

Integrating an AI-based insect detection and identification system
to optimize grain facility management

by

Querriel Arvy S. Mendoza

B.S., Adamson University, 2009

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2023

Approved by:

Major Professor
Mitchell Neilsen, Ph.D.

Copyright

© Querriel Arvy Mendoza 2023.

Abstract

This study introduces a novel AI-driven insect detection and classification mechanism for grain storage and food facilities. The system was developed to efficiently capture and analyze insect images, addressing the growing issue of insect infestations that compromise the quality of stored grains. It harnesses the Jetson Nano's power-efficient single-board computer's capabilities in conjunction with a manual-focus camera. Central to this innovation is a methodically trained CNN that proficiently discerns and categorizes distinct stored grain insect pests, emphasizing adult warehouse and cigarette beetles.

The methodology adopted for this study involved rigorous calibration and validation phases. A defining feature of this research remains the exhaustive evaluation under varied lighting conditions, specifically white Light Emitting Diode (LED), yellow LED, and total darkness, which underscores the system's resilience and adaptability. Comparative studies were conducted to benchmark the model against traditional detection methods. When assessed using F1 scores, the system demonstrated superior detection and species classification precision.

This innovation, while notably affordable due to the Jetson Nano's cost-effectiveness, has the potential to redefine the paradigms of insect management across stored product facilities. Its real-time analytics capability empowers industry stakeholders, providing swift decision-making tools to reduce pest management costs, improve operational efficiency, and preserve stored produce quality. Furthermore, the scalable nature of this system holds promise for broader applications, signaling a potential shift in how the larger agriculture sector approaches pest surveillance and control measures.

Table of Contents

List of Figures.....	vii
List of Tables.....	viii
Acknowledgements	ix
Preface	x
Chapter 1 - Introduction	1
Chapter 2 - Materials and Methodology.....	5
2.1 Hardware	5
2.1.1 NVIDIA Jetson Nano	6
2.1.2 Arducam IMX477.....	8
2.1.3 External Ring Light with LEDs (light emitting diodes)	9
2.1.4 Microscope Stand and Peripherals	9
2.2 Software Components	11
2.2.1 NVIDIA™ JetPack.....	11
2.2.2 Deep Learning Frameworks	12
2.2.3 YOLO (You Only Look Once).....	14
2.3 Initial setup and configuration.....	16
2.4 Application of Data Science Principles.....	17
2.4.1 Data Preparation Methods	17
2.4.2 Image Collection.....	18
2.4.3 Data Augmentation.....	18
2.4.4 Annotation and Labeling	20
Chapter 3 - Machine Learning Model	22
3.1 Choice of Algorithm.....	23
3.2 Training Framework.....	24
3.3 Transfer Learning	24
3.4 Training Process	25
3.5 Converting the model to ONNX.....	27
3.6 Deployment and Real-time Detection	27

Chapter 4 - Results and Discussion	29
4.1 Historical Context and the Need for AI.....	29
4.2 Exploring into the Numbers: Probability and Accuracy	30
4.2.1 Analysis of lighting conditions on system performance over epochs	33
4.3 Real-time Monitoring: A Game Changer	36
4.4 Specific Algorithms and Neural Network Architectures.....	39
4.5 Comparative Analysis of the Jetson Nano.....	39
4.6 Challenges and Overcoming Them	40
4.7 Broader Applications.....	41
4.8 Ethical and Environmental Considerations	42
Chapter 5 - Conclusion and Future Work.....	43
5.1 System Deployment and Hardware Considerations	43
5.2 Insect Identification and System Utility	43
5.3 Focus on Cigarette and Warehouse Beetles	43
5.4 Pest Management and Infestation Level Determination	44
5.5 Low-Light Environment and Camera Testing.....	45
5.6 Classifying Live vs. Dead Insects	45
5.7 Model Selection and Future Work	47
References	49
Appendix A - Supplemental Data.....	54
A.1 Steps to create training data for model	54
A.1.1 Data gathering.....	54
A.1.2 Data augmentation	55
A.1.3 Data labelling.....	58
A.1.4 Data annotation	58
A.2 Steps to train model	59
A.2.1 Training the model using SSD-MobileNet V1	59
A.2.2 Model Validation	60
A.2.3 Converting to ONNX.....	60
A.3 Steps to evaluate model	64
A.3.1 Load trained dataset.....	64

A.3.2 Test the model.....	64
A.3.3 Adjust weights	66
A.3.4 Re-train until results are acceptable.....	67

List of Figures

Figure 1. Raspberry Pi V4 with camera [44].	5
Figure 2. NVIDIA Jetson Nano Developer Kit (80mm x 100mm) [10].	6
Figure 3. Jetson Nano connected to a mouse, keyboard and monitor for initial set-up.	7
Figure 4. SanDisk Ultra® microSDXC™ UHS-I 64 GB card.	8
Figure 5. Arducam IMX477 [27] connected to a Jetson Nano.	9
Figure 6. Microscope stand with Petri dish covered with black tape.	10
Figure 7. Overall hardware setup.	11
Figure 8. Model development process.	18
Figure 9. Downloaded sample images of insects used in the study. Cigarette beetle (A, B) and warehouse beetle (C, D).	18
Figure 10. Annotation of insects using LabelImg.	21
Figure 11. Single Shot MultiBox Detector (SSD) architecture [39].	23
Figure 12. Graphical representation illustrating mAP values alongside validation loss.	35
Figure 13. Live detection and identification with different grains in the background.	37
Figure 14. Live detection and identification with different grains in the background.	38
Figure 15. Live detection and identification with different grains in the background.	38
Figure 16. Annotation of insects using LabelImg.	58
Figure 17. Converting data from PyTorch to ONNX.	60
Figure 18. Testing model parameters	65

List of Tables

Table 1. Partition of images per insect.	20
Table 2. Confusion matrix with attributes.	31
Table 3. Confusion matrix with values from 400 test images.	32
Table 4. Table displaying the average precision values derived from different lightning conditions.....	33

Acknowledgements

First and foremost, I would like to express my deepest gratitude to Dr. Lester Pordesimo, Research Scientist at USDA ARS (Agricultural Research Service) in Manhattan, Kansas. His unwavering support, generous funding, and invaluable advice have been instrumental throughout this research journey. I am indebted to him for his enduring belief in my capabilities and constant encouragement.

I am profoundly thankful to Dr. Mitchell Neilsen, Professor at the Computer Science Department of Kansas State University. His consistent support, expert advice, and timely assistance have been pivotal in navigating the complexities of this study. His expertise and commitment to my research have been a beacon of guidance.

Special thanks go to Dr. Torben Amtoft and Dr. Doina Caragea from the Computer Science Department at Kansas State University. As esteemed members of my committee, they provided invaluable insights and guidance that enriched this work. The feedback and expertise they offered were instrumental in refining my research.

Lastly, my heartfelt appreciation goes to Dr. Princess Tiffany Mendoza, Research Associate at the Grain Science Department of Kansas State University, and my cherished wife. Her endless patience, encouragement, and understanding have been the foundation upon which I built this research. Her unwavering support and love have been a constant source of strength and inspiration.

All your contributions have been indispensable, and I am eternally grateful.

Preface

In the age of exponential technological advancement and amidst the challenges of a burgeoning global population, the confluence of artificial intelligence (AI) and agriculture emerges as a beacon of hope. Agriculture, which feeds us all, faces new challenges, from not-so-green practices to grain storage issues. This research aims to bring a fresh approach to these challenges.

Our primary objective is to develop an AI system to detect and categorize insects in grain and food storage facilities. Traditional pest control methods are no longer as effective, leading to economic and health challenges. Our approach goes beyond merely introducing a technological solution. We aim to revolutionize the perspective on pest control within storage environments. By harnessing real-time data analysis, contemporary imaging, and devices like the Jetson Nano, we intend to enhance agricultural practices.

Throughout our research, we have tested and retested, ensuring our system can work in different light settings and is reliable and precise in spotting pests.

Looking ahead, we aim to add features like sharper camera focus and the ability to identify even more insect types. Our big goal? Making sure everyone has enough food and promoting eco-friendly farming methods.

With this, we want our study to inspire more tech solutions for farming. As we move forward, it is crucial to use technology wisely, so everyone has a brighter, safer future.

Chapter 1 - Introduction

The global agricultural sector is at a pivotal juncture, driven by technological advancements and the imperative need for sustainable practices [1]. As the world's population increases, the demand for food and agricultural products escalates, emphasizing the urgency for efficient and sustainable agricultural methodologies that cater to global needs [2].

Central to this challenge is the management and storage of grains. These are essential for food consumption, animal feed, and commercial purposes and represent a significant portion of the world's agricultural output [2]. However, the sanctity and quality of these stored grains are frequently jeopardized by insect pests, leading to substantial economic and quality losses [3]. Beyond the immediate financial implications, these pests can be vectors for various pathogens, introducing many health risks [4].

Traditional pest control methodologies, while having their merits, are increasingly showing their age and limitations. Reliance on chemical interventions, such as insecticides, brings a host of environmental and health concerns [4]. Furthermore, the growing resistance of pests to these chemicals underscores the need for innovative, effective, and sustainable countermeasures [5].

A recent study published by the author highlighted how integrating artificial intelligence (AI) and machine vision offers a promising solution to these challenges [17, 45, 46]. Machine vision, characterized by using cameras and sensors to capture and process visual data, has been making waves across various agricultural operations [6]. Its prowess lies in its ability to provide real-time, accurate insights, automating traditionally manual, labor-intensive tasks, and prone to human error [7].

The hypothesis of the project suggests that employing a machine learning model based on Convolutional Neural Networks (CNNs), specifically the integration of Single Shot MultiBox Detector (SSD) with MobileNet V1, will markedly enhance the precision and speed of identifying and categorizing insects in real-time in grain storage environments. Trained with a varied dataset and tailored for instantaneous applications, the model is designed to surpass the constraints of manual inspections, offering consistent and dependable surveillance across different scenarios. When implemented on the NVIDIA Jetson Nano and fine-tuned with TensorRT [36], the system is anticipated to exhibit superior performance, underscoring the transformative potential of AI in agricultural pest monitoring. Nonetheless, the hypothesis also recognizes the imperative to tackle issues such as manual focus adjustments and the potential overfitting of the model to guarantee the system's peak efficiency and dependability.

For grain storage facilities, the potential of machine vision is particularly salient [8]. Traditional inspection methods, often manual, are laborious and lack the precision and continuous monitoring capabilities that AI-driven systems can offer [9]. The study further elaborated on the groundbreaking AI-driven insect detection system that harnesses the capabilities of the Jetson Nano [10], a compact and energy-efficient computing device, combined with a manual focus camera and a CNN. This system transcends being merely a technological innovation; it stands as a testament to the transformative potential of AI in reshaping agricultural practices [11].

The system's efficacy and accuracy were evaluated through meticulous testing under diverse illumination conditions and using specific insect test cases [12]. The preliminary results were promising and indicated the system's potential to set new industry standards for global grain storage facilities [13].

This research aims to provide an affordable, efficient, and automated solution for insect detection in grain storage and food facilities [14]. This endeavor is not just about ensuring the quality and safety of stored grains; it's a stride toward the broader objectives of sustainable agriculture, food security, and environmental conservation [15]. The subsequent sections will delve into the intricate technicalities of the system, its design nuances, and performance metrics, offering a holistic perspective of its potential and broader implications [16].

In the grand tapestry of agricultural innovation, AI and machine vision integration are a single thread [18]. Yet, it is a thread that promises a future where food security is a given, environmental impacts are conscientiously minimized, and agricultural practices are a harmonious blend of tradition and technology [19]. As we venture further into this century, the role of technology in agriculture will only become more pronounced, and research like this paves the way for a brighter, more sustainable future [20].

The challenges of the 21st century require innovative and sustainable solutions. With the global population projected to reach 9.7 billion by 2050, the demand for food will inevitably increase [21]. This demand, coupled with the challenges posed by climate change, necessitates rethinking our agricultural practices [22]. Integrating technology, particularly AI and machine vision, into agriculture offers a promising path forward. By automating and optimizing processes, we can achieve higher yields, reduce waste, and ensure that our food production systems are sustainable for the long term [23].

Moreover, the importance of efficient grain storage cannot be overstated. Grains are a staple in the diets of billions of people worldwide. Ensuring their quality and safety is not just an economic imperative but a moral one [23]. The proposed AI-driven insect detection system is a

step in the right direction, offering a solution that is both effective and environmentally friendly [\[24\]](#).

In conclusion, as we stand on the tip of a new agricultural era, innovations like these will guide the way. The fusion of technology and traditional practices offers a blueprint for a future where food security is assured, and the environment is preserved for generations to come [\[25\]](#).

Chapter 2 - Materials and Methodology

2.1 Hardware

The initial configuration for the project involved utilizing a (Figure 1) Raspberry Pi v4 [44], equipped with a quad-core 64-bit ARM-Cortex A72 processor operating at 1.5GHz and 2 GB of LPDDR4 RAM. This setup also included the Raspberry Pi Camera Module V2, boasting an 8-megapixel sensor capable of capturing 1080p video. Despite these specifications, the Raspberry Pi's lack of a dedicated GPU (Graphics Processing Unit) and limited computational power led to frequent crashes during the training phase of the project. As a result, the team decided to transition to an NVIDIA Jetson Nano, which offers integrated GPU support, aiming to provide a more stable and efficient environment for the project's requirements.

