APPLICATION FOR COW LAMENESS DETECTION USING COMPUTER VISION

BY

KONG ZI LIN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere thanks and appreciation to my supervisor, Dr. Ng Hui Fuang who has given me this bright opportunity to engage in this meaningful project of a field I am passionate about. A million thanks to you.

Also, I must say thanks to my parents, my family and friends for their love, support, and continuous encouragement throughout the course. This project would not have been possible without all the love and support I have received from them. On top of that, special thanks to cow T048 from Farm Fresh Farm for letting me to work with her for a few hours. Above all, I give thanks to God, to whom be all the glory.

ABSTRACT

Lameness is a major welfare and economic issue in dairy farming, causing reduced milk yield, reproductive failure, and premature culling. Small and medium-sized farms often lack affordable tools for early detection, leading to delayed treatment. This project develops a cost-effective mobile application that uses computer vision to detect lameness in cows and provides basic herd management functions. The lameness detection model applies YOLOv8-based pose estimation to extract anatomical key points and calculate back arching through Root Mean Squared Error (RMSE), identifying abnormal postures linked to lameness. A cow recognition system, based on ResNet18 and ArcFace embeddings, allows farmers to register cows by coat patterns and maintain health records. The system is supported by a FastAPI backend with Firebase integration for storage, authentication, and data management. Through a simple Flutter-based mobile app interface, farmers can upload images, receive predictions, confirm cow identities, and update records. This approach enables early diagnosis, improves animal welfare, and reduces economic losses.

Area of Study: Computer Vision, Mobile Application Development

Keywords: Cow Lameness, Biosensing, Computer Vision, Machine Learning, Mobile Application Development, YOLOv8, ResNet18

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AHDB	Agriculture and Horticulture Development Board
API	Application Programming Interface
BCRC	Beef Cattle Research Council
CLAHE	Contrast Limited Adaptive Histogram Equalization
CNNs	Convolutional Neural Networks
CV	Computer vision
CVS	Computer Vision Systems
CRUD	Create, Read, Update, Delete
EER	Equal Error Rate
GPU	Graphic Processing Unit
P	Precision
PCA	Principal Component Analysis
R	Recall
RMSE	Root Mean Squared Error
RNNs	Recurrent Neural Networks
t-SNE	t-distributed Stochastic Neighbor Embedding
YOLO	You Only Look Once

Chapter 1
Introduction

1.1 Background

In the dairy industry, one of the costliest cow illnesses is lameness, which is a condition where the cow has foot pain that affects its movement. According to the Beef Cattle Research Council (BCRC) of Canada [3], pain from lameness frequently restricts growth because it causes animals to be unwilling to eat or drink. This condition can be hereditary, or caused by the environment, nutrition, injuries, or medical conditions. Not only does it cause pain for the cows, but lame feedlot cattle grow at a slower rate than non-lame cattle, and calves diagnosed with foot rot during the final growth period take as much as fourteen days longer to reach slaughter weight. Chronic examples may be rescued if all medication withdrawal conditions have been satisfied, and the animal can be relocated without further suffering. Thomsen et al. [8] discussed the seriousness of the issue, noting that when the prevalence of recognizable locomotor issues in dairy cattle exceeds 10%, it shows that the preventative plan is insufficient. In accordance with Thomsen et al. [8], the prevalence of lameness in dairy cows varies greatly among countries, ranging from 5.1% in Sweden to 45% in the United States. Additionally, the prevalence of severe lameness ranges between 1.8% in Norway and 21.2% in Brazil. Unfortunately, farmers underestimate the prevalence of lameness due to difficulty spotting it early on. Therefore, several scoring systems have been developed to detect early lameness in dairy cows through observable changes in their walking pattern. These are often scored on a 5-point scale, ranging from normal to very lame. Symptoms of lameness include back arching, leg swinging, short steps, jerky head bobbing, joint stiffness, and difficulty bearing weight on one or more hooves. [7]

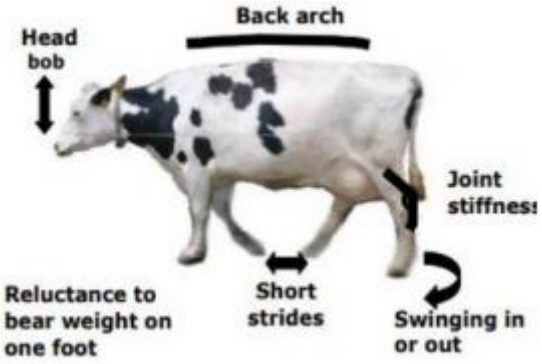


Figure 1.1 Symptoms of lameness.

1.2 Problem Statement and Motivation

Lameness in cows initiates a detrimental chain reaction with many repercussions for both the farmer and the animal, including a decrease in milk output of about 20%, reproductive failure, stunted weight gain, and, in many instances, culling of the animal. Lameness is rarely seen in preclinical cases of illness, but milk supply might begin to diminish, stressing the necessity of early diagnosis. [6] According to a study done by Gessese et al. [1] at the dairy farms in Hawassa Town, Ethiopia, only one of the 19 farms studied practiced early detection of lameness. The rate of lameness in milk cows is significantly greater (10.59%) in farms without methods for early detection than in those with such technologies, where lameness is observed at 7.7%, stressing the necessity of proper preventive and curative solutions.

1.3 Project Objectives

The aim of this project is to develop a portable, affordable mobile application that will identify cow lameness for farmers early on by analyzing pictures of the cow's back posture. Using computer vision (CV) techniques and Root Mean Squared Error (RMSE) computations from specified anatomical key points, the application specifically focuses on detecting abnormal back arching, a key sign of lameness. By facilitating early diagnosis, this method aims to lessen cow suffering, enhance animal welfare, and assist farmers in avoiding financial losses brought on by slower growth rates, lower milk production, and higher veterinary expenses.

Apart from detecting lameness, the application includes a basic herd management function that makes farm management tasks easier by enabling users to register cows by matching their coat patterns and keep track of all their medical history.

To summarise, this project aims for:

- early identification of back arching in cows as a sign of lameness to ensure timely treatment and prevent unnecessary pain and suffering.
- simplify and expedite the lameness identification process for farmers, reducing their workload.
- enhance cow welfare and contribute to a more humane and compassionate dairy sector.

1.4 Project Scope and Direction

This project aims to develop a cost-effective and portable mobile detection application tailored for dairy farmers to identify lameness in cows through image analysis. The application will have the capability to classify images as either lame or not lame by calculating the RMSE of a line formed by joining the key anatomical points along the cow's back, which will be further discussed in Chapter 3. An arching pattern in the back will indicate lameness and potential discomfort.

Additionally, the application will allow users to register individual dairy cows into a database, differentiating them based on their coat colour and pattern for easy identification. Once registered, users will be able to input and maintain detailed medical histories for each cow, providing a comprehensive tool for managing the health and well-being of their livestock. This solution aims to offer an accessible and practical alternative to more expensive and complex systems, making it particularly suitable for small to medium-sized dairy operations.

However, lightweight herd tracking and image-based lameness identification by detecting the key points of the cow’s back are the main focuses for this project. Other lameness identification methods like gait analysis or live video detection are not supported. Furthermore, it isn't comprehensive enough to diagnose various kinds of health problems, predict future health risks, or integrate data from external sensors like pressure mats or accelerometers.

1.5 Contributions

This project is intended to help small and medium-sized dairy farms and presents a novel and easily understood method for identifying cow lameness. The mobile application uses photo recognition technology to provide farmers with a portable, affordable tool for identifying early indicators of lameness in their herds. This proactive strategy is a useful and important resource for farms with limited access to advanced diagnostic technologies because it not only helps minimize the financial losses associated with decreased dairy production but also lowers veterinary bills.

The application's utility is further enhanced by the ability to register individual cows and keep track of extensive medical histories, giving farmers access to a simple and comprehensive health management system. This contribution is especially important for small and medium-

sized farms because it gives them the ability to adopt cost and complexity-effective health monitoring and management approaches that were previously unaffordable. This project promotes the dairy industry's long-term profitability, overall production, and animal welfare by encouraging early diagnosis and sustainable farming techniques.

1.6 Report Organization

This report is organized into seven chapters. Chapter 1 introduces the project, outlining the problem statement, objectives, scope, and contributions. Chapter 2 reviews related literature and existing methods for lameness detection, cow recognition, and herd management. Chapter 3 describes the system methodology, including the development approach and data preparation. Chapter 4 details the system design, covering the architecture, models, backend, and mobile application structure. Chapter 5 focuses on the implementation of the models, backend APIs, and mobile application. Chapter 6 presents the evaluation of the system, analyzing performance, limitations, and discussion of results. Finally, Chapter 7 concludes the report by summarizing key findings and offering recommendations for future work.

Table 1.1 Report Organization.

Chapter 1	Introduction
Chapter 2	Literature Review
Chapter 3	System Methodology/Approach
Chapter 4	System Design
Chapter 5	System Implementation
Chapter 6	System Evaluation And Discussion
Chapter 7	Conclusion and Recommendations

Chapter 2
Literature Review

2.1 Previous Works on Cow Lameness Detection

In a study done by Barney et al. [10] in 2023 with the goal of developing a fully automated system that could identify lameness in numerous cows at the same time utilizing advanced deep learning techniques. The system detects and estimates cow posture using a deep convolutional neural network, specifically the Mask-RCNN method. Using a triple cross-validation procedure, the system accurately identified lameness in dairy cows at 94%. Unlike previous systems, which simply provided a simple binary classification (lame or not lame), this approach divides cows into four stages of lameness using the Agriculture and Horticulture Development Board (AHDB) mobility rating system. The approach examines numerous major markers of lameness, including the cow's back posture, head position, and the spatial relationship between several anatomical sites on its body. [20] By recording the cows' movements over time, the technology provides a more accurate and thorough assessment of their lameness than standard methods based on single-frame analysis.

One major conclusion of the study is that the system's analysis of back posture, namely the RMSE of the back's regression residuals, has a considerable correlation with lameness. [10] In addition, the system examines the cow's back area and head movement to acquire a better understanding of the severity of lameness. The study shows that this automated technology is extremely accurate and may be used on farms without interfering with the traditional milking procedure. However, the paper also states that the algorithm struggles to recognize the cows' feet and knees due to poor contrast, which will be addressed in future versions. Additional work will focus on verifying the system across multiple dairy farms and increasing its accuracy beyond the present 94%.

In another study done in 2020 by Kang et al. [11] that aims to provide a highly accurate approach for identifying cow lameness using CV technologies, specifically by analyzing the supporting phase of each hoof. The sustaining phase is the time a cow's hoof remains in touch with the ground when walking. The study found that differences in the supporting phases of the four hooves are highly related to lameness severity. The suggested method achieved 96%

accuracy, beating current systems such as StepMetrix, which had 85% accuracy. The study emphasizes the limitations of individual variability in existing lameness detection approaches, which frequently compare different cows. On the other hand, by comparing the four hooves of the same cow, the supporting phase method lessens the impact of these individual variations. This method works especially well for pinpointing the exact damaged hoof and diagnosing mild to moderate lameness.

With an 87% placement accuracy, the study also investigates the use of deep learning for cow leg recognition. Nevertheless, certain obstacles were noted, including false positives (such as misidentifying human limbs for cow limbs) and false negatives caused by fast motion or shadows. The paper proposes enhancements such as eliminating human interference in images, utilizing high-speed cameras, and combining several detection systems to improve accuracy to solve these problems. In order to develop a more reliable, multi-feature detection system, the authors suggest merging the supporting phase method with additional lameness detection techniques, such as those based on back arch, head bob, and gait speed, in future research. The study concludes that the suggested CV technology has a great deal of potential for accurate, real-time lameness diagnosis in dairy cattle. This might enhance animal welfare and lessen the detrimental effects of lameness on the dairy sector.

2.2 Digital Image Analysis and Computer Vision Systems (CVS)

The process of digital image analysis involves collecting descriptive statistics or detecting structures in order to extract useful data from photographs. Researchers then apply techniques such as thresholding, sharpening, and segmentation to this data. Segmenting an image into meaningful portions can be done in a variety of ways, from basic binary divisions to complex methods that take texture, colour, shape, and spatial information into account.

The goal of CV is to represent the outside world by deciphering and identifying features in images. CV uses image processing techniques to address visual issues and is closely related to machine learning and pattern recognition. In order to analyze visual data and detect geometrical structures, pattern recognition is necessary, often using modified machine learning methods.

CVS, which analyze animal behaviour, bodily condition, meat quality, and other variables automatically, replaced manual inspections in the animal and veterinary sciences that were

formerly done with trained eyes or imaging technology. Although prior techniques worked, they were laborious, costly, and could cause discomfort to the animals. CVS decreases human involvement while increasing accuracy and efficiency. Variable lighting, different backgrounds, and real-time demands are still problems, but new technologies like 3D cameras, depth sensors, and machine learning help solve them. CVS enables high-throughput phenotyping and real-time monitoring, which improves breeding programs, animal welfare, and livestock management.

2.3 Ultralytics YOLOv8 (You Only Look Once)

In the field of CV, the YOLO series of models has gained widespread recognition. The popularity of YOLO can be attributed to its high accuracy while keeping its model size small. A broad range of developers can use YOLO models because they can be trained on a single Graphic Processing Unit (GPU). It may be inexpensively deployed on edge hardware or in the cloud by machine learning practitioners. Since its initial introduction by Joseph Redmond in 2015, YOLO has been fostered by the CV community. [15]

An important development in the YOLO family of real-time object identification models is YOLOv8, which Ultralytics released on January 10, 2023. Compared to its predecessors and other object identification methods, YOLOv8 has several advantages. It is perfect for time-sensitive applications like robotics and autonomous driving because it maintains real-time inference speeds while achieving state-of-the-art accuracy across a variety of detection benchmarks. Its effective and lightweight architecture enables deployment on edge devices with constrained processing power. Additionally, because YOLOv8 is an open-source and community driven project, it enjoys continuous improvements and widespread use in the field of computer vision research. [21]

YOLOv8 is made up of three basic parts: the detection head, neck, and backbone. To enhance training and information flow, the backbone processes the input image using CSPDarknet53 to extract important features. The neck, which is based on PANet, aids the model in detecting objects of varied sizes by combining information from numerous visual regions. Lastly, the model's predictions are made by the detecting head. By employing a more exact technique to evaluate box overlap and match predicted boxes with real items, YOLOv8

enhances this section and produces predictions that are more accurate, even for objects that have some overlap or close together.

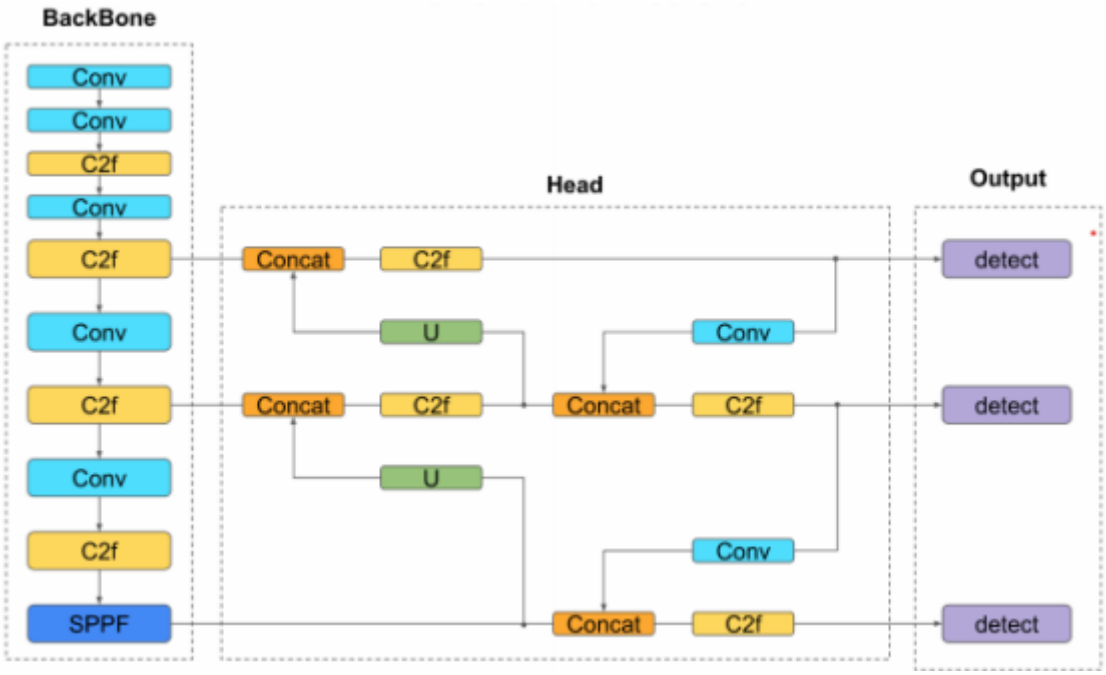


Figure 2.1 Architecture of YOLOv8. [15]

In addition to object detection, recent YOLOv8 variations have been enhanced to include pose estimation. Cai et al. [33] introduced a YOLOv8-based network that included an Efficient Multi-scale Receptive Field (EMRF) module and an Expanded Feature Pyramid Network (EFPN), which dramatically improved performance on human key point identification tasks. Similarly, Xu et al. [34] incorporated the C2F-SimDLKA module into YOLOv8-Pose, resulting in increased resilience and reliability under occlusion and scale variation situations. These investigations illustrate YOLOv8's adaptability for pose estimation challenges that are directly related to animal keypoint recognition.

Another significant architectural change in YOLOv8 is the implementation of an anchor-free detection head, which replaces the anchor-based designs of prior versions such as YOLOv5 and YOLOv7. This improvement minimizes dependence on manually defined box anchors, simplifies model training, and improves flexibility to objects of various forms and sizes [35].

Despite these advantages, YOLOv8 has limits when used in real-world settings. Performance may suffer in the presence of crowded backgrounds, high object overlap, or

extremely small objects, highlighting the necessity to supplement YOLOv8 with embedding-based or specialized posture estimation algorithms in fields such as livestock monitoring [35], [34].

2.4 Previous Works on Cow Identification

Cattle identification with computer vision has received more attention in recent years, with techniques ranging from conventional classification models to more complex embedding-based algorithms.

Sharma et al. published a paper in 2025 [16] that provided a deep metric learning framework for cattle recognition using depth data as a novel biometric. Using an off-the-shelf 3D camera, the researchers developed a breed-agnostic method for global cattle identification. Their approach revealed that depth information might overcome the constraints of coat-pattern-based systems, extending its usefulness to the 68% of UK cattle breeds that lack distinguishing visual marks. The method used Convolutional Neural Network (CNN) and Multi-Layer Perceptron (MLP) backbones for learning generic embedding spaces from body shape, which did not require species-specific coat patterns or muzzle prints.

In contrast, Dondona et al. [18] utilized 2D profile photos and texture characteristics for re-identification. Their approach used a dual-branch deep convolutional neural network (DCNN) to evaluate frontal and side profiles separately, resulting in a 128-dimensional L2-normalized embedding. By avoiding sharing of parameters between both branches, the framework was able to collect unique traits from each profile. The authors emphasized the advantages of deep models over shallow models because to their robustness to variations in posture, scenery, and illumination. They also observed that multi-view approaches outperform single-view methods, as cattle frequently lack symmetry between left and right profiles.

In addition to these individual investigations, Hossain et al. [23] published a thorough study of machine learning algorithms for cattle identification. The review underlined that existing approaches are often either classification-based (and hence limited to the animals in training) or metric learning-based, which often calls for complex training procedures. Traditional machine learning techniques like SVM, KNN, and ANN were among the most popular models, as were deep learning architectures including CNNs, ResNet, Inception, YOLO, and Faster R-

CNN. Their research highlighted both opportunities and limitations, especially in terms of data availability, generalization to unidentified individuals, and dealing with real-world agricultural settings.

These experiments, taken collectively, demonstrate the transition from traditional classification approaches towards more adaptable embedding-based techniques, as well as the importance of multi-view and depth modalities in enhancing performance. However, the reliance on specialized equipment (e.g., depth sensors), multi-view datasets, or complicated metric learning losses poses practical challenges for widespread use. This encourages the development of simpler but effective embedding approaches, such as the ResNet18-based method used in this study, which uses standard RGB images to generate normalized embeddings that generalize to new cows without the need for additional modalities or overly complicated training strategies.

2.5 Biometric Sensing

Animal recognition and tracking is the process of accurately identifying distinct animals and what they produce using a unique identity or marker. Ear tags, marking, tattooing, and electronic devices have all been used to identify and track cattle for a long time. However, their usefulness is restricted by their vulnerability to losses, duplications, counterfeiting, and security issues. Electrical identification technologies, particularly Radio Frequency Identification (RFID), came later than traditional methods and are more reliable. However, they raise privacy, installation, and operational problems [2]. Monitoring and tracking cattle's physiological and behavioural parameters can supplement the usage of livestock identification systems.

As a result, a better livestock identification technique is now desirable. Biometrics, a technology now utilized to identify humans, represents a promising advancement in the livestock identification field. Animal biometrics offers a wide range of uses, including grouping cattle, tracking them from being born to the very end of the food chain, and studying disease trends and population patterns. Biometric modalities automate authentication and identity systems, increasing security while maintaining accuracy and dependability.

Biometrics technology not only offers an automatic verification technique but also offers convenience to the user because they do not need to memorize information or carry a token

with them, which is a significant problem with traditional cattle identification systems that are all based on devices that attach to the animal rather than the animal itself. Biometric identifiers, such as muzzle prints and retina imaging, provide a simple and secure method for offering a failsafe livestock identification system that ensures traceability of animals back to their farm of origin.

2.6 Embedding Learning

Embedding learning is the technique of mapping data with high dimensions to a lower dimension while preserving semantic or structural connections. In computer vision, embeddings are frequently utilized for functions such as face recognition, human re-identification, and imagery retrieval, where the goal is to evaluate similarity between examples rather than classify them [24]. Early approaches frequently used customized characteristics and shallow models for describing visual input, but these methods have limitations in their capacity to generalize across different variables such as position, illumination, and backdrop changes [25]. CNNs emerged as the main feature extraction backbone as deep learning progressed, yielding stable and discriminative embeddings which better capture intra-class variability and inter-class separability [26].

Metric learning is a central paradigm in embedding learning, in which the network is trained to reduce the distance between embeddings of similar inputs while increasing the distance between embeddings of different ones. Several loss functions have been suggested for this reason, including contrastive loss [27], triplet loss [28], and newer variations like ArcFace [29]. These methods are especially beneficial for open-set issues, in which the system must recognize persons that were not seen during training. In animal identification, embeddings enable systems to progress beyond closed-set model classifications that are limited to a certain number of known individuals. Learning a discriminative embedding space enables new faces to be compared to a reference gallery rather than retraining the whole network. This adaptability makes embeddings especially useful for livestock monitoring and identification in areas where populations are big and dynamic.

