

University of Louisville

ThinkIR: The University of Louisville's Institutional Repository

Electronic Theses and Dissertations

8-2020

Inventory management of the refrigerator's produce bins using classification algorithms and hand analysis.

Sarah Virginia Morris
University of Louisville

Follow this and additional works at: <https://ir.library.louisville.edu/etd>



Part of the [Other Electrical and Computer Engineering Commons](#)

Recommended Citation

Morris, Sarah Virginia, "Inventory management of the refrigerator's produce bins using classification algorithms and hand analysis." (2020). *Electronic Theses and Dissertations*. Paper 3497.
Retrieved from <https://ir.library.louisville.edu/etd/3497>

This Master's Thesis is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.

INVENTORY MANAGEMENT OF THE REFRIGERATOR'S PRODUCE BINS
USING CLASSIFICATION ALGORITHMS AND HAND ANALYSIS

By

Sarah Virginia Morris
B.S., University of Louisville, 2017

A Thesis
Submitted to the Faculty of the
University of Louisville
J.B. Speed School of Engineering
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science
in Electrical Engineering

Department of Electrical Engineering
University of Louisville
Louisville, KY

August 2020

INVENTORY MANAGEMENT OF THE REFRIGERATOR'S PRODUCE BINS
USING CLASSIFICATION ALGORITHMS AND HAND ANALYSIS

By

Sarah Virginia Morris
B.S., University of Louisville, 2017

A Thesis Approved on

July 27, 2020

by the following Thesis Committee:

Dr. Karla Welch, Thesis Chair

Dr. Olfa Nasraoui, Committee Member

Dr. Jacek Zurada, Committee Member

DEDICATION

I dedicate this thesis to all my family and friends for their love and support throughout this process.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Welch, for her guidance, advice, and support during this past year. I would also like to thank Dr. Nasraoui and Dr. Zurada for their support and agreeing to serve on the committee.

Special thanks to Michael Schroeder, John Ouseph, and Stephanos Kyriacou. Your collective knowledge guided me throughout this thesis.

I would like to thank the staff of GE Appliances, my manager Tim O'Connell, Dr. Kelecy, and the rest of the staff who provided infinite opportunities to continuously learn and grow. Also, thank you for allowing me to conduct this research in your facilities and providing all the resources requested.

Lastly, to Devin, thank you for your support, wisdom, humor, and love.

ABSTRACT

INVENTORY MANAGEMENT OF THE REFRIGERATOR'S PRODUCE BINS USING CLASSIFICATION ALGORITHMS AND HAND ANALYSIS

Sarah Morris

July 27, 2020

Tracking the inventory of one's refrigerator has been a mission for consumers since the advent of the refrigerator. With the improvement of computer vision capabilities, automatic inventory systems are within reach. One inventory area with many potential benefits is the fresh food produce bins. The bins are a unique storage area due to their deep size. A user cannot easily see what is in the bins without opening the drawer. Produce items are also some of the quickest foods in the refrigerator to spoil, despite being temperature and humidity controlled to have the fruits and vegetables last longer. Allowing the consumer to have a list of items in their bins could ultimately lead to a more informed consumer and less food spoilage. A single camera could identify items by making predictions when the bins are open, but the camera would only be able to "see" the top layer of produce. If one could combine the data from the open bins with information from the user as they placed and removed items, it is hypothesized that a comprehensive produce bin inventory could be created. This thesis addresses the challenges presented by getting a full inventory of all items within the produce bins by observing if the hand can provide useful information. The thesis proposes that all items must go in or out of the refrigerator by the main door, and by using a single camera to