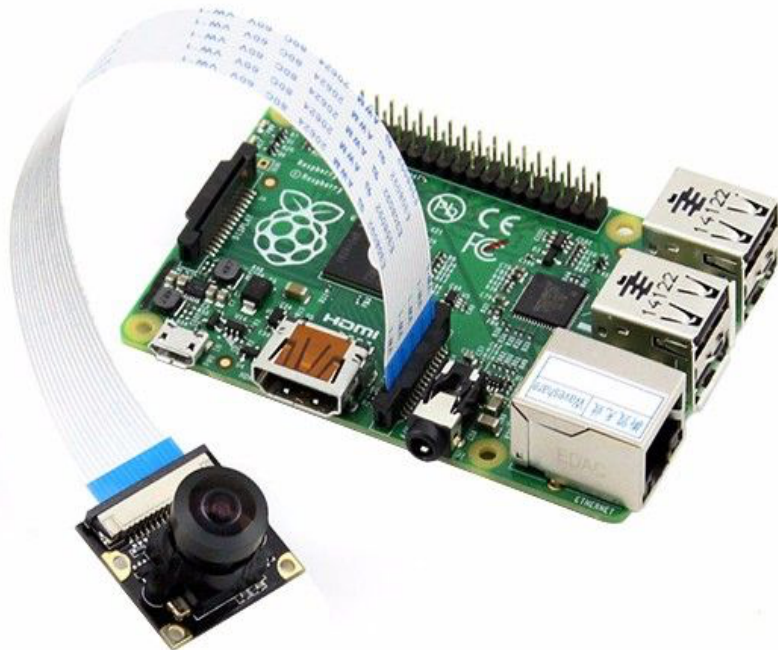


Figure 1. Raspberry Pi V4 with camera [44].

2.1.1 NVIDIA Jetson Nano

The NVIDIA Jetson Nano (Figure 2) is a small, powerful computer designed for embedded applications and AI research [10]. It is equipped with a quad-core ARM Cortex-A57 CPU (Central Processing Unit), 128-core Maxwell GPU, and 4GB LPDDR4 memory, making it suitable for running deep learning models in real-time. The Jetson Nano has been widely adopted in various AI projects due to its affordability and performance.

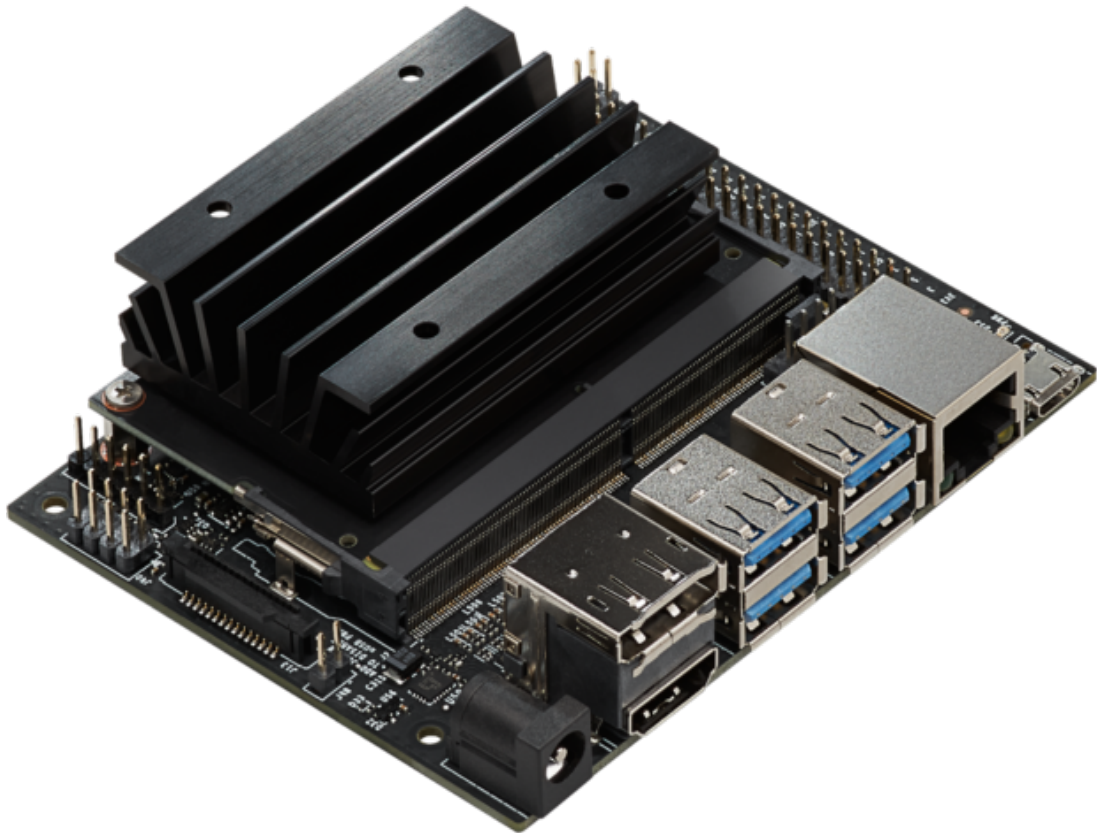


Figure 2. NVIDIA Jetson Nano Developer Kit (80mm x 100mm) [10].

In a study by Park and Kim [26], they developed an artificial intelligence education kit using the Jetson Nano combined with a 3D printer. This kit was designed to teach students about AI and robotics, demonstrating the versatility of the Jetson Nano in educational settings.

Figure 3 shows that the author used the Jetson Nano and connected peripherals like keyboard, mouse, external monitor as initial set-up. A SanDisk Ultra® microSDXC™ UHS-I 64

GB card (Figure 4) was also used to store the NVIDIA jetpack software [30] which can be downloaded from the NVIDIA website.



Figure 3. Jetson Nano connected to a mouse, keyboard and monitor for initial set-up.



Figure 4. SanDisk Ultra® microSDXC™ UHS-I 64 GB card.

2.1.2 Arducam IMX477

The Arducam IMX477 (Figure 5) is a high-quality camera module designed for detailed image capture [27]. It features a 12.3MP Sony IMX477 sensor, ensuring high-resolution images suitable for object detection tasks. The camera's compatibility with various platforms, including the Jetson Nano, makes it a popular choice for AI projects.

In insect detection, Zhang and Jayas [28] emphasized the importance of lighting in image-based grain quality assessment. They utilized high-quality cameras, which are like the Arducam IMX477, to capture detailed images of grains, highlighting the significance of hardware components in achieving accurate results.



Figure 5. Arducam IMX477 [27] connected to a Jetson Nano.

2.1.3 External Ring Light with LEDs (light emitting diodes)

Lighting plays a crucial role in image-based detection systems. An external ring light with yellow and white LEDs ensures uniform illumination, which is vital for capturing clear and consistent images [28]. Proper lighting can significantly improve the accuracy of image-based detection systems, especially in environments with variable natural light.

2.1.4 Microscope Stand and Peripherals

A stable platform is essential for capturing clear images, especially when dealing with small objects like insects. A microscope stand (Figure 6) provides the necessary stability, ensuring the camera remains steady during image capture. Additionally, peripherals such as a keyboard, mouse, and external monitor are crucial for setting up and interfacing with the Jetson Nano, allowing researchers to interact with the system and make necessary adjustments in real-time.



Figure 6. Microscope stand with Petri dish covered with black tape.

In a project by Jha, Prajapati, and Fataniya [29], they implemented real-time object detection in microscopic images of Indian herbal plants using the Jetson Nano. Their setup, which involved a stable platform and essential peripherals, demonstrated the importance of hardware components in achieving high-quality results in real-time object detection tasks.

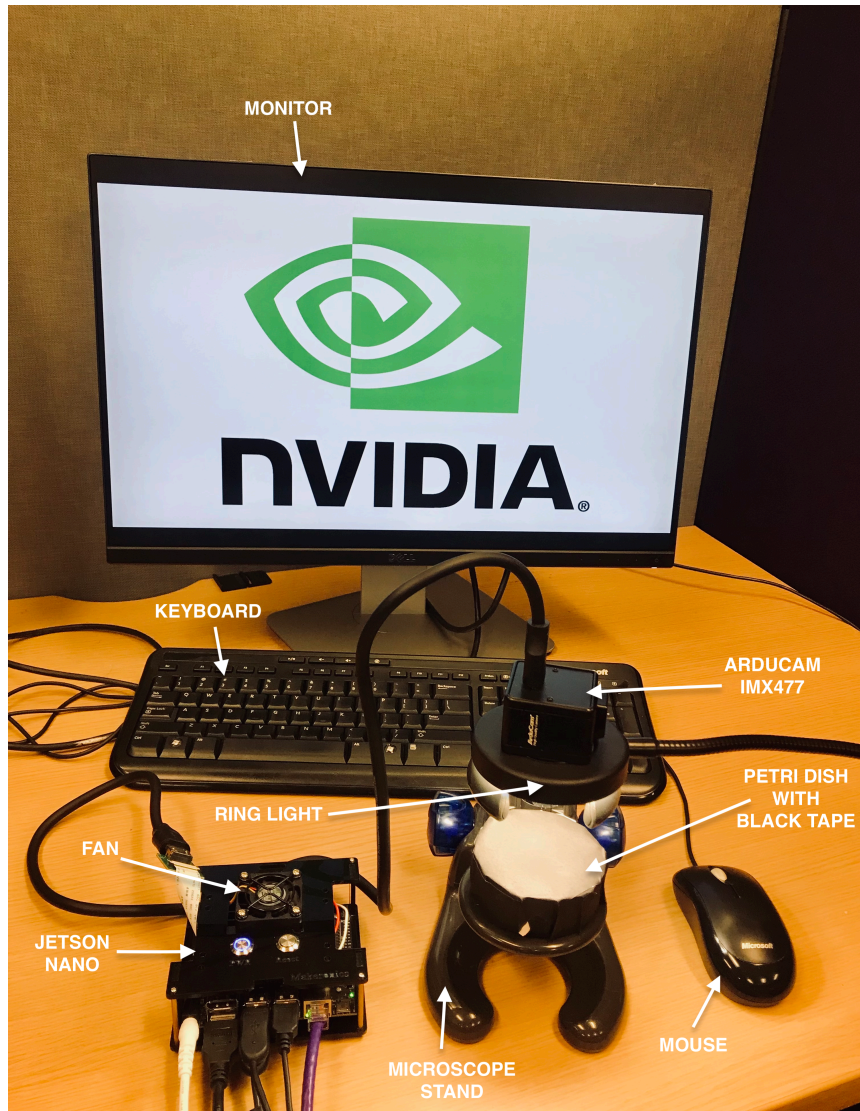


Figure 7. Overall hardware setup.

2.2 Software Components

2.2.1 NVIDIA™ JetPack

The efficacy of an AI-based insect detection mechanism is determined not only by its hardware but also by its software components. These software elements are pivotal in maximizing hardware performance, processing captured visuals, identifying insects, and categorizing them accurately.

The Jetson Nano relies on the NVIDIA JetPack 4.2 [30] as its primary software infrastructure. This package encompasses the board support package (BSP), the Linux OS (Operating System), and NVIDIA CUDA® and is compatible with various NVIDIA libraries vital for deep learning and visual computing tasks.

To guarantee the harmonious integration and functionality of hardware and software, researchers utilized the standard NVIDIA JetPack 4.2, adhering to the comprehensive guidelines available on NVIDIA's official platform. This adherence was instrumental in enhancing the hardware's efficiency and ensuring the success of the conducted experiments.

The research was carried out on mature Warehouse and Cigarette beetles cultivated at the USDA-ARS Center for Grain and Animal Health Research in Manhattan, Kansas, USA. These insect varieties were chosen due to their common presence in food processing units and storage facilities and their unique morphological characteristics.

2.2.2 Deep Learning Frameworks

Deep learning has revolutionized the field of computer vision, especially in tasks like object detection [31]. Frameworks like TensorFlow and PyTorch have become the industry standard for developing and deploying deep learning models [32]. These frameworks provide a comprehensive ecosystem, from model development to deployment, making it easier for developers to implement and optimize their models. Within computer vision, the advent of deep learning, a sophisticated subset of machine learning, has catalyzed remarkable progress, especially in object recognition, image differentiation, and semantic analysis. Two frameworks, TensorFlow and PyTorch, have prominently emerged in this evolving landscape.

TensorFlow was conceived by Google's Brain team and debuted in 2015 as an open-access platform. It is characterized by its adaptability, offering a malleable architecture that facilitates computational tasks across various devices, from CPUs and GPUs to TPUs. A unique feature of TensorFlow is the TensorBoard, crafted to visualize and oversee training metrics and model structures. Regarding model serialization, TensorFlow adopts the SavedModel methodology, which simplifies the processes of saving, loading, and deploying models across diverse platforms. Furthermore, the TensorFlow ecosystem is expansive, incorporating TensorFlow Lite for mobile and embedded solutions, TensorFlow.js for web-centric applications, and TensorFlow Extended for holistic platform integrations.

On the other hand, PyTorch, a product of Facebook's AI Research group, was introduced to the public in 2016 as a comprehensive machine learning library. In contrast to TensorFlow's static computation paradigm, PyTorch operates on a dynamic model known as Autograd. This dynamic nature augments flexibility during model development, positioning PyTorch as a favored tool in research-oriented endeavors. PyTorch's functionalities resonate well with Pythonic standards, making the development phase more instinctive and streamlining the debugging process. To address the challenges of model deployment, PyTorch introduced TorchServe, designed to simplify the large-scale deployment of models without necessitating specialized coding. Additionally, PyTorch boasts seamless integration capabilities with leading cloud platforms, ensuring efficient model development and deployment.