2.7 Residual Network and Its Variance

Residual Networks (ResNets) were first introduced by He et al. [12] to address the problem of vanishing gradients in deep neural networks. By introducing residual connections, ResNet enables the training of very deep architectures without suffering from performance degradation. This innovation dramatically improved trainability and allowed networks to scale to depths such as 50–152 layers, which outperformed earlier CNN architectures on ImageNet and other large-scale vision tasks. Since then, various versions of ResNet have been developed, such as ResNet-18, ResNet-34, ResNet-50, and ResNet-101, each differing mainly in depth and computational complexity [13]. Lighter models such as ResNet-18 and ResNet-34 are often preferred for mobile or resource-constrained applications, while deeper models such as ResNet-50 and ResNet-101 are used for large-scale recognition tasks where higher accuracy is required [14]. Despite being shallower, ResNet-18 provides a good trade-off between accuracy and efficiency, making it a suitable backbone for embedding-based recognition tasks in environments where computational resources are limited.

2.8 Limitations of Previous Studies

Previous research has shown that although proposed systems could detect and analyze cow lameness with high accuracy, they often relied on costly hardware such as mounted cameras, force plates, or motion sensors, which limited their practicality for widespread use. Most studies focused solely on the detection method without addressing herd management needs, such as linking results to individual cows or maintaining medical records. In addition, the absence of cow recognition meant that lameness detection results could not be associated with specific animals. Many approaches also depended on expensive setups, including high-speed cameras and continuous power supply, as they were designed to analyze video footage of cows walking rather than static images. These constraints significantly increased implementation costs and reduced the accessibility of such solutions for typical dairy farms.

Chapter 3
System Methodology/Approach

3.1 Proposed System Framework

This project proposes the development of a cost-effective and portable mobile application designed to classify images of one to five cows as either lame or not lame by calculating the RMSE of a line formed by joining the key anatomical points along their backs. If a cow's back displays an arching pattern, the application will classify the cow as potentially lame. The study focuses on back arching as it is a significant indicator of discomfort and sickness. [22] While not definitive on its own, it is an important marker in the early detection of lameness.

To further enhance the mobile app's usefulness, the application will allow users to register individual dairy cows into a database, where they can be differentiated using the cow recognition function based on their coat patterns. Once registered, users will be able to access a medical history for each cow. This system is designed with the farmers in mind, offering a practical and accessible tool to support early diagnosis and management of lameness, particularly benefiting small to medium-sized dairy operations by reducing the need for more expensive and complex equipment.

Therefore, we can divide this project into two phases: model training and mobile application development, each to be done during FYP1 and FYP2 respectively. In the first phase, models for lameness detection and cow biosensing are trained. In the second phase, the trained models are integrated into a mobile application for user access and operation. Further finetuning will also be done on the models top of developing the mobile app. In the first phase, RMSE is calculated to measure the deviations of key anatomical points along a cow's back from a predicted straight line, which represents a healthy posture. The system first identifies and marks five critical points along the cow's back:

1. Tail Setting
2. Hook Bone
3. Back
4. Withers
5. Lower Neck.

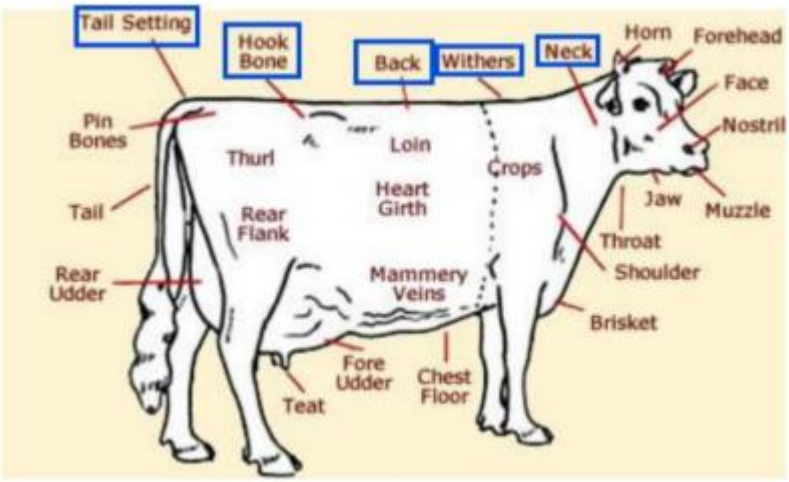


Figure 3.1 Anatomy of A Cow. [4]

A healthy cow would have these points aligned along a nearly straight line, as seen in Figure 3.2. The RMSE is then computed by measuring how far each of these marked points deviates vertically from this expected line. These deviations are squared, averaged, and then the square root is taken to yield the RMSE value. Larger RMSE values indicate greater departures from the healthy posture, suggesting a higher likelihood of lameness. This approach allows the application to quantify lameness by analysing the degree of deviation in the cow's back alignment.

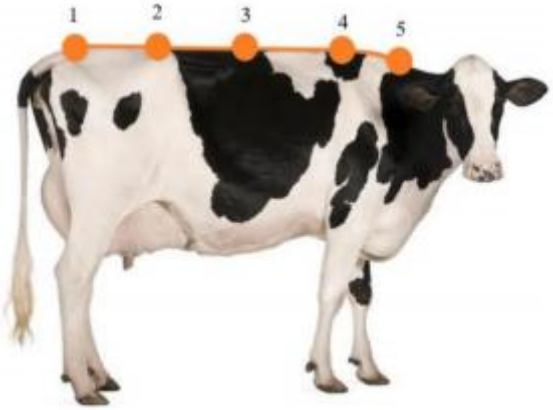


Figure 3.2 Standard Line on a Non-lame Holstein Friesian Cow. [10]

The Holstein Friesian breed will serve as the main sample for the biosensing portion of the device. This breed was chosen because of its size and distinct speckled black and white markings, which are perfect for precise point analysis and detection within the biosensing

framework. Once phase one development is done, the models will be deployed as a REST API to be used by the mobile app.

3.2 System Requirements

3.2.1 Hardware

A computer and an Android smartphone are the two main pieces of hardware used in this project. The primary development platform is the computer, which is used to build the application in Android Studio, train the picture recognition model, and perform computational operations such as lameness detection RMSE calculations. Farmers may register cows, take real-time images, and record medical histories with the app's practical deployment made possible by the Android mobile. Because of its price and portability, it is perfect for usage in rural farming settings, guaranteeing accessibility and use for a broad spectrum of users.

Table 3.1 Specifications of laptop.

Description	Specifications
Model	Victus by HP Gaming Laptop 16-r0039TX
Processor	Intel® Core™ i5 13500HX 13th Generation
Operating System	Windows 11
Graphic	NVIDIA® GeForce RTX™ 4060
Memory	16 GB DDR5 RAM
Storage	512 GB SSD

Table 3.2 Specifications of mobile phone.

Description	Specifications
Model	ELE-L29 Huawei P30
Processor	Huawei Kirin 980
Operating System	EMUI 9.1 (Based on Android 9)
Memory	8 GB RAM
Storage	128GB

3.2.2 Software

The software used in this project is:

1. **Android Studio (Meerkat):** The Integrated Development Environment (IDE) to develop the front end and back end of the mobile application
2. **Google Colaboratory (Colab):** The main development platform for training and testing models. It is very helpful due to the free cloud GPU, although it may take a long time for free computing resources to be available again.
3. **Google Cloud Run:** Used to deploy the backend API, making the model services accessible to the mobile application.
4. **Google Drive:** Cloud storage for pre-processed datasets and trained model files. It can be mounted with Google Colab.
5. **Google Firebase:** Serves as the backend database for the mobile application, storing cow records, health data, and images.
6. **Jupyter Notebook:** Used to run Python locally, mainly used for light weight tasks like preprocessing and manual annotation.
7. **Visual Studio Code (VS Code):** Used for code editing, debugging, and project management, particularly for scripting and API development outside Colab.

3.3 System Overview

When building dataset and pre-processing the data, the process is done in local computer and Jupyter Notebook since it is easier to save and manage the unprocessed images on local storage rather than uploading them to Google Drive before they are cleaned as Google Drive has a limited storage of 15GB for a free account. Once the data are pre-processed and labelled, they will be uploaded to the Google Drive for model training.

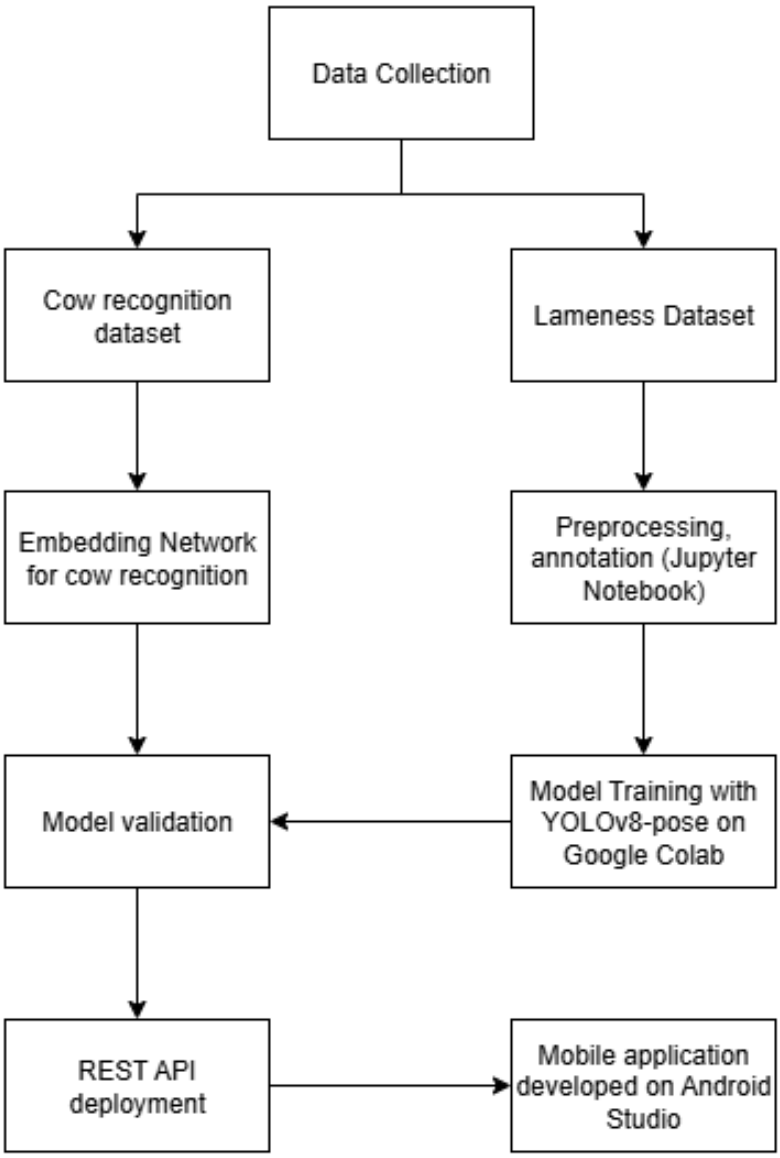


Figure 3.3 Overview of The System.

During training, for convenient access from any system, a Google Colab notebook was made and saved in a Google Drive account. The Python code required for model training was written in the Colab notebook, which is similar to a Jupyter notebook. An external cloud GPU from Google Colab was used to increase training speed and accuracy. Although the free GPU from Google Colab only has a certain limitation (in this case about 6 hours of usage, or two 150 epochs run of YOLOv8-pose), after that we will need to wait for it to cool down, it is still a valuable free resource that can increase the training speed of the model. Additionally, Python would be the main programming language for this project.

After the lameness detection and cow recognition models had been trained and tested, they merged into a single pipeline. This pipeline was subsequently published to Google Cloud Run as a RESTful API, allowing external systems to utilize the functions without requiring heavy local processing. Scalability and accessibility are ensured by deploying the models in this method, as the system may be accessed from any internet-connected android mobile device.

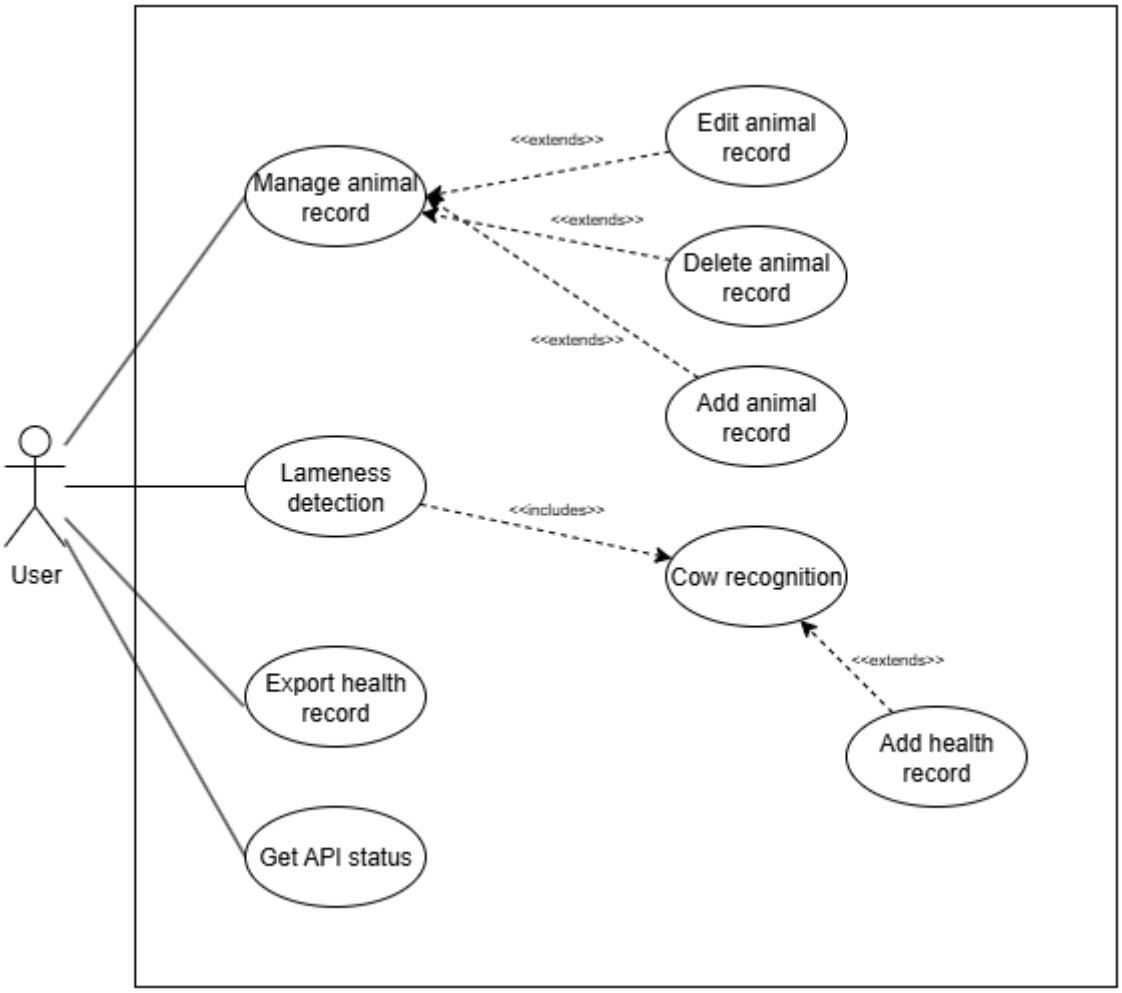


Figure 3.4 Use case diagram

This project's goal is to produce a straightforward and easy-to-use mobile application. As a result, the software focuses on essential features including cow recognition, lameness detection, and animal record administration via CRUD (Create, Read, Update, Delete) activities, with the ability to directly add results to each animal's health record. Users can also export health records from the system as they wish. Additionally, the application offers API status updates on the main page for transparency and dependability.

3.4 Software Development Lifecycle

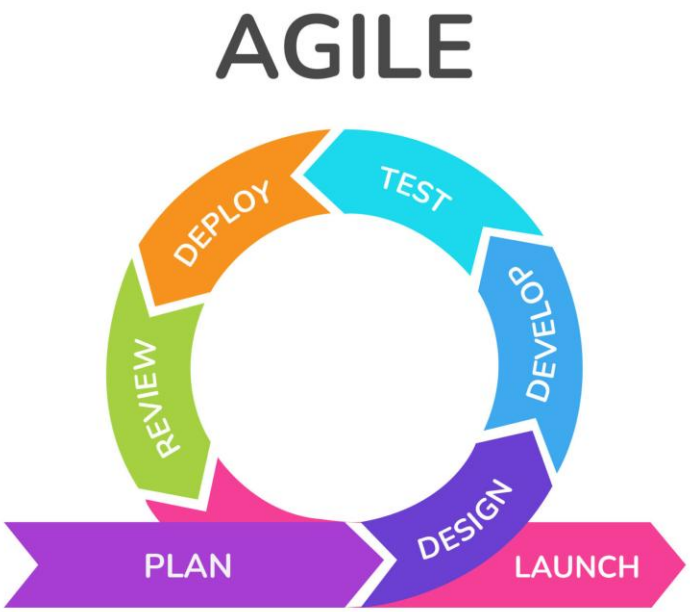


Figure 3.5 Agile Development Lifecycle.

This project followed an Agile development approach rather than a traditional Waterfall model. The main reason for adopting Agile was the uncertainty in the data collection and annotation process. Since no ready-to-use database for cow lameness detection and recognition was available, the dataset had to be created manually, introducing unpredictability in both quantity and quality of data. Agile allowed flexibility to adapt to these uncertainties by working in short, iterative cycles, where models, pipelines, and app components could be refined based on intermediate results.

3.4.1 Planning

The project began with a planning stage where deliverables were set to include:

- a system capable of detecting lameness in cows using pose estimation.
- a cow recognition system for identifying individual cows.
- a mobile application that allows farmers to upload cow images, receive results, and manage records.

The project scope, timeline, and tools were also defined. Technologies included YOLOv8 for pose estimation, ArcFace with ResNet18 for cow recognition, Google Colab for model training, and Firebase for cloud backend integration. In FYP1, the emphasis was placed on dataset creation, annotation, preprocessing, and lameness/recognition model training. In FYP2, focus shifted towards system integration, backend pipeline, and mobile application development.

Table 3.3 FYP1 Timeline.

FYP 1 Timeline												
Task/Week	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Project planning												
Data collection and preprocessing												
Annotation and model training												
Cow recognition model training												
Model testing												
Mobile app UI design												

Table 3.4 FYP2 Timeline.

FYP 2 Timeline													
Task/Week	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Mobile app front end development													
Model Finetuning													
Backend pipeline testing													
API development and testing													
Mobile app back end development													
App testing													

3.4.2 Analysis

At this stage, system requirements were analysed and divided into two categories:

1. Functional Requirements
    - image preprocessing.
    - manual annotation for key points and recognition labels.
    - training pipelines for both lameness detection and cow recognition.
    - backend API services for handling uploaded images, running inference, and returning predictions.
    - mobile application features: image upload, result visualization, and herd management.

2. **Non-Functional Requirements**
    - performance: real-time inference to provide predictions within seconds.
    - usability: simple and farmer-friendly mobile app interface.
    - scalability: cloud integration (Firebase + Google Cloud) for dataset expansion.

3.4.3 Design

The system architecture was structured into three interconnected components: model design, backend design, and application design. For the model design, YOLOv8 with key point annotations was implemented for lameness detection, while ResNet18 integrated with ArcFace was used to generate embeddings for cow recognition and identity verification. The backend design was built using FastAPI to provide RESTful endpoints such as /analyze and /gallery_status for processing and providing back end status. Firebase services were integrated to support the backend, with Firebase Storage managing image datasets and galleries, and Firestore managing structured data. Finally, the application design focused on a farmer-friendly mobile interface, offering simple features such as image uploads, status monitoring, and result visualization. Communication between the mobile application and backend was streamlined, enabling requests to flow from the app to the cloud inference pipeline and returning processed results efficiently.

3.4.4 Implementation

The implementation phase was carried out in iterative Agile sprints, with development split between model, backend, and mobile components. For the models, preprocessing and manual annotation scripts were first established to prepare the dataset. YOLOv8 was trained for lameness detection, while ResNet18 with ArcFace loss was trained for cow recognition, both optimized and saved for deployment. Importantly, the system was designed to continuously add new data and update the models throughout development, ensuring that performance improved as more annotated images became available. On the backend side, a FastAPI server was developed to expose REST endpoints for lameness detection, cow recognition, and combined analysis tasks. Additional endpoints such as /gallery_status and /rebuild_gallery were included to maintain updated recognition embeddings. Firebase was integrated to manage storage and data flow to ensure scalability and secure access.

3.4.5 Testing

Testing was carried out at both the model and application levels. The YOLOv8 model was tested using unseen validation datasets to ensure reliable lameness detection, while the cow recognition model was evaluated using Rank-1 accuracy to verify its ability to correctly identify individual cows. The backend pipeline and APIs were tested to confirm smooth communication between the mobile app and the server, including correct handling of both lameness detection and recognition requests. The mobile app itself underwent front-end and back-end testing to ensure proper integration, and accurate visualization of results.

3.4.6 Deployment

Once testing was completed, the trained YOLO model (best.pt), trained recognition model, and backend services were prepared for deployment. The mobile app was connected to Firebase for real-time use, enabling end-users to upload cow images and receive lameness predictions. Maintenance involves continuously monitoring the system's performance, updating the trained model as more data becomes available.

.

