RV College of Engineering®
Mysore Road, RV Vidyaniketan Post, Bengaluru - 560059, Karnataka, India

Go, change the world

Real Time Cashew Kernel Classification Using Deep Learning

An Interdisciplinary Project Report (XX367P)

Submitted by,

Ravikant	**1RV23EC408**
Sagar T Nayak	**1RV23EC410**
Kiran H R	**1RV23CS405**
Manoj Kumar B V	**1RV22CS407**
Yogeesh A S	**1RV23BT404**

Under the guidance of

Dr. Veena Devi S V
Associate Professor
Dept. of ECE
RV College of Engineering®

.

In partial fulfillment of the requirements for the degree of Bachelor of Engineering in respective departments

2024-25

RV College of Engineering®, Bengaluru

(*Autonomous institution affiliated to VTU, Belagavi*)

CERTIFICATE

Certified that the interdisciplinary project (XX367P) work titled ***Real Time Cashew Kernel Classification Using Deep Learning*** is carried out by **Ravikant** (**1RV23EC408**), **Sagar T Nayak** (**1RV23EC410**), **Kiran H R** (**1RV23CS405**), **Manoj Kumar B V** (**1RV22CS407**) and **Yogeesh A S** (**1RV23BT404**) who are bonafide students of RV College of Engineering, Bengaluru, in partial fulfillment of the requirements for the degree of **Bachelor of Engineering** in respective departments during the year 2024-25. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the interdisciplinary project report deposited in the departmental library. The interdisciplinary project report has been approved as it satisfies the academic requirements in respect of interdisciplinary project work prescribed by the institution for the said degree.

Dr. Veena Devi S V Guide	Dr. Ravish Aradhya Head of the Department
Dr. M.V. Renukadevi Dean Academics	Dr. K. N. Subramanya Principal

External Viva

Name of Examiners	Signature with Date
1.	
2.	

DECLARATION

We, **Ravikant** , **Sagar T Nayak**, **Kiran H R**, **Manoj Kumar B V** and **Yogeesh A S** students of sixth semester B.E., RV College of Engineering, Bengaluru, hereby declare that the interdisciplinary project titled '**Real Time Cashew Kernel Classification Using Deep Learning**' has been carried out by us and submitted in partial fulfilment for the award of degree of **Bachelor of Engineering** in respective departments during the year 2024-25.

Further we declare that the content of the dissertation has not been submitted previously by anybody for the award of any degree or diploma to any other university.

We also declare that any Intellectual Property Rights generated out of this project carried out at RVCE will be the property of RV College of Engineering, Bengaluru and We will be one of the authors of the same.

Place: Bengaluru

Date:

Name Signature

1. Ravikant(1RV23EC408)
2. Sagar T Nayak(1RV23EC410)
3. Kiran H R(1RV23CS405)
4. Manoj Kumar B V(1RV22CS407)
5. Yogeesh A S(1RV23BT404)

ACKNOWLEDGEMENTS

We are indebted to our guide, **Dr. Veena Devi S V**, Associate Professor, Department of ECE, RVCE for the wholehearted support, suggestions and invaluable advice throughout our project work and also helped in the preparation of this thesis.

We also express our gratitude to our panel member **Dr. Soumya A**, Professor, Department of CSE, RVCE for their valuable comments and suggestions during the phase evaluations.

Our sincere thanks to the project coordinator **Dr. Veena Devi S V**, Associate Professor, Department of ECE, RVCE for timely instructions and support in coordinating the project.

Our gratitude to **Prof. Narashimaraja P**, Department of ECE, RVCE for the organized latex template which made report writing easy and interesting.

Our sincere thanks to **Dr. Ravish Aradhya**, Professor and Head, Department of Electronics and Communication Engineering, RVCE for the support and encouragement.

We are deeply grateful to **Dr. M.V. Renukadevi**, Professor and Dean Academics, RVCE, for the continuous support in the successful execution of this Interdisciplinary project.

We express sincere gratitude to our beloved Professor and Vice Principal, **Dr. Geetha K S**, RVCE and Principal, **Dr. K. N. Subramanya**, RVCE for the appreciation towards this project work.

Lastly, we take this opportunity to thank our family members and friends who provided all the backup support throughout the project work.

ABSTRACT

Cashew kernel grading determines the commercial value of nuts based on shape, size, and surface quality.Traditional manual grading is slow and inconsistent due to human error, fatigue, and lighting variations, affecting both product quality and pricing.This project proposes a real-time automated grading system using Computer Vision (CV), embedded hardware, and Artificial Intelligence (AI)-based You Only Look Once — a one-stage object detection framework for real-time applications (YOLO) classification to improve accuracy, speed, and reliability.

The proposed work focuses on developing and implementing an intelligent cashew kernel classification system capable of operating in real time. A Deep Learning (DL) strategy was employed, utilizing a custom dataset recorded with a high-definition webcam under various lighting conditions to represent three commercial kernel grades: W180, W300, and W500. Labeling and data augmentation were performed using Roboflow, and the YOLOv5s model was trained for detection and classification. The model was trained to produce bounding boxes **B** and class predictions with good Mean Average Precision (mAP), by optimizing loss functions that include IoU, precision (P), and recall (R).

The trained YOLO model was run on a A compact ARM-based single-board computer used for real-time edge inference and control (Raspberry Pi) for real-time inference at 5 FPS. Video frames from a top-mounted webcam over a conveyor system were processed, and detected classes were sent to an Arduino Uno over Universal Asynchronous Receiver Transmitter (UART). Arduino drove stepper motors and mechanical flaps using L298N drivers to physically sort kernels into bins. The system obtained more than 93% accuracy in classification and doubled the throughput compared to hand grading. Hardware-software co-design guarantees strong edge inference and real-time physical sorting.

Hardware implementation confirmed the simulated results. Real-time inference on Raspberry Pi, complemented by UART-based communication to the Arduino-driven flap system, demonstrated physical sorting accuracy of 94–96% for all three classes—W180 (96%), W300 (94%), and W500 (94%). The combination of AI-based detection with low-cost embedded modules is found to be cost-effective and scalable and provides more than 87% classification accuracy and greatly minimizes operational cost and labor dependency in industrial applications.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AI Artificial Intelligence

Batch Size Number of samples processed before the model is updated

BPN Backpropagation Neural Network

C3 Cross Stage Partial Bottleneck block for feature reuse and reduced computation

CBS Convolution + BatchNorm + SiLU

CNN Convolutional Neural Network

CV Computer Vision

DL Deep Learning

Epochs One complete pass through the entire training dataset

FPN Feature Pyramid Network

GLCM Gray-Level Co-occurrence Matrix — a statistical texture feature used to describe spatial relationships between pixel intensities

GPIO General Purpose Input Output

GPU Graphics Processing Unit

L298N Dual H-Bridge Motor Driver used to control DC and stepper motors

LR Learning Rate — controls how much to change the model in response to the estimated error

LWP Large White Pieces (broken cashew grade)

mAP Mean Average Precision

ML Machine Learning

NEMA 17 stepper motor A type of stepper motor with a 1.7 inch × 1.7 inch faceplate, commonly used in 3D printers, CNC machines, and robotics for precise control of rotation

NMS Non-Maximum Suppression

PAN Path Aggregation Network

Raspberry Pi A compact ARM-based single-board computer used for real-time edge inference and control

RF Random Forest — an ensemble machine learning algorithm that builds multiple decision trees for classification or regression

SGD Stochastic Gradient Descent

SPPF Spatial Pyramid Pooling - Fast

SVM Support Vector Machine

UART Universal Asynchronous Receiver Transmitter

USB Universal Serial Bus

WS White Splits (cashew kernel grade)

YOLO You Only Look Once — a one-stage object detection framework for real-time applications

LIST OF SYMBOLS

P Precision — proportion of true positive detections

R Recall — proportion of relevant items detected

Concat Concatenation of feature maps along the channel axis

FPS Frames Per Second — inference speed

IoU Intersection over Union — metric to evaluate bounding box overlap

UpSample Upsampling operation that increases the spatial resolution of feature maps

W180 Whole cashew grade with 180 kernels per pound

W300 Whole cashew grade with 300 kernels per pound

W500 Whole cashew grade with 500 kernels per pound

Chapter 1

Introduction to Real Time Cashew kernel classification

CHAPTER 1

INTRODUCTION TO REAL TIME CASHEW KERNEL CLASSIFICATION

The precise classification of cashew kernels is important for good quality control in the agricultural packaging units. Historically, cashew kernels have been classed as whole, split and damaged manually—an activity that is widely agreed to be slow, subjective, and heavily dependent on experience and skill thus making processes not only slow but also inconsistent. As the demand for high-quality export grade kernels is growing, the importance of accurate, automated, and scalable sorting solutions is becoming more valid.

1.1 Introduction

With the advancement of Computer Vision (CV) and Deep Learning (DL) there is significant potential for classification problems in agriculture and food processing to deliver a solution. Especially with real-time systems enabling lightweight models like You Only Look Once — a one-stage object detection framework for real-time applications (YOLO) to be installed into edge devices like the Raspberry Pi, thus presenting a small, inexpensive, and efficient substitute for manual grading. Real-time systems can clearly highlight some of the visual features that can be used to classify kernels, such as shape, size, and surface integrity, and quickly classify based on these visual features.

The Fig1.1 Cashew Size Chart divides cashew kernels by size and shape. The "W" grades (e.g., W180, W210, W320) represent whole cashew kernels; the number indicates how many fit in a pound (lower = larger), Halves, White Splits (cashew kernel grade) (WS), and Large White Pieces (broken cashew grade) (LWP) denote broken or split kernels found in processed food products.

1.2 Literature Review

1.2.1 Introduction

Grading and classification of agricultural products have increasingly moved towards automation to maintain consistency in product quality, enhance efficiency, and lower labor expenses. Kernel grading in the cashew processing industry by size, shape, and surface quality is used directly for determining prices. Conventional manual processes are time-consuming and subject to variability. The introduction of CV and DL methods,

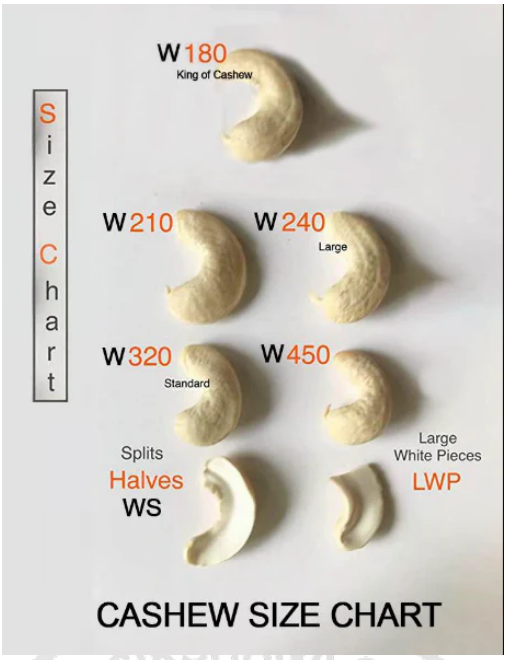


Figure 1.1: Cashew Size chart

more specifically YOLO-based object detection models, has made real-time and trustworthy grading a possibility. This section reviews existing approaches and highlights advancements in YOLO models relevant to automated cashew grading.

1.2.2 Cashew Kernel Classification Methods

Early publications like [1] employed Support Vector Machine (SVM) in a machine vision environment to sort kernels by visual features such as size and appearance. These systems were precise under controlled lighting conditions but were not robust under real-world conditions.

For betterment over conventional techniques, a hybrid of SVM and Backpropagation Neural Network (BPN) was introduced in [2]. It improved consistency and enabled real-time simulation but was still constrained by sensitivity to kernel overlap and non-integration with hardware sorting systems.

The authors of [3] suggested a classification model based on Convolutional Neural Network (CNN) that could process lighting and orientation variations. Their later work, CashNet-15 [4], showed improved performance through data augmentation and reported classification accuracies over 94%. These, however, were software-based systems.

The authors of [5] proposed a lightweight CNN for embedded hardware to classify kernels in real-time using fewer resources. The researchers of this paper [6], contrasted

several Machine Learning (ML) models and made CNNs stronger but did not address practical deployment issues.

1.2.3 Fruit and Agricultural Produce Classification

In related agricultural domains, fruit classification techniques offer valuable insight. In [7], the authors proposed a CNN model to classify durian ripening stages based on preprocessing methods such as histogram equalization. Despite being accurate under simulation, the system was not intended for real-time use.

Transfer learning has been effective in food categorization. The authors of [8] used Inception-V3 to classify fruit types with good Accuracy and minimal training burden through a comparatively smaller dataset. In [9], also used transfer learning on a deep CNN for the classification of dry fruits, highlighting data augmentation.

The research in [10], compared CNN and YOLO architectures for classifying fruits. CNN was outperformed by YOLO when it came to inference speed, which makes it adept for real-time processing systems.

Deep CNNs were employed for classifying dry fruits in [9], by leveraging transfer learning on pre-trained models. The study emphasized the role of data augmentation and preprocessing in maintaining classification accuracy for varying classes of dry fruits.

IoT-based solutions were also investigated in [11], where ML models were integrated with sensor networks to sort fruits. The work put high priority on preprocessing, lighting control, and augmentation techniques to simulate real-world scenarios as effectively as possible.

1.2.4 Real-Time Object Detection and YOLO

The YOLO family of versions v1 through v8 has become a prominent real-time object detection framework because of its single-pass detection and classification unified approach. The YOLOv5 architecture, as described in [12], includes feature extraction and computational optimizations and is, therefore, suitable for low-resource embedded systems like the A compact ARM-based single-board computer used for real-time edge inference and control (Raspberry Pi).

In [10], they performed a comparative analysis of CNNs and YOLO for classifying fruits and determined that YOLO greatly surpasses CNNs in terms of inference speed, which is a key consideration for on-the-fly grading in conveyor-based systems.

Also, YOLOv8 provides better detection precision on small and overlapping objects [10], essential in the case of cashew kernel sorting where kernel overlap and occlusions are common. These improvements rectify limitations identified in previous CNN-based models, especially in visual clutter and edge detection [4], [5].

1.2.5 Cashew Kernel Composition

Physical and chemical properties of cashew kernels depend on their geographic location, processing methods, and conditions of harvest. Nutritional analysis like [13], [14] indicate kernel composition variability, which further impacts their visual appearance—presenting dilemmas for automatic classification systems.

Post harvest processing brings in further variability, as elaborated in [15], whereas chemical extraction operations (e.g., shell liquid extraction) covered in [16] affect color and surface texture of kernels.

The Cashew Handbook [17], provides a global perspective on cashew production, grading standards, and trade, serving as a strong contextual guide for developing grading automation systems

In addition, quality standards are important not only to ensure grading productivity but also to protect consumer well-being and nutritional quality. Research like [18] and [19] highlights the necessity for careful sorting procedures to avoid allergic responses and maintain nutritional content of kernels—reinforcing the general drive toward instituting automated grading systems.