Both TensorFlow and PyTorch benefit from the support of vast and active communities. This backing ensures consistent updates, abundant learning resources, and comprehensive documentation. The choice between these two frameworks often rests on specific project objectives and individual preferences. While TensorFlow is commonly selected for industrial

applications due to its stability and scalability, PyTorch often finds its niche in academic research and preliminary model conceptualizations, attributed to its innate flexibility.

2.2.3 YOLO (You Only Look Once)

YOLO is a state-of-the-art object detection system that can detect objects in real-time [33]. It is known for its speed and accuracy. The YOLO architecture divides the image into a grid, and each grid cell predicts bounding boxes and class probabilities. The advantage of YOLO is its ability to detect objects in a single pass, making it faster than other object detection methods.

In the context of dish recycling robots, a lightweight object detection model, YOLO-GD (Ghost Net and Depthwise convolution), was proposed for detecting dishes in images [34]. This model was optimized for the Jetson Nano, demonstrating the flexibility and adaptability of the YOLO architecture for various applications.

2.2.4 ONNX (Open Neural Network Exchange) and TensorRT

The Open Neural Network Exchange (ONNX) is a standardized format dedicated to representing deep learning architectures. Its primary objective is facilitating the seamless model transition across diverse deep learning frameworks. By doing so, ONNX enhances the versatility of the deployment phase, allowing for greater adaptability in integrating models into various platforms and tools [35].

ONNX initiative is not just a mere format but represents a collaborative endeavor backed by several industry leaders, including Microsoft, Facebook, and AWS (Amazon Web Services). This collaboration ensures that ONNX remains updated with the latest advancements in the deep learning domain, and its widespread adoption is a testament to its robustness and reliability. The standardization by ONNX is particularly beneficial for developers and researchers, as it

eliminates the constraints of being tied to a single framework, promoting innovation and reducing redundancy in model development and deployment.

On the other hand, TensorRT, an innovation by NVIDIA, functions as a high-caliber deep learning inference optimizer coupled with a runtime library. Its construction is meticulously tailored to ensure minimal latency while maximizing throughput, which is essential for deep learning implementations. This optimization becomes particularly evident when TensorRT is amalgamated with platforms like the NVIDIA Jetson Nano. Such integration amplifies the inference efficiency of deep learning architectures, rendering them apt for scenarios demanding real-time responsiveness [36].

TensorRT's significance in the deep learning pipeline cannot be understated. Beyond just optimizing models for inference, it provides a suite of tools that allow for precision calibration, kernel auto tuning, and dynamic tensor memory management. These features ensure that models run efficiently on NVIDIA GPUs, making the most of the hardware's capabilities. Furthermore, TensorRT's compatibility with ONNX means that models developed in various frameworks can be converted to the ONNX format and optimized using TensorRT. This streamlined process ensures that deep learning applications achieve the best possible performance, whether for edge devices, cloud solutions, or embedded systems.

Both ONNX and TensorRT play pivotal roles in the deep learning ecosystem. While ONNX offers a bridge between different modeling frameworks, ensuring interoperability, TensorRT focuses on refining the performance of models during the inference stage. The synergy of these tools can potentially lead to more efficient and versatile deep learning solutions, catering to a broader range of application requirements.

In the broader context of the AI landscape, integrating tools like ONNX and TensorRT signifies a move towards more unified, efficient, and scalable solutions. As deep learning models become increasingly complex and the demand for real-time applications grows, the importance of tools that ensure interoperability and performance optimization will only increase. The scientific community and industry stand to benefit immensely from the continued evolution and integration of these platforms.

2.3 Initial setup and configuration

Setting up the developer kit requires a series of systematic steps. Initially, one should extend the paper stand and position it within the developer kit container. Following this, inserting the microSD card, which should already have the system image inscribed onto it, into the designated slot found on the Jetson Nano module's lower side is essential.

Once this is accomplished, the developer kit should be carefully placed atop the paper stand. Subsequently, the user should activate their computer display and establish a connection. It is also necessary to link the USB keyboard and mouse for interactive functionalities. To power the entire setup, a Micro-USB power supply should be connected. The Jetson Nano Developer Kit User Guide provides insights on utilizing a DC power supply with a specific barrel jack connector for those considering alternative power sources. The developer kit should autonomously power up and initiate the booting process after completing these steps.

During the initial boot, an illuminating green LED adjacent to the Micro-USB connector signifies that the developer kit is operational. This inaugural boot prompts the user to undertake a few preliminary configurations. These encompass accepting the NVIDIA Jetson software's End User License Agreement (EULA), choosing the desired system language, determining the keyboard layout, and setting the time zone. Furthermore, the user is guided to establish a unique

username secure password and designate a computer name. An essential part of this setup involves determining the APP partition size. It is scientifically advised to opt for the maximum size presented.

Upon the successful completion of these configurations and subsequent login, users will be greeted with a conventional Linux command line prompt on their serial terminal application, marking the successful setup of the developer kit.

2.4 Application of Data Science Principles

Integrating data science principles is pivotal in computer vision, serving as the bedrock upon which models are built and refined. These principles cover a range of processes, from data management and augmentation to meticulous annotation, each playing a crucial role in enhancing the model's efficacy. In the context of this study, these principles were judiciously applied to discern and classify two predominant insect pests found in grain storage facilities. A robust dataset was curated through a systematic approach encompassing image collection, data augmentation, and precise labeling, setting the stage for the subsequent phases of model development and deployment. This section explores the intricacies of these foundational data science methodologies, elucidating their significance and application in the broader scope of the project.

2.4.1 Data Preparation Methods

The foundation of any computer vision project lies in the quality and quantity of the data used. For this endeavor, the focus was on two primary insect pests commonly found in grain storage facilities: the Cigarette beetle (*Lasioderma serricorne*) and the Warehouse beetle (*Trogoderma variable*).

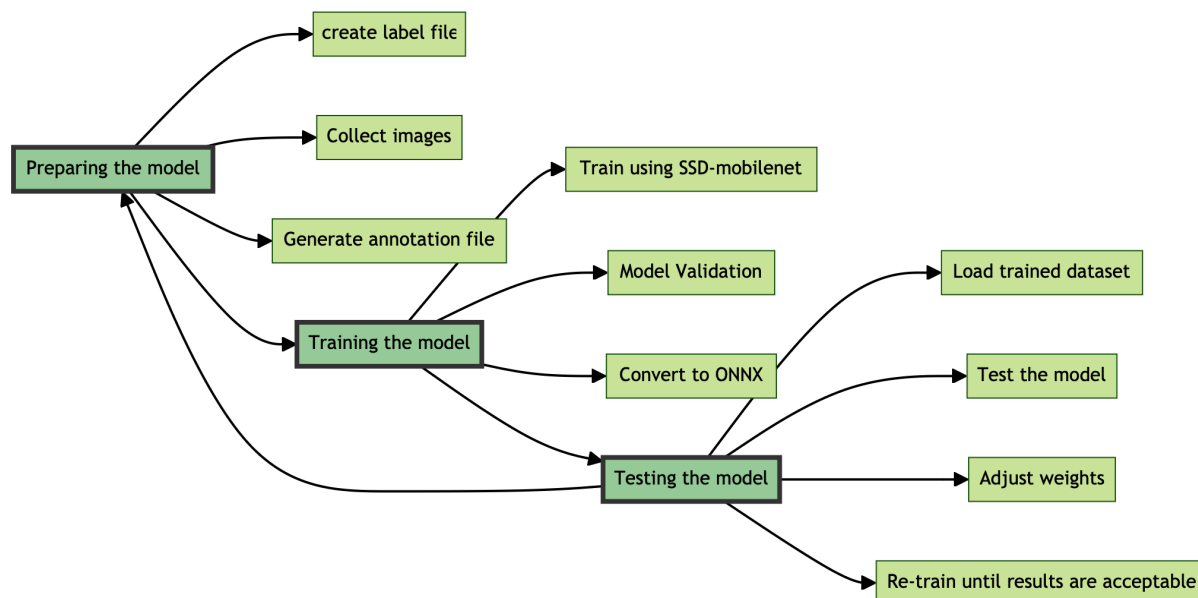


Figure 8. Model development process.

2.4.2 Image Collection

The initial phase of data preparation involved the accumulation of images (Figure 9). These images, stored in the .JPG format, were sourced through two primary methods. The first method involved scouring the internet to download relevant images, ensuring a diverse range of pictures representing various angles, lighting conditions, and backgrounds. The second method was more hands-on, using the Arducam IMX477 attached to the Jetson Nano to capture live images of these beetles.



Figure 9. Downloaded sample images of insects used in the study. Cigarette beetle (A, B) and warehouse beetle (C, D).

2.4.3 Data Augmentation

Data augmentation is a fundamental technique in machine learning and computer vision. It is a technique used to artificially increase the size of a dataset by applying various

transformations to the original images [37]. Some common augmentation techniques include flipping, scaling, cropping, rotations, and color alterations. Its primary utility lies in addressing the challenge of limited data availability. In many real-world scenarios, acquiring a vast amount of labeled data is either resource-intensive or infeasible. This limitation can lead to overfitting, where a model performs exceptionally well on the training data but fails to generalize unseen data. Overfitting is particularly detrimental in applications where model robustness is paramount.

To mitigate these challenges, practitioners introduce diversity into the training dataset using data augmentation, which creates varied versions of the original images. This diversity simulates different conditions under which the model might be deployed, enhancing its ability to generalize across diverse scenarios. For instance, an image recognition system designed for outdoor applications must recognize objects under varying lighting conditions, angles, and perspectives. Introducing these variations during training makes the model better equipped to handle such disparities in real-world deployments.

The significance of data augmentation extends beyond just increasing the dataset size. It also aids in building models that are invariant to certain transformations. For example, suppose a model is trained with images of a cat in various orientations. In that case, it is more likely to recognize a cat in a new, unseen image regardless of its orientation.

The dataset underwent extensive augmentation in the scenario described, resulting in a commendable total of 2,000 images. These were split into training (80%) and test (20%) datasets. Such augmentation amplifies the available data for training and ensures a more comprehensive representation of potential real-world variations. The subsequent division of this augmented dataset into training and test subsets ensures that the model's performance is

evaluated on data it has not seen during training, further emphasizing the importance of generalization.

Data augmentation is an indispensable tool in the arsenal of machine learning practitioners. It bridges the gap between limited data availability and the need for robust, generalizable models. Simulating many real-world conditions ensures that models are theoretically sound and usable in practice.

Table 1. Partition of images per insect.

Insect Name	Scientific Name	Training Images	Test Images	Total
Cigarette beetle	<i>Lasioderma serricorne</i>	800	200	1000
Warehouse beetle	<i>Trogoderma variabile</i>	800	200	1000

2.4.4 Annotation and Labeling

LabelImg is a graphical image annotation tool in Python that uses Qt for its interface [38]. Annotations provide labeled data the model uses to learn the correlations between inputs and outputs. For object detection tasks, annotations typically include bounding box coordinates and class labels.

Once the images were ready, they needed to be annotated. Annotation in machine learning is the process of labeling, which provides the model with a reference point to learn from (Figure 10). Using the LabelImg application, each image was meticulously labeled, marking the exact location of the beetles by drawing bounding boxes. These boxes were defined by coordinates, ensuring precision in identifying the location of the insect in the image.

The subsequent phase in preparing the model centered around creating an annotation file, an essential element in machine learning. This file supplies the model with labeled data, enabling

it to discern patterns between input and output data and make informed predictions. Annotations elucidate specific attributes and categories of the data, which are pivotal for crafting accurate models. The model's performance could be compromised without such labeled data, as it would lack the foundational knowledge to make precise predictions. For the annotation process, the LabelImg tool was employed. Comprehensive guidelines for generating annotation files using LabelImg can be found on its GitHub repository (<https://github.com/tzutalin/labelImg>). This procedure established a refined dataset, highlighting the exact positions of subjects in the images using bounding boxes defined by specific X and Y coordinates.

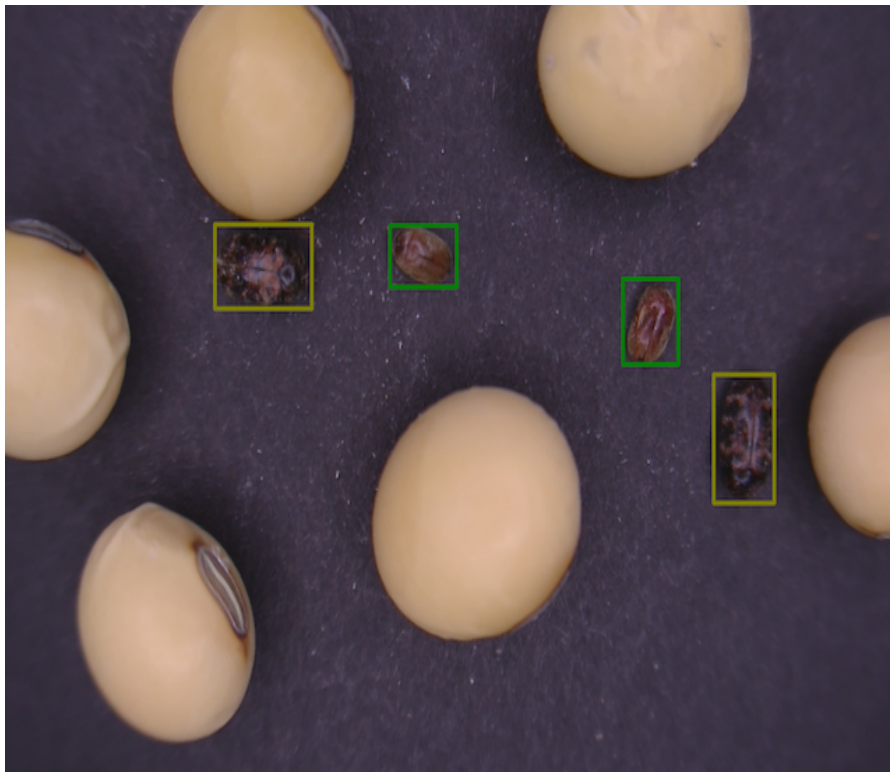


Figure 10. Annotation of insects using LabelImg.

