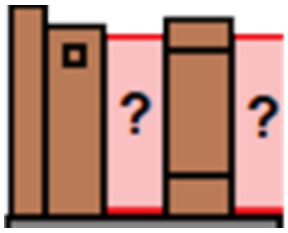
King Saud University
Collage of Computer and Information Sciences
Computer Science Department



"Retail Smart Inventory Management based on Real-time Analytics"

CSC 496 – Final Report

Prepared by:

Abdulrahman Alalyan	441102443
Ibrahim Alathbah	441103443
Khalid Alghamdy	441100858
Mohanned Alghonaim	441101453

Supervised by:

Dr. Bander AlSulami

Software project for a Bachelors degree in Computer Science

Third semester 1444

Spring 2023

I.Acknowledgements

We would like to thank all King Saud University faculty members, especially Dr. Bander AlSulami for his continuous support.

II.English Abstract

The Retailing market possesses a large share of the world's economy. This is especially true in Saudi Arabia's economy, with numbers exceeding billions of dollars. The losses therefore are of huge scale, resulting from issues such as product unavailability that can negatively affect customer experience. Despite the current existence of some systems that help reduce such problems like the POS(Point-of-Sale) system, it does not take into account lost or damaged items.

Traditional solutions appear in the form of routine inspections by the staff themselves throughout the day. A model of such is considered expensive, ineffective, time-consuming and error-prone, because it's managed by human beings.

Therefore, our aim in this project is to help solve this issue by offering product misplacement or absence detection. This will be done automatically, through surveillance cameras and the use of software that utilizes object recognition and artificial intelligence. The system will be integrated with other systems to achieve an all-in-one complete inventory management system. It will be based on real-time analytics to ensure reliability and dependability.

III.Arabic Abstract

تمتلك تجارة التجزئة حصة كبيرة في اقتصاد العالم، وخاصة في اقتصاد المملكة العربية السعودية تتجاوز أرقام تصل الى مليارات الدولارات. فتكون الخسائر بمقياس ضخم نتيجة معضلات مثل عدم توفر المنتج، وقد تساعد العملاء على فقد اهتمامهم. بغض النظر عن الأنظمة المتوفرة حاليا في سوق العمل لحل هذه العقبات مثل نظام نقاط البيع (Point-of-Sale System) فلا تستطيع هذه الأنظمة من التعرف والتعبير عن فقدان أو تلف البضاعة.

فالحلول التقليدية تظهر على شكل عمليات تفتيش روتينية من قبل طاقم من العاملين طوال اليوم. باعتبار هذا النموذج فهو مكلف من ناحية الوقت والمال، غير فعال، وقد ويكون عرضة للخطأ بسبب ادارته من قبل البشر.

وبالتالي، هدفنا في هذا المشروع هو حل هذه المشكلة بتقديم نظام كشف عن تغير مكان المنتج أو عدم توفره تلقائيا في الوقت الفعلي (real-time) من خلال تصوير كاميرات المراقبة واستعمال برامج تسخر التعرف على الأشياء والذكاء الاصطناعي لتحليل البضاعة. ومن الممكن دمج النظام مع أنظمة أخرى لتحقيق نظام إدارة مخازن متكامل مبني على نظام تحليل في الوقت الفعلي يكون موثوق وبالإمكان الاعتماد عليه.

Table of Contents

Table of Figures

List of Tables

Chapter 1: Introduction

Who's more reliable machines or humans? This question can have differing answers depending on the context in which it is asked. In this case however, we firmly believe in the former. And that is the question that birthed this project.

We aim to utilize machine deep learning and pair it with live video surveillance to solve the problems of manual inventory checking. Some of these tasks that are up until now been manually done by a human include, but are not limited to: Price checking, product placement, missing products, and low stock. We intend to use deep learning models that already exist and fine tune them to a relatively high degree of accuracy for detecting the afore mentioned inconsistencies. This approach will not only cut costs for retailers around the world, but it will also be much more efficient and reliable than their human counterparts.

1.1 Problem Statement

Retailers all around the world are losing significant amounts of money that could have potentially been theirs if their shelves were stocked properly. Humans are error-prone and that is why we are trying to find alternative, more reliable solutions. With the rise of Artificial intelligence and great things that have been achieved through machine learning. We believe it's our turn to contribute to its rise by implementing AI into retailing in new and innovative ways.

Our objective is to remove the need for human intervention as much as possible. Instead of having a worker check the available stock day in and day out, we hope to automize this process by the way of cameras.

1.2 Goals and Objectives

Goal – Implementing a deep learning model that detects missing/misplaced merchandise using live video surveillance.

Objectives -

1- Train a deep learning model to determine the location of missing/misplaced merchandise in an image with a relatively high degree of accuracy.

2- Transport and preprocess the image from the video camera to the model in question, in real-time.

3- Apply the model on an experimental case.

1.3 Solution

We aim to minimize the monetary resources that have been dedicated to shelf management. By monitoring all the shelves in a retailing store, we can take stock without the need of human intervention. This can be achieved through the analysis of raw camera footage, which is then followed by an update to the retail stock system.

1.4 Research Scope

The scope of this project is to perform an object detection task, in which we determine the location of missing/misplaced merchandise in real-time through a video camera. This will be done by utilizing pre-existing deep learning models. We will use these models and train them further on new datasets to fine tune them.

Chapter 2: Background

2.1 Real Time Image Recognition

Image Recognition is the process in which frames flow from an image-sensor(camera) onto a computer for processing. Real-time refers to the pace of this workflow that happens quickly with minimal delay. Every frame goes through a detection phase, where different objects in the frame get discovered. Then, the features from the frame are extracted and processed to then be categorized and named. This process is done in a high-pace fashion, which might demand for a fast CPU. **For more information See (figure 1).**

Figure 1 Real Time Image Recognition example. [1]

2.2 Artificial Neural Network

Neural networks are the core of deep learning algorithms. As the name suggests it is inspired by the human brain and how the neurons signal each other. It should not be thought of as a recreation of the human brain. The resemblance is there, however, it is not enough to faithfully make the comparison.

The structure of a Neural network is split up into 3 sections. An input layer followed by an output layer with a myriad of hidden layers in the middle that are used for the actual computations. Each of these layers is composed of small nodes that take up small pieces of data and depending on the weight and threshold of said data, it will either send it to deeper layers for further processing or keep it where it is.

When training a Neural network, the results at the end aren't just taken as is. First, they are compared to the actual results. From there, they are fed back into the Neural network in a process known as **Backward Propagation** where the weight and threshold values of the data are adjusted accordingly. Repeating this process for multiple instances and including any outlier cases will most likely produce a model with a high degree of accuracy. **For more information See (figure 2).**

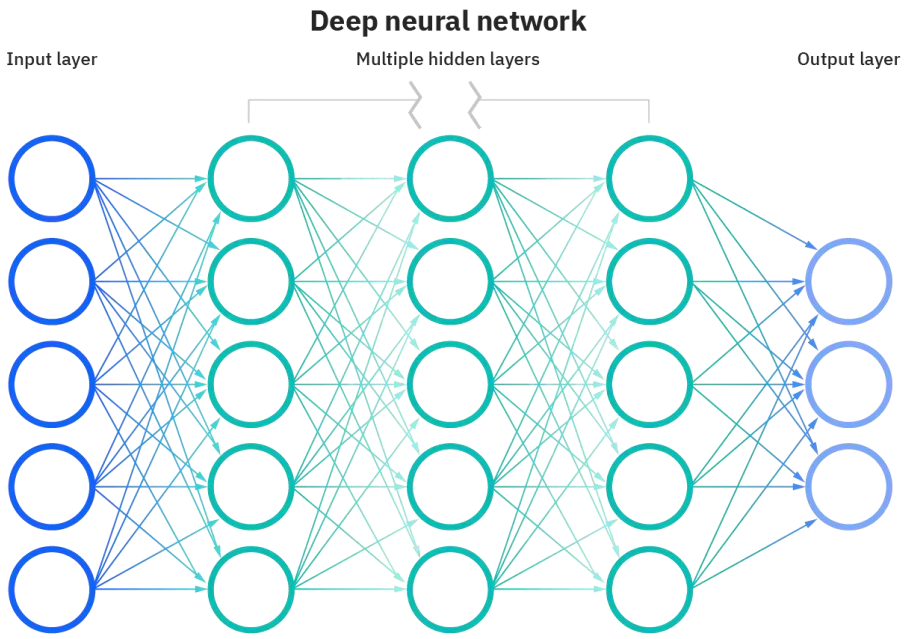


Figure 2 Neural network outline structure. [2]

2.3 Deep Learning

Deep learning is subset of machine learning that is based on artificial neural networks with multiple layers. The word "Deep" in deep learning refers to the number of hidden layers in the neural network. The layers correspond to a hierarchy of features or concepts where lower-level features define many higher-level features. Which why it is sometimes called hierarchical learning [3]. Because of the deep learning ability to draw features from raw data, it is widely used in many applications. For example, in image recognition, the lower layers could recognize edges and loops while the higher layers use them to identify the numbers.

Popular deep learning models in image recognition like YOLO use convolution neural network with multiple layers. Convolution neural networks are used in image recognition because it can extract features from adjacent pixels which are likely to be correlated. They also have the ability to reduce the number of parameters which are needed for pixel data like images.

The hidden layers in convolution neural network can be classified as such: convolutional layers, pooling layers and fully connected layers. Convolutional layers extract features from an area and output a feature map that is likely smaller than the input **(figure 3)**. Pooling layers further decrease the number of parameters by combining a group of neurons, typically by taking the max or average of them. Fully connected layers connect every neuron in one layer with every neuron in the next.

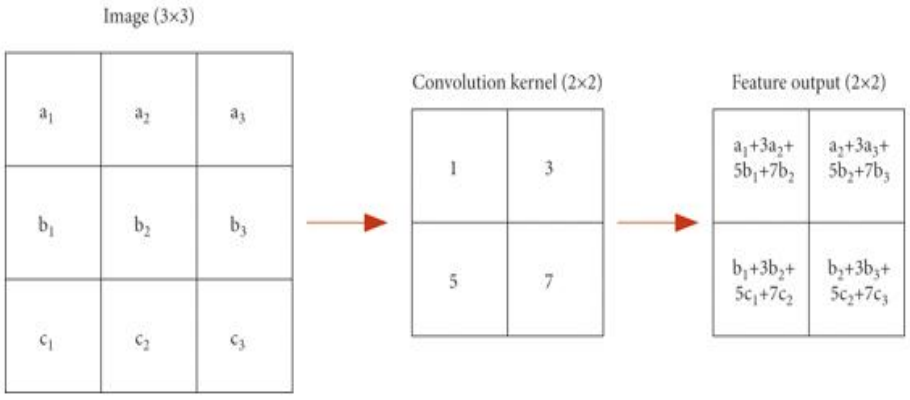


Figure 3 convolution operation example. [4]

2.4 Image Segmentation

The process of grouping together portions of an image that correspond to the same object class is known as image segmentation. This method is additionally known as "pixel-level categorization". To look at it another way, it includes dividing up images or video frames into different items or segments that are of the same nature**. For more information see (figure 4).**

Figure 4 image segmentation example. [5]

There are however, multiple types of image segmentation. Semantic segmentation, Instance segmentation and Panoptic segmentation are the three main types of segmentation. **For more clarification see (figure 5).**

2.4.1 Semantic Segmentation

In semantic classification pixels belonging to a specific class are simply categorized to that class, with no other information or context taken into account. When there are multiple instances of the same class in the image, as one might expect, it is an ambiguous problem statement. With no in-depth detail or information on the image, a semantic segmentation model would predict the entire stack region as belonging to the "plastic-plate" class in an image showing multiple plastic plates on top of each other.

2.4.2 Instance Segmentation

Dividing pixels into categories based on "instances" rather than classes. The algorithm has no idea which class a classified region belongs to. However, it can separate overlapping or very similar object regions using their box boundaries, if given the same picture of the plastic plates. The model would be capable of distinguishing each plate from

the stack as well as the surrounding objects. However, it would be unable to predict which objects are instances of which regions.

2.4.3 Panoptic Segmentation

The most recently developed segmentation tool. It can be described as a combination of semantic segmentation and instance segmentation in which each instance of an object in an image is separated and the object is identified.

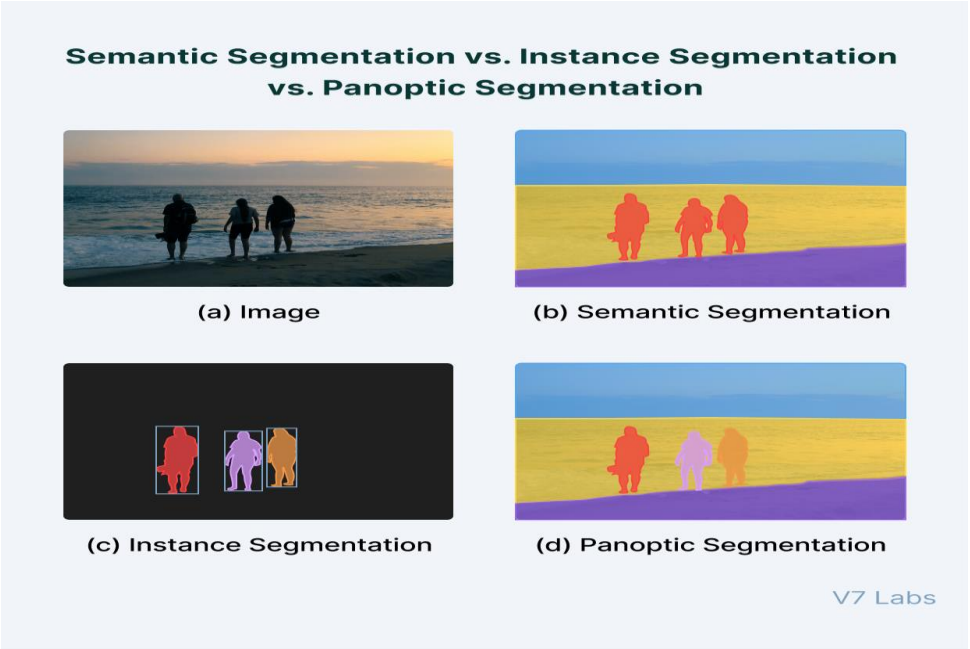


Figure 5 Types of Image Segmentation [6]

Chapter 3: Literature Review

The main idea behind the usage of product recognition was to facilitate shop commodity management and to improve the overall customer experiences. From recent years till now barcode scanning [7] product recognition has been used not only in research, but also in industries to help with management. This approach came with its own "solutions" that weren't so cost effective. Studies have shown that 45% of customers were dissatisfied with the convenience of barcode scanning methods, due to the uncertainty of the barcode position and the line-of-sight requirement that cost them time [8]. RFID (radio frequency identification) [9] is starting to be used in businesses that are forward thinking, and willing to try new things. This technology works by having RFID tags placed onto products which then transmit data to a detector to identify themselves. This approach, like many others has its limitations. It is not able to accommodate for multiple wave signal detections, due to the interference caused by each other. Also, RFID tags are expensive and

hard to recycle, which made this solution impractical.

Deep learning has been in the circle of focus for enterprises for many years. These enterprises aim to modify the way retailing stores operate. They hope to achieve a better, more innovative future. Studies have shown that the global expenditure by retailers on AI technology will increase to 12 billion dollars by 2023[10]. Deep learning is a subfield of machine learning that pivots around deep data representation. This concept was first proposed by scholars in the field of machine learning in 2006 [11]. Soon after, two researchers presented methods for solving the vanishing gradient problem [12]. A year later, deep learning attracted interest from many. In 2007 for example, a greedy layer-wise training method was developed to optimize the initial weights for deep networks [13]. In 2012 the dropout algorithm [14] was proposed in the effort to prevent overfitting and to speed the performance of deep networks. Recently, the application of deep learning in retailing, and specifically product recognition mainly covers two components. These are: Object detection and image classification. With the recent interest and growth of deep learning, many frameworks were developed to hasten the process of training model creations such as, PyTorch[15], TensorFlow[16] and Caffe[17].

The thrive and glory of deep learning leans mainly on and heavily profits from Convolutional Neural Networks(CNNs), which were inspired by the researches conducted on a cat's visual cortex [18]. LeCun and others, first proposed the idea of employing CNNs to classify images in 1988 [19]. They created a CNN model LeNet that had seven layers and trained it on a dataset of 32*32 handwritten character images. This model then showed its success in correctly identifying checks and was later then applied. The structure of the CNN changed in a drastically in a positive way beginning after the year of 2010. With the parallel advances in GPU power, a series of network structures were devised. Some of these include: GoogLeNet[20], VGG[21] and ResNet[22]. They were all based on LeNet. Classifying 3D objects also began picking up steam with advancements in "Multiview CNN", which allows the network to take in multiple images for one instance [23].

In 2015 a group of researchers from the university of Washington led by Joseph Redmon published a paper that introduced a new approach to object detection called *Yolo*, which stands for You Only Look Once. Unlike other approaches like R-CNN, which" first generate potential bounding boxes in an image and then runs a classifier on these proposed boxes" [24]. Yolo utilizes" a single convolutional network that simultaneously predicts multiple bounding boxes and class probabilities for those boxes" [24]. This new approach gives users many benefits that may not be available elsewhere.

First, Yolo is very fast. Since it doesn't require any pre-processing, it can immediately run on a new image and predict its contents. Second, Yolo sees the entire image when it's attempting to classify it. Comparing that with the R-CNN approach which has a habit of including some parts of the background in the bounding box. "YOLO makes less than half the number of background errors compared to Fast R-CNN" [24]. Third, Yolo is more general in its application. It is less likely to fail when given vastly different image contexts. Finally, as good as the YOLO approach is, it's not perfect. Yolo struggles when it attempts to classify objects in unusual angles or aspect ratios. [24]

In 2018 IEEE International Conference on Image Processing, Tonioni and Di Stefano proposed a new approach for product recognition on store shelves [25] that consist of three steps. First, is detection, it aims to obtain a set of bound boxes that surround every product given an image of a store shelf. A convolutional neural network object detection model is cable of doing the first step. The second is recognition, it takes a crop image of every bound box from the first step then uses K-NN similarity search on each one, using a database of the products in store. The third is refinement, it aims to remove false detections from the first K-NN in the second step to increase accuracy. The full system with the goal of recognizing all visible products belonging to food category (~ 3200 product) achieved a mean average precision of 73.50% and recall of 82.66%.

In 2019 a research paper under the name of "RPC: a large-scale retail product checkout dataset" was published. It showed how product recognition which is deep learning based accelerated the development of object detection. In this paper, we see product recognition as a sub problem of the object detection research problem [26]. Some large technology companies have utilized deep learning methods to recognize retail products in order to operate unmanned stores in recent years. Amazon Go was the first unmanned store to be open for the public, it opened in 2018. There they installed many CCTV cameras in the store. The cameras detect the customers' interactions and identify the products they are purchasing by using deep learning computer models. Nonetheless, the image recognition accuracy remains a major challenge. As a result, other technologies, such as Bluetooth and weight sensors, are utilized to ensure that retail products are correctly identified. Shortly after the Amazon opened their unmanned store, Walmart followed suit and designed a new retail store called Intelligent Retail Lab. The store was officially opened to the public in 2019. Deep learning was used with cameras to automatically detect out-of-stock products and alert staff members when to restock. Came soon after was DeepBlue Technology, a Chinese company, which developed automatic vending machines

and self-checkout counters based on deep learning algorithms that can accurately recognize products using cameras.

In Proceedings of the 2021 AAAI Conference. The authors of "On Rethinking object detection in retail stores" [27], believed that the traditional object recognition representation of an object instance with a bounding box is not practical in the context of retail. Due to extreme occlusions between clusters of instances belonging to the same category. Therefore, they proposed a new strategy for retail related systems. Which was to perform a task of object localization and counting at the same time, abbreviated as "Locount". However, a dataset or benchmark for Locount task does not exist. As a result, they were forced to gather their own object localization and counting data set, which included 1.9 million instances. They then developed a new evaluation protocol which provides penalties for missing and duplicated instances. Finally, they compare the performance of the algorithms on the Locount task.

All in all, most deep learning-based models and approaches are still considered to be in their early stages. More research and experiments are required in this area. We aim to utilize and improve upon the work of our predecessors.

Chapter 4: System Analysis *

4.1 Functional Requirements

1- The system shall be able to analyze the cameras footage and update the database accordingly.
2- The system shall be able to connect and display the contents of an external database.
3- The system shall be able to document all the changes that it does on the database.
4- The system shall provide a simple tool that aids the manager in linking the model's outputs to what is in the database.
5- The system shall provide the option to print a PDF report containing all the changes that the system did on the database.
6- The system shall be able to detect misplaced products and send a notification to the manager informing them about the specifics.
7- The system shall contain configurable settings that the manager can edit freely.
8- The manager shall be able to add any number of cameras.
9- The manager shall be able to print out the system's report documenting the changes it performed.

10- The manager shall be able to configure the system's configuration such as frequency of image snapshots taken.
11- The manager shall be able to view the camera's footage.
12- The manager shall be able to able to undo any of the system's updates to the database.
13- The manager shall be able be able to view all the model's possible outputs with images accompanied.
14- The manager shall be able to edit what the camera is supposed to have in its view. This can be done manually or by utilizing the model.

4.2 Non-Functional Requirements

1- All system functions must take less than 5 seconds to complete.
2- The system must be written in python.
3- The system database must be SQL.
4- The system must be designed with customer convenience in mind.

Chapter 5: System Design *

5.1 System use-cases:

5.1.1 Use-Case Diagram

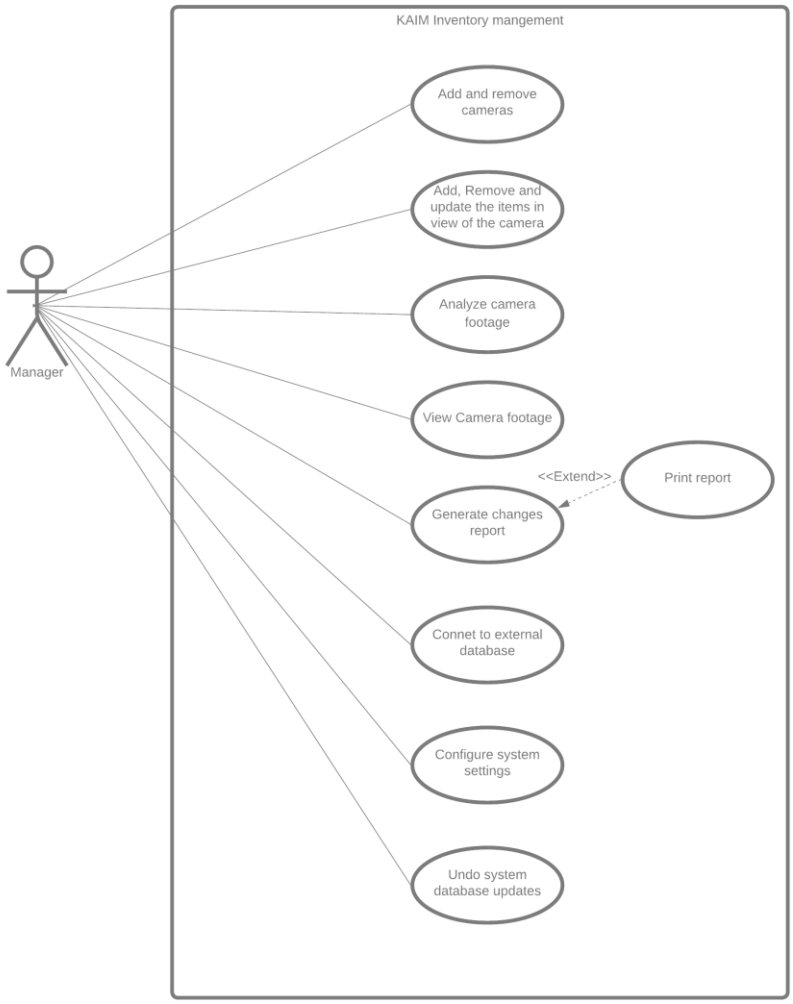


Figure 6 Use-Case Diagram

5.1.2 Use-Case Description

Use case Name: Analyze Camera Footage

Actor: Manager

Precondition: System settings, database and cameras are all setups.

Post condition: Database is updated accordingly.

User Actions	System Response
i. The manager starts the analysis.	i. System runs each camera in a separate thread and feeds their footage to the model periodically. Updates database whenever a change is detected. **Exp1, Exp2**
ii. The manager stops the analysis.	ii. System analyzes stops.