Chapter 4

System Design

4.1 System Architecture Diagram

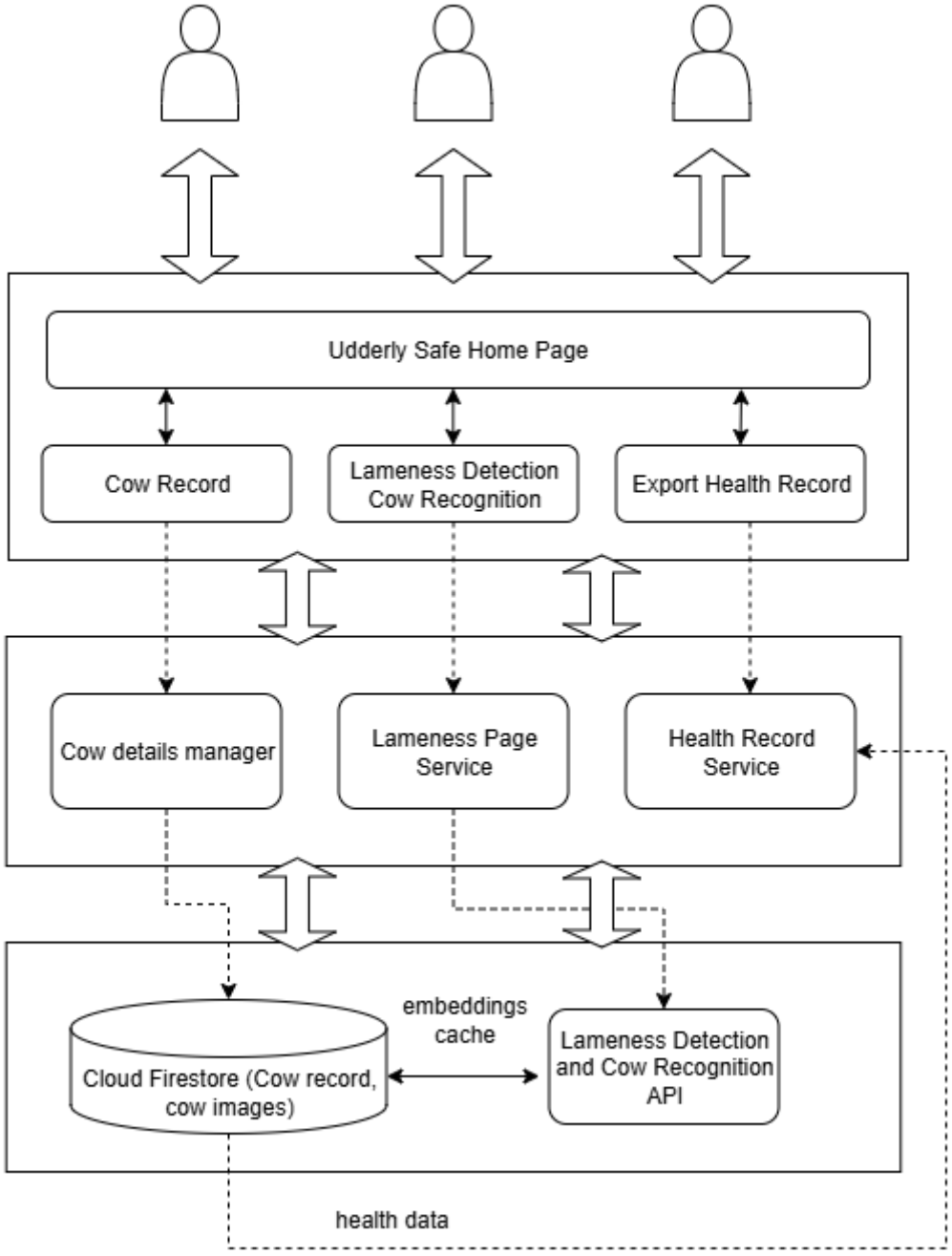


Figure 4.1 System Architecture Diagram.

4.2 Lameness Detection Model

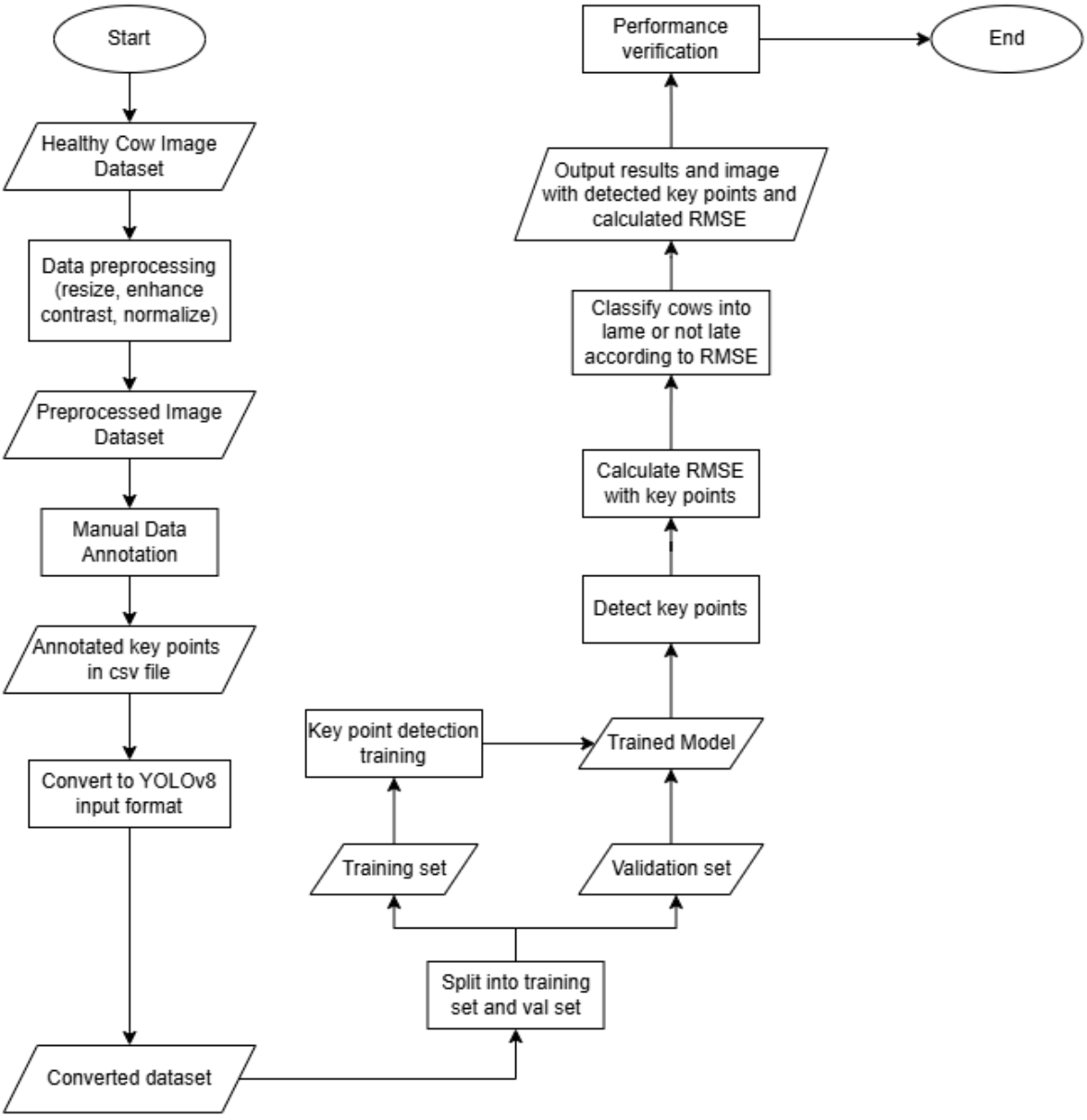


Figure 4.2 Flowchart of Key Point Detection Model's workflow with YOLOv8.

4.2.1 Data Collection

As shown in Figure 4.2, the development process starts with data collection. A major challenge of this project is the lack of existing datasets for lameness detection. To train a model with high accuracy, a reliable dataset of side-view images of standing cows is required. Since lame cow images are scarce online, this project adopts an anomaly detection approach, where healthy cows are treated as the baseline. Most images of cows found online are healthy, such as those from stock image repositories, cattle show winners, and breed catalogues. To simulate

lameness and improve the model's strength, the cow’s back is edited to be slightly curved to simulate lame cow backs which results in variances that reflect potential lameness indicators. The merged dataset with a total of 1586 images was then labelled and ready for model training.

Figure 4.3 Comparison between an original healthy cow image and edited curved cow back.

4.2.2 Model Training

The manually annotated CSV file is first converted into the YOLOv8-Pose format, which contains bounding boxes and key point coordinates normalized between 0 and 1 with respect to the image dimensions. This will be the first step in the training process for lameness detection. Five key points along the cow's will be taken out of each image and recorded to separate label files. The associated bounding box coordinates are calculated from the key points' extreme points. This is to establish the needs bounding boxes and key points for posture estimation so that it is compatible with YOLOv8-posture.

The dataset will be divided into training and validation sets after every image has been transformed, usually in an 80:20 ratio. To ensure that the model can learn efficiently while being assessed on unseen data, corresponding photos and label files are transferred into separate folders for training and validation.

Next, utilizing a GPU for faster computing (in this case, the T4 GPU from Google Colab), the YOLOv8-Pose model is trained on the processed dataset after being initialized with a pre-trained backbone (yolov8n-pose.pt). Following training, the top-performing weights are stored as best.pt, which can later be used for inference in the lameness detection pipeline.

4.2.3 RMSE Calculation

After the model is trained, it will be able to detect key points along the cow's back. This information will be stored for RMSE calculations. To calculate the RMSE, the algorithm predicts the Y-coordinate values for each set of key points using a linear regression model based on the key points' X-coordinate values. After the predictions are made, the difference between the real and expected Y-values for each detected cow's key points is evaluated using the RMSE formula in Figure 4.4.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Figure 4.4 Formula of RMSE.

4.3 System Workflow of Cow Recognition Model

4.3.1 Data Collection

Similar to the lameness detection model, the cow recognition system also faced challenges due to the lack of suitable public datasets. For this task, the data needed consisted of multiple images of the same cow, collected across different cows, with each image labelled by its respective cow ID. Unlike a traditional classifier, the model does not directly classify IDs. Instead, it generates embeddings that are stored and compared under each cow's ID for identification. To build this dataset, images were gathered from recorded videos of cows, mainly from the Youtube Channel "World Dairy Expo" [30], where individual frames were extracted to produce multiple images per cow.

Figure 4.5 Example of image data for a cow.

4.3.2 Model training

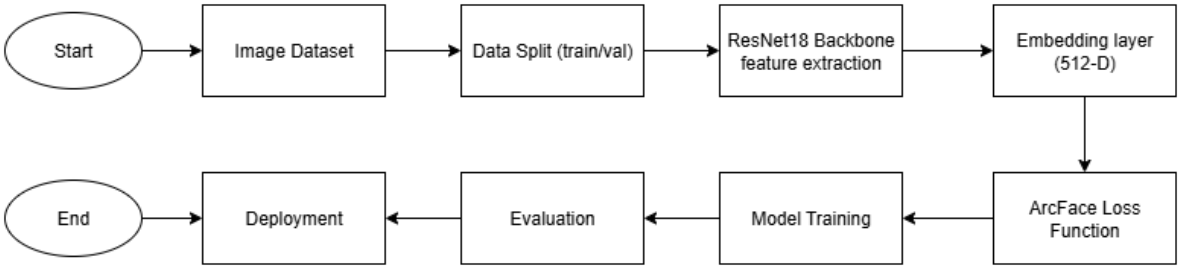


Figure 4.6 Flowchart of cow recognition model training.

Figure 4.6 shows the flowchart of training the cow recognition model. The system first splits the available cow images into training and validation sets. Each cow has its own folder, and the images are shuffled before being split into an 80:20 ratio. This ensures that the model can learn effectively while keeping some images unseen for validation. Next, a pre-trained ResNet18 backbone (from the timm library) is used as the feature extractor. This extracts high-level features from cow images, such as body shape, color patterns, and unique textures. Subsequently, a linear layer projects these features into a 512-dimensional embedding space, where each cow is represented by a unique embedding vector. Instead of normal classification, ArcFace loss is applied. This encourages embeddings of the same cow to cluster together, while pushing apart embeddings of different cows with an angular margin. This makes the system more robust for recognition. The model is trained using Adam optimizer and CrossEntropyLoss, where embeddings are compared with class weights. Training continues for several epochs, and the model with the best validation accuracy (Rank-1 accuracy) is saved.

During evaluation, embeddings of validation images are compared with each other using cosine similarity. Moreover, the system measures how often the most similar embedding belongs to the correct cow. Once trained, the model can be integrated into the recognition pipeline. During inference, an input cow image is converted into an embedding, which is then compared against the gallery to identify the cow.

4.4 REST API

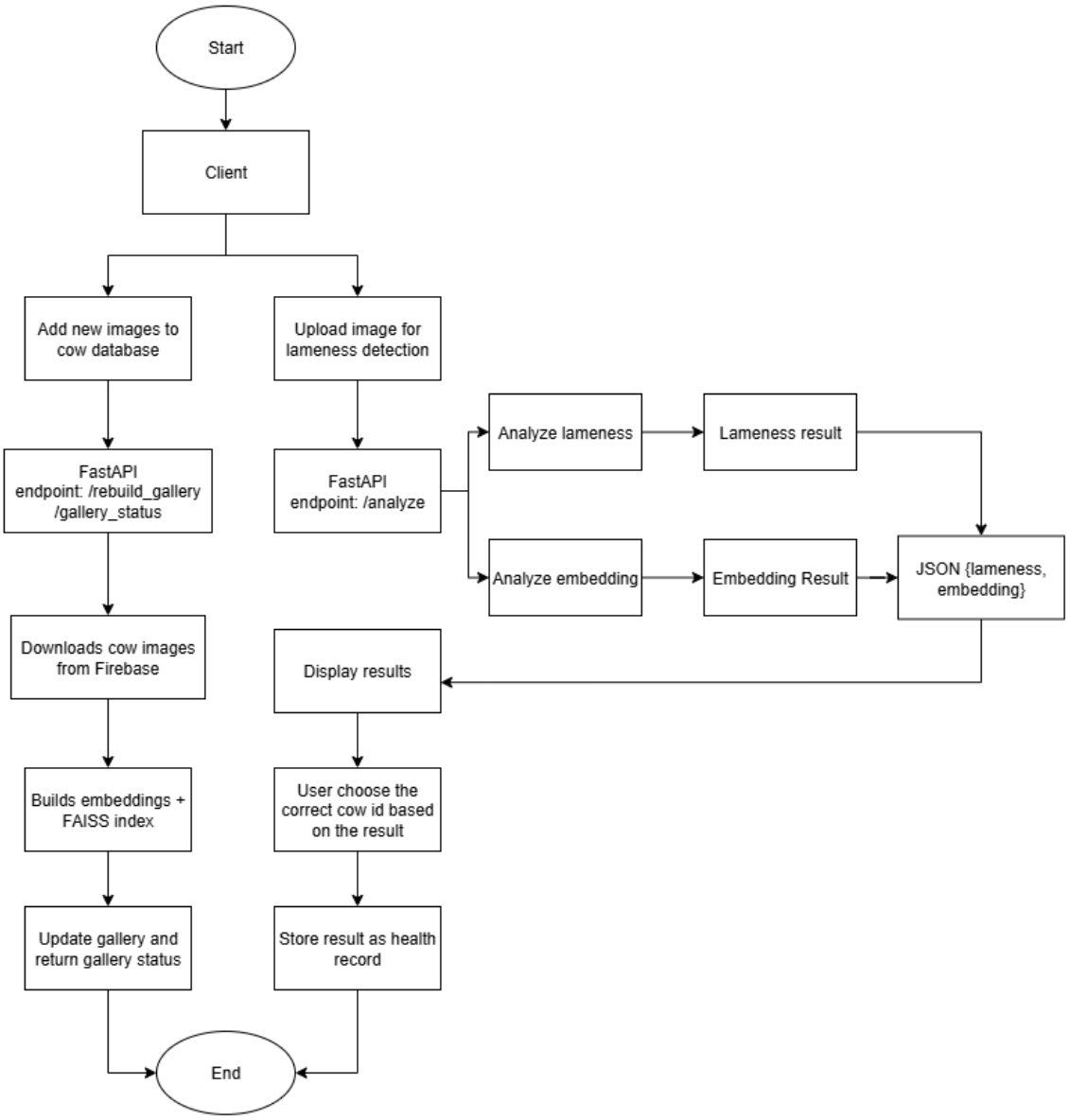


Figure 4.7 Flowchart of API call.

The cow recognition system was implemented as a REST API using FastAPI. In the final deployment, the system exposes a single endpoint, /analyze, which accepts an uploaded cow image, runs it through the recognition pipeline, and returns the closest matching cow ID from the gallery. This design streamlines the workflow by providing a unified interface for cow identification alongside lameness detection.

When a request is made, the uploaded image is first preprocessed into a normalized tensor to match the training conditions of the recognition model. During training, all images were

resized to a fixed dimension, converted to tensors, and normalized, ensuring consistent input distribution. The same preprocessing is applied during inference so that variations in raw input photos, such as resolution or pixel values, do not negatively affect performance. The processed image is then passed through the ResNet18 + ArcFace embedding model, which generates a unique feature vector. This embedding is compared against the stored embeddings in the gallery, and the cow ID with the highest similarity score is returned.

The gallery is built from cow images stored in Firebase Storage. When updated, all cow photos are embedded and organized under their respective IDs, ensuring the recognition system stays synchronized with the latest dataset. The gallery can be rebuilt dynamically via an internal API call, while its status, including the number of cows, available images, and last rebuild time, can be checked through the /gallery_status endpoint.

4.5 Mobile App Development

Figure 4.8 Metadata Structure.

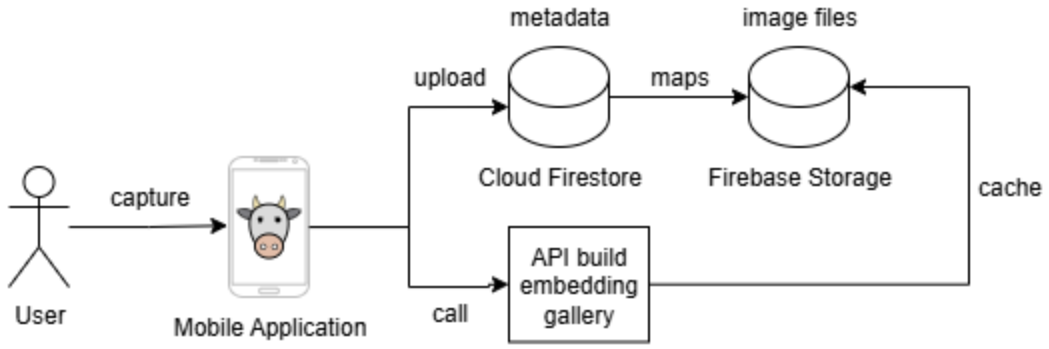


Figure 4.9 System architecture diagram for adding cow record

To utilize the application's functionality to its fullest, the user must create a cow database for their farm. This is required since the system not only detects lameness and recognizes cows but also offers herd management capabilities. To register a cow, the user must enter basic information such as its ID, name, breed, and other facts. In addition, the user must provide side-view pictures of the cow. It is important to offer photographs of both the left and right sides, as cow coat patterns are typically asymmetric, and having several perspectives increases recognition accuracy. Figure 4.8 shows the data structure of the application. The metadata of the cow will be stored in Cloud Firestore while all uploaded photographs are saved in Firebase Storage as Cloud Firestore can only handle data in JSON format. Hence, the mobile app requires internet access to sync the database and perform API calls. The API will be called to rebuild gallery every time a cow is added.

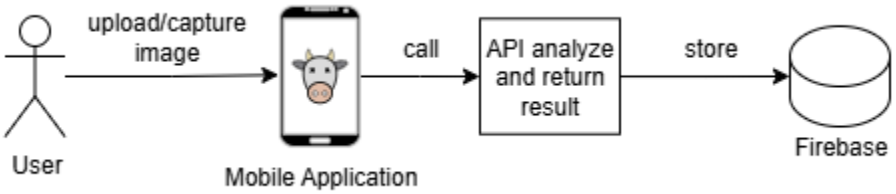


Figure 4.10 System architecture diagram for cow lameness detection

For lameness detection, the user must supply a horizontal side-view photograph of a cow. When the image is submitted, the application calls the backend API, which analyzes the input and returns both the lameness analysis, and the top-k recognition matches of cows in the database. The user can then save the lameness detection results to the cow's health record.

The recognition model, like any other machine learning system, is not always flawless. If the top three recognized cows do not include the proper match, the user can manually pick the correct cow ID. This ensures data integrity and allows the database to function reliably even when misidentified. Furthermore, the application allows users to generate health records as excel file over a specific time period. This tool helps with herd management by allowing farmers to track and monitor their cattle's health history in a systematic manner.

Chapter 5

System Implementation

5.1 Lameness Detection Model

5.1.1 Preprocessing

Before training, a preprocessing pipeline was applied to all images to improve input quality and ensure consistency. First, images were resized to a fixed resolution of 512×256, matching the horizontal shape of cows and standardizing dimensions across the dataset.

```
IMG_SIZE = (512, 256)  # Width x Height (Horizontal Rectangle)

# Resize and show an example
resized_image = cv2.resize(image, IMG_SIZE)

plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Resized Image (512x256)")
plt.show()
```

Figure 5.1 Resizing.

Next, contrast enhancement was performed using CLAHE (Contrast Limited Adaptive Histogram Equalization) on the luminance channel in LAB color space. This increased local contrast, making structural details of the cow's back more visible without amplifying noise.

```
def apply_clahe(image):
    lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)  # Convert to LAB color space
    l, a, b = cv2.split(lab)  # Split channels

    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    cl = clahe.apply(l)  # Apply CLAHE to L-channel

    lab = cv2.merge((cl, a, b))  # Merge channels back
    enhanced_image = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)  # Convert back to BGR

    return enhanced_image

# Apply CLAHE to an example image
enhanced_image = apply_clahe(resized_image)

plt.imshow(cv2.cvtColor(enhanced_image, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Contrast Enhanced Image")
plt.show()
```

Figure 5.2 Apply CLAHE.

```
normalized_image = enhanced_image / 255.0
```

Figure 5.3 Normalization.

Finally, images were normalized to the range [0,1], which stabilized training by ensuring uniform pixel intensity scales. Together, these preprocessing steps enhanced feature visibility and improved model robustness.

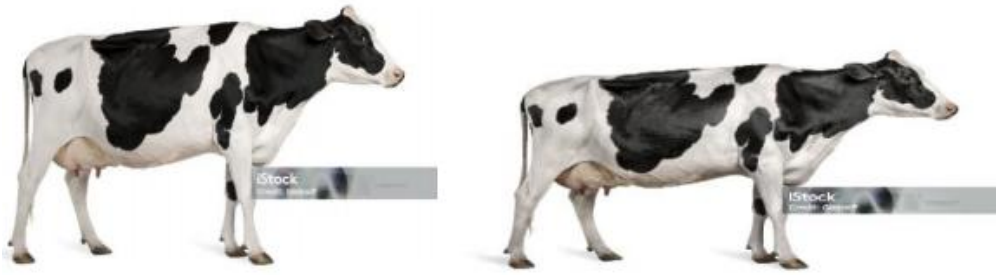


Figure 5.4 and 5.5 Comparison of unprocessed image and pre-processed image.