1.2.6 Recent Developments in Cashew Kernel Grading

Recent research demonstrates increasing emphasis on end-to-end integration of lightweight DL models in embedded systems for cashew kernel grading. In a study published in 2024, [20], the researchers implemented an embedded YOLOv8-based architecture (SC3T) that incorporates spatial pyramid pooling and transformer-enhanced modules. The system attained more than 97.8% detection precision and was optimized for deployment on edge hardware such as the NVIDIA Jetson Nano. It successfully managed occlusions, grade overlap, and real-time latency requirements. The work explained how current YOLO variants incorporating attention mechanisms can improve accuracy in physical kernel separation.

In another recent study, [21] presented a hybrid solution in which ResNet-50 was

utilized as a feature extractor while classification was done using SVM. Their classifier reported a classification accuracy of 97.4% on three commercial grades of cashew (W180, W320, W500). Though not an object detection pipeline, the research affirmed the stability of CNN–SVM pairs in representing high-level kernel features effectively and offered an alternate pipeline for memory-restricted or real-time detection speed-restricted environments.

Moreover [22], addressed the differentiation of cashew wholes and splits with hand-engineered shape descriptors and an SVM classifier. Their approach attained 93% accuracy at very low computational expense, providing a lightweight solution appropriate for low-resource settings. Even if they are not scalable for multi-class cases or sophisticated sorting pipelines, they are helpful in pre-sorting or fallback stages.

A study [23], focused on kernel defect detection with traditional image processing methods like Gray-Level Co-occurrence Matrix — a statistical texture feature used to describe spatial relationships between pixel intensities (GLCM) and color histograms. The paper utilized Random Forest — an ensemble machine learning algorithm that builds multiple decision trees for classification or regression (RF) and SVM classifiers and obtained accuracy of over 94%. While not having real-time inference capacity, the paper offers us insight into stable feature extraction techniques in unfavorable lighting and background conditions.

Lastly, a rule-based fuzzy logic classifier was constructed by the authors of the paper [24], in which kernel dimensions like length and thickness were utilized to identify WW grades. The system attained classification decisions within 0.4 seconds per kernel, indicating its viability for latency-restricted edge applications, albeit constrained by flexibility and grade scalability.

1.2.7 Comparative Analysis of Methods

A comparison of the various classification approaches used in the literature reveals a progressive improvement in performance and practicality, especially with the transition from traditional machine learning models to modern deep learning and object detection frameworks. Table 1.1 presents a summary of key methods based on their model types, accuracy, real-time capability, and deployment feasibility.

Table 1.1: Comparison of Kernel Grading Techniques

Method	**Model Type**	**Accuracy**	**Real-Time**	**Remarks**
SVM-Based Grading [1]	SVM	∼85%	No	Sensitive to lighting and orientation
CNN + BPNN Hybrid [2]	Hybrid	88–90%	No	Generalizable, but lacks hardware integration
CashNet-15 [4]	Optimized CNN	94–96%	No	High static accuracy, but no real-time capability
Low-Cost CNN Deployment [5]	Lightweight CNN	92–95%	Limited	Hardware-friendly, lacks sorting integration
YOLOv5s-Based Detection [10]	YOLOv5s	96–97%	Yes	Fast inference, Raspberry Pi deployable
YOLOv8n-Based Detection [10]	YOLOv8n	97–98%	Yes	Improved detection of small objects, anchor-free

This comparison highlights that traditional SVM and hybrid models offer only moderate accuracy and are impractical for real-time environments. Deep CNNs significantly improve performance but lack deployment flexibility without GPU resources. In contrast, YOLO-based models such as YOLOv5 and YOLOv8 balance speed, accuracy, and hardware efficiency, making them ideal for edge-based classification systems.

1.2.8 Summary of Findings

Various important observations emerge from the large body of literature reviewed about the development and present status of automated cashew kernel classification and allied agricultural product sorting:

- Classical machine learning methods like SVM and BPNN provide limited precision and are poor at generalization across different environmental conditions. These are mostly handcrafted feature-based methods and hence cannot be deployed in real-time applications [1], [2].

- Deep learning methods, particularly CNN-based models, contribute immensely to-

wards improving classification accuracy and minimizing the reliance on human feature engineering. The majority of the CNN systems were, however, implemented in static environments and did not integrate with mechanical sort hardware [3], [4].

- Transfer learning based on models such as Inception-V3 and enlarged training datasets enhances model performance in situations where there is limited labeled data [8].

- YOLO real-time object detection, particularly versions YOLOv5 and YOLOv8, is very accurate with very fast inference rates, even on embedded hardware systems. This makes YOLO very fit for industrial automation of kernel grading [5], [10].

- Notwithstanding the improvement in detection algorithms, few studies in the literature reviewed herein discuss combining classification outputs with actuation mechanisms for physical sorting in real-time.

The literature reviewed above validates the possibility of classifying Indonesian cashew kernels using deep learning but uncovers a limitation in actual, end-to-end real-time implementation with physical sorting integration. .

1.2.9 Research Gaps

Despite the significant progress in the area of automated produce grading, a number of essential gaps still exist in the particular area of real-time cashew kernel classification and sorting:

- **End-to-end automation absence:** Most current systems tend to concentrate on classification alone and lack integration with actuator mechanisms needed for physical sorting on conveyor belts. Real-time detection is not enough without matching actuation for useful deployment.

- **Hardware constraints in earlier work:** The majority of the models in the literature are designed for high-performance computing hardware or for GPUs and thus are not applicable on low-cost embedded systems utilized in small to medium-scale cashew processing units [5].

- **Minimum real-world deployment:** Models are usually evaluated under ideal conditions of static lighting and few dataset variations. Systems based on ours

often neglect lighting variation, kernel overlap, and conveyor belt speed variation, which are typical in real factory environments [4].

- **Sparse attention to mechanical-electronic communication:** Extensive research is very rare on how one can translate predictions from deep learning models into accurate control signals for microcontrollers such as Arduino to power servo or stepper motors to sort.

1.3 Motivation

The motivation for this project is to overcome these hurdles by developing a comprehensive real-time kernel classification and sorting system. The solution utilizes YOLOv5s for object detection that is run on Raspberry Pi for edge inference with classification results communicated to an Arduino Uno microcontroller for actuating stepper motors to segregate kernels. The system combines deep learning, edge deployment, and real-time actuation to close the research-to-deployment gap and present a scalable solution for cashew processing industries.

1.4 Problem statement

To develop an efficient and scalable cashew segregation system that can accurately classify cashew kernels according to their respective grades, overcoming challenges related to real-time processing, image quality, hardware limitations, defect detection, and dataset constraints.

1.5 Objectives

The objectives of the project are

1. Develop an image processing system to capture raw cashew kernel images and perform data preprocessing.
2. Design and implement algorithms for feature extraction, data analysis, and classification of kernel variations.
3. Apply DL techniques to accurately detect, classify, and grade defects in cashew kernels
4. Integrate reliable hardware and software components for efficient, real-time testing and grading

1.6 Brief Methodology of the project

1. **Image Acquisition:** Capture diverse images of cashew kernels for analysis.
2. **Labeling & Training:** Manually label the captured images by grade and train a deep learning model for classification.
3. **Real-Time Classification:** Deploy the trained model for real-time detection and grading of kernels.
4. **Hardware Integration:** Connect the classifier output to microcontrollers and sensors to enable physical control of sorting mechanisms.
5. **Automated Sorting:** Use actuators to sort the classified kernels into appropriate bins, enabling automated grading.

The overall process flow is illustrated in Figure 1.2.

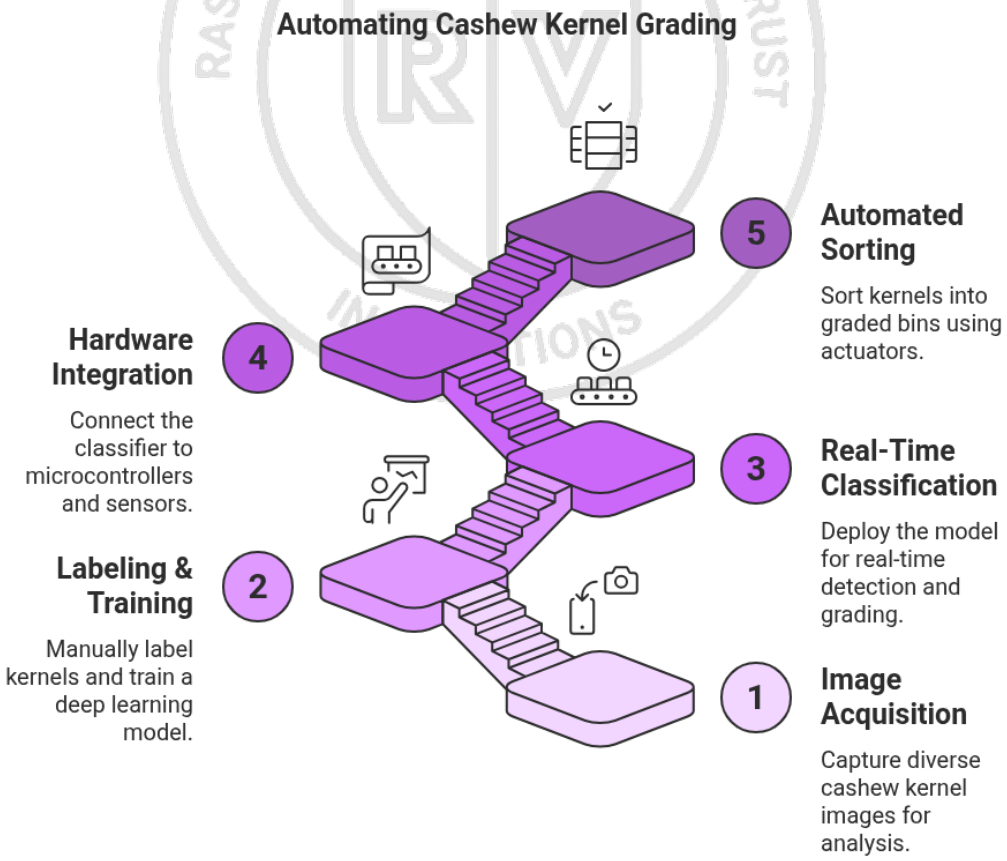


Figure 1.2: Image Showing Brief Methodology

1.7 Assumptions made / Constraints of the project

Assumptions:

- Cashew kernels are fed one at a time or in a spaced manner onto the conveyor belt to ensure clear visibility and reduce occlusion during image capture.
- Lighting conditions are maintained within a controlled range to avoid significant shadows or glare that may affect detection accuracy.
- The webcam used for image acquisition has a fixed position and angle relative to the conveyor to ensure consistency across frames.
- The classification model (YOLOv5s) has been pre-trained and optimized for the specific grades of kernels relevant to the system: W180, W300, W500.
- The Raspberry Pi 4 used for inference is assumed to have sufficient cooling and power supply for sustained operation.
- The mechanical sorting system (flaps and motors) operates without slippage or physical misalignment during segregation.

Constraints:

- The system may struggle with classification if multiple kernels overlap or cluster closely on the conveyor.
- Detection accuracy may drop under drastically changing lighting conditions, or if foreign particles or debris are present on the belt.
- Real-time inference speed is limited by the computational capability of Raspberry Pi 4, affecting maximum throughput.
- The system currently supports only a limited number of kernel grades and may require retraining for additional categories.
- Flap-based sorting assumes a fixed bin layout and may not adapt dynamically to changes in processing line configurations.
- Network or cloud connectivity is not integrated, so monitoring and updates must be performed locally.

1.8 Organization of the report

This report is organized as follows:

- Chapter 2 discusses the creation, acquisition, and refinement of the cashew kernel dataset used for training the classification model. It outlines the image capturing setup, preprocessing techniques, and augmentation strategies employed to ensure dataset diversity, quality, and model robustness.

- Chapter 3 discusses the design, development, and training processes involved in building a real-time cashew kernel classification and sorting system. It presents the dataset creation, YOLOv5s model architecture, training workflow, and integration with hardware components such as Raspberry Pi and Arduino. Furthermore, it explains the end-to-end pipeline from image acquisition to physical kernel sorting, ensuring a reliable and scalable solution for industrial cashew grading applications.

- Chapter 4 discusses the entire system integration of the cashew kernel classification and sorting system. It outlines the end-to-end process involving image capture, model inference with Raspberry Pi, and hardware-based kernel sorting using Arduino-driven actuators. The chapter describes each hardware part such as the Universal Serial Bus (USB) webcam, Dual H-Bridge Motor Driver used to control DC and stepper motors (L298N) motor driver, NEMA 17 stepper motor stepper motors, and elaborates on their functionality in real-time grading. A labeled hardware setup and working overview are also provided to show the integrated functioning..

- Chapter 5 evaluates the performance of the proposed cashew kernel classification and sorting system. It presents the results derived from model training and validation. Key performance metrics such as precision, recall, mean Average Precision (mAP), and confusion matrix analysis are discussed in detail. The system's real-time effectiveness is also validated through deployment on a live conveyor setup, followed by identification of limitations and insights for improvement.

- Chapter 6 summarizes the key outcomes of the proposed real-time cashew kernel classification and sorting system. It presents the concluding observations based on system design, training performance, and integration results. Furthermore, it

highlights possible future directions that can improve system accuracy, scalability, and operational robustness for broader industrial applications.

This chapter presents the motivation, background, and foundational concepts behind the development of a real-time cashew kernel classification using DL. It describes the limitations of manual grading and the potential of DL and CV using YOLO models on embedded platforms such as Raspberry Pi to provide automated, accurate, and scalable alternatives. The literature review addresses previous work, identifies major research gaps — such as absence of real-time deployment and integration with hardware and justifies the selection of YOLOv5s for edge inference. The chapter further outlines the problem statement, objectives, methodology, and system assumptions and constraints, and provides a structured overview of how the proposed system seeks to close the gap between research and practical application in cashew processing.

Chapter 2

Data Formulation and Pre-processing

CHAPTER 2

DATA FORMULATION AND PRE-PROCESSING

This chapter provides an overview of the dataset creation process, cashew kernels of various grades: W180, W300, W500 kernels. The images were taken with a USB webcam at different lighting conditions to reflect realistic working conditions and maximize model generalization.

2.1 Image Capturing

The cashew kernels were placed on a conveyor belt designed to mimic the real industrial processing conditions. the camera setup was calibrated for a fixed top-down view directly above the belt with the distance around **6cm** (as in the Fig 2.1) to maintain consistent and reliable data collection. Being perpendicular to the belt surface allows limited perspective distortion thus keeping the field of view uniform. Consistent background is essential for proper object detection and classification of images.