observe the hand-object interactions, a more complete inventory list can be created. The work conducted for this hand analysis study consists of three main parts. The first was to create a model that could identify hands within the refrigerator. The model needed to be robust enough to detect different hand sizes, colors, orientations, and partially-occluded hands. The accuracy of the model was determined by comparing ground truth detections for 185 new images to the model versus the detections made by the model. The model was 93% accurate. The second was to track the hand and determine if it was moving in or out of the refrigerator. The tracker needed to record the coordinates of the hands to provide useful information on consumer behavior and to determine where items are placed. The accuracy of the tracker was determined by visual inspection. The final part was to analyze the detected hand to determine if it is holding a type of produce or empty, and track if the produce is added or removed from the refrigerator. As an initial proof-of-concept, a two types of produce, an apple and an orange, will be used as a testing ground. The accuracy of the hand analysis (e.g., hand with apple or orange vs. hand empty) was determined by comparing its output to a 301-frame video with ground truth labels. The hand analysis system was 87% accurate classifying an empty hand, 85% accurate on a hand holding an apple, and 74% accurate on a hand holding an orange. The system was 93% accurate at detecting what was added or removed from the refrigerator, and 100% accurate determining where within the refrigerator the item was added or removed.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGMENTS	iv
ABSTRACT	v
LIST OF TABLES	x
LIST OF FIGURES	xii
INTRODUCTION	1
BACKGROUND	6
A. Trends in Computer Vision.....	6
1. Brief History of Computer Vision.....	7
2. Convolutional Neural Networks.....	9
B. Internet-Connected Appliances and Inventory Management	21
C. Trends in CV Applied to Hand Analysis	25
METHODOLOGY	28
A. Data Collection	29
1. Camera Setup and Specifications	29
2. Collecting Images to Develop the Dataset	31
3. Dataset Annotation	33
4. EgoHands dataset	36
B. Hand detection	37
1. TensorFlow Object Detection API	38
2. Python Environments	44
3. Metrics for Evaluating the Models and Algorithms	45
4. Model Training and Real-Time Metrics.....	48
5. Hand Detector Experiments	49
C. Hand tracking.....	50
1. Centroid Tracking	50
2. dlib Correlation Tracker	51
3. Determining Direction of Movement	52
4. Distinguishing Between Hands	53

5.	Determining if a Hand is Inside the Refrigerator	54
D.	Hand-Object Interaction.....	55
1.	Hand Holding Item or Not, using TensorFlow Model	56
2.	Hand/Object Segmentation.....	57
3.	Hand Analysis from Segmented Images	62
4.	Video for Testing	62
5.	Image Classifier using CNN.....	64
6.	Inventory Detection Pipeline.....	71
7.	Storing Inventory Information.....	74
RESULTS	76
A.	Hand Detection	76
1.	Determining the Best Model for Hand Detection.....	76
2.	Supplementing the Dataset with the EgoHands Dataset	77
3.	Results for Each Model in the Real-World Application	78
4.	Model Trained with Left and Right Hand Class	80
5.	Hand Detector Limitations	81
B.	Hand Tracking	82
1.	Centroid Tracking	82
2.	Correlation Tracker	86
3.	Direction of Movement	92
4.	Distinguishing Between Hands	97
C.	Hand Analysis.....	98
1.	Hand Empty or Not	98
2.	TensorFlow Object Detection API for Produce Detection.....	99
3.	Image Background Removal	102
4.	Image Classifier.....	112
D.	Object Add or Remove Logic	114
1.	Frames Per Second	115
2.	Hand within the Refrigerator.....	115
3.	The Algorithm	117
4.	Storing Inventory Information.....	127
ANALYSIS	129
CONCLUSIONS AND FUTURE WORK	135
Future Work	135

REFERENCES	140
APPENDIX A.....	150
APPENDIX B	153
APPENDIX C	155
CURRICULUM VITA	159

LIST OF TABLES

Table 1. TensorFlow detection model zoo metrics [92].	43
Table 2. Hand detector experiments and corresponding classes.....	49
Table 3. Class weights for each class to ensure a balanced dataset.....	68
Table 4. Performance results for various models, compared with published results.....	76
Table 5. mAP results for each model on the local dataset.....	77
Table 6. Precision and recall (left) and confusion matrix for EgoHands then local dataset (faster_rcnn_inception_v2_coco).....	77
Table 7. Precision and recall (left) and confusion matrix for local dataset (faster_rcnn_inception_v2_coco).....	77
Table 8. Precision and recall (left) and confusion matrix for models: (a) ssd_mobilenet_v2_coco (b) rfcn_resnet101_coco.....	78
Table 9. Disk size of each hand detection model.....	79
Table 10. Example hand tracking sequence where the tracker swaps the left and right hand.....	83
Table 11. dlib correlation tracker applied to a video sequence, detector running every fourth frame.	87
Table 12. Movement direction, shown by the red arrow, for one interaction.....	92
Table 13. Video sequence showing the original frame and the extracted foreground using MOG2.	103