5.1.2 Annotation

A manual annotation tool was implemented to collect ground truth key points for cow back detection. The system allows users to interactively select five points on each image by clicking with the mouse. Left-click places a point, while right-click removes the last point in case of errors. An image window as seen is Figure 5.7 will be opened for annotation. Exactly five points has to be selected on the cow's back by clicking on the image. Once finished, pressing ESC closes the window and returns the selected points. If the user selects fewer than 5 points, a warning is issued. After annotation, the selected points are recorded in a CSV file (cow_back_points.csv), which stores the image filename and corresponding (x, y) coordinates for all five keypoints. To streamline the process, previously annotated images are automatically skipped, and new annotations are appended to the existing CSV file. This approach ensured consistent, structured, and accurate dataset creation for training the YOLOv8-Pose model.

```python
def click_event(event, x, y, flags, param):
    """
    Handles left-click (add points) and right-click (undo last point).
    """
    global manual_points, image_copy

    if event == cv2.EVENT_LBUTTONDOWN:  # Left-click to add point
        if len(manual_points) < 5:
            manual_points.append((x, y))
            cv2.circle(image_copy, (x, y), 8, (255, 0, 0), -1)  # Blue dot
            cv2.imshow("Annotate Keypoints", image_copy)

    elif event == cv2.EVENT_RBUTTONDOWN:  # Right-click to undo last point
        if manual_points:
            manual_points.pop()  # Remove last point
            image_copy = param.copy()  # Redraw from original image

            # Redraw remaining points
            for point in manual_points:
                cv2.circle(image_copy, point, 8, (255, 0, 0), -1)

            cv2.imshow("Annotate Keypoints", image_copy)
```

Figure 5.6 Click event.

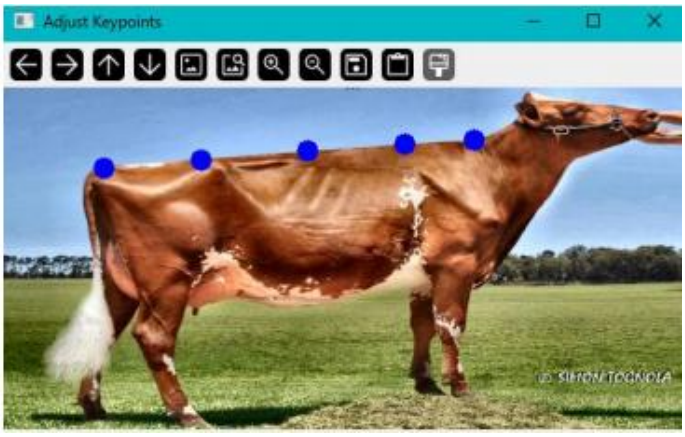


Figure 5.7 Manually Adjusted Annotation.

The preprocesses image database and CSV file are uploaded to Google Drive and ready for training. The images and CSV file used for training can be found in this link: https://drive.google.com/drive/folders/1WQQ2f7CJGby19KgmgSzNNDwHimBPD1V7?usp=sharing

```python
csv_path = os.path.join(image_folder, "cow_back_points.csv")

# Load existing annotations (to avoid duplication)
if os.path.exists(csv_path):
    df = pd.read_csv(csv_path)
    labeled_images = set(df["filename"].tolist())
else:
    df = pd.DataFrame(columns=["filename", "x1", "y1", "x2", "y2", "x3", "y3", "x4", "y4", "x5", "y5"])
    labeled_images = set()
```

Figure 5.8 CSV file set up.

```python
# Save annotations to CSV
new_data = {
    "filename": img_name,
    "x1": corrected_points[0][0], "y1": corrected_points[0][1],
    "x2": corrected_points[1][0], "y2": corrected_points[1][1],
    "x3": corrected_points[2][0], "y3": corrected_points[2][1],
    "x4": corrected_points[3][0], "y4": corrected_points[3][1],
    "x5": corrected_points[4][0], "y5": corrected_points[4][1]
}
new_df = pd.DataFrame([new_data])
df = pd.concat([df, new_df], ignore_index=True)
df.to_csv(csv_path, index=False)

print(f"✅ Saved annotations for {img_name}")
```

Figure 5.9 Saving annotation to CSV file.

filename	x1	y1	x2	y2	x3	y3	x4	y4	x5	y5
cow001.jpg	70	63	114	53	172	49	236	45	295	42
cow002.jpg	123	73	159	68	208	62	251	56	306	52
cow003.jpg	165	57	208	53	264	54	312	52	370	45
cow004.jpg	263	95	301	92	340	90	384	89	423	88
cow005.jpg	265	81	300	79	338	80	374	81	410	82
cow006.jpg	199	102	229	101	257	100	284	102	309	102
cow007.jpg	274	66	312	66	345	67	378	67	403	68
cow008.jpg	59	67	105	70	153	69	200	69	240	69
cow009.jpg	176	64	236	57	286	56	347	56	402	55
cow010.jpg	166	74	202	66	242	62	283	60	322	59
cow011.jpg	206	79	246	71	286	67	331	65	377	64
cow012.jpg	84	36	133	35	189	35	251	33	308	32

cow_back_points

Figure 5.10 Example CSV file output.

5.1.3 Model Training

Since we will be utilizing Google Colab's free T4 GPU during training, it is important to mount your Google drive to your Colab before running any codes.

```
# to remount in case of new files added
from google.colab import drive

# Unmount (to refresh)
drive.flush_and_unmount()

# Remount
drive.mount('/content/driverrr', force_remount=True)
```

Figure 5.11 Mounting Google Drive to Google Colab.

After completing the annotation process, it was necessary to convert the dataset into the YOLOv8-Pose format before training. A Python script was developed to automate this conversion. First, the program ensured that the output label directory existed and loaded the annotation data from a CSV file.

```
# Ensure output label folder exists
os.makedirs(label_folder, exist_ok=True)

# Load CSV file
df = pd.read_csv(csv_path)
```

Figure 5.12 Folder preparation and CSV loading.

Each row of the CSV was processed to extract the corresponding image and its annotated keypoints. The script normalized the keypoint coordinates relative to the image dimensions, assigned a visibility flag, and handled missing values by defaulting to (0,0). Additionally, a bounding box was dynamically calculated around the keypoints.

```
# Calculate bounding box
x_min, x_max = min(x_coords), max(x_coords)
y_min, y_max = min(y_coords), max(y_coords)
bbox_w = x_max - x_min
bbox_h = y_max - y_min
x_center = x_min + bbox_w / 2
y_center = y_min + bbox_h / 2
```

Figure 5.13 Bounding Box Calculation.

Finally, the converted annotations were written into YOLO-compatible .txt files, where each line followed the required format of class: x_center y_center width height keypoints etc. This ensured that the dataset was fully compatible with the YOLOv8-Pose training pipeline.

```
#  Write to YOLO txt file
with open(label_path, "w") as f:
    # Format: class x_center y_center width height keypoints
    yolo_line = "0 " + " ".join(map(str, [x_center, y_center, bbox_w, bbox_h] + keypoints))
    f.write(yolo_line + "\n")
```

Figure 5.14 Write converted annotation to .txt file.

To prepare the dataset for training, the images and their corresponding annotation files needed to be divided into training and validation sets. This was achieved through a Python script that first defined a split ratio of 80% training and 20% validation.

```
# Define train/val split ratio
train_ratio = 0.8  # 80% training, 20% validation

# Create train/val folders
os.makedirs(train_image_folder, exist_ok=True)
os.makedirs(train_label_folder, exist_ok=True)
os.makedirs(val_image_folder, exist_ok=True)
os.makedirs(val_label_folder, exist_ok=True)

# Get all image files
image_files = [f for f in os.listdir(image_folder) if f.endswith(('.jpg', '.png', '.jpeg'))]
random.shuffle(image_files)
```

Figure 5.15 Splitting the dataset.

The image files were shuffled to ensure randomness and prevent bias in the data distribution. Based on the split ratio, the dataset was partitioned into two subsets: training and

validation. Each image was then paired with its corresponding YOLO label file, and both were copied into their respective folders (train/images, train/labels, val/images, val/labels). This structured organization ensured compatibility with the YOLOv8 training pipeline and allowed for effective model evaluation during training.

The training process in YOLOv8 relies on a configuration file (data.yaml) that defines the structure of the dataset. In this project, the train and val parameters specify the directories containing the training and validation images, respectively. Since the task involves detecting only one class: cow back, the parameter nc (number of classes) was set to 1.

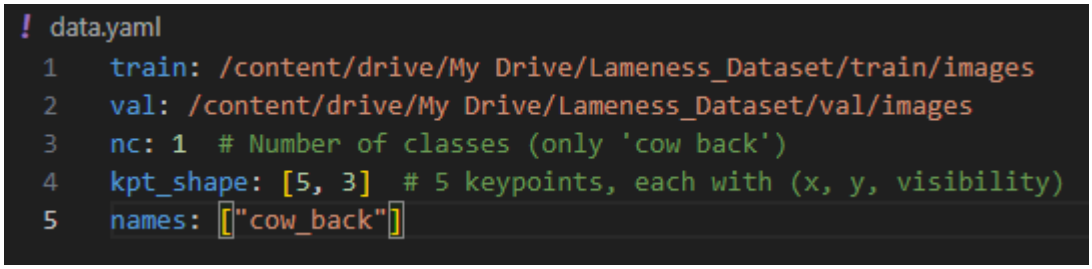


Figure 5.18 Contents of data.yaml file.

For pose estimation, YOLOv8 requires a kpt_shape definition, which indicates the number of keypoints and their attributes. In this case, [5, 3] was used, meaning each instance is represented by 5 keypoints, and each keypoint contains 3 values: x-coordinate, y-coordinate, and visibility. Finally, the names field defines the class label, which in this dataset is "cow_back". This configuration ensures that the YOLOv8 model correctly interprets the dataset during training and evaluation.

After training, the YOLOv8-Pose model produces a best.pt file, which represents the optimized weights of the network. This model was loaded in the inference stage and applied to a set of cow images for lameness detection. The inference pipeline consisted of several steps.

```
model_path = "/content/driverrr/My Drive/LamenessModels/best5.pt"
image_folder = "/content/driverrr/My Drive/Lame Cows"
output_folder = "/content/driverrr/My Drive/Lame Cows/Predictions"
os.makedirs(output_folder, exist_ok=True)

RMSE_THRESHOLD = 2
OUTLIER_THRESHOLD = 100
CLOSE_DISTANCE_THRESHOLD = 10
```

Figure 5.19 Model Path and Configurations.

Code snippet in Figure 5.18 defines the paths to the trained YOLO model (best.pt), the input images, and the output directory where processed results will be saved. It also sets important thresholds:

1. RMSE_THRESHOLD: used to classify cows as “Lame” or “Normal.”
2. OUTLIER_THRESHOLD: discards unrealistic key point distances.
3. CLOSE_DISTANCE_THRESHOLD: merges very close key points to reduce noise.

```
def rmse(y_actual, y_predicted):
    return np.sqrt(np.mean((y_actual - y_predicted) ** 2))
```

Figure 5.20 RMSE function.

Code snippet in Figure 5.19 calculates the RMSE between actual and predicted values. In this system, RMSE quantifies how well a straight line fits the detected key points. A higher RMSE indicates greater deviation from a straight back.

```
# ------------------- LOAD MODEL -------------------
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = YOLO(model_path).to(device)

# ------------------- RUN INFERENCE -------------------
image_paths = [os.path.join(image_folder, f) for f in os.listdir(image_folder)
if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
results = model(image_paths, save=True, conf=0.1)
```

Figure 5.21 Loading model and run inference.

All images from the dataset is loaded and run through the YOLO model for prediction. The conf=0.1 parameter sets a low confidence threshold to ensure all possible detections are considered. YOLO automatically saves annotated outputs. The key points detected are extracted. To ensure data quality, the system checks the distance between consecutive key points. If any distance is too large (greater than OUTLIER_THRESHOLD), the detection is considered invalid and skipped. Furthermore, the key points that are very close to each other are averaged to reduce noise in the predictions. This prevents redundant or jittery detections and makes the regression line more stable.

```
# Average close keypoints
averaged_keypoints = [keypoints[0]]
for i in range(1, len(keypoints)):
    distance = np.linalg.norm(keypoints[i] - keypoints[i-1])
    if distance < CLOSE_DISTANCE_THRESHOLD:
        averaged_point = np.mean([keypoints[i], keypoints[i-1]], axis=0)
        averaged_keypoints[-1] = averaged_point
    else:
        averaged_keypoints.append(keypoints[i])
```

Figure 5.22 Averaging key points.

A linear regression model is fitted to the detected key points, approximating the cow's back as a straight line. The RMSE is then calculated between the actual points and the regression line. If the RMSE is above the set threshold, the cow is classified as Lame. Otherwise, it is classified as Normal.

```
X = np.array([pt[0] for pt in averaged_keypoints]).reshape(-1, 1)
Y = np.array([pt[1] for pt in averaged_keypoints])
if len(X) < 2:
    print(f"⚠ {filename}: Too few points after averaging.")
    continue

model_line = LinearRegression().fit(X, Y)
Y_pred = model_line.predict(X)
error = rmse(Y, Y_pred)
```

Figure 5.23 RMSE calculation.

Finally, the processed results are visualized. Key points are drawn as circles, and the RMSE value along with the classification status is overlaid on the image. The results will be further discussed in the next chapter.

```
for x, y in averaged_keypoints:
    cv2.circle(image, (int(x), int(y)), 5, (0, 255, 0), -1)

# Put RMSE value and status
status = "Lame" if error > RMSE_THRESHOLD else "Normal"
cv2.putText(image, f"RMSE: {error:.2f}  {status}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9,
            (0, 0, 255) if error > RMSE_THRESHOLD else (0, 255, 0), 2)
```

Figure 5.24 Visualize RMSE output.

5.2 Cow Recognition Model

The cow recognition system was developed by first preparing a dataset of cow images and splitting it into training and validation sets to ensure fair evaluation. The dataset is divided into training (80%) and validation (20%) sets. Each cow has its own folder, and images are shuffled before splitting.

```
def split_dataset(source=r"C:\Users\user\Downloads\simula
    source = Path(source)
    target = Path(target)
    train_dir, val_dir = target/"train", target/"val"
    train_dir.mkdir(parents=True, exist_ok=True)
    val_dir.mkdir(parents=True, exist_ok=True)

    random.seed(42)
```

Figure 5.25 Dataset Splitting.

Embedding-based approach was used in this model instead of relying on standard classification, where each cow is represented as a vector in a high-dimensional space. To make these embeddings more discriminative, the ArcFace loss function was applied, which increases the separation between cows with similar appearances.

```python
class ArcMarginProduct(nn.Module):
    def __init__(self, in_features, out_features, s=30.0, m=0.70):
        super().__init__()
        self.weight = nn.Parameter(torch.FloatTensor(out_features, in_features))
        nn.init.xavier_uniform_(self.weight)
        self.s = s
        self.m = m

    def forward(self, embeddings, labels):
        cosine = F.linear(F.normalize(embeddings), F.normalize(self.weight))
        theta = torch.acos(torch.clamp(cosine, -1.0 + 1e-7, 1.0 - 1e-7))
        one_hot = torch.zeros_like(cosine)
        one_hot.scatter_(1, labels.view(-1, 1), 1)
        logits = cosine * (1 - one_hot) + torch.cos(theta + self.m) * one_hot
        return logits * self.s
```

Figure 5.26 ArcFace layer.

The recognition model uses a ResNet18 backbone (pretrained on ImageNet) to extract features. These features are projected into a 512-dimensional embedding space, and the ArcFace layer ensures they are well separated by identity.

```python
class CowArcFaceNet(nn.Module):
    def __init__(self, num_classes, embedding_size=512):
        super().__init__()
        self.backbone = timm.create_model("resnet18", pretrained=True, num_classes=0)
        self.embedding = nn.Linear(self.backbone.num_features, embedding_size)
        self.arcface = ArcMarginProduct(embedding_size, num_classes)
```

Figure 5.27 Cow Recognition model.

During training, the model was optimized using the Adam optimizer with a learning rate of 1e-3, while loss was measured using CrossEntropy combined with ArcFace logits. Training progress was monitored through two key metrics:

1. average training loss per epoch
2. validation Rank-1 accuracy, which measures the percentage of cows correctly matched to themselves

Model checkpointing was implemented to automatically save the best-performing model weights as cow_arcface.pth whenever an improvement in validation accuracy was observed.

```python
def train_model(train_loader, val_loader, num_classes, device, epochs=10):
    model = CowArcFaceNet(num_classes).to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
    criterion = nn.CrossEntropyLoss()

    best_acc = 0
    for epoch in range(1, epochs+1):
        model.train()
        total_loss = 0
        for imgs, labels in train_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            logits, _ = model(imgs, labels)
            loss = criterion(logits, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

        avg_loss = total_loss / len(train_loader)
        acc = evaluate(model, val_loader, device)
        print(f"Epoch {epoch}/{epochs} | Loss {avg_loss:.4f} | Val Rank-1 {acc:.4f}")

        if acc > best_acc:
            best_acc = acc
            torch.save(model.state_dict(), "cow_arcface.pth")
            print(f"✅ Saved new best model (Val Rank-1 {acc:.4f})")

    return model
```

Figure 5.28 Training Loop.

```python
def evaluate(model, val_loader, device):
    model.eval()
    embeddings, labels = [], []
    with torch.no_grad():
        for imgs, lbls in val_loader:
            emb = model(imgs.to(device))
            embeddings.append(emb.cpu())
            labels.append(lbls)
    embeddings = torch.cat(embeddings)
    labels = torch.cat(labels)

    sims = torch.mm(embeddings, embeddings.t())
    preds = sims.argmax(dim=1)
    correct = (labels[preds] == labels).float().mean().item()
    return correct
```

Figure 5.29 Evaluation (Rank-1 Accuracy).

5.3 API Set up

The backend of the system was implemented using FastAPI, chosen for its lightweight and high-performance capabilities. It provides the REST API endpoints that connect the trained machine learning models with the mobile application. Firebase was integrated as a cloud storage solution, where annotated cow images and gallery data are stored. Upon initialization, the backend authenticates with Firebase using a service account key and rebuilds the

embedding gallery to enable cow recognition. The backend system was deployed on Google Cloud Platform to ensure scalability, availability, and integration with Firebase services. Using Google Cloud Run, the FastAPI application was containerized and deployed as a serverless service, enabling automatic scaling based on request load while reducing infrastructure overhead.

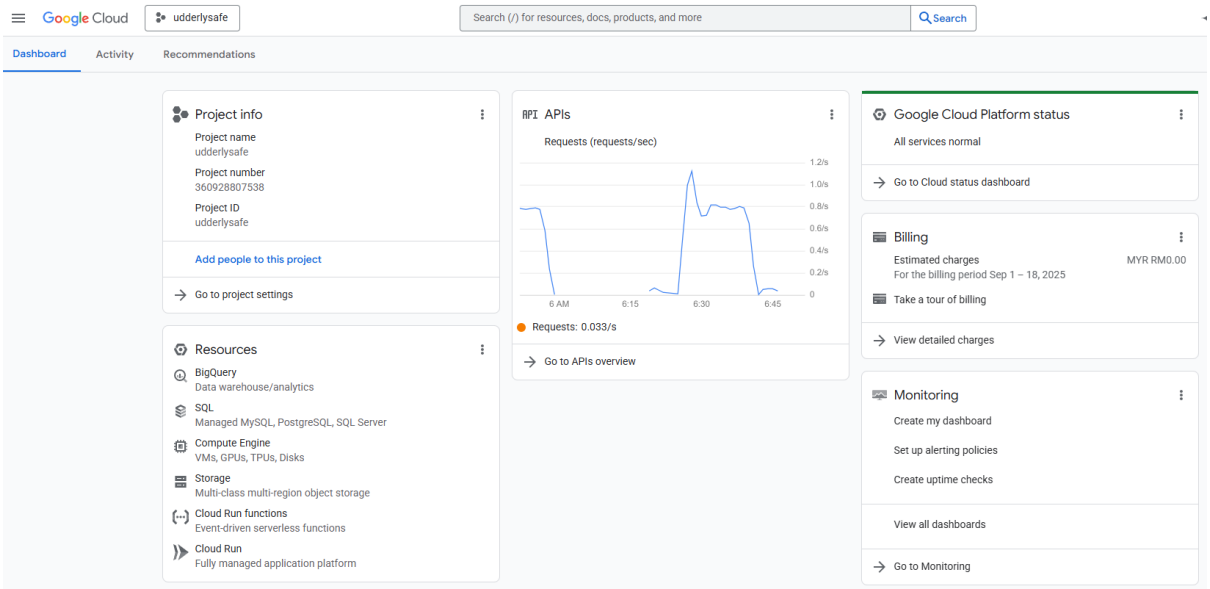


Figure 5.30 Project's Google Cloud Run dashboard.

```
# ---------------- FIREBASE ----------------
if not firebase_admin._apps:
    cred = credentials.Certificate("serviceAccountKey.json")
    firebase_admin.initialize_app(cred, {"storageBucket": "udderlysafe.firebasestorage.app"})
```

Figure 5.31 Firebase Initialization.

The key backend endpoint is /analyze, which combines both lameness detection and cow recognition into a single request. When an image is uploaded, the backend first processes it through the lameness detection model, which evaluates the cow's back and returns a health status. The same image is then passed to the embedding recognition module, which compares extracted embeddings against the gallery stored in Firebase to identify the individual cow. The endpoint returns a structured JSON response containing both the lameness result and the identity matches, allowing the mobile application to obtain health and recognition insights in one step. This design reduces the need for multiple API calls, making the system more efficient and user-friendly.

```python
@app.post("/analyze")
async def analyze(file: UploadFile = File(...), k: int = 3):
    try:
        await EMBEDDING_READY.wait()

        # Read file once
        contents = await file.read()

        # Create UploadFile-like objects for reuse
        file_for_lameness = UploadFile(filename=file.filename, file=BytesIO(contents))
        file_for_embedding = UploadFile(filename=file.filename, file=BytesIO(contents))

        # Call the existing route functions directly
        lameness_result = await detect_lameness(file_for_lameness)
        embedding_result = await recognize_cow(file_for_embedding, k=k)

        return {
            "lameness": lameness_result,
            "embedding": embedding_result
        }
    except Exception as e:
        tb = traceback.format_exc()
        print("❌ /analyze error:", e, tb)
        return {"status": "Error", "message": str(e)}
```

Figure 5.32 /analyze endpoint.

```python
@app.get("/gallery_status")
async def gallery_status():
    return GALLERY_STATUS
```

Figure 5.33 /gallery_status endpoint.

