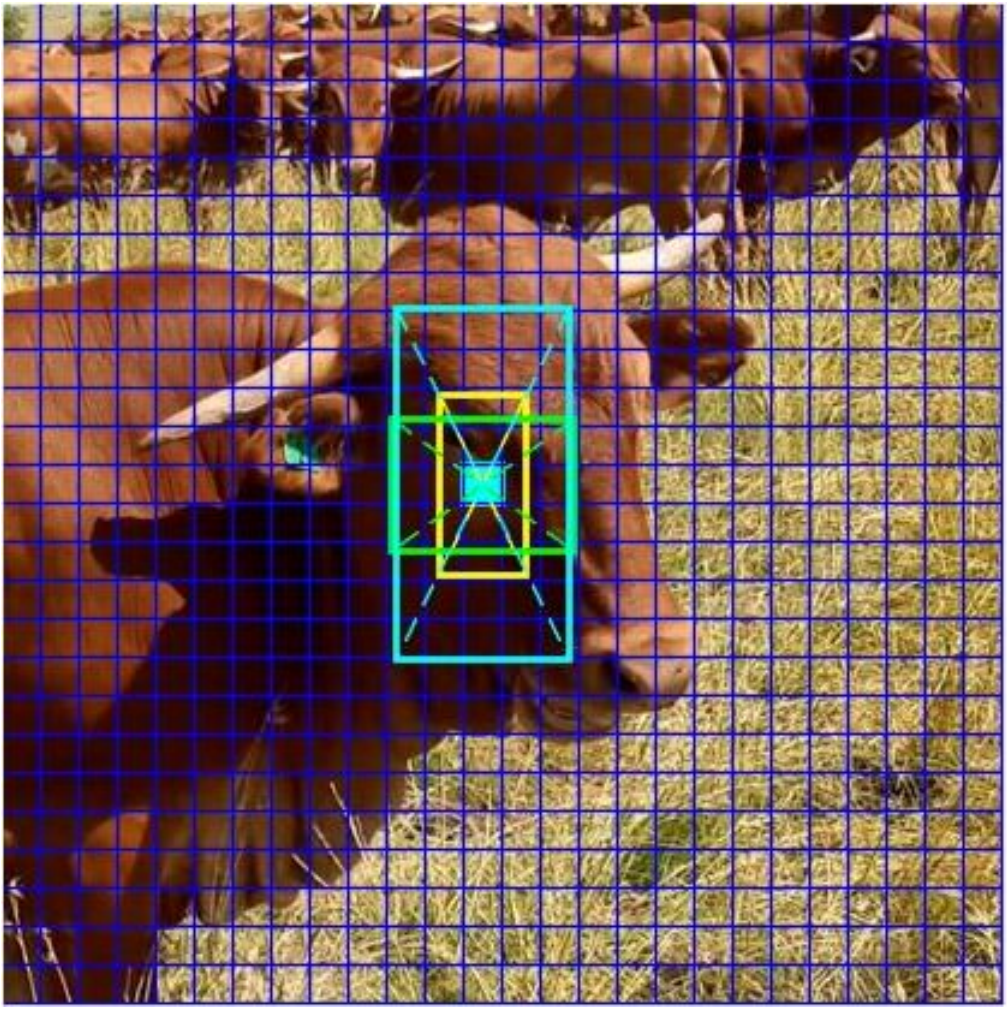


Figure 4.6: Resized image with 26 x 26 grid

Figure 4.6 depicts the resized image with the 26 x 26 grid. The anchor boxes for this grid are 36 x 75, 76 x 55 and 72 x 146 pixels, respectively.

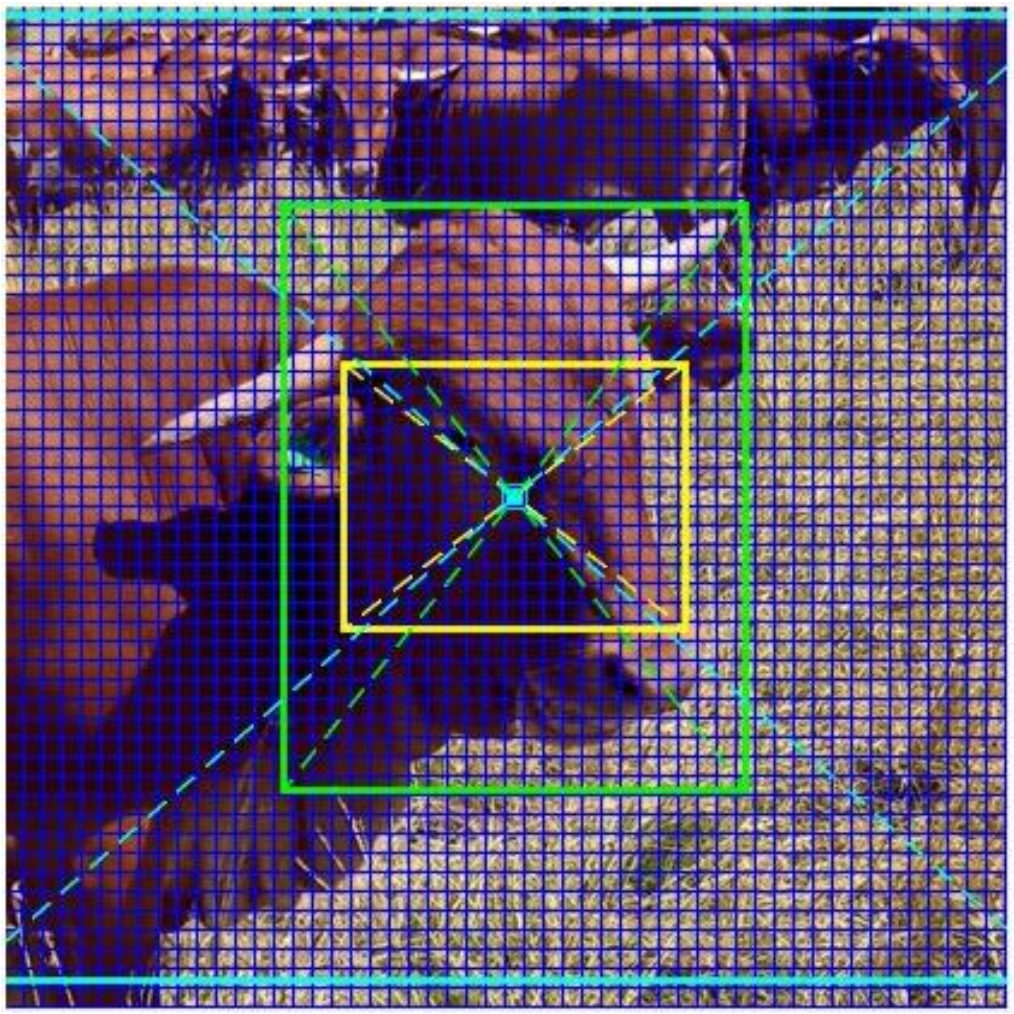


Figure 4.7: Resized image with 52 x 52 grid

Figure 4.8 depicts the resized image with the 52 x 52 grid. The anchor boxes for this grid are 142 x 110, 192 x 243 and 459 x 401 pixels, respectively.

4.3 YOLOv3 Training

When training, it has one ground truth bounding box for detecting one object. We must identify which cells the object belongs to. Use scale 1 and stride 32 with an input image of 416 x 416, which is downscaled to a size of 13 x 13.

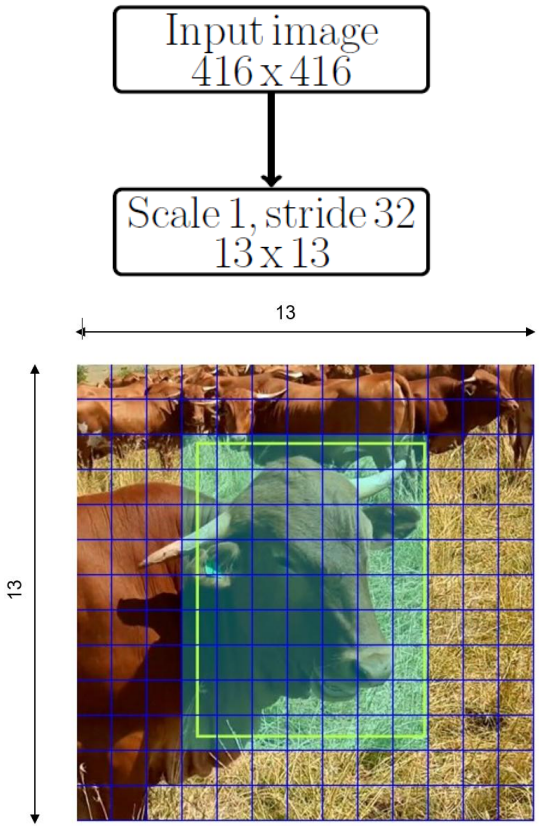


Figure 4.8: Ground truth bounding box

The grid in Figure 4.8 represents the output feature map. All cells to which the ground truth bounding box belongs are identified. The centre cell is assigned by YOLOv3 to be responsible for predicting the object. Objectness score for this cell is 1.

However, each cell produces three bounding boxes. How do we decide which bounding box to select?

4.3.1. Anchor Boxes (Demonstrates predefined priors)

To predict bounding boxes, YOLOv3 uses predefined bounding boxes of different sizes and aspect ratios, called Anchors or priors. Anchors are used to calculate predicted bounding boxes' real width and height. In total, nine anchor boxes are used; three anchor boxes for each scale. Each grid cell of every feature map on each scale output can thus output three bounding boxes using three anchor boxes.

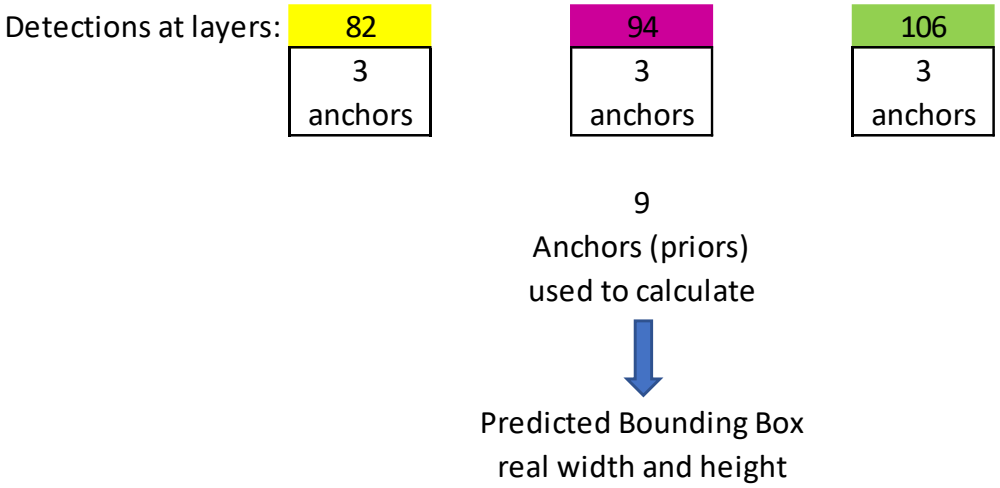


Figure 4.9: Detection at layers.

4.3.2. Calculating the anchors

To calculate the bounding boxes, k-means clustering is used. The widths and heights of the different scales are shown in Figure 4.10.

4.3.3. K-means Clustering

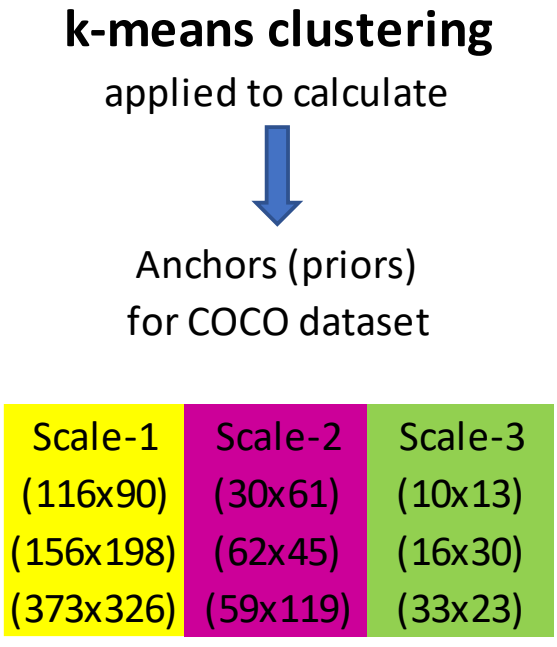


Figure 4.10: K-means clustering

	X	Y
13 x 13	16	12
	36	19
	28	40
26 x 26	75	36
	55	76
	146	72
52 x 52	110	142
	243	192
	401	459

Figure 4.11: Anchor box sizes for different scales [80]

In the context of YOLO (You Only Look Once), K-means clustering is used to determine the optimal anchor boxes for object detection. This process is depicted in Figure 4.12.

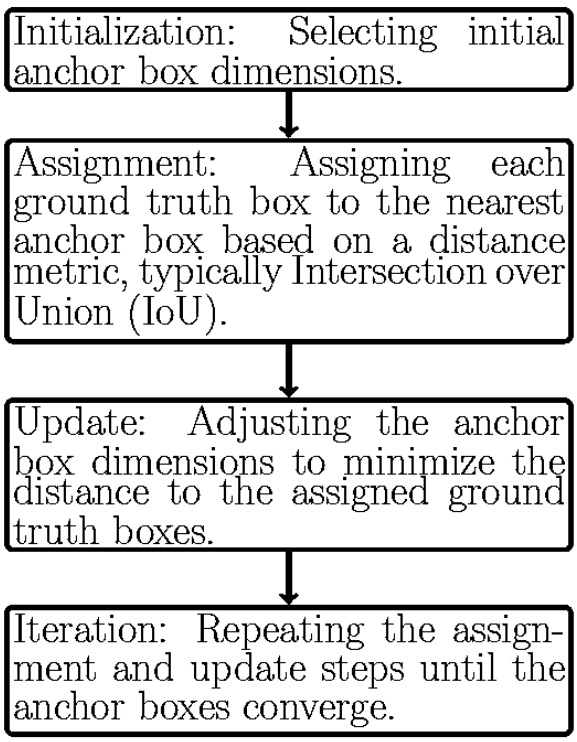


Figure 4.12: The process of K-means clustering

This approach helps in selecting anchor boxes that better match the objects in the dataset, improving detection accuracy and performance.

A cluster is a group of data points that are grouped due to their similarities. In the K-means algorithm, you determine a target number *k*, which represents the number of centroids needed in the dataset. A centroid is a point that signifies the centre of a cluster, either as an actual data point or a theoretical one.

4.3.4. The K-means algorithm operation

The K-means algorithm works by identifying *k* centroids and assigning each data point to the nearest cluster. This assignment is done in a way that minimises the sum of the squared distances within each cluster. Essentially, the goal is to make the clusters as tight as possible around their centroids. The term "means" in K-means refers to the process of averaging the data points to find these centroids. The K-means algorithm in data mining begins by randomly choosing an initial set of centroids. These centroids act as the starting points for each cluster. The algorithm then repetitively refines the positions of these centroids through iterative calculations to enhance the clustering. The process stops when one of the following conditions is met: either the centroids remain unchanged, indicating successful clustering, or a pre-determined number of iterations has been completed.

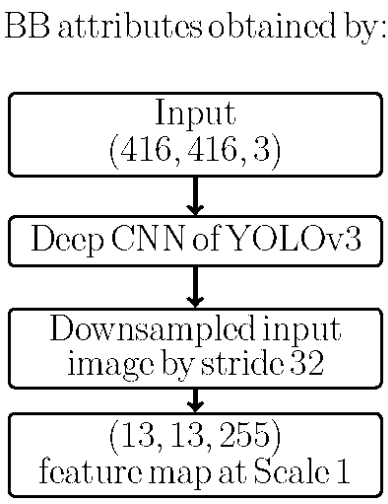


Figure 4.13: How BB attributes are obtained

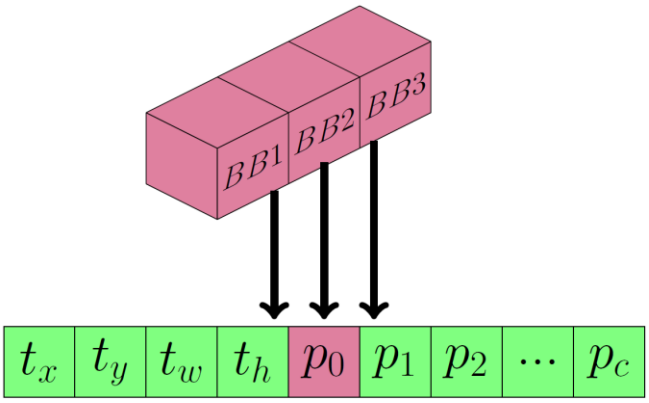


Figure 4.14: BB probabilities

Figure 4.14 indicates how the bounding box probabilities are calculated, and Figure 4.15 depicts the steps that need to be taken to identify the object in the picture.

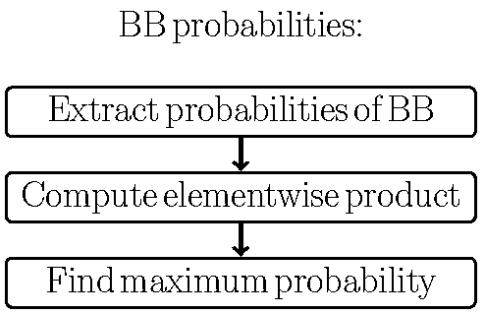
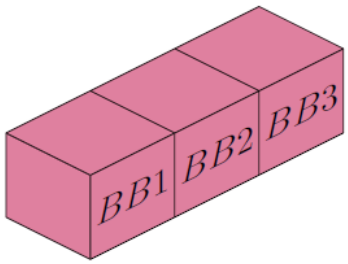


Figure 4.15: Steps to find maximum probability

Figure 4.16 indicates how the maximum probability for each bounding box is calculated.



$$BB_1 score = p_0 * \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_c \end{bmatrix} = \begin{bmatrix} 0.08 \\ 0.03 \\ \vdots \\ 0.05 \end{bmatrix} \xrightarrow{\text{MAX}} 0.55 \quad \text{class: horse}$$

$$BB_2 score = p_0 * \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_c \end{bmatrix} = \begin{bmatrix} 0.02 \\ 0.07 \\ \vdots \\ 0.09 \end{bmatrix} \xrightarrow{\text{MAX}} 0.33 \quad \text{class: dog}$$

$$BB_2 score = p_0 * \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_c \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.07 \\ \vdots \\ 0.05 \end{bmatrix} \xrightarrow{\text{MAX}} 0.48 \quad \text{class: cattle head}$$

Figure 4.16: Maximum bounding box probabilities

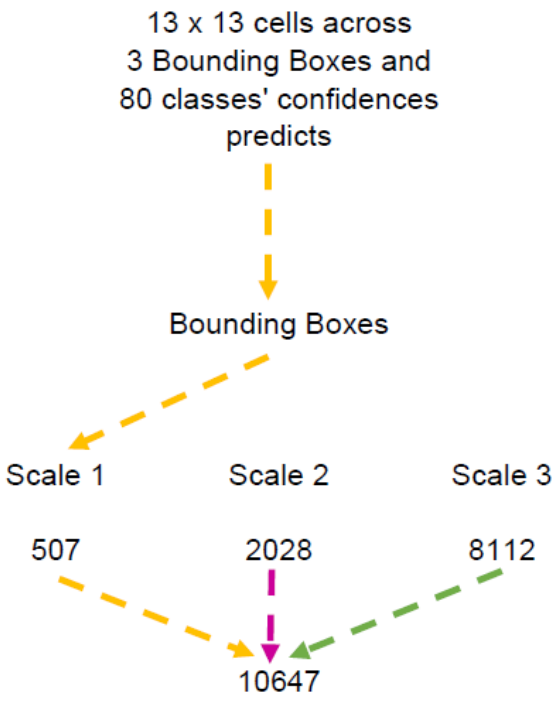


Figure 4.17: Filtered with non-maximum separation technique

Non-Max Suppression (NMS) is used to identify the most accurate bounding box for an object while eliminating or "suppressing" all other overlapping boxes. The process works iteratively, choosing the optimal bounding box, comparing overlaps, and discarding redundant boxes until no more boxes remain. NMS takes into consideration two key factors:

- The objectiveness score provided by the model
- The overlap, or Intersection over Union (IoU), between bounding boxes

The following is the process of selecting the best bounding box using NMS:

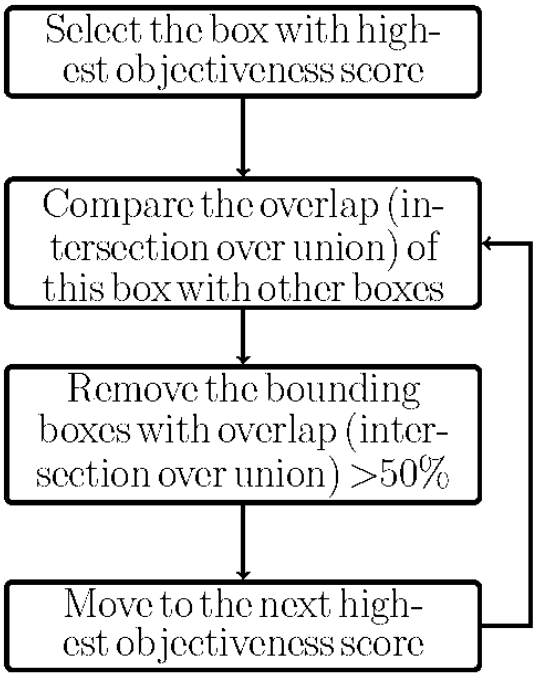


Figure 4.18: Non-max suppression code steps [81]

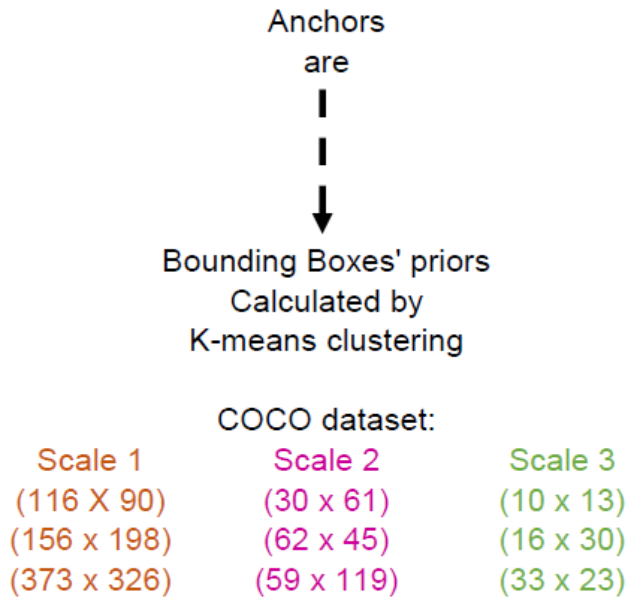


Figure 4.19: Anchors and Priors

To predict the BB:

- Calculate the offset to anchor, also called Log-space transform.
- Pass centre through the sigmoid function.

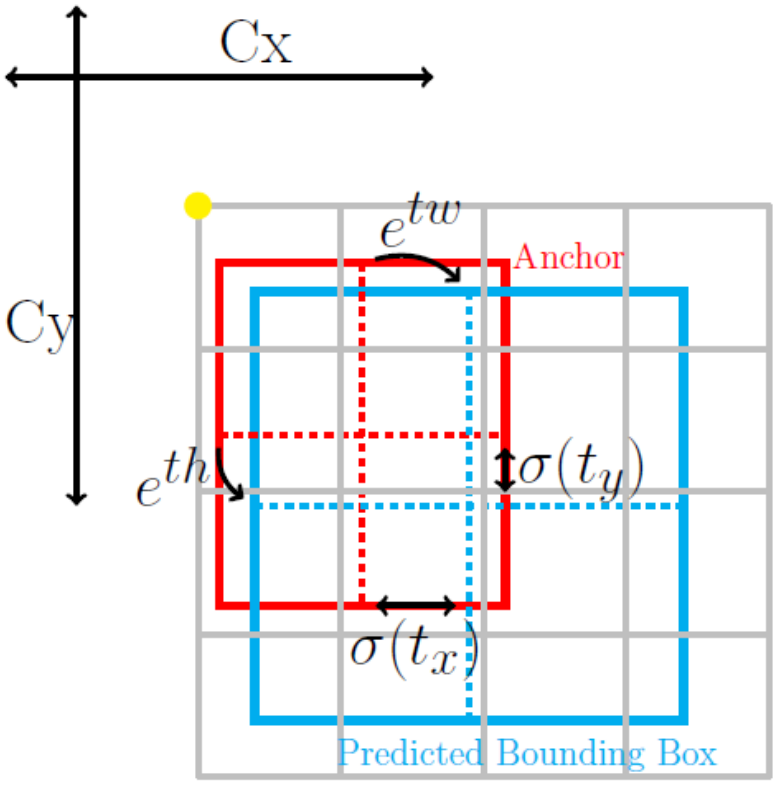


Figure 4.20: Bounding box prediction

Shown below are the equations to calculate the predicted bounding boxes in the image:

$$bx = \sigma(tx) + cx \quad \textbf{(4.5)}$$

$$by = \sigma(ty) + cy \quad \textbf{(4.6)}$$

$$bw = pw \text{ x } etw \quad \textbf{(4.7)}$$

$$bh = ph \text{ x } eth \quad \textbf{(4.8)}$$

where:

b_x, b_y, b_w, b_h = centre, width and height of predicted BB

t_x, t_y, t_w, t_h = outputs of NN after training

c_x, c_y = cell's top left corner of the anchor box (see yellow dot, top left)

p_w, p_h = anchor's width and height

When YOLOv3 is trained, it has one ground truth bounding box.

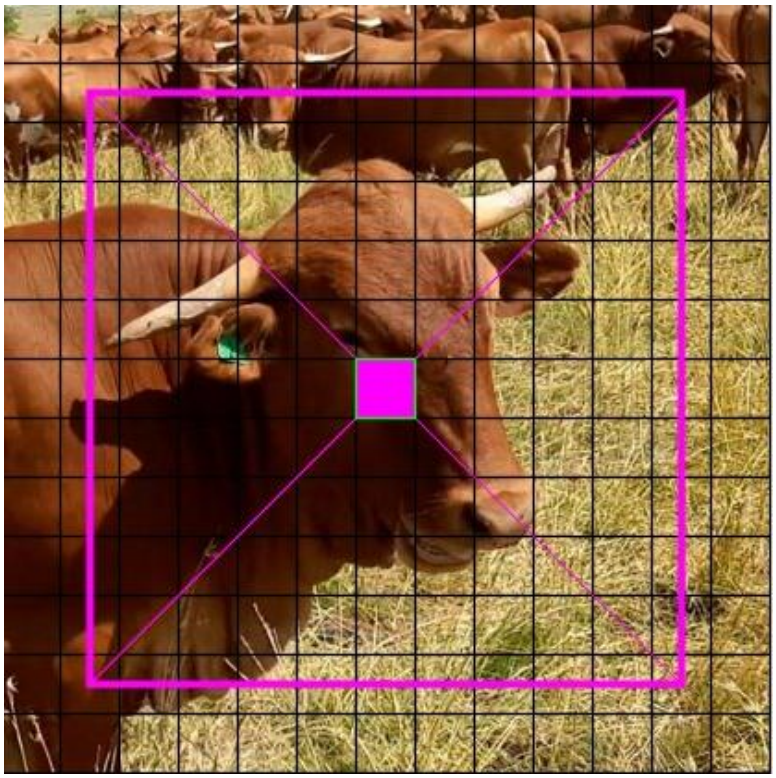


Figure 4.21: One centre cell responsible for this object

Weights of the network are trained to predict the centre cell and bounding box coordinates as accurately as possible. After training and forward pass, the network outputs coordinate t_x, t_y, t_w, t_h as well as c_x, c_y, which are the top left corner of the cell in the grid of the appropriate anchor box (see yellow dot in Figure 4.22).

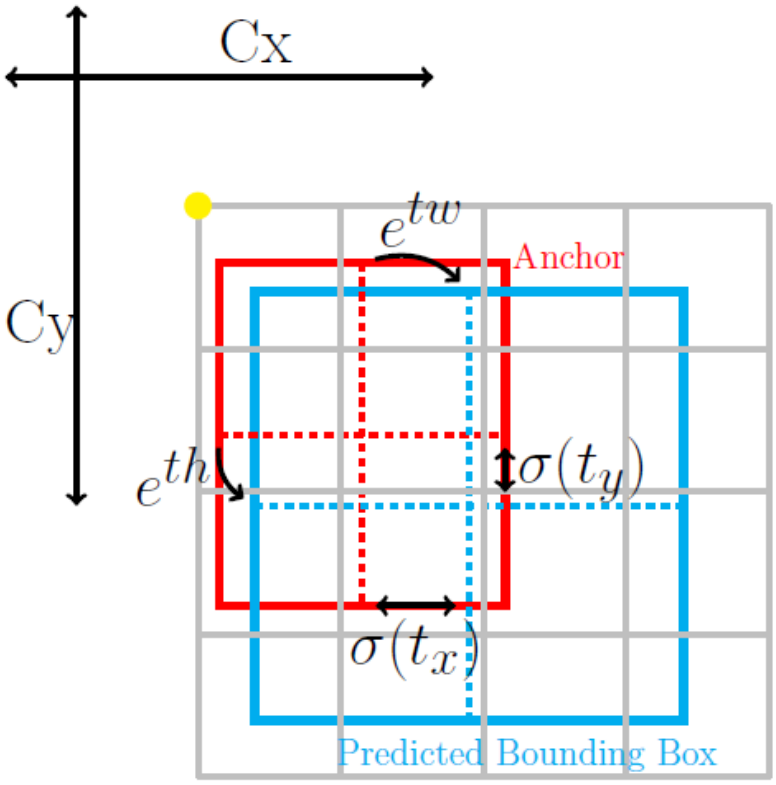


Figure 4.22: Calculating the predicted bounding box

Equations to calculate the predicted bounding boxes:

$$b_x = \sigma(t_x) + c_x \quad (4.9)$$

$$b_y = \sigma(t_y) + c_y \quad (4.10)$$

$$b_w = p_w * e^{tw} \quad (4.11)$$

$$b_h = p_h * e^{th} \quad (4.12)$$

where:

b_x, b_y, b_w, b_h = centre, width and height of predicted BB

t_x, t_y, t_w, t_h = outputs of NN

c_x, c_y = cell's top left corner of the anchor box

p_w, p_h = anchor boxes width and height

YOLOv3 does not predict absolute values of widths and heights, but predicts offsets to anchors, because it helps to eliminate unstable gradients during training. c_x,c_y,p_w,p_h are normalised to the real widths and heights.

$$c_x = c_x \,/\, \text{width of image} \quad (4.13)$$

$$c_y = c_y \,/\, \text{height of image} \quad (4.14)$$

$$p_w = p_w \,/\, \text{width of image} \quad (4.15)$$

$$p_h = p_h \,/\, \text{height of image} \quad (4.16)$$

Centre coordinates t_x and t_y are passed through sigmoid functions to obtain values between zero and one.

To obtain absolute values after predicting, the b_x, b_y, b_w and b_h values must be multiplied by the real and whole image width and height.