Exception 1: Database, settings or cameras are not setup, Exit use case.
Exception 2: Camera connection is lost mid process, Exit use case for this specific camera only.

Table 1 Use-Case Description

5.2 Interaction Diagrams:

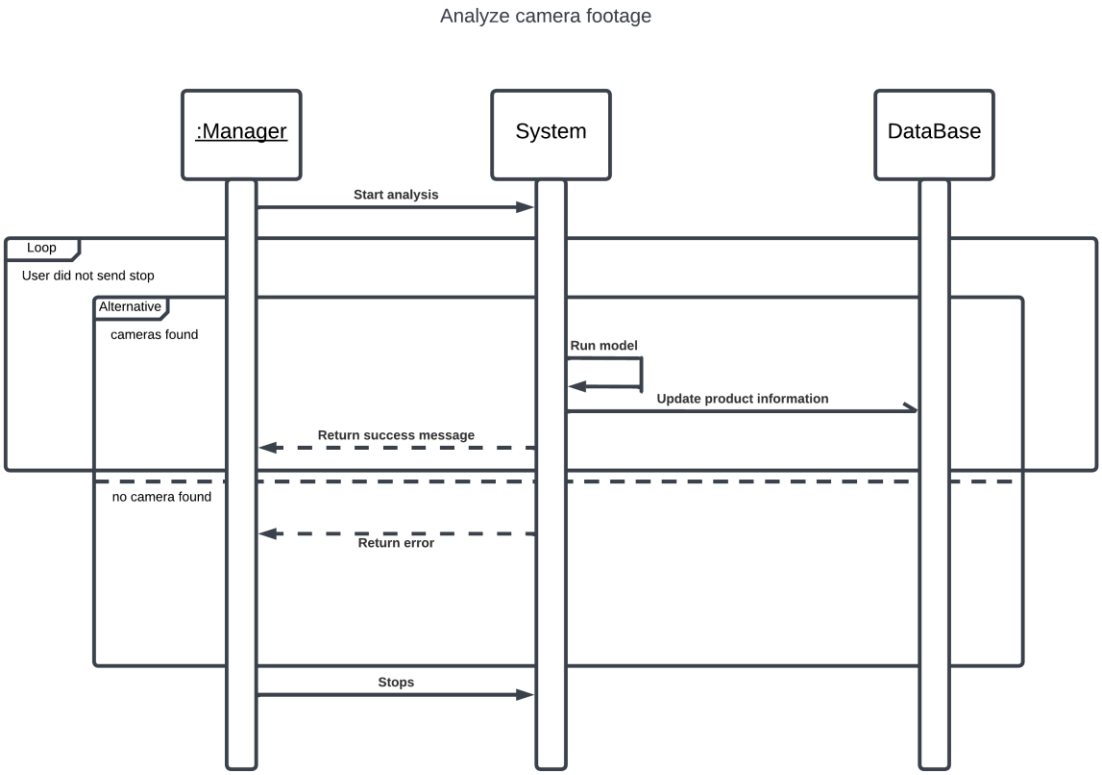


Figure 7 Sequence Diagram

5.3 Flowchart:

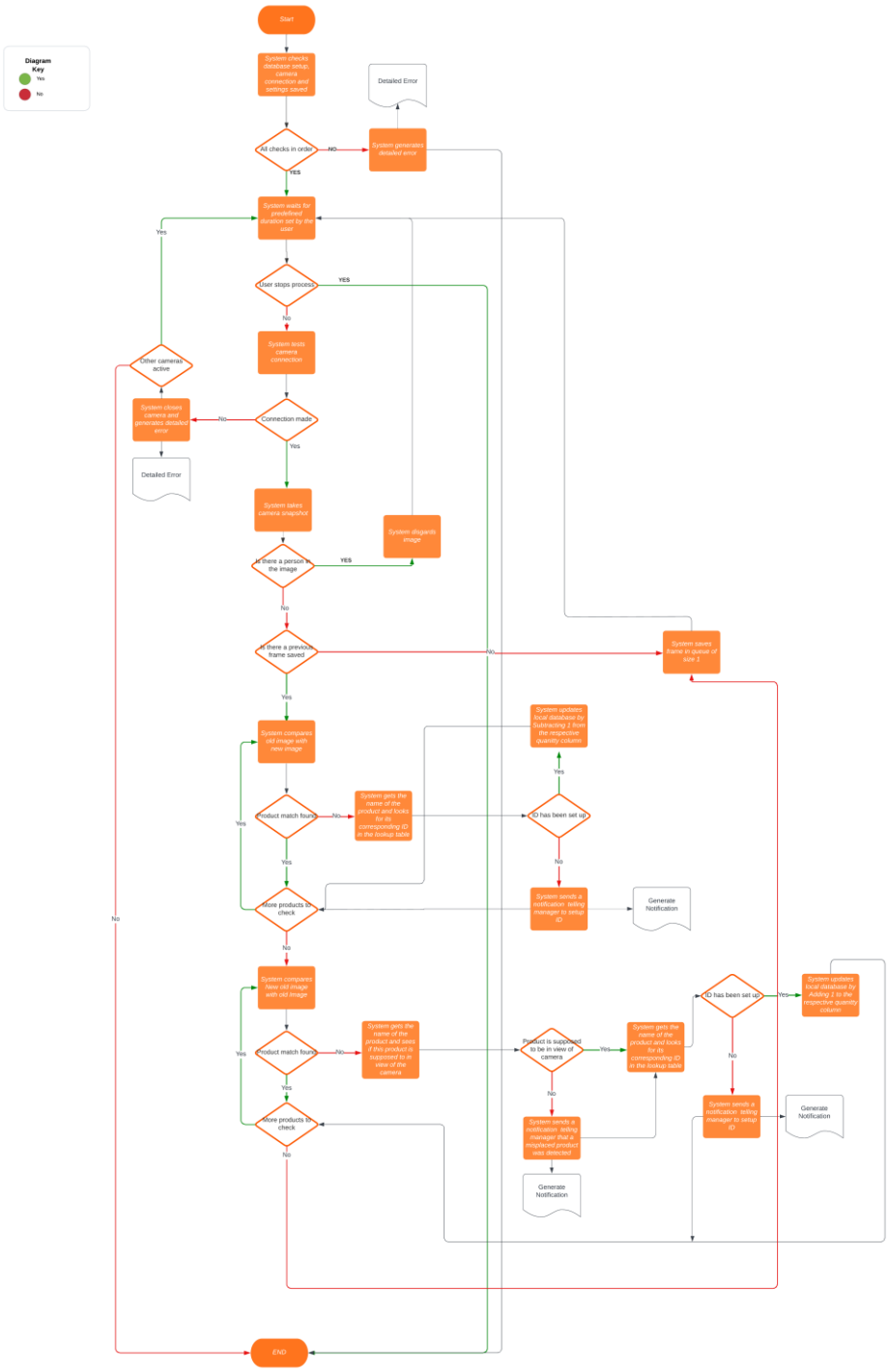


Figure 8 Main program loop flowchart

5.4 Class Diagram:

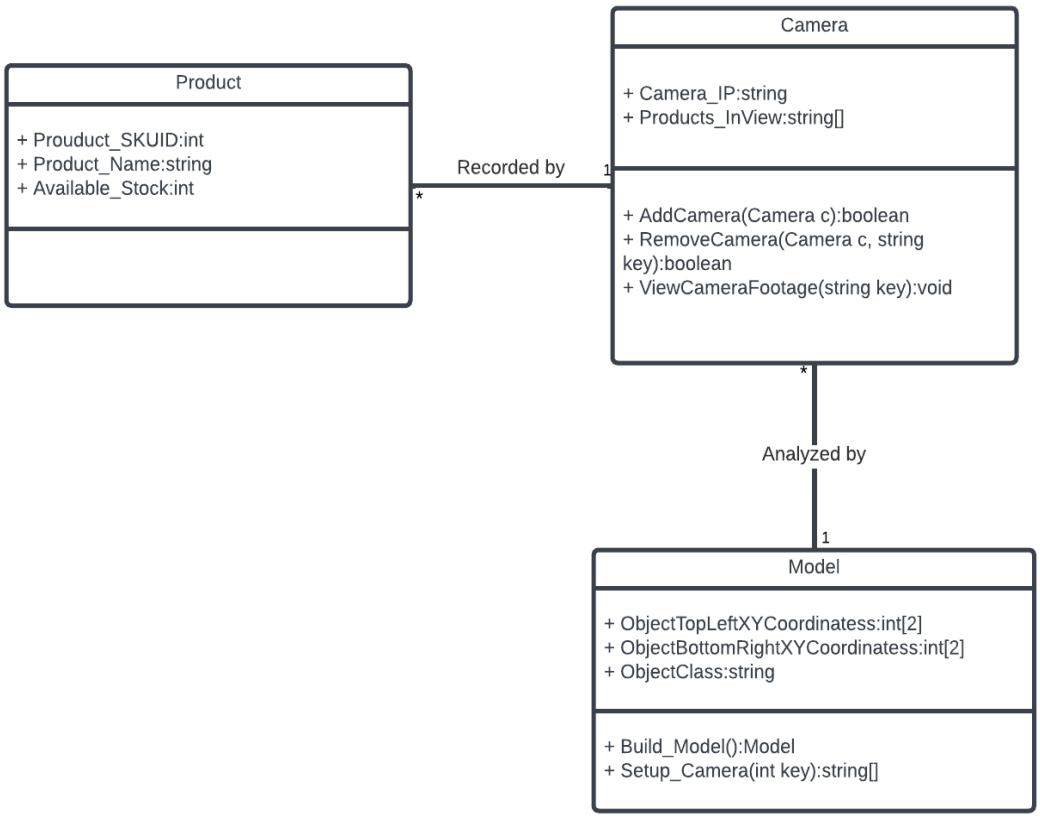


Figure 9 Class Diagram

5.5 System Architecture:

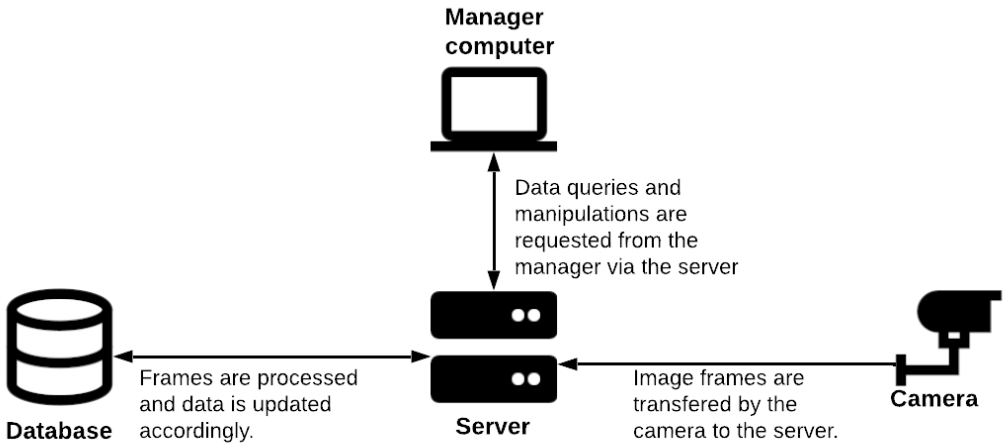


Figure 10 Software Architecture Diagram

5.6 Database Design:

5.6.1 Entity Relationship Diagram

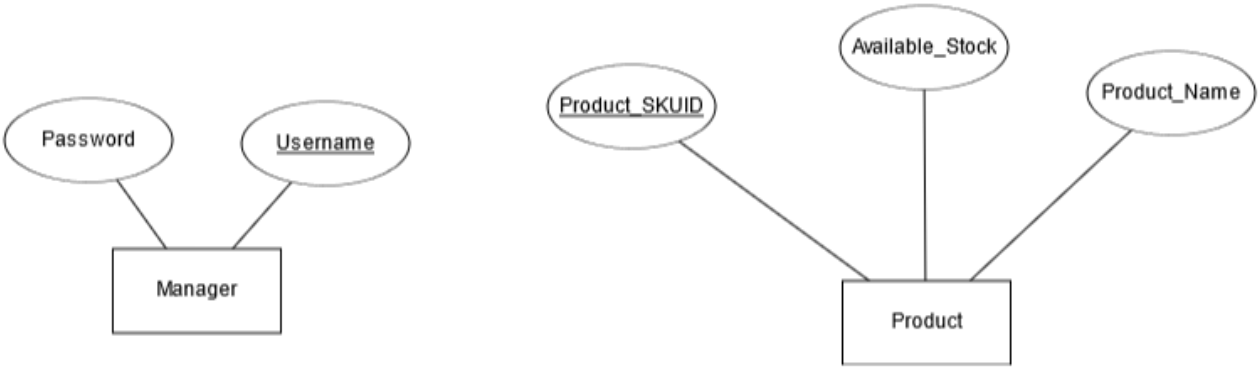


Figure 11 Entity Relationship Diagram

5.6.2 Database Schema

Manager

Manager_Username	Manager_Password

Product

Product_SKUID	Product_Name	Available_Stock

Figure 12 Database Schema Diagram

5.7 User Interface Prototype:

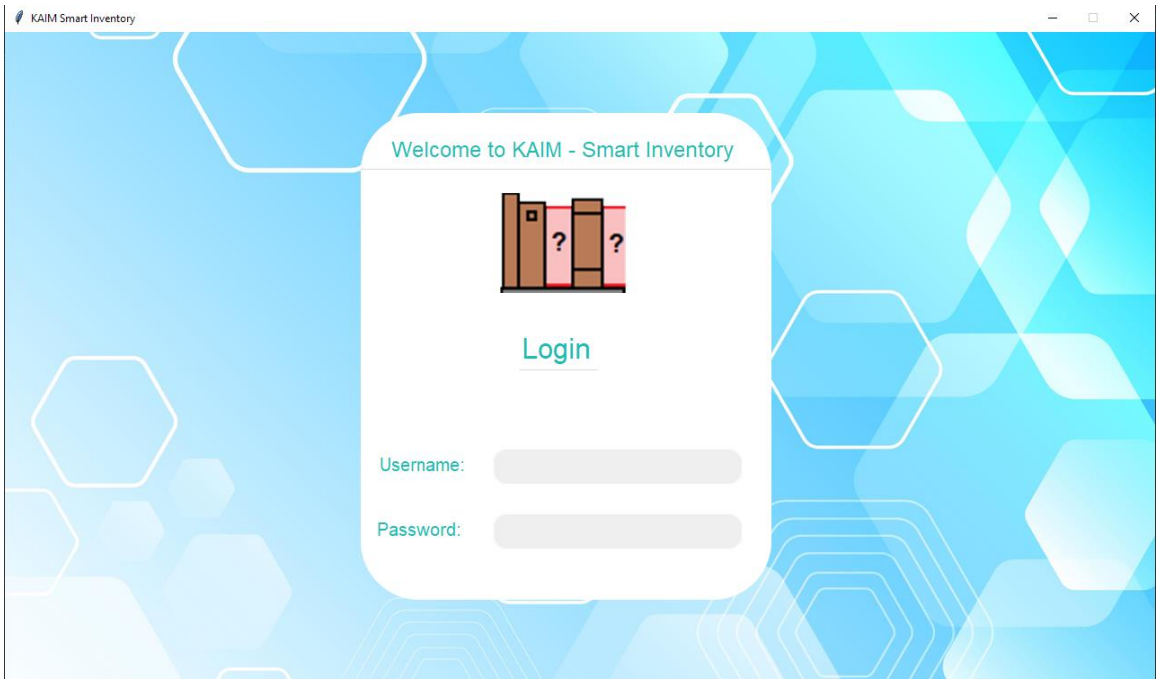


Figure 13 Login Screen

Figure 14 Main-menu window

Figure 15 Product view window

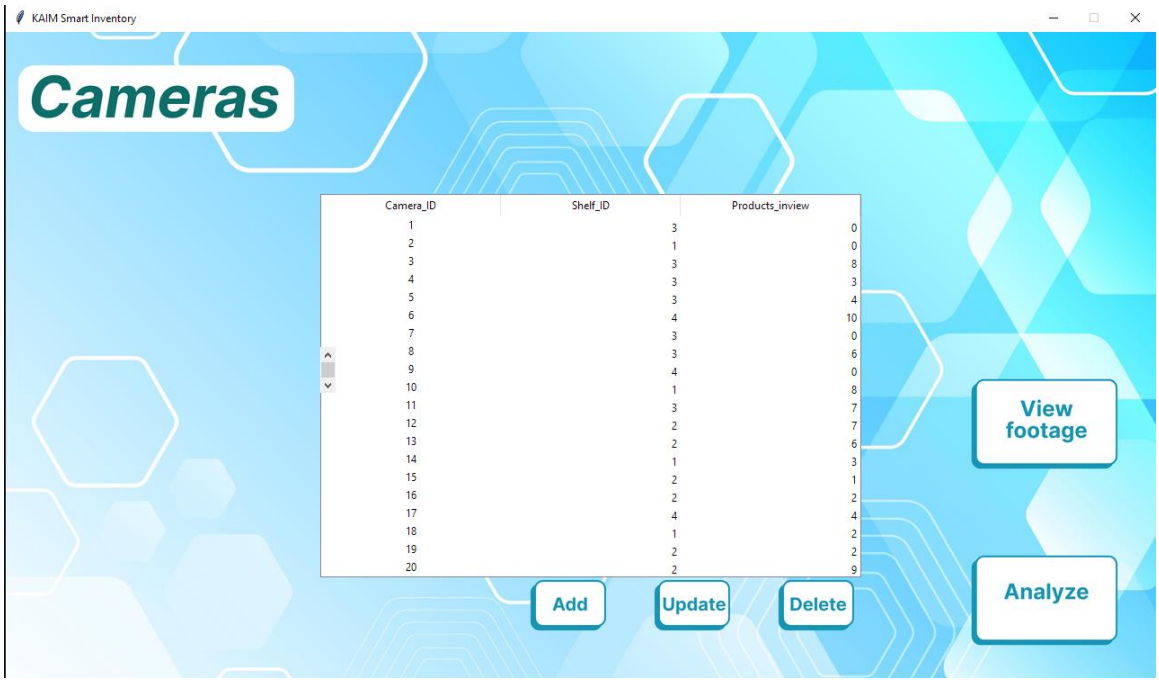


Figure 16 Camera view window

5.8 Algorithms:

The inventory system will use image processing algorithms. The model will run continuously and will take CCTV footage as input, analyze it, and then send feedback or changes in quantity to the linked database.

The System is composed of two models, the first take frame features as the input layer. The Second Model take first model output which is the objects as input. The second model output is the products which update the database.

Chapter 6: System Implementation**

6.1 Assets used:

In this section we are going to list all of the assets that were used in our project from libraries to datasets to programs.

6.1.1 Programs:

1- Visual Studio Code (Main programming Interface)
2- Jupyter Notebooks (For model training and testing)
3- Google Collab (For initial training purposes)
4- Microsoft Azure Cloud Services (For Database hosting)
5- QT Designer (GUI Creator)
6- IPWebcam (Use phone as Webcam)

6.1.2 Datasets:

1- SKU110K [28]
2- COmmon objects in COntext (COCO)[29]
3- ImageNet [30]
4- KAIM-RC (KAIM- Retail Classification)

6.1.3 Libraries:

1- PyQT5
2- Ultralytics
3- PyTorch
4- TensorFlow
5- Numpy
6- OpenCV
7- Requests
8- Os
9- Shutil
10- PIL
11- Mysql.connector
12- Webbrowser
13- Imutils
14- Sys

15- Time
16- Subprocess
17- Reportlab
18- QT-PyQt-PySide-Custom-Widgets
19- Datetime
20- Matplotlib
21- Pandas
22- Tqdm
23- Imgaug

6.2 System Implementation:

6.2.1 System overview:

In this section we'll briefly go over all the parts that make up our system. In the following sections we'll take each part individually and dissect it. Our system's main program was written primarily in python code, both front end and back end. However other "helping" languages were used such as Javascript, CSS and SQL. Our system consists of 2 major parts:

1- Main program
2- Computer Vision Deep Learning Models

When it comes to the main program, both front and backend were made by us. Our primary source of information were the documentation pages of the libraries in question. As for the Model approach we borrowed heavily from a research paper by the name of *"RP2K: A Large-Scale Retail Product Dataset for Fine-Grained Image Classification"*.[31]

This research paper tackled a similar problem to ours. As the name implies, they focused on fine grained classification of retailing products. They created a dataset with the name RP2K that consisted of 500000 images of 2000 different products. They then trained a Resnet34 model on their dataset to hit an accuracy of 95%.[31]

Their dataset however, contained products only found in Chinese retailing stores. Which is a problem because Chinese product branding is different from ours. For example, a coca cola found in a Saudi retailing store looks the same as one found in the US. However, this logic does not apply to China.

We were fortunate that they documented their entire dataset gathering process plus training. Their dataset was not usable to us, but their methods were, so we decided to gather our own dataset using their methods.

6.2.2 System pipeline:

To better understand our pipeline, we'll give a simple realistic scenario:

1- Start.
2- Camera monitors shelf
3- Server monitors camera footage
4- Server compares previous frame with current frame and detects a change
5- Server finds what that change was and updates database accordingly
6- End.

Now how does the server detect this change? This problem is split up into two parts, the "what" and the "where". The where in our case is "Object Detection" and the what is "Object Classification".

6.2.3 Object Detection:

For our object detection model, we utilized the SKU110K dataset.

"The SKU110k dataset provides 11,762 images with more than 1.7 million annotated bounding boxes captured in densely packed scenarios, including 8,233 images for training, 588 images for validation, and 2,941 images for testing. There are around 1,733,678 instances in total. The images are collected from thousands of supermarket stores and are of various scales, viewing angles, lighting conditions, and noise levels." [28]

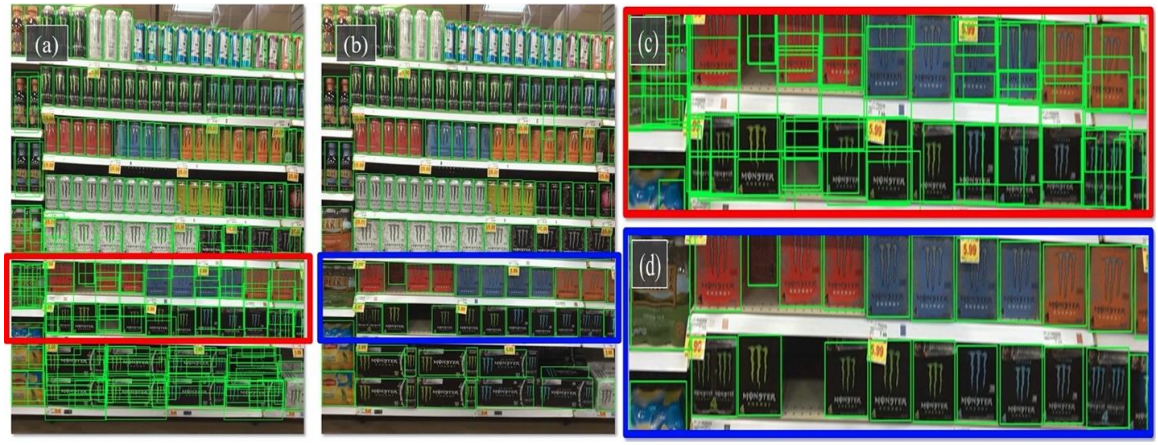


Figure 17 SKU-110K Example [28]

We utilized transfer learning and trained a pretrained YOLOV8 model which was trained on the COCO dataset on the SKU-110K Dataset. The model we chose was the "Yolov8n" detection model. [32]
We resized each image to 412px and trained the model for 25 epochs and we attained an accuracy of 82% .