In addition to the analysis endpoint, the backend also provides a /gallery_status endpoint. This endpoint returns metadata about the embedding gallery, including the status and the timestamp of the last rebuild. By exposing this information, users can confirm that the recognition system is working with the most up-to-date embeddings, ensuring reliability and transparency in cow identification. Through this pipeline, the backend acts as the central decision layer, handling image preprocessing, model inference, and returning structured responses to the mobile frontend in real time. This modular design ensures scalability, enabling new analysis models or recognition methods to be integrated in future iterations of the system.

5.4 Mobile Application

Firebase was set up as the backend for the mobile application to handle data storage, authentication, and communication between the app and the trained YOLO model. A dedicated Firebase project was created, and the mobile app was registered to securely connect with Firebase services. The backend utilized Cloud Firestore to store lameness detection results and metadata, and Firebase Storage to save raw and processed cow images.

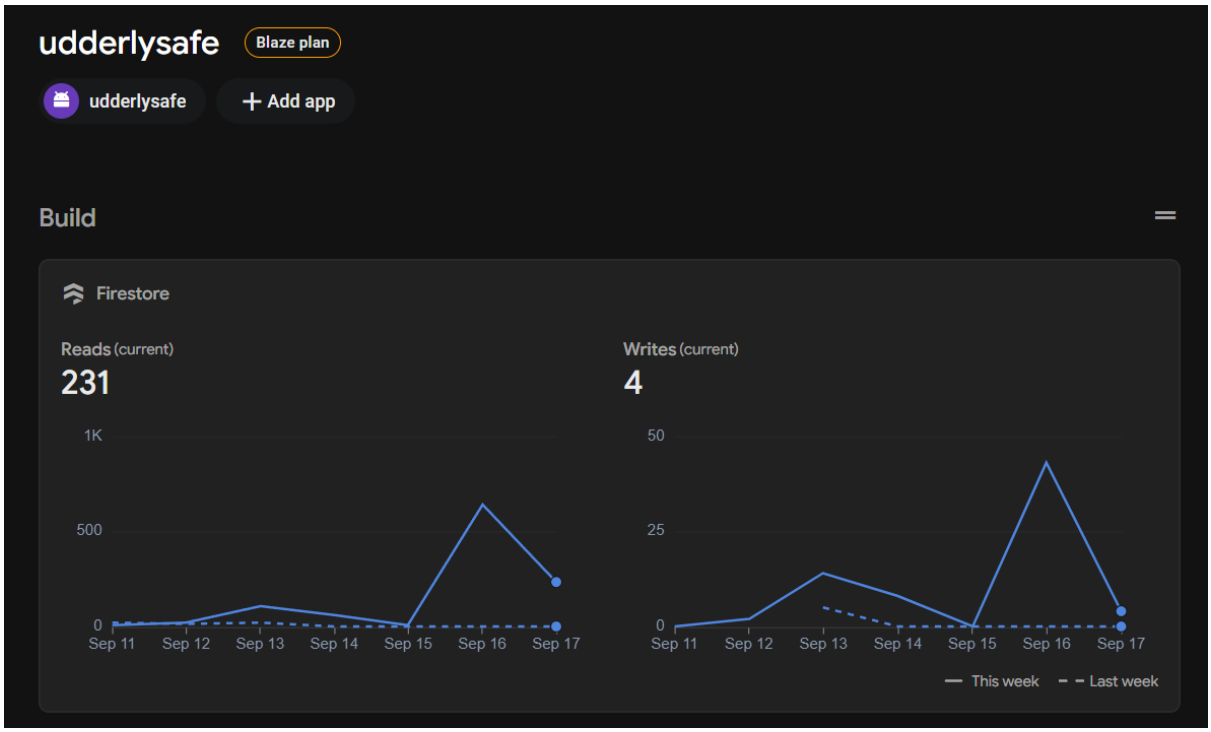


Figure 5.34 Firebase Storage.

```
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if true;
    }
  }
}
```

Figure 5.35 Permission Configuration

.

Configuration in Figure 5.35 grants open access to all files within the storage bucket, allowing both read and write operations without authentication. While this setup simplifies testing and ensures smooth data transfer during development, it is not secure for production

use. In a live deployment, stricter rules would be implemented to restrict access to authenticated users only, ensuring that only authorized farm staff or veterinarians can upload images and view results. After permission is set, we can go ahead and establish connection with the mobile app.

```
void main() async{
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  final storage = FirebaseStorage.instance;
  print('Bucket name: ${storage.bucket}');
  runApp(MyApp());
}
```

Figure 5.36 Initializing firebase in mobile app.

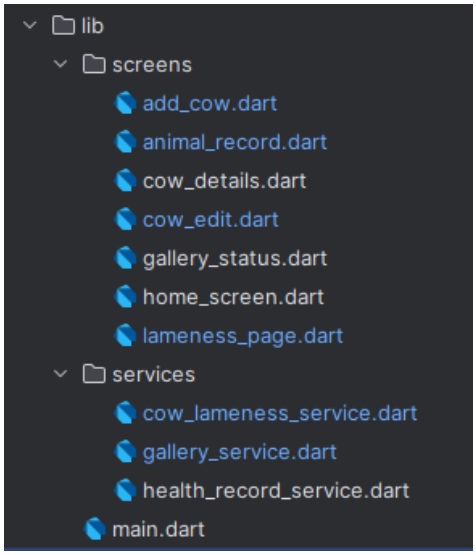


Figure 5.37 List of screens and services used in app.

Figure 5.37 shows the list of screens and services used in the mobile app. The functions of the screens are as the name suggests, for example, add_cow.dart handles the form that allows user fill in cow details, cow_details.dart shows the cow's detail page. Services like gallery_service.dart provides the backend code and pass data to screens. Gallery_service calls the /gallery_status endpoint from the API and the returned data are displayed by gallery_status.dart. The complete code of the mobile application can be found in this GitHub repository: https://github.com/icantfindmyhair/udderlysafe

Chapter 6

System Evaluation And Discussion

6.1 Model Testing and Verification

6.1.1 Lameness Detection Model

6.1.1.1 Model Testing

The dataset is split into training set (80%) and validation set (20%). The training set will then be used to train the YOLOv8-pose model. In order to maximize the cow key point detection model's performance, various training parameters were tested in this study. In these experiments, the input image size, batch size, and number of epochs were manipulated. Table 6.1 provides a summary of the experiments' outcomes.

Table 6.1 Summary of training results using different parameters.

Parameters			Box				Pose			
Epoch	Batch	Image Size	Precision (P)	Recall (R)	mAP50	mAP50-95	Precision (P)	Recall (R)	mAP50	mAP50-95
50	-	640	0.628	0.572	0.611	0.294	0.721	0.699	0.784	0.600
150	8	800	0.830	0.683	0.813	0.521	0.830	0.781	0.893	0.793
200	8	640	0.816	0.820	0.876	0.586	0.869	0.905	0.949	0.858
250	4	800	0.906	0.825	0.896	0.652	0.913	0.852	0.959	0.860
300	4	640	Training stopped early as best results observed at epoch 10 with mAP scores peaked around 0.400-0.500							

- Box – bounding box
- Pose – key points
- Precision (P) – predicted results that are actually correct
- Recall (R) – actual correct results that are successfully predicted
- mAP50 – overall accuracy when predictions overlap the true label by at least 50%
- mAP50-95 – overall accuracy averaged across different overlap thresholds from 50% to 95%

The results of the training experiments showed that hyperparameter configurations like picture size, batch size, and epoch count had an important effect on the model's performance. Underfitting was evident in shorter runs (e.g., 50 epochs, 640×640), but detection accuracy and mAP scores were greatly enhanced by utilizing higher image sizes and increasing epochs (e.g., 150–200 epochs, 800×800). In order to balance learning capacity and detail, the best results were obtained at 250 epochs with 800×800 images and batch size 4. Nevertheless, early stopping occurred with larger increases (e.g., 300 epochs), indicating overfitting or convergence. Overall, important point detection was enhanced by adjusted parameters; however, larger datasets and more optimization could yield even greater improvements.

However, training at higher epochs (200 and above) frequently resulted in early stopping or convergence as the dataset size grew, with no performance improvement. This suggests that more training increased the likelihood of overfitting because the model had already learnt the required features in fewer epochs. In order to achieve a suitable balance between adequate learning and training efficiency, future studies were standardized at 150 epochs. The model is trained 5 more times with the same parameters, with different training and validation set split each time to ensure the results are stable. Table 6.2 shows the results of the experiment. As seen in the table, run 5 gives the highest accuracy. Therefore, the best.pt file from run 5 will be used in our final pipeline.

Table 6.2 Summary of training result using 150 epochs.

	Box				Pose			
Run	P	R	mAP50	mAP50-95	P	R	mAP50	mAP50-95
1	0.926	0.888	0.935	0.645	0.95	0.919	0.968	0.855
2	0.878	0.857	0.900	0.496	0.884	0.897	0.939	0.708
3	0.934	0.909	0.95	0.683	0.953	0.932	0.975	0.893
4	0.881	0.847	0.879	0.475	0.903	0.894	0.94	0.714
5	0.935	0.893	0.946	0.688	0.951	0.915	0.975	0.897

6.1.1.2 Model Validation

As previously stated, the best.pt model from Run 5 was chosen for deployment because, with a mAP50–95 score of 0.897, it had the highest pose detection accuracy. Examples of key point detections and the associated RMSE calculations generated by this model are shown in Figure 6.1. Although the accuracy is normally high, Figure 6.2 illustrates several false positives, in which healthy cows were mistakenly labelled as lame.

Figure 6.1 Example correct output of key point and RMSE.

Figure 6.2 Example of healthy cows that are detected as lame.

Overall, the validation shows that the model generalizes rather well under many scenarios and achieves good accuracy. Misclassifications still happen, nevertheless, especially when healthy cows are mistakenly identified as lame. This is likely due to the limited availability of cow images and the reliance on edited samples to simulate lameness. Given these limitations, the current performance is the best that can be achieved with the dataset that is currently

available. Additional improvements would necessitate the use of more diverse, actual farm-collected photos.

6.1.2 Cow Recognition Model

6.1.2.1 Model Testing

The model was trained for 10 epochs using a ResNet-18 backbone with ArcFace loss. Table 6.3 presents the training loss and validation Rank-1 accuracy across epochs. At the beginning of training (Epoch 1), the loss was relatively high (15.6635), reflecting that the model initially struggled to discriminate between individual cows. Nevertheless, the validation Rank-1 accuracy started at 58.79%, which indicates that the model had already learned some useful representations.From Epochs 2–4, the training loss dropped steeply (from 10.2372 to 6.7379), while validation Rank-1 accuracy improved, peaking at 63.67% in Epoch 3. This demonstrates that the ResNet-18 backbone, combined with ArcFace, was able to learn discriminative embeddings within just a few iterations.

Table 6.3 ArcFace Loss in 10 Epochs.

Epoch	Loss	Rank-1
1	15.6635	0.5879
2	10.2372	0.6098
3	8.5370	0.6367
4	6.7379	0.6341
5	6.0056	0.6021
6	5.6599	0.5854
7	5.5176	0.5866
8	5.3594	0.5828
9	5.2190	0.5879
10	4.7815	0.5841

However, after Epoch 4, the model's performance plateaued. Validation Rank-1 accuracy fluctuated slightly between 58–63%, while the training loss continued to decrease more gradually, reaching 4.7815 by Epoch 10. This divergence suggests that while the model continued optimizing on the training set, it did not achieve further generalization gains on the validation set.

These results highlight that the ResNet-18 + ArcFace model was capable of moderately effective cow identification, but its performance was limited by factors such as dataset size, image diversity, or possible overfitting. A ResNet-50 backbone was also tested for 20 epochs. While it achieved strong ROC–AUC values, its Equal Error Rate (EER) plateaued around Epoch 12, indicating limited further improvement. This highlights that increasing model depth alone was insufficient, and that dataset scale and diversity likely constrain performance.

6.1.2.2 Model Validation

To assess the quality of the learned embeddings, we conducted several analyses beyond loss and Rank-1 accuracy. These included dimensionality reduction visualizations, similarity heatmaps, and distribution analysis of between-cow similarities.

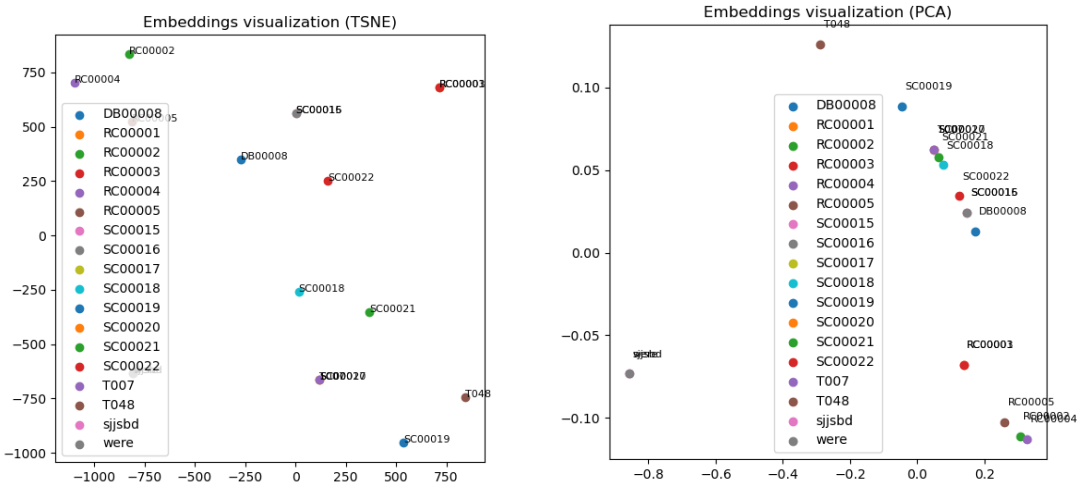


Figure 6.3 t-SNE and PCA Embedding visualization.

t-SNE (t-distributed Stochastic Neighbor Embedding) and PCA (Principal Component Analysis) visualizations are dimensionality reduction techniques that project high-dimensional embeddings into 2D space, enabling visual inspection of how well the model separates different cow identities [31]. The t-SNE plot in Figure 6.3 shows many cow identities clustering together, indicating that the model learns discriminative features. The PCA projection emphasizes global structure and shows tighter grouping of cows. While several identities remain distinguishable, there are some overlaps, particularly among visually similar cows, suggesting cases where identities are harder to separate [32].

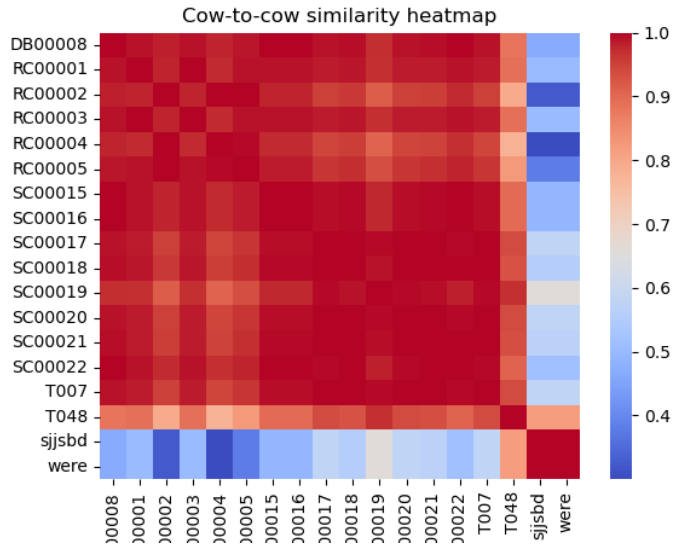


Figure 6.4 Cow-to-cow similarity heatmap.

The similarity heatmap reinforces this observation: while most cows show high within-class similarity, certain pairs display unexpectedly high cross-cow similarity, pointing to near-duplicate embeddings.

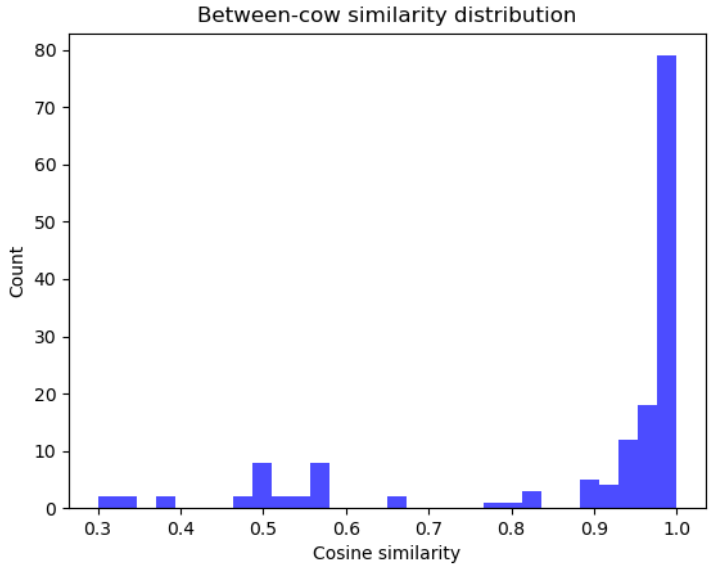


Figure 6.5 Between-cow similarity distribution.

This is further supported by the between-cow similarity distribution, which shows a strong peak at high cosine similarity values (close to 1.0). Such a distribution suggests that while many identities are well-separated, a subset of cows remain difficult to distinguish due to visual resemblance or limited variability in the dataset.

These findings suggest that the model is capable of learning discriminative representations, but its performance plateau (including Rank-1 accuracy and Equal Error Rate using ResNet50 stabilizing after ~12 epochs) indicates that dataset scale and visual similarity among cows constrain further improvements. In practice, this means the system performs reliably for many individuals but remains challenged by visually ambiguous cases.

6.2 API Testing

At this stage, we are working on combining both lameness detection and cow recognition into a single pipeline. Before deploying the API to Google Cloud Run, we conducted thorough testing in a local development environment. The FastAPI application was run using Uvicorn, which started a server accessible at http://127.0.0.1:8000/docs. This endpoint provided an automatically generated interactive documentation page (Swagger UI), allowing us to explore and test all API endpoints.

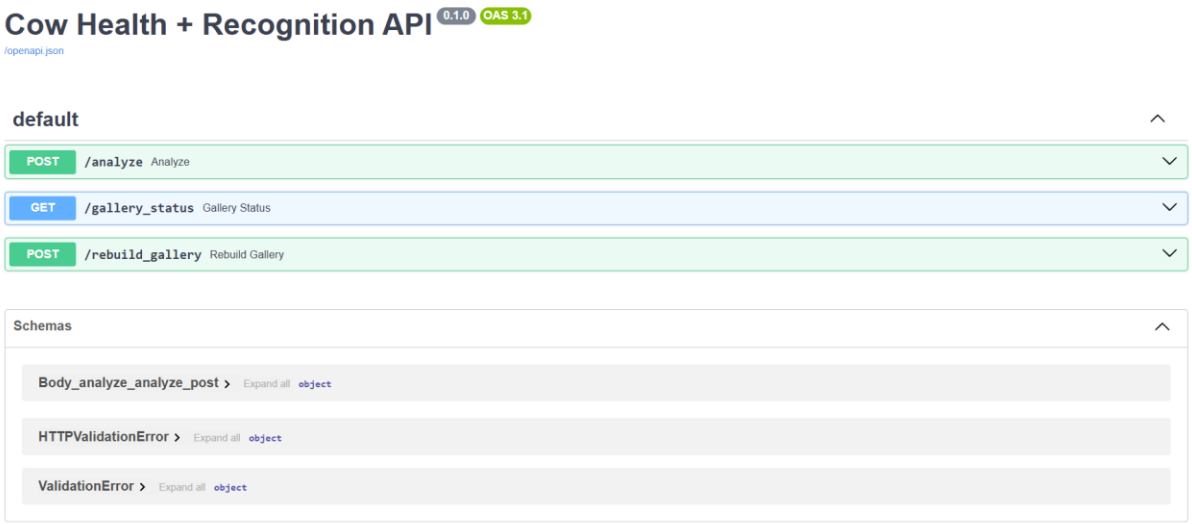


Figure 6.6 Local API in Swagger UI.

Through local testing, we confirmed that the server launched properly, endpoints operated as planned, and responses including those involving embedding analysis and image uploads were accurate. We used Docker to bundle all dependencies and configurations consistently after locally verifying the application's functionality. Following that, the Docker image was made available to Cloud Run, which served as a public endpoint and made sure the API retained its validated functionality while opening to the public.

200

Response body


```
{
  "lameness": {
    "score": 0.13299627602100372,
    "status": "healthy",
    "keypoints": [
      {
        "x": 147.73448181152344,
        "y": 121.17138671875
      },
      {
        "x": 178.11837768554688,
        "y": 121.0508041381836
      },
      {
        "x": 209.4471893310547,
        "y": 121.37672424316406
      },
      {
        "x": 239.53955078125,
        "y": 121.28594207763672
      },
      {
        "x": 270.233154296875,
        "y": 121.03178405761719
      }
    ],
    "image_url": null
```


Figure 6.7 API Response Body in JSON format.

6.3 Mobile Application

6.3.1 Home screen

To complement the backend API, a mobile application is developed to provide an intuitive interface for managing the herd, performing lameness detection, and accessing cow recognition features. The home screen appears when the user first launches the mobile application. The total number of cows in the herd as of the time of app launch is shown on the home screen and is taken from the database. Three main buttons are available for navigation: Animal Record, Lameness Detection, and Export Report, providing quick access to the core features of the application. The embedding gallery status, which shows the present condition of the cow recognition database and enables users to rebuild the gallery whenever necessary, may be found by scrolling to the bottom of the main page. This design guarantees that users can monitor the recognition system's status and quickly access essential functions.