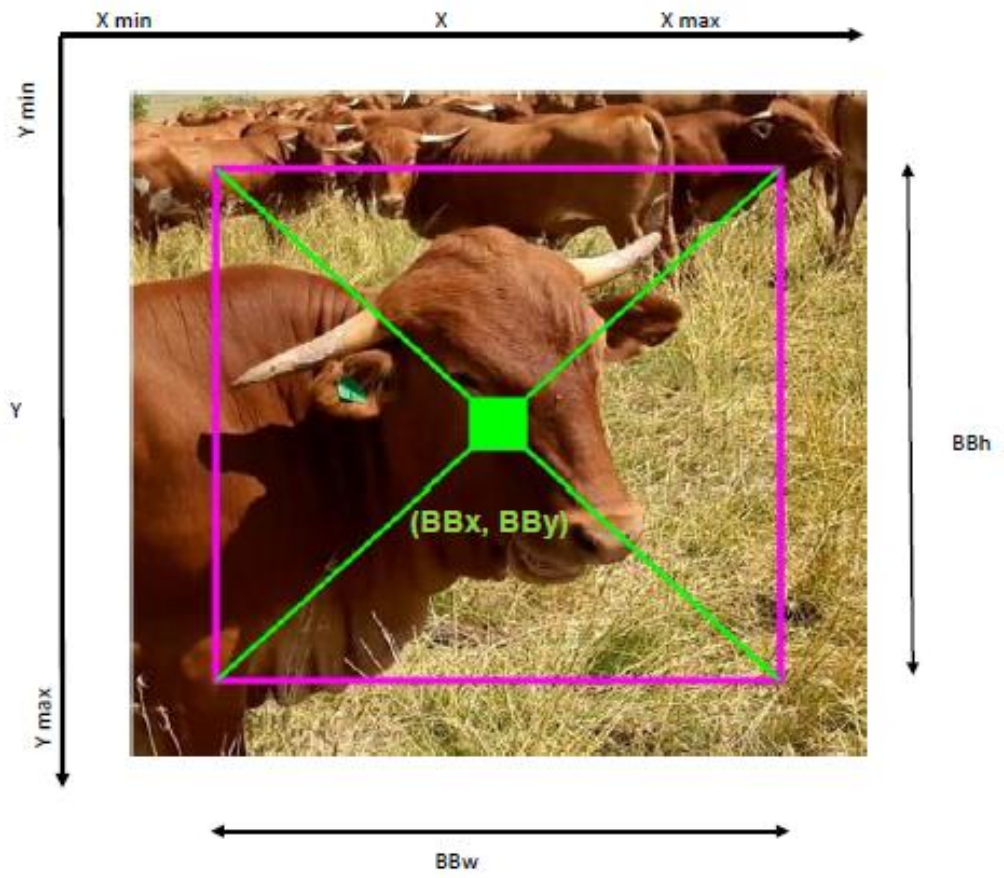


Figure 4.23: Bounding box

$$BBx = bx * \text{width of image} \quad (4.17)$$

$$BBy = by * \text{height of image} \quad (4.18)$$

$$BBw = bw * \text{width of image} \quad (4.19)$$

$$BBh = bh * \text{height of image} \quad (4.20)$$

4.3.5. Class confidence

The class confidence represents the likelihood that the detected object belongs to a specific category. The confidence score for each predicted bounding box is determined to evaluate the accuracy of both the object's classification and its exact position.

class confidence score = box confidence score * conditional class probability

(4.21)

4.3.6. Objectness Score

Objectness score interprets the probability of a centre cell.

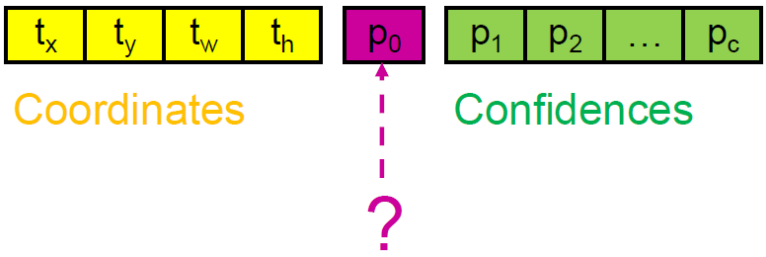


Figure 4.24: Objectness score

Training assigns a centre cell of a ground truth bounding box for predicting this object. Centre cells therefore have an objectness of nearly one, but corner cells have a low objectness score, ie, zero. The objectness score is thus the probability that this cell represents the centre of a bounding box containing a specific object.

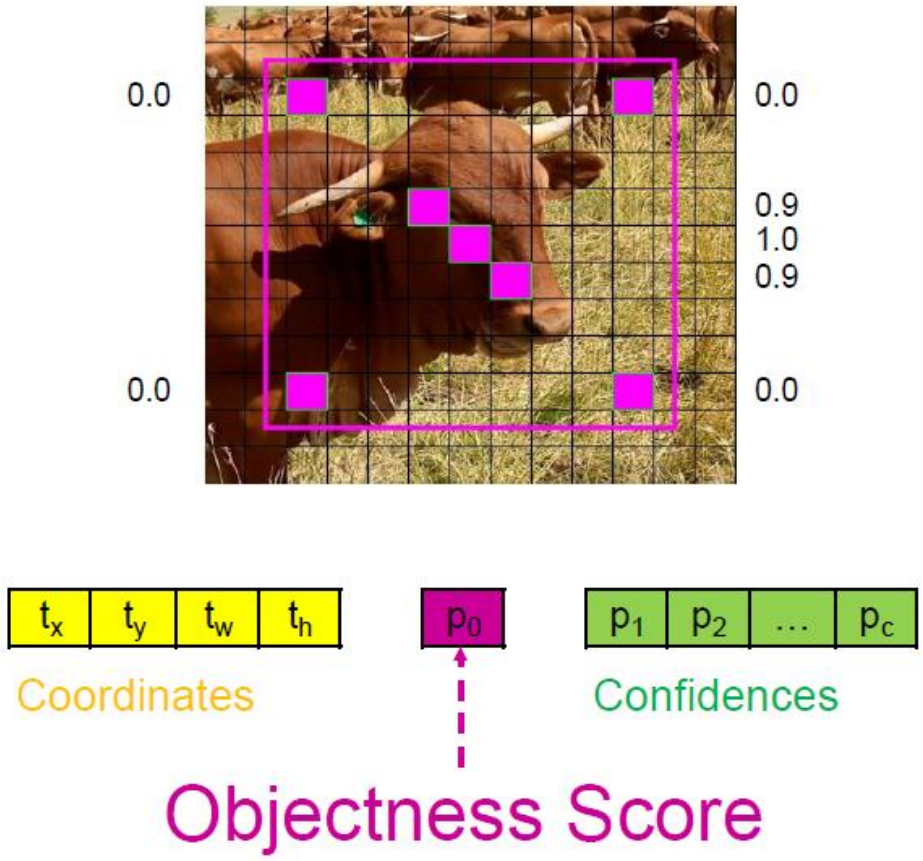


Figure 4.25: Bounding box attributes

Bounding Boxes attribute:

P0 = Objectness score

Centre cell of ground truth BB has p0 = 1

Corner cells of ground truth BB have p0 = 0

P0 is the probability that BB contains an object

Objectness score is the probability that the bounding box contains an object and can be calculated with the following equation:

$$\text{Box confidence score} = \text{Pobject} * \text{IoU} \tag{4.22}$$

Where:

P_{object} = the probability that the box contains an object

IoU = the Intersection of Union between the predicted box and ground truth

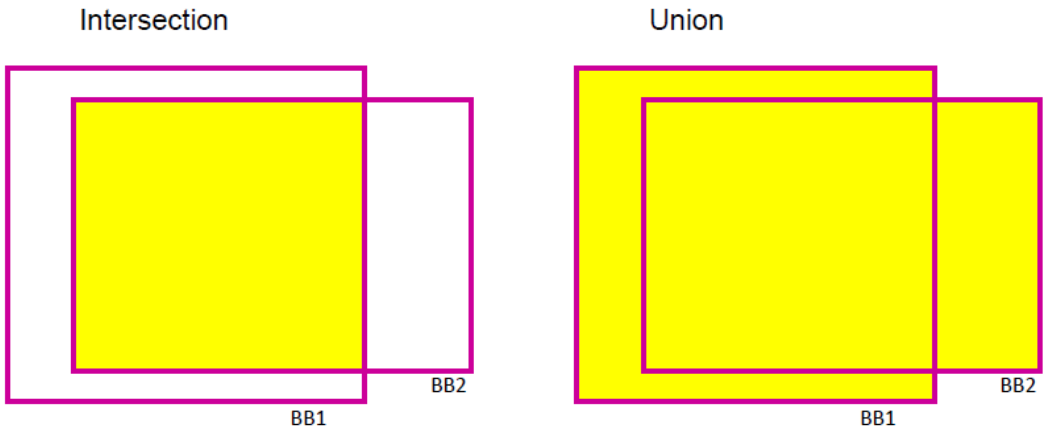


Figure 4.26: Intersection and Union

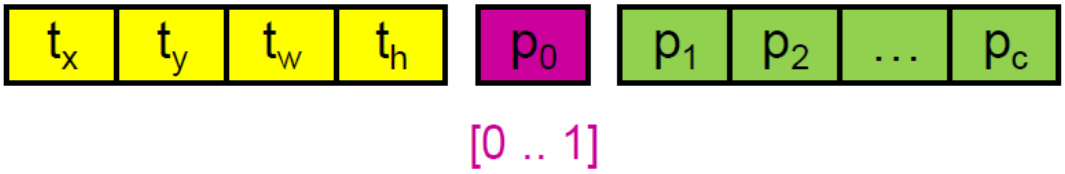


Figure 4.27: Objectness score

In YOLOv3, the final output is generated by passing the predictions through a sigmoid activation function. This function maps the output values to a range between 0 and 1, effectively transforming them into probabilities. The sigmoid function is applied to each class prediction, objectness score, and bounding box coordinate, allowing the model to produce confidence scores for object detection. By doing this, YOLOv3 ensures that each prediction represents the probability of an object's presence in a specific location and its classification into the correct category.

4.4 YOLOv3 Summary

While training kernels, YOLO applies a CNN to input images and detections happen at layers 82, 94, 106. The images are downsampled to three different sizes or scales (32, 16 and 8), one scale for each of the three layers. See Figure 4.28: Downsampling

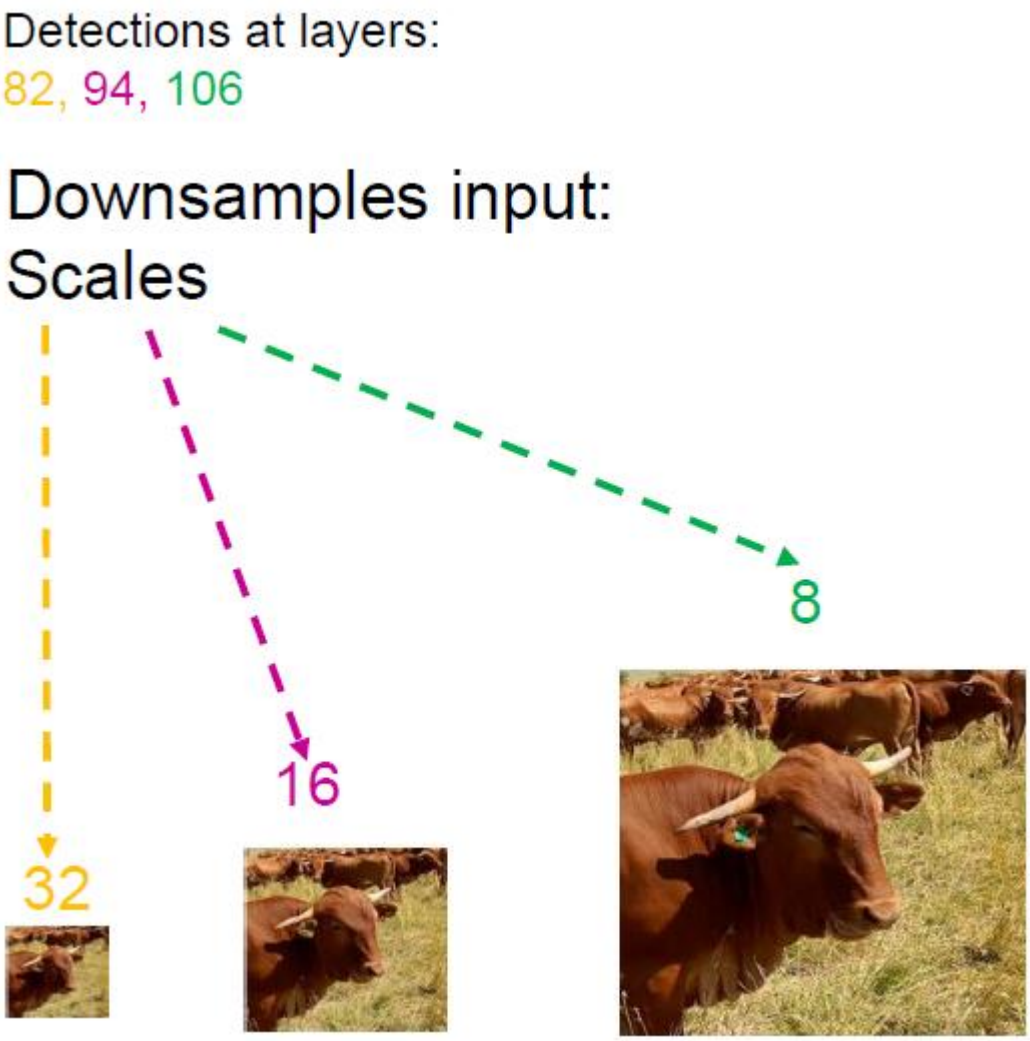


Figure 4.28: Downsampling

1 x 1 kernels are applied to the image (See Figure 4.29).

Figure 4.29: Kernels

YOLOv3 applies a CNN to an image and downsamples the image at three scales. 1 x 1 Kernels are applied to a grid of cells and one cell is responsible for detecting one object. The network is trained to assign one centre cell.

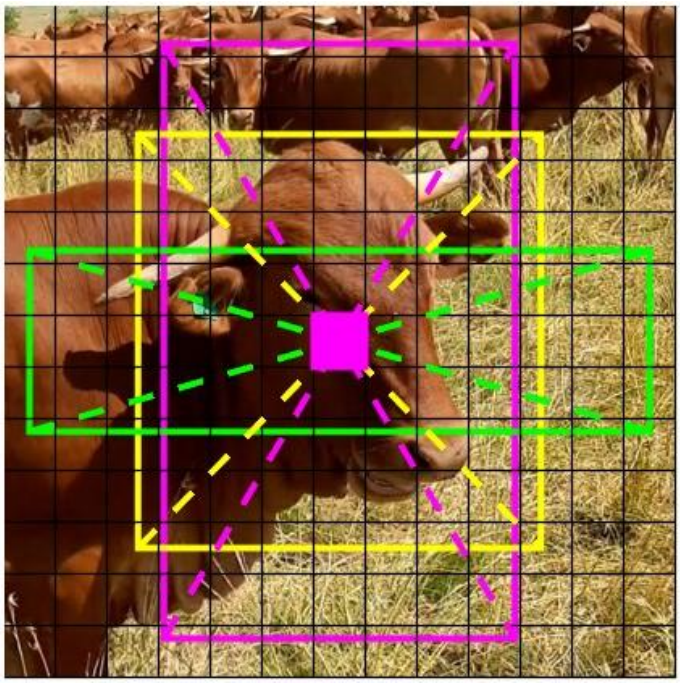


Figure 4.30: One centre cell

Nine pre-defined bounding boxes are used to calculate the spatial dimensions and coordinates of predicted bounding boxes. These are called anchor boxes. Three anchor boxes are used per scale. Nine anchors or priors predict BB. See Figure 4.31.

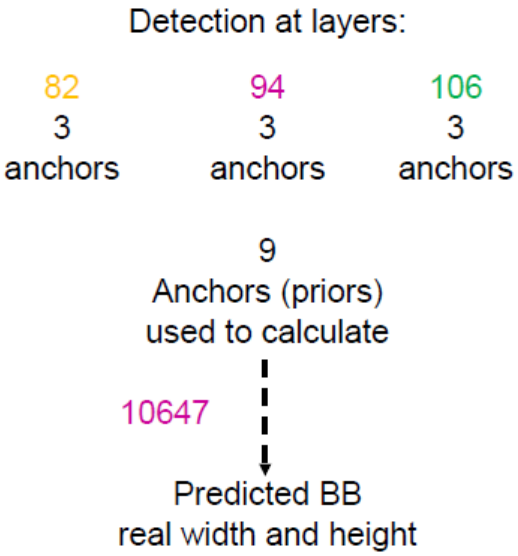


Figure 4.31: Calculation of predicted BB

YOLOv3 uses no Pooling layer, but uses Convolutional layers, which prevents loss of low-level features and can detect small objects. Figure 4.32 illustrates the results obtained from Convolution. Refer to Figure 3.22 to compare the results from Max Pooling with the results of Convolution shown in Figure 4.32.

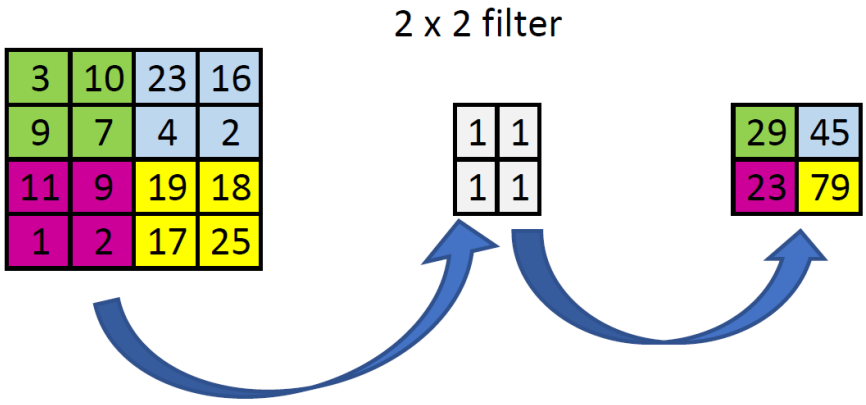


Figure 4.32: Convolution

4.5 Yolov3 vs Yolov4:

Table 4-1 illustrates the differences between YOLOv3 and YOLOv4. The main difference between YOLOv3 and YOLOv4 architectures is their backbone network: YOLOv3 uses the Darknet53 backbone, while YOLOv4 uses CSPDarknet53 for feature extraction, which improves efficiency and performance.

Neck: YOLOv4 incorporates additional modules such as SPP (Spatial Pyramid Pooling) and PANet (Path Aggregation Network) to enhance feature fusion and spatial information, which are not present in YOLOv3.

Head: The dense prediction block remains similar in both YOLOv4 and YOLOv3.

Table 4-1: YOLOv3 vs YOLOv4

	YOLOv3	YOLOv4
Backbone (Feature Extraction)	Darknet53	CSPDarknet53
Method of parameter aggregation from different backbone levels for different detector levels	FPN	PANet
Neck	-	Additional module - SPP, PANet
Head	Dense Prediction Block	Dense Prediction Block

Overall, YOLOv4 builds on the foundation of YOLOv3 with significant improvements in feature extraction and processing, leading to better performance and accuracy.

4.6 YOLOv4

Image Recognition using YOLOv4 involves several key steps, from input image preprocessing to the final prediction of objects and their bounding boxes. Below is an explanation of the process:

4.6.1. Steps to the final prediction of objects

Figure 4.33 shows the final prediction of objects.

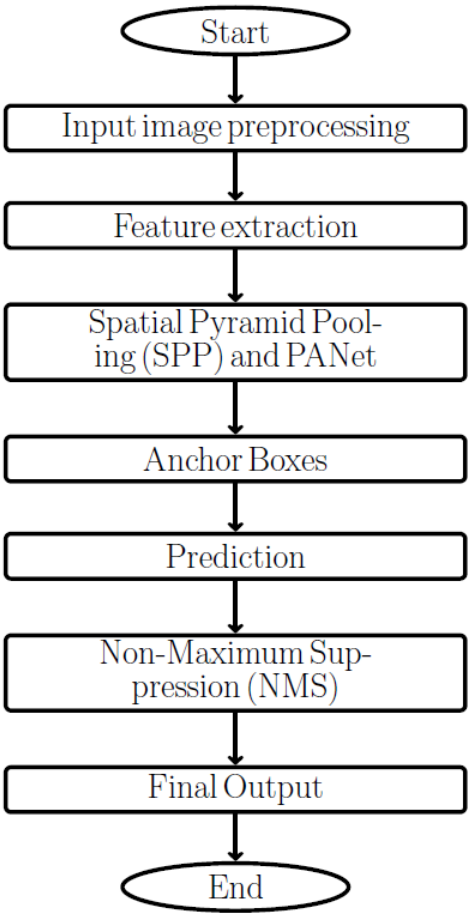


Figure 4.33: Steps to the final prediction of objects

4.6.1.1. Input Image Preprocessing

The first step involves feeding an image into the YOLOv4 model. The image is typically resized to a standard size (e.g., 416 x 416 pixels) to fit the input layer of the model. YOLOv4 uses a single neural network to process the entire image, unlike region-based methods that perform region proposals beforehand.

4.6.1.2. Feature Extraction

The input image is passed through a convolutional neural network (CNN) backbone (such as CSPDarknet53 in YOLOv4). This CNN is responsible for extracting features from the image at various levels, capturing both low-level (edges, textures) and high-level (object shapes) information. The network

produces feature maps that contain crucial information about the objects in the image.

4.6.1.3. Spatial Pyramid Pooling (SPP) and PANet

YOLOv4 incorporates advanced techniques like Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PANet). SPP allows the model to extract features at different scales, which is essential for detecting objects of varying sizes. PANet enhances the ability of the network to combine low-level and high-level features, improving accuracy in object detection.

4.6.1.4. Anchor Boxes

YOLOv4 uses predefined anchor boxes to predict the location and size of objects. These are fixed bounding boxes of various sizes and aspect ratios that help the model predict objects at different scales. The model adjusts these anchor boxes to fit the detected objects in the image during training.

4.6.1.5. Prediction

YOLOv4 predicts three key pieces of information for each anchor box: (1)*Class probabilities:* The likelihood of the object belonging to a specific class (e.g., animal 1, animal 2 or animal 3); (2) *Bounding box coordinates:* The precise location of the object in the image, represented as x, y coordinates, and the height and width of the box; (3) *Confidence score:* A score that indicates how confident the model is that the predicted bounding box contains a valid object.

The model makes predictions across three different scales to detect objects of different sizes (small, medium, large).

4.6.1.6. Non-Maximum Suppression (NMS)

Once the model predicts multiple bounding boxes, it may generate several overlapping boxes for the same object. Non-Maximum Suppression (NMS) is used to eliminate redundant boxes by keeping the one with the highest confidence score and discarding others. This step ensures that only one bounding box is retained per detected object.

4.6.1.7. Final Output

After NMS, the final output consists of a set of bounding boxes around detected objects, along with the class labels and confidence scores for each object. These predictions are then mapped back to the original image scale, providing the locations and identities of objects in the image.

The above process allows YOLOv4 to detect objects in real-time with high accuracy and efficiency.

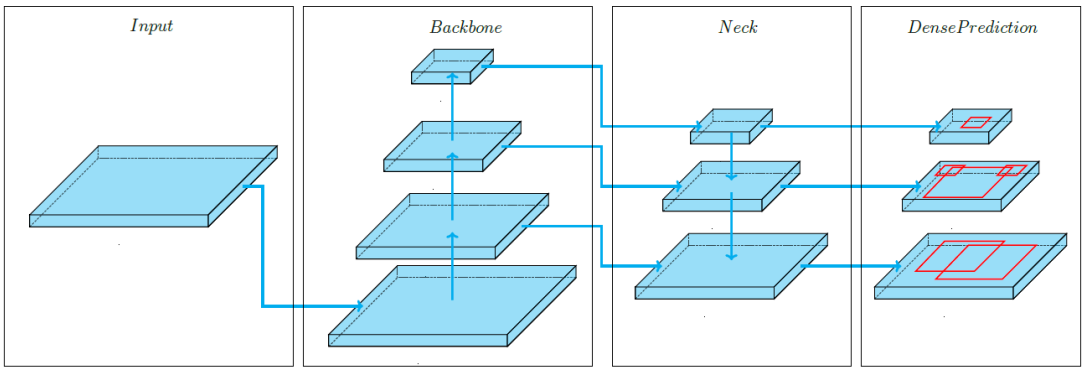


Figure 4.34: One-stage YOLO detector [82]

4.6.2. YOLOv4 block diagram

The block diagrams shown in Figure 4.35, Figure 4.36, Figure 4.37, Figure 4.38 and Figure 4.39 show the building blocks for YOLOv4.

Figure 4.35: YOLOv4 Convolution + Batch Normalisation + Mish

Figure 4.36: YOLOv4 Convolution + Batch Normalisation + Leaky ReLU

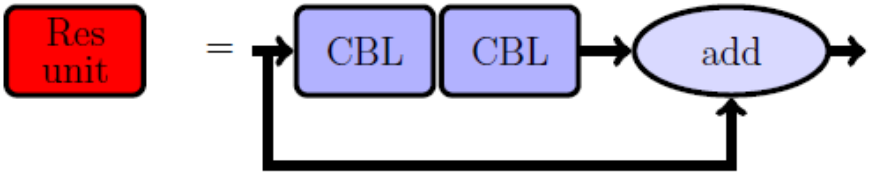


Figure 4.37: YOLOv4 Residual unit structure

Darknet-53 introduces a Residual Block to address challenges in training deep neural networks. As the network depth increases, accuracy can reach a maximum, producing higher training errors. The Residual Block incorporates a skip connection, which distinguishes it from a standard convolution block. This skip connection passes the input directly to deeper layers, helping to mitigate training difficulties [83].

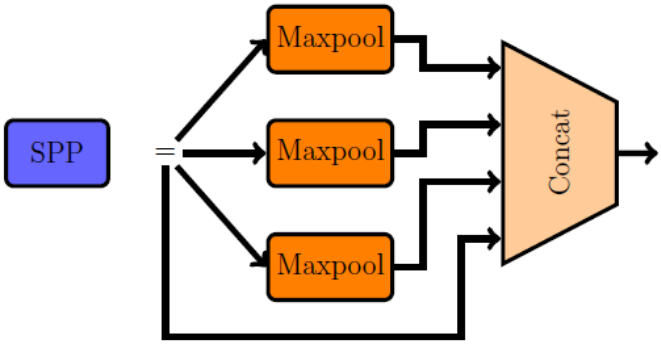


Figure 4.38: YOLOv4 Spatial Pyramid Pooling

Spatial Pyramid Pooling (SPP) is a pooling layer that eliminates the network's fixed-size input constraint. By adding an SPP layer on top of the last convolutional layer, it pools features to generate fixed-length outputs, which are then fed into fully connected layers or other classifiers. This approach aggregates information at a deeper network stage, between the convolutional and fully connected layers, thus avoiding the need for cropping or warping images at the input stage [84].

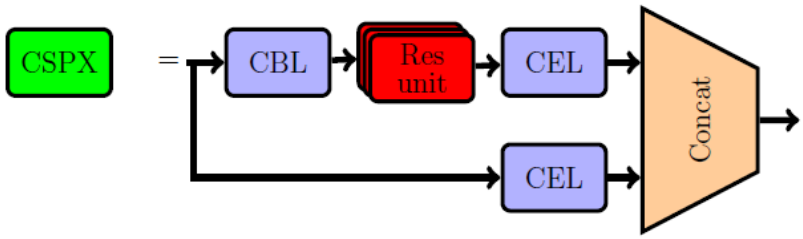


Figure 4.39: YOLOv4 Cross-Stage Partial structure

The main improvement in YOLOv4 compared to YOLOv3 is the adoption of a new CNN architecture known as CSPNet (Cross Stage Partial Network), which is a variant of the ResNet architecture tailored for object detection. Despite having a relatively shallow structure with just 54 convolutional layers, CSPNet achieves state-of-the-art performance on various object detection benchmarks [85].

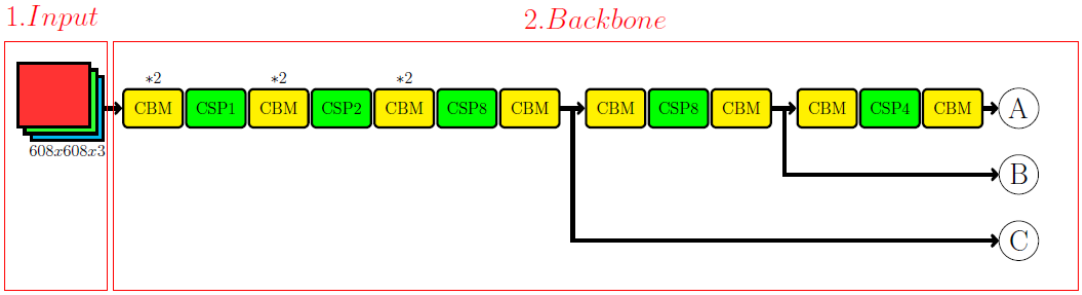


Figure 4.40: YOLOv4 Backbone

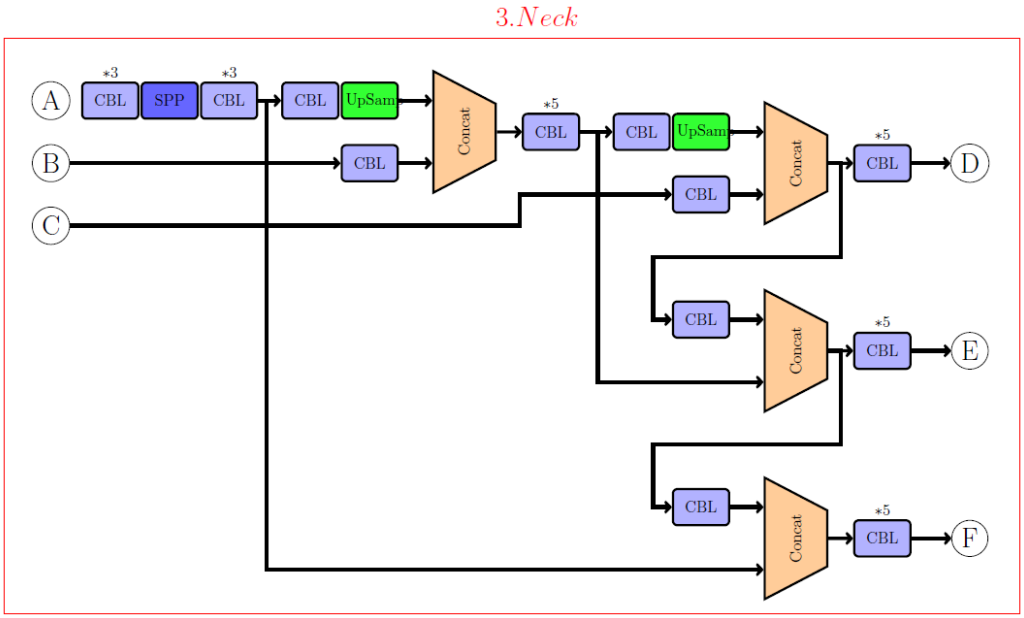


Figure 4.41: YOLOv4 Neck

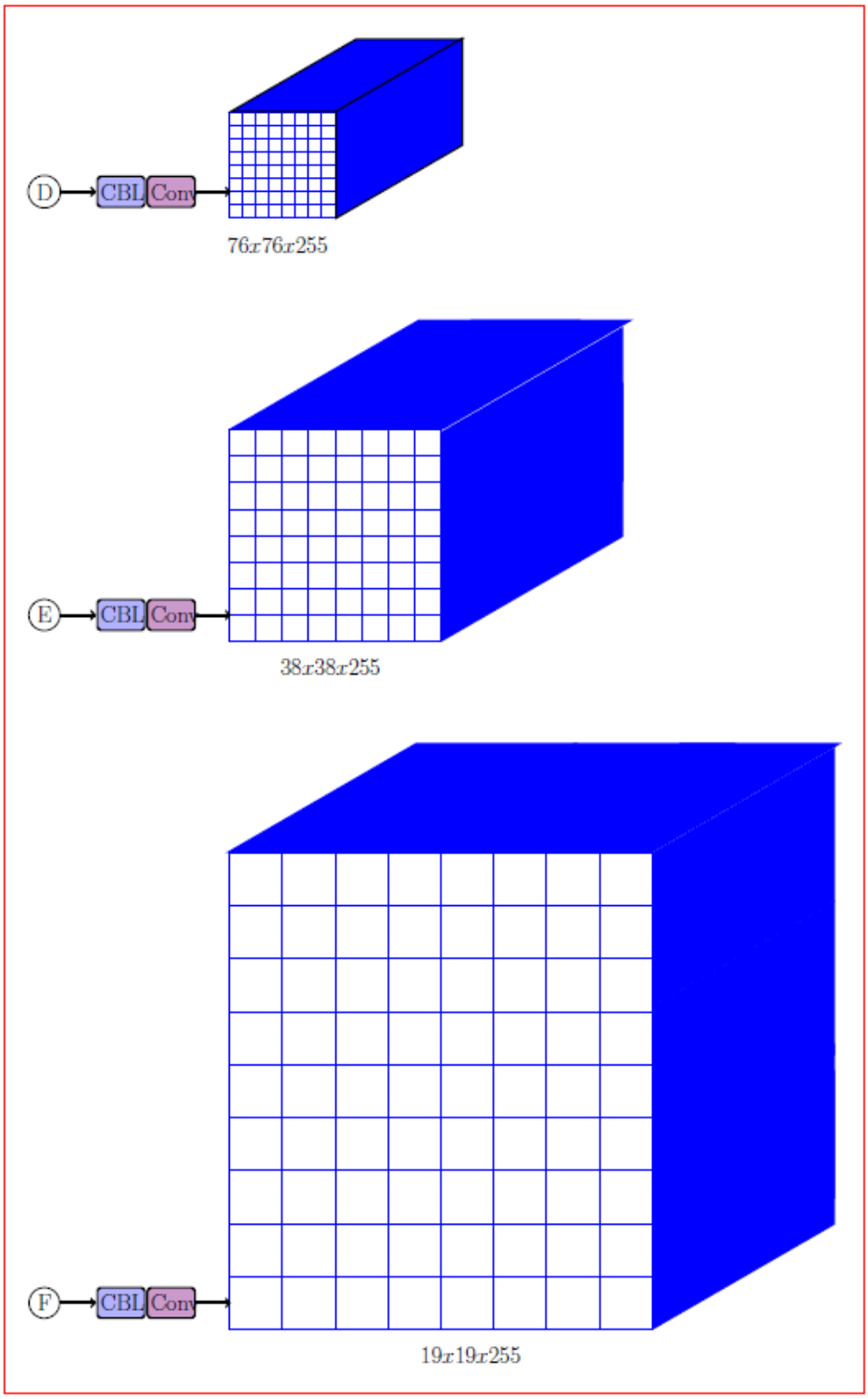


Figure 4.42: YOLOv4 Prediction

Figure 4.42: YOLOv4 Prediction depicts the final prediction at the three sizes.