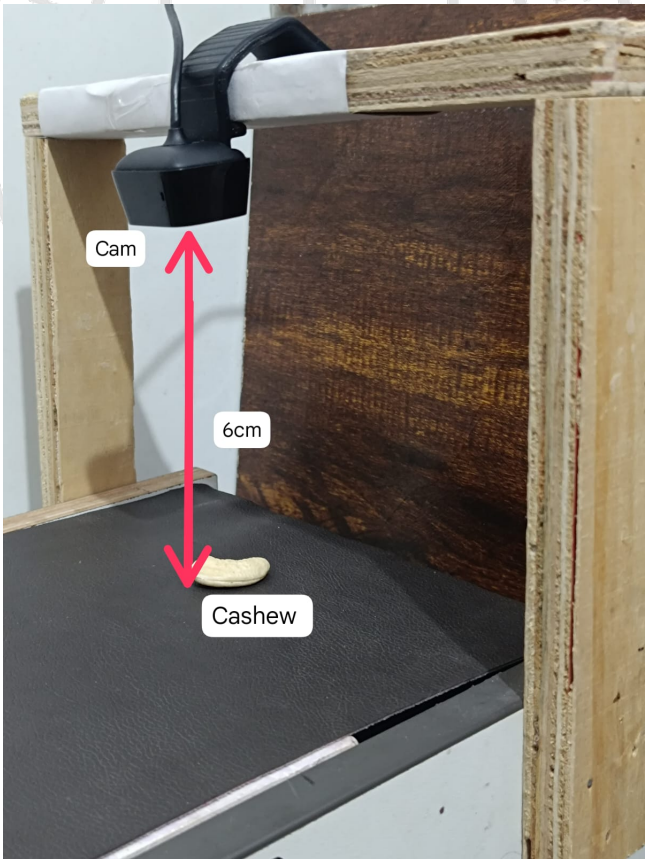


Figure 2.1: Camera setup

Each kernel was placed with adequate spacing to avoid overlapping or occlusion,

thereby ensuring clear visibility of individual kernel features such as shape, size, texture, and color.The images were captured in bursts as the conveyor belt moved at a controlled speed, replicating the actual dynamics of a production line. The lighting conditions were varied systematically—using combinations of natural and artificial lighting—to introduce diversity into the dataset and enhance the model's ability to generalize across different environmental settings.

In total, around 4993 high-resolution images were collected, comprising a well-distributed representation of the three target cashew kernel grades: W180 (1736 instances), W300 (2006 instances), and W500 (1931 instances). Figure 2.2 illustrates representative examples from each grade. The dataset's scale and diversity help reduce class imbalance and enable the deep learning model to generalize effectively, supporting high classification accuracy across all grades in real-time industrial scenarios.

(a) W180 (b) W300 (c) W500

Figure 2.2: Cashew Grades: (a) W180, (b) W300, (c) W500

2.2 Preprocessing

Before moving to training process a series of steps or image processing has been done to enhance the dataset and it's quality so that the model can work on any variability.

The following key operations were implemented:

- **Resizing:** All images were resized to a uniform resolution of **640** × **640** pixels. This standard input size is compatible with the YOLOv5s model and ensures consistent feature extraction across the dataset.

- **Noise Reduction:** To improve the clarity of the kernel edges and reduce background interference, noise reduction techniques such as **Gaussian blurring and median filtering** were applied selectively during preprocessing. This helped in removing grainy artifacts, especially in low-light images.Although the differences are

subtle to the human eye, these techniques contribute to improved model robustness during training, as illustrated in Figure 2.3.

Figure 2.3: Noise Reduction

- **Image Enhancement:** Image contrast and brightness were adjusted to highlight the fine details of the cashew kernels. Histogram equalization and adaptive enhancement techniques were used to make features more distinguishable, especially under non-uniform lighting conditions.These enhancements, although subtle visually, play a key role in making important kernel features more detectable during training, as demonstrated in Figure 2.4.

Figure 2.4: Image Enhancement

- **Data Augmentation:** To improve the robustness and generalization of the model, a variety of augmentation techniques were used, including:
  - Rotation (clockwise and counter-clockwise)
  - Horizontal and vertical flipping
  - Random scaling and zoom
  - Brightness and contrast variation
  - Synthetic noise addition

These augmentations simulated different real-world scenarios and variations in kernel orientation and appearance, helping the model become invariant to such transformations and this is achived through **Albumentations** A fast and flexible Python library used to apply image augmentations like rotation, flipping, brightness changes, and noise. A visual summary of these augmentation techniques is shown in Figure 2.5, highlighting the diversity introduced in the training data.

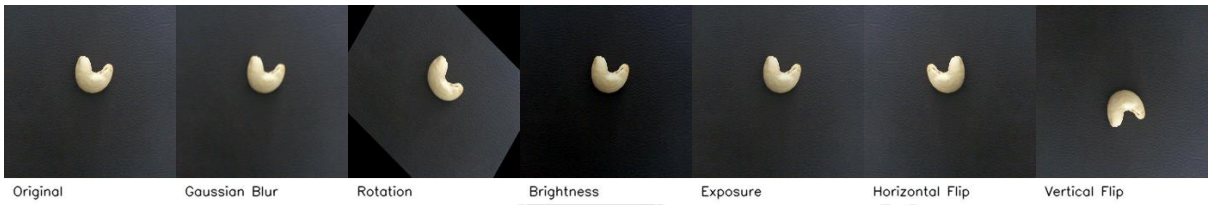


Figure 2.5: Image augmentation

These preprocessing steps collectively contributed to creating a high-quality, diverse, and well-prepared dataset suitable for training an accurate and reliable object detection model.

2.3 Cashew Kernel Dataset Summary

The dataset includes three commercial grades of cashew kernels W180, W300, and W500 compiled from a total of 1,540 original high-resolution images captured at 1280×800 resolution. Through extensive data augmentation and labeling, the dataset was expanded to 5,673 annotated kernel instances (W180: 1,736, W300: 2,006, W500: 1,931), improving class balance and model generalizability. All images were resized to 640×640 pixels prior to training to ensure consistency and compatibility with the YOLOv5s architecture. A detailed overview of the dataset composition and preprocessing steps is provided in Table 2.1.

Grade	**Labeled Instances**	**Original Image Resolution**	**Resized Shape**
W180	1,736	1280×800	640×640
W300	2,006	1280×800	640×640
W500	1,931	1280×800	640×640
Total	**5,673 instances**	**1,540 images**	640×640

Table 2.1: Cashew Kernel Dataset Summary After Augmentation and Labeling

The dataset was carefully built by taking more than 4800 high-resolution 1280 x 800 px, cashew kernel images, equidistributed across grades W180, W300, and W500, under a

calibrated top-down USB webcam system that replicates industrial conveyor conditions and mixed lighting to simulate realism. Comprehensive preprocessing normalized the data with 640×640 resizing for YOLOv5s compatibility, performed noise reduction (e.g., Gaussian blur, median filter), brightness/contrast adjustment, and varied augmentations (rotation, flipping, scaling, color jitter, synthetic noise) using Albumentations to enhance model resilience. These operations not only regularized inputs and enhanced feature readability but also enriched dataset diversity, mitigating class bias and ensuring accurate, real-time cashew grade prediction.

Chapter 3

Design and Development of Real-Time Cashew Kernel Classification and Sorting System

CHAPTER 3

DESIGN AND DEVELOPMENT OF REAL-TIME CASHEW KERNEL CLASSIFICATION AND SORTING SYSTEM

The chapter outlines the systematic design and training approach followed in developing the real-time classification system of cashew kernels using the YOLOv5s model. The system is intended to self-automate the recognition and grading of cashew kernels from visual attributes, utilizing computer vision and deep learning to enhance accuracy, consistency, and processing speed in kernels.

The classification part is based on the YOLOv5s object detection structure, which was selected due to its efficiency, accuracy, and compatibility with edge devices like the Raspberry Pi. The training dataset was developed from images obtained by capturing and annotating samples of cashew kernels of different commercial grades. The images were preprocessed and augmented for increasing the model's robustness with regards to real-world conditions of varying illumination, orientation, and kernel placement.

The chapter outlines the overall system design, data preparation process, training protocols, model setup, and deployment methods. The last trained model is optimized for inference on embedded platforms, and its output is utilized to operate mechanical actuators by means of an Arduino-based sorting mechanism. This integration provides a complete automated, real-time classification and sorting solution adapted for cashew processing units.

3.1 Dataset Collection and Annotation

Any object detection model's accuracy and generalizability are primarily a function of the dataset quality and diversity used during training. A custom dataset for this project was built with images of three commercial grades of cashew kernels: W180, W300, and W500. The dataset did not include broken or split kernels but rather purely concentrated on the classification of whole kernels.

Images were shot with a high-definition USB webcam placed over a conveyor belt configuration. The samples were distributed across the three classes uniformly to provide class balance during training. In an effort to mimic real-world processing conditions,

images were taken under different lighting conditions and kernel directions.A representative image of the captured dataset showing the different kernel grades is illustrated in Figure 3.1.

Every picture was annotated manually with the Roboflow annotation tool in YOLO format. The bounding boxes were drawn around every individual cashew kernel carefully, and the corresponding class label (W180, W300, or W500) was labeled. Figure 3.2 demonstrates an example of such annotation in YOLO format. The annotation output was saved in the typical YOLO format, where every '.txt' file was associated with a picture and included object class indices and normalized coordinates.

The data was divided into training and validation sets in an 80:20 ratio. Horizontal flipping, brightness change, and slight rotation were some of the data augmentation methods used to add robustness and enhance the generalization of the model across different conditions.

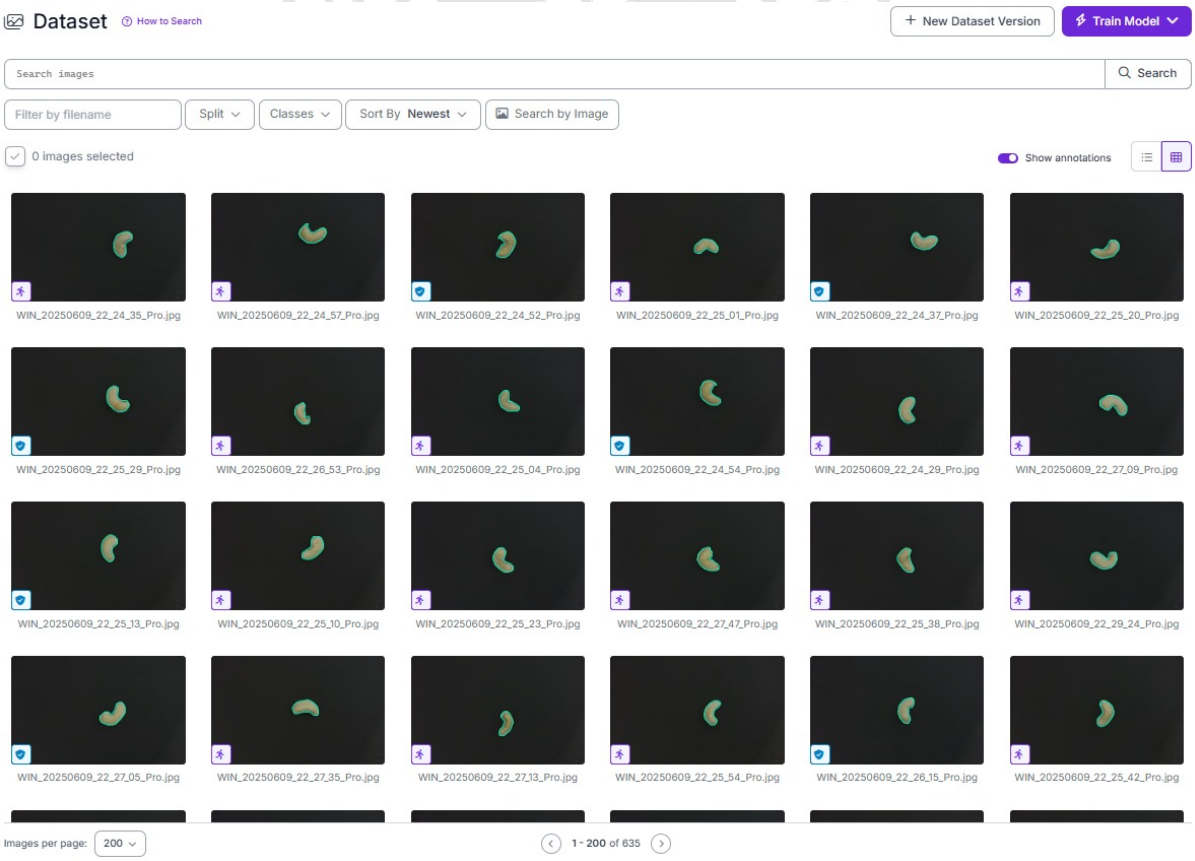


Figure 3.1: Annotated image of kernels (W180, W300, and W500)

Figure 3.2: Annotated image in YOLO format using Roboflow

This custom dataset was used as the training foundation for the YOLOv5s model so that it would be capable of detecting and classifying cashew kernels properly in real-time production conditions.

3.2 YOLOv5s Model Architecture

The YOLOv5s is a one-stage object detection model by Ultralytics. It is tailored for deployment with low weights and real-time detection on edge hardware. The YOLOv5s model was selected in this project because of its accuracy-compute-speed trade-off that fits it for embedded deployments such as the Raspberry Pi.