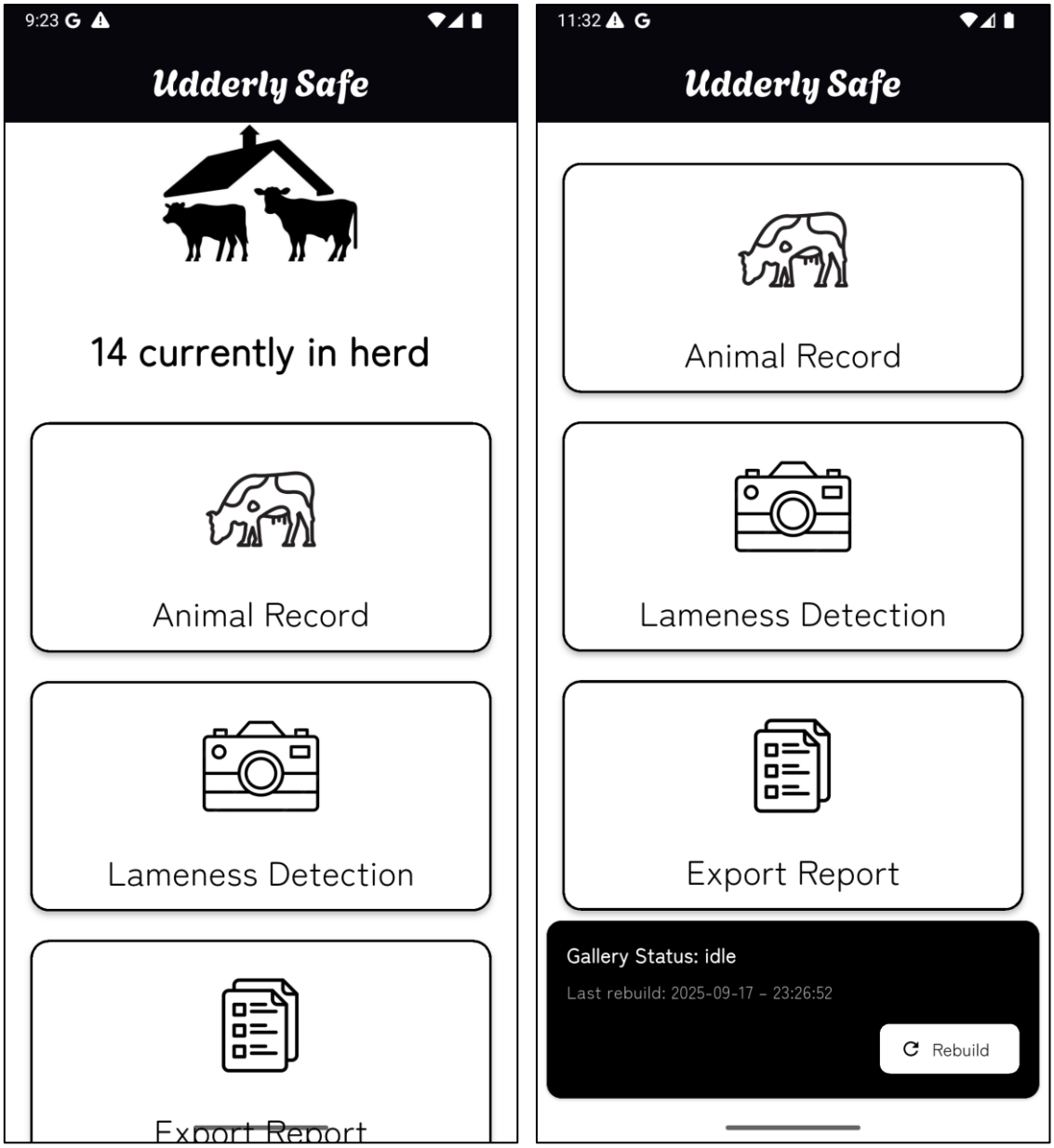


Figure 6.8 Home screen of the mobile application.

6.3.2 Animal Record

The user is shown a scrollable grid with all of the cows in the database when they click the Animal Record button. Every cow is represented by a picture, with its ID shown beneath the picture. Additionally, cow IDs can be used for searching and filtering a particular cow. When you select a cow's image, you are taken to the cow details page, which shows all the cow's pictures along with its basic details and medical history. Users can update or remove the cow from this page, giving them complete control over how each record is managed.

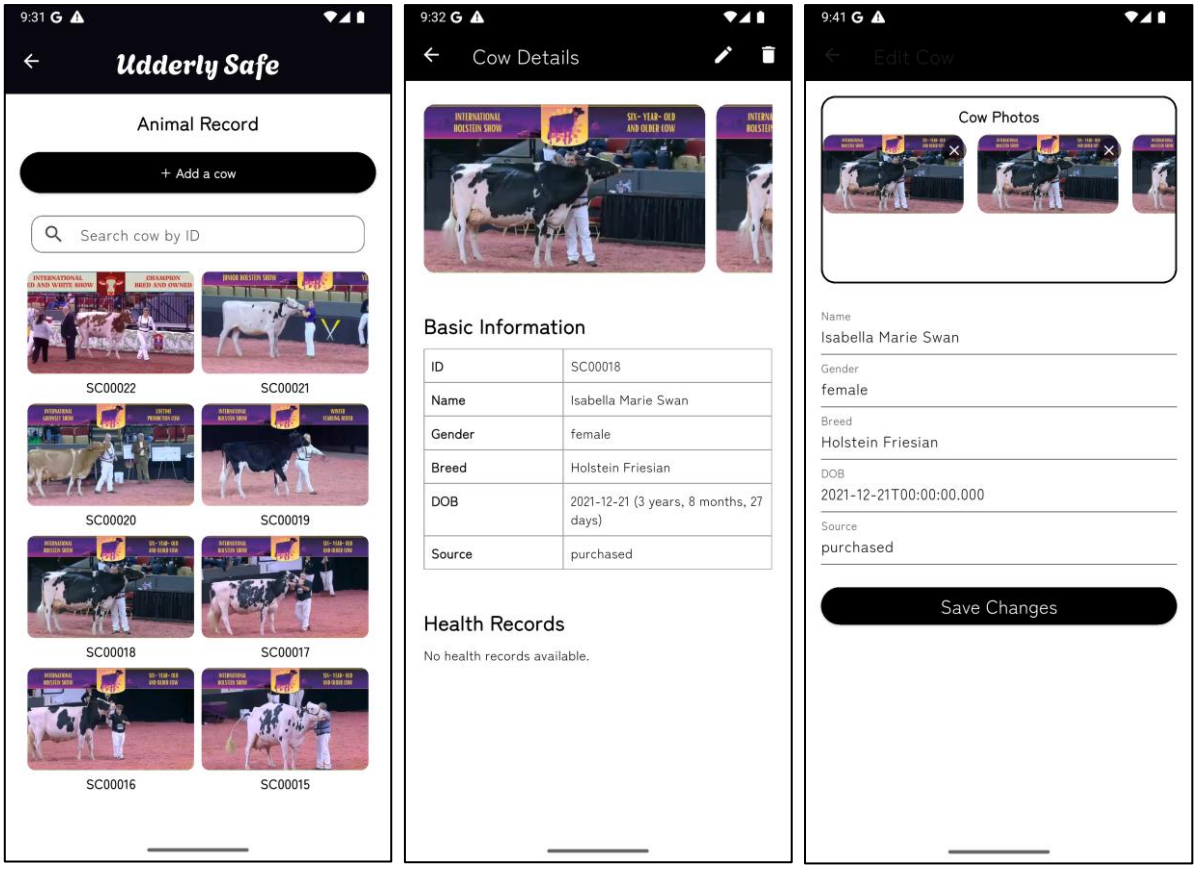


Figure 6.9 Animal Record Interface.

There is also a "Add a Cow" button on the Animal Record screen, as seen in Figure 6.9. When this button is tapped, an empty form appears (Figure 6.10), allowing the user to enter the cow's ID, name, breed, gender, date of birth, source, and submit multiple pictures. The animal record grid is instantly updated when the form is submitted since Firebase receives the data and adds the new cow record to the database. With this capability, customers may effectively maintain and grow the herd database right from the mobile app.

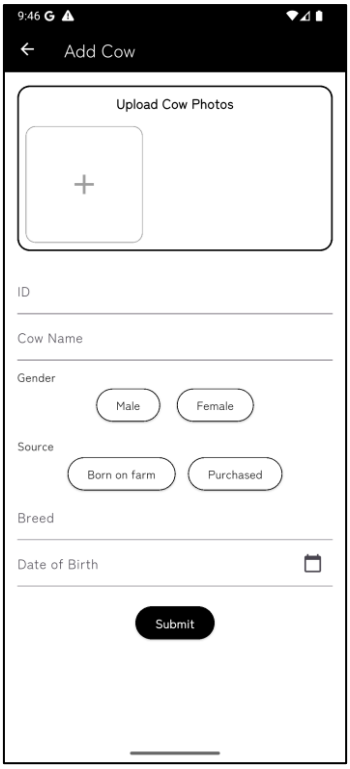


Figure 6.10 Add a cow page.

6.3.2.1 Cow record CRUD Operations

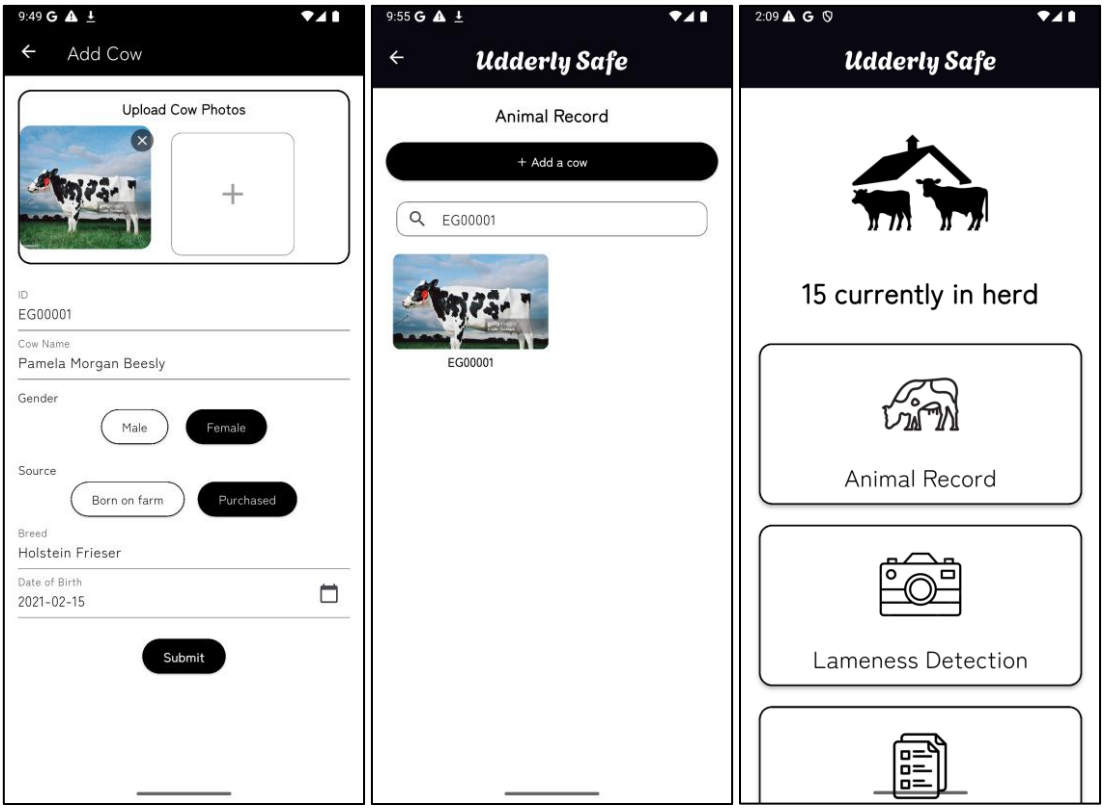


Figure 6.11 Adding a new cow.

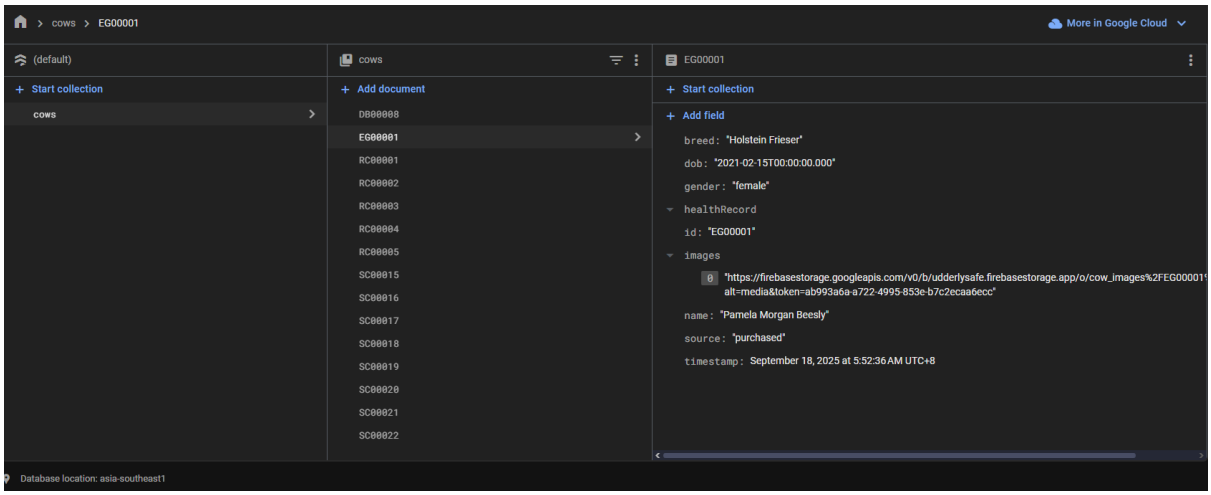


Figure 6.12 Data of new cow shows in Firestore database.

6.3.2.2 View Cow Details

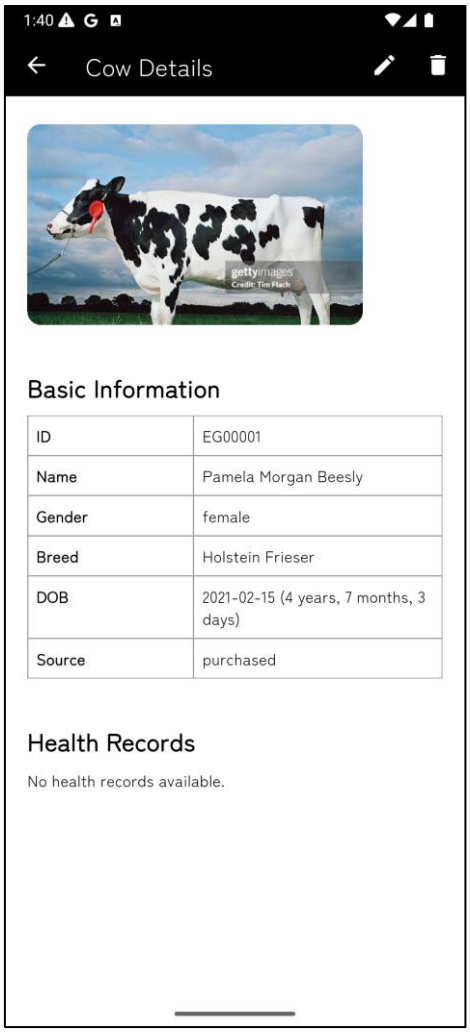


Figure 6.13 Cow details.

6.3.2.3 Edit Cow Details

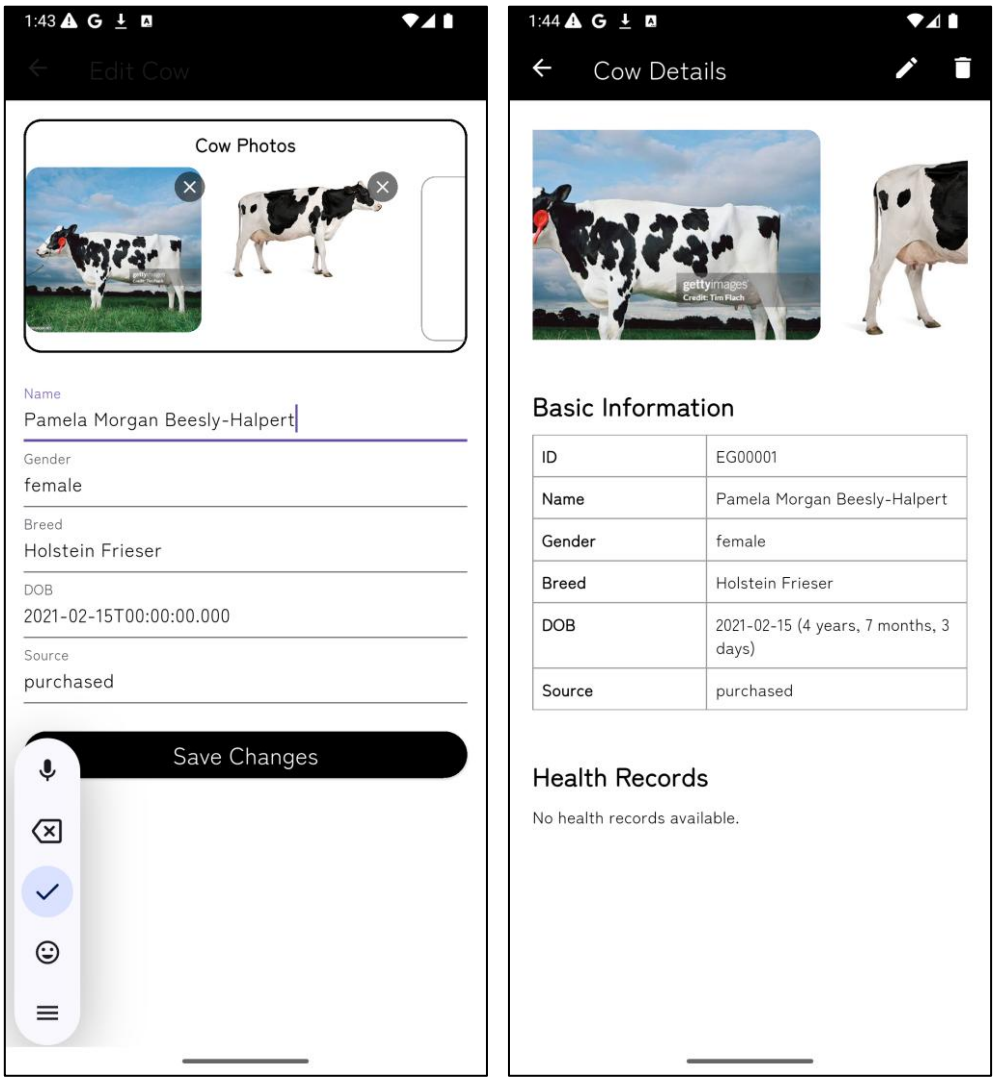


Figure 6.14 Editing the cow's name and add a new image.

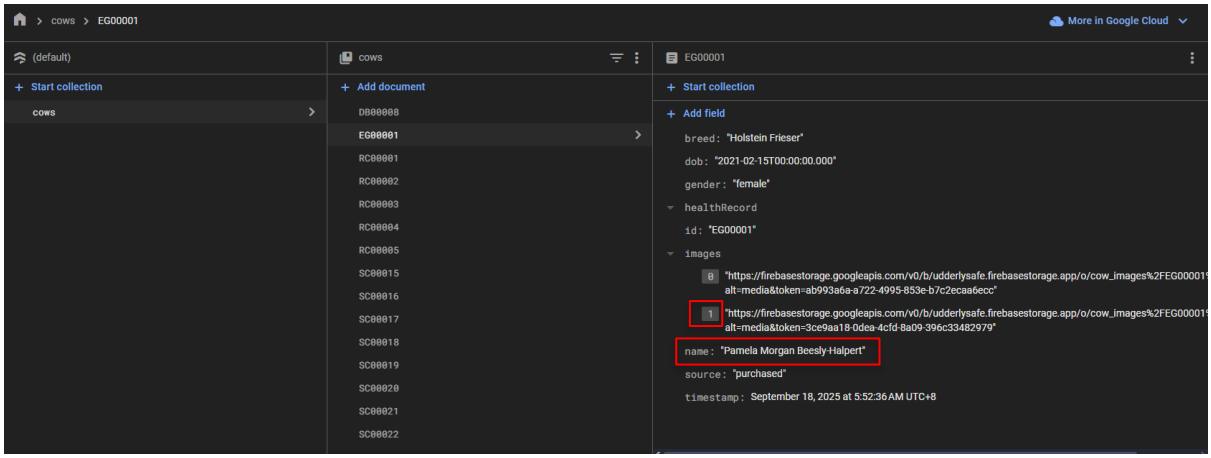


Figure 6.15 Updated Firebase.

6.3.2.4 Delete Cow Details

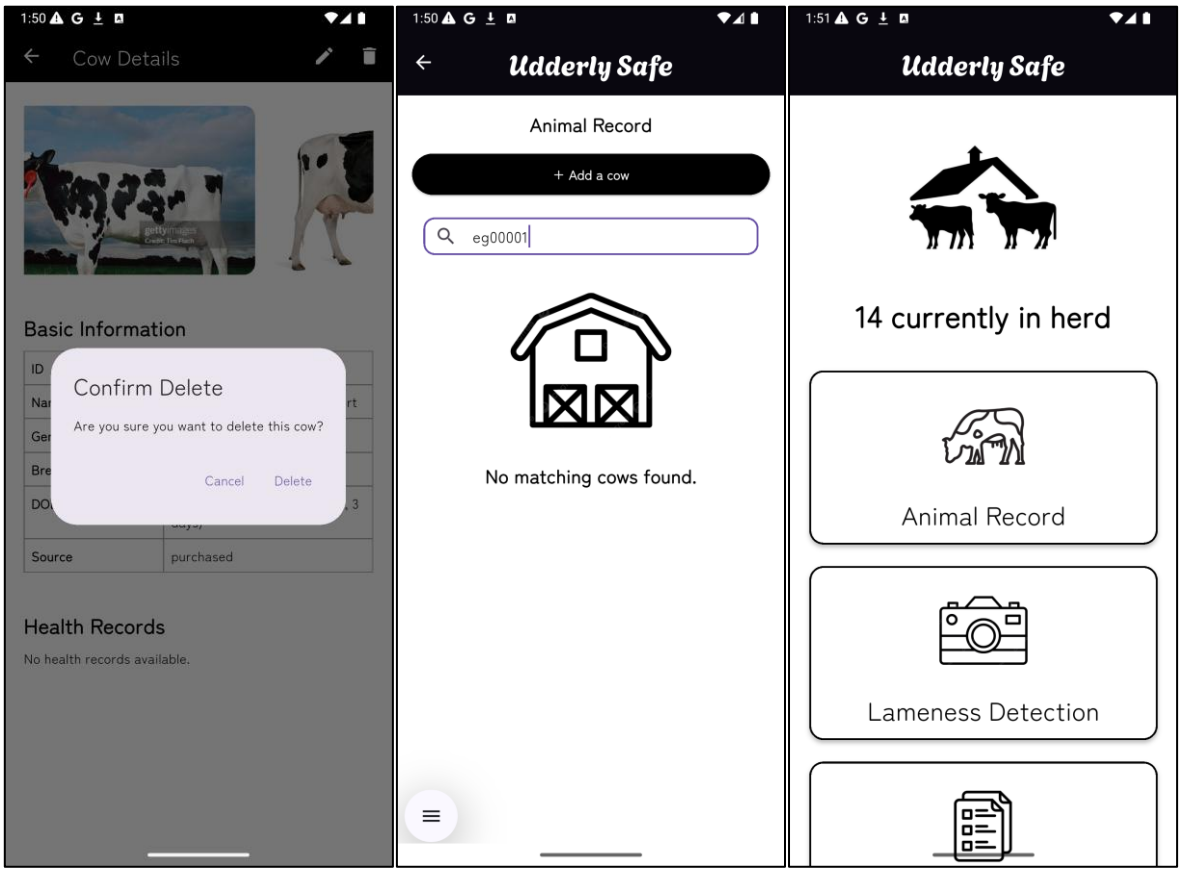


Figure 6.16 Deleting a cow.