CHAPTER 5: INITIAL WORK

5.1 Animal facial recognition

Animal facial recognition systems and research lag far behind human facial recognition [86]. Animal facial recognition can be improved by adding side profiles of animals. Cattle interactions with cameras and equipment used in agricultural and environmental monitoring can lead to substantial damage, resulting in expensive repairs and disruptions in data collection.

5.1.1. Primary Causes of Damage

- *Curiosity and Behaviour:* Cattle are naturally inquisitive animals and may explore unfamiliar objects in their environment. Their investigation can involve nudging, licking, or even chewing on equipment, leading to potential damage [87].
- *Physical Contact:* As cattle move about, they may inadvertently bump into, step on, or rub against equipment. This physical contact can result in the displacement or destruction of sensitive devices [87], [88].
- *Environmental Exposure:* Equipment left exposed to harsh weather conditions can become more vulnerable to damage. When combined with cattle interactions, the risk of malfunction or breakage increases [88], [89].

5.1.2. Types of Damages

- *Mechanical Damage:* Cameras and other devices can suffer from broken lenses, cracked casings, or dislodged components due to physical impacts.
- *Electrical Issues:* Interaction with equipment can lead to broken wires, disconnected power supplies, or damaged electronic circuits.
- *Data Integrity:* Damage to equipment can result in the loss of valuable data, affecting research accuracy and continuity [86].

Capturing, analysing pictures and creating a database of animals

The identification of different types and individual animals necessitated the building of a database of images of individual animals. A library of known animals

was created, and a deep neural network was designed to perform the animal recognition. Research included the capturing and collection of high-quality or low-quality photo or video images that could be used to identify animal “faces” [18].

Various data sources were collected and integrated to balance the quantity of captured data with the reliable identification of animals. [18]. Shown below is a graphical overview of the project.

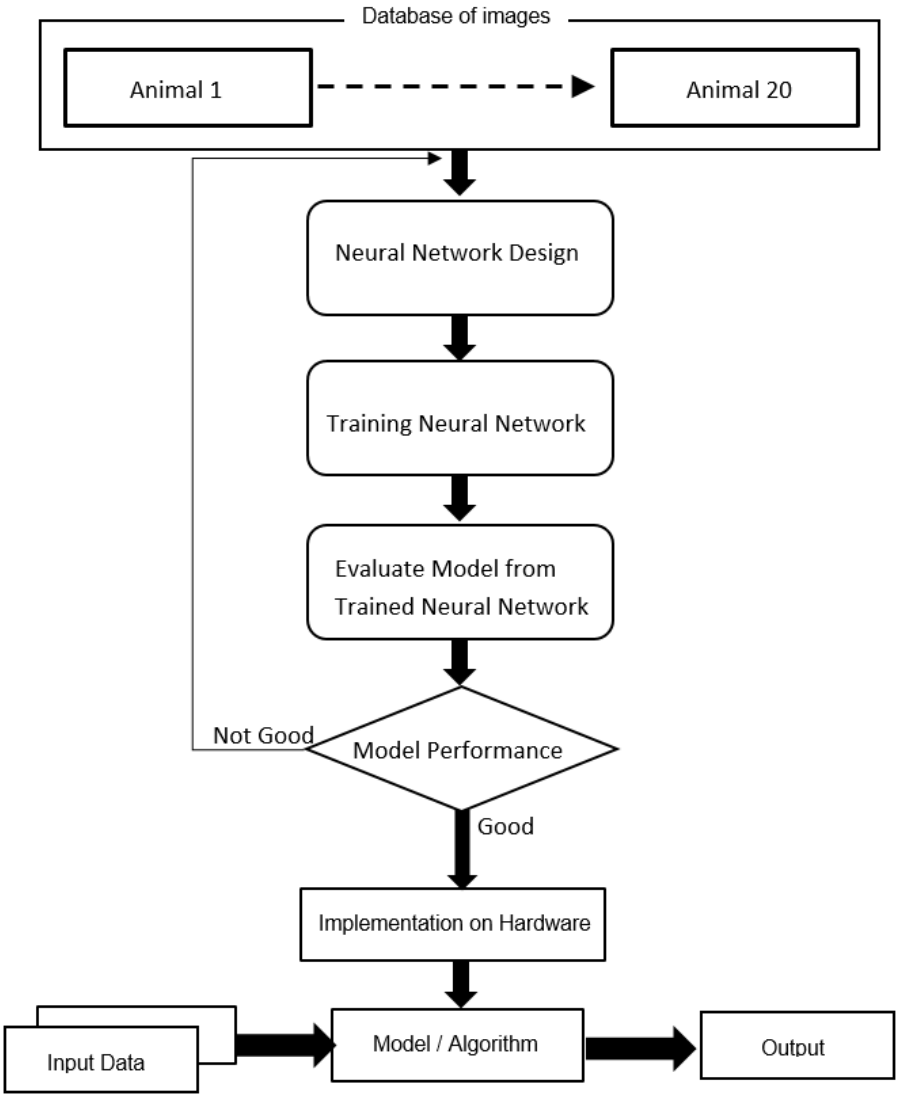


Figure 5.1: Research process

5.3 Extraction of images

A MATLAB® M file was used to capture the frames in the reduced video. Photos of each frame were extracted from the captured videos. Due to the probability that consecutive frames would look alike, every twentieth frame was captured.

5.4 Video data collection and processing of data

Figure 5.2 shows the steps followed in the collection and processing of the data for the initial tests.

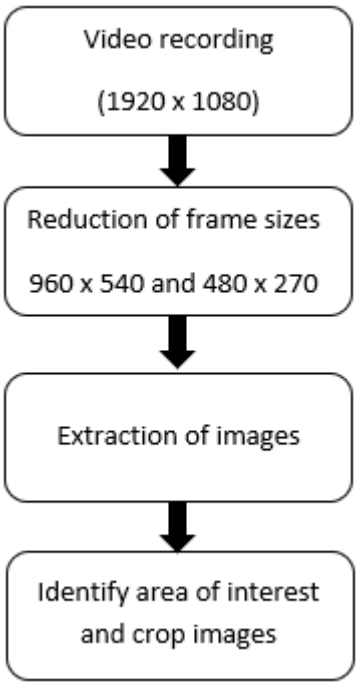


Figure 5.2: Data collection and processing steps

5.4.1. Video recording

When walking in the veld with a camera to capture images of individual animals, there is a constant milling and moving of cattle in a herd, which makes them difficult to photograph. Consequently, video captures were chosen over individual photographs for data collectional, as depicted in Figure 5.2. The first video footage was recorded in 1920 x 1080 format.

5.4.2. Reduction of frame sizes

This video footage was converted (resized) to 960 x 540 and 480 x 270 video formats. Figure 5.3 and Figure 5.4 show typical images that were extracted from

the video frames and illustrate the resolution size reduction of the captured images.

Figure 5.3: 960 x 540 Image (reduced size 50% to fit on page)

Figure 5.4: 480 x 270 Image (reduced size 50% to fit on page)

MATLAB® was used to reduce the size of the frames in the captured video.

5.4.3. Identify the area of interest and crop images

After the captured frames were collected, smaller portions of the frames were extracted. These frames were then identified and labelled by hand for each individual animal. Another MATLAB® M file was used to capture and crop areas of interest from the captured pictures from the reduced video.

5.4.4. MATLAB® Speeded-Up Robust features

One of the first tests of this study was to use MATLAB® to identify Speeded-Up Robust features (SURF) on cattle.

MATLAB® code was used to extract and display the Corner Features from an Image, depicted in Figure 5.5.

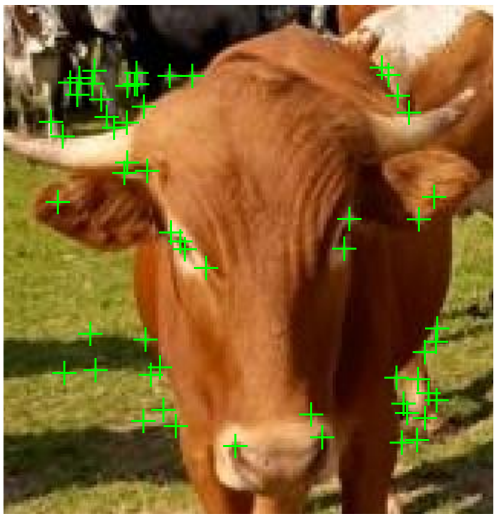


Figure 5.5: Plot valid corner points

MATLAB® was then used to extract and display the SURF Features from an Image. (See Appendix 1.)

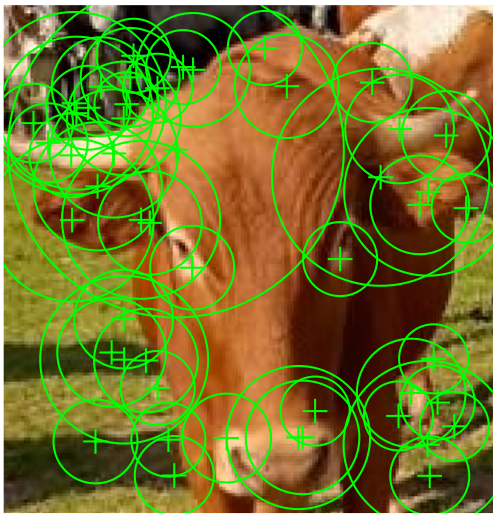


Figure 5.6: Reference image

The interest areas shown in Figure 5.7 were identified by the Matlab® program.

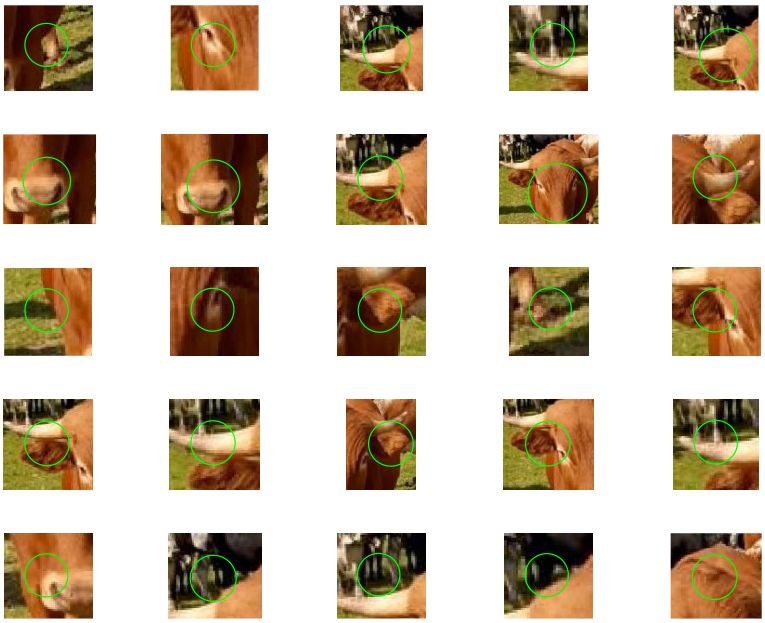


Figure 5.7: 25 interest areas identified with Matlab®

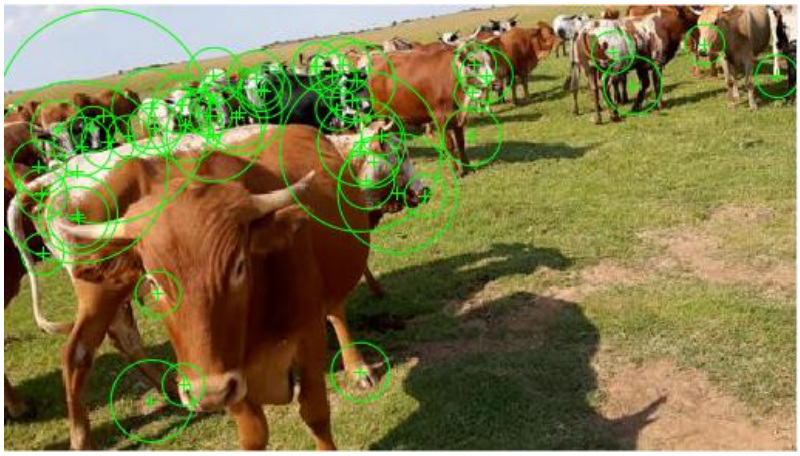


Figure 5.8: Interest areas identified with Matlab®

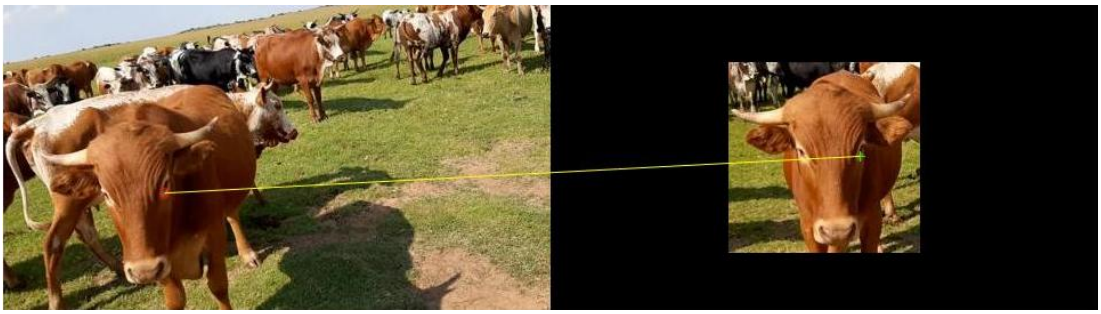


Figure 5.9: Feature Matching in Images

The SURF algorithm could identify points of interest in images, but Figure 5.9 demonstrated that it was not suitable for individual cattle identification as it only matched one feature, despite the images being nearly identical. To address this, the decision was made to explore the You Only Look Once (YOLO) deep learning model for image recognition. YOLO is popular for real-time object detection in images or videos, known for its speed and efficiency, making it ideal for applications like autonomous driving, surveillance, and robotics.

YOLO provides bounding boxes, class labels, and confidence scores as its output, which offer information about detected objects and their locations in the image. YOLO's key advantage is its single-pass detection approach on the entire image, allowing real-time performance without sacrificing accuracy [18].

CHAPTER 6: METHODOLOGY FOR YOLO-BASED OBJECT DETECTION

6.1 Introduction to You Only Look Once

YOLOv3 employs a convolutional neural network (CNN) for object detection, organised into three main components: the backbone, the neck, and the head. The backbone consists of convolutional layers that extract features from the input image. The neck integrates features from various scales to enhance detection accuracy, and the head comprises fully connected layers that determine the locations and classes of objects. Additionally, YOLOv3 utilises anchor boxes, pre-defined bounding boxes with various sizes and aspect ratios, to predict the objects' positions and dimensions within the image.

In YOLOv3, each grid cell outputs five parameters for bounding box detection: x, y, w, h and confidence. The box centre coordinates are x and y, whilst w and h are the width and height. Confidence indicates object absence or presence. Additionally, class probabilities are assigned to each grid cell. To handle potential multiple bounding boxes for the same object, the Non-Max Suppression method is used.

The model processes input images in batches of size n, with each image having dimensions 416 × 416 and three colour channels (red, green and blue). YOLOv3's architecture avoids pooling layers and uses a stride of 2 in its convolutional layers for downsampling. Multiple filters in each convolutional layer create various feature maps.

Predictions are made at three different network stages: the 82nd, 94th, and 106th layers. The 82nd layer, with a stride of 32, produces 13x13 grids for detecting large objects (416 / 32 = 13). The 94th layer, with a stride of 16, results in 26x26 grids for medium-sized objects (416 / 16 = 26). The 106th layer, using a stride of 8, generates 52x52 grids for detecting smaller objects (416 / 8 = 52).

The YOLOv3 algorithm utilises the Darknet-53 feature extraction model, which consists of 53 layers, leading to a total of 106 layers in its fully convolutional architecture [90]. Object detection occurs at three specific layers: the 82nd, 94th, and 106th. Each of these layers incorporates batch normalisation and the Leaky ReLU activation function. YOLOv3 works by dividing an image into grid cells, with each cell responsible for predicting the presence of objects within its area.
Creating a YOLOv3 Keras model and saving it to a file using You Only Look Once (YOLOv3).

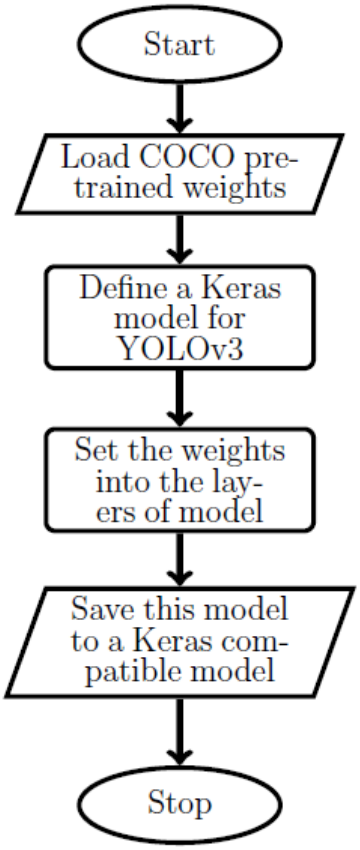


Figure 6.1: Create a YOLOv3 Keras model and save it to a file

Figure 6.1 shows the flow chart for a Python program to save a Keras model to be used to identify objects or animals, in this study [91].

Python Saving model

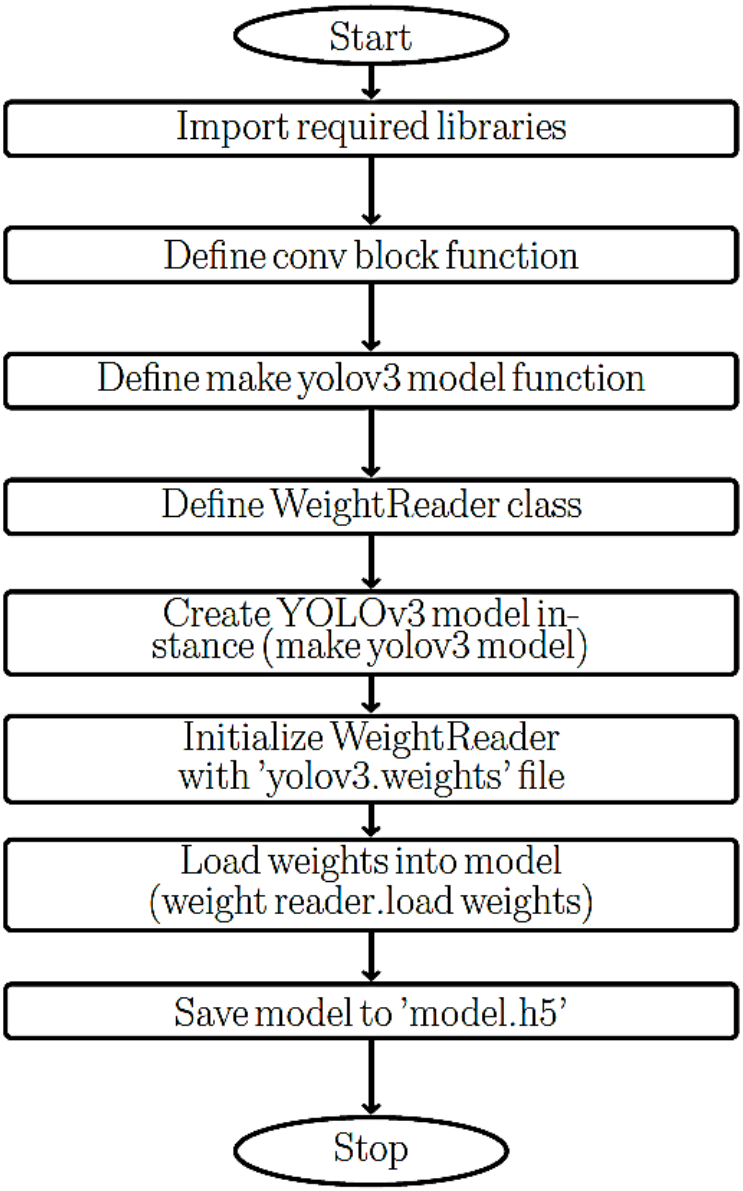


Figure 6.2: Flowchart representing the steps of the provided YOLOv3 model creation and saving Python program

This flowchart and its explanation detail how the YOLOv3 model is constructed, how weights are loaded, and how the final model is saved using Keras. Each function and class are broken down into their specific tasks, and the flowchart visually represents the sequential operations of the program.

6.3 Identification of cattle from images using You Only Look Once (YOLOv3) and Python for object detection

Figure 6.3 shows the flowchart of the complete Python program to use the Keras model to identify cattle [91].

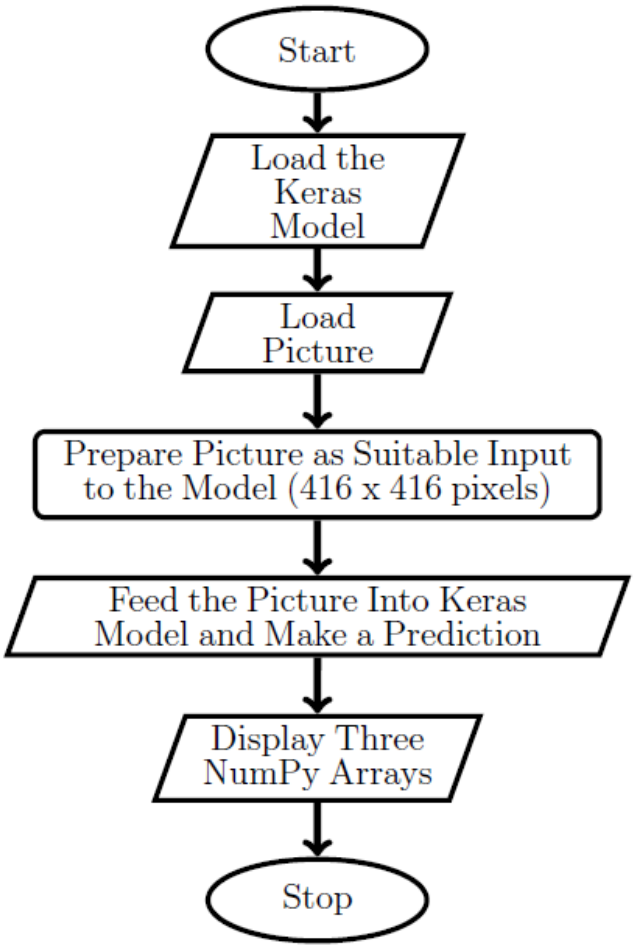


Figure 6.3: Flowchart of the implementation of the Keras model to identify cattle

Figure 6.4 depicts a flowchart representing the Python program for YOLOv3 object detection:

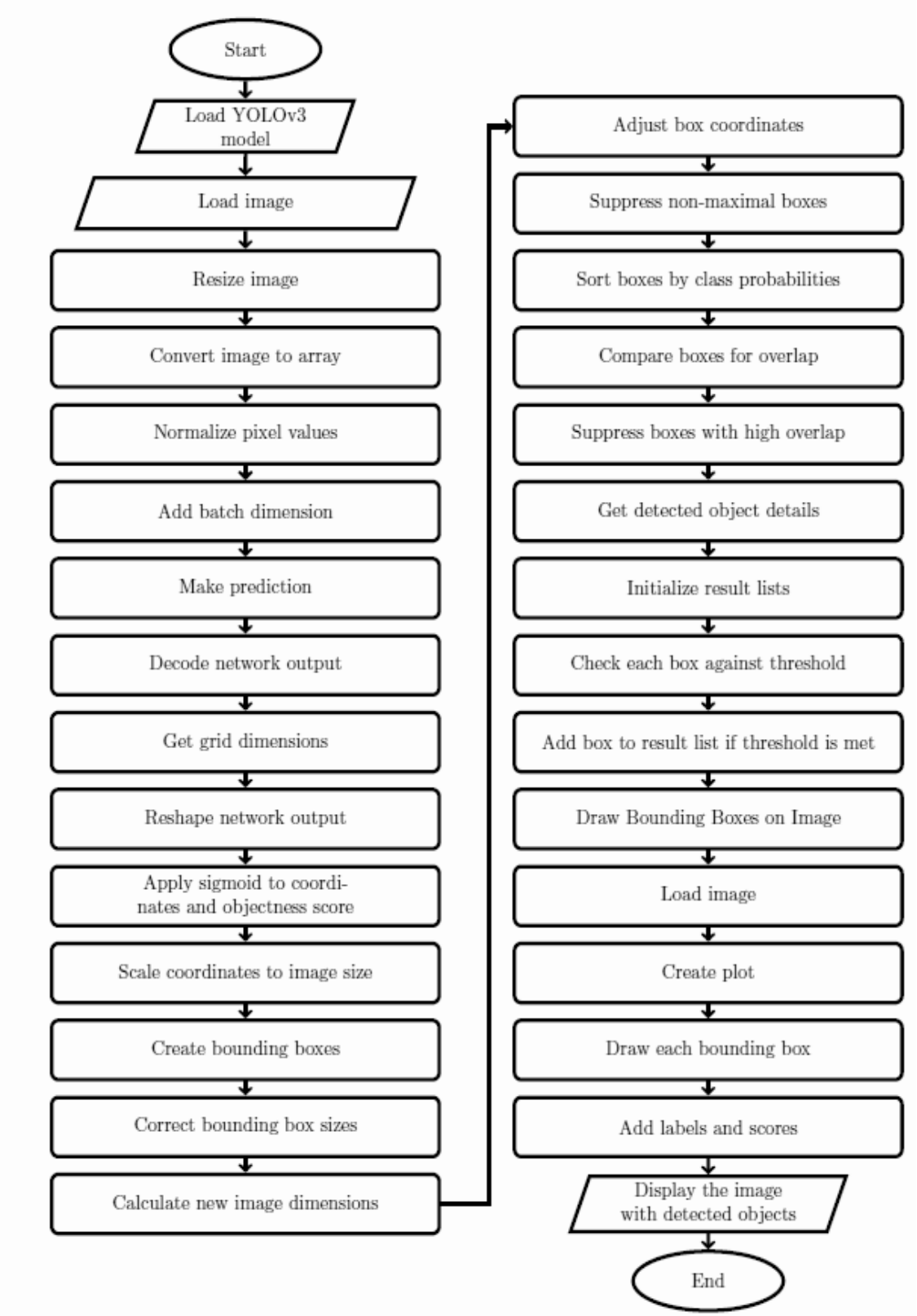


Figure 6.4: Flowchart of the implementation of YOLOv3 in Python for object detection

Figure 6.5 depicts a flowchart that outlines the process of loading a YOLOv3 model, performing object detection on an image, and visualising the results:

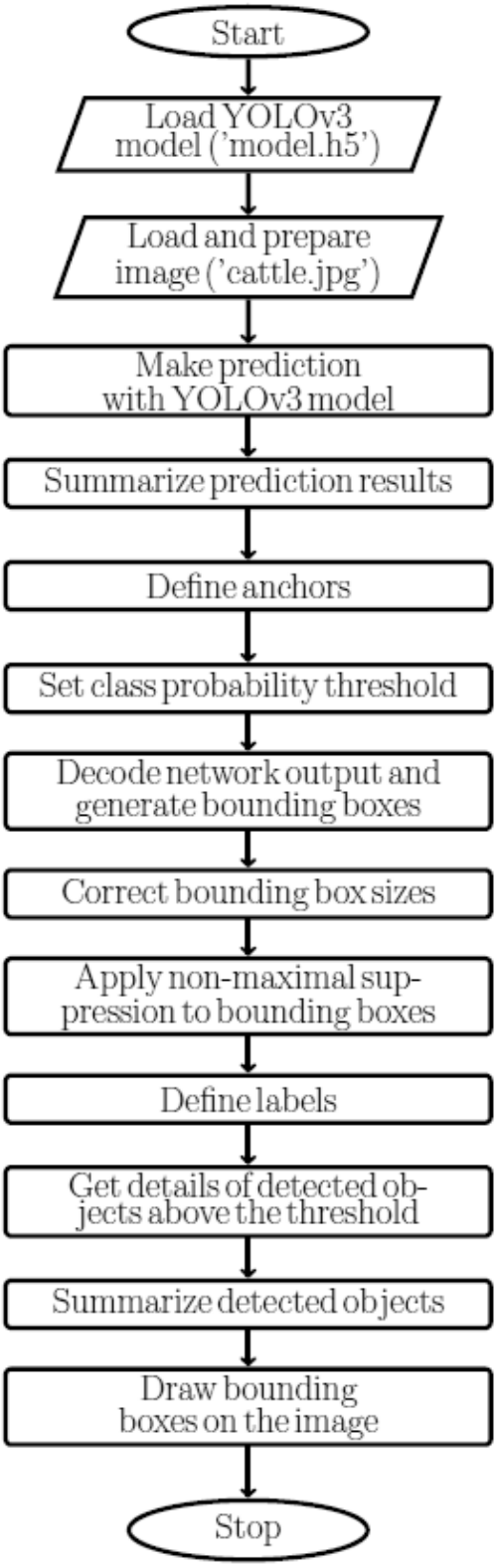


Figure 6.5: Using YOLOv3 to do object detection

Figure 6.6 depicts the output from the Python program written from the flow chart shown in Figure 6.3.