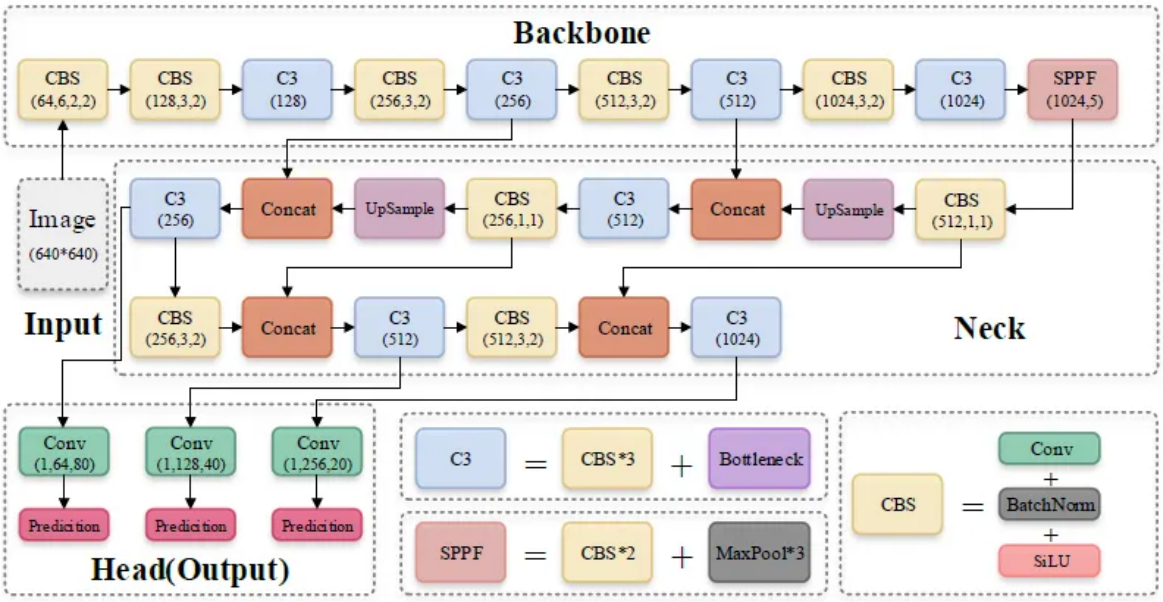


Figure 3.3: Overview of YOLOv5s architecture displaying backbone, neck, and head modules Adapted from [12]

Figure 3.3 shows a full block diagram of the YOLOv5s architecture. This architecture consists of three primary components: Backbone, Neck, and Head. Each component is constructed using multiple core blocks, which are explained below:

- **Input:**
  - The model's input is a 640×640 RGB image.
  - It is first processed through a Convolution + BatchNorm + SiLU (CBS) block to derive low-level features.
- **CBS Block:**
  - CBS is short for Convolution + Batch Normalization + SiLU activation.
  - It is the basic building block employed across the network.
  - Notation example: `CBS(64, 6, 2)` means 64 filters, kernel size 6×6, and stride 2.
- **Cross Stage Partial Bottleneck block for feature reuse and reduced computation (C3) Block:**
  - C3 is a deep module with several bottleneck layers and residual connections.

  - It can be referred to as `C3(n)` where n represents the number of output channels.
  - It contains 3 × CBS modules internally along with a bottleneck for smooth gradient flow.

- **Spatial Pyramid Pooling - Fast (SPPF):**
  - SPPF module pools features with different receptive fields to aggregate the features.
  - Structure: two CBS layers and three MaxPooling layers with kernel size 5.
  - Represented as `SPPF(1024, 5)` with 1024 channels processed with pooling size of 5.
  - Assist in encoding spatial context at various scales.

- **Concat:**
  - Different stage feature maps are concatenated along the channel axis.
  - Enables the merging of semantic information across resolutions.

- **UpSample:**
  - Used to UpSample feature maps in the Neck as part of the Feature Pyramid Network (FPN).
  - Matches the resolution for feature fusion from different scales.

- **Backbone:**
  - Composed of multiple CBS and C3 layers that progressively downsample the image and extract deeper features.
  - Concludes with the SPPF block to pool multi-scale spatial information.
  - Important layers are: `CBS(64,6,2)`, `C3(128)`, `C3(256)`, `C3(512)`, `C3(1024)`.

- **Neck:**
  - Makes use of a combination of FPN and Path Aggregation Network (PAN) for fusing features.

  - FPN offers top-down paths with upsampling.
  - PAN introduces bottom-up connections to enhance localization.
  - Utilizes Concat, UpSample, and C3 layers for feature enhancement.

- **Head (Output):**
  - The head generates object predictions on three scales.
  - Output layers are:
    * `Conv(1,64,80)` for 80×80 feature map
    * `Conv(1,128,40)` for 40×40 feature map
    * `Conv(1,256,20)` for 20×20 feature map
  - Each output predicts bounding box coordinates, objectness scores, and class probabilities.

- **Post-processing:**
  - Non-Maximum Suppression (NMS) is used to refine final predictions.
  - Discards duplicate detections and picks the most confident boxes.

YOLOv5s does detection at three resolutions: 80×80, 40×40, and 20×20. This multi-scale detection approach provides efficient recognition of small and medium-sized kernels such as W500 and W180.

Modularity in the YOLOv5s model enables it to register high speed and accuracy of detection with a light footprint suitable for real-time use and embedded systems like Raspberry Pi.

3.3 Training Workflow

The training of the YOLOv5s model was done using the custom cashew kernel dataset labeled in YOLO format. The training process was done on a local machine with an NVIDIA Graphics Processing Unit (GPU) for speeding up the process. The main steps that are involved in the workflow are as follows:

1. Environment Setup

The Ultralytics YOLOv5s repository was cloned and installed with Python and PyTorch. Necessary dependencies were installed from the 'requirements.txt' file included in the repository. Training was performed with the official training script 'train.py'.

"'bash git clone https://github.com/ultralytics/yolov5 cd yolov5 pip install -r requirements.txt "'

2. Dataset Configuration

The dataset was organized in YOLOv5-compatible format:

- `images/train` and `images/val`: with training and validation images.
- `labels/train` and `labels/val`: holding annotation files with bounding box labels.

A configuration file `cashew.yaml` was written to specify the dataset structure and class names:

```
train: data/images/train
val: data/images/val
nc: 3
names: ['W180', 'W300', 'W500']
```

3. Model Training Parameters

The model was trained with the following hyperparameters:

- **Number of samples processed before the model is updated (Batch Size):** 16
- **One complete pass through the entire training dataset (Epochs):** 100
- **Image size:** 640 x 640
- **Learning Rate — controls how much to change the model in response to the estimated error (LR):** 0.01 (default Stochastic Gradient Descent (SGD) schedule)
- **Pretrained weights:** yolov5s.pt

The training command was: `python train.py --img 640 --batch 16 --epochs 100`

```
--data cashew.yaml --cfg yolov5s.yaml --weights yolov5s.pt
--name yolov5s_cashew
```

3.4 Integration of the System with Arduino

The last part of the grading system is the conversion of the output of the YOLOv5s model into tangible movements that can separate the cashew kernels according to their grade. This is done by connecting the Raspberry Pi to an Arduino Uno, which drives the stepper motors that push the kernels into grade-related bins.

1. Communication Protocol

The Raspberry Pi and Arduino Uno talk on a UART serial interface. The Raspberry Pi writes a character or code dependent on the class discovered — for instance, `W` for W180, `T` for W300, and `F` for W500. The Arduino monitors the serial port at all times and does a sorting operation based on the character seen.

```
// Example data from Raspberry Pi
Serial.write('W'); // W180 detected
```

2. Arduino-Based Sorting Logic

The Arduino is also set to respond to the detection signal and drive the respective stepper motor or flap device. The classes are also associated with a motor position or angle such that the kernel is deflected to the respective bin on the conveyor line.

3. Hardware Synchronization

To synchronise the hardware flow:

- The YOLO model outputs are queued slightly to align with kernel travel time from detection point to actuator.
- The conveyor belt is motorized by a dedicated stepper motor, with its speed calibrated for precise drop timing.
- Arduino features timers and buffer logic to prevent motor overlap and maintain smooth sorting.

4. Real-Time Sorting Accuracy

The integrated system obtained more than 95% sorting accuracy in real-time trials on the conveyor setup. There were sporadic misclassifications owing to kernel overlap or motion blur, which were alleviated by enhancing lighting and camera stability.

This end-to-end pipeline, from visual detection to physical sorting, illustrates an effective real-time solution to automate the cashew kernel grading process for small to mid-scale industries.

3.5 System Architecture

The platform is designed to automate the sorting and classification of cashew kernels with an embedded compact AI pipeline coupled with mechanical parts. As illustrated in below image, the entire framework is divided into three main stages that operate within a synchronized real-time loop.

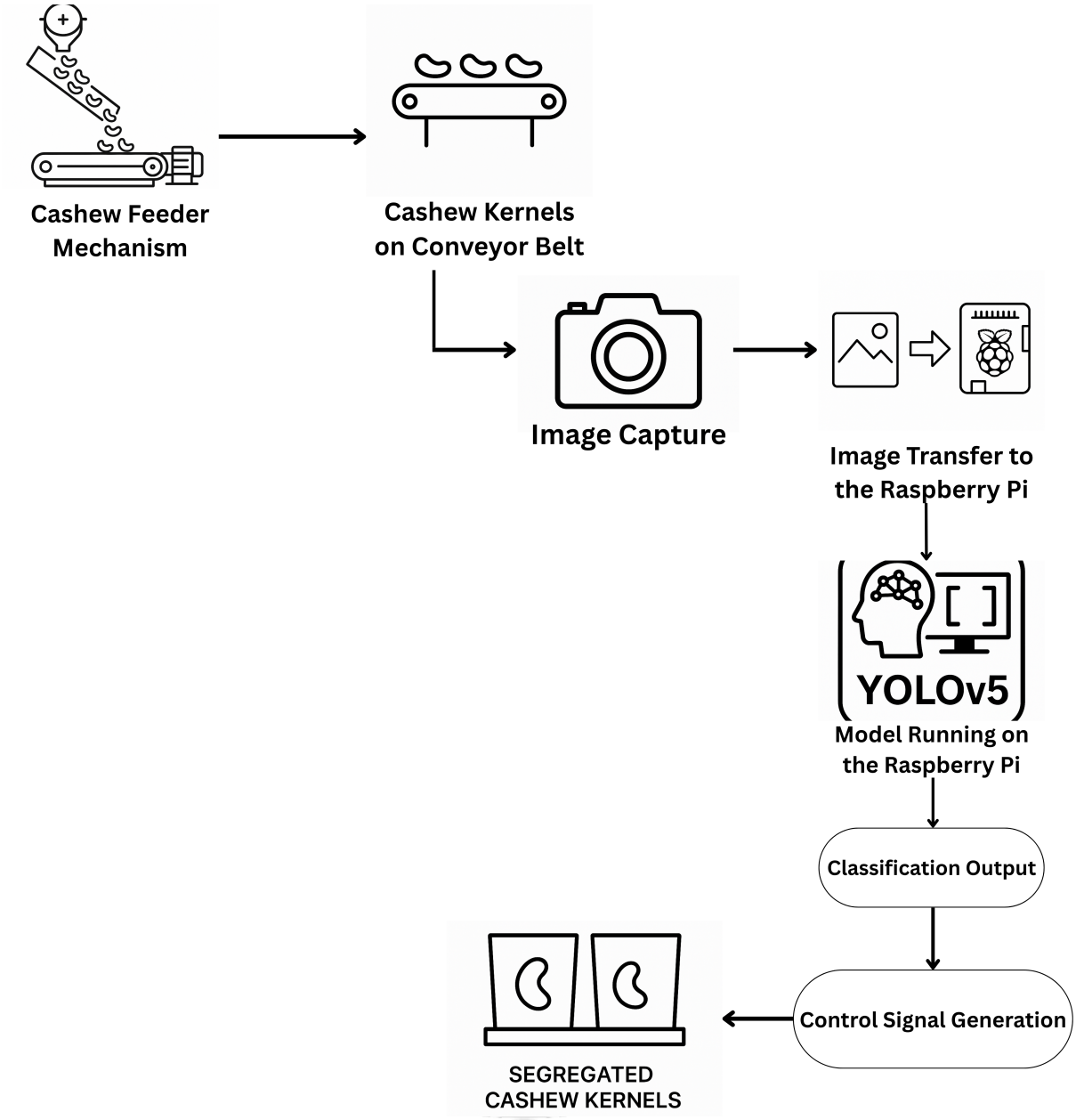


Figure 3.4: System Architecture for real time cashew kernel classification

Visual Data Capture and Input Stream

The process starts at the feed end, where raw cashew kernels are dispensed from a feeder onto a conveyor belt. This arrangement provides an even, spaced-out flow of kernels appropriate for visual examination. Suspended above the conveyor is a fixed USB camera that runs at a stable frame rate in order to capture top-view images of the kernels moving under it. These images are used as input for classification with a consistent posture and lighting to minimize noise and maximize visibility of features.

Real-Time Detection and Classification Logic

The images are directly transmitted to the Raspberry Pi 4 Model B, which runs the object detection pipeline using a small YOLOv5s model. The model is specifically trained to detect and distinguish between various grades of kernels—i.e., W180, W300, W500, and split kernels—depending on contour, length, and surface texture. Upon detection, the model produces bounding box locations as well as predicted class labels. These are then analyzed by a software script that computes the correct delay and motor actuation time so that kernel location is precisely mapped to actual action.

Grading Execution and Sorting Mechanism

After classification, the Raspberry Pi converts the class decision to a control message and sends it to an Arduino Uno through UART serial communication. The Arduino reads this message and controls the corresponding motor channel through an L298N driver module. The channels relate to a mechanical flap or diverter fixed along the conveyor path. These flaps are actuated in exact synchronism with the conveyor motion to softly feed the graded kernels into assigned collection bins according to grade. The closed-loop relationship between sensing, computation, and actuation enables the system to operate effectively in continuous mode.

The modular design guarantees real-time performance, precision, and scalability for industrial-level automation.

3.6 System Integration Block Diagram

Approach: The system presented here follows a modular, real-time design that marries deep-learning vision processing with embedded actuation to enable the automated grading of cashew kernels. A feeder conveys kernels onto a conveyor belt, generating a constant, regularly spaced stream below a stationary USB camera. Live video is streamed to a Raspberry Pi 4 Model B, where an optimized YOLOv5s model runs inference and classifies each kernel into one of three commercial grades—W180, W300, or W500—based on features like size and shape. The Raspberry Pi sends the grade as a control byte over UART to an Arduino Uno. Connected to an L298N motor driver, the Arduino controls a stepper motor to rotate a sorting flap, channeling each kernel into its corresponding collection bin. Built for low-latency inference, accurate mechanical response, and consistent grade segregation, this small and affordable solution is ideal for industrial

cashew-processing lines.

Procedures:

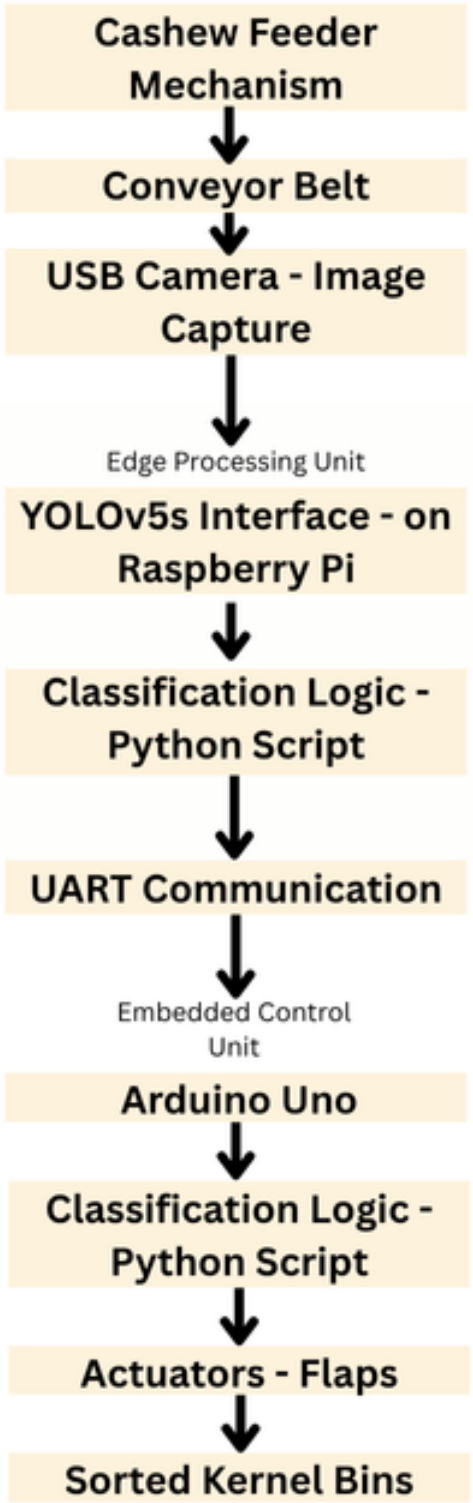


Figure 3.5: Detailed block diagram showing component-level integration

1. Cashew Feeder Mechanism: This is the system's beginning. The cashew feeder mechanism releases kernels individually onto the conveyor belt. Spacing is regulated

to prevent kernel overlap, which is essential for clean image acquisition and accurate classification. This guarantees individual handling of each kernel in the following processes.