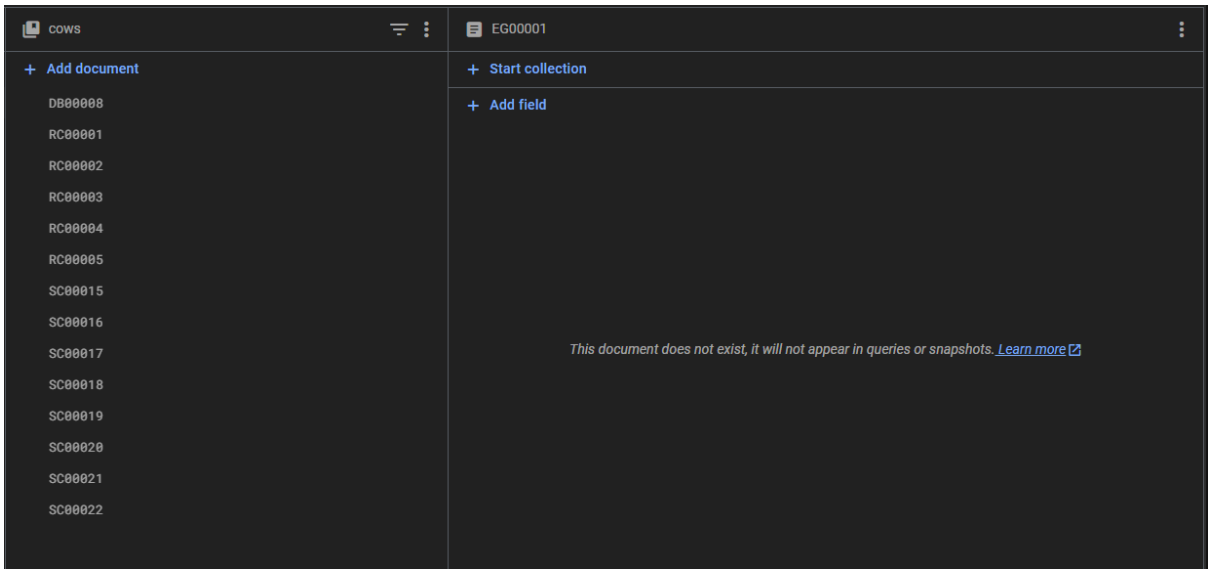


Figure 6.17 Cow data deleted from database.

6.3.3 Lameness Detection and Health Record

The Lameness Detection feature allows the user to detect lameness in a cow by either taking a photo or uploading an existing horizontal image of the cow's back. The image must be horizontal for the detection algorithm to function correctly. Once the detection is completed, the result is displayed on the screen, and the user has the option to add the result to the cow's health record. The process is designed to be straightforward and user-friendly, enabling quick and easy recording of lameness observations.

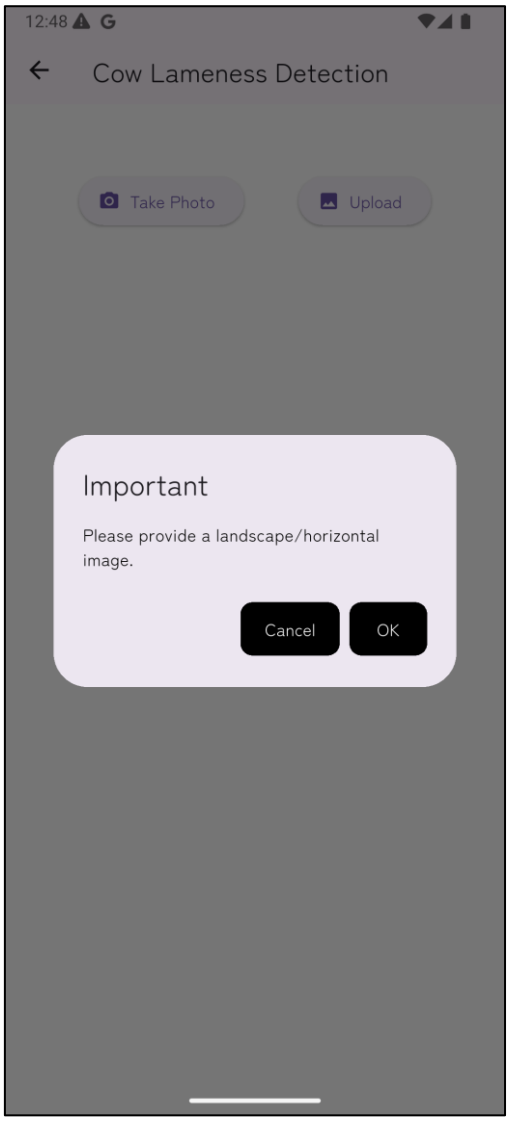


Figure 6.18 Lameness detection page warning before uploading photo.

After the user captures or uploads a cow photo, the system displays the predicted lameness status along with the cow recognition result. Users can decide if they want the key points to be displayed on the image. To ensure accuracy in herd management, the user is given the option

to manually confirm or correct the predicted cow ID before submitting it to the health record. This hybrid design between automatic recognition with human confirmation balances automation with reliability, making the system usable even with imperfect datasets. As more farm-specific data is collected, the recognition model can continue to improve, and reliance on manual confirmation will decrease.

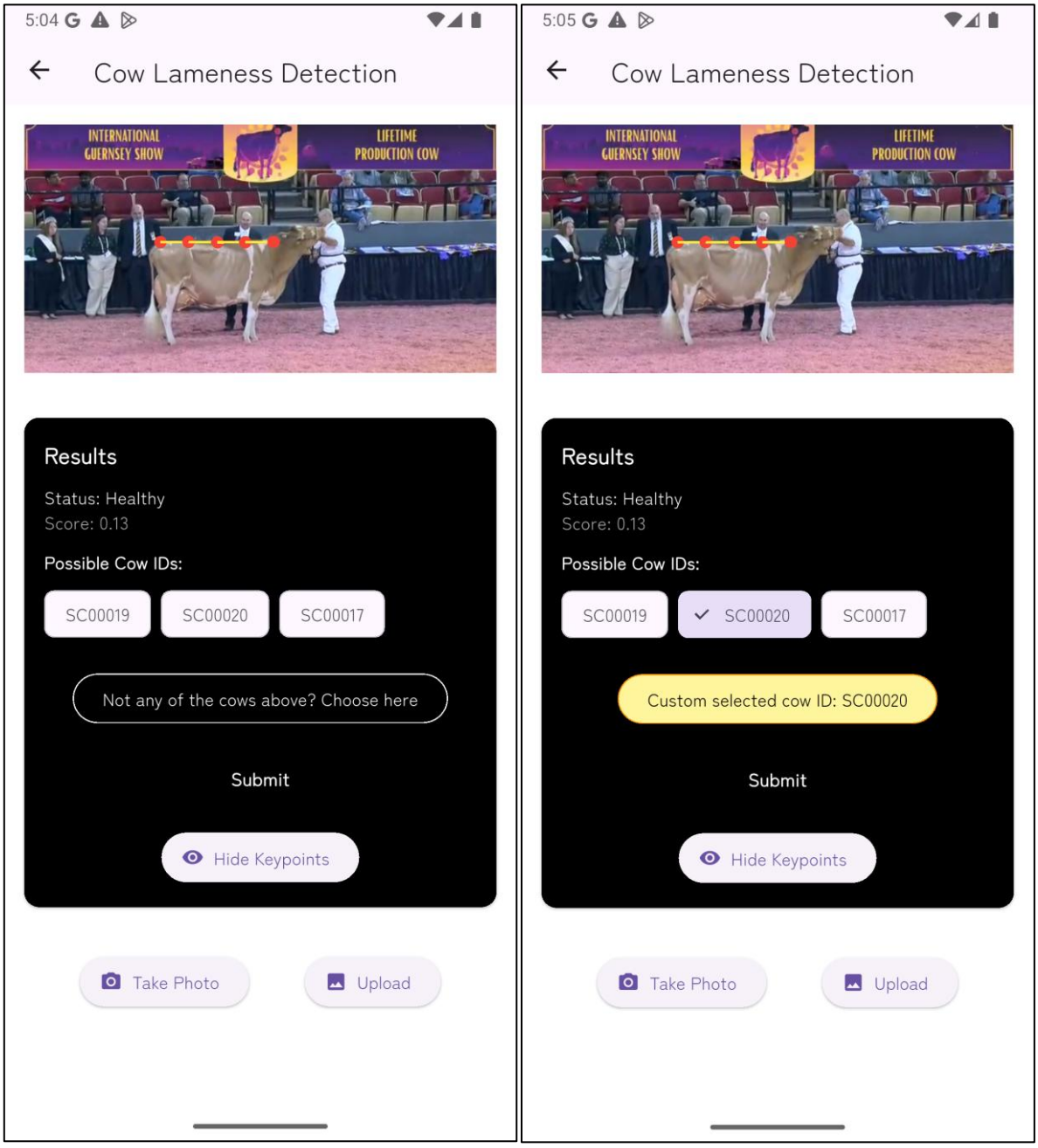


Figure 6.19 Example result of non-lame cow.

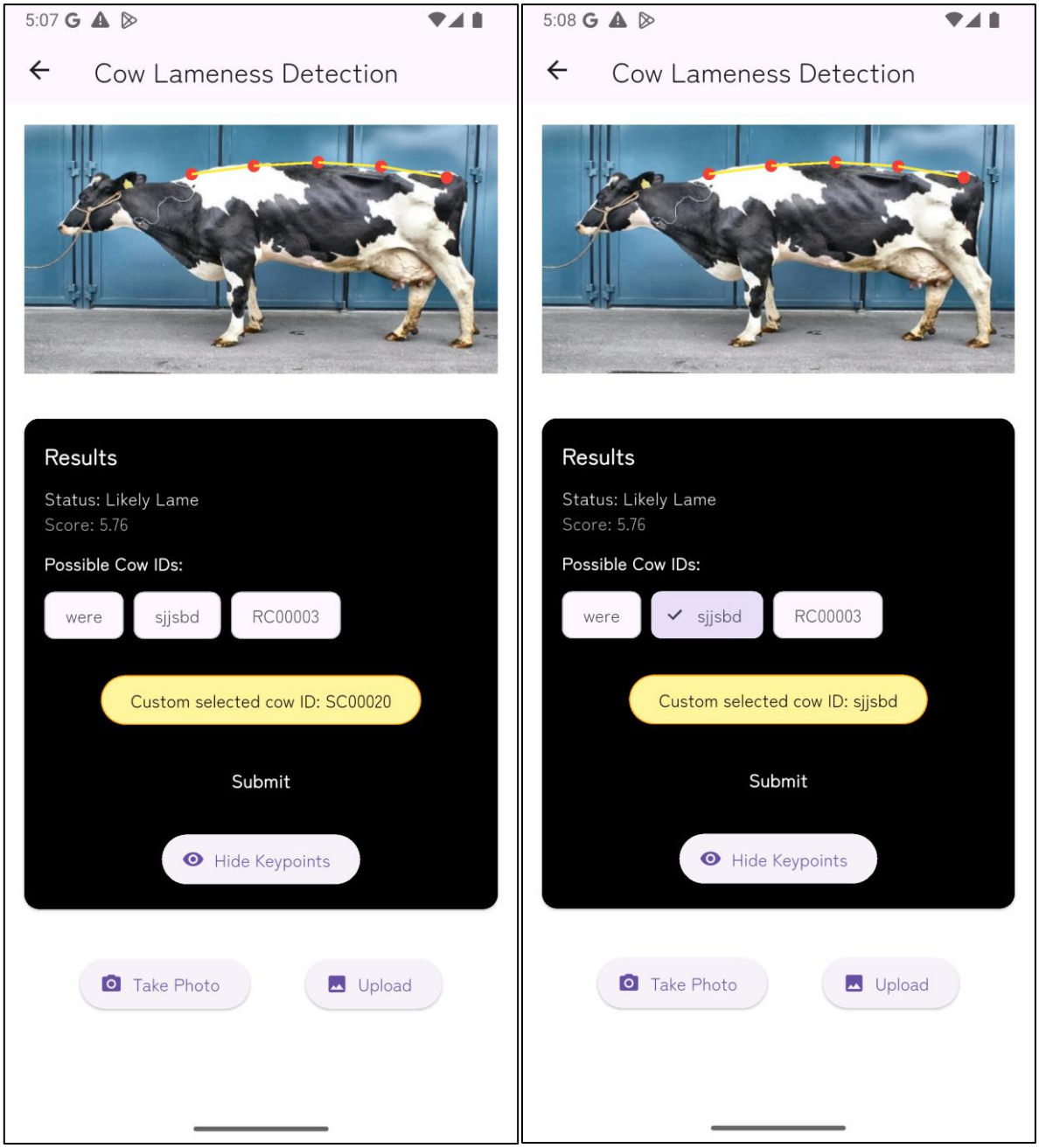


Figure 6.20 Example result of lame cow.

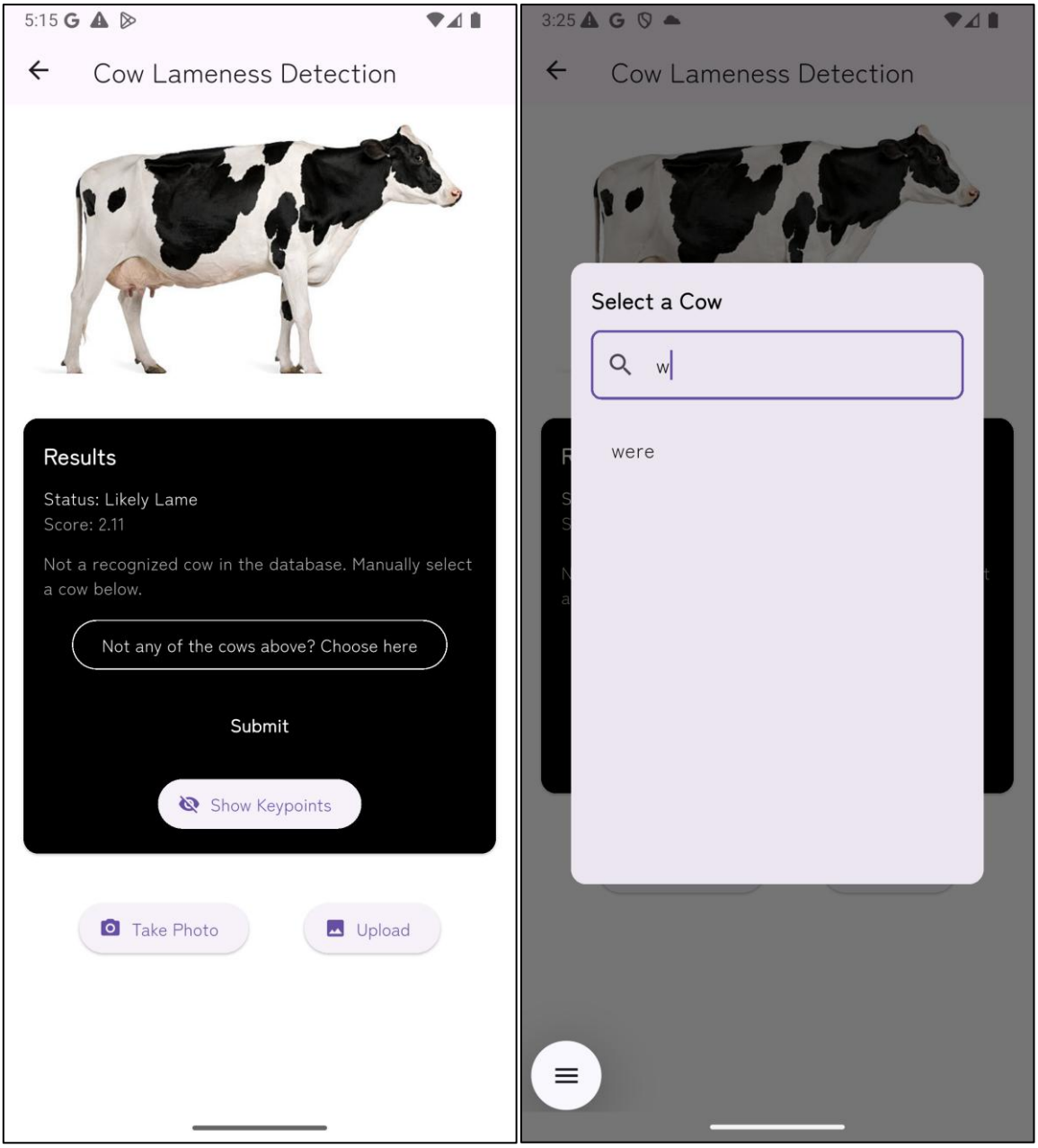


Figure 6.21 User manually choosing cow when no cow recognition result matches.

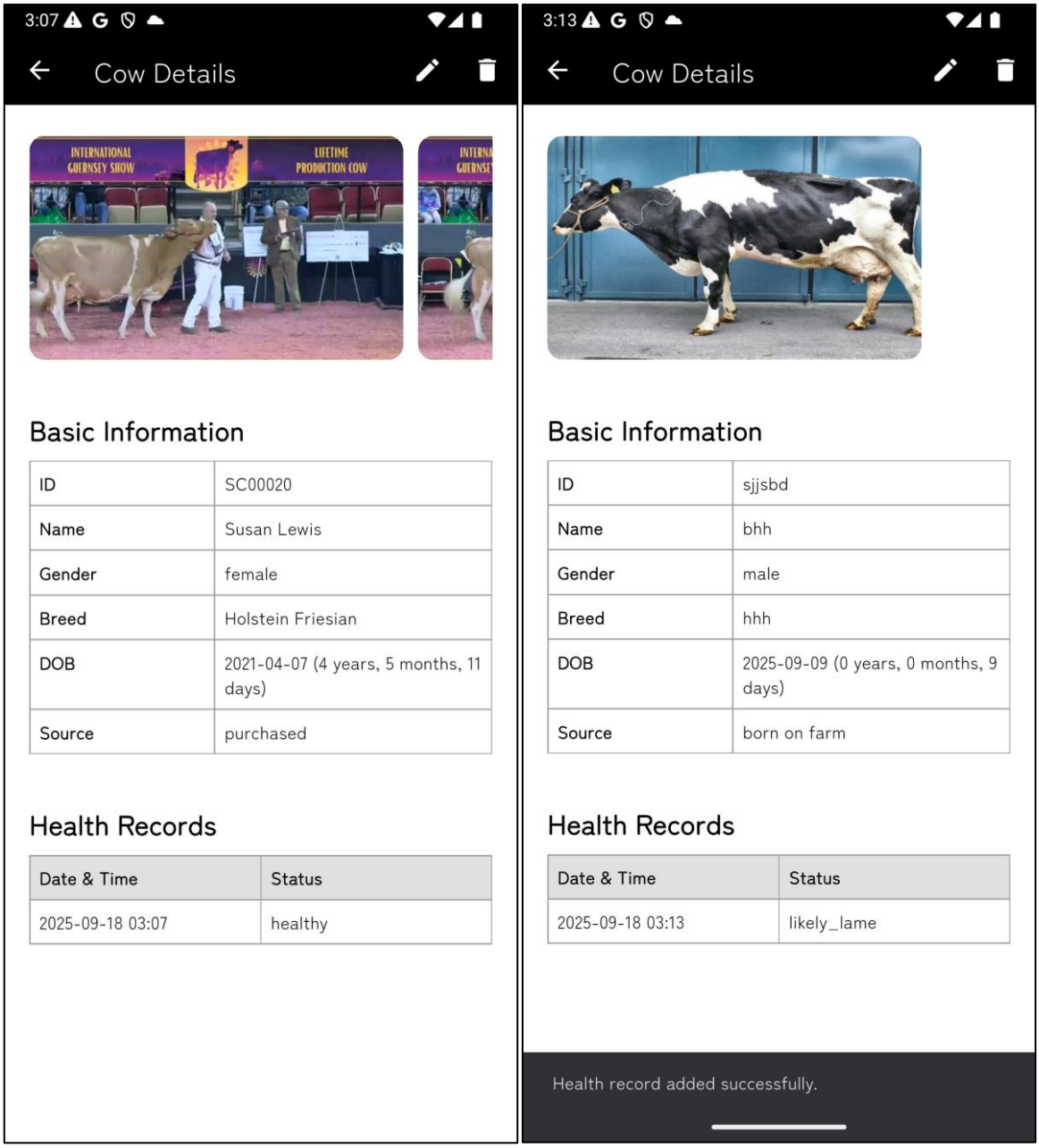


Figure 6.22 Cow health record updated from previous detections.

Figure 6.23 Cow health record updated from previous detections.