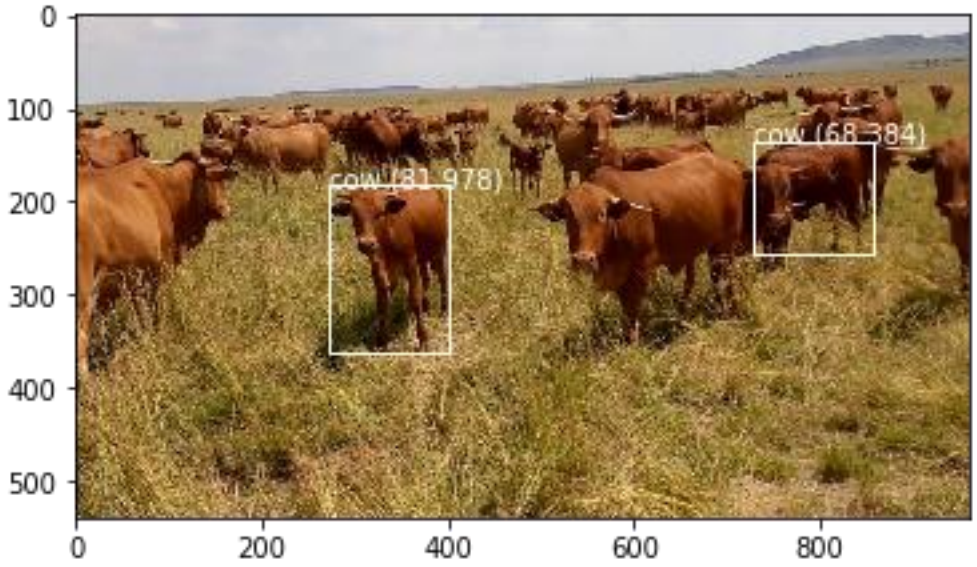


Figure 6.6: Classified cow image

Figure 6.7 outlines a simplified flowchart that covers the main components and interactions of the program in the listing. This flowchart will focus on the key steps involved in setting up the camera, defining the dataset, training the model, and performing live predictions.

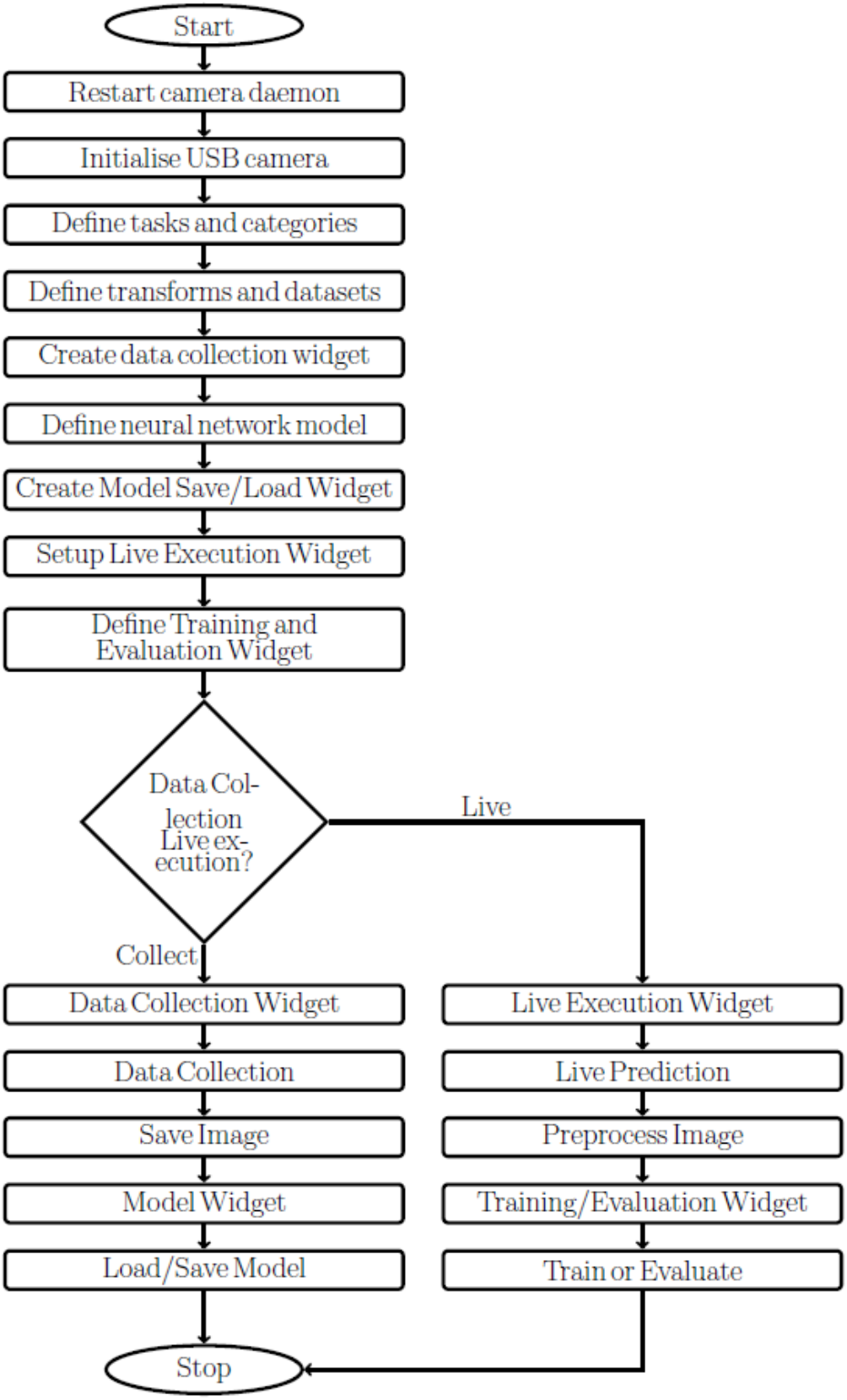


Figure 6.7: Program for camera input

6.4 YOLOv3 training and detection

Initial tests for this study were done with YOLOv3.The Network's input consisted of a batch of images with the following shape: (n, 416, 416, 3) Images, width, height, RGB channels. The width and height could be any number if it was divisible by 32. Numbers like 608 x 608, 832 x 832 and 1024 x 1024 were used. Input images, however, can be any size and were resized to the network size. Aspect ratios could be changed.

A YOLOv3 system was trained with cattle head images and implemented on MATLAB®. Figure 6.8 depicts the flow chart for the process of YOLOv3 implemented in MATLAB® [18].

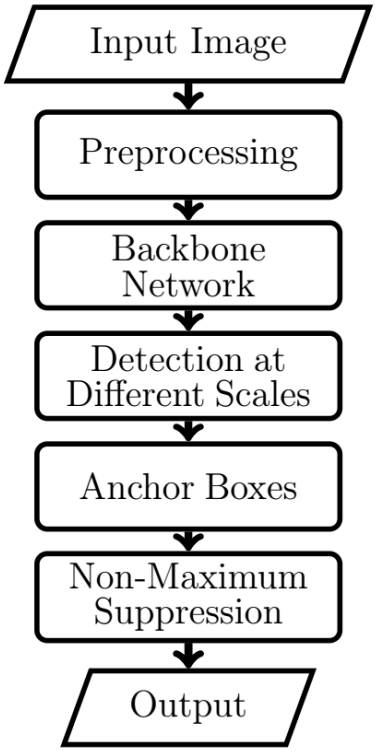


Figure 6.8: YOLOv3 implemented in MATLAB®

The flowchart can be explained in the following way:

6.4.1. Input Image

The algorithm takes an input image as its input.

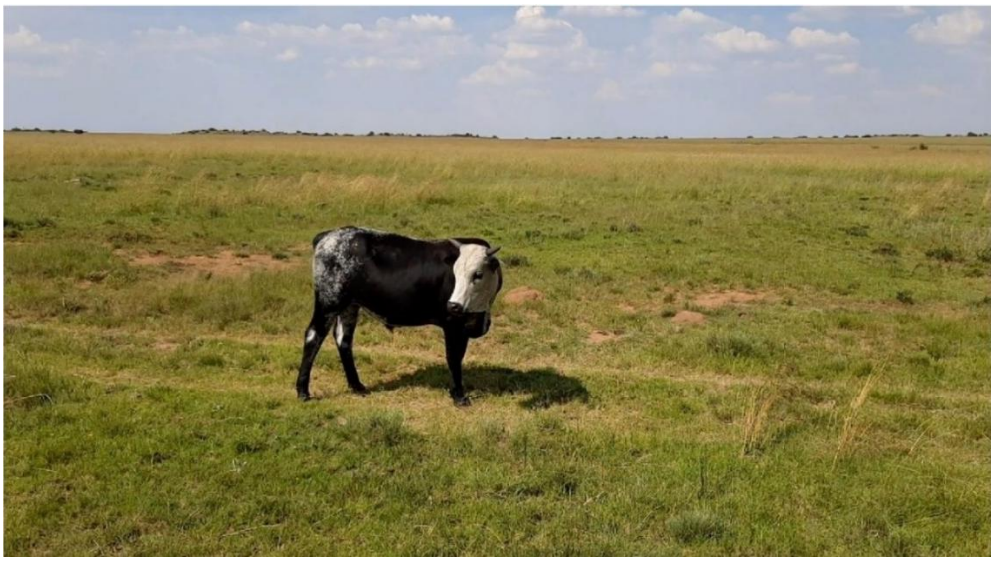


Figure 6.9: Input image (1080 1920 x 3) (Reduced size to fit on page)

6.4.2. Preprocessing:

The input image was resized to a fixed size and normalised to have pixel values in the range [0, 1]. This pre-processed image was then fed into the neural network.

Figure 6.10: Resized image (416 x 416 x 3)

```
val(:,:,1) =

  Columns 1 through 16

    0.7293    0.7307    0.7304    0.7233    0.7215    0.7187
    0.7293    0.7308    0.7305    0.7212    0.7175    0.7178
    0.7293    0.7307    0.7307    0.7192    0.7134    0.7169
    0.7294    0.7296    0.7300    0.7226    0.7167    0.7149
    0.7294    0.7294    0.7299    0.7232    0.7172    0.7146
    0.7294    0.7294    0.7299    0.7232    0.7172    0.7146
    0.7231    0.7221    0.7198    0.7186    0.7176    0.7167
    0.7215    0.7203    0.7173    0.7175    0.7178    0.7172
    0.7218    0.7206    0.7176    0.7176    0.7179    0.7172
    0.7187    0.7184    0.7178    0.7178    0.7138    0.7139
    0.7176    0.7169    0.7152    0.7166    0.7125    0.7126
    0.7179    0.7155    0.7093    0.7143    0.7132    0.7127
    0.7176    0.7184    0.7210    0.7147    0.7073    0.7138
    0.7174    0.7194    0.7251    0.7142    0.7046    0.7118
    0.7178    0.7180    0.7190    0.7117    0.7052    0.7088
    0.7151    0.7158    0.7181    0.7116    0.7044    0.7048
    0.7151    0.7149    0.7144    0.7114    0.7066    0.7051
    0.7210    0.7201    0.7180    0.7150    0.7106    0.7088
```

Figure 6.11: Normalised to have pixel values in the range 0 to 1

Figure 6.11 shows an extract of the normalised values of the picture. All values were in the range of zero to one.

6.4.3. Backbone Network:

The pre-processed image was passed through a backbone neural network (typically a convolutional neural network like Darknet-53) to extract features at multiple scales. These features captured both low-level and high-level information from the image.

6.4.4. Detection at Different Scales

YOLOv3 performs detection at three different scales by applying detection heads to feature maps at three different levels of the backbone network. Each detection head predicts bounding boxes, objectness scores, and class probabilities for objects within a predefined set of anchor boxes.

6.4.5. Anchor Boxes

YOLOv3 utilises anchor boxes, which are predefined bounding box priors of different shapes and sizes. These anchor boxes were used to predict bounding box coordinates relative to the anchor box shapes and to estimate the likelihood of object presence (objectness score).

6.4.6. Non-Maximum Suppression (NMS)

After predictions were made at multiple scales, non-maximum suppression was applied independently to each scale to remove redundant bounding boxes. NMS selected the most confident bounding boxes while suppressing overlapping detections with lower confidence scores.

6.4.7. Output

The final output of YOLOv3 consisted of a list of bounding boxes, each associated with a class label and a confidence score. These bounding boxes represented the objects detected in the input image.

Figure 6.12, Figure 6.13 and Figure 6.14 illustrate the original input image processed with different grid sizes, showing how anchor boxes are distributed across each grid. Specifically, Figure 6.12 shows the 416 × 416 image with a 13 × 13 grid.

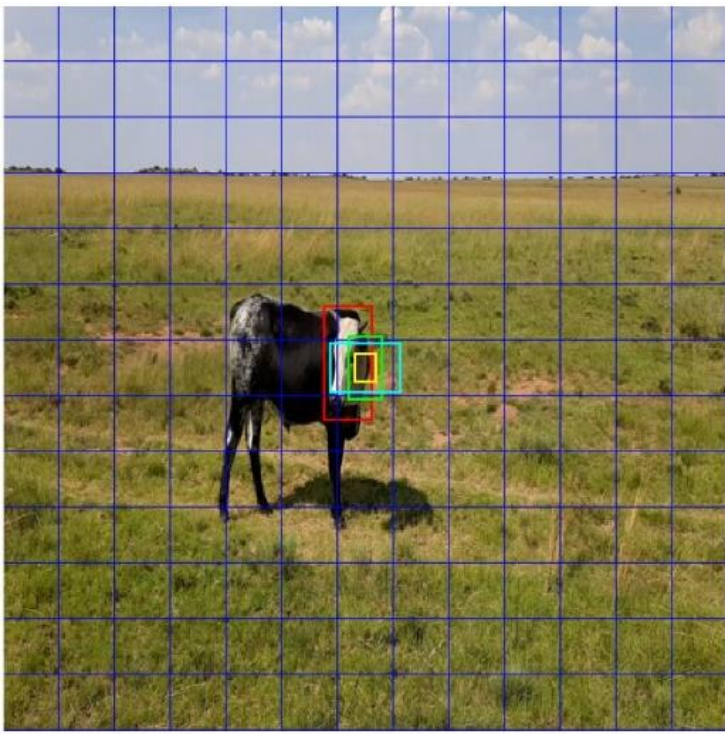


Figure 6.12: 416 x 416 with 13 grid

Figure 6.13 shows the same image with a 26 × 26 grid.

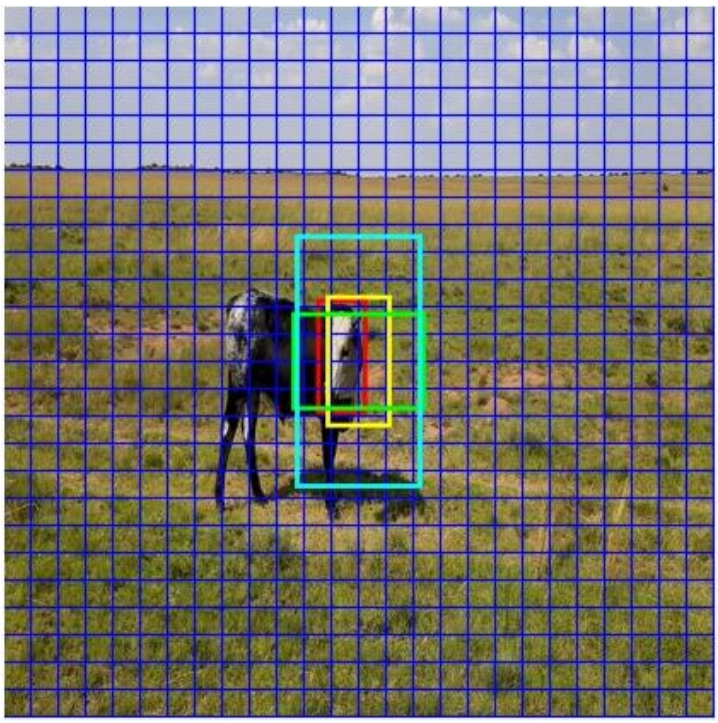


Figure 6.13: 416 x 416 with 26 grid

and Figure 6.14 shows the 52 × 52 grid, highlighting how YOLOv3 divides the image into increasingly finer grids for object detection.

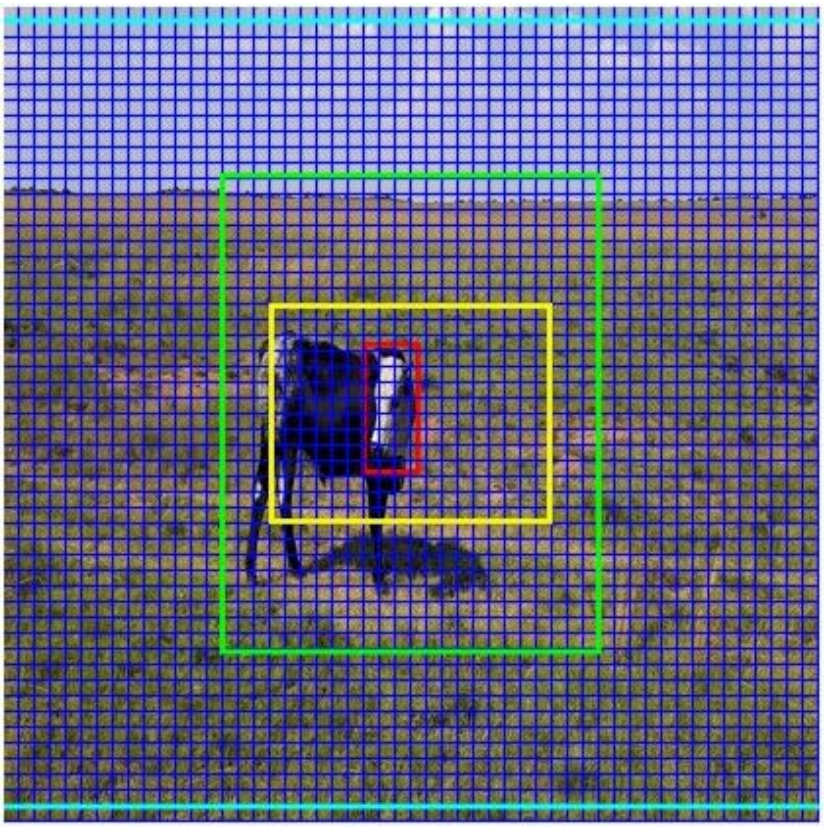


Figure 6.14: 416 x 416 with 52 grid

Figure 6.15 depicts the final cattle head identification, where the red rectangle indicates the detected cattle head in the 416 × 416 resolution image. This visualisation demonstrates the system's capability to accurately localise individual animals within the image.

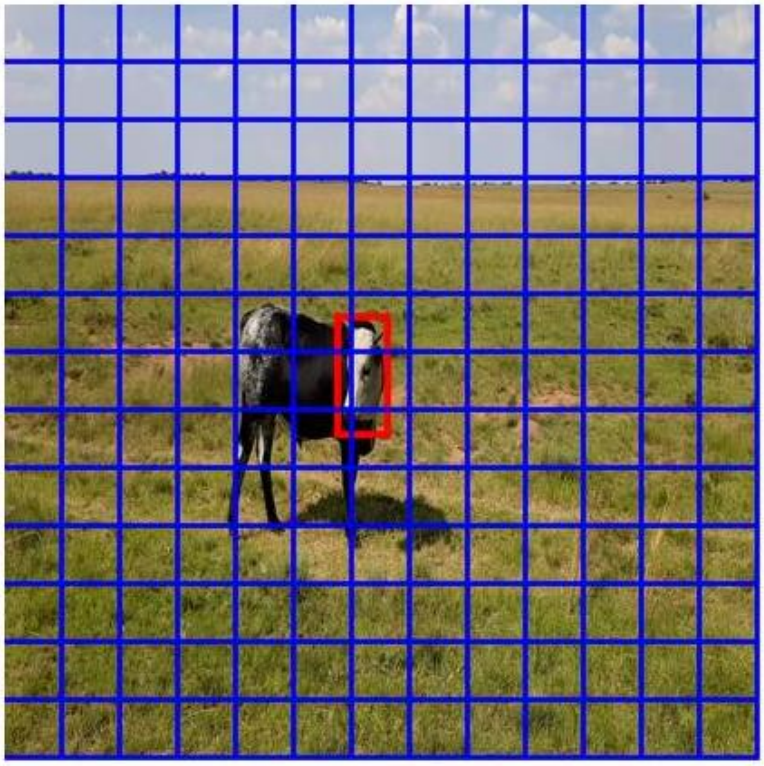


Figure 6.15: 416 x 419 resolution cattle head identification

Figure 6.15 depicts the cattle head identification, with the red rectangle indicating the final cattle head identification.

In an image, objects may vary in size and shape, and to accurately identify each one, object detection algorithms generated several bounding boxes around potential objects (as shown in Figure 6.16). Ideally, each object should be enclosed by a single bounding box, like Figure 6.22. To determine the most accurate bounding box among the multiple predictions, object detection algorithms employed a technique called non-max suppression. This method is designed to eliminate less likely bounding boxes and retain only the most probable one.

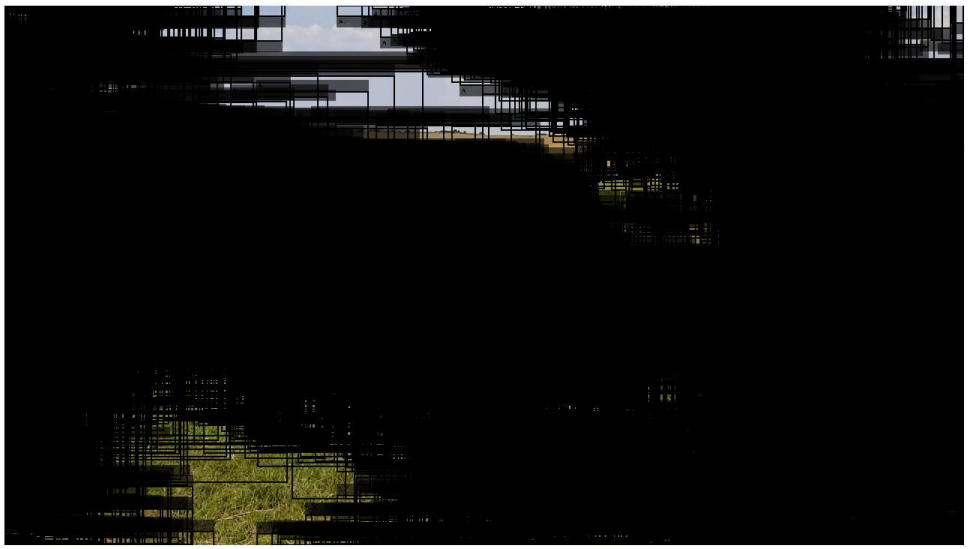


Figure 6.16: Showing every predicted bounding box with a rating of 0.9 or higher

It is important to note that although multiple predictions were made, they were all generated simultaneously - the neural network only needed to run once. This is what makes YOLO both highly efficient and fast [92].

Figure 6.17: Showing every 10th predicted bounding box with a rating of 0.9 or higher

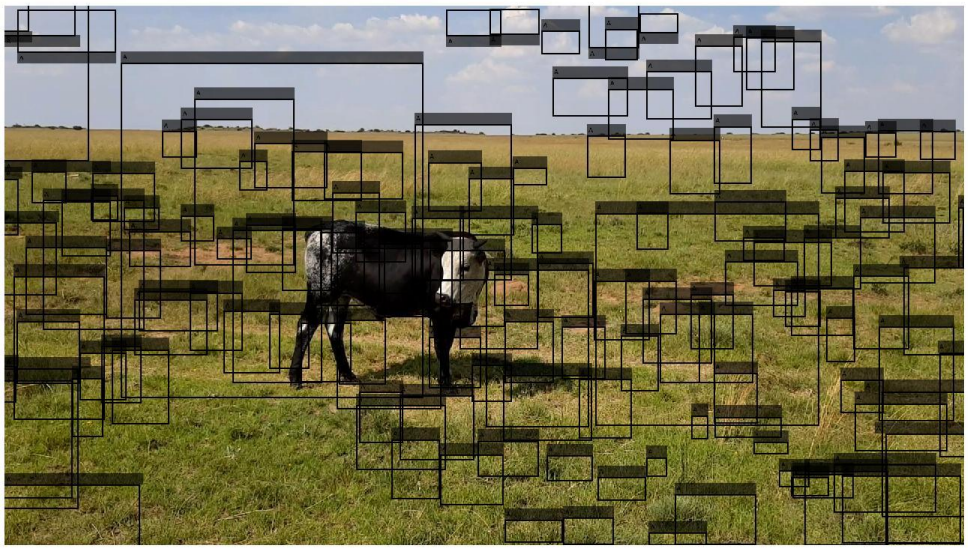


Figure 6.18: Showing every 50th predicted bounding box with a rating of 0.9 or higher

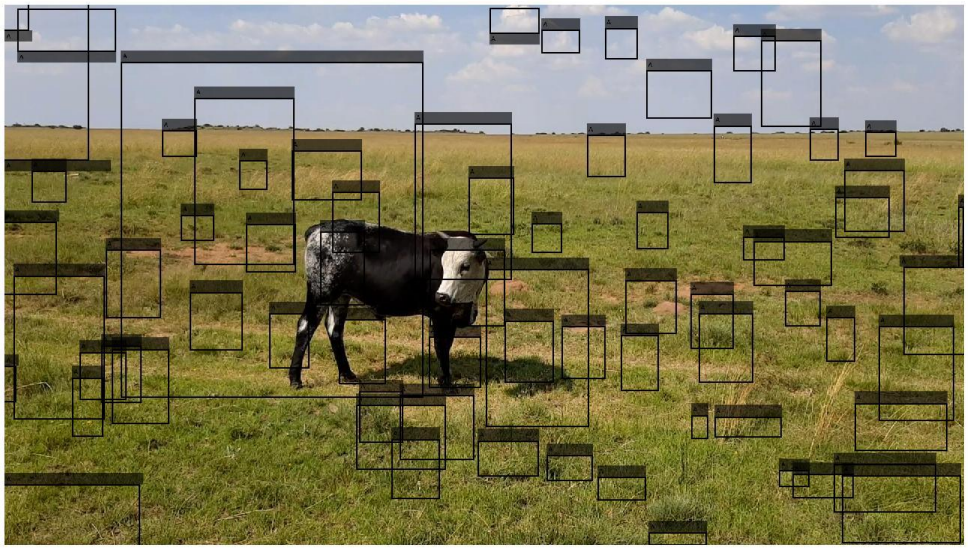


Figure 6.19: Showing every 100th predicted bounding box with a rating of 0.9 or higher

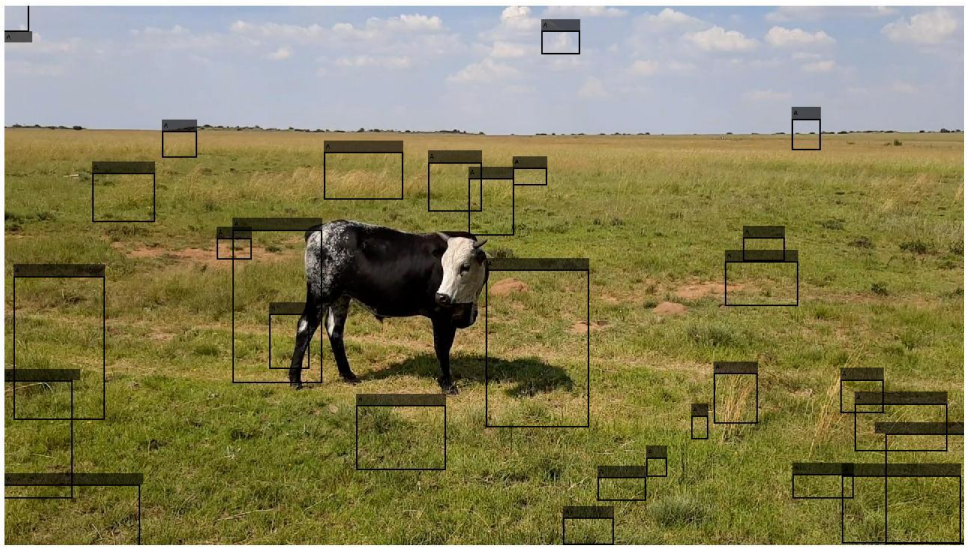


Figure 6.20: Showing every 250th predicted bounding box with a rating of 0.9 or higher

Figure 6.21: Showing every 1000th predicted bounding box with a rating of 0.9 or higher

Although multiple bounding boxes were identified, non-maximum suppression was used to identify the box with the highest confidence score. This process then identified the object in the image. See Figure 6.22.

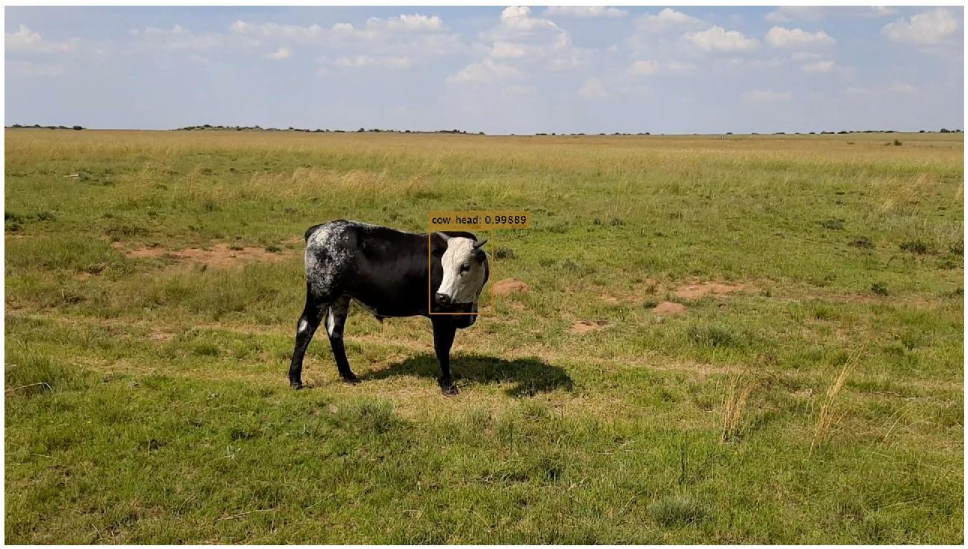


Figure 6.22: Showing the classified image

Figure 6.23 showcases the successful results of YOLOv3 identifying cattle heads from a photo, with training conducted using Google Colab Notebooks [18].