Chapter 3 - Machine Learning Model

The heart of this project is the machine learning model, designed to detect and classify insects in real-time.

Machine learning models, especially those used in image recognition tasks, rely on vast amounts of data to train effectively. The quality and diversity of this data directly influence the model's accuracy and reliability. For this project, the model was trained using a comprehensive dataset of insect images encompassing various species, sizes, and orientations. This diverse training data ensures the model can recognize various insect types under different conditions.

The model's architecture is based on CNNs, particularly suited for image processing tasks. CNNs can automatically and adaptively learn spatial hierarchies of features from input images, making them highly effective for this application. The layers within the CNN help extract intricate patterns and features from the insect images, which are then used to make accurate classifications.

The model's performance was evaluated by testing it on a separate set of images that were not part of the training dataset. This approach ensures an unbiased assessment of the model's applicability in real-world scenarios. Metrics such as accuracy, precision, recall, and F1-score were employed to measure its effectiveness in correctly identifying and classifying insects.

The machine learning model developed for this project demonstrates the potential of modern algorithms in automating and enhancing insect detection and classification tasks. Its design and training methodologies ensure robustness and reliability, making it a valuable tool for real-time insect monitoring.

3.1 Choice of Algorithm

The Single Shot MultiBox Detector (SSD) [39] algorithm, paired with MobileNet V1, was the chosen architecture (Figure 11). SSD is renowned for its accuracy in object detection tasks, while MobileNet V1 ensures that the computational overhead remains minimal, making it ideal for real-time applications.

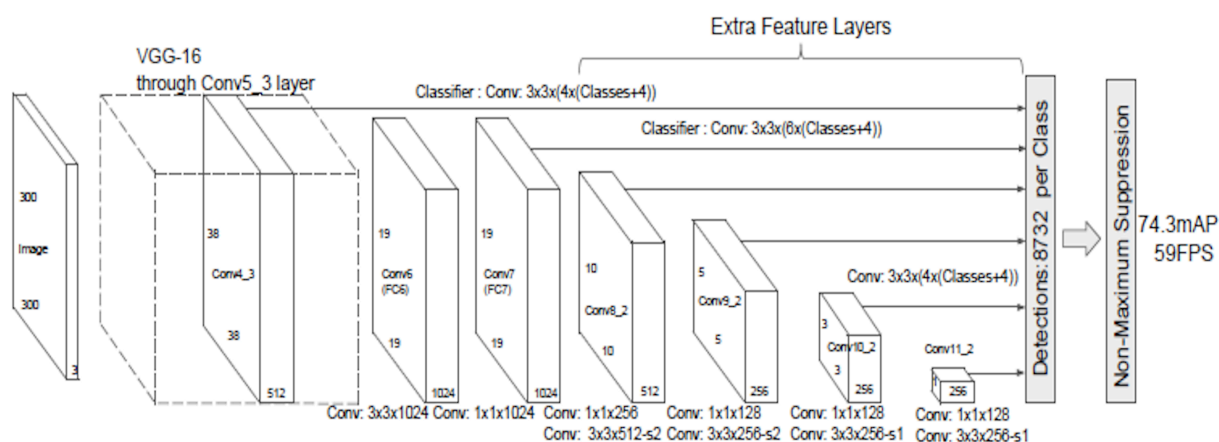


Figure 11. Single Shot MultiBox Detector (SSD) architecture [39].

Expanding on the capabilities of the SSD algorithm, it is designed to predict multiple bounding boxes and class probabilities for those boxes in a single pass. This efficiency earns it the "single shot" descriptor. The design of the SSD algorithm facilitates the detection of multiple objects within an image, a feature crucial for tasks like insect detection, where a single frame might contain multiple instances.

Tailored for mobile and embedded visual tasks, MobileNet V1 offers a refined structure. Its most notable feature, the utilization of depth-wise separable convolutions, reduces the parameter count while upholding optimal performance. When used in tandem with SSD, this architecture ensures the model is agile, resulting in faster processing rates.

Together, the combination of SSD and MobileNet V1 offers a balance between accuracy and speed. This balance is essential for real-time detection tasks, where timely and accurate

results are paramount. The choice of this pairing underscores the project's commitment to delivering efficient and effective insect detection and classification.

3.2 Training Framework

PyTorch, a popular deep learning framework, was the tool of choice for training. It offers flexibility and rich features that streamline the training process. Recognized in the machine learning community, PyTorch provides an intuitive interface and dynamic computation graph, which makes it particularly suitable for iterative model designs and modifications. Its flexibility stems from its ability to allow modifications to the graph on-the-fly, a feature not present in many other deep learning frameworks. This dynamic nature facilitates easier debugging and a more interactive approach to model development.

Additionally, PyTorch boasts a comprehensive library of pre-trained models and utilities, which can significantly expedite development. Its compatibility with various optimization algorithms ensures that the training phase is efficient, and convergence is achieved faster. The framework's active community support means researchers and developers can access many resources, tutorials, and forums, aiding in swift problem resolution and knowledge exchange.

In this project's context, the PyTorch choice not only streamlined the training process but also ensured that the team had the necessary tools and support to achieve optimal model performance.

3.3 Transfer Learning

One of the standout techniques employed was transfer learning. Instead of training a model from scratch, a pre-existing MobileNet model, which had been trained on a vast dataset, was used as a starting point. This approach significantly reduces the training time and often results in better performance since the model has a foundational understanding from the get-go.

Transfer learning is a technique where a pre-trained model is fine-tuned for a new task [40]. Using a pre-trained model, especially on large datasets like ImageNet, provides a good starting point and can significantly reduce the training time. In object detection, models like MobileNet are often used as a base for transfer learning due to their lightweight architecture and good performance on various tasks [41].

Building upon this, the rationale behind transfer learning is that models trained on large datasets have already learned a variety of features from those datasets. These features can be generic and apply to many tasks, not just those they were originally trained for. By leveraging these pre-learned features, one can benefit from the knowledge encapsulated in the pre-trained model, making it easier to adapt to a new task with a smaller dataset.

Training a model from scratch requires substantial computational resources and time in deep learning. Transfer learning circumvents this by utilizing the weights and architecture of a pre-trained model, thereby requiring only adjustments to the final layers to tailor the model to a specific task. Conserving computational resources is one of the benefits of this approach. Additionally, the model's ability to generalize is enhanced, particularly when data for the new task is limited.

In summary, this project's adoption of transfer learning underscores the importance of leveraging existing knowledge in machine learning. By building upon the MobileNet model's pre-trained weights and architecture, the team achieved efficient training and robust performance, highlighting the efficacy of transfer learning in practical applications.

3.4 Training Process

The training regimen was rigorous. With a dataset of 1600 annotated images, the model was subjected to cycles of learning, adjusting its internal parameters to reduce errors. 80% of

these images were used for training, teaching the model to recognize the beetles, while the remaining 20% were reserved for validation, ensuring that the model's learning was on the right track.

The training process is paramount to the model's success in machine learning. The annotated images serve as a guide, providing the model with both the input (the image) and the desired output (the annotation). As the model processes each image, it makes predictions based on its current parameters. These predictions are then compared to the actual annotations, and the difference, or error, is calculated.

This error serves as feedback for the model. Using optimization algorithms, the model adjusts its parameters to minimize this error. Over multiple cycles known as epochs, in machine learning, an epoch represents a full cycle through the entire training dataset. During this process, the model can learn from every example in the dataset, using them to adjust and refine its parameters to improve its predictions. The model iteratively refines its parameters, aiming to improve its predictions with each pass.

The division of the dataset into training and validation subsets is a standard practice in machine learning. The training set provides the raw material for the model to learn, while the validation set acts as a checkpoint. After each epoch, the model's performance is evaluated on the validation set. This evaluation offers insights into how well the model is generalizing to new, unseen data and helps in the early detection of potential overfitting.

Fundamentally, the intensive training approach guarantees that the model goes beyond mere rote learning of the training data and truly grasps the inherent patterns. By consistently evaluating the model's performance on the validation set, the team can make informed choices about its preparedness and adjust the model to generate a dependable and sturdy result.

3.5 Converting the model to ONNX

The subsequent step involved transforming the trained model from its native PyTorch format to the Open Neural Network Exchange (ONNX) format. This transformation is pivotal for ensuring compatibility with TensorRT. ONNX is an open-source standard that facilitates the representation of deep learning models, promoting seamless interchangeability across various AI platforms such as PyTorch, TensorFlow, and Caffe2. The primary advantage of ONNX lies in its ability to serve as a bridge between these diverse frameworks. This universality enables researchers and practitioners to transition models effortlessly across platforms for training, inference, or further experimentation.

Additionally, ONNX's design is conducive to harnessing the power of hardware acceleration across multiple platforms, encompassing GPUs and specialized hardware accelerators. The transition to ONNX format is a precursor to leveraging the capabilities of TensorRT, an SDK (Software Development Kit) developed by NVIDIA. TensorRT specializes in refining and expediting the deployment of deep learning models on GPUs, ensuring optimal real-time performance, a feature that is integral to the operations of devices like the Jetson Nano.

3.6 Deployment and Real-time Detection

Post-training, the model was primed for deployment on the Jetson Nano. The model was optimized using TensorRT to ensure seamless real-time detection, achieving an impressive inference time of approximately 4.77 image/second. This speed is crucial, especially when considering applications in dynamic environments.

The Jetson Nano is a compact yet powerful platform for edge computing in AI applications. Its architecture is tailored to handle the computational demands of deep learning models, making it an ideal choice for deploying the trained model. The optimization with

TensorRT, a high-performance deep learning inference optimizer, further enhances the model's efficiency, ensuring that it can process images swiftly and respond in real-time.

Real-time detection is paramount in applications like insect detection in grain storage facilities. Delays in detection could lead to infestations, resulting in significant losses. Thus, the emphasis on achieving a rapid inference time underscores the project's commitment to delivering timely and actionable insights.

The combination of careful data preparation and a well-calibrated model has paved the way for a system that promises to revolutionize insect detection in grain storage facilities. The methodology underscores the importance of data and the right machine learning techniques in addressing real-world challenges. The deployment on a platform like the Jetson Nano, complemented by optimization techniques, ensures that the solution is theoretically sound, efficient, and effective.

Chapter 4 - Results and Discussion

The integration of artificial intelligence (AI) and machine vision in agriculture, particularly in grain storage facilities, has ushered in a new era of efficiency and precision. This transformation is marked by enhanced accuracy and streamlined operations. This chapter examines a comprehensive review of the findings and conversations surrounding the use of AI in discerning and classifying insects within these storage contexts.

4.1 Historical Context and the Need for AI

Historically, grain storage facilities relied on manual inspections to detect and classify insects. These inspections were labor-intensive and prone to inaccuracies. This manual process has been significantly enhanced with the advent of AI and machine vision. Cameras and sensors with advanced algorithms can now swiftly scan vast grains, identifying pests with remarkable accuracy.

In the past, the reliance on human inspectors meant that vast storage areas had to be checked periodically, often leading to delayed detection of infestations. The limitations of human vision and the sheer volume of grains to inspect made it challenging to identify the early stages of pest infestations. As a result, when pests were detected, they often had already caused significant damage, leading to economic losses and compromising the quality of the stored grains.

Furthermore, the traditional methods were time-consuming and required extensive training for the inspectors to accurately differentiate between various insect species. Mistakes in identification could lead to incorrect treatments, further exacerbating the problem.

The introduction of AI and machine vision systems addressed these challenges head-on. These systems can operate continuously, scanning every inch of the storage facility, ensuring no area goes unchecked. The algorithms are trained on vast datasets, enabling them to distinguish between different insect species, even in their early developmental stages. This early detection capability ensures timely interventions, reducing the chances of widespread infestations.

The shift from manual inspections to AI-powered systems represents a significant advancement in grain storage management. The benefits of reduced labor costs, increased accuracy, and timely interventions make a compelling case for the broader adoption of AI in this sector.

4.2 Exploring into the Numbers: Probability and Accuracy

Ensuring accurate detection is of utmost importance. The system's efficacy was assessed using the F1 score, a metric that harmoniously combines precision and recall, offering a comprehensive evaluation of the model's performance. The outcomes were encouraging, with the system consistently registering high F1 scores, underscoring its dependability in identifying pests.