Table 14. Video sequence showing the original frame and the extracted foreground using GrabCut.....	105
Table 15. HSV color space values for skin thresholding.....	107
Table 16. Skin threshold applied to frames corresponding to Figure 44 interaction.....	107
Table 17. Skin threshold applied to the detected hands corresponding to Figure 44 interaction.	109
Table 18. Precision and recall data for image classifier in production application.	113
Table 19. Confusion matrix for image classifier in production application.	114
Table 20. Number of frames per interaction (adding an item to the shelf) for different fps.	115
Table 21. Example interaction and add/remove logic	120
Table 22. Predicted and ground-truth classes for 14 interactions. Incorrect predictions are highlighted.	127
Table 23. Selected papers on research into inventory management systems in the refrigerator.	150
Table 24. Selected papers on research into produce classification.	151
Table 25. Annotation count by item for LabelImg annotations.....	153
Table 26. Annotation count by item for Semi-Automatic Image Annotation tool, Anno-Mage.	154
Table 27. Class breakdown for object classifier dataset.	154

LIST OF FIGURES

Figure 1. Household food waste by category.....	3
Figure 2. An image of a typical consumer’s produce bin.	4
Figure 3. Object detector performance on the popular PASCAL VOC dataset over time [14].....	9
Figure 4. An example of a convolution with a 3x3 kernel, no padding, and stride of 1 [18].....	11
Figure 5. An illustration of how the early convolution layers allow further layers to create more complex features [20].	12
Figure 6. Activated feature maps after different convolutional layers (left), and the corresponding image patch the activated the feature map (right) [21] [22].	13
Figure 7. Architecture of LeNet-5 [24].	16
Figure 8. Models available in Keras [37].	17
Figure 9. Speed and mAP comparison for each generation of R-CNN [42]	19
Figure 10. The Faster R-CNN Network [23].	20
Figure 11. FridgeEye, the battery-operated camera and automatic detection system to make any refrigerator a smart refrigerator.	24
Figure 12. Camera location in the refrigerator and corresponding field of view.	30
Figure 13. Example image from the webcam used for data collection.....	31
Figure 14. USDA Data [81].	32
Figure 15. Example of bounding box annotations for an image.....	33

Figure 16. Bounding box coordinate convention.....	35
Figure 17. Images from the EgoHands Dataset [87].....	37
Figure 18. Flowchart for preparing data for the TensorFlow Object Detection API.....	39
Figure 19. Example rows from the csv file used to convert the dataset to the format required for the TensorFlow Object Detection API.....	40
Figure 20. Example label map to use for the TensorFlow Object Detection API.	41
Figure 21. Training the model using TensorFlow Object Detect API in Google Colab ..	43
Figure 22. Intersection over union (IoU) is an important metric for evaluating an object detection model [99].	46
Figure 23. Image showing visualization of TP, FP, FN, and IOU. Red boxes are what the model predicts, yellow are ground-truth labels.....	47
Figure 24. Direction is determined by looking at the sign of the delta between the y centroid of the current frame versus that of the previous frame. W is the total width of the image frame, and H is the total height.	53
Figure 25. GUI to create holding/not holding annotations.	56
Figure 26. Left: The input image and blue rectangle provided to the program to designate the foreground. Right: The output of the algorithm. [111].....	58
Figure 27. Background subtraction works by developing a background model from prior frames, and then comparing that model to the current frame. The pixels that are different from the background model, with respect to a specified threshold, are considered foreground. The mask is created by setting all pixels above the threshold to white, and all others to black. [113]	59