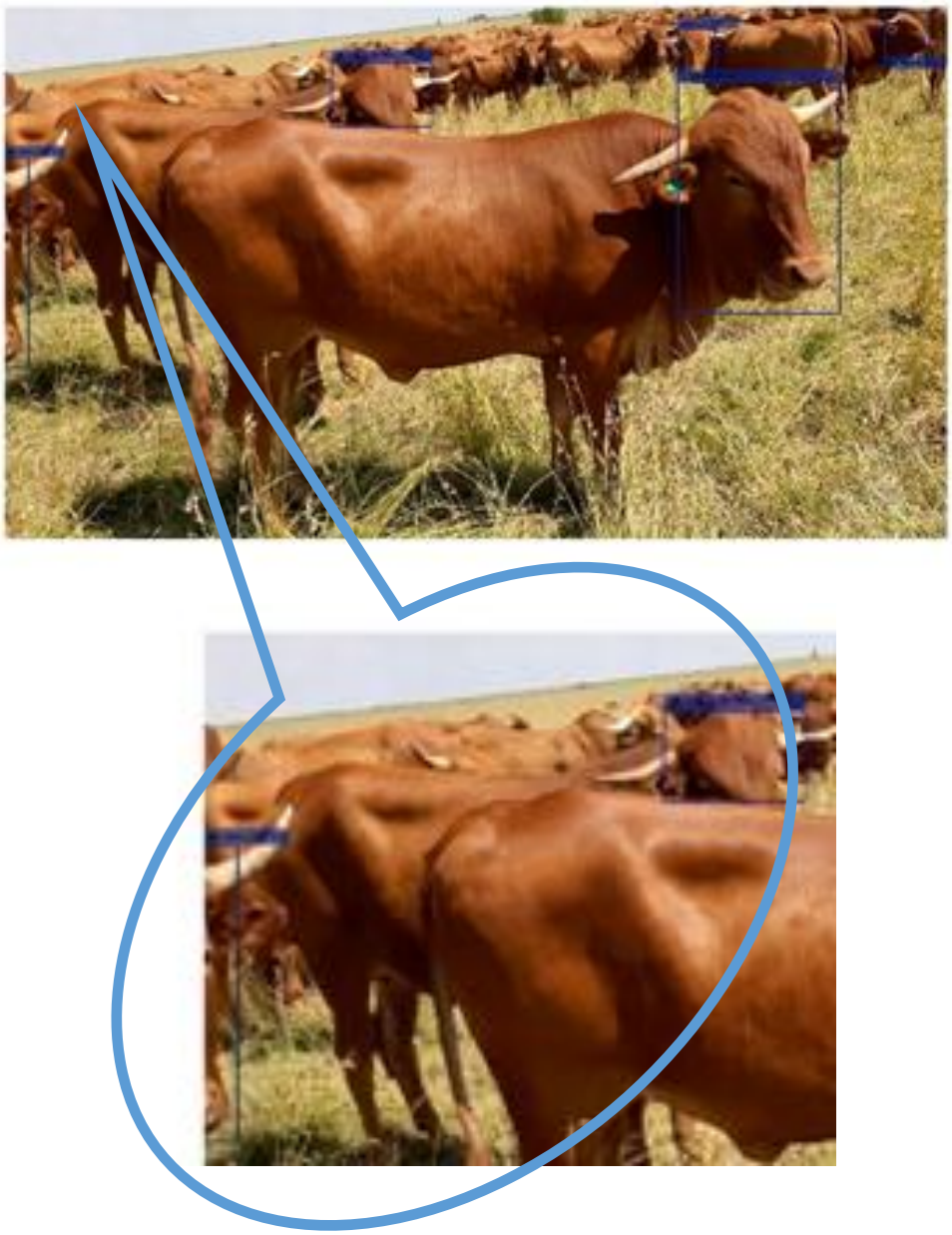


Figure 6.23: Implementation of YOLOv3 to identify cattle heads [18]

In Figure 6.23, there are partial cattle heads that were also correctly identified.

Figure 6.24 shows a captured frame from a more ‘crowded’ video source to identify cattle heads with YOLOv3 on MATLAB® [18].

Figure 6.24: Implementation of YOLOv3 on video source [18]

In this instance, the systems successfully detected 35 cattle heads with a confidence threshold of 0.9 and above. Figure 6.25 and Figure 6.26 display the 15 cattle heads with the highest threshold scores, marked with yellow rectangles. The inference time for each video frame varied between 82.6981 and 87.5616 seconds.

The tests were conducted on a computer with an Intel® Core™ i5-6500 CPU @ 3.20GHz, 32GB RAM, and a Windows 10 Pro 64-bit operating system. While the computer possessed an NVIDIA® GeForce® GTX1070 graphics adaptor, it was not utilized in the initial tests.

Furthermore, comparable results for cattle head identification were obtained when implementing a YOLO system on a Jetson Nano [18].

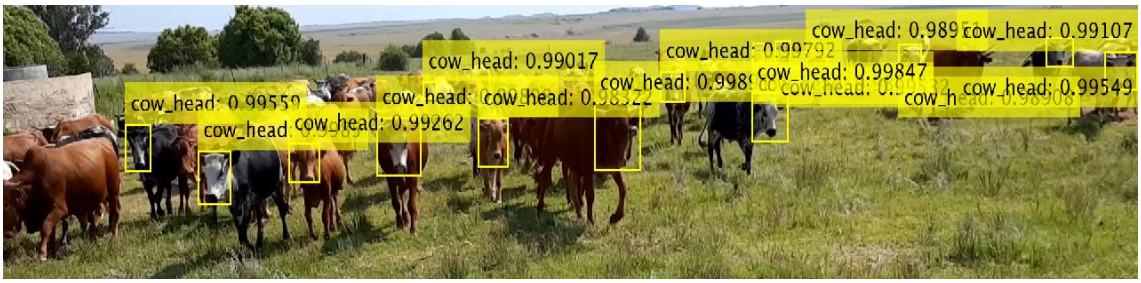


Figure 6.25: Implementation of YOLOv3 to identify cattle heads [18]

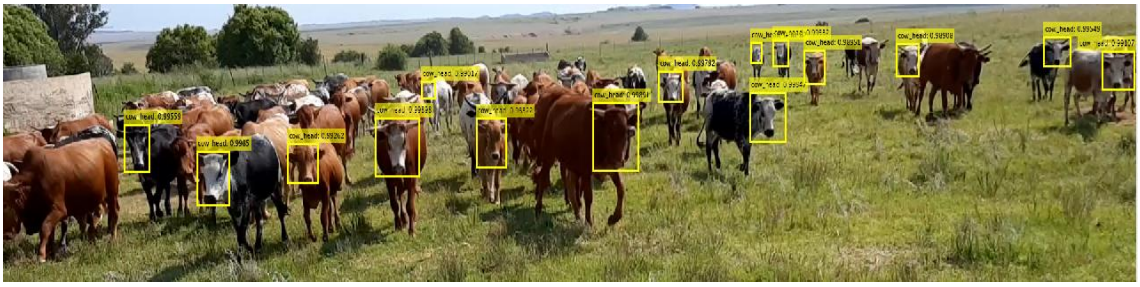


Figure 6.26: Identification of cattle heads [18]

Figure 6.26 depicts the 15 identified cattle heads, but the box labels were reformatted, to see the identified cattle heads better [18].

Figure 6.27: Identified animals from trained data

6.5 YOLOv4 Training

The AlexeyAB Darknet repository [93] was used for training the YOLOv4 Neural Network. Training a YOLOv4 model for object classification involves a series of steps that prepare the model to accurately identify objects in images. The following is a breakdown of the key steps in the training process:

6.5.1. Dataset Preparation

Labelling and Annotation: The first step is to prepare the dataset, which includes many images with annotated bounding boxes for each object. Each image is labelled with its corresponding object classes and bounding box coordinates. Common annotation formats include the Common Objects in Context dataset (COCO) and Pattern Analysis, Statistical Modelling, Computational Learning (PASCAL), and Visual Object Classes (VOC).

Before YOLO can be utilised, training data needs to be prepared. Each image must be accompanied by a text file that specifies the bounding box coordinates,

sizes, and classes for each object. The images for training the network were captured from a video file. Each image was acquired as a frame in the video file.

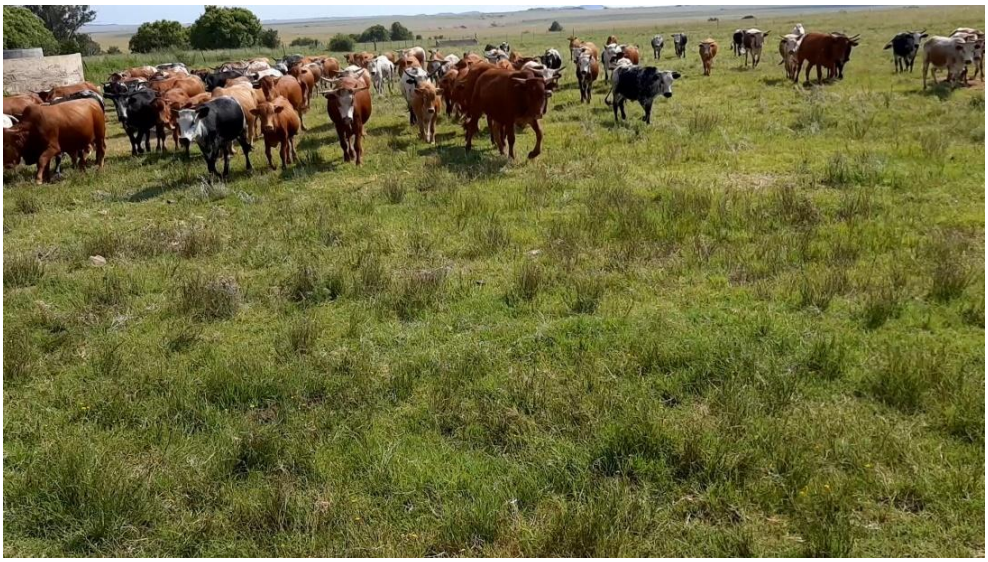
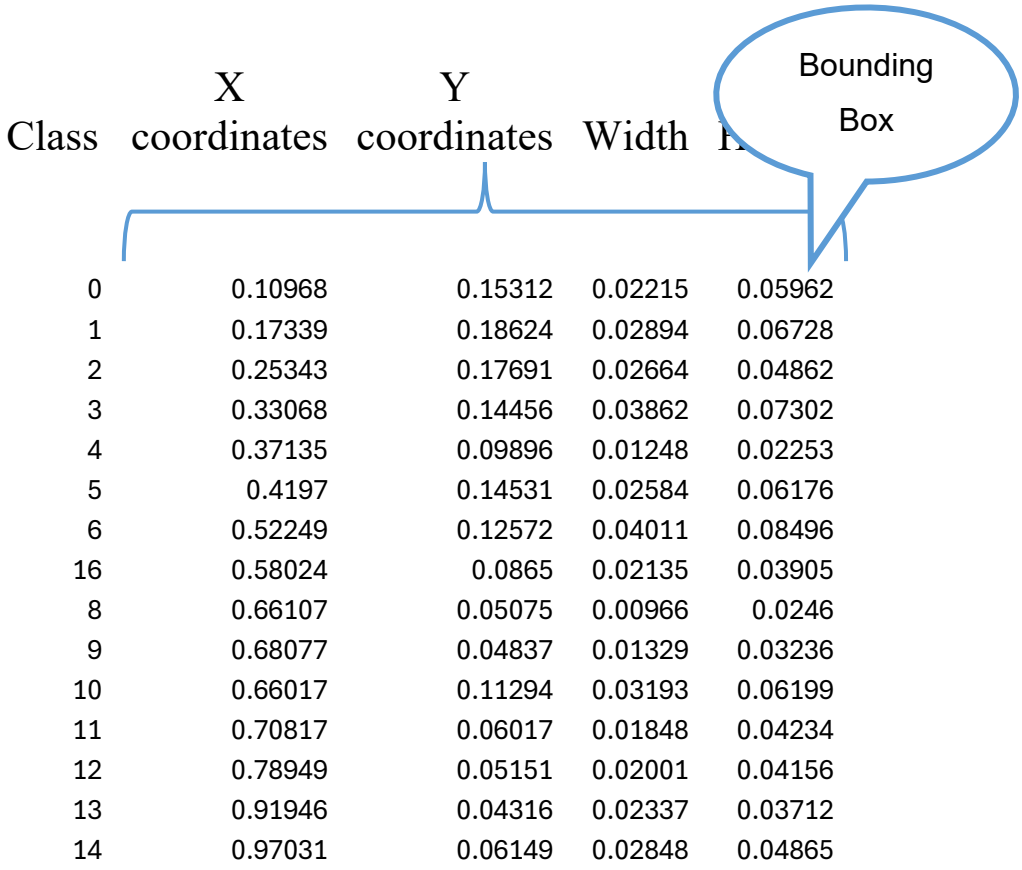


Figure 6.28: Original image



Class	X coordinates	Y coordinates	Width	H
0	0.10968	0.15312	0.02215	0.05962
1	0.17339	0.18624	0.02894	0.06728
2	0.25343	0.17691	0.02664	0.04862
3	0.33068	0.14456	0.03862	0.07302
4	0.37135	0.09896	0.01248	0.02253
5	0.4197	0.14531	0.02584	0.06176
6	0.52249	0.12572	0.04011	0.08496
16	0.58024	0.0865	0.02135	0.03905
8	0.66107	0.05075	0.00966	0.0246
9	0.68077	0.04837	0.01329	0.03236
10	0.66017	0.11294	0.03193	0.06199
11	0.70817	0.06017	0.01848	0.04234
12	0.78949	0.05151	0.02001	0.04156
13	0.91946	0.04316	0.02337	0.03712
14	0.97031	0.06149	0.02848	0.04865

Figure 6.29: YOLOv4 training data text file contents

Figure 6.29 shows an example of the text file that is needed with the original image (Figure 6.28) to train the data. The first column is the class of the objects, the second and third columns are the x and y coordinates of the bounding box. Columns four and five are the width and height of the bounding box. In this specific example, 16 classes or different individual animals were identified. Class 15 is not shown in this image.

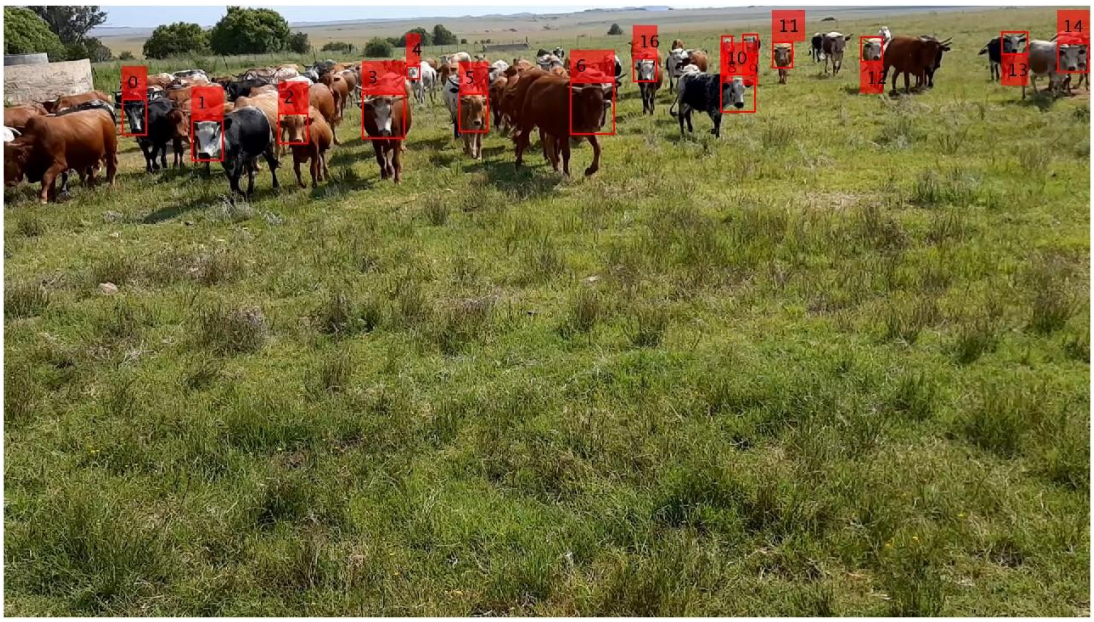


Figure 6.30: YOLOv4 Bounding boxes on training image

Figure 6.30 shows the bounding boxes with the labelled classes for training of the YOLO network. Not all animals are clearly visible and were therefore not used in the image labelling.

Due to the large amount of data needed to train the YOLOv4 NN, the previously developed and trained YOLOv3 NN was used to identify cattle heads in the captured images. The identified cattle heads were then labelled using a written MATLAB® program, which enabled the user to label the bounding boxes of the individual animals [18].

Seventy images were used to train the network for this test. Twenty-two animals were identified in the 70 images. Table 6-1 indicates which animals appear in each of the 70 images during training.

Table 6-1: Number of animals labelled during training in images

Image Nr	Animal 1	Animal 2	Animal 3	Animal 4	Animal 5	Animal 6	Animal 7	Animal 8	Animal 9	Animal 10	Animal 11	Animal 12	Animal 13	Animal 14	Animal 15	Animal 16	Animal 17	Animal 18	Animal 19	Animal 20	Animal 21	Animal 22
1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	1	0	0	0	0	0
2	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	0	1	0	0	0	0	0
3	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0
4	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0
5	1	1	1	1	0	1	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0
6	1	1	1	1	0	1	1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0
7	1	1	1	1	0	1	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0
8	0	1	1	1	0	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0
9	0	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1	0	0	0	0	0	0
10	0	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1	0	0	0	0	0	0
11	0	1	1	1	0	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0
12	0	1	1	1	0	1	1	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0
13	0	1	1	1	0	1	1	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0
14	0	1	1	1	0	0	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0
15	0	1	1	1	0	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0
16	0	1	1	1	0	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0
17	1	1	1	1	0	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0
18	0	1	1	1	0	1	1	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0
19	1	1	1	1	0	1	1	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0
20	0	1	1	1	0	1	1	1	0	0	1	1	1	1	0	1	0	0	0	0	0	0
21	0	1	1	1	0	1	1	1	0	0	1	0	1	1	0	1	0	0	0	0	0	0
22	0	1	1	1	0	1	1	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0
23	0	1	1	1	0	1	1	1	1	0	1	0	1	1	0	1	0	0	0	0	0	0
24	0	1	1	1	0	1	1	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0
25	0	1	1	1	0	1	1	0	1	0	1	0	1	1	0	1	1	0	0	0	0	0
26	1	1	1	1	0	1	1	0	1	0	1	0	1	1	0	1	1	0	0	0	0	0
27	0	1	1	1	0	1	1	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0
28	1	1	1	1	0	1	1	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0
29	1	1	1	1	0	1	1	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0
30	1	1	1	1	0	1	1	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0
31	1	1	1	1	0	1	1	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0
32	1	1	1	1	0	1	1	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0
33	1	1	1	1	0	1	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0
34	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
35	1	1	1	1	0	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
36	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
37	1	1	1	1	1	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
38	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
39	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0
40	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0
41	1	1	1	1	0	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0
42	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0
43	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
44	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
45	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0
46	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
47	1	1	1	1	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
48	1	1	1	1	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0
49	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0
50	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
51	1	1	1	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0
52	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
53	1	1	1	1	0	1	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0
54	0	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
55	0	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
56	0	1	1	1	0	1	1	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0
57	0	1	1	1	0	1	1	0	0	0	1	1	0	0	0	0	0	1	1	0	1	0
58	1	1	1	1	0	1	1	0	0	0	1	0	1	0	0	0	1	1	1	0	1	0
59	1	1	1	1	0	1	1	0	0	0	1	0	1	0	0	1	1	1	1	0	0	0
60	0	1	1	1	0	1	1	0	0	0	1	0	1	0	0	1	1	1	1	0	1	0
61	1	1	1	1	0	0	1	0	0	0	1	0	1	0	0	1	1	1	1	0	1	0
62	1	1	1	1	0	1	1	0	0	0	1	0	1	0	0	1	1	1	1	0	0	0
63	1	1	1	1	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0
64	1	1	1	1	0	1	1	0	0	0	1	0	1	0	0	1	0	0	1	0	1	0
65	1	1	1	1	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0
66	1	1	1	1	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0
67	1	1	1	1	0	1	1	0	0	0	1	0	0	0	0	1	1	1	1	0	1	0
68	1	1	1	1	0	1	1	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0
69	0	1	1	1	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0
70	1	1	1	1	0	1	1	0	0	0	1	0	1	0	0	1	1	0	1	0	1	1
	47	70	70	70	5	65	70	16	27	4	70	33	62	33	12	22	24	16	15	1	12	1

6.5.2. Dataset Splitting:

The dataset is usually split into training, validation, and testing sets. The training set is used to train the model, the validation set is used to adjust and optimise hyperparameters, and the test set is used to evaluate the final model performance. Common ratios that are used for training are 80:20 and 90:10. The images were split into a 90:10 ratio, with 90% used for training and 10% reserved for verification. This was done due to the small number of images available for training.

6.5.3. Data Augmentation

YOLOv4 applies data augmentation techniques to improve the robustness of the model by simulating different real-world conditions. Augmentation methods include:

- *Random Scaling:* Changing the size of the image.
- *Cropping:* Randomly cropping parts of the image.
- *Flipping:* Horizontally or vertically flipping the image.
- *Colour Jittering:* Altering the brightness, contrast, or saturation. These augmentations help the model generalise better and learn to recognise objects under various transformations.

6.5.4. Anchor Boxes Initialisation

Before training, YOLOv4 calculates optimal anchor boxes based on the size distribution of objects in the dataset. Anchor boxes represent the expected shapes and sizes of objects in the dataset, and YOLOv4 uses them to make initial predictions. During training, these anchor boxes are adjusted to better fit the detected objects.

6.5.5. Define Loss Function

YOLOv4 optimises a multi-part loss function during training, which includes:

- *Localisation Loss:* Measures how well the predicted bounding box matches the ground truth box.
- *Confidence Loss:* Assesses the model's confidence in whether an object is present in the predicted box.

- *Classification Loss:* Evaluates how accurately the model predicts the correct class label for each detected object. The goal of training is to minimise this loss across the dataset.

6.5.6. Network Initialisation and Fine-Tuning

- *Pre-trained Weights:* YOLOv4 training typically starts with pre-trained weights from a model trained on a large dataset (such as ImageNet). This provides a strong starting point and speeds up the convergence of the model. For this study, the COCO pretrained weights were used.
- *Fine-Tuning:* The network is then fine-tuned on the specific dataset. This step involves adjusting the weights of the model to adapt to the objects and classes of the new dataset.

6.5.7. Training Loop

- *Forward Pass:* The input images are fed through the YOLOv4 network, and predictions for bounding boxes, class probabilities, and confidence scores are generated.
- *Loss Calculation:* The loss function calculates the error between the predicted bounding boxes, classes, and confidence scores and the ground truth annotations.

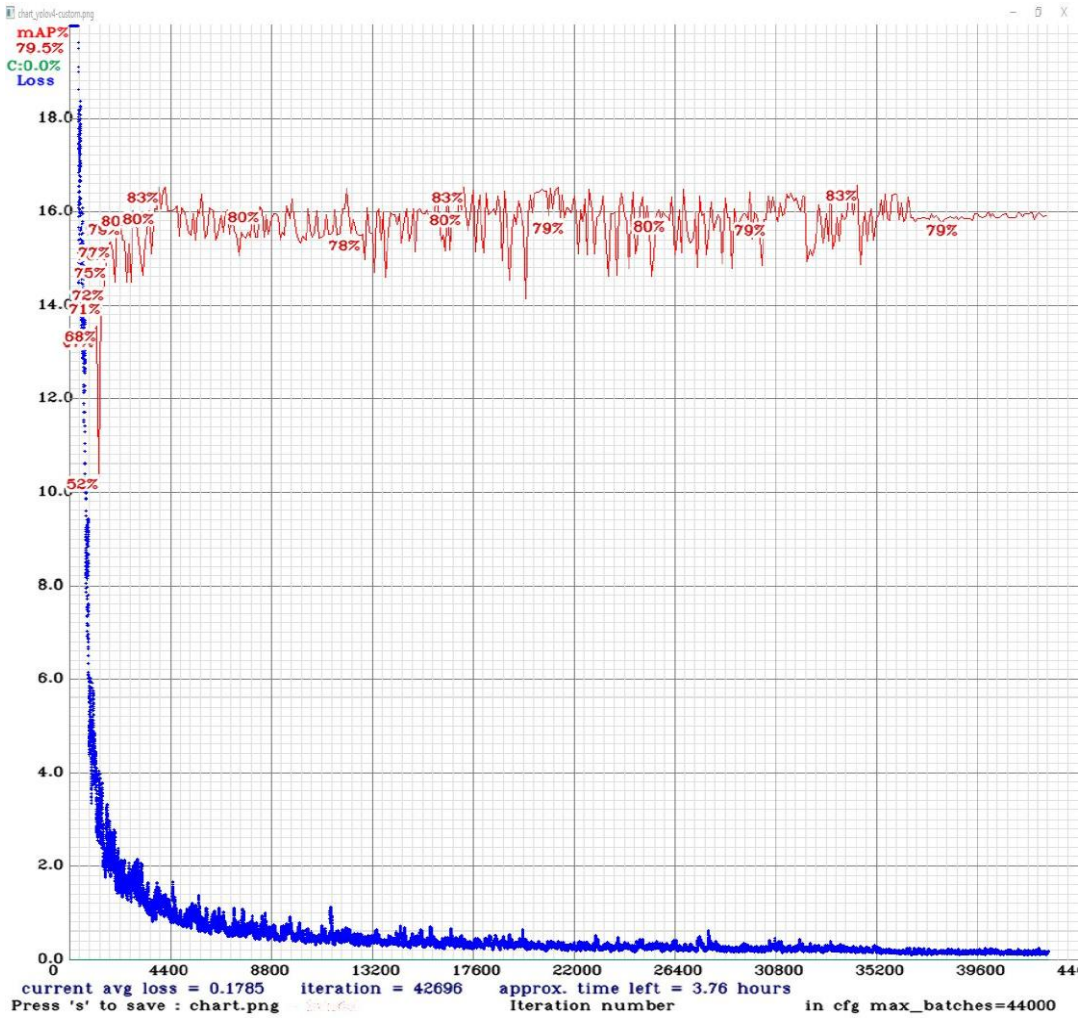


Figure 6.31: YOLOv4 Training graph

The YOLO training graph in Figure 6.31: YOLOv4 Training graph typically shows two key metrics over time: the training loss and the mean average precision (mAP).

- *The blue curve (Training Loss):* This represents the error or loss the model experiences during training. In the case of YOLOv4, the loss is calculated using a metric called Complete Intersection-over-Union (CIoU), which measures how well the predicted bounding boxes overlap with the ground-truth boxes. As training progresses, this curve is expected to decrease, indicating that the model is improving its accuracy in detecting objects in the training dataset.
- *The red curve (mAP@0.5):* The red line corresponds to the mean average precision (mAP) at a 50% Intersection-over-Union (IoU) threshold, commonly referred to as mAP@0.5. This evaluates how well the model generalises to unseen data by testing it on a validation set. A higher mAP

score means the model is more accurate in detecting objects in new images. During training, if the mAP score improves, it indicates that the model is not just overfitting to the training data but is also performing well on validation data. The blue curve helps monitor the model's training performance, while the red line indicates how well the model generalises to unseen data. The ideal scenario is for the training loss to decrease while the mAP increases or stabilises at a high value.

Due to network and Google Colab issues, it was decided to train the weights for the YOLO network on a desktop i5 personal computer. The training took six days to complete.

6.5.8. Backpropagation

The model updates its weights using backpropagation and an optimisation algorithm, typically Stochastic Gradient Descent (SGD) or its variant, such as Adam or momentum-based optimisers.

6.5.9. Learning Rate Scheduling

A learning rate schedule is used to control the learning process. YOLOv4 often uses techniques like cosine annealing to gradually reduce the learning rate during training, helping to fine-tune the model more effectively.

6.5.10. Validation

During training, the model's performance is periodically evaluated on the validation set. The model's predictions are compared to the ground truth labels, and performance metrics such as precision, recall, mean average precision (mAP), and loss are tracked. This step helps in monitoring overfitting and adjusting hyperparameters.

- *Hyperparameter Tuning:* Based on validation results, various hyperparameters are tuned to optimise the model's performance. These include:

- *Learning Rate:* Adjusted for optimal convergence.

- *Batch Size:* Number of images processed in each training iteration.
- *Number of Epochs:* How many times the entire dataset is passed through the network during training. Hyperparameter tuning is critical for finding the best model performance without overfitting.
- *Early Stopping (Optional):* If the model's performance on the validation set stops improving, early stopping can be used to prevent overfitting. This step halts training if the model begins to perform worse on the validation set, even if it continues improving on the training set.
- *Model Testing:* After training, the model is evaluated on the test dataset. This final testing phase assesses how well the model generalises to unseen data. Performance metrics such as precision, recall, and mAP are computed to measure the effectiveness of the trained model.
- *Model Deployment:* Once the model has been trained and tested, it can be deployed in a real-world environment. The model is saved with its trained weights and can be integrated into applications where it can process real-time video feeds or images for object detection.

This structured approach ensures that the YOLOv4 model effectively learns to detect objects accurately in diverse images.

6.6 Conclusion

The algorithms developed on the PC demonstrated successful identification of cattle heads and, to a lesser extent, individual animals. The effectiveness of AI-based identification heavily depends on the quality and diversity of the training data. Improvements are needed in the algorithms and training process to achieve better identification of individual animals. However, the successful identification of cattle heads will aid in automating the generation of training data, which will ultimately enhance the training of algorithms.

CHAPTER 7: HARDWARE IMPLEMENTATION

7.1 Hardware including cameras.

The neural network was implemented on the NVIDIA® Jetson Nano platform. The NVIDIA® Jetson Nano offers relatively low power consumption and a small footprint which made it ideal for implementation in a feeder system. Shown below are the technical specifications of the NVIDIA® Jetson Nano.

Table 7-1: NVIDIA® Jetson Nano Specifications

GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I^2C, I^2S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector

As can be seen in the specifications above, the NVIDIA® Jetson Nano can accommodate camera hardware for the capturing of images to be used in the implementation of the neural network. The carrier board also offers GPIO pins that were useful when implementing the image system with the feeder system.

The images and video used in this study were collected by a cell phone as well as a Logitech C270 USB webcam connected to the Jetson Nano.

7.2 Jetson Nano implementation

The initial testing with the Jetson Nano included eight cows and 31 images each to construct the algorithm. To access the Python code, Jupyter Notebook was used on Google Chrome Browser. The default IP address was 192.168.55.1:8888 (default password = dlinano). This notebook is an interactive data collection, training, and testing tool, provided as part of the NVIDIA® Deep Learning Institute (DLI) course, "Getting Started with AI on Jetson Nano". It is designed to be run on the Jetson Nano in conjunction with the detailed instructions provided in the online DLI course pages.