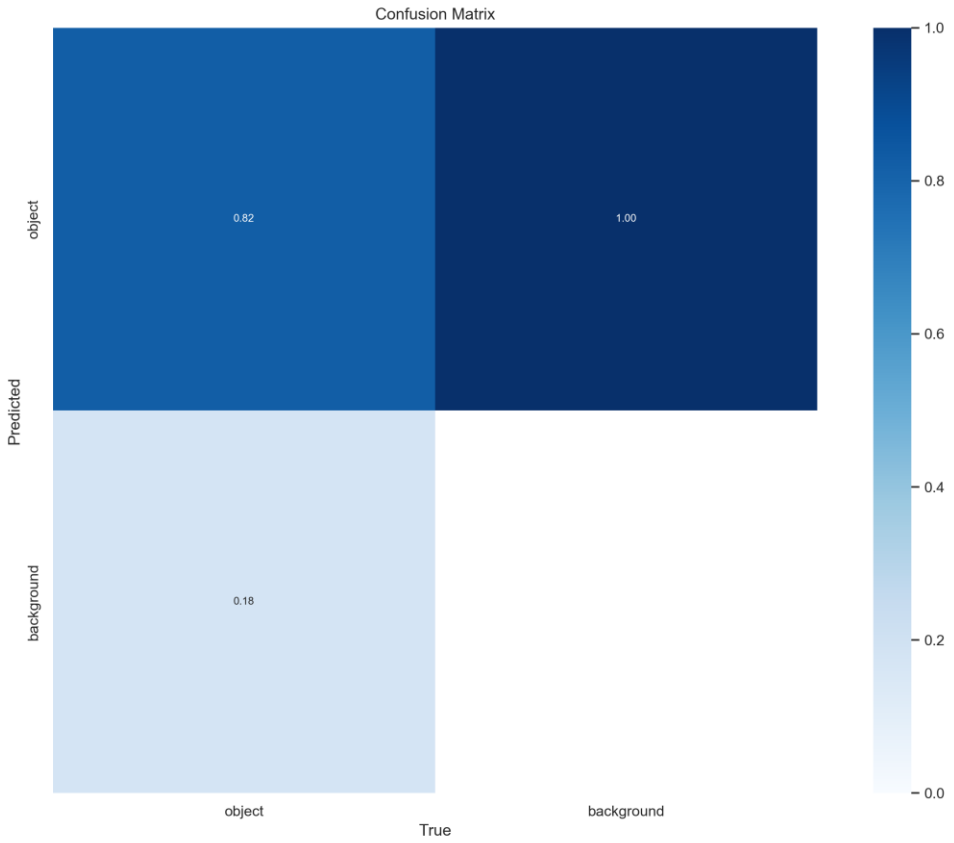


Figure 18 YOLOV8 25 epoch training results

We also used the Yolov8n model in our detection, not just for transfer learning. As we have stated before, the Yolov8n model was trained on the COCO dataset which among other things contains a "person" detection class. This is very important for our System and will be talked about in detail in later sections.

6.2.4 Object classification:

As stated in part 6.2.1, our classifier in both dataset and model, borrows heavily from the RP2K research paper methods. And thus, we believe we don't need to justify our model choice or preparation steps.

6.2.4.1 Dataset collection:

We decided to narrow down our collection to in our opinion two of the most prominent grocery retailing stores in Riyadh. Danube and Tamimi. The images were taken

by a Galaxy and an iPhone with image sizes 4032 x 1816 and 4032 x 3024 respectively. The phones were positioned on the opposing shelf before taking the image, roughly mimicking the position the shelf-monitoring cameras would be placed in the context of our system.

After collecting our images, we passed each image through our YOLOV8 model. Our YOLOV8 model then gave us the bounding boxes of all the products that it could make out. We then cropped all the bounding boxes out of the image. That left us with images of individual products. Each image gave us anywhere from 100-300 product images depending on the density of the products in that image.

Figure 19 An example of 1 of our images passed through our YOLO model.

6.2.4.2 Dataset Preparation:

After running the process in figure 18 on all our images, we had all we needed to begin the classifying our images. Before that however, we agreed on a naming scheme that all our classes would follow:

Object Type – Product Type – Company Name – Product Name

Object Type and Product type were pre-set, and any class would have to fall into one of these categories even if we had to take some liberties. They were as follows:

Object type (12):

- Pack
- Bag
- Box
- Jar
- Wrapping
- Bottle
- GlassBottle
- Container
- Can
- PlasticBag
- HandledBottle
- BoxedBottle

Product Type (21):

- SoftDrinks
- Dairy
- Non-SoftDrinks
- Tea
- Coffee
- Confectionery
- Canned
- Sauces
- Seasoning
- Jarred
- Fermented
- Soups
- DryFood
- BakingGoods
- AnimalFood
- Cleaners
- FrozenFoods
- PersonalCare
- Tissues

- PaperGoods
- PlasticGoods

As for the company name and the Product name these were not pre-set. Let's take Coca Cola as an example.

Figure 20 An example of how classes are named in our dataset

After a long monotonous process of classification and with deadlines approaching, we settled on 12,621 images classified into 602 classes. A problem that we faced after classifying our dataset was class imbalance. To fix this problem, we applied image augmentation on different subsets of our dataset a different number of times. We did this to close the large gap between our classes and to also make our dataset more generalized.

Image augmentation preset (Imgaug python library):

1. Random Horizontal flip
2. Random Crops
3. Adjust Contrast
4. Gaussian Blur
5. Gaussian Noise
6. Adjust Brightness

As stated above, we applied these transformations a different number of times for each subset of our data in the following order as follows:

- 4 times for classes with less than 5 instances
- 3 times for classes with less than 30 instances
- 1 time for classes with less than 50 instances
- 1 time for classes with instances between 30 and 90

Figure 21 An example of class that had 4 augmentations applied to each of its instances

After all the pre-processing steps taken, we were left with a dataset that contains 41,695 images of 601 classes. Below is a plot of the class distribution, pre and post augmentation. It's not perfect, but at least it's better than what it was.

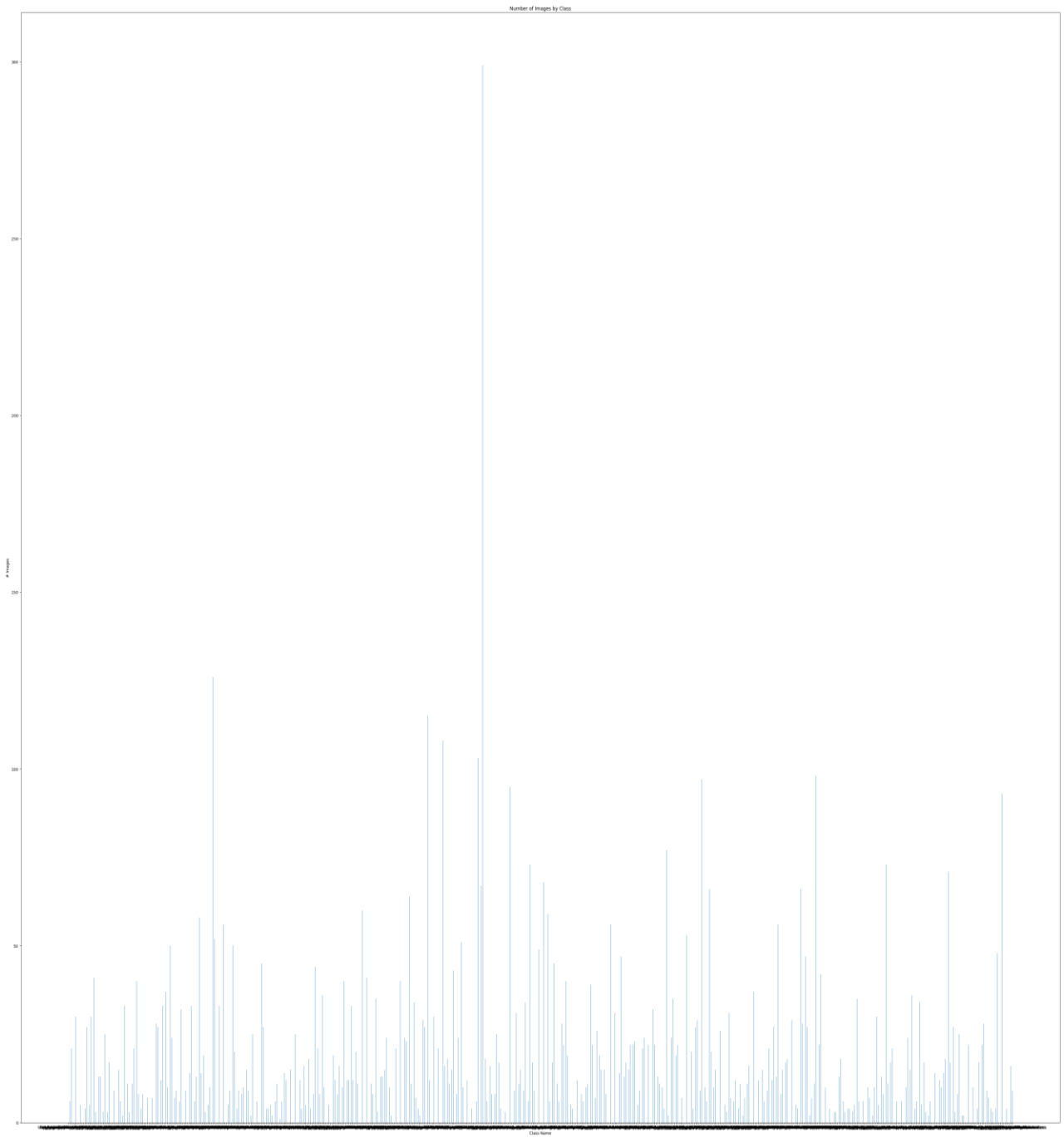


Figure 22 PRE-AUGMENTATION dataset class distribution

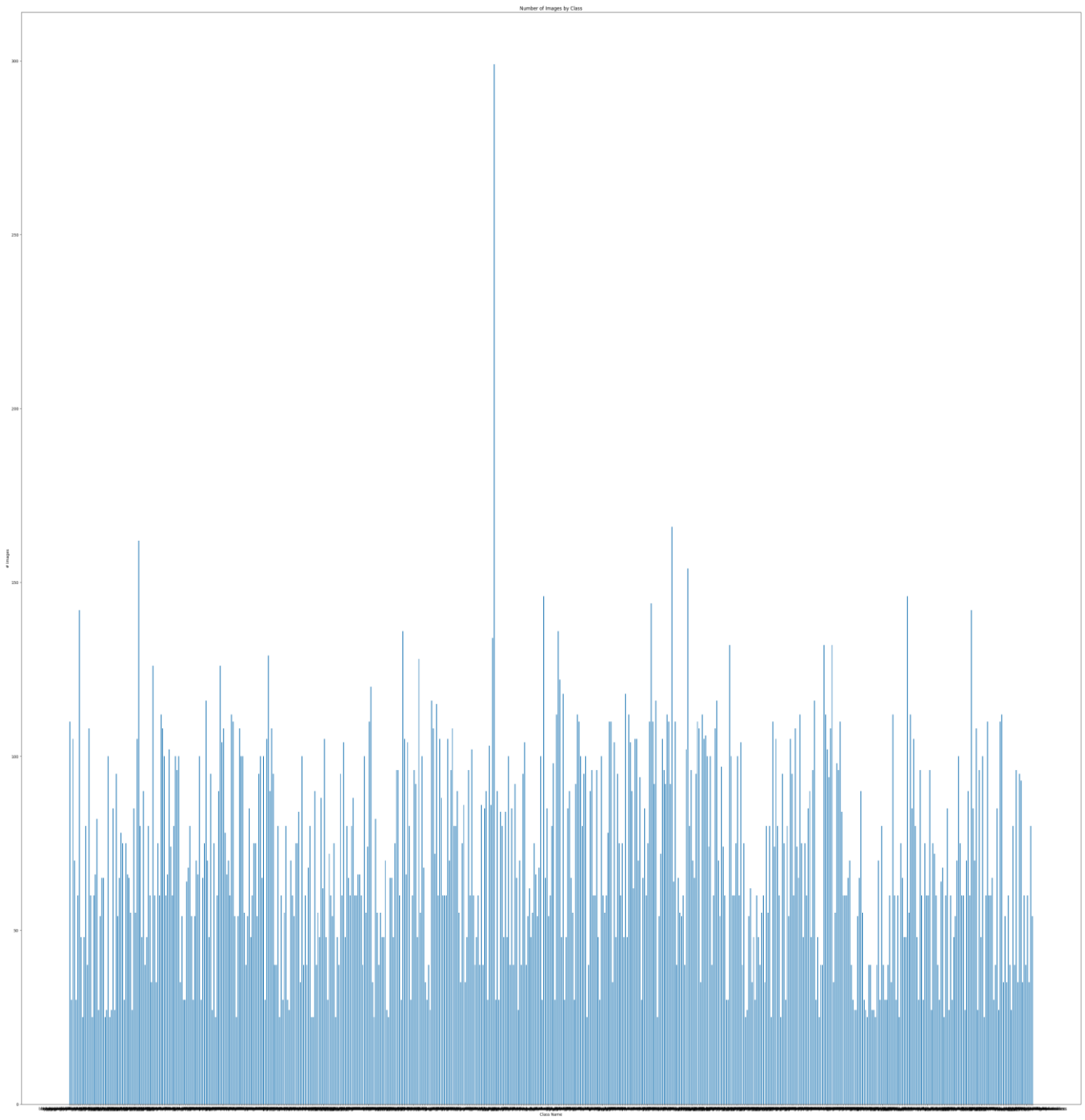


Figure 23 POST-AUGMENTATION dataset class distribution

6.2.4.3 Model Training:

With now our dataset being ready for use we immediately began training it. We again mirrored the RP2K approach and trained our dataset using a Resnet34 model[33]. And same as with the YOLO model we utilized transfer learning, the Resnet34 model was pretrained on the ImageNet[30] dataset.

We set up the training environment to support CUDA and trained the model on a NIVIDA GTX 1060 6GB GPU. We used Stochastic Gradient Descent (SGD) as our optimizer and cross entropy loss as our criterion. We set the learning rate to 0.1 and the momentum to 0.9 while leaving all the other parameters to Pytorch's default value and trained for 50 epochs. After training, our model had an accuracy of 96.738%. However, due to nature of our dataset and the imbalance that it contains this number may be misleading.

6.2.5 Main program:

In this section we'll talk about all the features of our program from GUI elements all the way to the main loop of our program. The GUI of this program was created entirely in QT Designer. Both CSS and JS were used in the process. The PYQT5 library was used to convert it to python code.

6.2.5.1 Login Page:

This is the first page that will be displayed on program startup. A simple interface prompting the user to input their username and password. The user has no option to sign up, leaving account allocation to us. After inputting a username and password and clicking verify, the program will search for those credentials in Kaim's own cloud hosted database hosted by Microsoft Azure. As for the system response to the validity of the data, we'll go more in depth about that in our testing section.

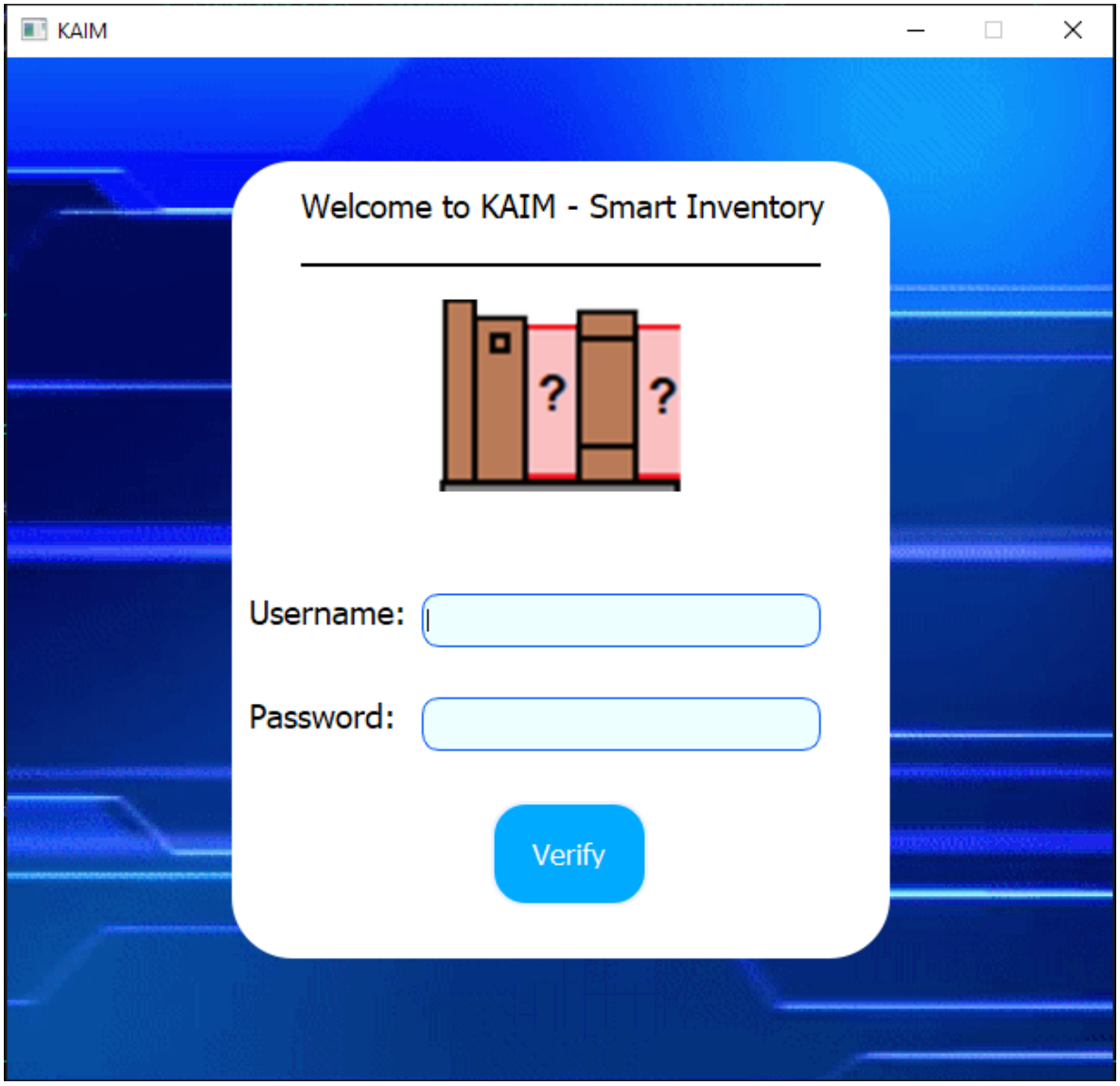


Figure 24 Kaim login page

6.2.5.2 Database Viewer Tab:

Upon a successful login, the program will terminate the old window and open the main program window. The default tab displayed will be the database viewer tab. The menu on the left is collapsible. The user has the option to cycle through the tabs on the left by clicking on them. This page will contain the database that will be updated by our model. We built our system with the assumption that the database will be stored on the cloud and not on premise.

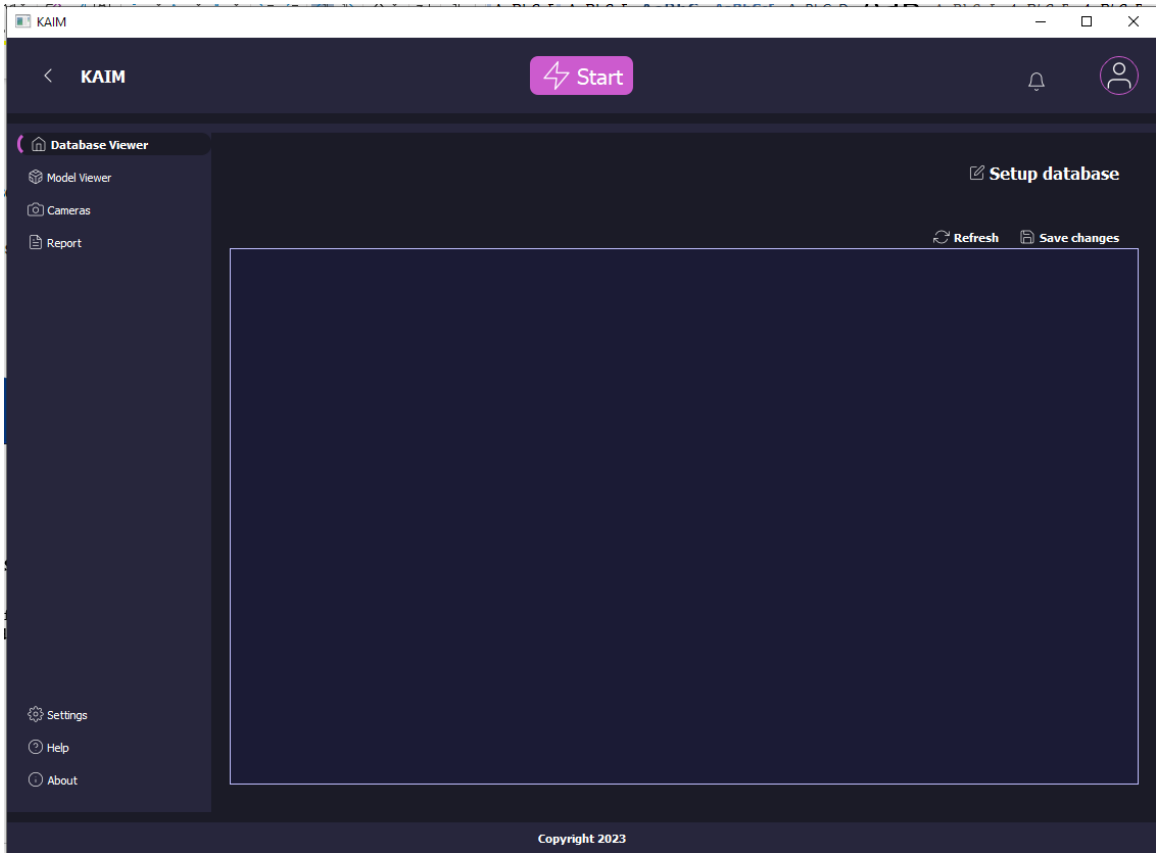


Figure 25 Database Viewer page

If the user clicks the "Setup database" button, the program will display a small tab that will ask the user to fill then their database's information. The window will be displayed in a sliding fashion. Javascript was integrated with our program to achieve this.

After the user fills in the data and clicks Setup, a loading icon will be displayed while program does it's work in another thread. While this loading icon is displayed, the program will not register any of the user's clicks. After confirming the information is correct, the system will then connect to the database and pull all the data from it. It will then display the data is an easy-to-read table format. This table is not edited directly by the user and is only editable by our model. The tabs on the top represent the tables and the user can cycle through them by clicking on them. After clicking Setup the information will be saved in a text file which functions like a database. This text file is read every time the program is started to load the user's saved information.