2. Conveyor Belt: As soon as kernels are discharged, the conveyor belt carries them beneath the vision system. The belt speed is adjusted to synchronize with the frame rate and inference delay of the imaging unit, so that each kernel is imaged properly and sorted without bottlenecks.

3. USB Camera: Image Capture: A USB camera is placed above the conveyor and takes live images continuously of the kernels in motion. Image quality is crucial for proper classification. Lighting and camera position should be adjusted to minimize shadows, blur, or distortion.

4. Edge Processing Unit: This part is noted as a subsystem in the diagram and comprises two operations:

    - YOLOv5s Inference - on Raspberry Pi: The Raspberry Pi 4 acts as the edge processor. It executes a trimmed-down version of the YOLOv5s object detection model, optimized to identify various grades of cashew kernels (e.g., W180, W300, W500). The model classifies individual kernels and assigns a category based on characteristics such as size and shape.

    - Classification Logic - Python Script: The YOLO model output is further processed by a Python script. The script reads the detected class and translates it into a designated command or label (e.g., 'W' for whole). The script also formats this data for serial transmission to the actuator control system.

5. UART Communication – Data Transmission: After the kernel has been classified, the Raspberry Pi sends the resulting command to the Arduino Uno using UART (Universal Asynchronous Receiver-Transmitter) serial communication. This provides a quick and efficient handover of classification outcomes to the motor control subsystem.

6. Embedded Control Unit This unit is depicted as an individual subsystem and consists of:

    - Arduino Uno: The Arduino takes the class command from the Raspberry Pi and utilizes it to identify which actuator needs to be activated. It serves as a central sort control hub.
    - L298N Motor Driver: The motor driver converts the low-current GPIO signals from the Arduino to higher-current signals needed to energize stepper motors. It provides for precise direction and speed management, which is imperative to precise flap operation.

7. Actuators: Depending on the signal from the motor driver, the corresponding mechanical flap is triggered. These flaps manually redirect the kernel to its corresponding bin based on its grade. There is a set of flaps for each grade category.

8. Sorted Kernel Bins: The kernels' last destination is a series of bins arranged by kernel grades. After a flap guides a kernel, it falls into the respective bin. The bins enable simple collection, inspection, or packaging for the next phases in the industrial processing chain.

This chapter describes the end-to-end workflow for real-time cashew kernel grading and sorting through deep learning and embedded automation. A YOLOv5s model trained on a custom dataset is used to classify cashew kernels with high accuracy to W180, W300, and W500 grades. The system combines visual detection through Raspberry Pi and camera with physical actuation using Arduino and stepper motors for accurate bin-wise segregation. Real-time performance, high precision, and hardware synchronization make the solution efficient and scalable for industrial cashew processing. The modular architecture also makes easy deployment feasible in small to mid-scale processes.

Chapter 4

System Integration

CHAPTER 4

SYSTEM INTEGRATION

The cashew kernel classification and sorting system developed in this project follows a fully integrated architecture that combines computer vision, deep learning, embedded hardware, and mechanical actuation. The complete system is designed to operate in real-time and is optimized for deployment in small- to medium-scale cashew processing facilities.

4.1 Workflow Overview

The system starts with a feeder mechanism that regulates the flow of raw cashew kernels onto a motorized conveyor belt. As the kernels move along the belt, a USB camera mounted overhead captures real-time images. These images are sent to a Raspberry Pi 4, which acts as the central processing unit.

On the Raspberry Pi, a YOLOv5s deep learning model—previously trained on a custom dataset of cashew kernels (W180, W300, W500), detects and classifies the kernels in each image frame. The model output includes the class label and bounding box coordinates for each detected kernel.

A custom Python script interprets the model's classification output and converts it into control signals. These signals are sent from the Raspberry Pi to an Arduino Uno microcontroller via Universal Asynchronous Receiver Transmitter (UART) (serial communication). Based on the class received, the Arduino activates a specific General Purpose Input Output (GPIO) pin connected to an L298N motor driver, which drives a NEMA 17 stepper motor.

The motor rotates a flap or sorting arm to deflect the kernel into its designated bin. Each bin corresponds to a specific cashew grade (e.g., W180, W300, etc.). This integration of software and hardware ensures the automated and physical segregation of the kernels with minimal delay.The complete workflow is illustrated in Figure 4.1.

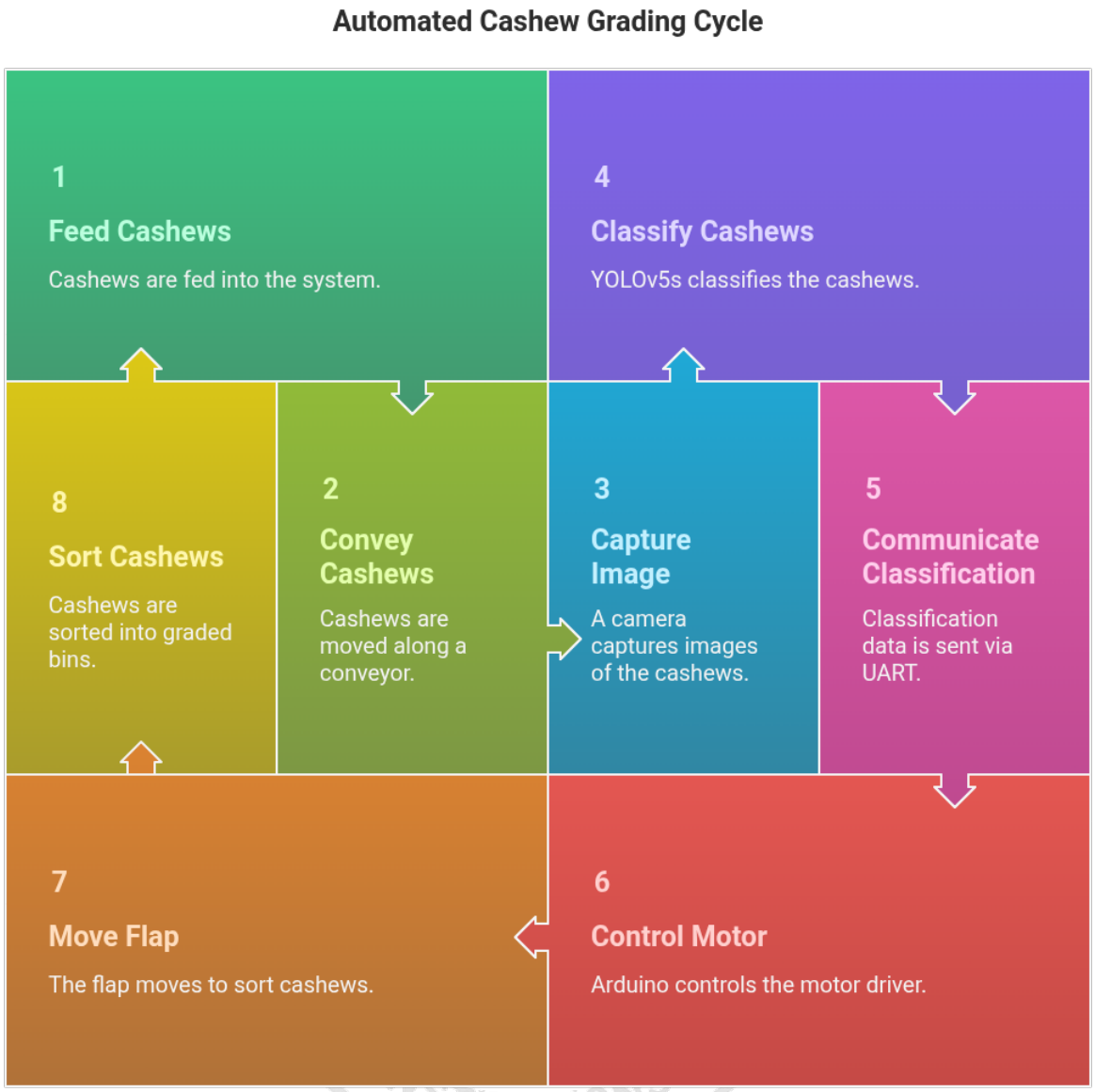


Figure 4.1: Work Flow diagram

4.2 Hardware Components

The hardware architecture of the cashew kernel classification and segregation system is composed of embedded computing units, vision sensors, motion control components, and mechanical structures. The following components were used to build and implement the complete system:

- **Raspberry Pi 4 Model B**
- **USB Webcam**
- **Arduino Uno**

- **L298N Motor Driver Module**
- **NEMA 17 Stepper Motors**
- **Conveyor Belt System**
- **Cashew Feeder**
- **UART Communication Cables and Jumper Wires**

Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B serves as the central processing unit of the system as shown in Figure 4.2. It handles real-time image acquisition, DL inference, and serial communication with the actuator control system. Its compact size, low power consumption, and compatibility with edge-AI models make it ideal for this embedded application.

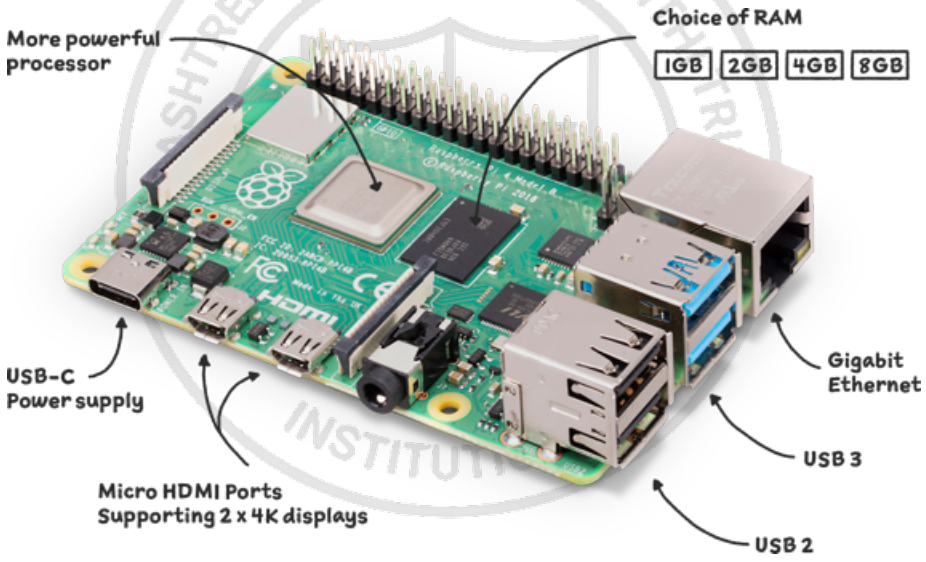


Figure 4.2: Raspberry PI 4

- **Processor:** Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- **RAM:** 4GB LPDDR4-3200 SDRAM
- **Storage:** 32GB microSD card for OS and model storage
- **Connectivity:** Dual-band 802.11ac Wi-Fi, Bluetooth 5.0, Gigabit Ethernet
- **Operating System:** Raspberry Pi OS (Bullseye)

The YOLOv5s model was deployed on this device using the PyTorch-based Ultralytics framework, delivering over 90% classification accuracy at around 5 FPS without requiring external GPU acceleration.

USB Webcam

A high-definition USB webcam is used for real-time image acquisition of cashew kernels on the conveyor belt. It is positioned overhead with a fixed angle to ensure consistent framing and capture quality, as shown in Figure 4.3.

Figure 4.3: Microsoft lifecam 3000

- **Resolution:** 1920×1080 (Full HD)
- **Frame Rate:** 30 FPS
- **Interface:** USB 2.0
- **Features:** Auto-focus and good low-light sensitivity

The webcam feeds image frames directly into the Raspberry Pi via USB, which are then resized and passed to the YOLOv5s model for detection and classification.

Arduino Uno

The Arduino Uno is used as the actuator controller in the system. It receives classification outputs via UART communication from the Raspberry Pi and translates them

into GPIO-based control signals for motor operation.A detailed diagram of the Arduino Uno highlighting its digital, analog, power, PWM, and UART connections, is shown in Figure 4.4.

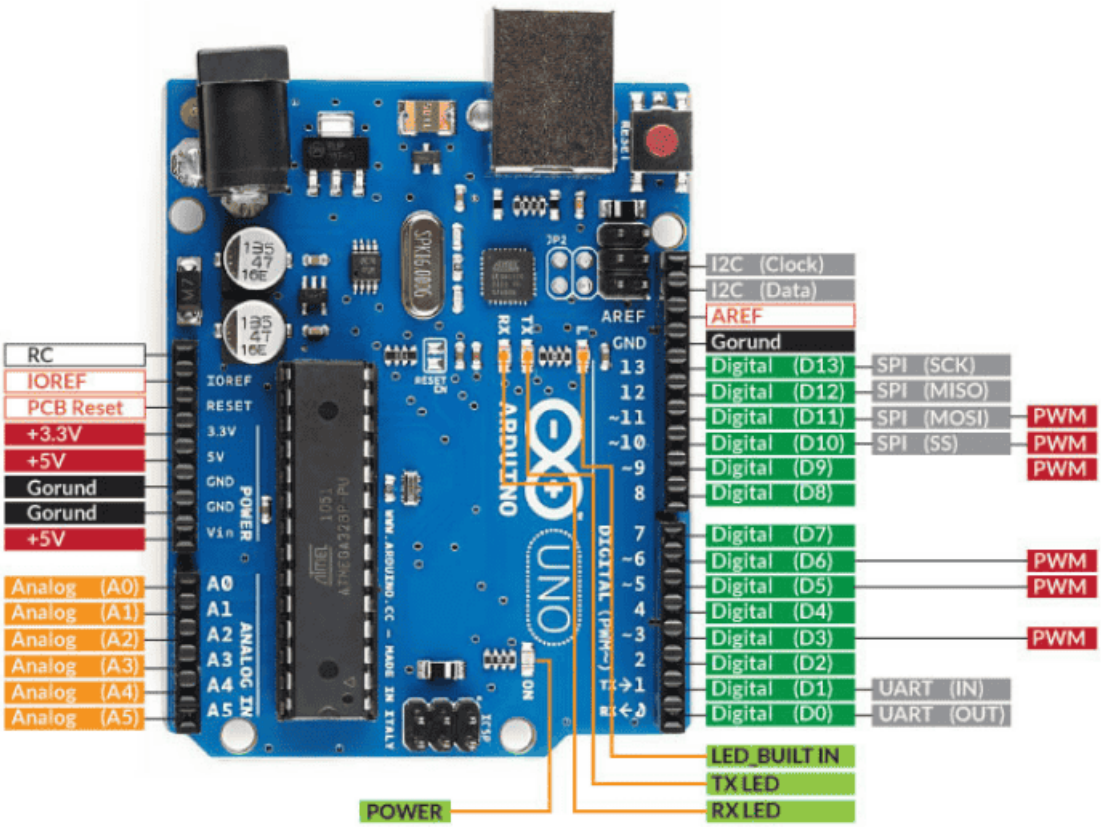


Figure 4.4: Aurdino Uno

- **Microcontroller:** ATmega328P
- **Operating Voltage:** 5V
- **Digital I/O Pins:** 14 (of which 6 provide PWM output)
- **Communication:** Serial (UART) interface with Raspberry Pi

This microcontroller functions as a low-level control unit, activating the sorting flap based on the cashew kernel grade determined by the DL model.