To start the tool, set the Camera and Task code cell definitions, then execute all cells. The interactive tool widgets at the bottom of the notebook will display. The tool can then be used to gather data, add data, train data, and test data iteratively and interactively

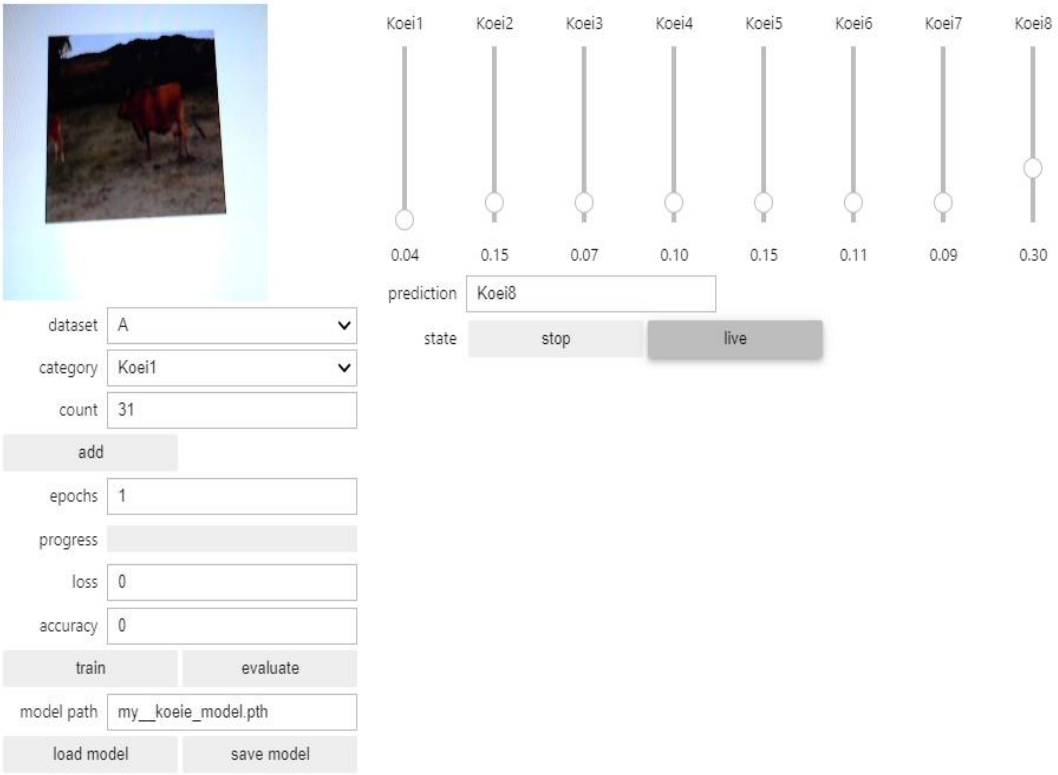


Figure 7.1: Jetson Nano

To implement YOLOv4 on a Jetson Nano, the process begins by setting up the environment and optimising it for the device's GPU capabilities to ensure efficient object detection.

- *Setup and Dependencies:* First, install necessary dependencies, including CUDA and cuDNN, to leverage the Nano's GPU. This setup will also involve installing Darknet, the framework for running YOLO, and ensuring it is compiled with GPU and OpenCV support for optimal performance and visualisation capabilities.
- *Model Download and Configuration:* Download the YOLOv4 model weights and configuration files. The model's configuration may require adjustments to suit the Jetson Nano's processing limitations, such as resising input dimensions to balance speed and accuracy.
- *Optimisation for Inference:* Use TensorRT, NVIDIA's deep learning inference optimiser, to enhance YOLOv4's efficiency on the Nano. TensorRT converts YOLOv4 into a format that runs faster on the Nano's hardware, reducing inference time significantly.
- *Running Inference:* After setup and optimisation, run YOLOv4 on test images or video feeds. The output should display bounding boxes and labels for detected objects, demonstrating real-time or near-real-time detection capabilities, depending on input resolution and model size.

This implementation maximises the Jetson Nano's efficiency while maintaining YOLOv4's high detection accuracy, making it suitable for embedded and edge applications.

7.3 Intelligent feeder

Although the hardware design of the automatic feeder did not form part of the study, the final step was the investigation of the implementation and integration of all the different systems in an intelligent feeder system for use on farms. If a reliable automated vision system can be developed, each animal can be fed the correct diet and amount of food, and the correct medicine can be administered. At the same time, statistics can be gathered about the animal's behaviour. To

implement this system, a user-friendly way to add new animals to the database will have to be developed in future studies. The successful implementation of the system may lead to a master's study and finally a commercial product. The data gathered can be used to reduce the cost of farming the animals, as well as predict the growth in animal numbers.

Lastly, the completed vision system was tested, and the results were evaluated.

7.4 Prototype for proof of concept of the animal feeder.

Shown below is a simple Arduino Uno implementation to show the proof of concept of a feeder that can be used with the developed Jetson Nano animal identification system. The Arduino Uno will accept RS232 serial inputs from the 40-pin Expansion header (J6) of the Jetson Nano. The serial pins (pins eight and ten) of the expansion header are used as a console serial input/output, by default and this must be switched off to be used as a normal serial port.

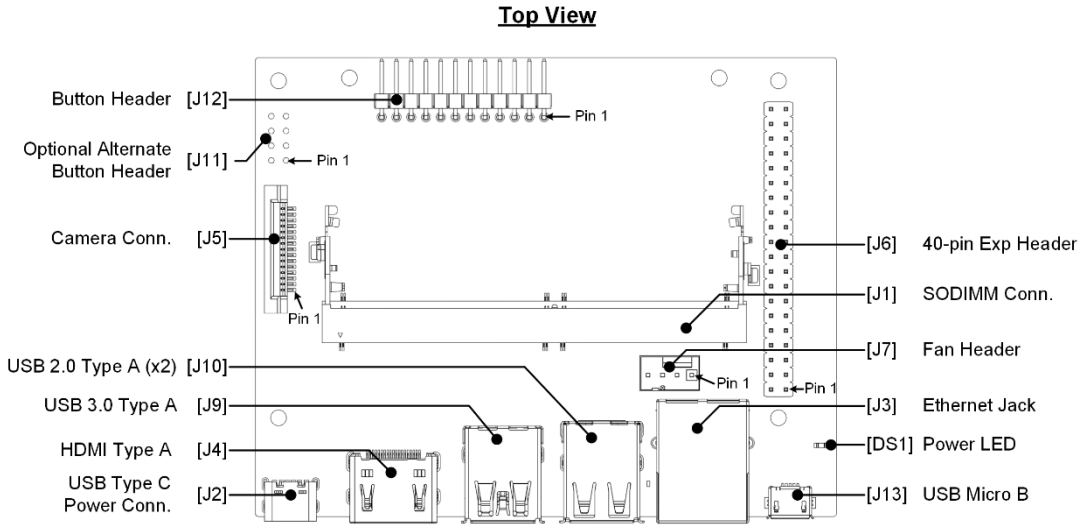


Figure 7.2: Connections of the Jetson Nano carrier board [94]

Figure 7.3: NVIDIA® Jetson Nano in metal enclosure

The Jetson Nano was placed in a metal enclosure for better protection of the printed circuit board.

Table 7-2: Pins on J6 Expansion header [95]

SoC GPIO	Linux GPIO #	Alternate Function	Default Function			Default Function	Alternate Function	Linux GPIO #	SoC GPIO
			3.3 VDC	1	2	5 VDC			
PJ.03	75	GPIO	I2C1_SDA	3	4	5 VDC			
PJ.02	74	GPIO	I2C1_SCL	5	6	GND			
PBB.00	216	AUD_CLK	GPIO	7	8	UART1_TXD	GPIO	48	PG.00
			GND	9	10	UART1_RXD	GPIO	49	PG.01
PG.02	50	UART1_RTS	GPIO	11	12	GPIO	I2S0_SCLK	79	PJ.07
PB.06	14	SPI1_SCK	GPIO	13	14	GND			
PY.02	194		GPIO	15	16	GPIO	SPI1_CS1	232	PDD.00
			3.3 VDC	17	18	GPIO	SPI1_CS0	15	PB.07
PC.00	16	SPI0_MOSI	GPIO	19	20	GND			
PC.01	17	SPI0_MISO	GPIO	21	22	GPIO	SPI1_MISO	13	PB.05
PC.02	18	SPI0_SCK	GPIO	23	24	GPIO	SPI0_CS0	19	PC.03
			GND	25	26	GPIO	SPI0_CS1	20	PC.04
PB.05	13	GPIO	I2C0_SDA	27	28	I2C0_CLK	GPIO	18	PC.02
PS.05	149	CAM_MCLK	GPIO	29	30	GND			
PZ.00	200	CAM_MCLK	GPIO	31	32	GPIO	PWM	168	PV.00
PE.06	38	PWM	GPIO	33	34	GND			
PJ.04	76	I2S0_FS	GPIO	35	36	GPIO	UART1_CTS	51	PG.03
PB.04	12	SPI1_MOSI	GPIO	37	38	GPIO	I2S0_DIN	77	PJ.05
			GND	39	40	GPIO	I2S0_DOUT	78	PJ.06

Figure 7.4: LCD display on Arduino

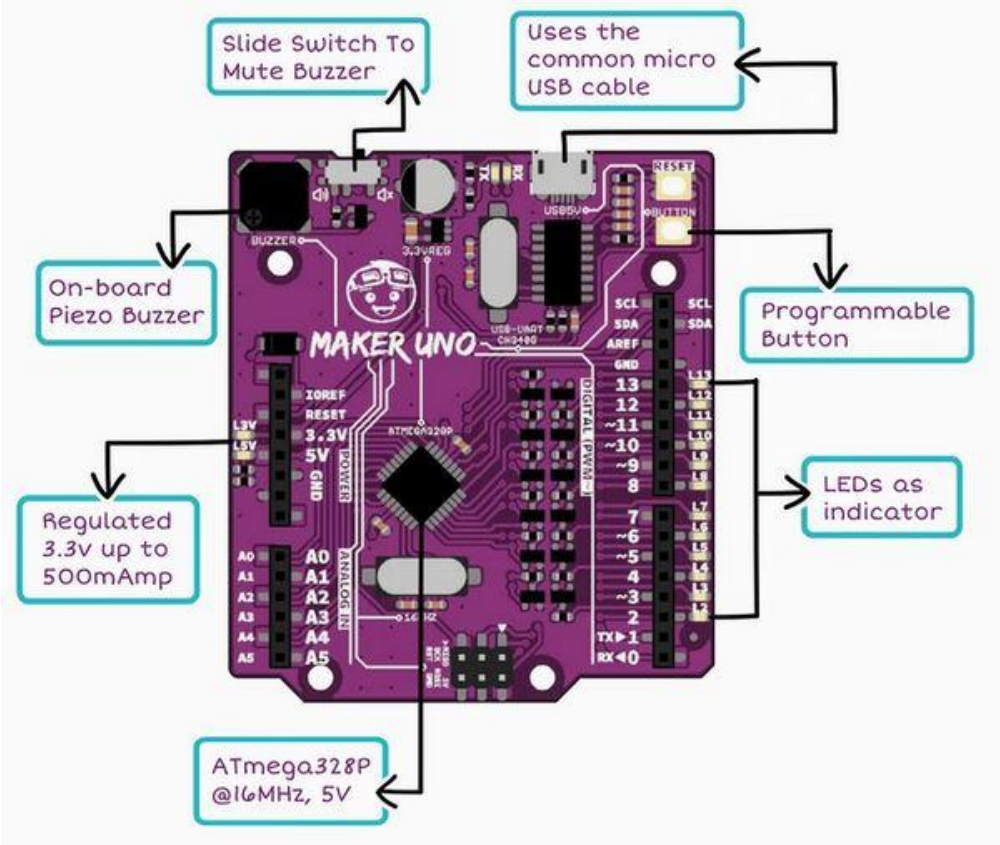


Figure 7.5: Arduino board layout [96]

CHAPTER 8: RESULTS

8.1 Introduction

This chapter focuses on, analysing and interpreting the results of using YOLO for animal recognition. Key metrics like accuracy, precision, recall, F1 score and inference speed will be discussed.

8.2 Experimental Setup Summary

Data (images) with video footage were collected of the cattle in the veld. This video was converted to images by capturing 70 frames from the video. For the training, 22 animals were identified. Table 6-1 provides a summary of the frames which contain each animal, as well as the total number of times each animal is used for training. All animals used for training were cattle of different breeds. The original image dimensions were 1920 x 1080.

The bounding boxes were identified and manually drawn on the images. Training was done on a Windows 10, i5 PC. Due to the poor performance of the screen card, training on average took 1 week for 70 images.

The general sequence of YOLOv4 image detection is outlined as follows:

8.2.1. Input Image Preprocessing

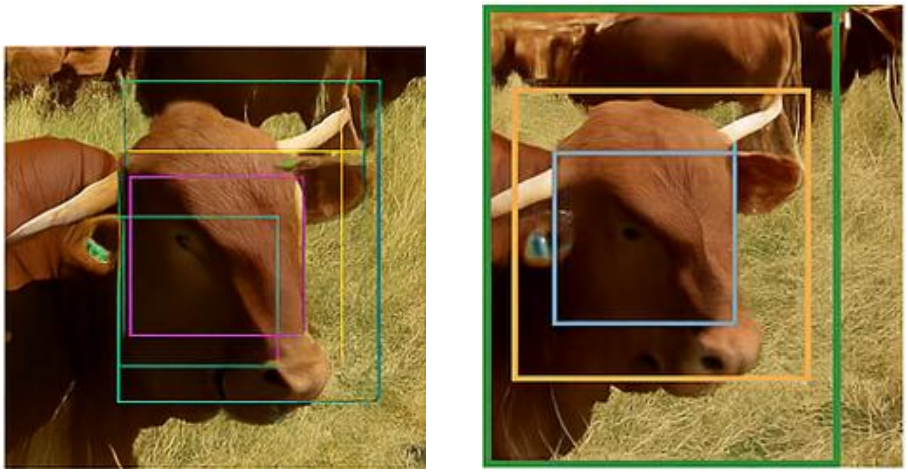
The input image is resized to a fixed size (e.g., 416×416 or 608×608), Pixel values are normalised (e.g., to the range [0,1]), and the image is divided into a grid of cells. Letterbox padding is also done when resizing the image, and PANET and SPP are applied.

(a) (b)

Figure 8.1: Image preprocessing

8.2.2. Feature Extraction (Backbone)

The image passes through CSPDarknet53, which extracts multi-level features. This stage learns edges, textures, and higher-level patterns. The backbone applies Feature Extraction, Edge detection and texture features like horn tips and eyes.



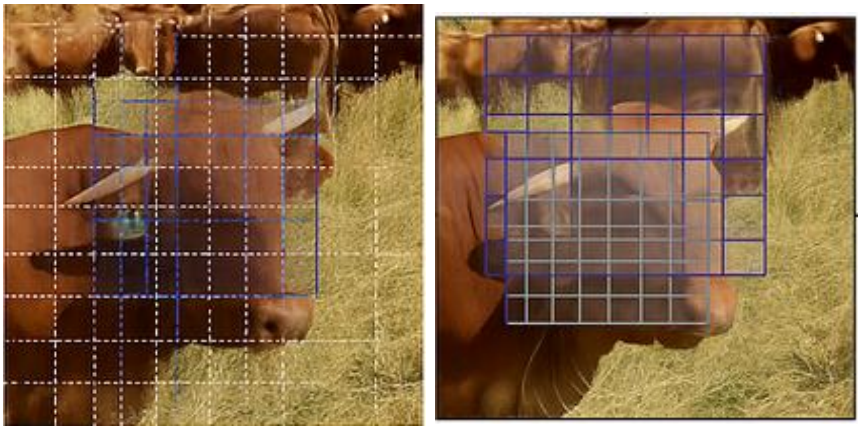
(a) (b)

Figure 8.2: Feature extraction

8.2.3. Neck (Feature Aggregation)

In the neck, Spatial Pyramid Pooling (SPP) enhances receptive fields by combining features from different scales, while Path Aggregation Network (PANet)

integrates low-level and high-level features for better localisation and classification.



(a) (b)

Figure 8.3: Multiscale feature maps

8.2.4. Head (Detection)

YOLOv4 predicts bounding boxes, objectness scores, and class probabilities at three different scales.

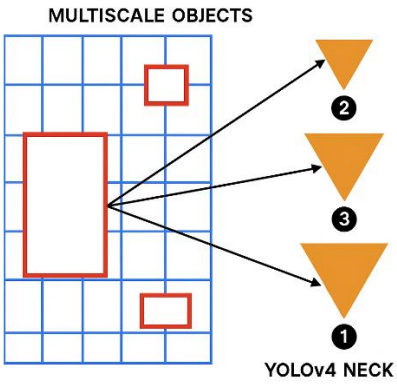


Figure 8.4: Multiscale objects

Anchor boxes are used to predict object sizes at each scale. These multiscale features were detected at d*etection layers 139, 150 and 161.*

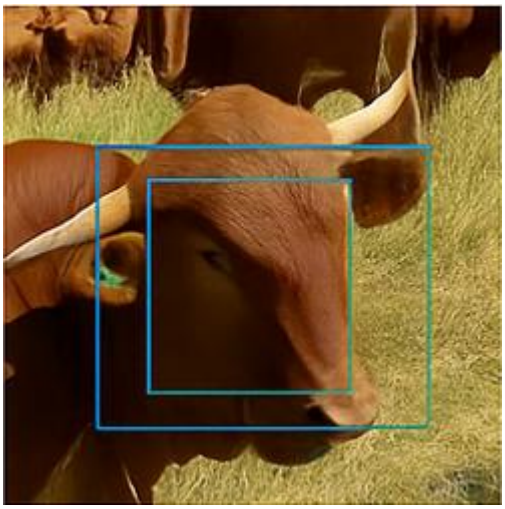


Figure 8.5: Multiscale objects

8.2.5. Bounding Box Prediction

Each cell outputs several bounding box candidates with confidence scores.
Each box has coordinates (x, y, width, height), an objectness score (probability of containing an object) and class probabilities.

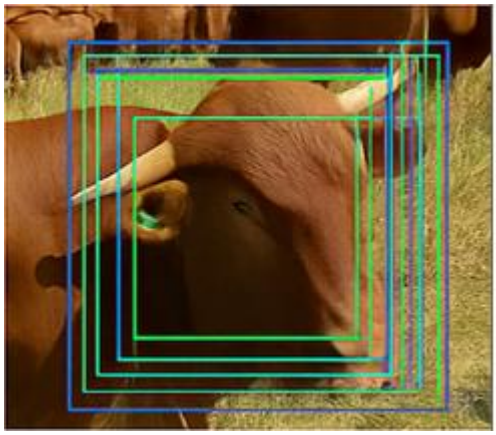


Figure 8.6: Possible bounding boxes

8.2.6. Post-Processing

Sigmoid and softmax activations refine scores, while Non-Maximum Suppression (NMS) removes overlapping boxes, keeping only the best ones.
Final bounding boxes and class labels are thus produced.

Figure 8.7: Detected bounding boxes

8.3 Performance Metrics and Evaluation

The following command can be used to determine the performance metrics of the YOLOv4 model:

"darknet detector map data/obj.data cfg/yolov4-custom.cfg yolov4-custom_final.weights"

The results from this command are as follows:
From the above command, the model's detection performance metrics were as follows:

mAP@0.50 (Mean Average Precision at IoU threshold 50%): 79.55%

Detection Stats:

- Precision: 0.87 (87%)
- Recall: 0.97 (97%)
- F_1-score: 0.92
- True Positives (TP): 67
- False Positives (FP): 10
- False Negatives (FN): 2
- Average IoU: 78.85%

The mean average precision (mAP) calculation for this YOLOv4 object detection model revealed several important insights into its performance. Three detection layers (layers 139, 150, and 161) were used, processing a total of 81 detections, with 69 unique ground truths across the dataset. The mAP@0.50, calculated at an Intersection over Union (IoU) threshold of 50%, reached 79.55%, which shows a relatively strong detection accuracy across classes. The high mAP indicated that the model was performing well across various classes, although there were certain classes (e.g., An_1, An_9, An_16) where precision could improve due to a few false positives. If needed, you could adjust the confidence threshold or the IoU threshold to fine-tune precision and recall based on the application requirements.

Class-specific average precision (AP) scores varied, with most classes achieving an AP of 100%, indicating high detection accuracy. However, some classes, such as An_1 (91.67%) and An_6 (92.86%), had slight misclassifications, and a few classes, like An_9 (50%) and An_16 (25%), struggled with false positives, bringing down their AP. Additionally, several classes had zero detections, implying potential difficulties in identifying certain objects or a lack of true positives.

The model achieved an average IoU of 78.85%, indicating that, when correct, its bounding boxes were generally well-aligned with ground truths. With a total detection time of 4 seconds, YOLOv4 performed efficiently, confirming its suitability for real-time detection tasks.

Options like adjusting the -points flag for different datasets, such as MS COCO or Pascal VOC, are useful in aligning the evaluation approach to specific dataset standards, further enhancing this model's adaptability to various applications.

The F_1-score of 0.92 suggested that the model had a balanced performance in precision and recall at a confidence threshold of 0.25. For the specified confidence threshold of 0.25, the model's overall precision was 87%, with a high recall of 97% and an F_1-score of 0.92, demonstrating balanced performance between accuracy and detection sensitivity. This indicated a high recall rate and a decent precision.

The F_1 score is a metric used to evaluate the accuracy of a model, especially in classification and object detection tasks. It provides a balance between precision and recall, which are both crucial when assessing model performance, especially in situations where the classes might be imbalanced. F_1 Score: The F_1 score combines precision and recall into a single metric by taking their harmonic mean. It is particularly useful when you need a balance between precision and recall, and when you have an uneven class distribution.

Precision is the proportion of correctly identified positive observations out of all observations that the model predicted as positive.

$$\textbf{Precision} = \frac{\textbf{True Positives (TP)}}{\textbf{True Positives (TP)+False Positives (FP)}} \qquad \textbf{(8.1)}$$

Recall is the proportion of correctly identified positive observations out of all actual positive observations.

$$\textbf{Recall} = \frac{\textbf{True Positives (TP)}}{\textbf{True Positives (TP)+False Negatives (FN)}} \qquad \textbf{(8.2)}$$

The F_1 score ranged from 0 to 1, where 1 indicated perfect precision and recall, meaning the model had correctly identified all positives with no false positives or false negatives.

$$\boldsymbol{F_1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}} \qquad \textbf{(8.3)}$$

In YOLOv4, inference speed refers to how quickly the model can process an input image to detect and classify objects within it. This speed is crucial for real-time applications, as it determines how fast the model can provide results after receiving data. YOLOv4 achieves high inference speed through optimised architecture and efficient feature extraction techniques, allowing it to process multiple frames per second (FPS) on standard hardware.

YOLOv4 improves upon previous versions by incorporating enhancements such as Cross-Stage Partial (CSP) connections, which reduce computational load while

maintaining accuracy. The model also uses optimised activation functions and carefully chosen hyperparameters to accelerate processing time without sacrificing detection precision. These features allow YOLOv4 to balance accuracy with speed, making it suitable for time-sensitive tasks, like surveillance and autonomous driving, where rapid and accurate object detection is essential. The Jetson Nano performed facial recognition at 1.5 frames per second.

8.4 Qualitative Analysis of Detection Results

8.4.1. Sample Detection Outputs:

For this experiment, the trained YOLOv4 network discussed in Chapter 6.5, was used. A video stream was passed through the YOLOv4 network running on a Jetson Nano system with the AlexeyAB Darknet repository [93]. The frame rate for processing the data was 1.4 to 1.5 frames per second.

Shown below, in Figure 8.8, is the last of the 70 images that were used for training the YOLOv4 network. To test the operation, the image was passed through the YOLO network running on the Jetson Nano. All the classified animals had a confidence rating of 100%.

Figure 8.8: Last image of the training data

Figure 8.9 depicts frame 110 of a video stream that was not part of the training data of the YOLOv4 network. The bounding boxes and confidence rating can be seen. Two classified animals had a confidence rating of 100%, one had a rating of 99%, and two had ratings of 51% and 43%, respectively.

Figure 8.9: Image 110 of video stream (Not part of the training data)

8.4.2. Partial detections (incomplete or fragmented bounding boxes)

Although only the body of An13 in Figure 8.11 is not fully visible, the animal was still correctly identified.

Figure 8.10: Partial An13

8.4.3. False positives (incorrect animal detections)

Refer to Figure 8.11, where two animals were labelled Animal 6. It should also be noted that the bounding box of the second Animal6 was not around an animal's head. Although this animal was identified as Animal 6, the confidence score was very low (44%) whereas the correct animal 6's score was 87%. Wrong labelling of the training data will lead to the wrong identification of the animals.

Figure 8.11: Image 704 of video stream (Not part of the training data)

8.4.4. False negatives (missed animals)

In some instances, classification might not happen. This could be due to insufficient training data or a very low confidence score, if detected. See Figure 8.12, where Animal 6 was missed in frame 705 but identified in frame 704.

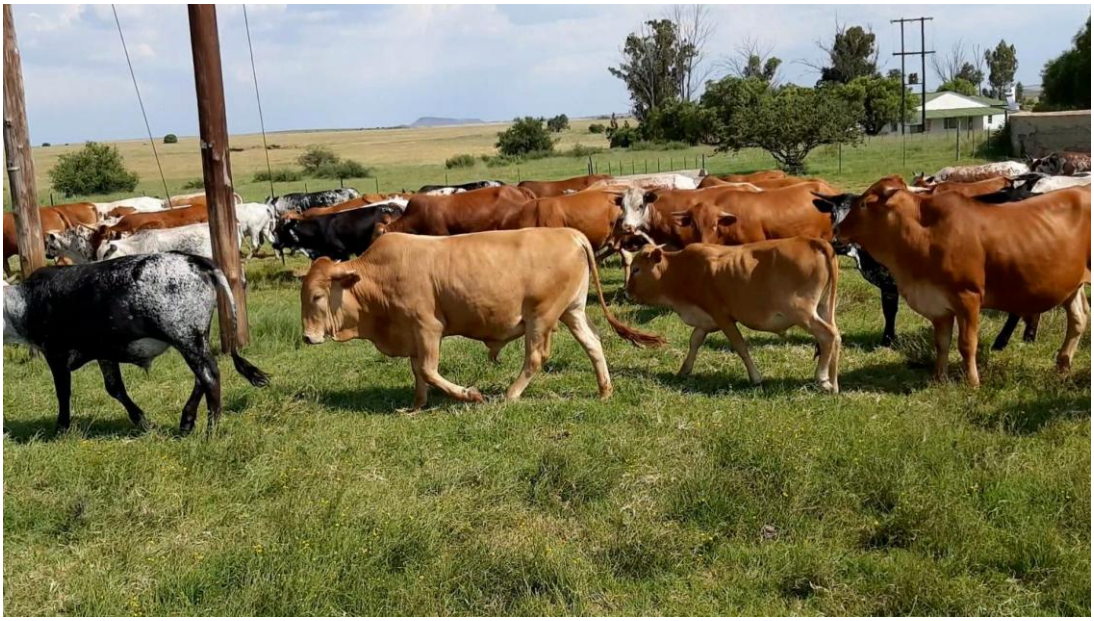


Figure 8.12: Image 705 of video stream (Not part of the training data)

Classification with YOLO in video frames can be inconsistent due to various factors related to the dynamics of video content and the model's design. YOLO performs object detection and classification independently on each frame, which means that variations between consecutive frames can affect its predictions. Key reasons for such inconsistencies include:

8.4.5. Motion Blur

Fast-moving animals or camera movement can cause motion blur, which obscures critical features necessary for accurate classification. If the model cannot detect these features clearly, it may fail to classify the object in certain frames.

8.4.6. Partial Occlusions

An animal may become partially obscured by objects in the environment (e.g., trees, fences, or other animals) or may move out of the frame momentarily. This can prevent YOLO from identifying the object reliably in every frame.

8.4.7. Lighting and Shadows

Variations in lighting conditions, such as flickering light, shadows, or sudden changes in brightness, can alter the appearance of the object. YOLO might classify

the object successfully in one frame where the lighting is favourable, but fail in the next due to these changes.

8.4.8. Pose and Orientation Changes

If the object changes orientation between frames, features critical for classification may no longer be visible. For example, an animal turning its head might obscure key markers, causing the model to miss or misclassify the object.

8.4.9. Resolution and Scaling Issues

In some frames, objects may appear smaller or blurrier due to changes in camera zoom or movement. YOLO has a lower confidence threshold for smaller or less distinct objects, leading to missed detections in these cases.

8.4.10. Noise and Artefacts

Video compression artefacts, or environmental noise (e.g., dust or rain) can degrade image quality in certain frames, making it harder for YOLO to detect and classify objects consistently.

8.4.11. Threshold Settings

YOLO uses confidence thresholds to determine whether a detected object is classified. Slight variations in the object's appearance between frames might cause the confidence score to fall below the threshold, leading to missed detections.

8.4.12. Potential Solutions

To address these issues, incorporating object tracking algorithms alongside YOLO, such as DeepSORT or ByteTrack, can help maintain consistent identification across frames by linking detections over time. Additionally, training the model with diverse datasets that simulate variations in motion, lighting, and occlusions can improve its robustness in dynamic video environments.

8.5 Comparative Analysis

Research into identifying individual cattle has grown rapidly, especially with the need for reliable livestock management, disease control, and precision farming. A variety of methods have been investigated, ranging from traditional tagging to advanced biometric and AI-driven techniques. Shown below are the main research areas considered.

8.5.1. Traditional and Semi-Digital Methods

8.5.1.1. Ear tags, branding, RFID

- Widely used in commercial farming.
- Research has focused on improving the RFID range, reducing tag loss, and integrating with herd management software.
- Tags can be lost, damaged, or tampered with.

8.5.1.2. DNA profiling

- Used in breeding and traceability studies.
- High accuracy, but expensive and impractical for day-to-day farm management.

8.5.2. Biometric Identification Research

Researchers have studied cattle biometric features for uniqueness and reliability:

8.5.2.1. Muzzle pattern recognition

- The pattern of ridges and grooves on a cow's nose is unique, like a human fingerprint.
- Studies (since the 1920s but modernised with digital imaging) show >95% accuracy using computer vision.

8.5.2.2. Facial recognition

- Machine learning and deep learning methods (CNNs, YOLO, ResNet) have been used to distinguish cattle by facial features.
- Research shows accuracy above 97% under controlled lighting, though performance drops with occlusion or poor image quality.

8.5.2.3. Retinal scanning

- Like human iris recognition.

- High accuracy but requires specialised equipment and close contact, limiting practicality.

8.5.2.4. Body patterns and coat markings

- Used especially in breeds with distinctive colour patches, like Nguni cattle.
- Research involves image-based identification from drones or cameras in barns.

8.5.3. Deep Learning and Computer Vision

Recent studies focus on non-invasive, automated systems:

8.5.3.1. YOLO, Faster R-CNN, and SSD models trained on cattle images for real-time identification

- Integration with Precision Livestock Farming (PLF) systems for automatic monitoring of feeding, health, and movement.
- Datasets: Researchers have built cattle image datasets (e.g., cattle face datasets, muzzle print databases) to train AI systems.

8.5.4. Multimodal Research

Some studies combine several methods for higher accuracy:

- *Muzzle and facial recognition* for redundancy.
- *RFID and vision systems* for cross-verification.
- *Thermal imaging* is explored for health and identification.

8.5.5. Summary of Findings:

- *Muzzle prints and facial recognition* are the most promising biometric methods.
- *Deep learning* has significantly improved accuracy, making camera-based identification feasible on farms.
- *Ongoing challenges:* varying lighting, dirt/mud on animals, cost of implementation, and scalability.