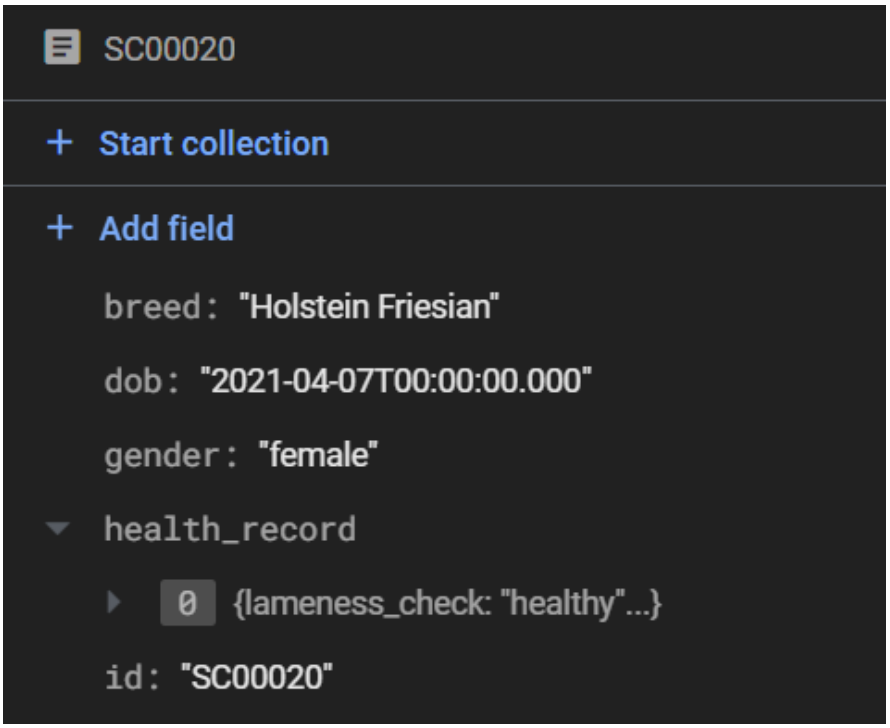


Figure 6.24 Cow's database updated with health record.

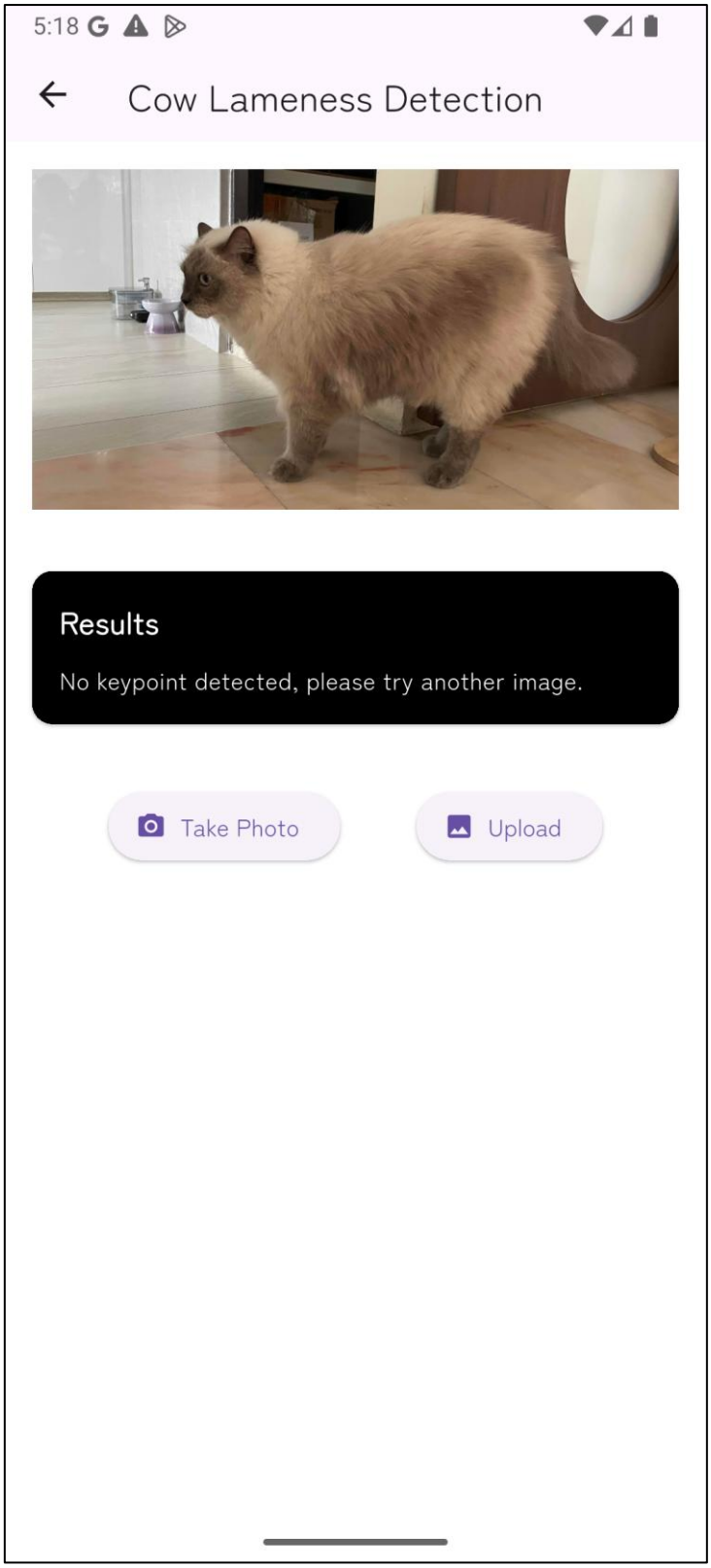


Figure 6.25 Example output of non-cow images.

6.3.4 Export Health Record

The Export Report feature allows the user to generate a summary of cow records within a selected date range. Once the date range is specified, the application compiles all cows that have health records during that period into a CSV file. This file contains the relevant information for each cow, making it easier for farmers to review the herd's health status. The user can then share the CSV file through any available platform, simplifying the process of identifying cows that require special attention and supporting better herd management decisions.

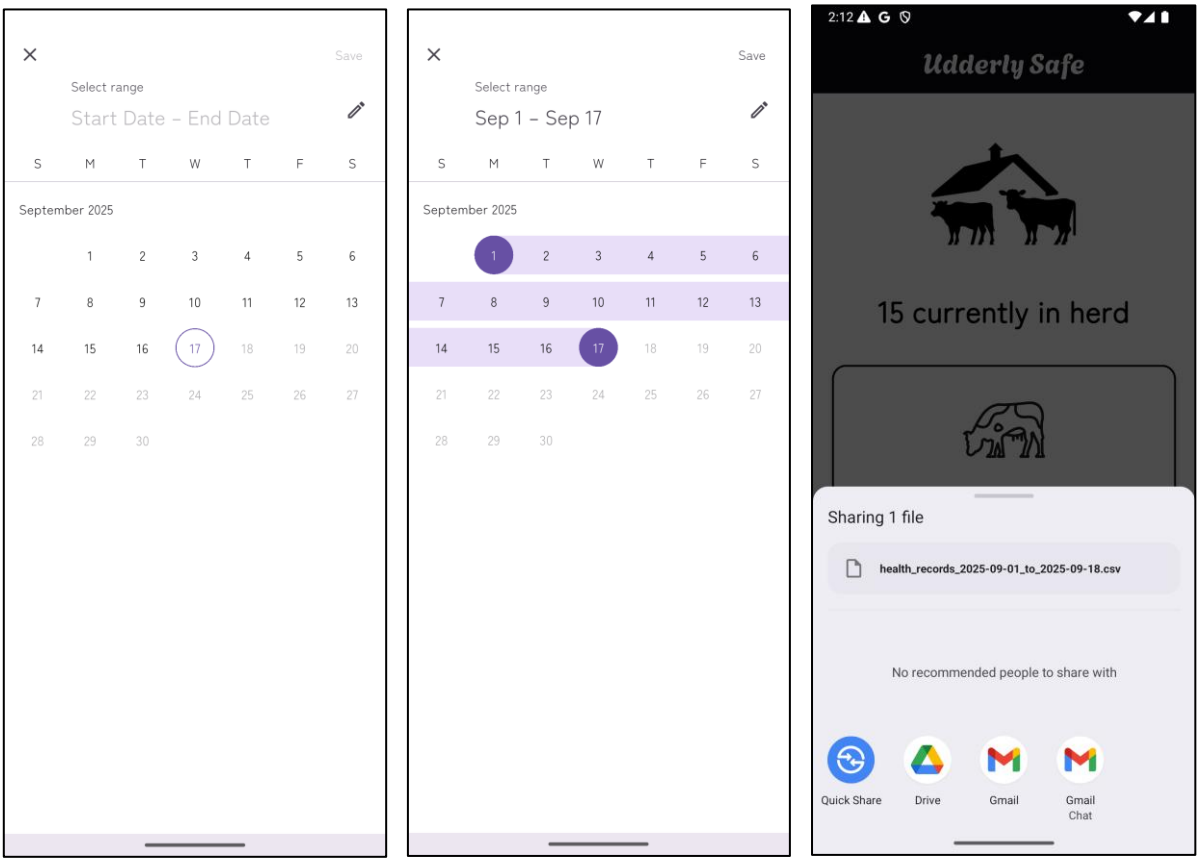


Figure 6.26 Choosing date range for health record CVS file export.

health_records_2025-09-01_to_2025-09-18.csv

	A	B	C	D
1	cow_id	name	timestamp	lameness_check
2	SC00020	Susan Lewis	2025-09-18 3:07:48	healthy
3	sjjsbd	bhh	2025-09-18 3:13:30	likely_lame
4	were	sdfdsf	2025-09-18 3:25:59	likely_lame

Figure 6.27 Sample content of CVS file.

6.4 Challenges

The biggest challenge encountered with this project was the lack of a suitable and comprehensive dataset. A significant portion of the project timeline was dedicated to sourcing and scaling data from online resources, which delayed the model development phase.

6.4.1 Challenges in Lameness Detection Model Development

For the lameness detection model, one major challenge was the tedious process of editing images and performing manual key point annotations. This task was time-consuming and labor-intensive, yet necessary to create a usable training set. In addition, the lack of verified lame cow images available online limited the model's robustness. As a result, the system could only provide a binary classification (lame or not lame), rather than a more detailed severity scale that would be more practical for farm management.

6.4.2 Challenges in Cow Recognition Model Development

For the cow recognition model, several limitations were identified. First, harvesting frames from videos to build the dataset caused the test samples to be relatively homogeneous, making it difficult for the model to generalize to real farm conditions where variations in lighting, weather, mud, and partial occlusions are common. While this approach provided a sufficient volume of data, the high similarity between consecutive frames introduced redundancy and made the training process more challenging. Second, the dataset scale was limited, covering only about 30 cows with roughly 20 images each. Real-world deployment would require recognition across hundreds of cows, where visual similarities between individuals could make discrimination significantly harder.

Another challenge relates to data augmentation. Although augmentation techniques improved accuracy during testing, they cannot fully replicate real-world variations such as extreme lighting shifts, motion blur, mud covering, or gradual changes in cow appearance over time. Lastly, the “unknown cow problem” presents a limitation. The system currently assumes that all cows in query images are already present in the gallery, but in practice, new or unregistered cows may appear. The present implementation does not explicitly handle these unknown cases, which is something that can be addressed in future developments.

6.5 Objective Evaluation

1. Lameness Detection

The goal of identifying cow lameness was accomplished. The model can identify whether a cow is healthy or lame by processing horizontal photographs of the animal. Although the system yields trustworthy binary outputs, the absence of validated annotated data prevents it from supporting severity grading at this time.

2. Cow Recognition

Part of the aim of recognizing individual cows was accomplished. The embedding-based recognition system works well when evaluated on the controlled dataset, where cows are plainly visible and conditions are stable.

3. Integration of Pipeline into mobile app

The objective of integration was effectively accomplished. The mobile application provided access to a single pipeline that included the lameness detection and cow recognition features. The database allows users to upload or take pictures, get detection results, and link them to specific cow records.

4. Detection of Multiple Cows in a Frame

This objective was not entirely accomplished. The cow recognition model is currently unable to assign identities to several cows at once, even if the system is able to identify multiple cows and their respective key points inside a frame.

5. Data Management and Reporting

The objective of providing data management and reporting features was successfully achieved. Users are able to add, edit, and delete cow records, add health records from lameness detection, and export reports in CSV format. This functionality enhances practicality by supporting herd-level record keeping and simplifying data sharing.

In summary, the system successfully meets its core objectives of lameness detection, cow recognition, pipeline integration, and record management, but falls short in handling multi-cow recognition and fine-grained lameness severity classification.

Chapter 7
Conclusion and Recommendations

7.1 Conclusion

In this project, an end-to-end system that integrates lameness detection and cow recognition into a mobile application connected to a cloud-based backend has been successfully developed. The objectives of detecting lameness, recognizing individual cows, integrating both functions into a unified pipeline, and providing data management and reporting features were largely achieved. Functional testing confirmed that users can add, edit, and delete cow records, perform lameness detection, and export reports, demonstrating the practicality of the system for herd management. Limitations remain in handling multiple cows in a single frame and providing fine-grained lameness severity, but the system provides a strong foundation for future enhancements. Overall, this project illustrates the feasibility of combining computer vision models with cloud and mobile technologies to support farmers in monitoring and managing their herds efficiently.

7.2 Future work

The ultimate objective of this project is to create a herd monitoring and lameness detection system that is completely automated. Under the setup at that stage, cameras would be positioned across the farm to continuously take pictures of every cow as they walk by, identify lame cows and rank its severity of lameness, and record health information based on the results of the cow recognition. Instead of actively managing data entry or analysis, farmers would be able to merely check health reports periodically as all these operations would be carried out without manual intervention. By offering a useful interface for data collecting, detection, and reporting, the current mobile application acts as an interim solution and paves the way for a future automated system that can perform at its peak efficiency.

While the current system successfully integrates lameness detection and cow recognition into a functional pipeline, several improvements can be made to enhance its robustness and usability in real-world farm environments. One key area is expanding and diversifying the dataset, particularly by collecting more verified images of lame cows and cows under varying

lighting, weather, and farm conditions, which would allow the lameness detection model to provide severity grading rather than binary outputs. Another important improvement is enabling multi-cow recognition in a single frame, which could be achieved by combining object detection with embedding-based identification for each detected cow. The recognition model could also be further generalized and scaled to handle larger herds, while better augmentation techniques or domain adaptation methods could improve robustness against extreme lighting, motion blur, mud, and partial occlusions. Additionally, mechanisms to handle unknown or newly introduced cows in the gallery would increase the system's adaptability, and enhancements to the mobile app interface such as batch processing, notifications for lameness alerts, and advanced health analytics could further support practical farm management. Addressing these areas would help transform the current framework into a more scalable, reliable, and user-friendly herd monitoring solution.

REFERENCES

[1] T. Gessese, A. Ayele, M. Z. Kinde, and A. Asmare, “Prevalence of Lameness in Dairy Cows and Associated Risk Factors at Hawassa Town Dairy Farms, Ethiopia,” *Veterinary Medicine International*, vol. 2024, pp. 1–7, Mar. 2024, doi: https://doi.org/10.1155/2024/2732333.

[2] A. I. Awad, “From classical methods to animal biometrics: A review on cattle identification and tracking,” *Computers and Electronics in Agriculture*, vol. 123, pp. 423–435, Apr. 2016, doi: https://doi.org/10.1016/j.compag.2016.03.014.

[3] “Lameness - BeefResearch.ca,” Beef Research. Accessed: Mar. 3, 2024. [Online]. Available: https://www.beefresearch.ca/topics/lameness/#

[4] “Cow Anatomy - Diagrams Of Cows & Calves,” Animal Corner. Accessed: Mar. 5, 2024. [Online]. Available: https://animalcorner.org/cow-anatomy/

[5] R. L. A, A. K. S, K. B. E, A. N. D, and K. K. V, “A Survey on Object Detection Methods in Deep Learning,” *IEEE Xplore*, Aug. 01, 2021. Accessed: Mar. 5, 2024. [Online].Available:https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9532809

[6] M. Garvey, “Lameness in Dairy Cow Herds: Disease Aetiology, Prevention and Management,” *Dairy*, vol. 3, no. 1, pp. 199–210, Mar. 2022, doi: https://doi.org/10.3390/dairy3010016.

[7] “Animal Welfare On Organic Farms Fact Sheet Series Identifying Lameness In Dairy Cattle” ECOA Animal Welfare Task Force. Accessed: Mar. 5, 2024. [Online]. Available:https://cdn.dal.ca/content/dam/dalhousie/pdf/faculty/agriculture/oacc/en/livestock/Welfare/Lameness_Dairy.pdf

[8] P. T. Thomsen, J. K. Shearer, and H. Houe, “Prevalence of lameness in dairy cows: A literature review,” *The Veterinary Journal*, vol. 295, p. 105975, May 2023, doi: https://doi.org/10.1016/j.tvjl.2023.105975.

[9] S. Chain Tun, Thi Thi Zin, P. Tin, and I. Kobayashi, “Cow Lameness Detection Using Depth Image Analysis,” *2022 IEEE 11th Global Conference on Consumer Electronics (GCCE)*, Oct. 2022, doi: https://doi.org/10.1109/gcce56475.2022.10014268.

[10] S. Barney, S. Dlay, A. Crowe, I. Kyriazakis, and M. Leach, “Deep learning pose estimation for multi-cattle lameness detection,” *Scientific Reports*, vol. 13, no. 1, Mar. 2023, doi: https://doi.org/10.1038/s41598-023-31297-1.

[11] X. Kang, X. D. Zhang, and G. Liu, “Accurate detection of lameness in dairy cattle with computer vision: A new and individualized detection strategy based on the analysis of the supporting phase,” *Journal of Dairy Science*, vol. 103, no. 11, pp. 10628–10638, Nov. 2020, doi: https://doi.org/10.3168/jds.2020-18288.

[12] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[13] A. G. Howard et al., “MobileNets: Efficient convolutional neural networks for mobile vision applications,” arXiv:1704.04861, 2017.

[14] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1492–1500.

[15] J. Torres, “YOLOv8 Architecture; Deep Dive into its Architecture -Yolov8,” YOLOv8, Jan. 15, 2024. https://yolov8.org/yolov8-architecture/#YOLOv8_Architecture_A_Deep_Dive_into_its_Cutting-Edge_Design (accessed Apr. 29, 2025).

[16] A. Sharma *et al.*, “Universal bovine identification via depth data and deep metric learning,” *Computers and Electronics in Agriculture*, vol. 229, p. 109657, Dec. 2024, doi: https://doi.org/10.1016/j.compag.2024.109657.

[17] A. Dhiman, K. Gupta, and D. K. Sharma, “Chapter 1 - An introduction to deep learning applications in biometric recognition,” ScienceDirect, Jan. 01, 2021. https://www.sciencedirect.com/science/article/pii/B9780128222263000015.

[18] M. E. Hossain, A. Kabir, L. Zheng, D. Swain, S. McGrath, and J. Medway, “A systematic review of machine learning techniques for cattle identification: Datasets, methods and future directions,” *Artificial Intelligence in Agriculture*, Sep. 2022, doi: https://doi.org/10.1016/j.aiia.2022.09.002.

[19] X. Pei et al., “Robustness of machine learning to color, size change, normalization, and image enhancement on micrograph datasets with large sample differences,” Materials & Design, vol. 232, p. 112086, Aug. 2023, doi: https://doi.org/10.1016/j.matdes.2023.112086.

[20] AHDB, “Mobility scoring: How to Score Your Cows | AHDB,” ahdb.org.uk, 2023. https://ahdb.org.uk/knowledge-library/mobility-scoring-how-to-score-your-cows

[21] J. Solawetz, “What is YOLOv8? A Complete Guide.,” Roboflow Blog, Sep. 04, 2024. https://blog.roboflow.com/what-is-yolov8/

[22] A. Poursaberi, C. Bahr, A. Pluk, A. Van Nuffel, and D. Berckmans, “Real-time automatic lameness detection based on back posture extraction in dairy cattle: Shape analysis of cow with image processing techniques,” Computers and Electronics in Agriculture, vol. 74, no. 1, pp. 110–119, Oct. 2010, doi: https://doi.org/10.1016/j.compag.2010.07.004.

[23] L. Bergamini *et al.*, “Multi-views Embedding for Cattle Re-identification.” Accessed: Sep. 15, 2025. [Online]. Available: https://iris.unimore.it/retrieve/handle/11380/1169525/208775/Multi_views_Embedding_for_Cattle_Re_identification.pdf

[24] A. Bengio, Y. LeCun, and G. Hinton, “Representation learning: A review and new perspectives,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[25] D. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[26] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.

[27] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality reduction by learning an invariant mapping,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2006, pp. 1735–1742.

[28] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 815–823.

[29] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, “ArcFace: Additive angular margin loss for deep face recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 4690–4699.

[30] “World Dairy Expo,” *YouTube*.
https://www.youtube.com/channel/UCZah3iZ3JsNBbpTYg4axnNQ

[31] L. van der Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[32] A. Aquino, J. P. Malpartida, J. P. Romero, and M. A. G. Ballester, “Dimensionality reduction for deep feature embedding visualization in image classification tasks,” *Applied Sciences*, vol. 13, no. 2, p. 879, 2023.

[33] S. Cai, Y. Huang, Y. Wei, and X. Yang, “A human pose estimation network based on YOLOv8 framework with efficient multi-scale receptive field and expanded feature pyramid network,” *Scientific Reports*, vol. 15, no. 1628, pp. 1–11, Jan. 2025.

[34] X. Xu, Y. Li, and Z. Zhou, “Enhanced human pose estimation using YOLOv8 with C2F-SimDLKA module,” *PLOS ONE*, vol. 20, no. 1, e0318578, Jan. 2025.

[35] Y. Liu, “What is YOLOv8: An in-depth exploration of the internal architecture,” *arXiv preprint*, arXiv:2408.15857, Aug. 2024.

POSTER

Application for Cow Lameness Detection Using Computer Vision

Kong Zi Lin (22ACB00584)
Supervisor: Dr. Ng Hui Fuang

UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

Introduction

Lameness in dairy cattle is a **costly condition** affecting movement, growth, and milk yield, leading to potential weight loss and culling. Caused by genetics, environment, injuries, or medical issues, early detection is challenging, but crucial for improving animal welfare and farm productivity.

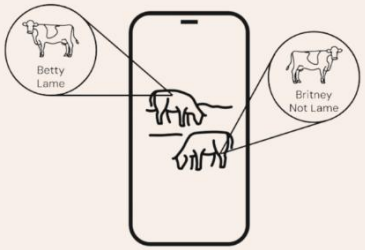
A healthy cow. A lame cow.

Lameness in cows can often be detected by observing the **arching of the cow's back.** It is however **often dismissed** due to the **difficulty of spotting it early on**.

Objective

1. Early identification of back arching in cows
2. Reduce farmer's workload
3. Enhance cow welfare

Scope



1. **Mobile application** for cow lameness detection through image analysis and RMSE calculation.
2. Cow registration and **medical history management** feature for efficient livestock health tracking.

Lameness Detection

YOLOv8 trained on manually annotated images of healthy cows' backs. **automatically detect five key points** along the cow's back and **calculate the RMSE** to **classify** the cow as healthy or lame.

Biosensing

Cow recognition based on a ResNet18 backbone with ArcFace loss, designed to generate discriminative embeddings that enable reliable identification of individual cows from images.

Results