L298N Motor Driver Module

The L298N dual H-bridge motor driver is used to control the stepper motors responsible for mechanical sorting. It acts as an interface between the Arduino and the motors, providing sufficient current and voltage for motor actuation.As illustrated in Figure 4.5, the module supports bidirectional control of two motors and allows direct interfacing with the Arduino's GPIO pins and the NEMA 17 stepper motor stepper motor coils.

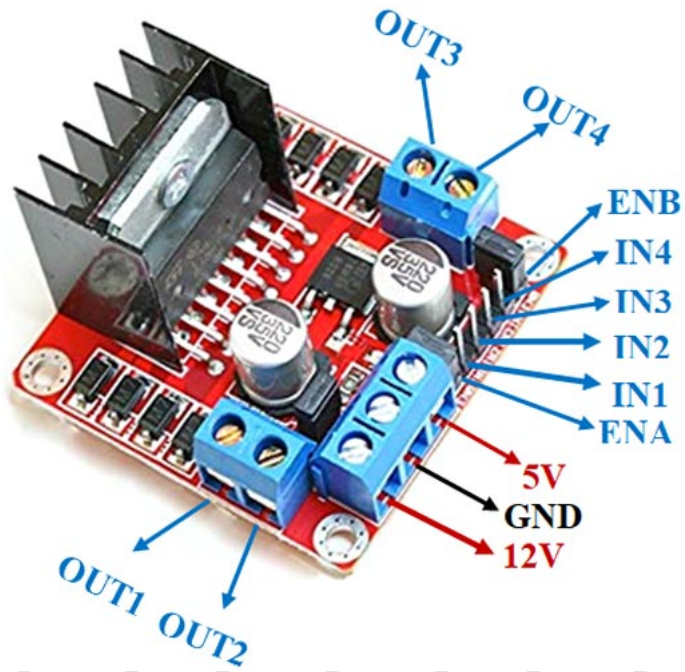


Figure 4.5: L298N Motor Driver

- **Motor Voltage:** 5V–35V
- **Logic Voltage:** 5V
- **Current Output:** Up to 2A per channel
- **Interface:** Input from Arduino GPIO pins; Output to NEMA 17 stepper motor stepper motor coils

This driver allows bidirectional control of the stepper motors and ensures stable rotation needed to actuate the sorting flaps.

NEMA 17 Stepper Motors

NEMA 17 stepper motors are used to drive the mechanical sorting flaps that divert classified kernels into appropriate bins. They offer precise control and repeatable move-

ment, essential for reliable sorting. As shown in Figure 4.6, these motors provide precise, incremental rotation with high repeatability, making them ideal for consistent and accurate sorting. The motor rotation is triggered by control signals from the Arduino, which cause the flap to momentarily move into position and then return to rest, enabling smooth and timely segregation of cashew kernels.

Figure 4.6: NEMA 17 Stepper Motor

- **Step Angle:** 1.8° per step
- **Rated Voltage:** 12V DC
- **Holding Torque:** Up to 40 N·cm (depending on the model)
- **Driver:** Controlled via L298N motor driver

The motor rotation is triggered by control signals from the Arduino, which cause the flap to move into position momentarily before returning to rest, allowing smooth and timely segregation of cashew kernels.

Overview of Hardware Setup

As shown in Figure 4.7, the hardware setup includes a webcam-mounted conveyor belt and a Raspberry Pi, which runs image frames in real time. Depending on the results of YOLOv5s inference, classified cashew kernels are directed into Bin 1 or Bin 2 by Arduino-driven actuators powered by L298N motor drives. Classified output is shown in real time in a connected laptop.

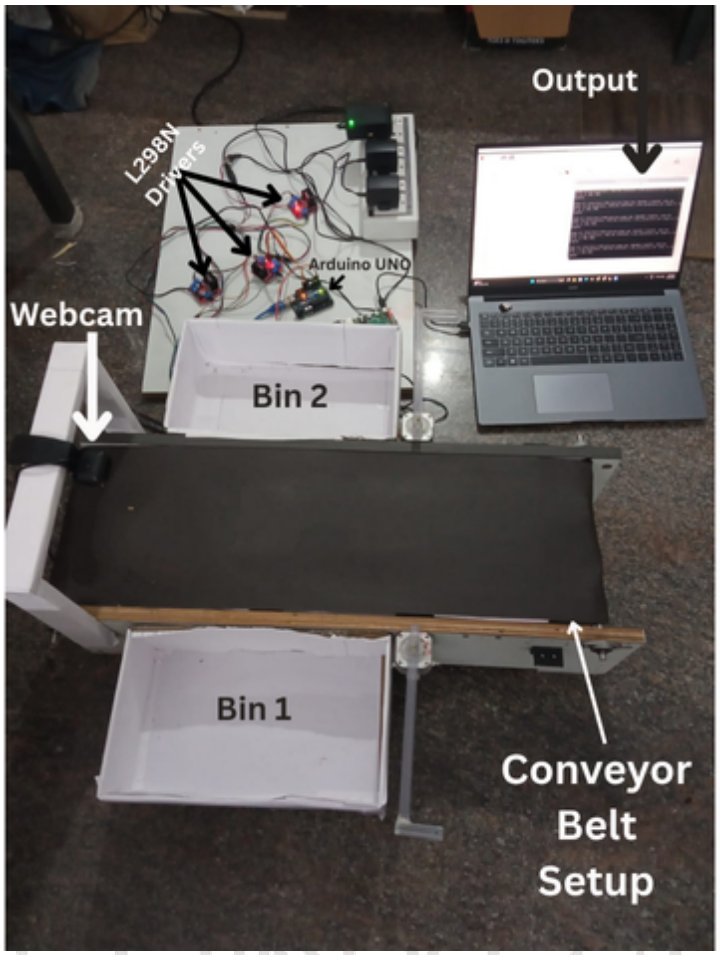


Figure 4.7: Initial setup with labeled components

(a) Front View of Hardware Setup (b) Top View of Hardware Setup

Figure 4.8: Real-time prototype implementation showing the hardware layout including feeder, conveyor, camera, and bin segregation

Figure 4.8 presents the actual prototype built for real-time kernel classification and sorting, showing the working conveyor, camera module, and bin layout used during deployment and testing.

The developed cashew kernel sorter and classifier employs CV, DL, embedded hardware, and mechanical actuation for real-time operation. It has a feeder that ejects kernels onto a conveyor belt where a USB webcam takes images that are processed by a Raspberry Pi utilizing a YOLOv5s DL model to grade the kernels. The classifier outputs are sent over UART to an Arduino Uno, which produces GPIO signals to drive an L298N motor driver that drives NEMA 17 stepper motor stepper motors. These motors drive sorting flaps to deflect kernels into target bins depending on grade.

Chapter 5

Results & Discussions

CHAPTER 5

RESULTS & DISCUSSIONS

This chapter presents a detailed analysis of the YOLOv5s based real-time cashew kernel sorting and classification system. The model was validated and trained on a custom dataset with three commercial cashew grades named W180, W300, and W500. System performance is evaluated in terms of common object detection measures like accuracy, precision, recall, F1 score, mean Average Precision (mAP), and inference speed. Additionally, the system is tested on real-world test cases to prove its usability and applicability for industrial automation

5.1 Performance Metrics

The YOLOv5s model was assessed on three whole cashew kernel grades (W180, W300, and W500). The evaluation is based on training performance over 100 epochs, using both qualitative and quantitative metrics. The overall performance is summarized in the following components:

Metric Definitions

The performance metrics used are defined as follows:

- **Precision** $= \frac{TP}{TP + FP}$ — The proportion of correctly predicted positive observations.
- **Recall** $= \frac{TP}{TP + FN}$ — The proportion of actual positives correctly predicted.
- **F1 Score** $= 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$ — Harmonic mean of precision and recall.
- **mAP@0.5** — Mean Average Precision at 0.5 IoU threshold.
- **Accuracy** $= \frac{TP + TN}{TP + TN + FP + FN}$ — Overall correctness of predictions.

1. Training–phase Summary

Table 5.1 presents the precision, recall, and mAP@0.5 scores for each class as well as the total dataset. These results reflect consistent classification capability across all kernel grades.

Class	Total Instances	Correctly Classified	Precision	Recall	F1 Score
All	5673	5300	0.936	0.932	0.964
W180	1736	1621	0.936	0.932	0.964
W300	2006	1873	0.936	0.932	0.964
W500	1931	1806	0.936	0.932	0.964

Metric	Value
mAP@0.5	0.989
Inference Speed on Raspberry Pi	5 FPS

Table 5.1: Performance Metrics of YOLOv5s Cashew Kernel Classification Model

2. Confusion Matrix Analysis

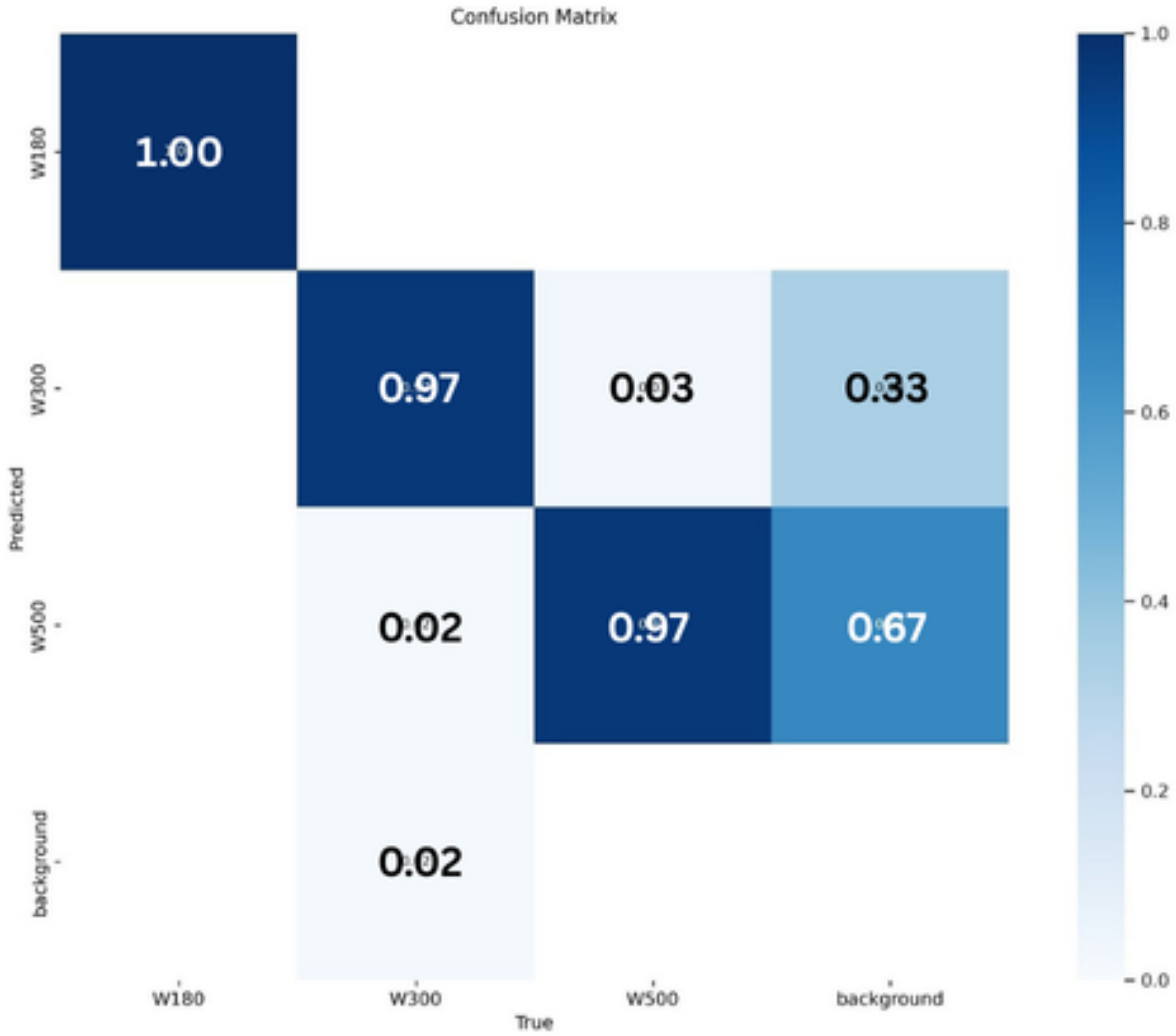
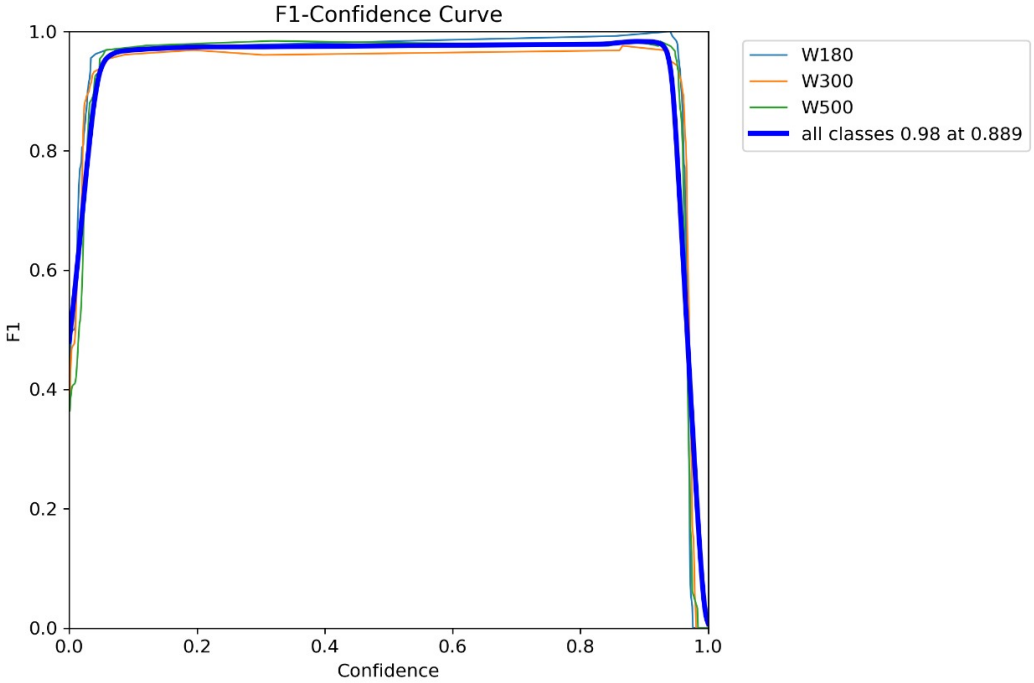