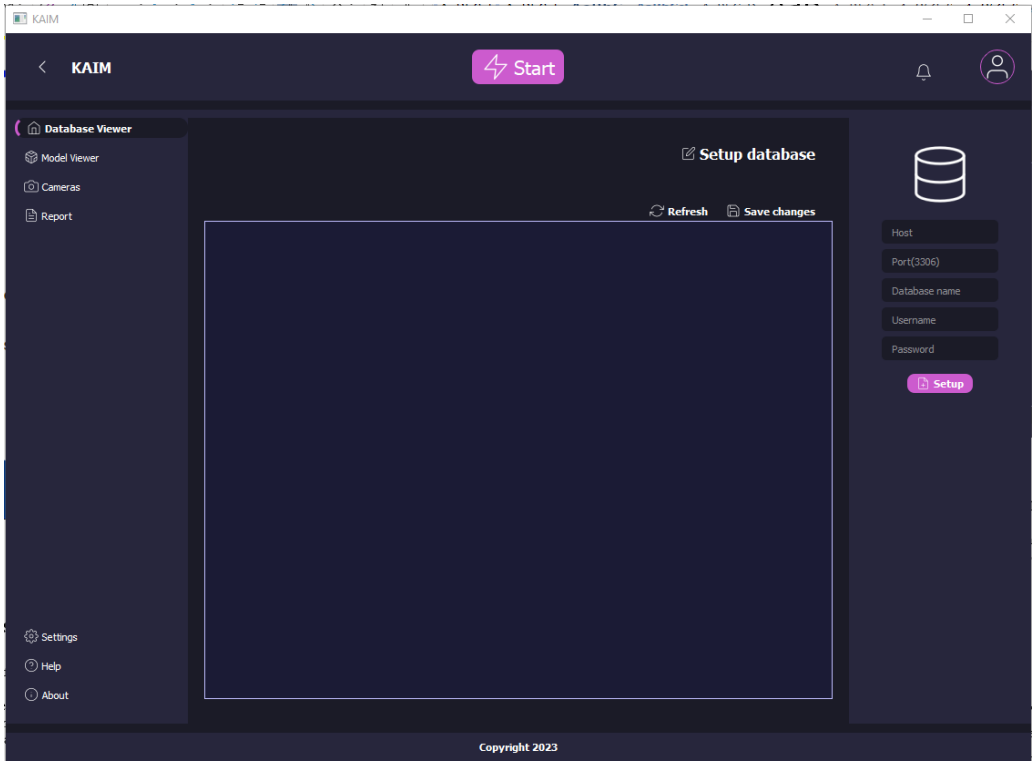


Figure 26 Setup database button clicked

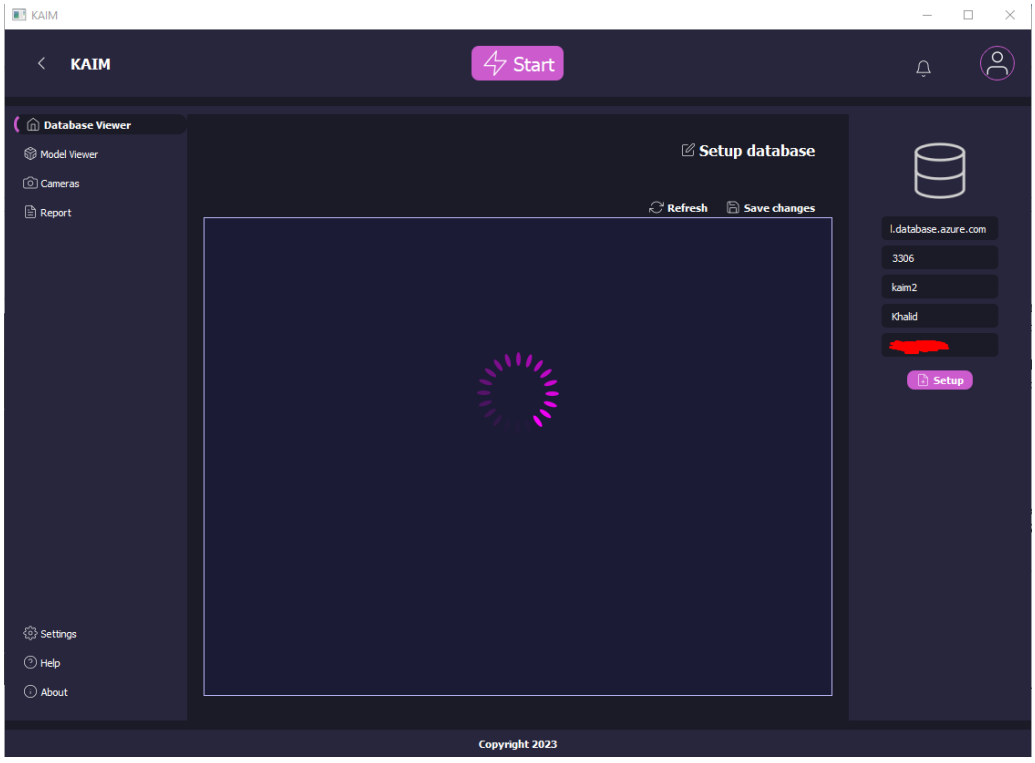


Figure 27 Loading icon displayed once Setup button is clicked

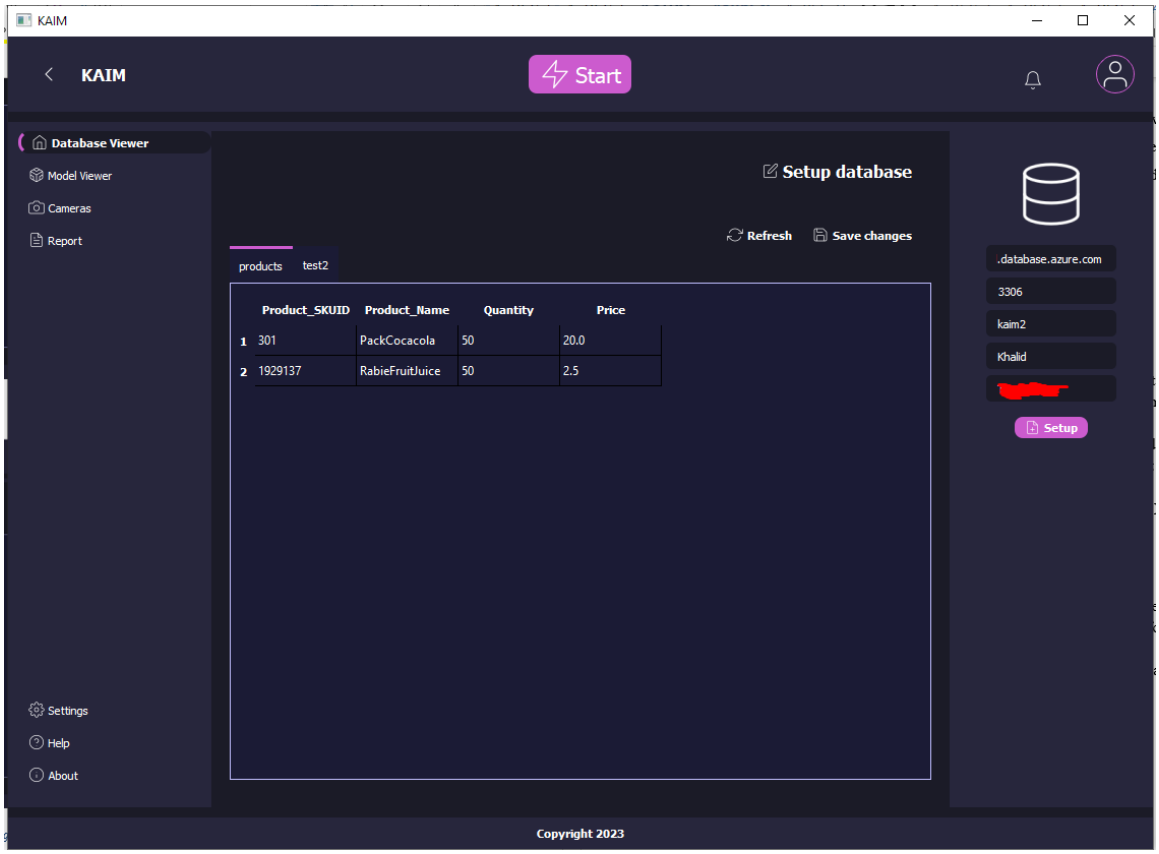


Figure 28 After loading database into the program

The user can perform two operations on the database, he can save and refresh the database. Refresh refreshes the database and displays the actual data in the database. When the save button is clicked the system will update the database with the new values in the table. However, it's important to point out that it doesn't replace the values in the database but adds either a positive or negative value to it.

Whenever a database is set up, the "Quantity" column which is set up by the user in the settings tab is read and its values are saved in a text file. Then whenever the user decides to update the actual database, the respective quantity value is read from text file and compared to the quantity value in the table. The difference whether positive or negative is then added to the database. This is done to make sure that changes to the database from external sources are not overwritten.

6.2.5.3 Settings tab:

The settings tab is very important and contains some parameters that must be setup by the user.

Snapshot Frequency: How frequent the user wants the system to capture a frame from the camera and process it.

- Column that contains the SKU ID of the products
- Column that contains the Quantity of the products:
- Column that contains the Name of the product

All these are self-explanatory and are needed so the system can know which columns to update. All three are combo boxes and their information is populated during the initial database setup. After clicking Save the information will be saved in a text file which functions like a database. This text file is read every time the program is started to load the user's saved information.

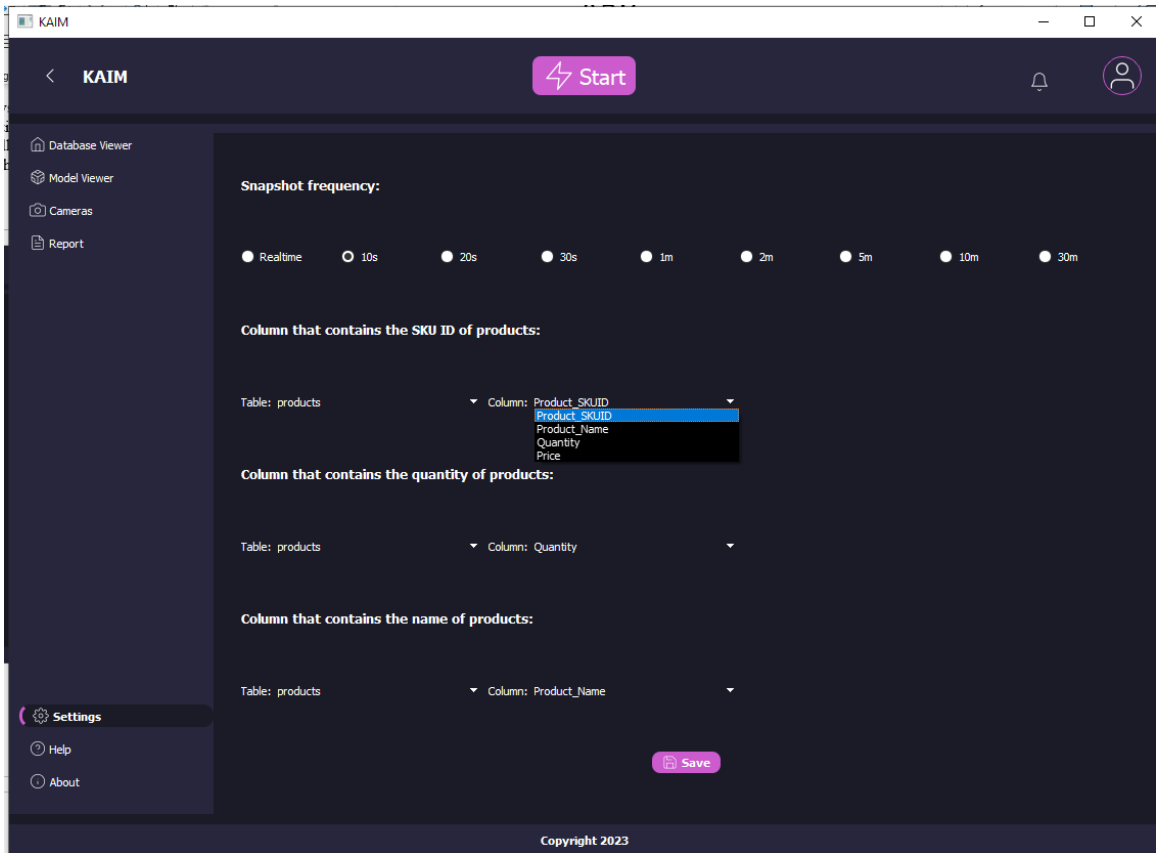


Figure 29 Settings tab

6.2.5.4 Model Viewer tab:

This page contains a table with two columns: Product Name and Product SKU ID. The latter column is initially filled with zeros. The Product Name tab contains all the possible outputs from our classification model.

This table functions like a lookup table and is a onetime setup process that is required by the user. The user is tasked with providing the respective SKU_ID to each of our possible model outputs. The user can either press on the setup model button on the bottom or double click on any of the rows to begin the process.

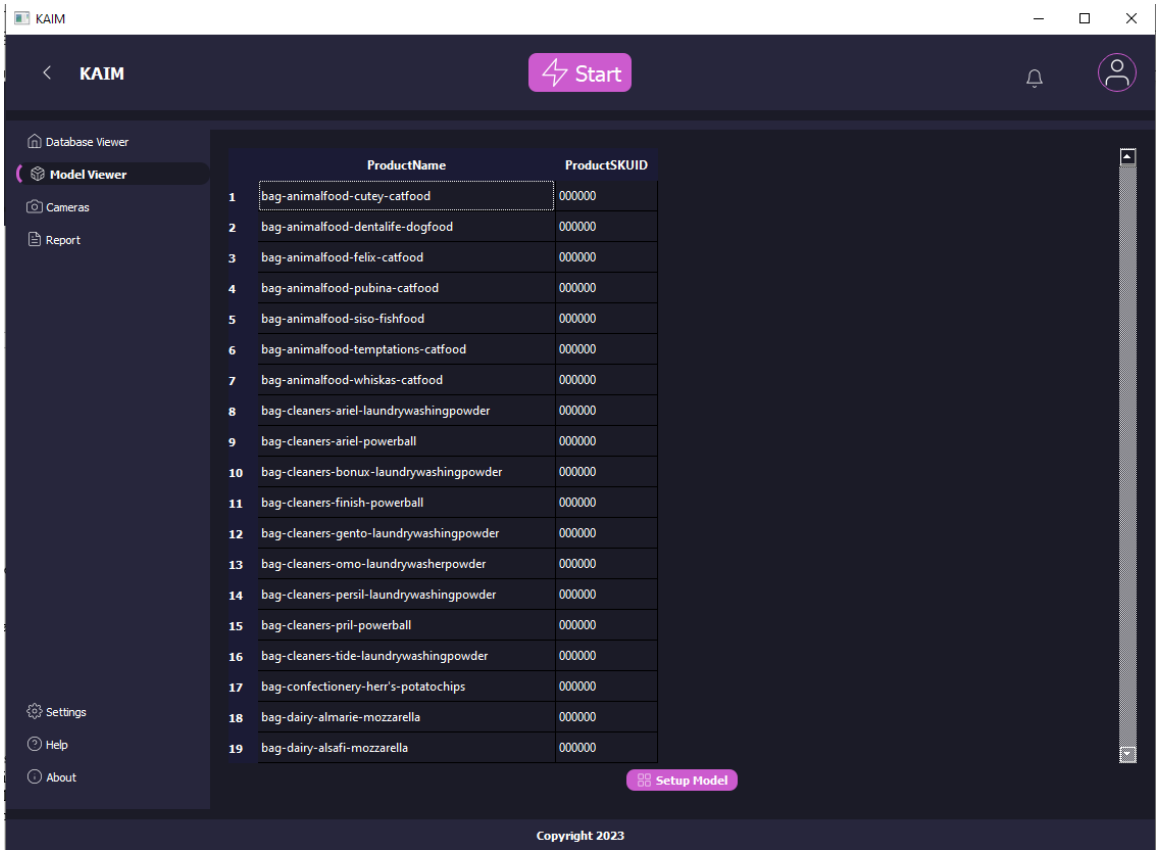


Figure 30 Setup Model page

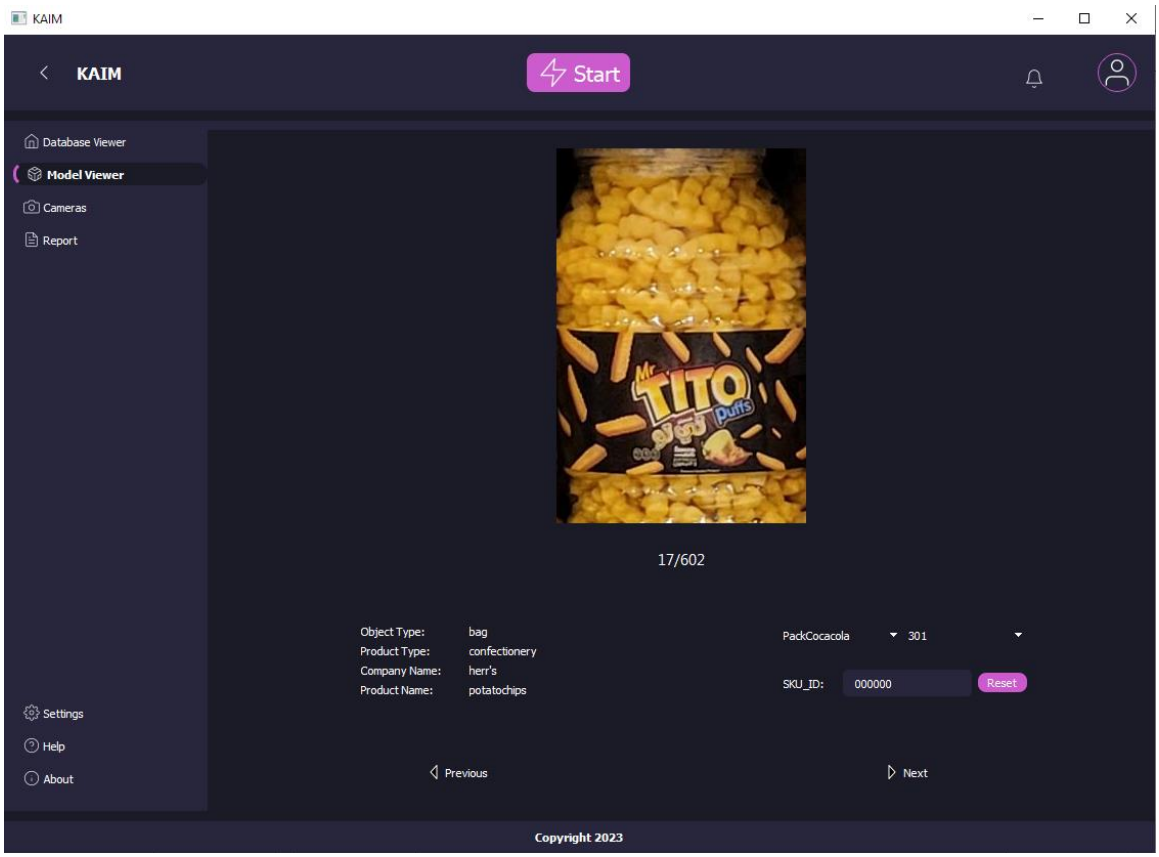


Figure 31 After double clicking on the herr's potato chips row

We created this simple tool to help the user in setting up the model. It provides an image of the product in question accompanied by its information. The user must then select the corresponding SKU_ID from the database. Assuming that both the database and settings tab were setup beforehand. The user simply selects the name of the product from the first combo box and it's corresponding SKU_ID will be retrieved and linked to product in question. The reset button will unlink the SKU_ID from this product and return it's value to the default 000000. The user will also have the option to navigate either the next or previous product by clicking the respective buttons on the bottom.

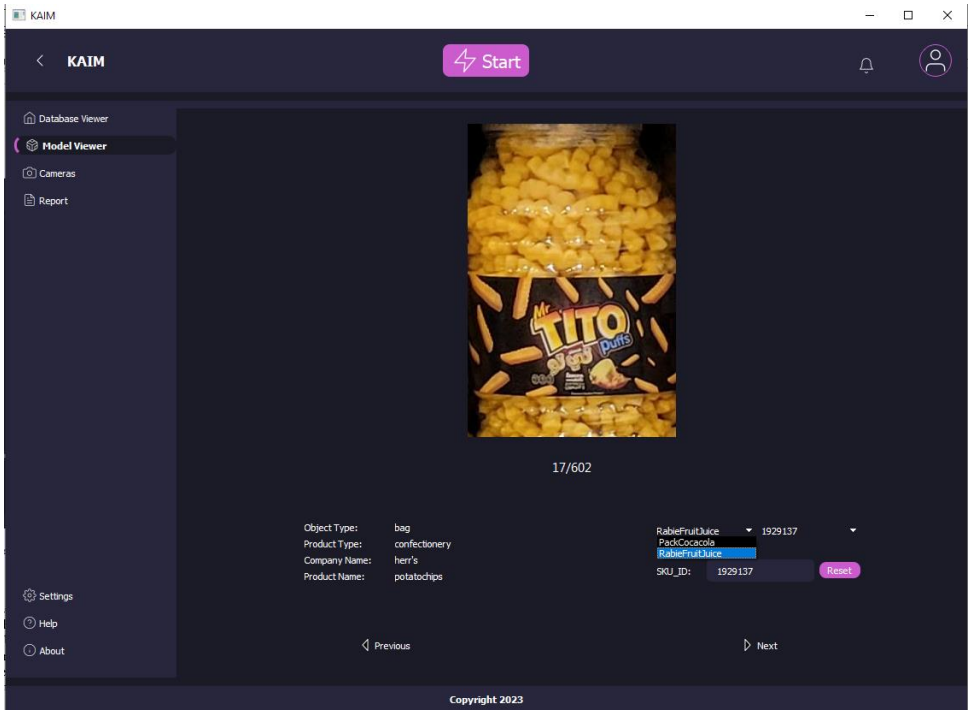


Figure 32 Linking SKU_ID with product

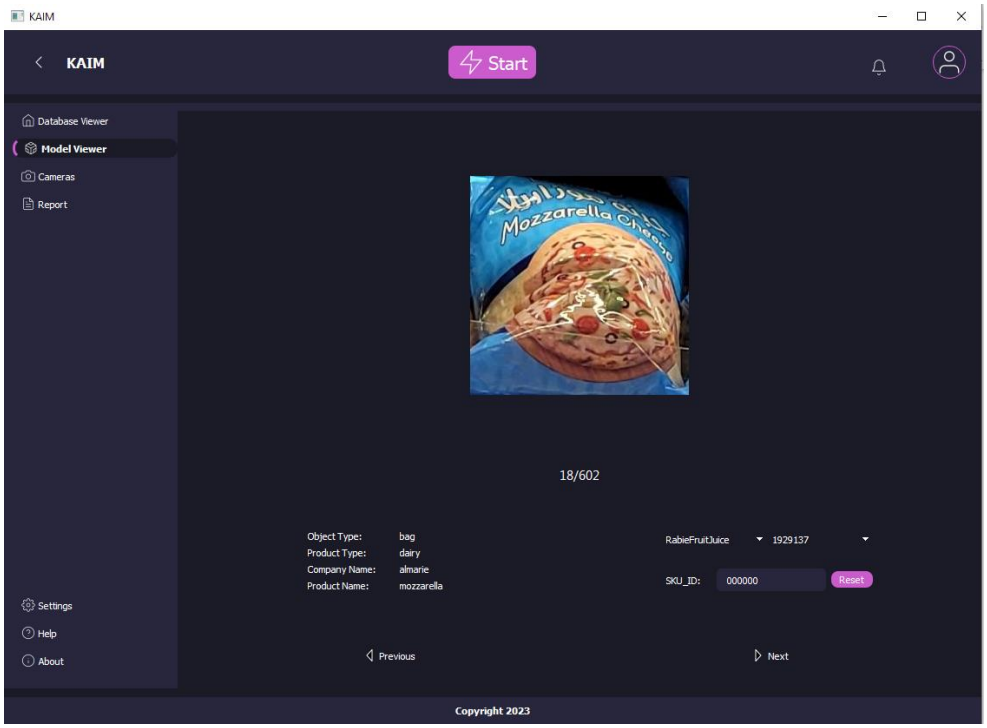


Figure 33 Clicking Next

6.2.5.5 Camera Tab:

The camera tab allows the user to add cameras and perform other needed procedures.

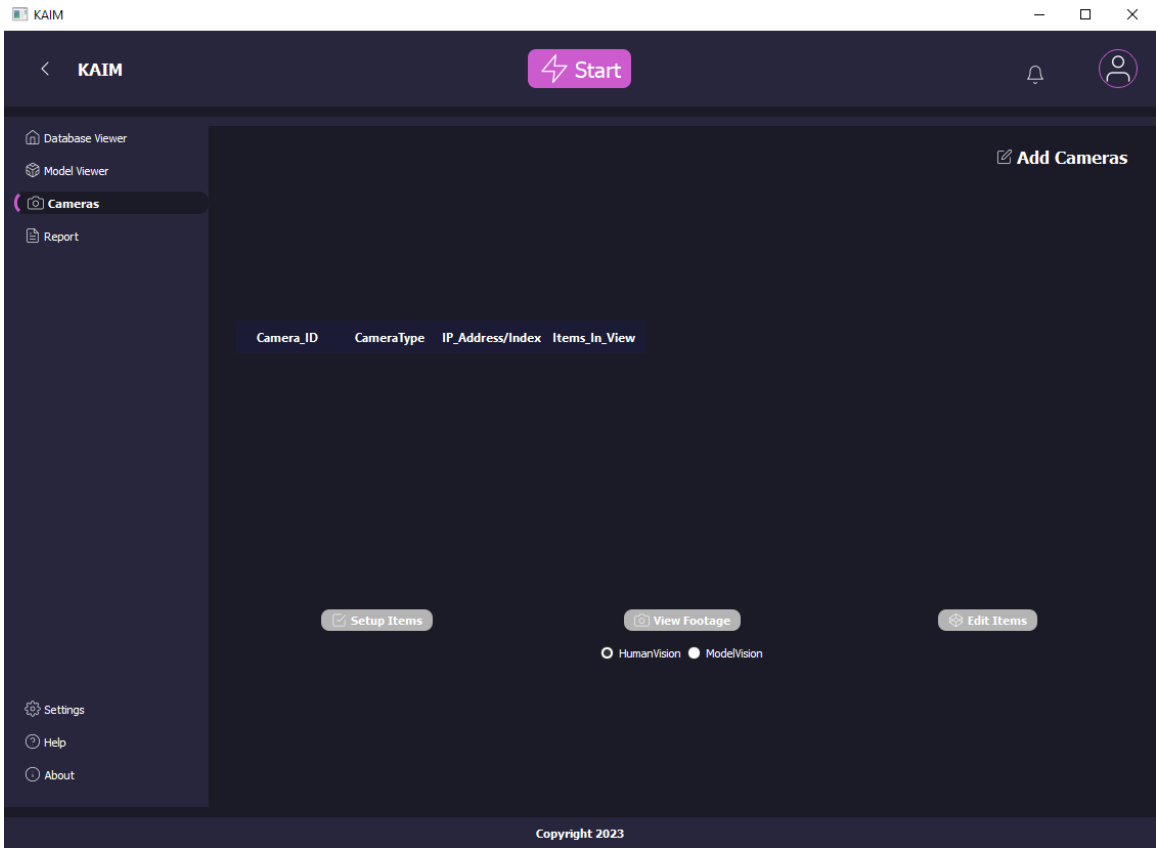


Figure 34 Cameras page

The user can Add a camera by clicking on the Add Cameras button which will display a small window for adding the camera specific information.