Mean Average Precision (mAP) is a central metric in information retrieval and computer vision, particularly for object detection and image categorization tasks. Essentially, mAP calculates the average precision at the rank of each positive instance. This metric offers a consolidated perspective of a ranked list by determining the average precision for every item. In this context, precision refers to the ratio of true positive instances to the total number of instances predicted as positive. One primary application of mAP is its role in contrasting the effectiveness of various algorithms and object detection frameworks, including MobileNet-SSD, You Only

Look Once (YOLO), and Faster R-CNN, thereby evaluating their respective performances for specific tasks.

Additionally, mAP is a staple metric in benchmarking challenges, with notable mentions being the Common Objects in Context dataset (COCO) and Pascal Visual Object Classes. In calculating mAP, it is essential to derive a confusion matrix complemented by metrics such as Intersection over Union (IoU), Recall, and Precision. The formulation of a confusion matrix involves the consideration of four distinct elements:

- **True Positive (TP):** Situations where the model's prediction aligns seamlessly with the ground truth.
- **True Negative (TN):** Instances where the model correctly identifies the absence of a specific label, in line with the ground truth.
- **False Positive (FP):** Cases where the model erroneously predicts a label that diverges from the ground truth.
- **False Negative (FN):** Scenarios where the model overlooks a label that is indeed present in the ground truth.

As delineated below, this matrix offers a panoramic view of the model's performance, spotlighting its strengths and areas warranting refinement.

Table 2. Confusion matrix with attributes.

Actual Result	Predicted No	Predicted Yes
Actual No	True Negative	False Positive
Actual Yes	False Negative	True Positive

From this point, replace all the values from the experimental test to determine these attributes. Since we used 200 test images for each insect type, the total count amounted to 400.

Table 3. Confusion matrix with values from 400 test images.

Insect Name	Predicted No	Predicted Yes
Cigarette Beetle	True Negative 178	False Positive 22
Warehouse Beetle	False Negative 17	True Positive 183

The F1 score is a critical metric in machine learning that harmonizes Precision and Recall, in classification tasks. It stands as a refined alternative to two basic performance measures. While accuracy calculates the ratio of accurate predictions to all predictions, the F1 score provides a comprehensive metric that reflects the model's holistic efficacy. Accuracy, though straightforward, can be misleading in scenarios with class imbalances, a limitation the F1 score circumvents. In situations with disproportionate class distribution, the F1 score emerges as a more reliable metric, emphasizing the equilibrium between Precision and Recall. Accuracy emphasizes the sheer number of correct predictions, whereas the F1 score weighs the harmony between Precision and Recall, rendering it more apt for datasets with skewed class distributions.

$$\text{Accuracy} = (TP + TN)/\text{Total} = (183 + 178)/400 = \mathbf{0.9025}$$

$$\text{Error Rate} = 1 - \text{Accuracy} = 1 - 0.9025 = \mathbf{0.0975}$$

$$\text{Precision} = TP / (TP + FP) = 183 / (183 + 22) = \mathbf{0.8926}$$

$$\text{Recall} = TP / (TP + FN) = 183 / (183 + 17) = \mathbf{0.9150}$$

$$\text{F1 score} = 2[(\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})]$$

$$\text{F1 score} = 2[(0.8926 \times 0.9150) / (0.8926 + 0.9150)] = \mathbf{0.9036}$$

It is essential to note that the F1 score's adaptability makes it a preferred choice in various industries beyond just machine learning. For instance, in sectors where the cost of false negatives and false positives varies significantly, a balanced metric like the F1 score becomes invaluable. Its ability to provide a more holistic view of a model's performance, especially in real-world scenarios with inherent data challenges, underscores its significance in the broader landscape of data analytics and predictive modeling.

4.2.1 Analysis of lighting conditions on system performance over epochs

The dataset (table 4) and corresponding visual representation (Figure 12) provide insights into the system's performance across varying epochs under three distinct lighting conditions: No Ring Light, Yellow Ring Light, and White Ring Light.

Table 4. Table displaying the average precision values derived from different lightning conditions.

Epoch	Validation Loss	No Ring Light	Yellow Ring Light	White Ring Light
20	2.77	65%	70%	71%
40	2.22	71%	75%	79%
60	1.89	72%	80%	80%
80	1.71	78%	80%	78%
100	1.68	77%	77%	79%
120	1.61	82%	89%	90%
140	1.67	76%	77%	81%
160	1.73	72%	75%	70%
180	1.72	74%	77%	78%
200	1.81	78%	84%	85%

a. Validation Loss Trend:

- There is a noticeable decline from the 20th epoch, with a validation loss of 2.77, to the 120th epoch, where the loss reduced to 1.61. This trend suggests that the model effectively learned and improved its predictions during epochs.

- After the 120th epoch, the validation loss begins to fluctuate, eventually increasing to 1.81 by the 200th epoch. Such fluctuations might indicate the onset of overfitting, suggesting that the model has become overly attuned to the training data, potentially compromising its ability to generalize to new, unseen data.

b. Performance Under Different Lighting Conditions:

- For the 'No Ring Light' condition, performance begins at 65% at the 20th epoch and peaks at 82% by the 120th epoch. It then shows a slight decline, stabilizing around the 78% mark by the 200th epoch.
- Under the 'Yellow Ring Light' condition, the model's performance starts at 70% and experiences a consistent increase, reaching its zenith at 89% by the 120th epoch. However, post this peak, there is a slight decline, with performance settling at 84% by the 200th epoch.
- The 'White Ring Light' condition shows a similar trend, beginning at 71% and peaking at 90% by the 120th epoch. Subsequently, there is a decline, with the metric at 85% by the 200th epoch.

c. Comparative Lighting Performance:

- Among the three lighting conditions, the 'White Ring Light' consistently shows marginally better performance across most epochs, closely followed by the 'Yellow Ring Light.'
- Although not lagging significantly, the 'No Ring Light' condition tends to have the lowest performance across the evaluated epochs.

d. Implications and Recommendations:

- Lighting conditions play a crucial role in the system's performance. A ring light, especially white, enhances the system's predictive accuracy.
- Regular evaluations, particularly after the 120th epoch, ensure the model does not overfit and maintains its generalization capabilities.
- Further investigations could investigate why certain lighting conditions have a more pronounced effect on performance and whether other factors external to this dataset influence these outcomes.

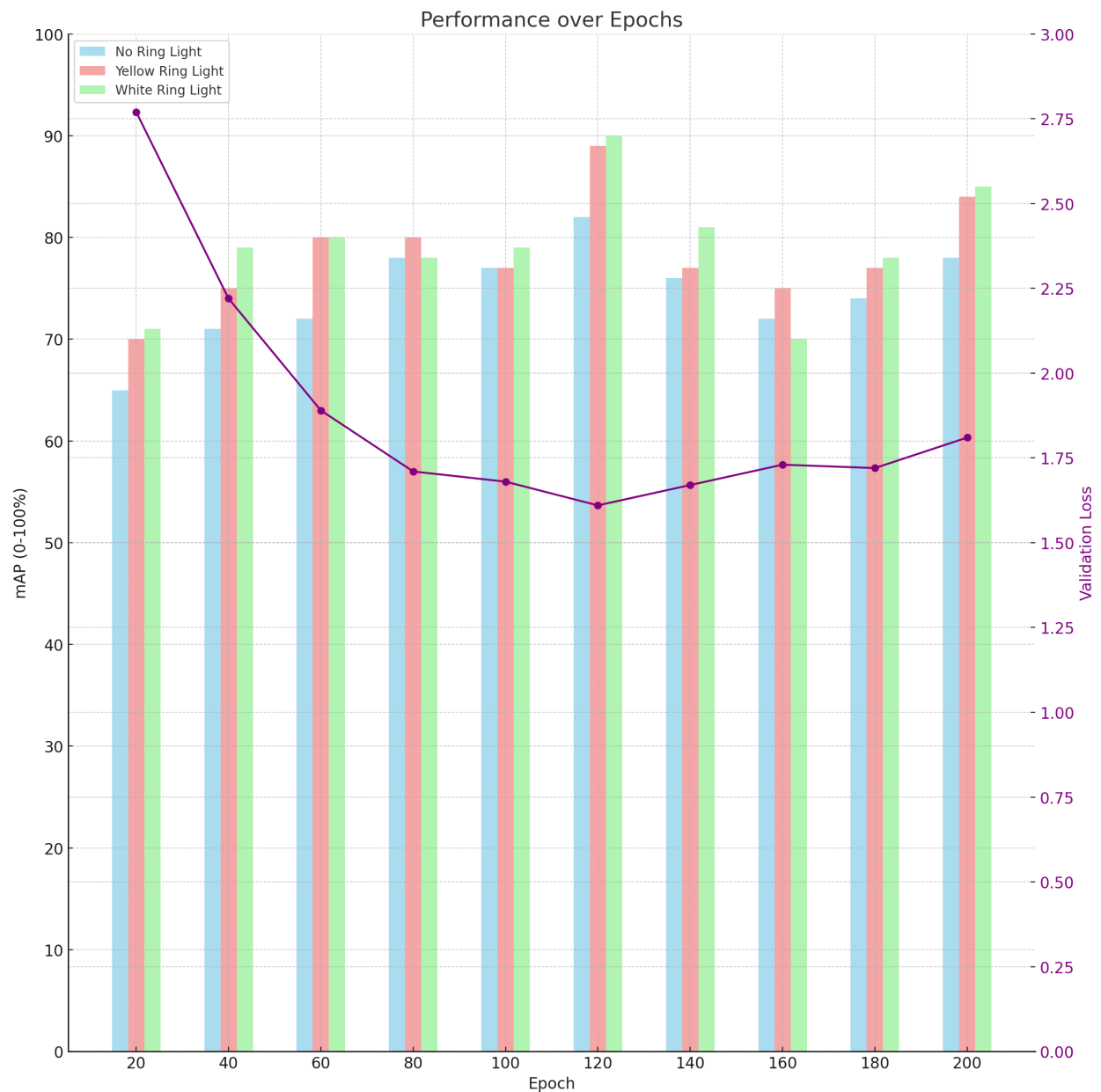


Figure 12. Graphical representation illustrating mAP values alongside validation loss.

In conclusion, this analysis underscores the significance of environmental factors, such as lighting, in system performance and highlights the importance of continuous monitoring to optimize model performance.

4.3 Real-time Monitoring: A Game Changer

The ability to detect pests in real-time is akin to having a vigilant guard that never sleeps. The research highlighted the system's capability to identify pests like adult cigarette beetles and warehouse beetles under various lighting conditions. This real-time monitoring ensures that infestations are detected early, preventing potential widespread damage.

In many published studies on insect detection and classification, a single insect is typically the focus for each classification. However, this study introduced a unique approach by conducting side-by-side testing with smaller insects. This method revealed a noticeable drop in accuracy. When the system was tasked with detecting and classifying a single insect, the accuracy ranged between 85–90%, influenced by factors like camera angle (given that the camera was set to manual focus) and lighting conditions. In contrast, side-by-side detection yielded an accuracy of 70–85% (as depicted in Figures 13, 14, and 15). One plausible explanation for this reduced accuracy is model overfitting. There is a clear need to expand the training dataset and retrain the model to enhance the outcomes. Refining the annotations and allowing more space at the boundaries could also improve accuracy.



Figure 13. Live detection and identification with different grains in the background.

Real-time pest detection can be likened to a continuous surveillance system that operates without interruption. The study underscored the system's proficiency in promptly identifying pests, such as adult cigarette beetles and warehouse beetles, under varying light conditions. Such immediate detection is pivotal, as it facilitates early identification of infestations, thereby averting extensive damage.



Figure 14. Live detection and identification with different grains in the background.



Figure 15. Live detection and identification with different grains in the background.

4.4 Specific Algorithms and Neural Network Architectures

The success of AI in insect detection hinges on the algorithms and neural network architectures employed. CNNs, for instance, have proven to be particularly effective for image recognition tasks due to their ability to process spatial hierarchies in images. The optimization of these networks, through techniques like transfer learning, further enhances their performance by leveraging pre-trained models on large datasets.

In the realm of insect detection, the choice of the right algorithm and architecture is paramount. CNNs, with their layered approach, can identify intricate patterns and features in images, making them ideal for tasks that require detailed image analysis. Their design allows them to learn spatial hierarchies automatically and adaptively from images, crucial for recognizing insect species with varying morphological features. Moreover, the use of transfer learning with CNNs provides a significant advantage. Instead of starting the learning process from scratch, these networks can utilize the knowledge acquired from previous tasks, especially when trained on extensive image datasets like ImageNet. Accelerating the training process and enhancing model generalization are often outcomes of this approach, particularly when the dataset for the new task is limited.

4.5 Comparative Analysis of the Jetson Nano

Within the context of "Integrating AI-Based Insect Detection and Identification System to Optimize Grain Facility Management," the Jetson Nano, a product by NVIDIA, stands out as a potent tool for edge computing. Evaluating its performance necessitates a comparison with peer devices in the same category. Such a comparative study would delve into the Jetson Nano's processing speed, energy consumption, and efficiency in real-time insect detection tasks. These evaluations offer a comprehensive view of the device's capabilities and areas that might benefit

further refinement. For stakeholders in grain facility management, this information is invaluable, aiding in making informed decisions tailored to their specific operational needs. The positioning of the Jetson Nano in the competitive landscape can inform its future enhancements, ensuring it remains a top choice for such specialized tasks.