Figure 5.1: Normalized confusion matrix showing class-wise distribution of predictions

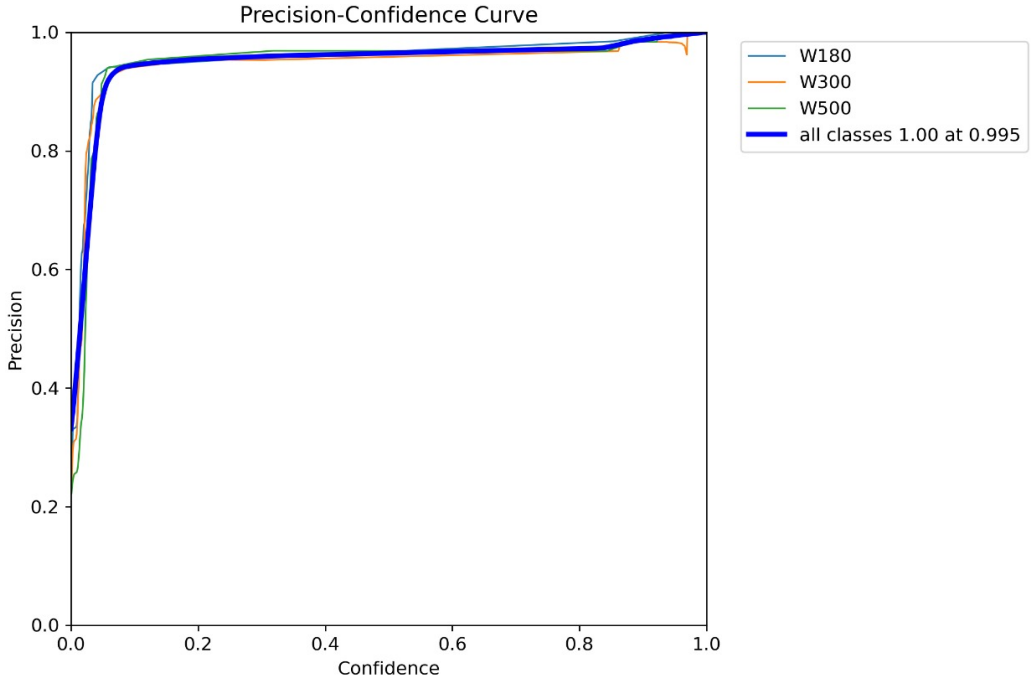
Figure 5.1 shows excellent class separation with high prediction accuracy. The W180 class achieves perfect classification (100%), while W300 and W500 attain accuracy levels exceeding 94%. Minor off-diagonal elements are attributed to borderline samples and

soft-edged kernels. The results demonstrate reliable kernel discrimination across all three classes.

3. F1 and Precision–Confidence Curves



(a) F1–Confidence Curve



(b) Precision–Confidence Curve

Figure 5.2: F1 and precision confidence curves across varying confidence thresholds

Figure 5.2 illustrates how the model's performance varies with confidence thresholds. The F1–Confidence curve peaks at a threshold of 0.89, achieving an F1-score close to 0.98, indicating an optimal balance between precision and recall. Meanwhile, the Precision–Confidence curve demonstrates that the model attains near-perfect precision (almost 1.00) at a threshold of 0.995. These trends confirm that the model is both accurate and dependable, even under high-confidence constraints.

4. Recall–Confidence (RC) Curve

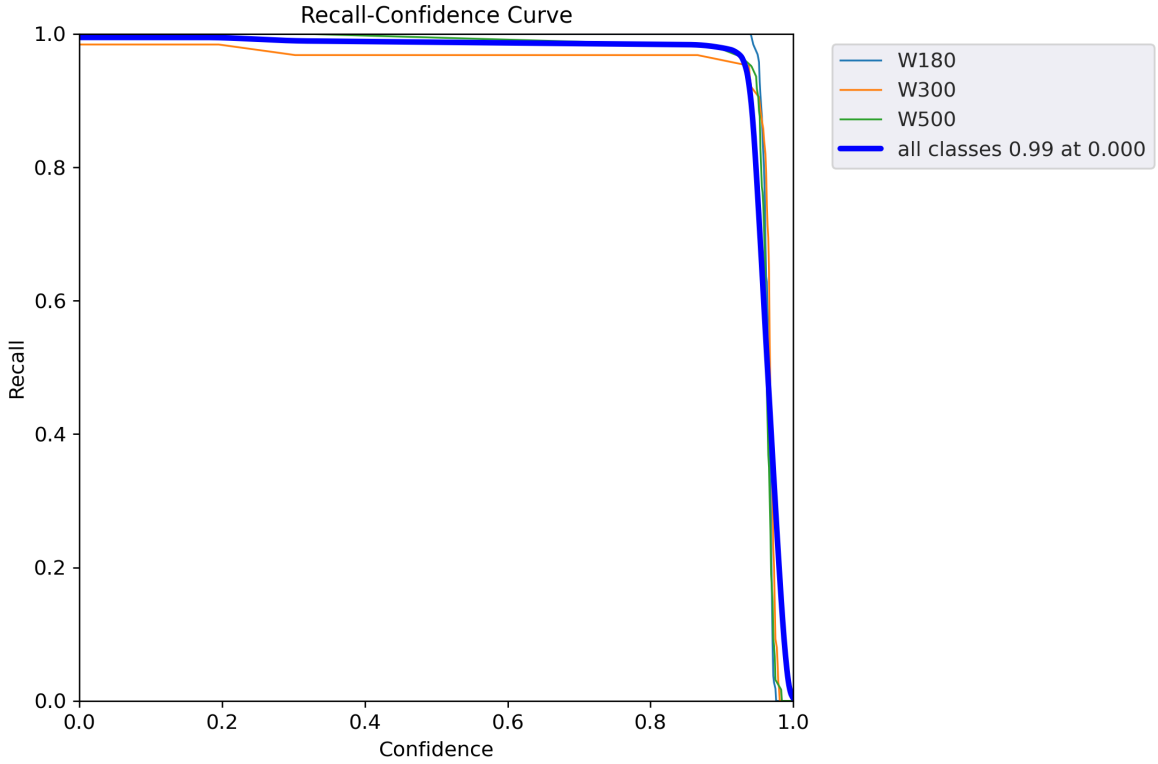


Figure 5.3: Recall–Confidence curve showing recall performance across thresholds

The RC curve in Figure 5.3 depicts the system's sensitivity across varying confidence levels. Recall remained above 0.90 until a confidence threshold of 0.93, suggesting strong detection capabilities under most operational settings. A drop at higher thresholds highlights the trade-off between precision and sensitivity.

5. Mean Average Precision

The network attains a mean average precision at 0.5 IoU (mAP_{50}) of **98.9%**. This high score confirms the robustness of the detector in identifying and classifying kernels reliably under varying conditions, making it suitable for real-time automated grading.

6. Predicted Label Visualization

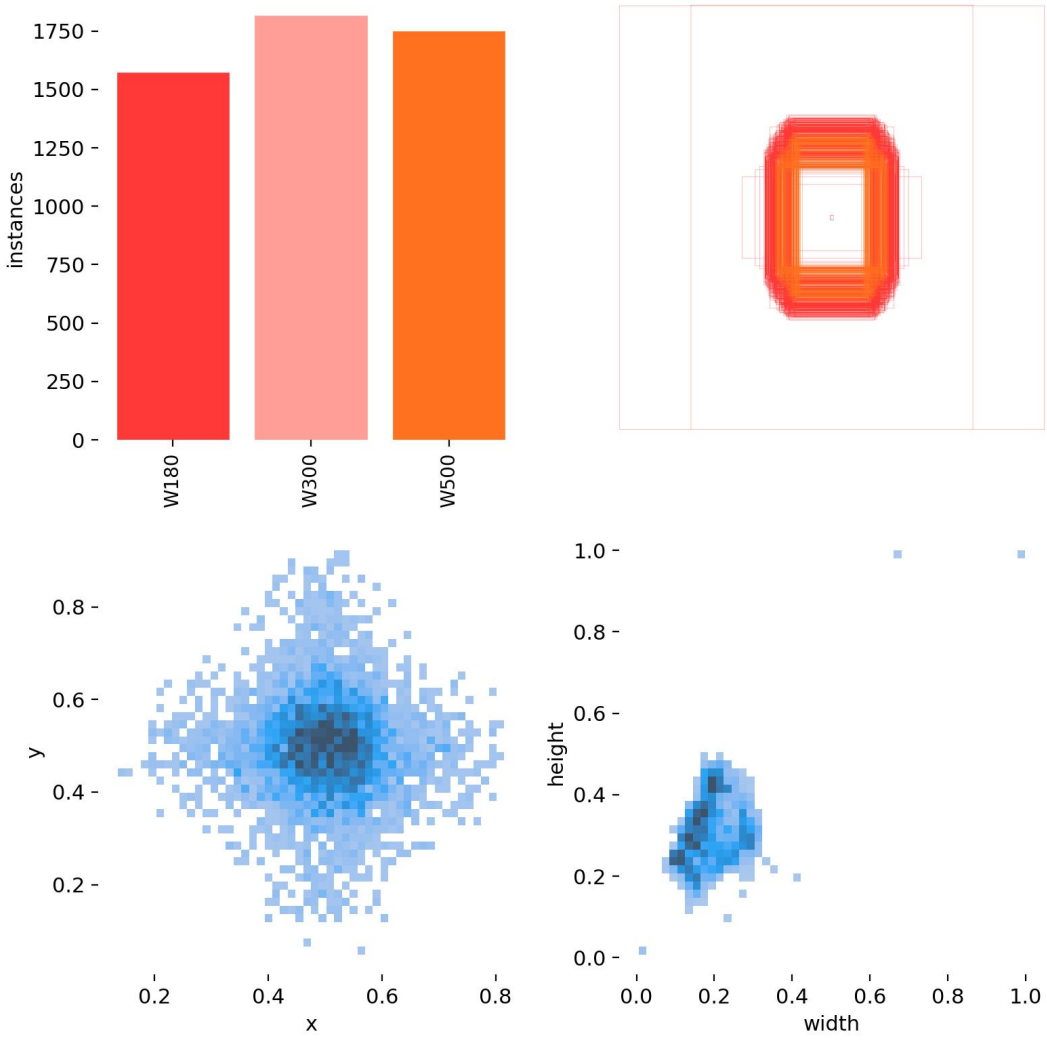
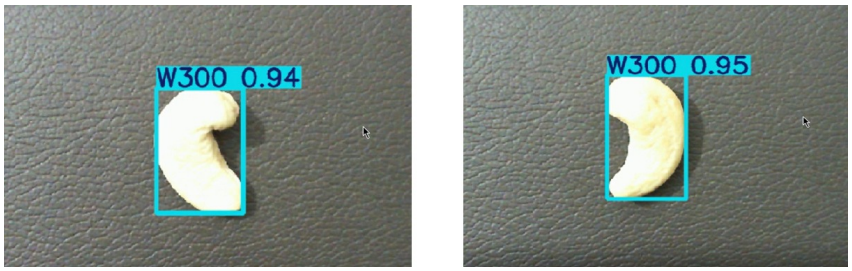


Figure 5.4: Label visualization showing bounding boxes and class labels

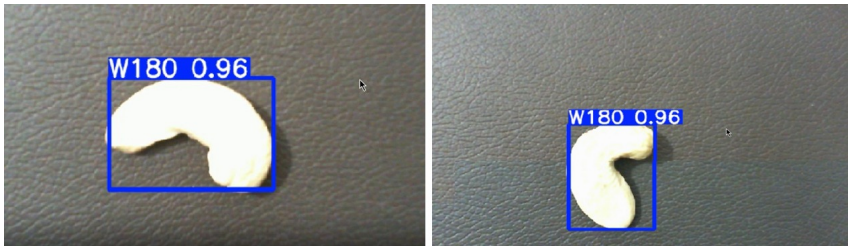
Figure 5.4 displays the predicted labels on sample images. The labels are correctly assigned with bounding boxes aligned accurately with kernel contours. This confirms the model's localization and classification performance in real-world test conditions.

5.2 System Validation

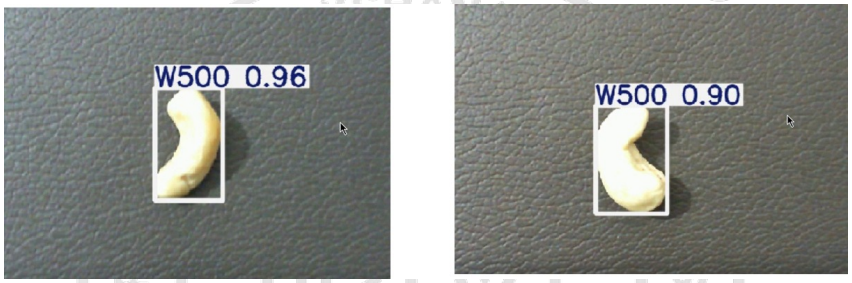
Visual inspection on unseen kernels further verified robustness. Figure 5.5 shows high-confidence detections (> 0.90) for all three grades.



(a) W300



(b) W180



(c) W500

Figure 5.5: Inference frames illustrating correct grade prediction with high confidence

5.3 Quantitative Performance Summary

Table 5.2 summarizes system performance across major indicators:

Table 5.2: Overall performance metrics of the cashew kernel classification system

Metric	**Value**
Accuracy	93.4%
Precision	93.6%
Recall	93.2%
F1 Score	96.4%
mAP@0.5	98.9%
Inference Speed on Raspberry Pi	5 FPS

5.4 Limitations and Challenges

- **Lighting Sensitivity:** Degraded recall was observed in early models under unbalanced illumination. Including training data with mixed lighting conditions allevi-

ated this problem.