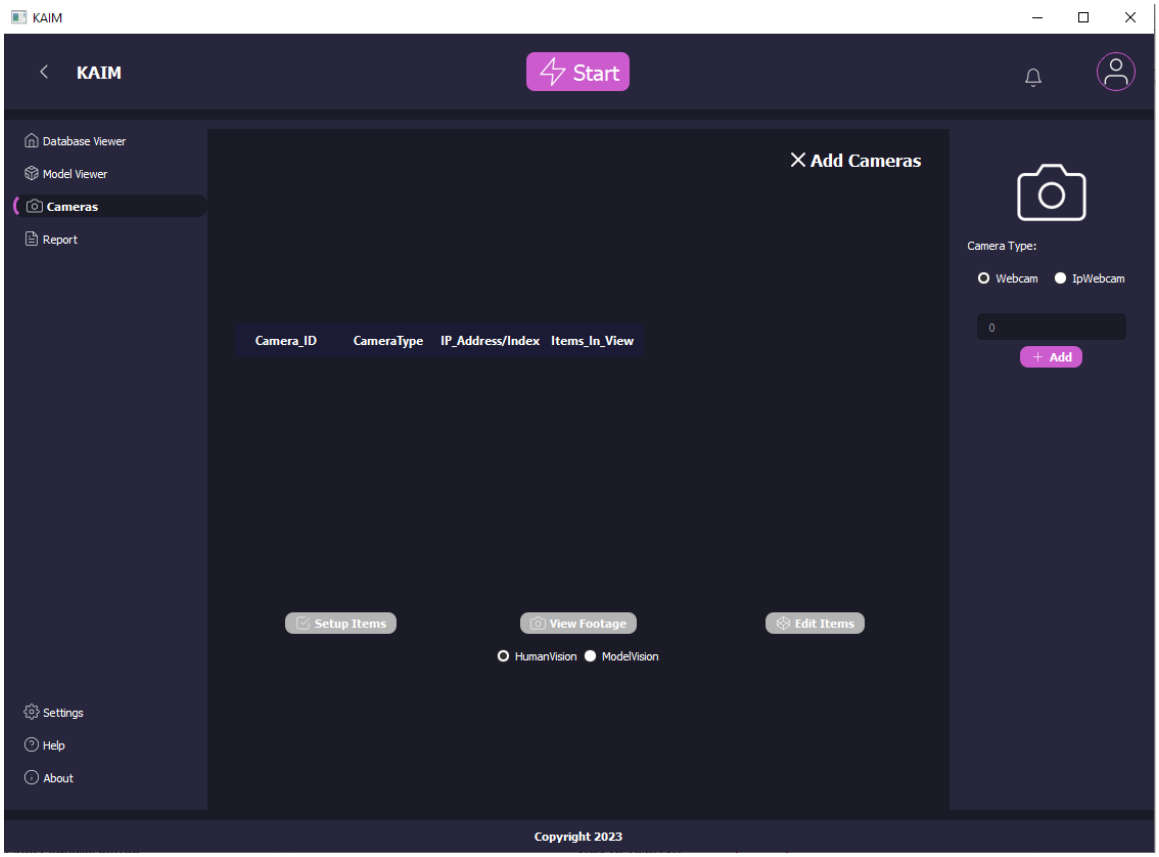


Figure 35 Add Camera

The user has the choice between a regular webcam and an IP Webcam. A regular webcam is any webcam that the device recognizes immediately, this can be both real and virtual. Cameras that the device recognizes can be accessed by index. For example, to access your laptop webcam you would enter 0. To access a connected webcam, you would enter 1 and so on. There are many guides online to know how many cameras are connected to your device and their respective indices, thus we don't delve too deeply into it.

As for the other option IP Webcam it differs slightly. IP Webcam is a free to download application that can turn your android device into a webcam by hosting it's footage locally on port 8080. For this method to work both phone and camera need to be on the same Wi-Fi network. The program is very straight forward to set up. After starting it an Ip will be displayed on the phone which then should be added to the system through the input box displayed.

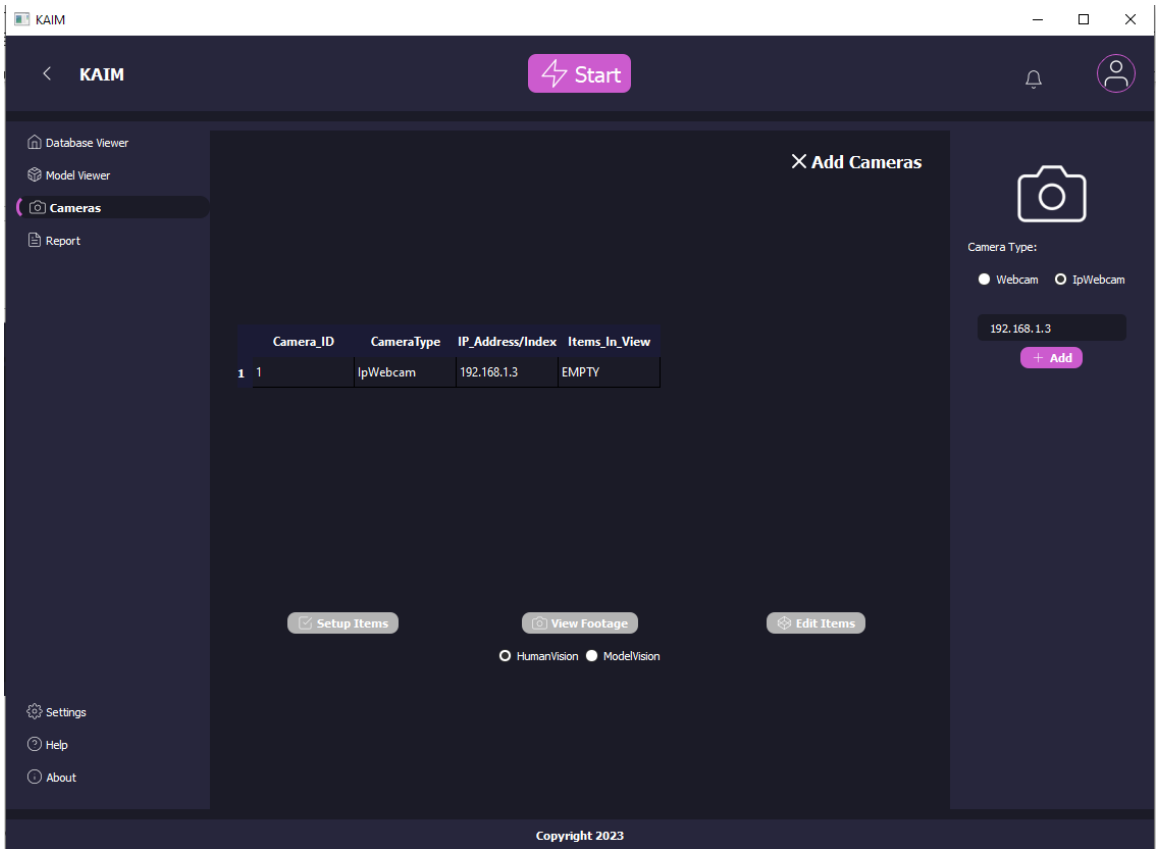


Figure 36 User adds Camera

After clicking Add the System will check the validity of the data and test the connection. If no red flags are raised, then the camera information is added to the table and saved to an external text file. This text file read on program start up and all information is retrieved.

The buttons on the bottom are all initially grayed out and inaccessible until the user selects a camera by clicking on a row in the table.

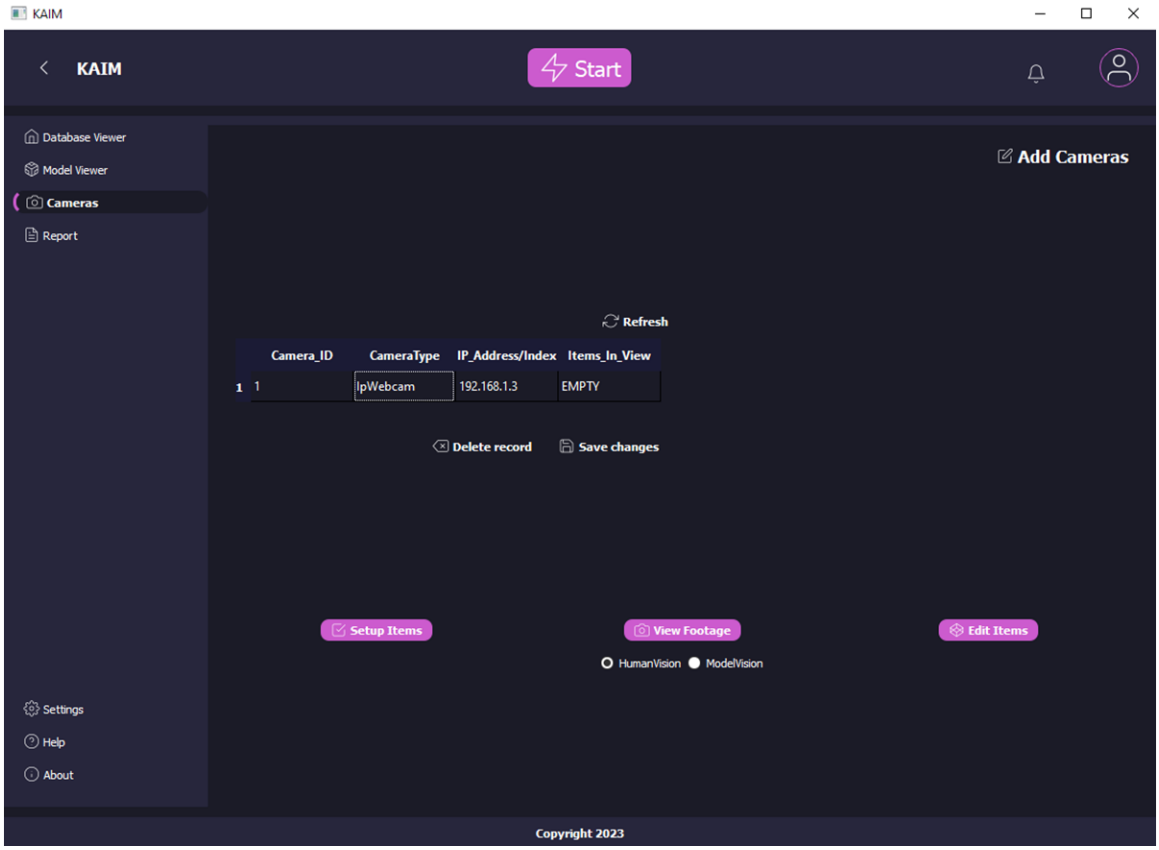


Figure 37 User selects camera

Upon selection all buttons become accessible and three initially hidden buttons are displayed:

1. Refresh
2. Delete Record
3. Save Changes

Delete record will delete the camera selection but will not update the text file.
Save changes will save the current table to the text file.
Refresh will reread the camera information from the text file and update the table accordingly.

View Footage:

View footage opens an OpenCV window displaying the camera footage. There are two modes to this that we have keenly named “Human Vision” and “Model Vision”.

Human vision will display the camera footage as is with no alteration.

Model vision will display the Camera Footage with the bounding boxes that the model has detected. This process is achieved by taking each frame our camera sends, and passing it to our model first before displaying the changes. This process takes time and thus the camera's perceived FPS will be drastically lower than normal.

Each frame will be passed to 2 models before being displayed. The first model is our own YOLO model for object detection. As for the second Model, it's a pretrained YOLO model that was trained on the COCO dataset that we mentioned earlier. However, we only care about one class that the COCO dataset contains and that's the "person" class. After both models finish their process, the frame then displays the correct annotation and boxes. The significance of the "Person" class will be discussed during the main loop section of our program.

Figure 38 View Footage (Human Vision)

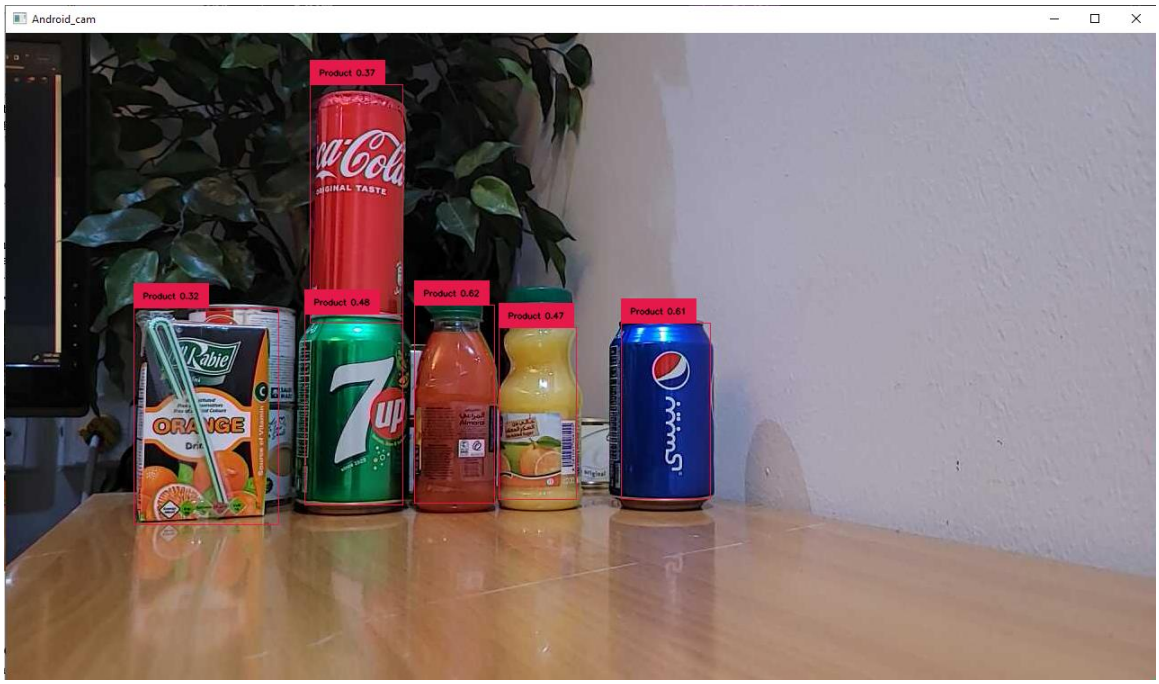


Figure 39 View Footage (Model Vision)

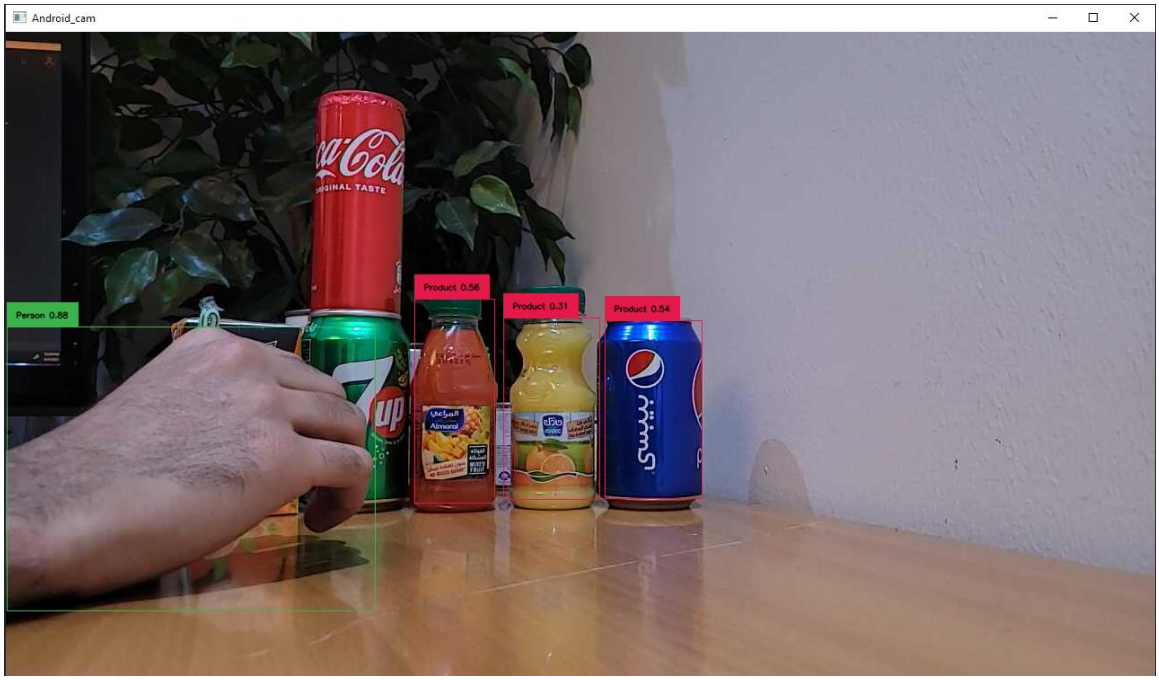


Figure 40 View Footage 2 (Model Vision)

Setup Items:

Setup items fills in the "items in view" column. Before we get in the how, let's talk about the why. The items in view column serves as a baseline, what we are essentially doing is telling our model these are the only items that should be visible to this camera. If the model detects any other items. We would consider the product detected "misplaced" and raise an alarm. This setup is only required to be done once, but the user has the option to do it as many times as he likes. Throughout this procedure a loading icon will be displayed, and user's clicks will not be registered.

Once the user selects a camera and clicks on the setup items button. The System will take the latest frame produced from the camera, and it will run that image through our 3 models. It will first run it through the pretrained YOLO model, and if a person is detected in the frame, it will cancel the process and return and error. If not. it will then run it through our YOLO model, and it will draw bounding boxes around all our products. It will then crop out each of those products and pass them to our classification model. Finally, it will populate the "items in view" column with the returned results ignoring duplicates and update the text file accordingly.

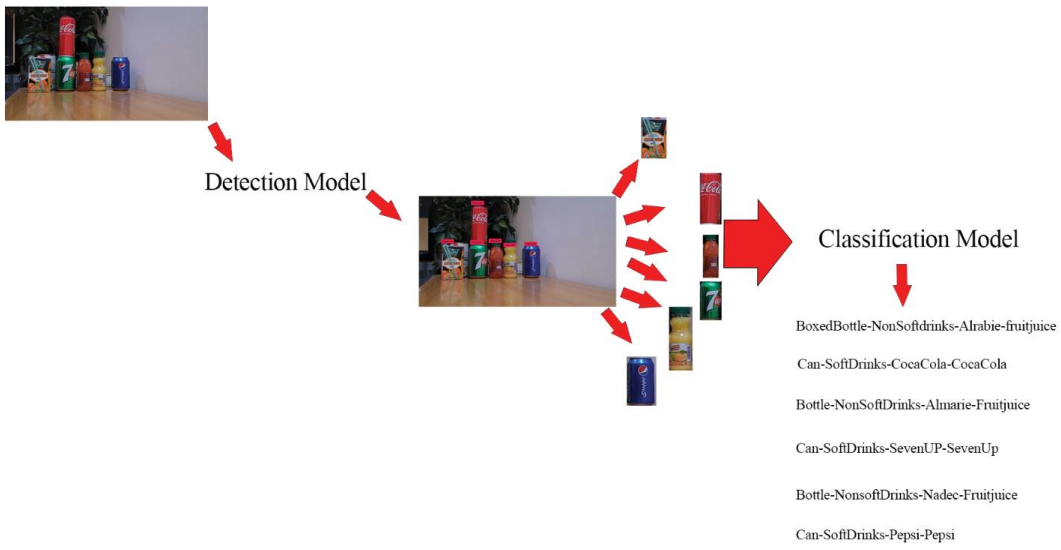


Figure 41 "Setup items" pipeline

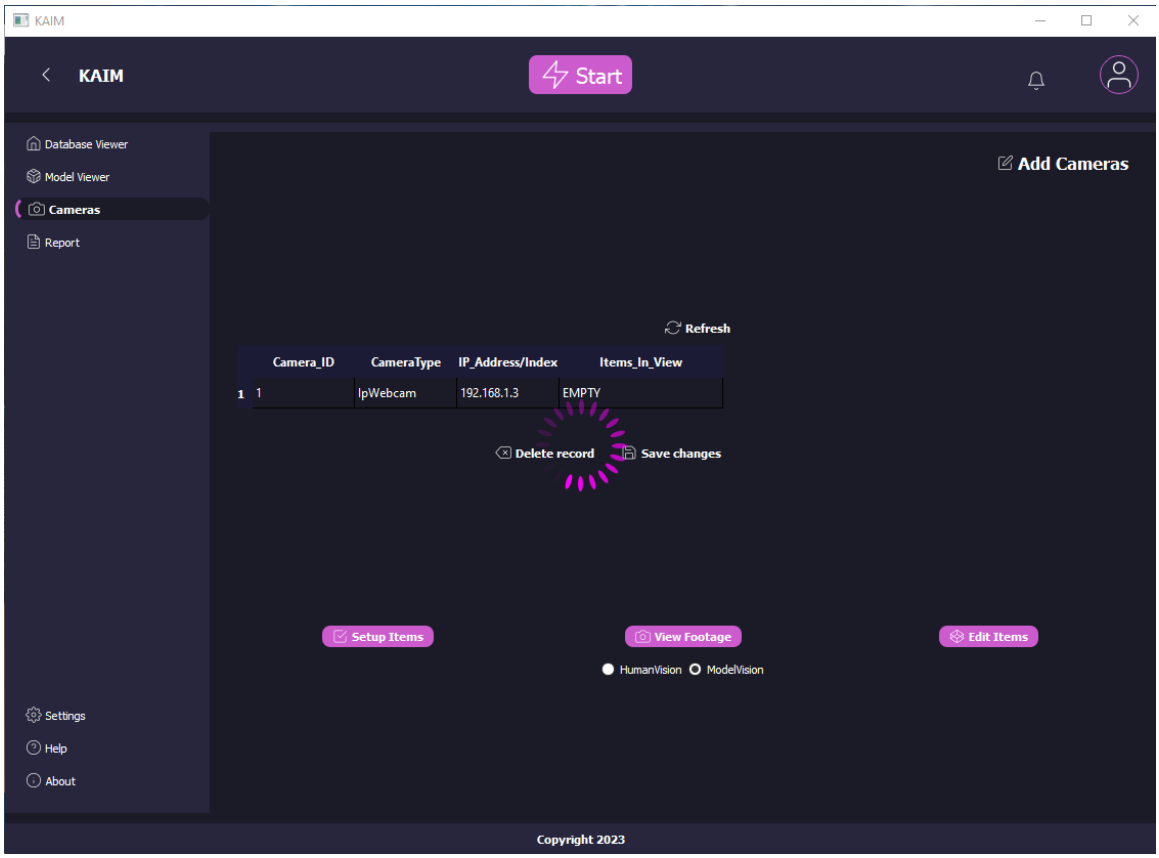


Figure 42 Setup items processing

After setting up our items this was our output.