Although there are several studies on cattle identification, they are not focused on the use of portable computer equipment for cattle identification. Furthermore, they

mostly focus on recognising parts of the face or other parts or features of the cattle and not the face.

The differences between Speeded Up Robust Features (SURF), YOLOv3 and YOLOv4 lie primarily in their approaches to object detection and feature extraction, as well as their underlying architectures and applications. Shown below is a comparison of these methods:

8.5.6. Methodology of detection

SURF is a feature detection algorithm primarily used for image analysis and matching. It identifies key points in images and computes descriptors that capture the surrounding features. SURF is particularly effective for tasks involving image registration, stitching, and recognition but it is not designed for real-time object detection.

YOLOv3 is an object detection model that treats detection as a regression problem. It divides an image into a grid and predicts bounding boxes and class probabilities for objects within those grid cells. YOLOv3 is optimised for speed and can process images in real time, making it suitable for applications like surveillance and autonomous driving.

YOLOv4 builds on the strengths of YOLOv3 while incorporating several enhancements. It features a more advanced architecture, utilising techniques like Cross-Stage Partial connections (CSP), new data augmentation strategies, and better loss functions to improve detection accuracy and speed. YOLOv4 is designed to achieve state-of-the-art performance while maintaining real-time processing capabilities.

8.5.7. Architecture

The SURF algorithm relies on traditional computer vision techniques, focusing on detecting and describing local features in images. It uses a scale-space representation and applies a fast Hessian matrix for keypoint detection, followed by a descriptor that encodes local image information.

The architecture of YOLOv3 is based on a convolutional neural network (CNN) and employs several convolutional layers, batch normalisation, and Leaky ReLU activation functions. It predicts multiple bounding boxes at three different scales, allowing it to detect objects of various sizes.

YOLOv4 enhances the YOLOv3 architecture by integrating additional layers and techniques, such as the CSPDarknet backbone, which improves feature extraction efficiency. It also employs advanced techniques like Spatial Pyramid Pooling (SPP) and PANet (Path Aggregation Network) for better information flow across the network.

8.5.8. Performance

SURF is designed for robustness and speed in feature extraction rather than for detecting objects in real time. While it can effectively identify keypoints, it is generally slower than modern deep learning approaches for tasks involving object detection.

YOLOv3 strikes a balance between speed and accuracy, capable of detecting objects in real time with reasonable accuracy. It performs well across various datasets but may struggle with small object detection compared to more advanced models.

YOLOv4 improves upon YOLOv3's performance by achieving higher accuracy while maintaining fast inference speeds. It is optimised for modern hardware, making it suitable for real-time applications with complex object detection requirements.

8.5.9. Applications

SURF is commonly used in applications that require feature matching, such as image stitching, 3D reconstruction, and image retrieval. SURF is effective where local feature representation is essential.

YOLOv3 is used in various applications, including security surveillance, traffic monitoring, and autonomous vehicles. YOLOv3 is popular for scenarios requiring rapid object detection.

YOLOv4, with its enhanced capabilities, is suitable for more demanding applications such as drone monitoring, robotics, and any real-time object detection tasks that require high accuracy across diverse object classes.

In summary, SURF focuses on feature extraction, while YOLOv3 and YOLOv4 are dedicated to real-time object detection, with YOLOv4 representing an evolution of YOLOv3 that incorporates advanced techniques for improved performance. Each method serves different purposes and is best suited for specific applications in computer vision.

8.6 Challenges and Limitations

8.6.1. Challenges in Animal Recognition

Recognising animals in various settings poses unique challenges, especially in natural environments where they are not easily distinguished from their surroundings. One significant difficulty is overlapping. Animals are often partially hidden behind others or objects, making it hard to identify their boundaries or even distinguish separate individuals. This overlapping can obscure important features, like an animal's shape or colouring, that would otherwise help in recognition.

Another challenge is camouflage. Many animals have evolved colours and patterns that blend into their environments to avoid predators or hunt prey. For example, animals in wooded areas may have fur or feathers with shades of brown and green that mimic leaves, while those in snowy landscapes may be white to match the snow. This camouflage can make it nearly impossible to distinguish an animal from its surroundings without careful examination.

Varying postures also complicate recognition. Animals move in diverse ways and adopt different stances for different activities, for instance, standing, lying down,

or in motion, which changes the visible parts of their bodies and can alter their apparent shape. This variability means that a single animal might appear entirely different depending on its position, which is challenging for recognition systems that rely on identifying consistent features.

Finally, movement can make identification harder, especially when animals are fast-moving or unpredictable. A running animal, for instance, may appear as a blur, making it hard to capture enough clear visual information to identify it accurately. Moreover, movement introduces motion blur in photos or video footage, which can distort the image and obscure defining details. This can happen in low-light, slow shutter speed applications.

Each of these factors, such as overlapping, camouflage, posture variability, and movement, adds layers of complexity to recognising animals accurately, making it a task that often requires sophisticated techniques and technologies.

8.6.2. Model Limitations

YOLOv4 has several limitations, particularly when detecting smaller objects and adapting to varying lighting conditions. Its performance tends to drop with smaller objects in complex scenes because they contain fewer pixels, which reduces the network's ability to extract distinguishing features effectively. This limitation can lead to lower detection accuracy, especially when multiple small objects are close together or partially obscured.

Additionally, YOLOv4 is sensitive to changes in lighting, such as shadows, glare, or dimly lit environments. Variations in illumination can impact the model's performance since it relies on clear contrast and well-defined features for object recognition. Under low-light or high-contrast conditions, YOLOv4 might struggle to detect objects accurately or might yield false positives and negatives.

8.6.3. Dataset Limitations

When working with datasets, several common issues can impact the effectiveness of a model, including class imbalance, low-resolution images, and limited diversity.

Class imbalance occurs when some categories have significantly more samples than others, which can cause the model to become biased. As a result, it may perform well on the overrepresented classes but struggle with accuracy on underrepresented ones, leading to skewed predictions and reduced generalisability.

Low-resolution images present another challenge, as they often lack sufficient detail for the model to learn important features. When images are low in quality or pixel density, critical characteristics of objects can be blurred or lost, making it harder for the model to recognise and distinguish between different classes accurately.

A lack of diversity in the dataset can also limit the model's performance, especially in real-world applications. If the training data lacks variations in lighting, angles, backgrounds, and object appearances, the model may struggle when faced with new or diverse scenarios. This limitation can reduce the model's robustness, as it has not been exposed to the wide range of conditions it may encounter in practice.

8.6.4. Classification Challenges Due to Visual Similarities in Trained Animal Models

Figure 8.13 shows that two animals may be grouped as the same type. However, it is noteworthy that the brown animal is also identified as Animal 2 with a 63% confidence score. During model training, this animal was labelled as Animal 4. The brown animal was classified as Animal 2 because training focused only on the animals' heads, and both animals share a similar white face pattern.

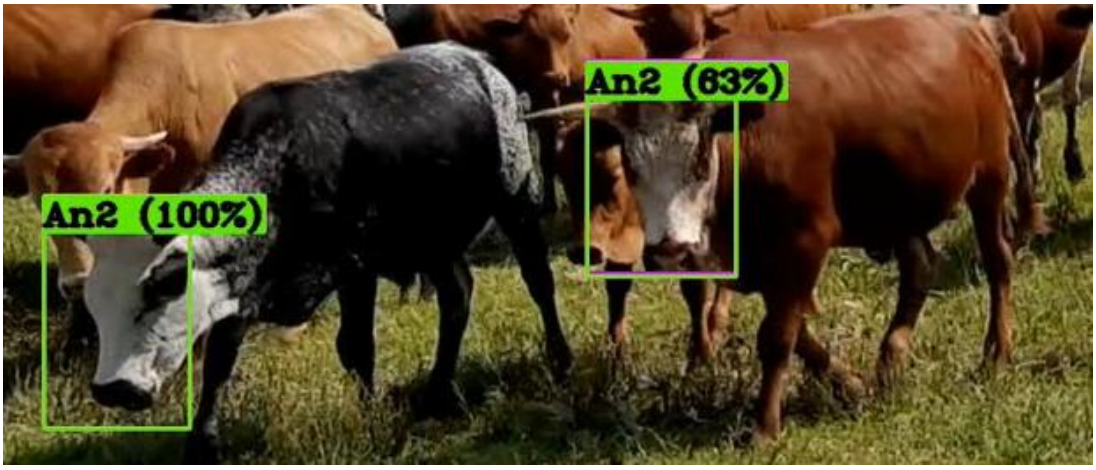


Figure 8.13: Wrong classification

Table 8-1: Confidence scores of recognised animals

Video frame	Animal	Confidence score
123	Animal 4	80.3204%
123	Animal 2	63.0183%
123	Animal 2	99.9611%

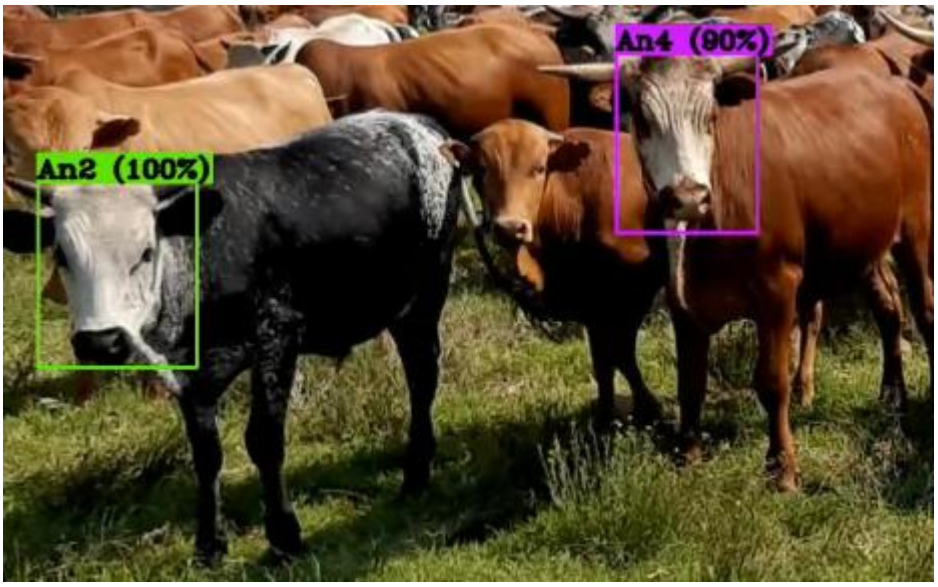


Figure 8.14: High confidence score of Animal 4

In the majority of the other video frames, Animal 4 is identified correctly.

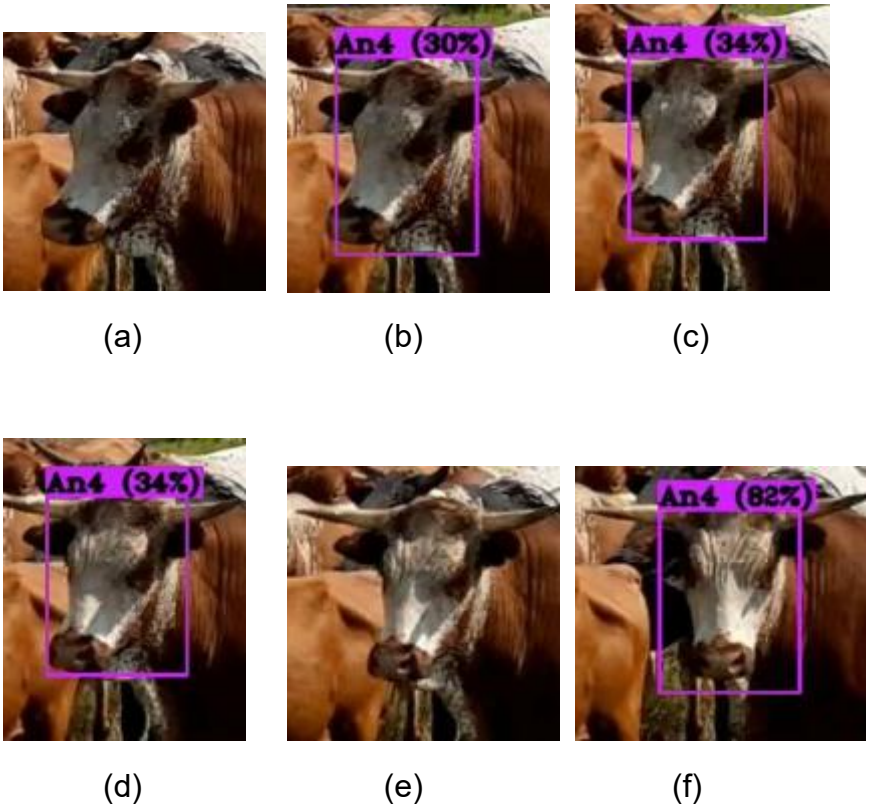


Figure 8.15: Face direction and shadows affecting confidence

One of the challenges faced when using YOLO for object detection, including animal identification, is the influence of face direction and shadows on the model's confidence levels. Variations in an animal's head orientation can lead to partial occlusions or distortions in key features, reducing the accuracy of detection and classification. Similarly, shadows caused by natural lighting conditions or environmental factors can obscure important visual details, making it harder for the model to recognise patterns or features with high confidence.

These issues arise because YOLO, like most object detection models, relies heavily on the visibility and clarity of specific features to make predictions. Changes in face direction can obscure distinctive markers such as eye placement, muzzle shape, or coat patterns, which are often critical for identification. Shadows further compound this problem by introducing variations in brightness and contrast that can be misinterpreted as noise or missing information.

To address these challenges, future work could involve incorporating advanced data augmentation techniques during training, such as random rotations, lighting adjustments, and shadow simulation. These methods can help the model become more resilient to variations in face orientation and lighting conditions. Additionally, incorporating multi-view datasets or using ensemble models that combine predictions from multiple perspectives could improve confidence and accuracy despite these visual challenges.The image with the least amount of shadow on the face has the highest confidence score. See Figure 8.14.

8.7 Improvements and Future Work

One of the first methods to improve the performance of the system is to have a larger data base. This will allow better training of the system as well as better or more accurate results, i.e. better identification of animals. Expanding the data base will not only add more general photos, but also more images of specific animals. The use of newer models like YOLOv5 or higher will increase inference time as well as accuracy.

8.8 Summary of Results

In this study, it was possible to identify individual animals from a video stream applied to a NVIDIA® Jetsan Nano. Inference times were slower than expected, but whenever animals were part of the training data, the system was successful in identifying specific animals.

CHAPTER 9: CONCLUSION

This chapter presents a comprehensive summary of the research conducted on animal recognition using YOLOv4, highlighting the key findings, their significance, and their potential applications. The study aimed to develop a deep learning-based system capable of identifying individual animals, addressing critical challenges in livestock management, wildlife monitoring, and conservation efforts. While the proposed system demonstrates promise, several limitations and operational challenges were identified, which provide insights into areas for further refinement.

9.1 YOLO timeline with speed comparisons

2015 - YOLOv1: Introduction

Release: YOLOv1 was introduced in the paper “You Only Look Once: Unified, Real-Time Object Detection” by Joseph Redmon and others [97].

- *Speed:* YOLOv1 was revolutionary for its speed, achieving real-time object detection at ~45 frames per second (FPS) on a typical GPU, which was significantly faster than traditional methods like R-CNN, which required separate region proposal and classification stages.
- *Accuracy:* It provided a trade-off between speed and accuracy, as it was faster but less accurate compared to some other models at the time.

9.1.2. 2016 - YOLOv2 (Darknet-19)

Release: YOLOv2 introduced improvements, including a new architecture and the use of higher-resolution input images [98].

- *Speed:* YOLOv2 ran at ~40 FPS, maintaining real-time performance with improved accuracy. Compared to YOLOv1, it became more accurate while retaining similar speed, showing a better trade-off between speed and performance.
- *Accuracy:* YOLOv2 achieved a notable increase in mean Average Precision (mAP) compared to YOLOv1, resulting in a substantial enhancement in overall performance.

9.1.3. 2018 - YOLOv3

Release: YOLOv3 [99] came with even better performance, using a new architecture based on residual networks (ResNet) and multi-scale predictions.

- *Speed:* YOLOv3 ran at around 30 FPS, still offering real-time performance but with a slight decrease in speed due to the more complex model.
- *Accuracy:* YOLOv3 brought a substantial boost in accuracy, especially in detecting small objects and improving mAP. Its ability to detect objects at different scales made it more versatile and accurate than previous versions.

9.1.4. 2020 - YOLOv4

Release: YOLOv4, developed by Alexey Bochkovskiy [100], introduced new features, including the use of CSPDarknet53, data augmentation techniques, and better training strategies.

- *Speed:* YOLOv4 achieved speeds of up to 60 FPS on high-end GPUs (like the RTX 2080 Ti) while maintaining high accuracy. This was a significant improvement over YOLOv3 and marked a new milestone in balancing speed and accuracy.
- *Accuracy:* The mAP increased, and YOLOv4 became one of the most optimised models, capable of high-quality detection on a wide range of hardware, including edge devices.

9.2 Future of YOLO

YOLO's future expected developments focus on making further improvements in speed and accuracy, particularly in edge device applications, autonomous systems, robotics, and augmented reality (AR). Models are expected to leverage more efficient architectures, such as transformers and neural architecture search (NAS), to continue pushing the boundaries of real-time detection.

9.2.1. Speed Comparison Summary:

Shown below is a comparison of the different frames per second (FPS) achievable with the different YOLO versions.

YOLOv1: ~45 FPS

YOLOv2: ~40 FPS

YOLOv3: ~45 FPS

YOLOv4: ~60 FPS (high-end GPUs)

YOLOv4-CSP and variants: ~60 FPS (optimised for edge devices)

YOLOv5: Up to 140 FPS (depending on model size)

YOLOv7 and v8: ~120 FPS and higher with optimisations for edge devices

Take note that the FPS varies greatly with image resolution, batch size, and GPU architecture. The values shown above apply to high-end GPUs and will not be possible with the Jetson Nano. This timeline highlights the continued improvements in both speed and accuracy, making YOLO a top choice for real-time object detection tasks across a wide range of applications.

9.3 Restate the Research Objective and Problem Statement

Research question 1: How can a deep learning vision system be designed and implemented to accurately identify individual farm animals?

This research confirms that an individual animal identification system can be successfully designed and implemented using a YOLOv4-based deep learning architecture. By training the model on distinctive visual features, such as breed-specific coat patterns, the system achieved high recognition performance for most individual classes. Deployment on a low-power embedded platform (NVIDIA® Jetson Nano) further demonstrates that reliable individual recognition is feasible under practical computational constraints. These findings indicate that, with appropriate data preparation, model optimisation, and hardware selection, deep learning vision systems can deliver accurate and deployable solutions for individual livestock identification.

Research question 2: What benefits does an AI-based identification system offer compared to traditional livestock identification methods?

The results show that an AI-driven vision system provides clear advantages over conventional identification techniques such as ear tags and branding. The proposed approach enables non-invasive, automated, and repeatable identification, thereby reducing human intervention and the risk of identification errors. Unlike physical identifiers, which can be lost or damaged, visual identification remains persistent and does not require direct animal handling. While

traditional methods remain widely used, the findings suggest that AI-based identification can complement or, in some cases, replace these methods by improving accuracy, efficiency, and long-term reliability.

Research question 3: What are the practical applications of an AI-driven vision system in livestock management?

The research demonstrates that accurate individual recognition enables several practical livestock management applications. These include targeted feeding optimisation, individual behavioural and health monitoring, and improved herd management through continuous data collection. The system also supports security-related use cases, such as theft detection and loss prevention. By moving beyond species-level detection to individual-level identification, the proposed system addresses a key limitation of existing technologies and provides a foundation for more intelligent, data-driven livestock management practices, particularly within the resource-constrained and cost-sensitive South African farming context.

In conclusion, this study demonstrates that deep learning–based vision systems can be practically applied to individual animal recognition using affordable embedded hardware. The findings support the feasibility, advantages, and applicability of AI-driven livestock identification systems, while also highlighting their potential to enhance efficiency, sustainability, and decision-making in modern livestock farming. Although further work is required to improve performance for underrepresented classes and to validate the system at larger scales, the research establishes a solid foundation for future development and real-world deployment.

9.4 Summary of Key Findings

In YOLOv4, F1 Score and mAP are key evaluation metrics used after model training to assess the performance of the object detection system. This outlines the methods and locations used for their calculation and identification. They are usually found in the training log, validation output, or the results of a test run. If

Darknet (original YOLOv4 implementation) is used, the following metrics can be seen after running:

./darknet detector map cfg/coco.data cfg/yolov4.cfg yolov4.weights

This evaluates the trained model on the validation set and prints: mAP@IoU=0.5 (COCO-style)

The results from this command is as follows:

Calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
detections_count = 81, unique_truth_count = 69

class_id = 0, name = An_1, ap = 91.67% (TP = 3, FP = 3)
class_id = 1, name = An_2, ap = 100.00% (TP = 7, FP = 0)
class_id = 2, name = An_3, ap = 100.00% (TP = 7, FP = 0)
class_id = 3, name = An_4, ap = 100.00% (TP = 7, FP = 0)
class_id = 4, name = An_5, ap = 100.00% (TP = 0, FP = 0)
class_id = 5, name = An_6, ap = 92.86% (TP = 6, FP = 1)
class_id = 6, name = An_7, ap = 100.00% (TP = 7, FP = 0)
class_id = 7, name = An_8, ap = 100.00% (TP = 2, FP = 0)
class_id = 8, name = An_9, ap = 50.00% (TP = 1, FP = 1)
class_id = 9, name = An_10, ap = 0.00% (TP = 0, FP = 0)
class_id = 10, name = An_11, ap = 100.00% (TP = 7, FP = 0)
class_id = 11, name = An_12, ap = 100.00% (TP = 4, FP = 0)
class_id = 12, name = An_13, ap = 90.48% (TP = 6, FP = 1)
class_id = 13, name = An_14, ap = 100.00% (TP = 3, FP = 1)
class_id = 14, name = An_15, ap = 100.00% (TP = 1, FP = 0)
class_id = 15, name = An_16, ap = 25.00% (TP = 1, FP = 2)
class_id = 16, name = An_17, ap = 100.00% (TP = 2, FP = 0)
class_id = 17, name = An_18, ap = 100.00% (TP = 1, FP = 1)
class_id = 18, name = An_19, ap = 100.00% (TP = 1, FP = 0)

class_id = 19, name = An_20, ap = 0.00% (TP = 0, FP = 0)
class_id = 20, name = An_21, ap = 100.00% (TP = 1, FP = 0)
class_id = 21, name = An_22, ap = 0.00% (TP = 0, FP = 0)

for conf_thresh = 0.25, precision = 0.87, recall = 0.97, F1-score = 0.92
for conf_thresh = 0.25, TP = 67, FP = 10, FN = 2, average IoU = 78.85 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.795455, or 79.55 %
Total Detection Time: 4 Seconds

Set -points flag:
`-points 101` for MS COCO
`-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

Precision is an indication of how many detected boxes are correct, while Recall shows how many actual objects were correctly detected. The F1 Score combines precision and recall, giving a balanced measure of how well the model is doing in terms of both false positives and false negatives. These are global values (across all classes). From: conf_thresh = 0.25, precision = 0.87, recall = 0.97, F1-score = 0.92, these are global values (across all classes) and calculated as follows (See equations 8.1, 8.2 and 8.3):

Precision = TP / (TP + FP)
= 67 / (67 + 10) = 0.8701

Recall = TP / (TP + FN)
= 67 / (67 + 2) = 0.9710

Where:
TP = True Positive (The model correctly detects an object (correct class and location). YOLO correctly detects a cow in the image, and the bounding box overlaps well with the ground truth (based on the IoU threshold, usually ≥ 0.5).

FP = False Positive (The model predicts an object, but it is incorrect (wrong class, wrong location, or no object exists). YOLO detects a cow in empty grass or mislabels a sheep as a cow).

FN = False Negative (The model misses detecting an actual object that is present. There is a cow in the image, but YOLO fails to detect it.)

$$F1\ Score = 2\frac{(\mathbf{Precision} \times \mathbf{Recall})}{(\mathbf{Precision} + \mathbf{Recall})}$$

$$F1\ Score = 2\frac{(0.87 \times 0.97)}{(0.87 + 0.97)} \approx 0.917$$

mAP is the mean of the average precision (AP) for each class. AP is calculated from the Precision-Recall curve, by integrating the area under the curve. In YOLOv4, mAP@0.5 typically means the IoU threshold is 0.5. COCO-style uses mAP@[0.5:0.95] with step 0.05 for stricter evaluation.

If you want to manually calculate mAP, sum all the per-class APs and divide by the number of classes:

There are 22 classes:

Sum of APs = (91.67 + 100 + 100 + 100 + 100 + 92.86 + 100 + 100 + 50 + 0 + 100 + 100 + 90.48 + 100 + 100 + 25 + 100 + 100 + 100 + 0 + 100 + 0)

= 1749.01

mAP = 1749.01 / 22 = 79.5%

Matches: mAP@0.50 = 79.55%

Table 9-1: Summary of YOLOv4 values achieved with the developed system.

Table 9-1: Summary of YOLOv4 values

Metric	Value	Description
mAP@0.5	79.55%	Overall detection accuracy across all classes
Precision	87%	Proportion of correct detections
Recall	97%	Proportion of ground truth objects found

F1 Score	92%	Harmonic mean of precision and recall
Avg IoU	78.85%	Average overlap between prediction and ground truth

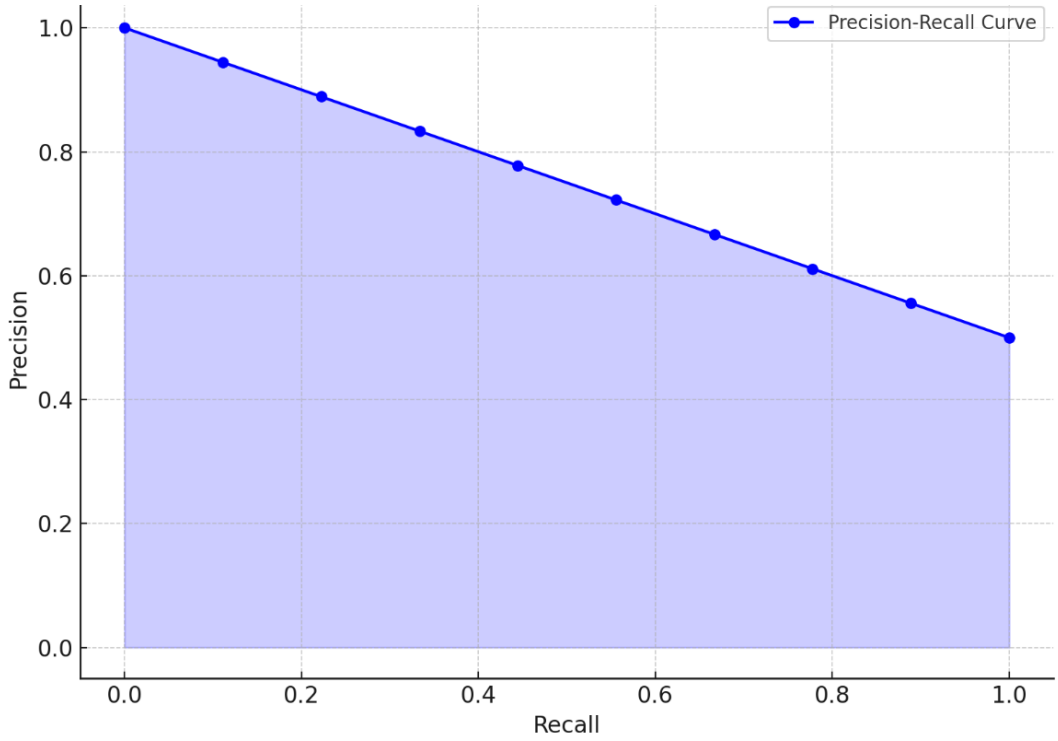


Figure 9.1: Precision-Recall curve

Figure 9.1 shows the Precision-Recall curve based on global values from the YOLOv4 research evaluation. It visually shows how precision decreases as recall increases. This is a common pattern in object detection models. This curve helps to illustrate the trade-off between catching all objects (recall) and avoiding false positives (precision).

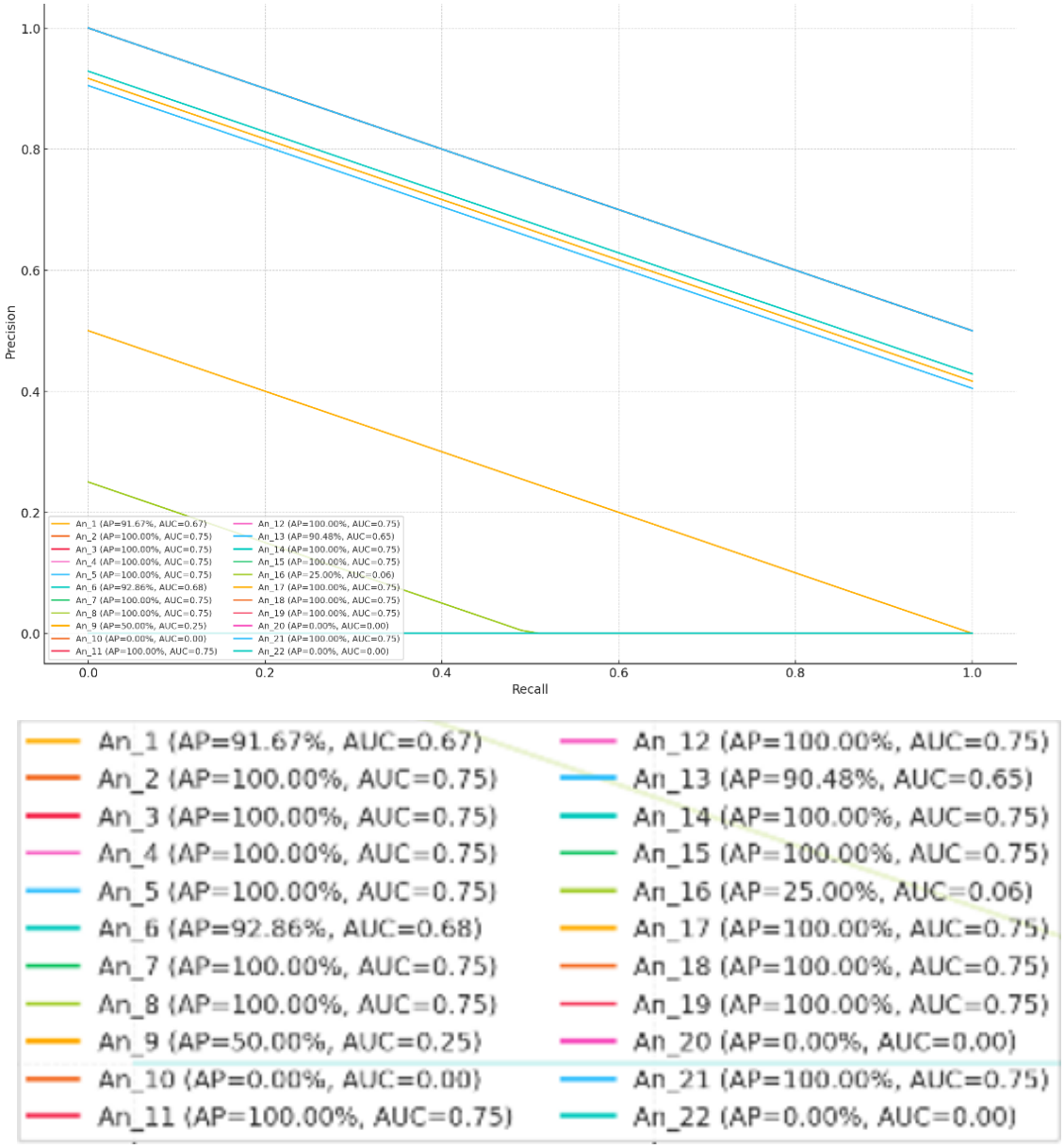


Figure 9.2: Class-wise Precision-Recall curves

Figure 9.2 shows the class-wise Precision-Recall curves, each simulated to match the shape of the AP (Average Precision) values from the YOLOv4 model.
The AUC (Area Under the Curve) is shown for each class in the legend. Classes with high AP (e.g., An_2 to An_14) have wide, flat curves — indicating high precision across a range of recall. Classes like An_9, An_15, and An_16 have lower AUCs, reflecting weaker detection performance.