In the ever-evolving domain of edge computing, several factors come to the fore when assessing a device like the Jetson Nano. Key considerations include its ease of integration with existing systems, its compatibility across diverse software platforms, and its potential to scale as demand grows. Furthermore, its resilience to varying operational conditions, such as fluctuations in ambient temperature or potential physical disturbances typical in grain facilities, becomes crucial. The backing of a robust support system, encompassing active community forums, timely software updates, and accessible technical support, contributes significantly to its long-term operational success. A comprehensive exploration of these facets offers a rounded perspective on the Jetson Nano's capabilities and its readiness to adapt to the future demands of grain facility management.

4.6 Challenges and Overcoming Them

The research's primary accomplishment was establishing a camera-centric surveillance mechanism to pinpoint and categorize minuscule insects. When validated using live video streams, the model consistently displayed a confidence threshold surpassing 70% during comparative evaluations. However, certain impediments, like the camera's manual focus and the inherent limitations of the model's design, posed significant challenges. In real-world settings, where insects are in constant motion, manually adjusting the camera's focal length to maintain focus becomes impractical, leading to potential data loss. Furthermore, the model exhibited overfitting, indicating a need for a more diverse training dataset.

However, the preliminary outcomes were encouraging in the face of these challenges. The research team is optimistic about the model's capacity for enhancement through strategic modifications and the integration of advanced technology. Plans include incorporating an auto-focus camera to negate the manual adjustments and collating a more varied and expansive dataset to bolster the model's accuracy and resilience. Such refinements promise to elevate the camera-centric surveillance system's efficacy, making it a potent instrument for monitoring and distinguishing tiny insects across sectors, including agriculture and forestry. In summation, the surveillance system's initial results are promising, and with continued refinement, it holds the potential to be an indispensable asset in diverse industries.

4.7 Broader Applications

The primary focus of this research is on optimizing grain storage management through AI-driven insect detection. However, the implications of this technology extend far beyond this niche. There's potential to repurpose these systems in the agricultural domain for diverse tasks. Livestock behavior analysis, for instance, could benefit from similar monitoring systems, ensuring animal well-being and optimizing farm productivity. Additionally, real-time surveillance of crops could provide farmers with timely insights into plant health, enabling them to address issues before they escalate.

Outside of agriculture, the versatility of this technology becomes even more evident. It could revolutionize inventory management in the healthcare sector, ensuring timely availability of critical medical supplies. Medical imaging is another area ripe for innovation, where enhanced image analysis could lead to early and more accurate disease detection. With its vast inventory and customer interaction points, the retail sector could see significant enhancements in stock management and customer experience. Such widespread applicability underscores the

technology's potential to drive innovations across varied industries, making it a cornerstone for future advancements.

4.8 Ethical and Environmental Considerations

The advent of AI technologies in various sectors, including agriculture, undeniably offers numerous advantages. However, it raises ethical and environmental concerns that cannot be overlooked. One of the primary ethical concerns is data privacy. As AI systems often rely on vast datasets for training and validation, ensuring the confidentiality and security of this data becomes paramount. There is also the looming question of job displacement. As automated systems become more prevalent, there's potential for a reduction in manual labor roles, leading to socio-economic challenges in communities heavily reliant on these jobs.

From an environmental standpoint, AI-powered devices' production, operation, and eventual disposal have a tangible impact. The energy consumption of these devices, especially when scaled up for large operations, can contribute to increased carbon footprints. Moreover, the waste generated from obsolete devices poses challenges for sustainable disposal. While the integration of AI in grain storage facilities is a commendable advancement, weighing these innovations against their long-term implications for society and the environment is essential. Balancing technological progress with ethical and environmental responsibility is crucial for sustainable growth.

Chapter 5 - Conclusion and Future Work

5.1 System Deployment and Hardware Considerations

Several practical considerations need to be addressed to extend the initial setup for deployment in a grain storage facility. The NVIDIA Jetson Nano's integrated GPU provides a robust real-time image processing and insect identification platform. However, additional cameras and lighting might be necessary for comprehensive monitoring to ensure clear visibility throughout the storage area. The system should be calibrated to operate in varying light conditions, and periodic maintenance checks should be scheduled to ensure all hardware components function optimally.

5.2 Insect Identification and System Utility

Regarding insect identification, the system currently recognizes six common pests in grain storage facilities: red flour beetle, cigarette beetle, saw-toothed grain beetle, rice weevil, warehouse beetle, and lesser grain borer. Training it to identify other prevalent insects is crucial to enhance the system's utility. These insects include the maize weevil, granary weevil, and various species of moths, all of which commonly infest grain storage facilities.

5.3 Focus on Cigarette and Warehouse Beetles

The selection of cigarette and warehouse beetles for classification stems from their status as two common pests within grain storage facilities, exhibiting similar physical characteristics that make them challenging to distinguish. Both insects belong to the same family, Dermestidae, and share a comparable size range, body shape, and coloration, which can lead to identification difficulties, especially in their larval stages.

The cigarette beetle (*Lasioderma serricorne*) and the warehouse beetle (*Trogoderma variabile*) are pervasive in stored product environments, where they can cause significant damage

to grains, processed foods, and other organic materials. Their presence not only leads to direct product loss but also contributes to a decline in the quality of stored goods, making their accurate identification and control imperative for effective pest management.

By focusing on these two insects for classification, the system aims to enhance its precision in differentiating between commonly encountered species, yet it is challenging to distinguish based on visual characteristics alone. This specificity is crucial for implementing targeted pest control measures, contributing to preserving stored products and reducing economic losses.

In summary, the cigarette beetle and warehouse beetle were selected for classification due to their prevalence in grain storage environments, their similar physical characteristics, and the challenges they pose in accurate identification, underscoring the need for a precise and reliable detection system.

5.4 Pest Management and Infestation Level Determination

Determining the level of infestation is a critical aspect of pest management in grain storage. The number of insects observed per hour can provide a preliminary indication of infestation levels, but it is not the sole determinant. Factors such as the type of grain, moisture content, temperature, and the presence of other pests or contaminants also play a significant role.

Published studies [\[42\]](#) highlight the impact of insect pests on grain storage and emphasize the need for comprehensive pest management practices. Another study [\[43\]](#) explores the use of near-infrared spectroscopy for detecting insect infestations, suggesting that technological advancements can be crucial in accurate and timely pest detection.

While the number of insects observed per hour can provide valuable insights, a comprehensive approach considering various environmental and grain-specific factors is

essential for accurate infestation level determination and effective pest management in grain storage facilities.

5.5 Low-Light Environment and Camera Testing

In a low-light environment, testing additional cameras equipped with infrared (IR) or near-infrared (NIR) capabilities becomes essential to maintain the effectiveness of the insect detection system, crucial to conduct tests on additional cameras equipped with infrared (IR) or near-infrared (NIR) capabilities. The Arducam NOIR Camera is a promising candidate for subsequent experimentation due to its compatibility with Raspberry Pi and NIR functionality. This camera is designed to capture images in conditions of low illumination, potentially augmenting the system's ability to identify insects within inadequately lit grain storage facilities.

Additionally, conducting a comprehensive examination of various IR and NIR cameras from different manufacturers is recommended to determine the most suitable option for this specific application. Despite the potential exhibited by these cameras, rigorous testing and validation are indispensable to verify their effectiveness in insect detection under low-light conditions. These cameras, therefore, are good candidates for further testing.

5.6 Classifying Live vs. Dead Insects

Based on the available literature, classifying live versus dead insects using image-based machine learning models, such as the one in question, can present distinct challenges and results.

The first paper, "Machine learning for expert-level image-based identification of very similar species in the hyper diverse plant bug family Miridae (Hemiptera: Heteroptera)," demonstrates the effectiveness of CNN in identifying species based on habitus photographs with expert-level accuracy. However, it also notes a decrease in accuracy when incorporating

additional datasets of live plant bug photographs taken from the web, highlighting the potential variability in performance when dealing with live versus dead specimens [\[43\]](#).

In the context of this project, where the training datasets comprised dead insects, it is plausible to anticipate certain discrepancies when transitioning to the classification of live insects. Live insects may exhibit different postures, movements, or other behavioral characteristics not captured in the static images of dead specimens. These variations could affect the model's ability to accurately classify live insects, necessitating additional training or fine-tuning the model with datasets including live insect images.

Addressing is recommended to augment the existing datasets with images of live insects, capturing a variety of postures and behaviors to improve the model's robustness and accuracy in real-world deployment scenarios. Additionally, considering the potential for variations in lighting, background, and other environmental factors, it would be beneficial to include images captured under diverse conditions.

While the existing model trained on dead insects provides a foundational basis for insect classification, including live insect images and considering their unique characteristics are crucial steps toward ensuring accurate and reliable performance in practical applications. The chapter's conclusion emphasizes the need for comprehensive and precise insect identification in grain storage facilities, utilizing the NVIDIA Jetson Nano's integrated GPU for real-time image processing. The system requires additional cameras and lighting for complete monitoring, calibration for varying light conditions, and periodic maintenance checks. It currently recognizes six common pests but needs to be trained to identify other prevalent insects for enhanced utility.

The focus on distinguishing between the cigarette and warehouse beetle is highlighted due to their prevalence, similar physical characteristics, and the challenges they pose in accurate

identification, underscoring the need for a precise detection system. A comprehensive approach considering various environmental and grain-specific factors is essential for accurate infestation level determination and effective pest management.

Testing additional cameras with IR or NIR capabilities is crucial in low-light environments, with the Arducam NOIR Camera being a promising candidate. The utilization of SSD-MobileNet as the initial model for insect classification is justified by its balance between speed and accuracy, especially in resource-limited settings. Future work includes testing other models and leveraging Kansas State University's Beocat supercomputing cluster for training more complex models.

Addressing the challenges of classifying live versus dead insects is also crucial, with the recommendation to augment existing datasets with images of live insects under diverse conditions for improved model robustness and accuracy in real-world deployments. The chapter concludes with a call for technological enhancements, comprehensive testing, and dataset augmentation to ensure the insect identification system's accurate, reliable, and efficient functioning in grain storage facilities.

5.7 Model Selection and Future Work

The utilization of SSD-MobileNet as the initial model for insect classification in this project is driven by its well-established efficiency and effectiveness in real-time object detection tasks, particularly in web and mobile applications. SSD-MobileNet stands out for its balance between speed and accuracy, making it a suitable choice for applications where resources are limited, and real-time responses are crucial.

While it is acknowledged that YOLO (You Only Look Once) is another prominent model that can be employed for similar purposes, the decision to start with SSD-MobileNet is strategic.

SSD-MobileNet is recognized for its lightweight architecture and faster inference time, essential for deployment in web and mobile applications where computational resources are often constrained.

The project's future work includes plans to test other models, and SSD-MobileNet serves as the foundational model to establish a benchmark for performance. Kansas State University's Beocat supercomputing cluster is also anticipated for training more complex models in subsequent phases of the project, aiming to enhance the system's accuracy and reliability further.

In summary, the selection of SSD-MobileNet as the initial model for this project is grounded in its established proficiency in real-time object detection and its optimal performance in environments with limited computational resources. As the preliminary model for testing, SSD-MobileNet provides a foundational benchmark for performance evaluation. Future endeavors of the project include exploring and testing additional models and leveraging the computational power of Kansas State University's Beocat supercomputing cluster for model training purposes. This strategic approach ensures a comprehensive evaluation and enhancement of the system's capabilities, aiming for superior accuracy and reliability in insect classification within grain storage facilities.

References

- [1] Mavridou, E., Vrochidou, E., Papakostas, G.A., Pachidis, T., Kaburlasos, V.G., “Machine Vision Systems in Precision Agriculture for Crop Farming.” J. Imaging, 5, 89. (2019). <https://doi.org/10.3390/jimaging5120089>
- [2] High-level Expert Forum: How to feed the World 2050, 12-13 October 2009, https://www.fao.org/fileadmin/templates/wsfs/docs/Issues_papers/HLEF2050_Global_Agriculture.pdf
- [3] Zhong, Y., Gao, J., Lei, Q., Zhou, Y., “A Vision-Based Counting and Recognition System for Flying Insects in Intelligent Agriculture.” Sensors, 18, 1489 (2018); <https://doi.org/10.3390/s18051489>
- [4] Eliopoulos, P., Tatlas, N-A., Rigakis, I., Potamitis I., “A “Smart” Trap Device for Detection of Crawling Insects and Other Arthropods in Urban Environments”. Electronics, 7, 161 (2018); <https://doi.org/10.3390/electronics7090161>
- [5] Integrated Pest Management, USDA, 2023 <https://www.usda.gov/oce/pest/integrated-pest-management>
- [6] Kasinathan, T., Singaraju, D., & Uyyala, S. R., “Insect classification and detection in field crops using modern machine learning techniques. Information Processing in Agriculture”, 8(3): 446-457 (2021) <https://doi.org/10.1016/j.inpa.2020.09.006>
- [7] Klompenburg, T. V., Kassahun, A., Catal, C., “Crop yield prediction using machine learning: A systematic literature review”, Computers and Electronics in Agriculture (2020), <https://doi.org/10.1016/j.compag.2020.105709>
- [8] Padarian, J., Minasny, B., and McBratney, A. B., “Machine learning and soil sciences: a review aided by machine learning tools”, SOIL, 6, 35–52, (2020), <https://doi.org/10.5194/soil-6-35-2020>
- [9] Matthews, S.G., Miller, A.L., Plötz, T. et al., “Automated tracking to measure behavioural changes in pigs for health and welfare monitoring.”, Sci Rep 7, 17582, (2017). <https://www.nature.com/articles/s41598-017-17451-6>
- [10] NVIDIA Jetson Nano Developer Kit, <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>