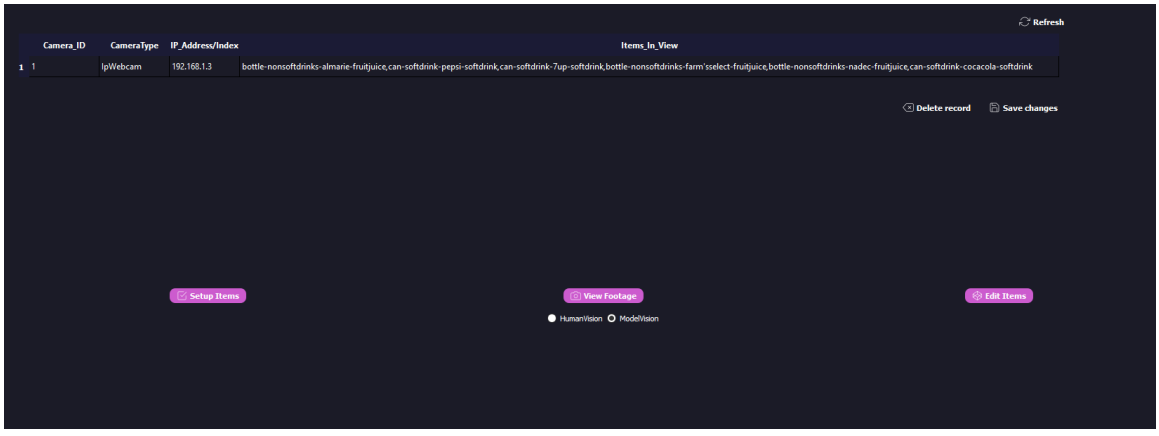


Figure 43 Setup items output

Edit Items:

It got 5 our of 6 products right and mistakenly classified Alrabie fruitjuice as farm's select fruit juice. This is okay, we can now edit our items by clicking on the edit items

button which will take us to another page. From there we can remove and add any item simply by clicking on the corresponding buttons

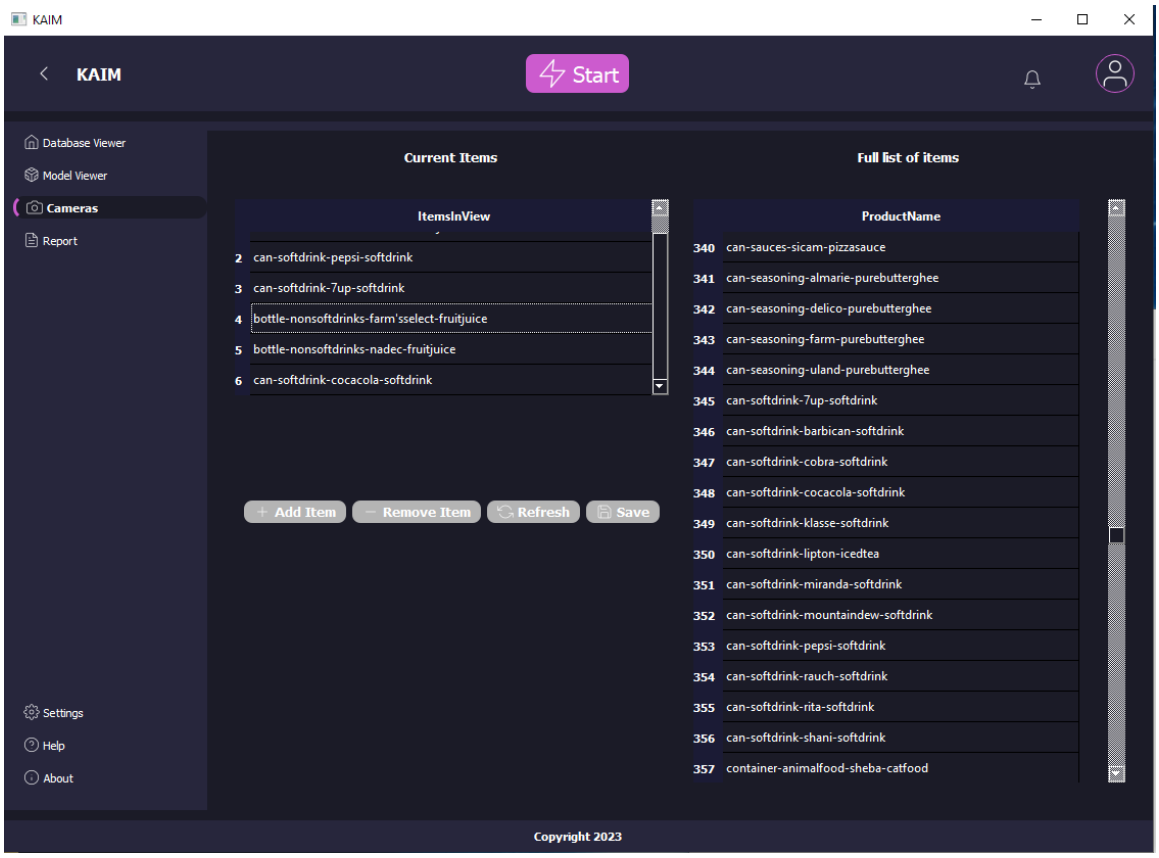


Figure 44 Edit items page

The table on the left is our current items in view for our selected camera. The table on the right contains all the possible outputs from our model. We will remove the farm's select fruitjuice from our items and replace it with alrabie fruit juice which is the correct item.

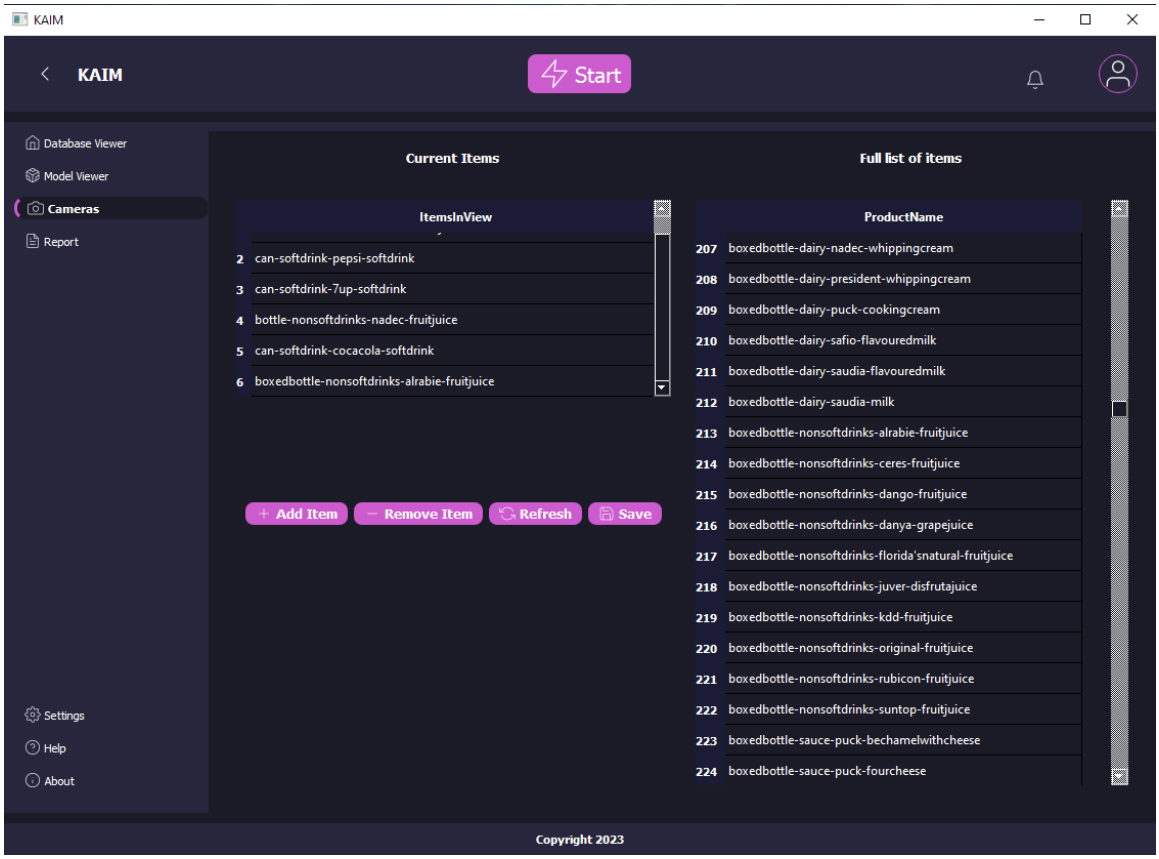


Figure 45 Changing the wrongly classified product with the right one

Finally, we can save and move on.

6.2.5.6 “Start” Program main loop:

In this part we’ll discuss the main loop of the program. After everything is setup from the database to the settings, the user can then click the “Start” button and watch the updates.

Once the user clicks “start” the system will first start by performing a series of checks in the following order:

1. Database connection test
2. Camera connection test
3. Settings are saved

If all these checks are passed, the system takes each camera runs it in a separate thread. All camera functions are run in parallel except when updating the local database table. This is done to ensure that no camera can overwrite the updates of another camera. The system will continuously test the camera’s connection during each loop to ensure that it’s still online. Since each camera runs separately, if any camera fails for any reason, no other camera will be affected.

Our system works by comparing the current camera frame with the previous camera frame. The time difference between when each frame was taken is not important to us, that’s for the manger to choose. So, the system will take the current frame and pass it first through the pretrained YOLO model, this model’s only purpose is to detect if a person is in the frame. If a person IS in the frame, then the camera will discard this frame and take another snapshot. This is done because if a person is in the frame, then that means that they are blocking the camera’s vision. And this will lead to false updates on the database. If no person is detected, then the System will pass it to our YOLO model that will draw bounding boxes around each object in its view.

The system will now look to see if there is a previous image to compare this current image with. If no image exists which means that this is it’s first iteration, it will save the current image in a queue of size 1. It will then do the same process for the next frame.

The system will then compare the frames. YOLO provides are array of coordinates of all the bounding boxes that it detects. These are the same coordinates that are used to draw the bounding boxes. The system can detect two things, if a product is taken from the frame and if a product was added to the frame. i.e. Subtraction and Addition. Below we will break each one down.

Subtraction:

The system will begin by cycling through each bounding box coordinate from the old image and see if it can find a match in the new image. If it doesn't detect a match, then the system will take those coordinates and crop them out. It will then pass the cropped image to our classification model to get the name of the product. After getting the product name it will search the lookup table (figure 30) for the relative SKU_ID. If the SKU_ID was never set and is the default 000000, the system will send a notification informing the manager that it detected a change but wasn't able to update the database. If the SKU_ID was set, the system will then search the database using the SKU_ID as a key. After finding a match, the system will navigate to the quantity column and subtract 1 from its value. The system will then continue checking all other bounding boxes until none remain. Finally, it will exit the this loop and return to square 1 to take another snapshot.

Addition:

The addition process is similar to the subtraction process. When the system is cycling through the bounding boxes of the old image and finds a match in the new image. It will mark the bounding box that it found in the new image. And after the system exits the initial "subtraction" loop, it will then cycle through the bounding boxes of the new image. If it detects a bounding box that was not marked, then that means that this bounding box is one that is not found in the old image. This means that a new product was added to its view. The system will crop the product out and pass it to the classification model. After getting the product name, the system will first check if this is a product that it's supposed to have in it's view. This is where the "items in view" column from figure 43 comes into play. The system will check the items in view column, and if the product name is not among that list, then the system will send a notification. The notification will inform the manager that a there's a misplaced product in view of this specific camera. The procedure from here on out is the same as the subtraction method with the only difference being that instead of subtracting 1 from the quantity column it will add 1.

Error range:

One last thing we haven't discussed is how exactly system looks for a bounding box. When the system searches for a match, it doesn't look for an exact 1 to 1 match. This is done for multiple reasons. Number 1, if the system were to search for an exact match, then any movement no matter how small would change the bounding box coordinates and thus, would be considered a new bounding box. And number 2, the YOLO model when

drawing bounding boxes is not 100% accurate, the slightest change in pixel value will alter the position of the bounding box by a few pixels. This can be seen clearly by viewing the camera's footage with the "model vision" option selected. Even if the camera is stable and nothing is moving the bounding boxes will still move around slightly.

To remedy this, we selected an error range that seems reasonable. The number that we settled on after testing was 50px. Whenever the system is searching for a bounding that matches the current one, it will search for those coordinates in a range of 50px. This means that 50 pixels will be added to each rectangle's side. If all corners of a bounding box fall into that range, then the system will consider that a match.

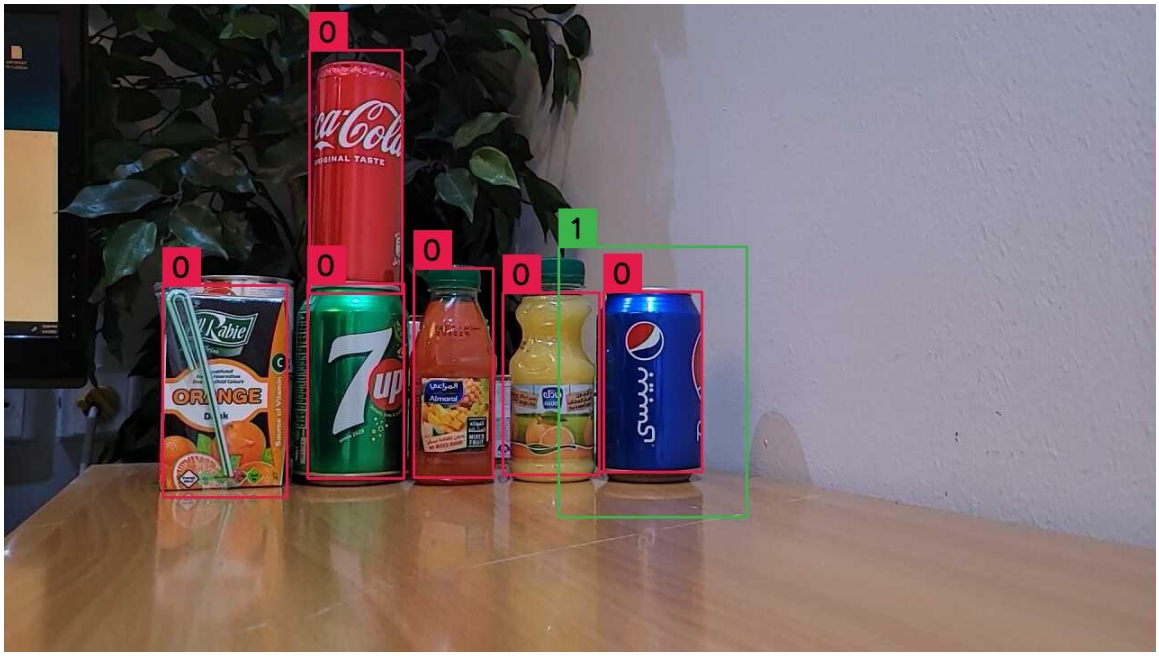


Figure 46 Error range example, red represents the actual bounding box, and the green represents the error range.

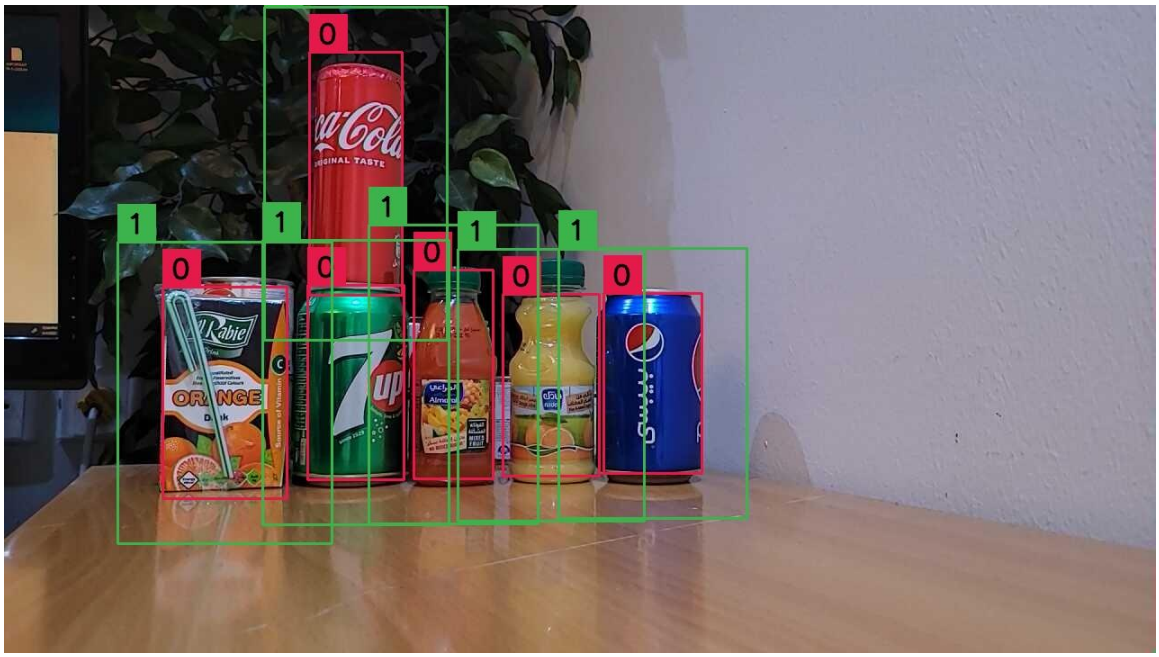


Figure 47 Error range example 2, red represents the actual bounding box, and the green represents the error range.

6.2.5.7 Notification Tab:

The notification tab is simple, it's a table that contains 3 columns labeled Notification Type, Timestamp and Alert. There are two notification types:

Missing SKUID:

This notification will be sent when the model classifies an object that has not been properly setup in the Model Viewer tab and thus the database cannot be updated. By "Not Setup" we specifically mean that the SKU_ID has been left to the default 000000.

Misplaced product:

This notification will be sent when the model classifies an object that the camera was not supposed to have in its view.

The timestamp column is self-explanatory, and the Alert column gives more information on the notification. Whenever a notification is sent a number will pop up next to the bell icon representing the unread notifications. When clicked the number will reset, disappear, and then redirect to the notifications tab. Below is an example of both these notification types in action. The clear button clears the notification list.

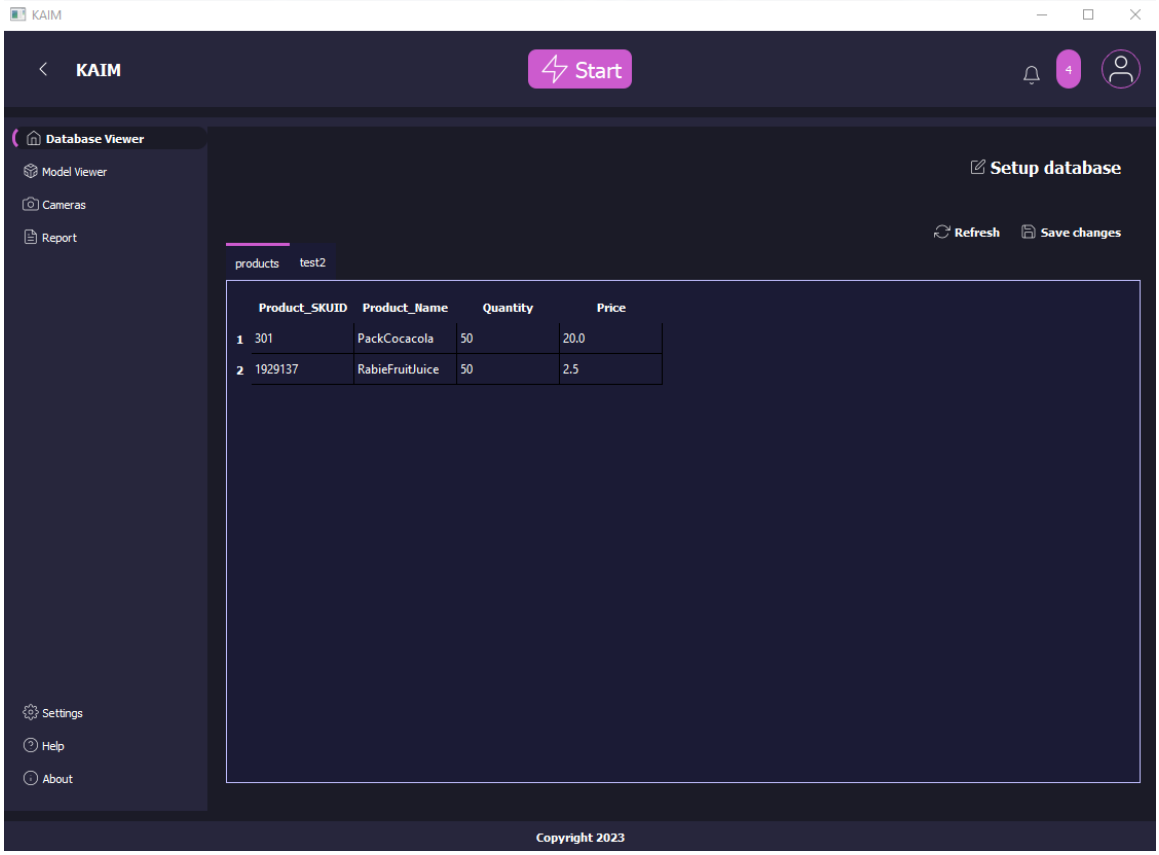


Figure 48 Notification sent after main program loop was started and stopped.

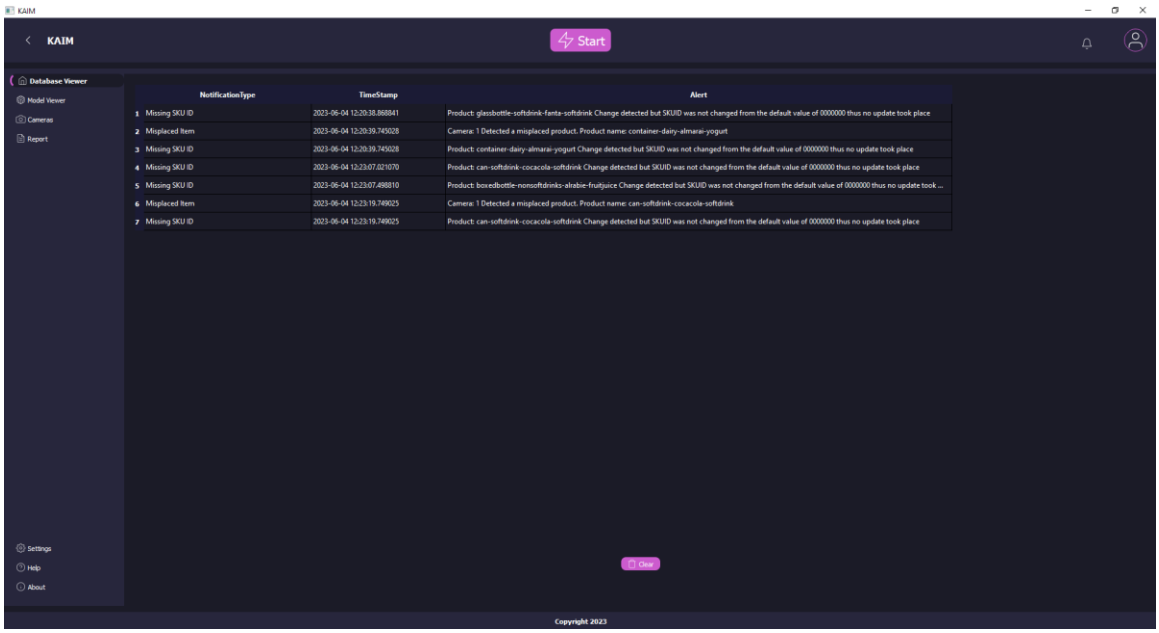


Figure 49 Notification tab in action

6.2.5.8 Report Tab:

The report tab documents all changes made by our model to our database. The tab is made up of a table with 3 columns labeled ReportType, TimeStamp and change. And also 3 buttons labeled Undo, Clear and Print.

The report type corresponds to the two types of changes our model can make, addition or subtraction. Addition as we have discussed in the main loop section, is when the model detects a new object that was not in the previous frame. Subtraction is when the model detects an object that was in the previous frame but missing the current one.

The Timestamp is again self-explanatory, and the change gives more information on the update. The Undo button gives the user the option to undo the change and reverse whatever the model did whether it's addition or subtraction.

And finally, the print button takes the table and converts to an easily readable pdf file.