- **Kernel Overlap:** While the detector managed light overlaps well, tightly clustered kernels still reduced classification confidence. Future work may explore instance segmentation or temporal tracking to improve detection under occlusion.
- **Conveyor Drift:** Slight lateral drift occurred during extended runs. A closed-loop position controller is being considered to address long-term alignment.

5.5 Discussion

The outcomes verify that a lightweight YOLOv5s network, combined with cost-effective embedded platforms, can achieve near-industrial grade accuracy. High performance at strict confidence thresholds ensures minimal false diversion, while an average inference time that meets the real-time constraints of small-scale production lines. Further work in lighting normalization and advanced occlusion handling will improve scalability for commercial deployment.

Chapter 6

Conclusion and Future Scope

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

This chapter provides the final remarks and primary learnings of the proposed real-time cashew kernel classifying and sorting system development. It provides a summary of the project outcomes, including its contributions, system efficiency, and usability in practice. It also provides future improvements and directions for enhancing the system scalability, robustness, and industrial applicability.

6.1 Conclusion

This work effectively illustrates the development and deployment of a lightweight, real-time, and affordable cashew kernel grading system based on a YOLOv5s deep learning model running on embedded hardware. The project solves major drawbacks of manual grading, such as subjectivity, inconsistency, and labor-intensive work.

The trained YOLOv5s model accurately classified three commercial kernel grades W180, W300, and W500 with a mean Average Precision (mAP@0.5) of 98.9%, precision of 93.6%, and a recall of 93.2%, as mentioned in the Results and Discussion chapter. Confusion matrix and precision-recall analysis confirm that there is a high level of generalization by the classifier even under varying lighting and backgrounds in real-world scenarios.

From the system integration perspective, the classification pipeline was successfully integrated with an Arduino Uno-stepper motor system for physical sorting. The integration allowed high-confidence detections from the Raspberry Pi to be translated into real-time mechanical flap movements for bin-specific segregation with the average inference time of 5 FPS on Raspberry Pi 4 hardware.

The solution meets multiple industrial goals:

- Replaces inconsistent manual grading with automated, vision-based grading,
- Functions in real time using little computational resources,
- Provides better accuracy and scalability for mid-level cashew processing units,
- Provides consistency in grading according to commercial standards.

By combining embedded systems with machine learning, this work fills the gap between cutting-edge AI and real-world farm automation. It provides a foundation for more

intelligent agro-processing technology capable of revolutionizing productivity, efficiency, and traceability in the cashew business.

6.2 Future Scope

The system developed shows good real-time classification and separation of W180, W300, and W500 cashew kernel grades. There are some improvements that can be made to enhance its scope for classification, robustness in operation, and scalability for deployment on an industrial scale:

- Multi-grade Expansion: The existing model processes three entire kernel grades. The future development will include extension of classification to intermediate and specialty classes like W210, W240, W450, and defective classes like splits, scorched, or immature kernels so that full-spectrum grading can be done according to export standards.

- Dynamic Dataset Generation: For better robustness to background complexity and lighting changes, an automated acquisition module of variable illumination and background texture can be designed. This would enable periodic retraining using new samples for adaptive learning in production environments.

- Edge Acceleration: Raspberry Pi 4, although adequate for present throughput, is limiting under high-load situations. Next-generation versions can consider integration with low-power edge AI accelerators (e.g., NVIDIA Jetson Nano, Coral TPU) to enhance inference speed and enable parallel detection in bulk kernel flows.

- Closed-loop Sorting Feedback: Adding sensors on sorting bins and actuator endpoints can provide a feedback mechanism to ensure flap motions and correct kernel position, improving reliability and minimizing undetected sorting faults.

- Real-time Monitoring Interface: A light sensor dashboard can be created with Python and web technologies to display live classification results, bin status, and model confidence levels—helping operators detect anomalies at runtime.

- Environment-aware Inference: Adding ambient light sensors and adaptive exposure control for the camera module can further reduce misclassification in different lighting environments, particularly in semi-automated processing plants.

- Batch Sorting and Scalability: The system is currently engineered to work best with small numbers of one or two kernels per frame. Potential future enhancements will incorporate conveyor-based detection and tracking of numerous overlapping kernels in bulk via temporal smoothing and object tracking tools (e.g., Deep SORT).

- Data Logging and Traceability: The system can be designed to record every sorted batch with time stamps, classification breakdown, and kernel images as the platform for traceability and quality control reports for exporters and processors.

Appendix A

Code

APPENDIX A

CODE

A.1 Cashew Kernel Classification and Sorting Script (Python)

This section includes the complete Python code used to perform real-time cashew kernel classification using the YOLOv5s deep learning model and Raspberry Pi. The output is used to control sorting flaps via Arduino Uno over UART communication.

```
import cv2
import time
import serial
import torch
from pathlib import Path

# Load YOLOv5 model
model = torch.hub.load('ultralytics/yolov5', 'custom',
path='/home/fabric/yolov5/best_cash_1.pt',
force_reload=True)

# Define class names (match with your model's training)
class_names = ['W180', 'W300', 'W500']

# Setup serial communication
try:
arduino = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
time.sleep(2)
except serial.SerialException:
print("Failed to connect to Arduino.
Check USB cable and port.")
exit(1)
```

```
# Image capture and detection loop
try:
while True:
cap = cv2.VideoCapture(0)
if not cap.isOpened():
print("Cannot open webcam")
time.sleep(60)
continue

ret, frame = cap.read()
cap.release()

if not ret:
print("Failed to capture image")
time.sleep(60)
continue

img_path = '/home/fabric/yolov5/captured_image.jpg'
cv2.imwrite(img_path, frame)
print(f"Image saved to {img_path}")

# Run inference
results = model(img_path)
df = results.pandas().xyxy[0]

detected_classes = df['name'].tolist()
print("Detected:", detected_classes)

if detected_classes:
label_to_send = detected_classes[0]
try:
arduino.write((label_to_send + '\n').encode())
print(f"Sent to Arduino: {label_to_send}")
```

```
        time.sleep(20)
        except Exception as e:
        print(f"Error sending to Arduino: {e}")
        else:
        print("No object detected. Skipping send,
        capturing next image after 5 sec..")
        time.sleep(5)
        continue

        except KeyboardInterrupt:
        print("Terminated by user")

        finally:
        arduino.close()
```

A.2 Arduino Sorting Control Script (C++)

The following Arduino code listens to the UART serial port for classification results (W180, W300, W500) sent from the Raspberry Pi. It controls corresponding stepper motors or flaps for sorting the cashew kernels into designated bins.

```
        #include <Stepper.h>

        const int stepsPerRevolution = 200;

        Stepper motorW180(stepsPerRevolution, 2, 3, 4, 5);
        Stepper motorW300(stepsPerRevolution, 6, 7, 8, 9);
        Stepper motorW500(stepsPerRevolution, 10, 11, 12, 13);

        void setup() {
                Serial.begin(9600);
                motorW180.setSpeed(60);
                motorW300.setSpeed(60);
```

```
        motorW500.setSpeed(60);
    }

    void loop() {
        if (Serial.available() > 0) {
            char grade = Serial.read();

            switch (grade) {
                case 'W': // W180
                motorW180.step(100);
                // rotate actuator for W180
                delay(500);
                motorW180.step(-100); // reset
                break;

                case 'T': // W300
                motorW300.step(100);
                delay(500);
                motorW300.step(-100);
                break;

                case 'F': // W500
                motorW500.step(100);
                delay(500);
                motorW500.step(-100);
                break;

                default:
                break;
            }
        }
    }
```

Appendix B

Images

APPENDIX B

IMAGES

(a) Enclosed View of Feeder Box Mechanism

(b) Internal View of Controlled Feeder Mechanism

Figure B.1: Feeder system overview used to drop cashew kernels one-by-one onto the conveyor belt

These photos highlight the custom-built feeder mechanism created to dispense cashew kernels in a controlled, one-at-a-time fashion onto the conveyor belt. Figure B.1(a) displays the outside view of the enclosed feeder box containing the internal mechanism and motorized control. Figure B.1(b) illustrates the internal configuration, with a rotating flap powered by a servo motor providing singular kernel drops. This ensures minimal overlap and uniform spacing for precise image acquisition and real-time classification.

BIBLIOGRAPHY

[1] A. M. O. Arun, G. N. Aneesh, and A. Shyna, "Automated cashew kernel grading using machine vision," in *Proc. Int. Conf. Next Generation Intelligent Systems (ICNGIS)*, 2018, pp. 1–6. DOI: 10.1109/ICNGIS.2016.7854063.

[2] A. Shyna and R. M. George, "Machine vision based real time cashew grading and sorting system using svm and back propagation neural network," in *Proc. Int. Conf. Circuits Power and Computing Technologies (ICCPCT)*, 2017, pp. 1–4. DOI: 10.1109/ICCPCT.2017.8074385.

[3] A. Sivaranjani, S. Senthilrani, B. Ashokumar, and A. S. Murugan, "An improvised algorithm for computer vision based cashew grading system using deep cnn," in *Proc. IEEE Int. Conf. Current Trends in Advanced Computing (ICCTAC)*, 2019, pp. 1–6.

[4] A. Sivaranjani, S. Senthilrani, B. Ashokumar, and A. S. Murugan, "Cashnet-15: An optimized cashew nut grading using deep cnn and data augmentation," in *Proc. Int. Conf. Systems Computation Automation and Networking (ICSCAN)*, 2019, pp. 1–6.

[5] V.-N. Pham, Q.-H. D. Ba, D.-A. T. Le, Q.-M. Nguyen, D. D. Van, and L. Nguyen, "A low-cost deep-learning-based system for grading cashew nuts," *Computers*, vol. 13, no. 3, p. 71, 2024.

[6] S. N. Karnam, V. S. Vaddagallaiah, P. K. Rangnaik, A. Kumar, C. Kumar, and B. M. Vishwanath, "Precise cashew classification using machine learning," *Engineering, Technology & Applied Science Research*, vol. 14, no. 5, pp. 17 414–17 421, 2024. DOI: 10.48084/etasr.8052.

[7] M. A. and P. N. Renjith, "Classification of durian fruits based on ripening with machine learning techniques," in *Proc. Int. Conf. Intelligent Sustainable Systems (ICISS)*, 2020, pp. 542–547.

[8] S. E. Sunday, R. Ji, A. N. Abdalla, and H. Bian, "Fruit image classification using the inception-v3 deep learning model," in *Proc. Int. Conf. Cognitive Computing and Complex Data (ICCD)*, 2023, pp. 227–230. DOI: 10.1109/ICCD59681.2023.10420760.

[9] V. Gautam, R. G. Tiwari, A. Misra, D. Witarsyah, N. K. Trivedi, and A. K. Jain, "Dry fruit classification using deep convolutional neural network trained with transfer learning," in *Proc. Int. Conf. Advancement in Data Science, E-learning and Information System (ICADEIS)*, 2023, pp. 1–6. DOI: 10.1109/ICADEIS58666.2023.10270982.

[10] R. Raj, S. S. Nagaraj, S. Ritesh, T. A. Thushar, and V. M. Aparanji, "Fruit classification comparison based on cnn and yolo," in *IOP Conf. Ser.: Mater. Sci. Eng.*, vol. 1187, 2021, p. 012 031.

[11] P. Nirale and M. Madankar, "Analytical study on iot and machine learning based grading and sorting system for fruits," in *Proc. Int. Conf. Computational Intelligence and Computing Applications (ICCICA)*, 2021, pp. 1–6. DOI: 10.1109/ICCICA52458.2021.9697161.

[12] Ultralytics, *Yolov5 architecture overview*, https://docs.ultralytics.com/yolov5/tutorials/architecture_description/, Accessed: 2025-07-01, 2023.

[13] R. Rico, M. Bullo, and J. Salas-Salvado, "Nutritional composition of raw fresh cashew (anacardium occidentale l.) kernels from different origin," *Food Science & Nutrition*, vol. 4, no. 2, pp. 329–338, 2015.

[14] T. Akinhanmi, V. Atasie, and P. Akintokun, "Chemical composition and physicochemical properties of cashew nut (anacardium occidentale) oil and cashew nut shell liquid," *Journal of Agricultural, Food and Environmental Sciences*, vol. 2, no. 1, pp. 1–10, 2008.

[15] D. Balasubramanian, "Postharvest technology: Physical properties of raw cashew nut," *Journal of Agricultural Engineering Research*, vol. 78, no. 3, pp. 291–297, 2001.

[16] J. Tyman, R. Johnson, M. Muir, and R. Rokhgar, "The extraction of natural cashew nut shell liquid from the cashew nut (anacardium occidentale)," *J. Am. Oil Chemists' Soc.*, vol. 66, no. 4, pp. 553–557, 1989.

[17] S. K. G. Srivastava and V. Meharwade, *Cashew Handbook 2014 – Global Perspective*. 2014.

[18] T. T. F. Saeed, H. Bader, B. Niaz, M. A. Fzaal, A. Din, and H. A. R. Suleria, "Cashew nut allergy: Immune health challenge," *Trends in Food Science & Technology*, vol. 86, pp. 209–216, 2019.

[19] B. Gonçalves et al., "Composition of nuts and their potential health benefits—an overview," *Foods*, vol. 12, p. 942, 2023. DOI: 10.3390/foods12050942.

[20] R. Singh, P. Karthikeyan, and R. Anand, "Sc3t: A low-cost transformer-enhanced deep learning architecture for real-time cashew kernel grading on edge devices," *Computers and Electronics in Agriculture*, vol. 215, p. 108 274, 2024. DOI: 10.1016/j.compag.2024.108274.

[21] A. Mishra and S. Awasthi, "Hybrid deep learning and svm model for grading of cashew kernels using resnet-50 features," *Journal of Food Engineering*, vol. 349, p. 111 395, 2024. DOI: 10.1016/j.jfoodeng.2024.111395.

[22] Y. Li, H. Zhou, and J. Wang, "Lightweight shape-based svm classifier for whole and split cashew kernel detection," *Journal of Imaging*, vol. 9, no. 11, p. 198, 2023. DOI: 10.3390/jimaging9110198.

[23] S. Bhattacharya, S. Roy, and P. Das, "Detection of defective cashew kernels using glcm and color features with machine learning techniques," *International Journal of Food Properties*, vol. 27, no. 1, pp. 154–172, 2024. DOI: 10.1080/10942912.2024.2345120.

[24] V. Sharma and A. Ghosh, "Fuzzy logic-based grading system for cashew kernels using dimensional features," *International Journal of Agricultural and Biological Engineering*, vol. 11, no. 5, pp. 122–129, 2018. DOI: 10.25165/j.ijabe.20181105.4350.