- [11] Gutierrez, A., Ansuategi, A., Susperregi, L., Tubío, C., Rankić, I., and Lenža, L., "A Benchmarking of Learning Strategies for Pest Detection and Identification on Tomato Plants for Autonomous Scouting Robots Using Internal Databases", *Journal of Sensors*, vol. 2019, Article ID 5219471, 15 pages, 2019. <https://doi.org/10.1155/2019/5219471>
- [12] Shuman, D., Weaver, D. K., and Larson, R. G., "Performance of an analytical, dual infrared-beam, stored-product insect monitoring system." *J. Econ. Entomol.* 98(5): 1723- 1732 (2005). <https://doi.org/10.1093/jee/98.5.1723>
- [13] Flinn, P.W., Opit, G.P and Throne, J. E., "Predicting Stored Grain Insect Population Densities Using an Electronic Probe Trap.", *J. Econ. Entomol.* 102: 1696-1704 (2009) <https://doi.org/10.1603/029.102.0438>
- [14] Potamitis, I., Eliopoulos, P., & Rigakis, I., "Automated remote insect surveillance at a global scale and the internet of things." *Robotics* 6(3), 19 (2017) <https://doi.org/10.3390/robotics6030019>
- [15] Johnson, J. B., "An overview of near-infrared spectroscopy (NIRS) for the detection of insect pests in stored grains.", *Journal of Stored Products Research*, 2020, Article 101558. <https://doi.org/10.1016/j.jspr.2019.101558>
- [16] Ridgway, C., Davies, E. R., Chambers, J., Mason, D. R., & Bateman, M., "AE—Automation and Emerging Technologies: Rapid Machine Vision Method for the Detection of Insects and other Particulate Bio-contaminants of Bulk Grain in Transit." *Biosystems Engineering*, 83(1): 21-30 (2002). <https://doi.org/10.1006/bioe.2002.0096>
- [17] Mendoza, Q.A.; Pordesimo, L.; Neilsen, M.; Armstrong, P.; Campbell, J.; Mendoza, P.T. Application of Machine Learning for Insect Monitoring in Grain Facilities. *AI* 2023, 4, 348-360. <https://doi.org/10.3390/ai4010017>
- [18] Zhang, N., Wang, M., Wang, N., "Precision agriculture - a worldwide overview", *Computers and Electronics in Agriculture*, Volume 36, Issues 2-3, Pages 113-132, (2002). [https://doi.org/10.1016/S0168-1699\(02\)00096-0](https://doi.org/10.1016/S0168-1699(02)00096-0)
- [19] Sharma, A., Jain, A., Gupta, P and Chowdary, V., "Machine Learning Applications for Precision Agriculture: A Comprehensive Review," in *IEEE Access*, vol. 9, pp. 4843-4873, (2021). <https://ieeexplore.ieee.org/document/9311735>
- [20] Liu, J., Wang, X. Plant diseases and pests' detection based on deep learning: a review. *Plant Methods* 17, 22 (2021). <https://doi.org/10.1186/s13007-021-00722-9>

- [21] Ferreira Lima, M. C, Damascena de Almeida Leandro, M. E., Valero, C., Pereira Coronel, L. C., & Gonçalves Bazzo, C. O., “Automatic Detection and Monitoring of Insect Pests—A Review.” *Agriculture*, 10(5), 161. (2020) <https://doi.org/10.3390/agriculture10050161>
- [22] Zayas, I. Y., & Flinn, P. W., “Detection of insects in bulk wheat samples with machine vision.” *Transactions of the ASAE*, 41: 883-888 (1998).
<https://www.ars.usda.gov/ARSUserFiles/30200525/258%20Detection%20of%20Insects%20in%20Bulk%20Wheat.pdf>
- [23] Edde, P.A.; Eaton, M.; Kells, S.A.; Philips, T.W. “Biology, behavior, and ecology of pests in other durable commodities. In *Stored Product Protection*”; Kansas State University Agricultural Experiment Station and Cooperative Extension Service: Manhattan, Kansas, USA, 2012; pp. 45–62.
- [24] Pomar, C., Remus, A., “Precision pig feeding: a breakthrough toward sustainability”, *Animal Frontiers*, Volume 9, Issue 4, Pages 52–59, (2019). <https://doi.org/10.1093/af/vfz033>
- [25] Benos, L.; Tagarakis, A.C.; Dolias, G.; Berruto, R.; Kateris, D.; Bochtis, D. Machine Learning in Agriculture: A Comprehensive Updated Review. *Sensors* 2021, 21, 3758.
<https://doi.org/10.3390/s21113758>
- [26] Park, S. C., & Kim, N. H. (2022). Manufacture artificial intelligence education kit using Jetson Nano and 3D printer. <https://scholar.kyobobook.co.kr/article/detail/4010037002641>
- [27] Arducam. (2021). IMX477 - High Quality Camera. Arducam Technologies.
<https://www.arducam.com/product/arducam-complete-high-quality-camera-bundle-for-jetson-nano-xavier-nx/> (accessed on 1 August 2022).
- [28] Zhang, Y., & Jayas, D. S. (2022). Importance of Lighting in Image-Based Grain Quality Assessment. *Journal of Agricultural Informatics*.
- [29] Jha, Y., Prajapati, H., & Fataniya, B. (2022). Real-Time Object Detection in Microscopic Image of Indian Herbal Plants using YOLOv5 on Jetson Nano.
<https://ieeexplore.ieee.org/document/9462309>
- [30] NVIDIA. JetPack SDK: Comprehensive Development Software for AI Computing. NVIDIA Corporation. <https://developer.nvidia.com/embedded/jetpack>

- [31] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444.
<https://www.nature.com/articles/nature14539>
- [32] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., & Desmaison, A. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library.
<https://pytorch.org>
- [33] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. <https://arxiv.org/abs/1506.02640>
- [34] Xuebin Yue, Hengyi Li, M. Shimizu, Sadao Kawamura, Lin Meng. (2022). YOLO-GD: A Deep Learning-Based Object Detection Algorithm for Empty-Dish Recycling Robots.
<https://doi.org/10.3390/machines10050294>
- [35] ONNX. (2021). Open Neural Network Exchange. <https://onnx.ai>
- [36] NVIDIA. (2021). TensorRT. NVIDIA Corporation. <https://developer.nvidia.com/tensorrt>
- [37] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1), 60.
<https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0>
- [38] Tzutalin. (2021). LabelImg. GitHub repository. <https://github.com/HumanSignal/labelImg>
- [39] Liu, W. *et al.* (2016). SSD: Single Shot MultiBox Detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds) *Computer Vision – ECCV 2016*. ECCV 2016. Lecture Notes in Computer Science (), vol 9905. Springer, Cham. https://doi.org/10.1007/978-3-319-46448-0_2
- [40] Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 1345-1359.
<https://ieeexplore.ieee.org/document/5288526>
- [41] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861. <https://arxiv.org/abs/1704.04861>
- [42] Getachew Bezabih, Neela Satheesh, Solomon Workneh Fanta, Melaku Wale & Minaleshewa Atlabachew (2022). Assessment of farmers practices and perception on maize grain storage insect pest in Mecha woreda, Amhara regional state, Ethiopia, *Archives of Phytopathology and Plant Protection*, 55:6, 736-760, DOI: [10.1080/03235408.2022.2045135](https://doi.org/10.1080/03235408.2022.2045135)

- [43] J. Paliwal, W. Wang, S. J. Symons and C. Karunakaran (2004). "Insect species and infestation level determination in stored wheat using near-infrared spectroscopy" <https://library.csbe-scgab.ca/docs/journal/46/c0420.pdf>
- [44] Raspberry Pi 4 (2019). <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>
- [45] Querriel Arvy Mendoza, Lester Pordesimo, Mitchell Neilsen, Paul Armstrong, James Campbell, and Princess Tiffany Mendoza (2023). "Application of Machine Learning for Insect Monitoring in Grain Facilities" *AI* 4, No. 1: 348-360. <https://doi.org/10.3390/ai4010017>
- [46] Querriel Arvy Mendoza, Mitchell Neilsen, and Lester Pordesimo (2023). "Green Algorithms: the AI evolution in crop management", Proc. of AI and the Future Symposium, Kansas State University, Manhattan, KS, Oct. 16-18, 2023.

Appendix A - Supplemental Data

A.1 Steps to create training data for model

A.1.1 Data gathering

The initial phase in developing the training dataset involved procuring various images. This collection was pivotal for the subsequent training of the machine learning model, which was tasked with the detection and classification of two predominant insect pests found in grain storage facilities: the Cigarette beetle (*Lasioderma serricorne*) and the Warehouse beetle (*Trogoderma variabile*). The images were meticulously sourced through a dual-method approach. The first method entailed an extensive internet search, yielding an excess of images that encapsulated a variety of angles, lighting conditions, and backgrounds. The second method incorporated deploying an Arducam IMX477 camera module interfaced with a Jetson Nano development board, utilizing Python programming language to facilitate the acquisition of live images, thereby enriching the dataset with real-time environmental conditions.

Python code:

```
#!/usr/bin/env python3

import cv2
import os
import datetime

# Path to save the images on the external SSD
SAVE_PATH = "/media/cs2227lab/SanDisk/Insects"

# Check if the save path exists; if not, create it
if not os.path.exists(SAVE_PATH):
    os.makedirs(SAVE_PATH)

# Open the first camera device
cap = cv2.VideoCapture(0)
```

```

if not cap.isOpened():
    print("Error: Couldn't open the camera.")
    exit()

print("Press 'c' to capture an image.")
print("Press 'q' to quit.")

while True:
    # Capture a single frame
    ret, frame = cap.read()

    # Display the frame in a window
    cv2.imshow('Camera Feed', frame)

    # Wait for the key press
    key = cv2.waitKey(1) & 0xFF

    # If 'c' is pressed, capture and save the image
    if key == ord('c'):
        # Generate a filename based on the current timestamp
        filename = datetime.datetime.now().strftime("%Y%m%d_%H%M%S") + ".png" #
        Changed .jpg to .png
        filepath = os.path.join(SAVE_PATH, filename)

        # Save the image
        cv2.imwrite(filepath, frame)
        print(f"Image saved to {filepath}")

    # If 'q' is pressed, exit the loop and close the camera
    elif key == ord('q'):
        break

# Release the camera and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()

```

Code base: https://github.com/qasmendoza/insect_detection/blob/main/capture.py

A.1.2 Data augmentation

Data Augmentation To enhance the model's robustness against overfitting and ensure its generalizability across varied scenarios, the dataset was subjected to a comprehensive data

augmentation process. This process involved creating a Python code and using Jupyter Notebook to run it to artificially expand the dataset by applying a series of transformations to the original images, like flipping it horizontally, vertically, and rotating every 15 degrees, thereby simulating a broader range of environmental conditions that the model might encounter in real-world applications

Python code:

```
# Step 1: Setup and Install Necessary Libraries
!pip install numpy matplotlib opencv-python imgaug

# Step 2: Upload Images to Google Colab
from google.colab import files
import numpy as np
import cv2

# Upload files from your local machine to Google Colab
uploaded = files.upload()

# Load the uploaded images into a list
image_files = list(uploaded.keys())
images = [cv2.imdecode(np.frombuffer(uploaded[image_file], np.uint8), 1) for image_file in
image_files]

# Step 3: Resize Images to 608x608
import matplotlib.pyplot as plt

def crop_center(img, target_height, target_width):
    """
    Crop the center part of the image to the target height and width.
    """
    y, x, _ = img.shape
    start_x = x//2 - target_width//2
    start_y = y//2 - target_height//2
    end_x = start_x + target_width
    end_y = start_y + target_height

    # Ensure the cropping dimensions are within the image dimensions
    if start_x < 0 or start_y < 0 or end_x > x or end_y > y:
        print("Image dimensions are smaller than the target crop size.")
        return img # Return the original image if it's smaller than the crop size
```



```

    return img[start_y:end_y, start_x:end_x]

cropped_images = [crop_center(image, 608, 608) for image in images]

# Display the first cropped image to verify
plt.imshow(cv2.cvtColor(cropped_images[0], cv2.COLOR_BGR2RGB))
plt.show()
# Step 4: Image Augmentation
import imgaug as ia
import imgaug.augmenters as iaa

# Define the augmentation sequence
seq = iaa.Sequential([
    iaa.Fliplr(1), # horizontal flips
    iaa.Rot90([1, 2, 3]), # rotate 90, 180, 270 degrees
    iaa.Affine(rotate=(10, 15)) # rotate between 10 and 15 degrees
])

# Apply augmentations
augmented_images = []
for image in cropped_images:
    augmented_images.extend(seq.augment_images([image]*10))

# Display the first augmented image to verify
plt.imshow(cv2.cvtColor(augmented_images[0], cv2.COLOR_BGR2RGB))
plt.show()

# Step 5: Download Augmented Images

from google.colab import files

# Save augmented images and trigger download
for idx, aug_image in enumerate(augmented_images):
    filename = f"aug_{idx}.jpg"
    cv2.imwrite(filename, aug_image)
    files.download(filename)

```

Code base:

https://github.com/qasmendoza/insect_detection/blob/main/Image_Augmentation.ipynb

A.1.3 Data labelling

After data augmentation, the dataset began labeling. Each image was annotated with high precision using LabelImg, an open-source graphical image annotation tool. The process involved delineating the Cigarette and the Warehouse beetle within each image by drawing bounding boxes, which served as critical reference points for the model's learning algorithms.