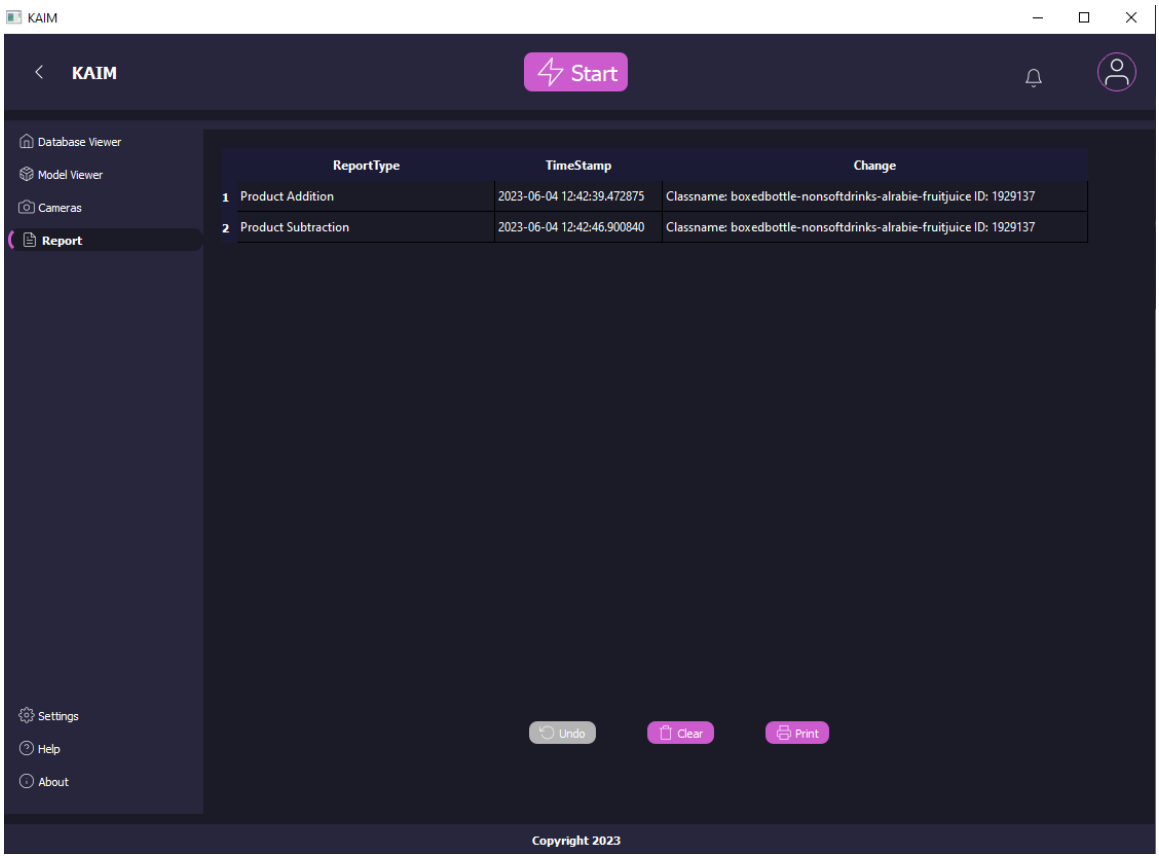


Figure 50 Report tab

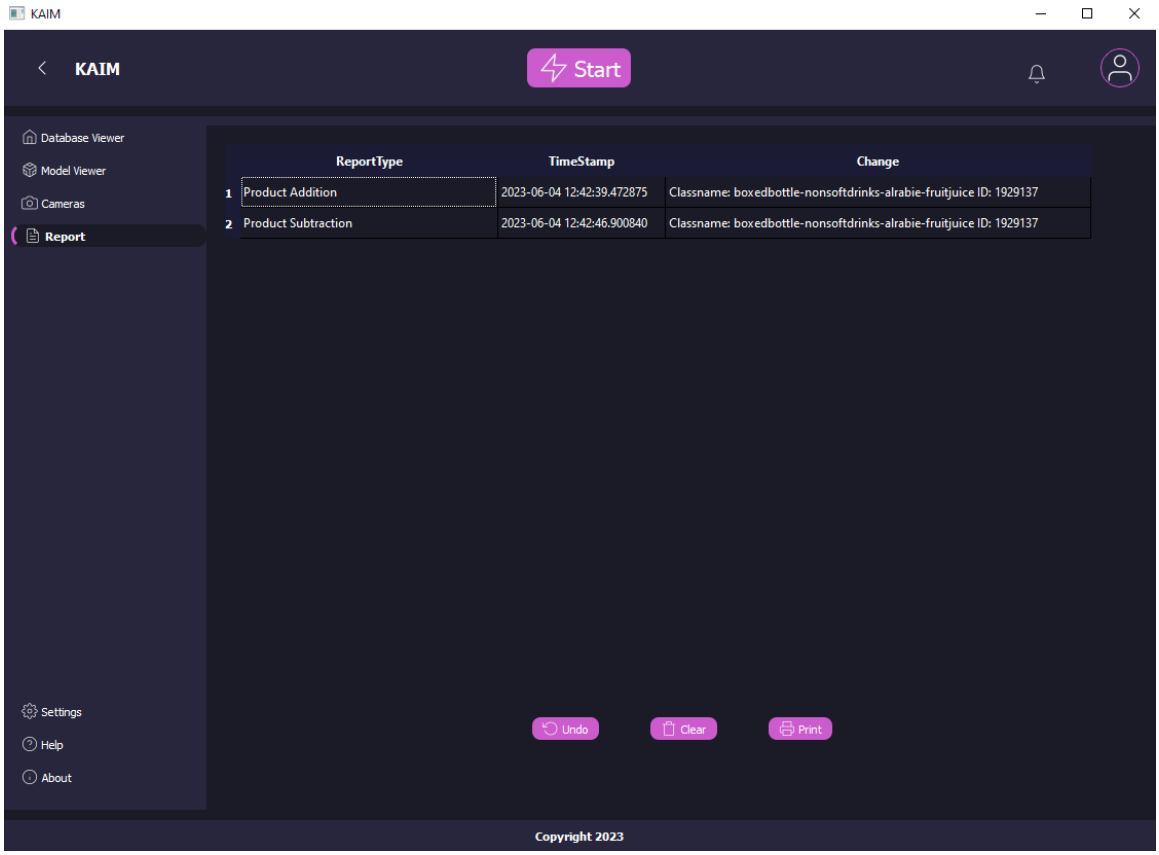


Figure 51 Report tab Undo Addition

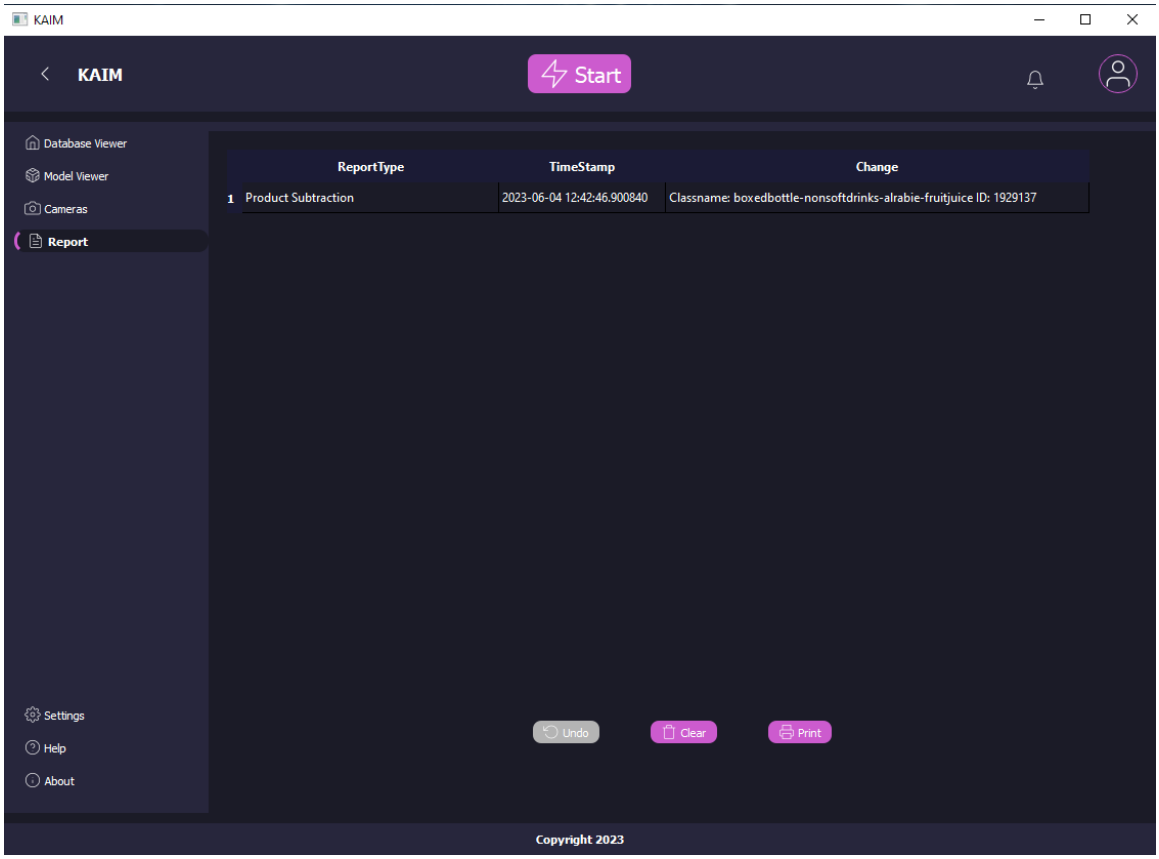


Figure 52 After Undoing addition.

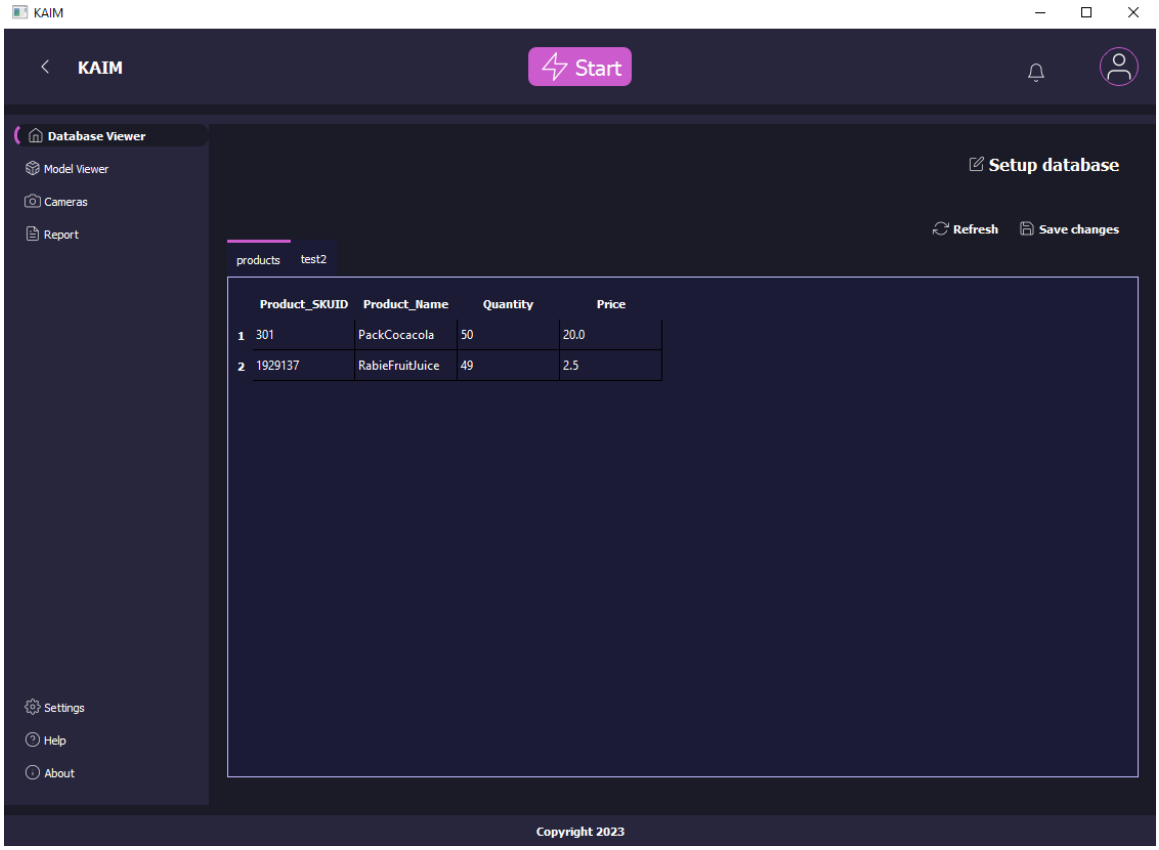


Figure 53 Database after Undoing Alrabie addition (Initially 50)

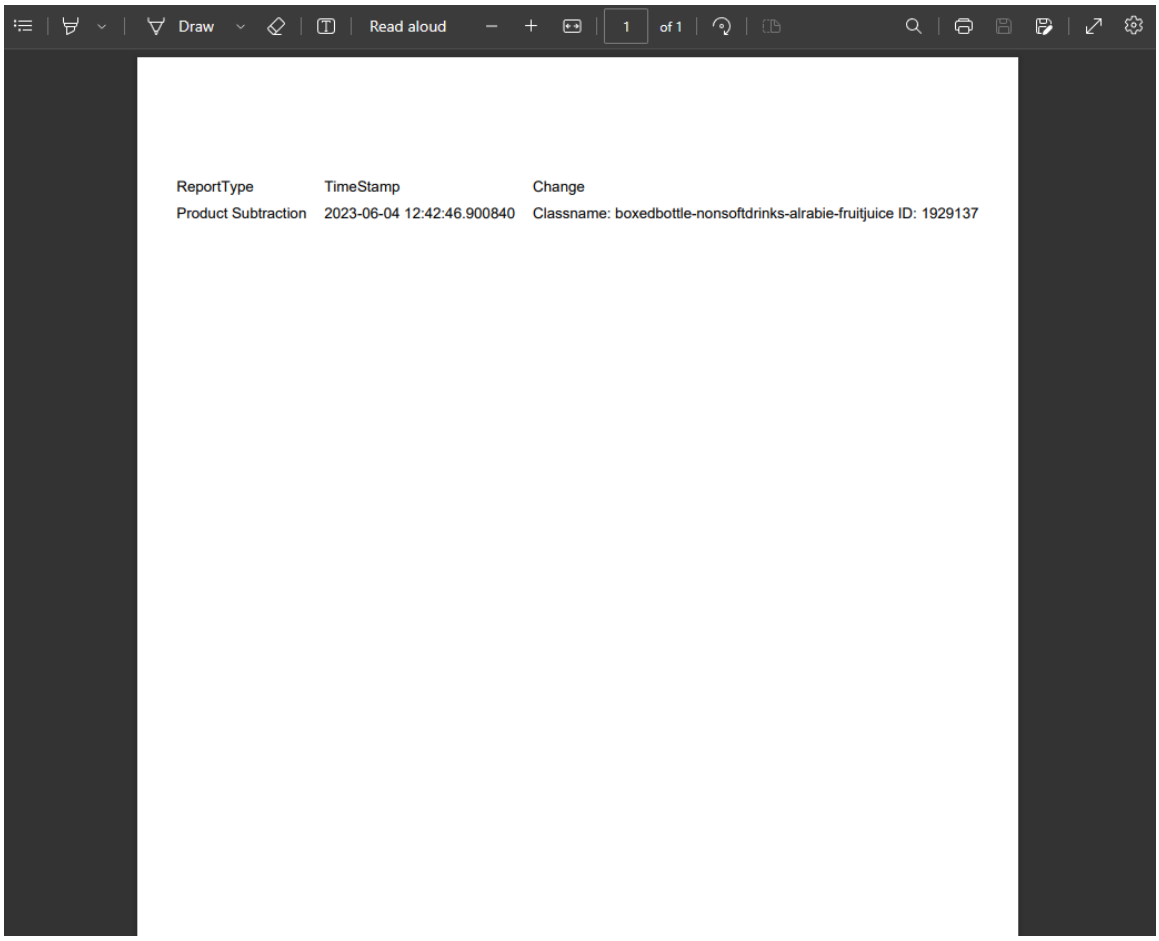


Figure 54 Printing Report

6.2.5.9 Help Tab:

The help tab will link to a document that contains information on all the system's inner workings. This document can be found here:

https://drive.google.com/file/d/1t6BCu1YojVjJiXCIupiUBdNybxlm7R6N/view

6.2.5.10 About:

The About tab will link to a document that contains all our contact information. This document can be found here:

https://drive.google.com/file/d/1PnlIUhGZILnpEbOCB52onB0AB6TheFRA/view

6.3 System limitations:

In this section we will discuss the biggest weakness in our system and our attempt to fix it.

The biggest weakness in our system is that it only works when products are limited to 1 row. If you put two products back-to-back and remove the product in the first row, the model won't be able to detect the change. We attempted to solve this problem by making the error range much smaller and utilizing both distance and area in our calculations.

Lets take an example: Lets assume we take a product and place it directly facing the camera at a distance of 1 meter. And then have our YOLO model draw a bounding box around it. If we then take the same product and push it background so that it's 2 meters away from the camera but on the same angle as it was before. The YOLO model will draw a smaller bounding box around that product. From this example the model can now understand the concept of depth. If two products are placed back to back and it detects them on two different snapshots, then the model can understand that the one with the smaller bounding box (area wise) is farther away and thus was initially behind the first product it detected.

Now let's apply the above example to our system. The system takes a snapshot and detects 2 products. It then takes another snapshot and begins comparing. The comparison is done in three steps:

Firstly, is this bounding box located in the current image within the confines of our range? If not, then that means that the product was taken, and no other product has taken its place. I.e., there is no other product behind it. This is a best-case scenario. What if it does detect another product in that range? The system in this case can't just assume that a match was found, because we introduced an extra layer complexity by adding another row of products. The system will have to do some extra computations and now we move to step 2.

Secondly, how far are the two bounding boxes from each other? Since the bounding boxes are drawn in a 2D format we can calculate how far one box is from another by summing the Euclidean distance between each corner and its counterpart. Afterwards if the distance is larger than a certain threshold, we can assume that the product we are dealing with in the current image is not the same as the one in the previous image. And then we move to our third and final step.

Thirdly, how big is the difference in area between both bounding boxes? If the bounding box of the product in the previous image is larger by a certain threshold after performing all the previous steps. Then we can assume that the product in the current image is not the same as the one in the previous image. And since the bounding box of the product in the current image is smaller, then we can assume that the current product was behind the initial product. In this scenario the system would assume that the product

has been taken. The system would then move to classify the product in the first image and update the database by Subtraction. If the reverse happens and the bounding box of the product in the current image is larger than the one in the previous image, the process would be addition.

This approach was tested extensively however the results were never consistent. As we have stated before YOLO is sensitive to the smallest change in pixel value. So even if everything is stable and the frame is still, the bounding boxes drawn would not be the same from one frame to another. This inconsistency was the downfall this approach. We had to find threshold values that were large enough to cover YOLO's bounding box inconsistencies while also being small enough to detect real changes. And we weren't able to find those threshold values. During the test implementation of this approach, the system would raise false flags at a fast rate. Eventually we were forced to abandon our approach and settle for a simpler more reliable one.

Chapter 7: System Testing **

7.1 Unit testing

When it comes to unit testing, a python built in module by the name of unittest was utilized. we created 14 test cases, where each case calls the assertEqual (actual results, expected results) to check for the expected result or assertTrue / assertFalse to check for a condition. All cases were passed in 5 under seconds.

7.2 Integration and regression testing

We realized that simply combining the two models will result in a huge waste of computing power, since the output of the first model might be fifty images for the second model.

So, we utilized our predefined error range to determine if a product has changed. If a change is detected, then that image will be passed to the next model. All the test cases were done using unittest and were passed in under 10 seconds.

7.3 Performance and stress testing

YOLO (First Model):

The performance of object detection models can't be represented by common metrics like accuracy, so we need to define some definitions before we measure the performance of the model.

Intersection Over Union: Intersection Over Union (IOU) is area of overlap of the bounding boxes over the union of the boxes.

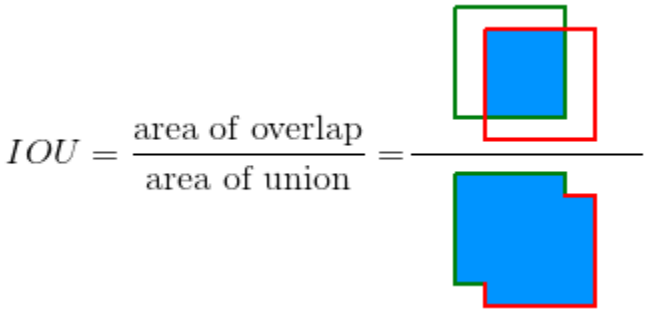


Figure 55 Illustration of the intersection over union (IOU) [34]

True Positive: True Positive (TP) is correct detection (IOU higher or equal to selected threshold).
False Positive: False Positive (FP) is wrong detection (IOU less than selected threshold).
False Negative: False Negative (NP) is ground truth not detected.
Precision: Precision is the percentage of correct predictions to all predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\text{all detections}}$$

Figure 56 Precision Formula [34]

Recall: Recall is the percentage of correct predictions to all ground truths.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{all ground truths}}$$

Figure 57 Recall Formula [34]

Average Precision: Average Precision (AP) is the area under the precision recall curve.
Mean Average Precision: Mean Average Precision (mAP) is the mean of average precision of all classes.
Results: The model achieved high results (see figure 18), the metrics we tested our model on are seen in the table below.

Model	Precision	Recall	mAP50	mAP50-95
YOLO	0.87428	0.77996	0.85641	0.518

Table 2 YOLOV8 Metrics.

RESNET (Second Model)

While accuracy can be misleading since not all classes are equal, in our case we believed that it is the most important metric because our data has come from the real

world, and with augmentation the model can simulate some extreme conditions (see figure 20). The model has reached accuracy of 96.738%.

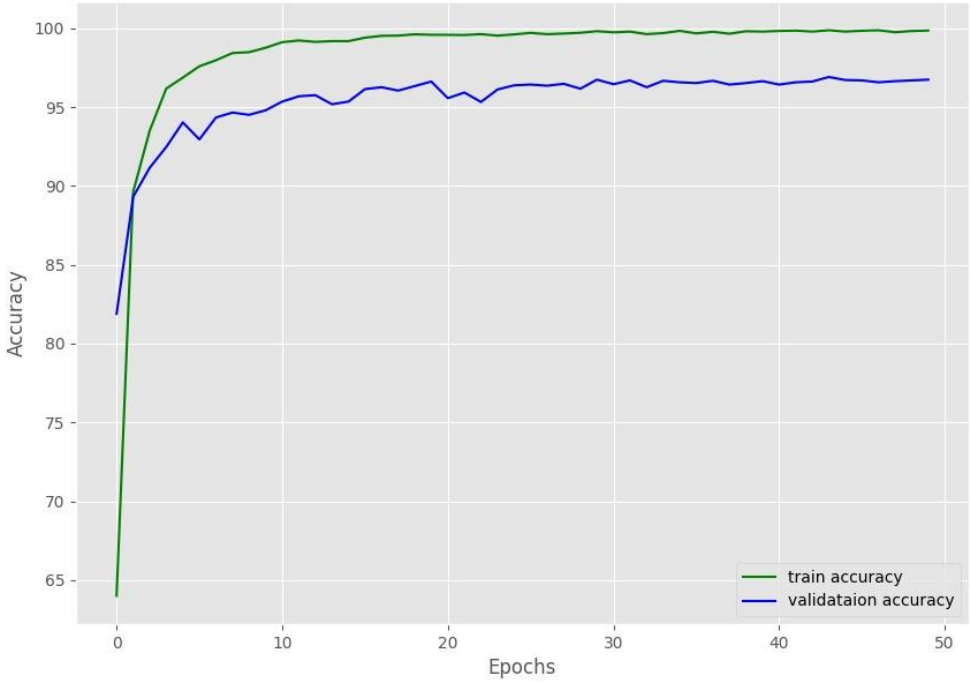


Figure 58 Illustration of Accuracy over Epochs

Main program stress test:

In this section, we will be testing the amount of resources our main system loop takes. All these tests were run on an Intel Core i5-8600k 3.60GHz CPU. In each test we will add more cameras and see how our system reacts. All readings were taken from the task manager with the 5 values in question representing the following things.

1- CPU usage
2- Memory usage
3- Disk usage
4- Network usage
5- GPU usage

1 camera:

Figure 59 Stress test on 1 camera

2 cameras:

Figure 60 Stress test on 2 cameras

3 cameras:

Figure 61 Stress test on 3 cameras

6 cameras:

Figure 62 Stress test on 6 cameras

It's clear from this test that our program is CPU demanding. We thought about using a CUDA approach to run the cameras. This would greatly increase the amounts of cameras that can run in parallel. However, that would limit the user base to only Nvidia GPU users, so we opted not to follow this approach.