Table 9-2 shows the **Average Precision (AP)** and **AUC (Area Under Curve)** values per class from the YOLOv4 development model evaluation.

Table 9-2: AP and AUC

Class Name	AP (%)	AUC (PR Curve)
An_12	100	0.75
An_17	100	0.75
An_3	100	0.75
An_4	100	0.75
An_5	100	0.75
An_7	100	0.75
An_8	100	0.75
An_21	100	0.75
An_19	100	0.75
An_11	100	0.75
An_2	100	0.75
An_18	100	0.75
An_14	100	0.75
An_15	100	0.75
An_6	92.86	0.6786
An_1	91.67	0.6667
An_13	90.48	0.6548
An_9	50	0.25
An_16	25	0.0625
An_10	0	0
An_20	0	0
An_22	0	0

9.5 Discussion of Research Contributions

Currently, research on vision systems focuses on the identification of species and not the identification of individual animals. The original contribution of this study was the development of an automatic, artificial intelligence deep neural network vision system for the identification of individual animals, such as cattle and large wild animals [18].

While the study did not specifically address the identification of stolen livestock, its findings could still prove beneficial in such scenarios. Projects such as “Snapshot Serengeti” [20], which achieved great success in the identification of different wildlife species in Africa, suggest that it is possible to achieve good results using neural networks. This project was driven by volunteers who helped with the

labelling and identification of animals in the images. The dataset for this project consisted of about 6.7 million images [21].

9.6 Implications of Findings

When YOLOv4 was used with the Jetson Nano, it was possible to identify individual animals. A proof-of-concept project was also developed to demonstrate that it is possible to use the Jetson Nano with an Arduino interface to develop an automatic feeding system. Although the full development was not part of this study, possible cost savings for the farmer can come from such a fully developed feeding system. The system will also be able to be expanded to monitor cattle for other applications, such as theft detection.

9.7 Limitations of the Study

This study aimed to develop a deep neural network for the identification of individual animals using YOLOv4 [18]. However, several limitations are acknowledged. One key challenge was the performance trade-offs of YOLOv4 when detecting smaller or partially obscured animals (see Figure 8.9), which may impact accuracy. Dataset limitations, such as a lack of diversity and imbalance in animal classes, further constrain the system's ability to generalize effectively to broader contexts.

Hardware limitations also posed challenges, particularly in achieving real-time detection in field conditions where computational resources may be restricted. Additionally, the system may have encountered overfitting issues, which could have reduced its effectiveness when applied to new species or unfamiliar environments. Operational challenges, including data collection methods and the durability of equipment exposed to environmental elements or animal interference, added to the complexity.

The training dataset was limited to 20 animals, which narrowed the scope of the study and could affect the system's scalability and generalisability. While the tool was not specifically designed for identifying stolen livestock, its potential utility in such scenarios is acknowledged. Despite these limitations, the study drew

inspiration from successful projects like Snapshot Serengeti, which highlights the promise of neural networks in animal identification.

9.8 Suggestions for Future Research

Future work may include areas such as exploring newer versions of YOLO (like YOLOv5, YOLOv7 or newer). Other deep learning models that might improve detection accuracy or speed can also be investigated. Methods for gathering a more diverse dataset and addressing class imbalances to improve model robustness across environments could be developed. Further research can include the implementation of post-processing techniques or hybrid models that combine YOLO with other methods (e.g., tracking algorithms for better identification). Building on the findings of this study, several areas of future research can be explored to enhance the capabilities and applicability of AI-driven animal recognition systems:

9.8.1. Exploration of Advanced Deep Learning Models

Future research could focus on leveraging newer versions of YOLO, such as YOLOv5, YOLOv7, or subsequent iterations, to improve detection accuracy and computational efficiency. These models often incorporate optimisations that can address challenges in recognising smaller or occluded animals, thereby enhancing performance. Additionally, experimenting with other state-of-the-art models, such as EfficientDet or transformer-based architectures like Vision Transformers (ViTs), could yield further improvements in precision and scalability.

9.8.2. Diversification and Expansion of Datasets

A key limitation in this study was the lack of diversity in the dataset, which affected the model's robustness. Future research should prioritise the collection of more comprehensive datasets, representing a wider variety of species, breeds, environments, and conditions. Addressing class imbalances and incorporating data augmentation techniques can further improve the model's ability to generalise to new contexts.

9.8.3. Integration of Post-Processing Techniques

Combining YOLO with complementary algorithms, such as object tracking systems (e.g., DeepSORT or ByteTrack), can improve the accuracy of individual animal identification over time. These hybrid models can reduce false positives and provide additional insights, such as movement patterns or group interactions, by maintaining consistent identification across frames.

9.8.4. Behaviour Analysis Extensions

Expanding the system to recognise specific animal behaviours, such as feeding, grooming, or hunting, could provide valuable data for applications in wildlife conservation and farm management. Behaviour analysis would require a more nuanced dataset annotated with activity labels and the development of specialised models tailored to multi-task learning.

9.8.5. Adapting for Edge Devices and Field Deployment

To facilitate practical applications, future studies should focus on optimising the model for deployment on edge devices such as drones, Raspberry Pi systems, or the NVIDIA® Jetson series. These platforms enable real-time processing in field conditions, offering portability and efficiency. Developing lightweight models with reduced computational requirements can ensure compatibility with such devices without compromising accuracy.

9.8.6. Cross-Species Generalisation

Extending the system's capabilities to identify not only individual animals within a species but also across different species would broaden its usability. This could involve multi-species datasets and the design of models capable of hierarchical classification, recognising both species-level and individual-level distinctions.

9.8.7. Ethical and Privacy Considerations

Future research should also explore the ethical implications of using AI for animal monitoring, particularly in the context of wildlife conservation. Ensuring that data collection and usage respect the habitats and well-being of animals is crucial for responsible implementation.

By addressing these areas, future research can overcome existing limitations, expand the range of applications, and further refine the integration of AI technologies in animal recognition and monitoring.

9.9 Final Remarks

If further advanced, the research has the potential to become a valuable resource for farmers in managing their livestock effectively. It could facilitate more efficient livestock management and reduce costs by minimising feed waste for each animal. Although there were a few exceptions, the developed YOLO system proved to be reliable in detecting the animals. Another advantage of the YOLO system is that it can be successfully implemented on a small computer like the Jetson Nano. Further work with newer YOLO versions could improve accuracy and detection speed.

References

[1] J. Wäldchen and P. Mäder, “Machine learning for image based species identification,” Methods in Ecology and Evolution, vol. 9, no. 11, pp. 2216 - 2225, 2018.

[2] “Species Detection With Machine Learning: Simplifying Efforts for Conservation Organizations,” Gramener, 21 09 2021. [Online]. Available: https://gramener.medium.com/machine-learning-species-detection-3d97b3d1f65e. [Accessed 29 09 2024].

[3] J. Solawetz, “How to Train Scaled-YOLOv4 to Detect Custom Objects,” 12 2020. [Online]. Available: https://blog.roboflow.com/how-to-train-scaled-yolov4/. [Accessed 29 09 2024].

[4] Keylabs, “YOLOv8 vs SSD: Choosing the Right Object Detection Model,” 22 December 2023. [Online]. Available: https://keylabs.ai/blog/yolov8-vs-ssd-choosing-the-right-object-detection-model/. [Accessed 13 June 2025].

[5] Z. Keita, “YOLO Object Detection Explained,” Datacamp, 28 09 2024. [Online]. Available: https://www.datacamp.com/blog/yolo-object-detection-explained. [Accessed 23 01 2026].

[6] H. A. Ahmed, S. M. Md Ibrahim, M. S. Kumar and S. S. Ahmed , “Yolo-Based Fast and Accurate Object Detection for Real-Time Applications.,” International Journal of Intelligent Systems and Applications in Engineering (IJISAE), vol. 12, no. 23S, 2024.

[7] J. Wang, T. Zhang, Y. Cheng and N. Al-Nabhan, “Deep Learning for Object Detection: A Survey,” *Computer Systems Science & Engineering,* vol. 38, no. 2, 2021.

[8] “Real-Time Object Detection: YOLO’s Role in AI-Driven Applications,” 15 01 2025. [Online]. Available: https://www.linkedin.com/pulse/real-time-object-detection-yolos-role-ai-driven-applications-lgi5e.

[9] “What Makes YOLO the Fastest Eye in AI?,” MeisterIT systems, 23 07 25. [Online]. Available: https://meisteritsystems.com/news/yolo-fastest-object-detection-ai-2025/. [Accessed 02 01 2026].

[10] S. Taoua, “The Evolution and Applications of YOLO Object Detection: A Comprehensive Exploration,” Medium, 17 01 2024. [Online]. Available: https://medium.com/ubiai-nlp/the-evolution-and-applications-of-yolo-object-detection-a-comprehensive-exploration-9b8dbc0ef2a5. [Accessed 03 01 2026].

[11] “What Does YOLO Really Mean in 2024 ?,” ubiAI, 16 11 2023. [Online]. Available: https://ubiai.tools/what-does-yolo-really-mean/. [Accessed 22 01 2026].

[12] South African Society for Animal Science (SASAS), “About Animal Science,” [Online]. Available: https://www.sasas.co.za/about-animal-sciences/. [Accessed 10 10 2019].

[13] H. H. Meissner, M. M. Scholtz and A. R. Palmer, “Sustainability of the South African Livestock Sector towards 2050 Part 1: Worth and impact of the sector,” *South African Journal of Animal Science ,* vol. 43, no. 3, pp. 282-297, 2013.

[14] D. Coffey, K. Dawson, P. Ferket and A. Connolly, “Review of the feed industry from a historical perspective and implications for its future,” *Journal of Applied Animal Nutrition,* vol. 4, no. 3, pp. 1-11, 2015.

[15] W. M. Hilton, “Nutritional Requirements of Beef Cattle,” in *Merck Veterinary Manual*, Merck, 2014.

[16] A. Pezzuolo, A. Chiumenti, L. Sartori and F. Da Borso, “AUTOMATIC FEEDING SYSTEM: EVALUATION OF ENERGY CONSUMPTION AND LABOUR REQUIREMENT IN NORTH-EAST ITALY DAIRY FARM,” in *ENGINEERING FOR RURAL DEVELOPMENT*, Jelgava, 2016.

[17] M. Zuidhof, “Precision livestock feeding: matching nutrient supply with nutrient requirements of individual animals,” *Journal of Applied Poultry Research,* vol. 29, no. 1, pp. 11-14, 2020.

[18] P. S. Veldtsman and B. J. Kotze, “Development of an Artificial Intelligence Deep Neural Network for the Identification of Individual Animals,” *Tuijin Jishu / Journal of Propulsion Technology ,* vol. 44, no. 3, pp. 67 - 76, 2023.

[19] AnimalSmart.org, “What are animals fed and why?,” [Online]. Available: https://animalsmart.org/animal-health-animal-welfare/animal-feed. [Accessed 21 10 2019].

[20] Agriseta, Learner Guide Primary Agriculture Animal Feeding Procedures, Pretoria: Agriseta, 2006.

[21] T. M. Brown-Brandl and R. A. Eigenberg, “Development of a Livestock Feedingbehaviour Monitoring System,” in *Transactions of the American Society of Agricultural and Biological Engineers*, 2011.

[22] A. Grothmann, F. Nydegger, C. Moritz and C. Bisaglia, “Automatic feeding systems for dairy cattle – potential for optimization in dairy farming,” in *International Conference on Agricultural Engineering - AgEng 2010: Towards Envirmonmental Technologies*, Cemagref, Clermont-Ferrand, 2010.

[23] South Africa Online (Pty), Ltd, “Predation Management in Livestock Farming”.

[24] The Department of Agriculture, Forestry and Fisheries, “Draft Document For A Livestock Identification And Traceability System South Africa (LITS SA),” Pretoria, 2018.

[25] W. Clack, “The Extent of Stock Theft in South Africa,” *Acta Criminologica: Southern African Journal of Criminology,* vol. 26, no. 2, pp. 77-91, 2013.

[26] Department: Agriculture and Rural Development: Province of Kwazulu-Natal, “Beef Production: The Basics,” [Online]. Available: https://www.kzndard.gov.za/images/Documents/RESOURCE_CENTRE/GUIDELINE_DOCUMENTS/PRODUCTION_GUIDELINES/Beef_Production/Cattle%20Identification.pdf. [Accessed 21 10 2019].

[27] X. Yu, J. Wang, R. Kays, . P. A. Jansen, T. Wang and T. Huang, “Automated identification of animal species incamera trap images,” *EURASIP Journal on Image and Video Processin,* 2013.

[28] E. Matchar , “AI Plant and Animal Identification Helps Us All Be Citizen Scientists,” *Smithsonium Magazine,* 7 June 2017.

[29] A. Taha, A. Darwish, A. E. Darwish and A. ElKholy, “Arabian Horse Identification and Gender Determination System based on Feature Fusion

and Gray Wolf Optimization," *International Journal of Intelligent Engineering & Systems,* vol. 13, no. 4, pp. 145-155, 2020.

[30] H. Nguyen, S. J. Maclagan, T. D. Nguyen, T. Nguyen, P. Flemons, K. Andrews, E. G. Ritchie and D. Phung, "Animal Recognition and Identification with Deep Convolutional Neural Networks for Automated Wildlife Monitoring," in *International Conference on Data Science and Advanced Analytics (DSAA)*, Tokyo, Japan, 2017.

[31] S. Yukun, H. Pengju, W. Yujie, D. Baisheng, L. Runze and z. Yonggen, "Automatic monitoring system for individual dairy cows based on a deep learning framework that provides identification via body parts and estimation of body condition score," *Journal of Dairy Science,* vol. 102, no. 11, pp. 10140-10151, 01 November 2019.

[32] A. Salama, A. E. Hassaniem and A. Fahmy, "The Significance of Artificial Intelligence in Arabian Horses Identification System," in *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2019*, Cairo, Egypt, 2019.

[33] I. M. Nasser and S. S. Abu-Naser, "Artificial Neural Network for Predicting Animals Category," *International Journal of Academic and Applied Research (IJAAR),* vol. 3, no. 2, pp. 18-24, February 2019.

[34] Z. Ünal, "Smart Farming Becomes Even Smarter With Deep Learning - A Bibliographical Analysis," *IEEE Access,* vol. 8, pp. 105587-105609, 4 June 2020.

[35] A. Rivas, P. Chamoso, A. González-Briones and J. Corchado, "Detection of Cattle Using Drones and Convolutional Neural Networks," *Sensors (Basel),* vol. 18, no. 7, 27 June 2018.

[36] I. Goodfellow, Y. Bengio and A. Courville, in *Deep Learning*, Lodon, UK, MIT Press, 2016.

[37] R. Szeliski, in *Computer Vision: Algorythms and Applications*, London, UK, Springer, 2022.

[38] N. Sharma, A. Rani and V. Sharma, "From classical techniques to convolution-based models: A review of object detection algorithms," arXiv, 2024.

[39] J. O'Mahony, P. Krishnan and S. Das, “Deep learning vs. traditional computer vision: Comparative analysis and applications,” arXiv, 2019.

[40] G. Yeli, “Computer vision vs. image processing: A comparative overview,” 2023. [Online]. Available: https://medium.com/@gerayeli.kourosh/computer-vision-vs-image-processing-a-comparative-overview-7167a2b62fdb. [Accessed 12 01 2026].

[41] “Difference between traditional computer vision techniques and deep learning-based approaches,” GeeksforGeeks, 01 2023. [Online]. Available: https://www.geeksforgeeks.org/computer-vision/difference-between-traditional-computer-vision-techniques-and-deep-learning-based-approaches. [Accessed 12 1 2026].

[42] “Deep learning vs. traditional computer vision methods: Advantages and limitations,” Technolynx, 02 2023. [Online]. Available: https://www.technolynx.com/post/deep-learning-vs-traditional-computer-vision-methods. [Accessed 7 01 2026].

[43] R. Patel and S. Shah, “Classical and deep learning approaches in computer vision: A comparative review,” *International Journal of Artificial Intelligence,* vol. 19, no. 4, pp. 210 - 225, 2022.

[44] A. Opperman and B. Whitfield, “What Is Deep Learning and How Does It Work?,” 12 December 2023. [Online]. Available: https://builtin.com/machine-learning/deep-learning. [Accessed 13 June 2025].

[45] A. Ng, “CS230 Deep Learning,” Stanford University, 2017. [Online]. Available: https://cs230.stanford.edu/files/. [Accessed 13 June 2025].

[46] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and L. Zitnick, “Microsoft COCO: Common Objects in Context,” in *European Conference on Computer Vision – ECCV 2014*, Zurich, Switzerland, 2014.

[47] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and F.-F. Li, “ImageNet: a Large-Scale Hierarchical Image Database,” in *IEEE Computer Society*

Conference on Computer Vision and Pattern Recognition, Miami, Florida, USA, 2009.

[48] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature,* vol. 521, no. 28 May 2015, pp. 436-444, 2015.

[49] H. F. LTD, "Decoding the dichotomy: Traditional Image Processing vs. Deep Learning," *Artificial Intelligence for Industry Europe,* 25 August 2020.

[50] HCL Fasteners Ltd, "Imaging & Machine Vision Europe," *IEN Europe Magazine,* 25 August 2020.

[51] V. Schoor, "Van Schoor," 23 January 2020. [Online]. Available: https://www.vanschoorgate.co.za/af/watter-tipe-veevoer-voerder-moet-ek-kies-vir-beeste-of-skape/.

[52] "Bekostigbare selfvoerder maak lewe maklik," Landbouweekblaqd, 07 05 2008. [Online]. Available: https://www.netwerk24.com/landbou/Bedrywe/Meganisasie/bekostigbare-selfvoerder-maak-lewe-maklik-20170914. [Accessed 2024 09 2024].

[53] J. P. Mueller and L. Massaron, Artificial Intelligence for Dummies, New Jersey: John Wiley & Sons, 2018.

[54] Y. Serpa, "A Glossary of Neural Network Terms, A short tour through Deep Learning jargon," 8 July 2020. [Online]. Available: https://towardsdatascience.com/the-beginners-glossary-of-neural-network-terms-a9617354078. [Accessed 20 August 2021].

[55] S.-S. Shai and B.-D. Shai, Understanding Machine Learning From Theory to Algorithms, New York: Cambridge University Press, 2014.

[56] P. Kim, MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence, Seoul: Apress Media, 2017.

[57] P. DeBeasi, "Training vs Inference," Gartner Research, 23 January 2019. [Online]. Available: https://blogs.gartner.com/paul-debeasi/2019/02/14/training-versus-inference/. [Accessed 2020 August 2020].

[58] P. D. Carlton Sapp, "Architecting Machine Learning With IoT," Gartner Research, 23 January 2019. [Online]. Available:

https://www.gartner.com/en/documents/3898877. [Accessed 12 August 2020].

[59] I. Bangash, “NVIDIA Jetson Nano vs Google Coral vs Intel NCS: A Comparison,” [Online]. Available: https://towardsdatascience.com/nvidia-jetson-nano-vs-google-coral-vs-intel-ncs-a-comparison-9f950ee88f0d. [Accessed 12 August 2020].

[60] NVIDIA Developer, “Jetson Nano: Deep Learning Inference Benchmarks,” [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks. [Accessed 12 August 2020].

[61] E. Wu, “NVIDIA Jetson Nano and Jetson Xavier NX Comparison: Specifications, Benchmarking, Container Demos, and Custom Model Inference,” [Online]. Available: https://www.seeedstudio.com/blog/2020/06/04/nvidia-jetson-nano-and-jetson-xavier-nx-comparison-specifications-benchmarking-container-demos-and-custom-model-inference/. [Accessed 12 August 2020].

[62] D. Gupta, “25 Must Know Terms & concepts for Beginners in Deep Learning,” 21 May 2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/05/25-must-know-terms-concepts-for-beginners-in-deep-learning/. [Accessed 2021 August 20].

[63] “Freepik,” [Online]. Available: https://www.freepik.com/free-photos-vectors/neuron. [Accessed 29 09 2024].

[64] S. Krishnan, “How do determine the number of layers and neurons in the hidden layer?,” Medium, 08 09 2021. [Online]. Available: https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3. [Accessed 29 09 2024].

[65] P. Sharma, “Keras Dense Layer Explained for Beginners,” Making AI Simple, 20 10 2020. [Online]. Available: https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/. [Accessed 29 09 2024].

[66] Y. Verma, “Dense Layers,” 01 08 2024. [Online]. Available: https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/. [Accessed 29 09 2024].

[67] dishashree26, “25 Must Know Terms & concepts for Beginners in Deep Learning,” Analytics Vidhya, 25 06 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/05/25-must-know-terms-concepts-for-beginners-in-deep-learning/. [Accessed 29 09 2024].

[68] “Basic Concepts in Deep Learning,” New World Artificial Intelligence, 1 12 2022. [Online]. Available: https://www.newworldai.com/basic-concepts-deep-learning/. [Accessed 29 09 2024].

[69] Bharati DW Consultancy , 15 06 2019. [Online]. Available: https://www.youtube.com/watch?v=ye_EaPqEj00. [Accessed 29 09 2024].

[70] pawangfg, “YOLO : You Only Look Once – Real Time Object Detection,” 15 06 2022. [Online]. Available: https://www.geeksforgeeks.org/yolo-you-only-look-once-real-time-object-detection/. [Accessed 16 08 2024].

[71] B. Сичкар, “Introduction into YOLO v3,” 21 06 2020. [Online]. Available: https://www.youtube.com/watch?v=vRqSO6RsptU&t=51s. [Accessed 25 09 2024].

[72] V. Sichkar, “Train YOLO for Object Detection with Custom Data,” 11 2023. [Online]. Available: https://www.udemy.com/course/training-yolo-v3-for-objects-detection-with-custom-data/?referralCode=A283956A57327E37DDAD&couponCode=ST6MT103124. [Accessed 25 09 2024].

[73] A. Ojha, “You Only Look Once - YOLO: Object Detection using Convolutional Neural Networks,” 08 04 2020. [Online]. Available: https://www.youtube.com/watch?v=yDceQvxykq4. [Accessed 29 09 2024].

[74] A. Ojha, “You Only Look Once: Object Detection Algorithm Part 2,” 08 04 2020. [Online]. Available: https://www.youtube.com/watch?v=uvvRysktPEU. [Accessed 29 09 2024].

[75] “YOLO Object Detection Explained,” Datacamp, 28 09 2024. [Online]. Available: https://www.datacamp.com/blog/yolo-object-detection-explained. [Accessed 29 09 2024].

[76] M. Nagpal, “The Ultimate Guide to YOLOv3 Architecture,” ProjectPro, 21 03 2024. [Online]. Available: https://www.projectpro.io/article/yolov3-architecture/836. [Accessed 29 09 2024].

[77] B. Сичкар, “Introduction into YOLO v3,” 21 06 2020. [Online]. Available: https://www.youtube.com/watch?v=vRqSO6RsptU&t=51s. [Accessed 29 09 2024].

[78] A. Bhatt, “Understanding Object detection with YOLO,” Medium, 1 August 2024. [Online]. Available: https://medium.com/@anilaknb/understanding-object-detection-with-yolo-4b658e1efbab. [Accessed 05 August 2025].

[79] M. Nagpal , “The Ultimate Guide to YOLOv3 Architecture,” 21 03 2024. [Online]. Available: https://www.projectpro.io/article/yolov3-architecture/836. [Accessed 29 09 2024].

[80] Education Ecosystem (LEDU), “Understanding K-means Clustering in Machine Learning,” Towards Data Science, 12 09 2018. [Online]. Available: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1. [Accessed 29 09 2024].

[81] A. Singh, “Selecting the Right Bounding Box Using Non-Max Suppression (with implementation),” Analytics Vidhya, 20 02 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/. [Accessed 28 09 2024].

[82] Ultralytics, 12 November 2023. [Online]. Available: https://docs.ultralytics.com/models/yolov4/. [Accessed 13 June 2025].

[83] S. Islam, “Inverse.AI,” 8 December 2020. [Online]. Available: https://inverseai.com/blog/yolov3-object-detection. [Accessed 20 06 2024].

[84] K. Z. X. R. S. S. J. He, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” in *Computer Vision -- ECCV 2014*, Zurich, Switzerland,, 2014.

[85] R. Kundu, “YOLO: Algorithm for Object Detection Explained [+Examples],” V7 Labs, 17 January 2023. [Online]. Available: https://www.v7labs.com/blog/yolo-object-detection. [Accessed 05 August 2025].

[86] Y. Wang, “Facial-Recognition Software Meets Its Match: Barnyard Animals,” The Wall Street Journal, 30 04 2019. [Online]. Available:

https://www.wsj.com/articles/facial-recognition-software-meets-its-match-barnyard-animals-11556633879. [Accessed 29 09 2024].

[87] K. Creutzinger, “Merck Veterinary Manual,,” Merck & Co., Inc., 06 2025. [Online]. Available: https://www.msdvetmanual.com/behavior/behavior-of-production-animals/behavior-of-cattle. [Accessed 22 01 2026].

[88] A. C. T. van Staaveren, E. N. M. Kearney and L. Shalloo, “Human–animal interactions, animal behaviour and welfare in livestock systems,” *Frontiers in Animal Science,* vol. 2, pp. 1-12, 2021.

[89] D. Mancuso, G. Castagnolo and S. M. C. Porto, “Cow Behavioural Activities in Extensive Farms: Challenges of,” *Sensors,* vol. 23, no. 8, 2023.

[90] A. Kathuria, “What’s new in YOLO v3?,” Towards Data Science, 23 04 2018. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b. [Accessed 29 09 2024].

[91] J. Brown, “How to Perform Object Detection With YOLOv3 in Keras,” Machine Learning Mastery, 8 10 2019. [Online]. Available: https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/. [Accessed 29 09 2024].

[92] M. Hollemans, “Real-time object detection with YOLO,” 20 05 2017. [Online]. Available: https://machinethink.net/blog/object-detection-with-yolo/. [Accessed 29 09 2024].

[93] A. Bochkovskiy, “Darknet,” [Online]. Available: https://github.com/AlexeyAB/darknet. [Accessed 29 09 2024].

[94] NVIDIA, “Jetson Nano Data Sheet,” [Online]. Available: https://d29g4g2dyqv443.cloudfront.net/sites/default/files/akamai/embedded/images/docs/jetson/641393088.png. [Accessed 05 August 2025].

[95] NVIDIA, “Jetson Nano Data Sheet Header Layout,” NVIDIA, [Online]. Available: https://d29g4g2dyqv443.cloudfront.net/sites/default/files/akamai/embedded/images/docs/jetson/646984561.png. [Accessed 05 August 2025].

[96] Maker, “Maker Uno Data Sheet,” Maker, [Online]. Available: https://www.robofactory.co.za/img/cms/Maker%20Uno/5.jpeg. [Accessed 05 August 2025].

[97] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, 2016. [Online]. Available: https://pjreddie.com/static/papers/yolo_1.pdf. [Accessed 30 August 2020].

[98] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” 25 December 2016. [Online]. Available: https://pjreddie.com/static/papers/YOLO9000.pdf. [Accessed 21 February 2021].

[99] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” 8 April 2018. [Online]. Available: https://pjreddie.com/static/papers/YOLOv3.pdf. [Accessed 22 February 2021].

[100] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” 23 April 2020. [Online]. Available: https://arxiv.org/pdf/2004.10934. [Accessed 12 March 2021].

[101] M. L. Westendorf and C. A. Williams, “Nutrient Management on Livestock Farms: Tips for Feeding,” New Brunswick, 2007.

[102] Snapshot Serengeti, “Snapshot Serengeti,” [Online]. Available: https://www.zooniverse.org/projects/zooniverse/snapshot-serengeti. [Accessed 12 February 2020].

[103] Labeled Information Library of Alexandria: Biology and Conservation, “Snapshot Serengeti,” [Online]. Available: http://lila.science/datasets/snapshot-serengeti. [Accessed 12 February 2020].

Appendix 1: Load reference image, and compute surf features (Matlab®)

```
clear all; close all; clc;

%% Load reference image, and compute surf features
%%ref_img = imread('MWqueen_crop_small.bmp');
ref_img = imread('BeesCr.jpg');
ref_img_gray = rgb2gray(ref_img);
ref_pts = detectSURFFeatures(ref_img_gray);
[ref_features,  ref_validPts] =
extractFeatures(ref_img_gray,  ref_pts);

figure('Name','Reference Image'); imshow(ref_img);
hold on; plot(ref_pts.selectStrongest(50));
% Visual 25 SURF features
figure;
subplot(5,5,3); title('First 25 Features');
for i=1:25
    scale = ref_pts(i).Scale;
    image = imcrop(ref_img,[ref_pts(i).Location-10*scale
20*scale 20*scale]);
    subplot(5,5,i);
    imshow(image);
    hold on;
    rectangle('Position',[5*scale 5*scale 10*scale
10*scale],'Curvature',1,'EdgeColor','g');
end
% Compare to video frame
%image = imread('MWsample_full.png');
image = imread('Bees_4.jpg');
I = rgb2gray(image);

% Detect features
I_pts = detectSURFFeatures(I);
[I_features, I_validPts] = extractFeatures(I, I_pts);
figure;imshow(image);
hold on; plot(I_pts.selectStrongest(50));
```

```
% Compare card image to video frame
index_pairs = matchFeatures(ref_features, I_features);

ref_matched_pts =
ref_validPts(index_pairs(:,1)).Location;
I_matched_pts = I_validPts(index_pairs(:,2)).Location;

figure, showMatchedFeatures(image, ref_img,
I_matched_pts, ref_matched_pts, 'montage');
title('Showing all matches');

% Define Geometric Transformation Objects
gte = vision.GeometricTransformEstimator;
gte.Method = 'Random Sample Consensus (RANSAC)';

[tform_matrix, inlierIdx] = step(gte, ref_matched_pts,
I_matched_pts);

ref_inlier_pts = ref_matched_pts(inlierIdx,:);
I_inlier_pts = I_matched_pts(inlierIdx,:);

% Draw the lines to matched points
figure;showMatchedFeatures(image, ref_img, I_inlier_pts,
ref_inlier_pts, 'montage');
title('Showing match only with Inliers');

% Transform the corner points
% This will show where the object is located in the
image
 tform = maketform('affine',double(tform_matrix));
[width, height,~] = size(ref_img);
corners = [0,0;height,0;height,width;0,width];
new_corners = tformfwd(tform,
corners(:,1),corners(:,2));
figure;imshow(image);
patch(new_corners(:,1),new_corners(:,2),[0 1
0],'FaceAlpha',0.5);
```