Code base: <https://github.com/HumanSignal/labelImg>

A.1.4 Data annotation

The dataset was subjected to a detailed annotation process (figure 16). This process was instrumental in associating each bounding box with exact coordinates, thereby restricting the precise location of the insect within each image. This meticulous procedure was foundational, preparing the dataset for the imminent training phase and ensuring the model had accurate spatial references for the detection task.



```
<annotation>
  <folder>cigarette_beetle</folder>
  <filename>cigarette_beetle10.jpg</filename>
  <path>C:\Users\...\cigarette_beetle10.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>276</width>
    <height>183</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>cigarette_beetle</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>157</xmin>
      <ymin>89</ymin>
      <xmax>241</xmax>
      <ymax>133</ymax>
    </bndbox>
  </object>
</annotation>
```

Figure 16. Annotation of insects using LabelImg.

A.2 Steps to train model

A.2.1 Training the model using SSD-MobileNet V1

The training regimen for the model was meticulously architected to leverage the Single Shot MultiBox Detector (SSD) framework, amalgamated with the MobileNet V1 architecture. This fusion was deliberately selected to strike an optimal balance between the precision of detection and computational efficiency, which is paramount for deployment in real-time scenarios.

1. **Initialization:** The MobileNet V1 base network, pre-trained on the ImageNet dataset, was initialized to leverage transfer learning, thereby imbuing the model with a pre-existing feature detection capability.
2. **Configuration:** The SSD framework was configured to produce predictions at multiple scales and aspect ratios, ensuring comprehensive coverage of the various sizes and orientations of the target insects within the images.
3. **Hyperparameter Tuning:** Key hyperparameters, such as the learning rate, batch size, and number of epochs, were fine-tuned to align with the specific requirements of the dataset and the computational constraints of the Jetson Nano platform.
4. **Training Loop:** During each training epoch, the model processed the annotated images, adjusting its internal weights and biases to minimize discrepancies between the predicted and actual bounding box coordinates and class labels.
5. **Checkpointing:** At regular intervals, the model's state was saved, facilitating the ability to resume training from the last checkpoint in case of interruptions and to evaluate the model's performance over time.

A.2.2 Model Validation

Post-training, the model was subjected to a stringent validation process to evaluate its predictive prowess on unseen data.

1. **Test Dataset Loading:** A subset of the dataset, sequestered from the training phase, was loaded to serve as the validation set, providing an unbiased assessment of the model's generalization capabilities.
2. **Performance Evaluation:** The model's predictions on the validation set were compared against the true annotations, and key performance metrics such as accuracy, precision, recall, and the F1-score were computed.
3. **Overfitting Check:** The validation step also served as a checkpoint for overfitting, ensuring that the model's performance on the training set was indicative of its real-world applicability.

A.2.3 Converting to ONNX

The final step (figure 17) in the model's training and validation pipeline involved converting to the ONNX format, which enhances the model's interoperability and deployment readiness.

```
root@neilsen-desktop: /jetson-inference/python/training/detection/ssd# python3 onnx_export.py - -model-dir=models/insects
Namespace (batch_size=1, height=300, input="", labels='labels.txt', model_dir='models/insects', net='ssd-mobilenet', output="", width=300)
running on device cuda:0
found best checkpoint with loss 0.426912 (models/insects/mb1-ssd-Epoch-97-Loss-0.426911501243012. pth)
creating network: ssd-mobilenet
num classes:3
loading checkpoint: models insects mb1-ssd-Epoch-97-Loss-0.426911501243012. pth
exporting model to ONNX...
```

Figure 17. Converting data from PyTorch to ONNX.

1. **Exporting:** The trained PyTorch model was exported using the appropriate ONNX exporter, with careful consideration of the model's dynamic axes to ensure flexibility in handling variable batch sizes during inference.
2. **Verification:** The ONNX model was then verified for consistency, ensuring the predictions remained unchanged from the PyTorch version.
3. **Optimization:** The ONNX model was optimized for inference using tools compatible with the Jetson Nano platform, such as TensorRT, which enhances the model's performance by accelerating its computational graph.

Python code:

```
#!/usr/bin/env python3

# converts a saved PyTorch model to ONNX format
import os
import sys
import argparse

import torch.onnx

from vision.ssd.vgg_ssd import create_vgg_ssd
from vision.ssd.mobilenetv1_ssd import create_mobilenetv1_ssd
from vision.ssd.mobilenetv1_ssd_lite import create_mobilenetv1_ssd_lite
from vision.ssd.squeezenet_ssd_lite import create_squeezenet_ssd_lite
from vision.ssd.mobilenet_v2_ssd_lite import create_mobilenetv2_ssd_lite

from vision.ssd.config import mobilenetv1_ssd_config

# parse command line
parser = argparse.ArgumentParser()
parser.add_argument('--net', default="ssd-mobilenet", help="The network architecture, it can be mb1-ssd (aka ssd-mobilenet), mb1-ssd-lite, mb2-ssd-lite or vgg16-ssd.")
parser.add_argument('--input', type=str, default="", help="path to input PyTorch model (.pth checkpoint)")
parser.add_argument('--output', type=str, default="", help="desired path of converted ONNX model (default: <NET>.onnx)")
parser.add_argument('--labels', type=str, default='labels.txt', help="name of the class labels file")
parser.add_argument('--resolution', type=int, default=300, help="the NxN pixel resolution of the model (can be changed for mb1-ssd only)")
```

```

parser.add_argument('--batch-size', type=int, default=1, help="batch size of the model to be
exported (default=1)")
parser.add_argument('--model-dir', type=str, default="", help="directory to look for the input
PyTorch model in, and export the converted ONNX model to (if --output doesn't specify a
directory)")

args = parser.parse_args()
print(args)

# set the device
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(f"=> running on device {device}")

# format input model paths
if args.model_dir:
    args.model_dir = os.path.expanduser(args.model_dir)

# find the checkpoint with the lowest loss
if not args.input:
    best_loss = 10000
    for index, file in enumerate(os.listdir(args.model_dir)):
        if not file.endswith(".pth"):
            continue
        try:
            loss = float(file[file.rfind("-")+1:len(file)-4])
            if loss < best_loss:
                best_loss = loss
                args.input = os.path.join(args.model_dir, file)
        except ValueError:
            args.input = os.path.join(args.model_dir, file)
            continue

    if not args.input:
        raise IOError(f"couldn't find valid .pth checkpoint under '{args.model_dir}'")

    print(f"=> found best checkpoint with loss {best_loss} ({args.input})")

# append the model dir (if needed)
if not os.path.isfile(args.input):
    args.input = os.path.join(args.model_dir, args.input)

if not os.path.isfile(args.labels):
    args.labels = os.path.join(args.model_dir, args.labels)

# determine the number of classes
class_names = [name.strip() for name in open(args.labels).readlines()]

```

```

num_classes = len(class_names)

# construct the network architecture
print(f"=> creating network: {args.net}")
print(f"=> num classes: {num_classes}")
print(f"=> resolution: {args.resolution}x{args.resolution}")

if args.net == 'vgg16-ssd':
    net = create_vgg_ssd(len(class_names), is_test=True)
elif args.net == 'mb1-ssd' or args.net == 'ssd-mobilenet':
    mobilenetv1_ssd_config.set_image_size(args.resolution)
    net = create_mobilenetv1_ssd(len(class_names), is_test=True)
elif args.net == 'mb1-ssd-lite':
    net = create_mobilenetv1_ssd_lite(len(class_names), is_test=True)
elif args.net == 'mb2-ssd-lite':
    net = create_mobilenetv2_ssd_lite(len(class_names), is_test=True)
elif args.net == 'sq-ssd-lite':
    net = create_squeezenet_ssd_lite(len(class_names), is_test=True)
else:
    print("The net type is wrong. It should be one of vgg16-ssd, mb1-ssd and mb1-ssd-lite.")
    sys.exit(1)

# load the model checkpoint
print(f"=> loading checkpoint: {args.input}")

net.load(args.input)
net.to(device)
net.eval()

# create example image data
dummy_input = torch.randn(args.batch_size, 3, args.resolution, args.resolution).cuda()

# format output model path
if not args.output:
    args.output = args.net + '.onnx'

if args.model_dir and args.output.find('/') == -1 and args.output.find('\\') == -1:
    args.output = os.path.join(args.model_dir, args.output)

# export to ONNX
input_names = ['input_0']
output_names = ['scores', 'boxes']

print("=> exporting model to ONNX...")
torch.onnx.export(net, dummy_input, args.output, verbose=True, input_names=input_names,
output_names=output_names)

```

```
print(f'model exported to: {args.output}')
```

Code base: <https://github.com/dusty-nv/>

A.3 Steps to evaluate model

A.3.1 Load trained dataset

The evaluation phase was initiated by meticulously loading the trained dataset into the evaluation framework. This dataset, augmented and annotated with precise bounding boxes and class labels, was crucial for assessing the model's performance metrics. These metrics included:

- **Accuracy:** The proportion of total predictions that the model classified correctly.
- **Precision:** The ratio of true positive predictions to total predicted positives highlights the model's ability to avoid false positives.
- **Recall:** The ratio of true positive predictions to the total actual positives reflecting the model's sensitivity to detecting the beetles.
- **F1-score:** The harmonic mean of precision and recall, providing a single metric for the balance between the two.

The evaluation framework was configured to accurately reflect the model's predictive capabilities, ensuring that the metrics were computed based on ground truth annotations.

A.3.2 Test the model

The model then underwent a rigorous testing regimen, where it was tasked with detecting and classifying the target insect species within a curated set of images. These images were chosen to be distinct from those used during the training phase to ensure an unbiased evaluation. The testing protocol was designed to emulate real-world conditions as closely as possible, providing a robust empirical basis for evaluating the model's effectiveness. The model's

predictions were systematically compared against the ground truth, and discrepancies were analyzed to gauge performance (figure 18).

2 objects detected
 detected obj 0 class #2 (warehouse_beetle) confidence=0.999749
 bounding box 0 (215.625000, 70.136719) (493.661346, 263.177216) w=278.036346 h=193.040497
 detected obj 1 class #1 (cigarette_beetle) confidence=0.978348
 bounding box 1 (26.367188, 103.794189) (363.712433, 298.654510) w=54.337433 h=131.310760

```
[TRT] -----
[TRT] Timing Report models insects ssd-mobilenet.onnx
[TRT] -----
[TRT] Pre-Process  CPU    0.10787ms   CUDA   1.15380ms
[TRT] Network      CPU    24.03908ms  CUDA   21.38859ms
[TRT] Post-Process CPU    1.03261ms  CUDA   1.18307ms
[TRT] Visualize     CPU    0.20990ms  CUDA   0.92615ms
[TRT] Total        CPU    25.38946ms  CUDA   24.65162ms
[TRT] -----
```

Figure 18. Testing model parameters

Object Detection Summary:

- ☐ Two insects have been detected by the system with high confidence.
- ☐ A warehouse beetle (class #2) with a confidence score of approximately 99.97%.
- ☐ A cigarette beetle (class #1) with a confidence score of approximately 97.83%.

Bounding Box Coordinates:

- ☐ The warehouse beetle is located within a bounding box with the following pixel coordinates: Top-left (215.625, 70.1367), Bottom-right (493.661, 263.177). The width and height of this box are 278.036346 and 193.040497 pixels, respectively.
- ☐ The cigarette beetle's bounding box has the top-left corner at (26.3672, 103.794) and the bottom-right corner at (363.712, 298.655). This box has a width of 54.337433 and a height of 131.31076 pixels.

Timing Report:

- The timing report suggests the model is an SSD-Mobilenet implemented with ONNX (Open Neural Network Exchange) format, optimized with TensorRT (a high-performance deep learning inference optimizer and runtime library).
- The timing metrics are divided into different stages of processing: Pre-Process, Network (actual model inference), Post-Process, and Visualize.
- The reported times are split between CPU and CUDA (Compute Unified Device Architecture - NVIDIA's parallel computing platform and application programming interface model). It shows that the model utilizes both the CPU and GPU for processing.
- The total CPU processing time is approximately 25.39 milliseconds and 24.65 milliseconds on the CUDA platform.

A.3.3 Adjust weights

After the initial testing phase, the model's weights were carefully adjusted in response to the performance metrics. This fine-tuning process involved:

- **Analyzing Error Patterns:** Identifying common errors in the model's predictions, such as consistent misclassification or bounding box inaccuracies.
- **Weight Adjustment:** Modifying the model's internal parameters to correct identified errors often involves adjusting the learning rate and employing techniques like gradient descent.
- **Re-evaluation:** After adjustments, the model was re-evaluated to determine the impact of the changes on the performance metrics.

This iterative calibration process enhanced the model's detection and classification precision, ensuring high accuracy and reliability in its predictive outputs.

A.3.4 Re-train until results are acceptable

The model was engaged in multiple iterations of re-training and validation, each cycle informed by the insights gained from the previous testing outcomes. This iterative process was underpinned by a commitment to excellence, with the following steps:

- **Iterative Training:** The model was re-trained with adjusted weights, incorporating the lessons learned from each testing phase.
- **Validation:** After each re-training cycle, the model was validated against a separate subset of the dataset to monitor improvements and ensure no overfitting occurred.
- **Convergence Monitoring:** The training continued until the performance metrics converged to a satisfactory level, indicating that the model was ready for deployment in real-world scenarios.