7.4 User acceptance testing

Our system was built with user experience in mind. We aimed to make our system as "plug in and play" as possible. There are some inevitable setups that the user of our system must perform. However, they are few and are only required to be done once.

Our system provides a user-friendly interface that is easy to get around. Our system also provides extensive documentation which comes embedded in the program, namely the "help" tab. Our system also can connect to any cloud hosted database by passing the system the 5 lines of required information. From there the user can simply choose the columns that are relevant to our model. Then, by adding the camera(s), the user can instruct the system to begin analyzing the footage by a click of a button. The system also takes into consideration a lot of the user's actions that might "break" the system. These were tested and dealt with accordingly by displaying an error message instead of crashing the program.

No matter what, it's difficult to give a 100% unbiased opinion. After all this is OUR system. However, the fact is that a lot of effort went into the UI and overall user experience, and we believe these changes show.

7.5 Test cases

Test Case#1:

Test Description: Login successfully.

Test Sequence: 1-The user opens the program, 2-Enter his username and password, 3-Press verify.

Test Data: Username, and password.

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: The login page is changed to the main page.

Actual Results: As expected.

Test Status (Passed/Failed): Pass.

Test Case#2:

Test Description: Login with wrong credentials.

Test Sequence: 1-The user opens the program, 2-Enters a username and password, 3-Press verify.

Test Data: Username, and password.

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: Error message “Wrong Username or Password”.

Actual Results: As expected.

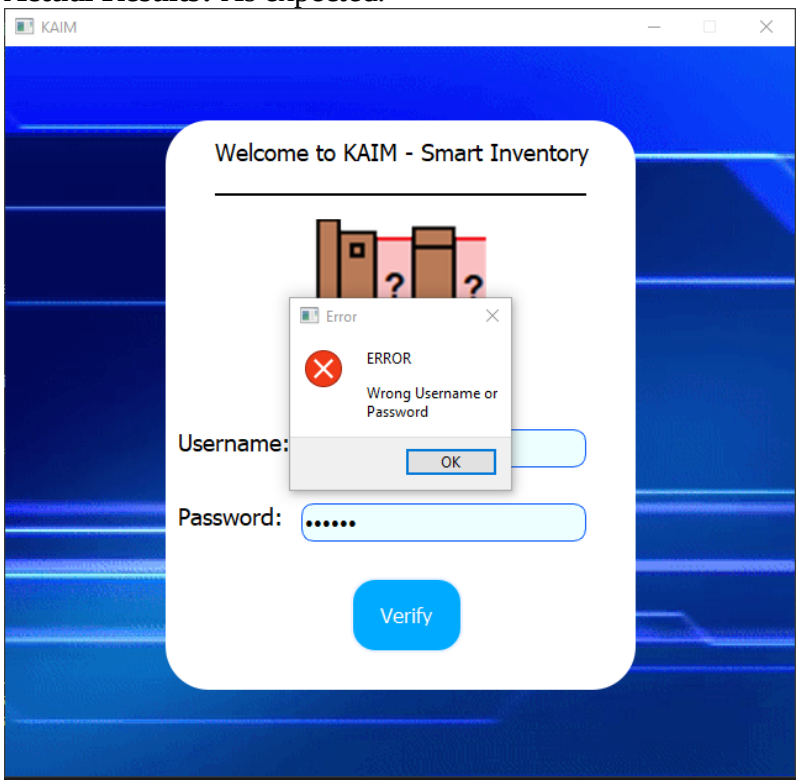


Figure 63 Test case 2

Test Status (Passed/Failed): Pass.

Test Case#3:

Test Description: Connect to database successfully.

Test Sequence: 1-From the program main page enter the user click setup database, 2-Enter his database: server IP, port, username and password, 3-Press setup.

Test Data: Server IP, port, username, and password.

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: The system is connected to the database and database products are displayed.

Actual Results: As expected.

Test Status (Passed/Failed): Pass.

Test Case#4:

Test Description: Connect to database with incorrect input.

Test Sequence: 1-From the program main page enter the user click setup database, 2-Enter a database: server IP, port, username and password, 3-Press setup.

Test Data: Server IP, port, username, and password.

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: Error message "Failed to setup Database. Make sure the connection is open".

Actual Results: As expected.

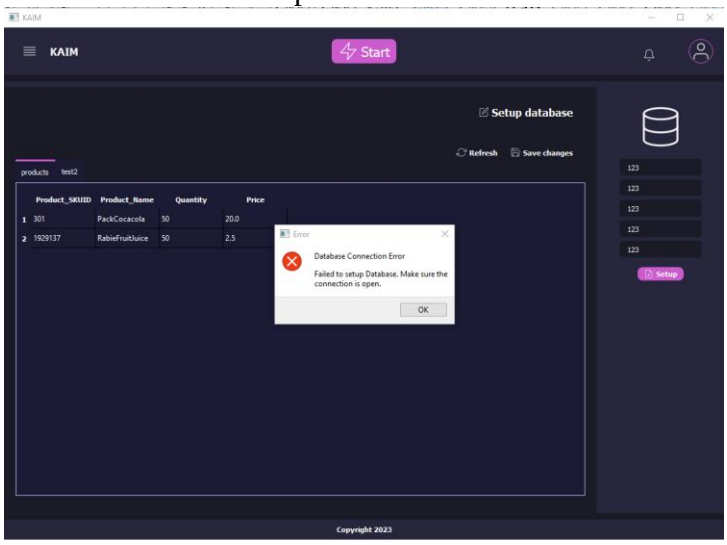


Figure 64 Test case 4

Test Status (Passed/Failed): Pass.

Test Case#5:

Test Description: Refresh the database.

Test Sequence: 1-From the program main page enter the user click refresh button.

Test Data: None

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: Update product display for changes.

Actual Results: As expected.

Test Status (Passed/Failed): Pass.

Test Case#6:

Test Description: Generate report successfully.

Test Sequence: 1-The user clicks on the report section from the left, 2-press print, 3-save PDF.

Test Data: -

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: The report is generated as a pdf file.

Actual Results: As expected.

Test Status (Passed/Failed): Pass.

Test Case#7:

Test Description: Clear changes history successfully.

Test Sequence: 1-The user clicks on the report section from the left, 2-presses clear.

Test Data: -

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: The changes are wiped.

Actual Results: As expected.

Test Status (Passed/Failed): Pass.

Test Case#8:

Test Description: Undo clear changes-history successfully.

Test Sequence: 1-The user clicks on the report section from the left, 2-presses clear, 3-presses undo.

Test Data: -

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: The changes are restored.

Actual Results: As expected.

Test Status (Passed/Failed): Pass.

Test Case#9:

Test Description: Camera Connection.

Test Sequence: 1-The user clicks on add camera, 2- The user chooses either a local camera, or an IP camera, 3- In the case of a local camera the user selects an index, and on the latter case the user enters the Webcams IP. 4- The user clicks on Add.
Test Data: IP or index.

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: The Camera record appears and is added.

Actual Results: As expected.

Test Status (Passed/Failed): Pass

Test Case#10:

Test Description: Starting the model without connecting to a database.

Test Sequence: 1-User open program, 2-Presses on Start button.

Test Data: -

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: Error message “Database Error”.

Actual Results: As expected.

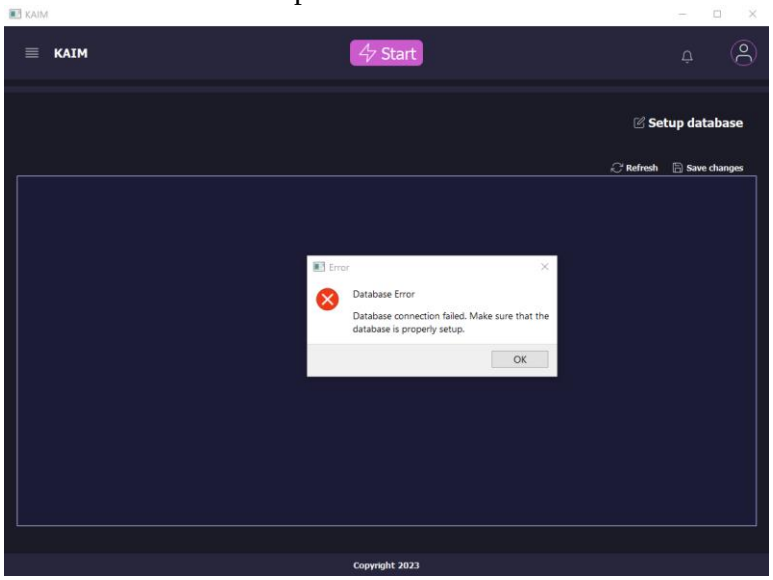


Figure 65 Test case 10

Test Status (Passed/Failed): Pass

Test Case#11:

Test Description: Starting the model after setting up the database but not settings.

Test Sequence: 1-User opens program, 2-User clicks on “setup database”, 3-User enters database information. 4-Presses on the Start button.

Test Data: Server IP, port, username, and password

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: Error message “Settings Error”.

Actual Results: As expected.

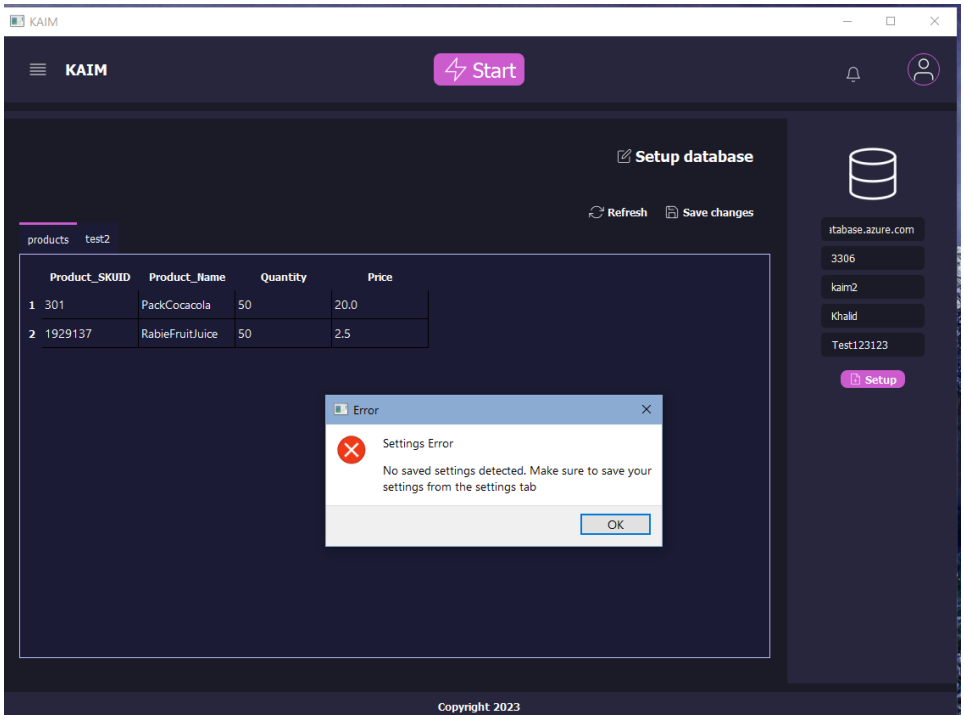


Figure 66 Test case 11

Test Status (Passed/Failed): Passed

Test Case#12:

Test Description: Starting the model after setting up the database and settings without camera setup.

Test Sequence: 1-User opens program, 2-User clicks on “setup database”, 3-User enters database information. 4-User opens settings tab, 5-User save changes, 4-Presses on the Start button.

Test Data: Server IP, port, username, and password

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: Error message “Camera Error”.

Actual Results: As expected.

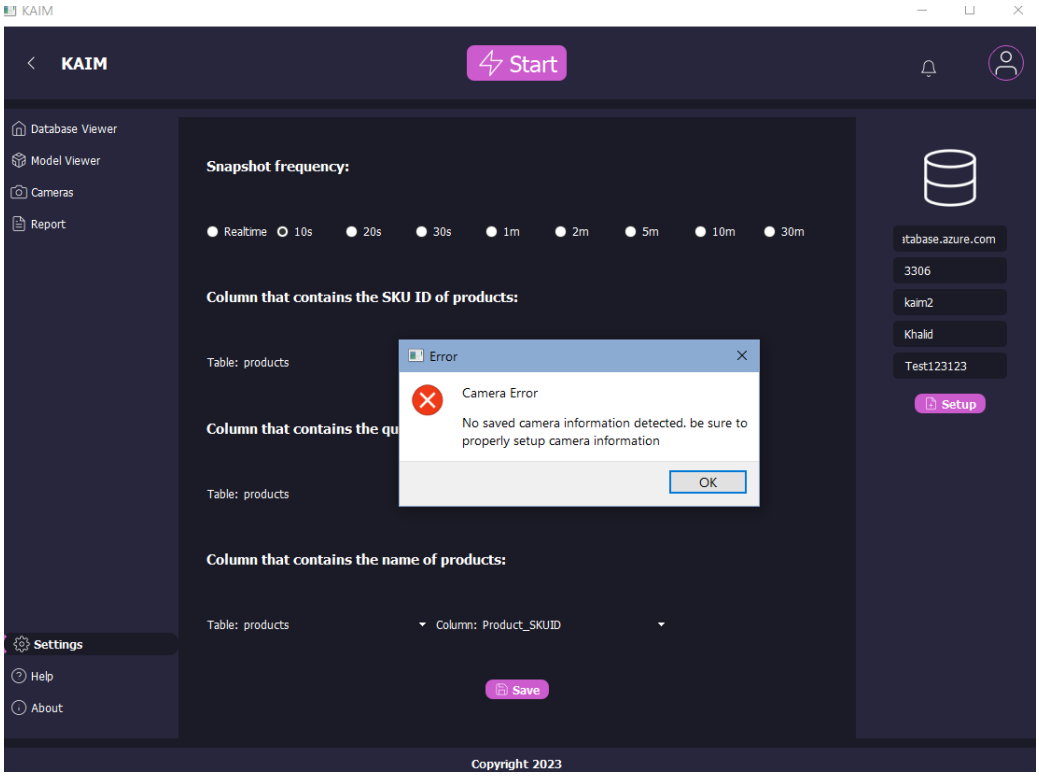


Figure 67 Test case 12

Test Status (Passed/Failed): Passed

Test Case#13:

Test Description: Starting the model after setting up the database and settings and connecting a camera with invalid IP.

Test Sequence: 1-User opens program, 2-User clicks on “setup database”, 3-User enters database information. 4-User opens settings tab, 5-User save changes, 6-User opens cameras tab, 7-User enters camera info “webcam index”, 8-Presses on the Start button.

Test Data: Server IP, port, username, and password, cam IP = 1.

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: Error message “Camera Error, Camera with IP 1 was never setup”.

Actual Results: As expected.

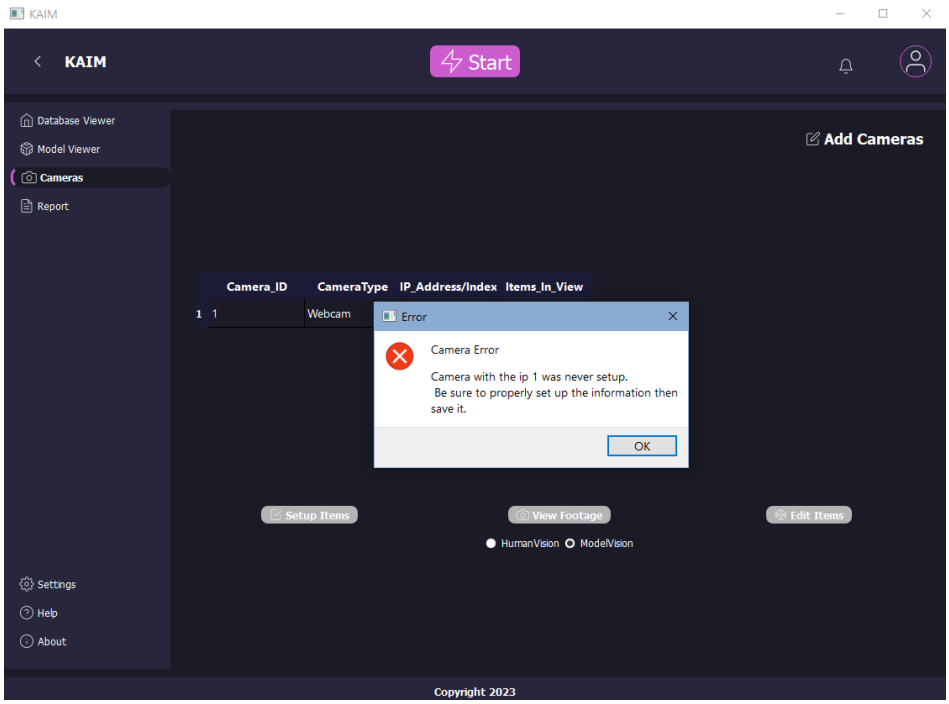


Figure 68 Test case 13

Test Status (Passed/Failed): Passed

Test Case#14:

Test Description: Printing a report with no records.

Test Sequence: 1-User navigates to the Report section, 2- User clicks on Print

Test Data: -

Testing Environment: Windows 11 22H2, 64-bit-based processor.

Expected Results: Error message “at least 1 record has to be present to print the report”

Actual Results: As expected.

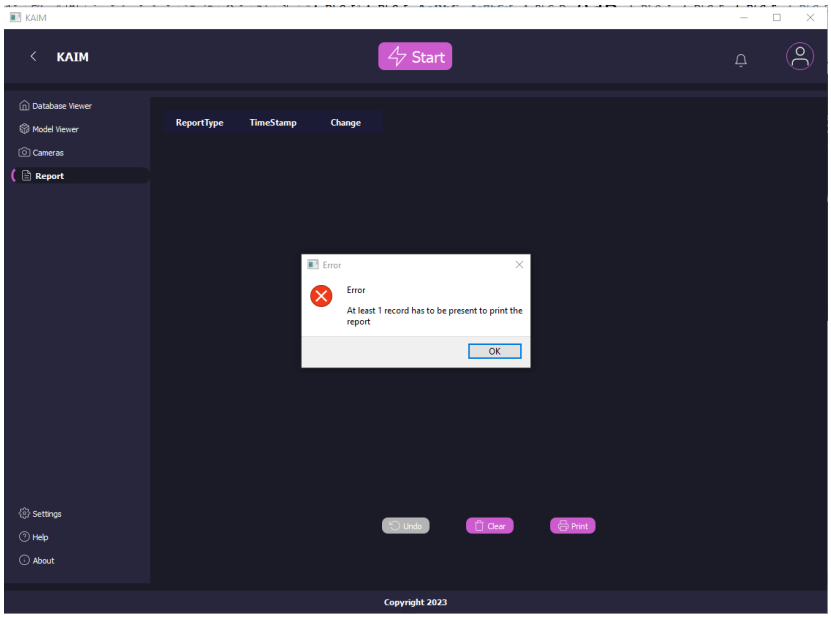


Figure 69 Test case 14

Test Status (Passed/Failed): Pass.

Chapter 8: Conclusion

Retail smart inventory management system depends on powerful deep learning algorithms. Using the latest and most efficient tools and algorithms to perform stocktaking with no human intervention, 24/7 stocktaking, and a cost-effective solution.

The system will have an easy-to-use interface. Where an administrator can edit and monitor the system. In addition to the cameras, you can access the database and request a changes report at any time.

The field of applying deep learning to retail store management is in continuous evolution, with a huge margin for improvement. We hope to contribute to that evolution and put our stamp on it.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[2] "How neural network models in machine learning work?," *How neural network models in Machine Learning work?*, 02-May-2022. [Online]. Available: https://www.turing.com/kb/how-neural-network-models-in-machine-learning-work. [Accessed: 17-Oct-2022].

[3] L. Deng and D. Yu, in *Deep learning: Methods and applications*, Boston: now Publishers Inc., 2014, pp. 9–10.

[4] Q. Lv, S. Zhang, and Y. Wang, "Deep learning model of image classification using machine learning," *Advances in Multimedia*, vol. 2022, p. 9, 2022.

[5] J. Jeong, T. Yoon, and J. Park, "Towards a meaningful 3D map using a 3D lidar and a camera," Sensors, vol. 18, no. 8, p. 2571, 2018.

[6] Hmrishav Bandyopadhyay, "An Introduction to Image Segmentation: Deep Learning vs. Traditional" 2022.

[7] T. Sriram, K. V. Rao, S. Biswas, and B. Ahmed, "Applications of barcode technology in automated storage and retrieval systems," in Proceedings of the 1996 22nd International Conference on Industrial Electronics, Control, and Instrumentation, vol. 1, pp. 641–646, Taipei, Taiwan, 1996.

[8] H. Poll, "Digimarc survey: 88 percent of U.S. adults want their retail checkout experience to be faster," 2015, https://www. digimarc.com/about/news-events/press-releases/2015/07/21/ digimarc-survey-88-percent-of-u.s.-adults-want-their-retailcheckout-experience-to-be-faster.

[9] R. Want, "An introduction to RFID technology," IEEE Pervasive Computing, vol. 5, no. 1, pp. 25–33, 2006.

[10] Hampshire, "AI spending by retailers to reach $12 billion by 2023, driven by the promise of improved margins," April 2019, https://www.juniperresearch.com/press/ press-releases/ai-spending-by-retailers-reach-12-billion2023.

[11] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, vol. 18, no. 7, pp. 1527–1554, 2006.

[12] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science, vol. 313, no. 5786, pp. 504–507, 2006.

[13] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," Advances in Neural Information Processing Systems, pp. 153–160, MIT Press, Cambridge, MA, USA, 2007.

[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012, https:// arxiv.org/abs/1207.0580.

[15] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch: tensors and dynamic neural networks in python with strong GPU acceleration," 2017, https://pytorch.org/.

[16] M. Abadi, A. Agarwal, P. Barham et al., "TensorFlow: largescale machine learning on heterogeneous distributed systems," 2016, https://arxiv.org/abs/1603.04467.

[17] Y. Jia, E. Shelhamer, J. Donahue et al., "CAFFE: convolutional architecture for fast feature embedding," in Proceedings of the 22nd ACM International Conference on Multimedia, Glasgow, UK, 2014

[18] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," >e Journal of Physiology, vol. 160, no. 1, pp. 106–154, 1962.

[19] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, and others, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[20] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 2015.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, https:// arxiv.org/abs/1409.1556.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 2016.

[23] Z. Gao, Y. Li, and S. Wan, "Exploring deep learning for viewbased 3D model retrieval," ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 16, no. 1, pp. 1–21, 2020

[24] *Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi*; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788

[25] A. Tonioni, E. Serra, and L. Di Stefano, "A Deep Learning Pipeline for product recognition on store shelves," 2018 IEEE International Conference on Image Processing, Applications and Systems (IPAS), 2018.

[26] X. S. Wei, Q. Cui, L. Yang, P. Wang, and L. Liu, “RPC: alarge-scale retail product checkout dataset,” 2019.

[27] Cai Y, Wen L, Zhang L, Du D, Wang W. “Rethinking object detection in retail stores. In Proceedings of the AAAI Conference on Artificial Intelligence 2021 (Vol. 35, No. 2, pp. 947-954).

[28] E. Goldman, R. Herzig, A. Eisenschtat, J. Goldberger, and T. Hassner, *Precise Detection in Densely Packed Scenes*. 2019. doi: 10.1109/cvpr.2019.00537.

[29] T.-Y. Lin *et al.*, “Microsoft COCO: Common Objects in Context,” in *Lecture Notes in Computer Science*, Springer Science+Business Media, 2014, pp. 740–755. doi: 10.1007/978-3-319-10602-1_48.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *ImageNet: A large-scale hierarchical image database*. 2009. doi: 10.1109/cvpr.2009.5206848.

[31] J. Peng, C. Xiao, Y. Li, *RP2K: A Large-Scale Retail Product Dataset for Fine-Grained Image Classification*. 2020. doi: 10.48550/cvpr. 2020.12634

[32] G. Jocher, A. Chaurasia, & J. Qiu, (2023). YOLO by Ultralytics (Version 8.0.0) [Computer software]. https://github.com/ultralytics/ultralytics

[33] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*. 2016. doi: 10.1109/cvpr.2016.90.

[34] R. Padilla, W. L. Passos, T. L. Dias, S. L. Netto, and E. A. da Silva, “A comparative analysis of Object Detection Metrics with a companion open-source toolkit,” *Electronics*, vol. 10, no. 3, p. 279, 2021. doi:10.3390/electronics10030279