Figure 28. Left: Image before the Gaussian blur is applied. Right: After the image is blurred.	61
Figure 29. Training images for the object classifier. The left image is an example in the class for Empty and the right is an example from the Apple class.	65
Figure 30. A flowchart showing the process of collecting data to training the classifier model.....	66
Figure 31. Folder structure for image dataset within Google Drive.	67
Figure 32. Left: the original training image for class Apple. Right: Example random augmentations using the ImageDataGenerator class.	69
Figure 33. VGG16 Model Structure [129].....	70
Figure 34. Flowchart for detection model applied to a video sequence.	72
Figure 35. “Loading zone” area within which the classifier will run to detect what a hand is holding.....	73
Figure 36. Precision and recall training metrics for EgoHands dataset with left and right hand class.....	80
Figure 37. Precision and recall training metrics for local dataset with left and right hand class.....	80
Figure 38. Prediction on a validation video for the left/right hand model.....	81
Figure 39. The hand detector did not detect the hand on the edge of the frame.....	82
Figure 40. An example image of an edge case where it is difficult to definitively determine if the left hand is empty or not.	98
Figure 41. An example image where it is unclear whether the left hand opening the bin should be considered empty or not.	99

Figure 42. Precision and recall for the model trained on all categories.....	100
Figure 43. Inconsistent annotation where the apple bounding box covers both single and multiple apples. Each red box denotes the <i>apple</i> class.	101
Figure 44. A challenging image to determine if the apples are in the right hand or within the bottom produce bin.	102
Figure 45. Original, thresholded, and k-means cluster for empty hand.....	112
Figure 46. Original, thresholded, and k-means cluster for hand holding an orange.....	112
Figure 47. Precision and recall data for image classifier.	113
Figure 48. Training loss and accuracy curve for image classifier.	113
Figure 49. An example where the hand is out of frame, but the arm can be seen.	117
Figure 50. Flowchart of hand analysis logic.	118
Figure 51. Thresholds for determining item location within the refrigerator.	120
Figure 52. Excel spreadsheet storing refrigerator inventory.....	128
Figure 53. Logic for updating the Excel spreadsheet storing the inventory information.	128
Figure 54. Example mobile or web app for displaying inventory information to the consumer.	138
Figure 55. Annotation naming conventions.....	153

INTRODUCTION

Analyzing systems to prevent food spoilage could address a costly problem in the homes of consumers. It is estimated that 21% of the food purchased by consumers in 2010 went to waste, resulting in a loss of \$114.9 billion dollars in the US, or \$371 per person per year [1]. Food waste within retail has long been identified as an issue. To reduce waste, retailers and restaurants use technology to track their food stores, utilize their stock more efficiently, and help make more-informed decisions when ordering new items. [2] This process, known as inventory management, has only recently been extended within the home, and more specifically the refrigerator, to combat consumer waste. Inventory management in the refrigerator is the process of maintaining an accurate record of the contents inside. Important information for inventory management includes what an item is, where it is located, how long it has been in the refrigerator, and how long the item will stay fresh. Additional technology within the fridge itself can make an automated inventory management system, which can lead to a more-informed consumer without overburdening them with food-tracking tasks, and provide automated information on the contents of the refrigerator to reduce food spoilage and waste.

Both Samsung and LG have provided solutions by offering refrigerators with cameras. The cameras are located within the appliance and promise to give consumers a constant view of their refrigerator's contents. The benefits of these products include the potential for that view to lead consumers to make more-informed decisions at the grocery store and decisions that can reduce spoilage, as well as providing insight and valuable data about how consumers use their refrigerator. However, the camera is only providing data, meaning the technology needs to extract and analyze information from the data

stream to be truly useful to the consumer. Additional drawbacks of the approach are that many cameras are needed to get a full view of all areas of the fridge, the consumer gets no information on how long an item has been in the fridge, and the produce bins remain invisible to the camera.

One topic of research to improve upon inventory management inside consumer refrigerators is to use artificial intelligence to extract information from the camera data. Artificial intelligence (AI) is an important topic in today's world, with uses stretching from driverless cars to computers making cancer predictions. A subset of AI, computer vision, is a promising field for analyzing the refrigerator camera feed. Computer vision uses cameras to allow a computer to interpret the world around it. Computer vision topics include identifying, classifying, and locating objects within a scene. With computer vision techniques, an inventory management system could identify and classify food items, register when an item is placed inside, log how long an item has been inside, or even identify spoiling food and alert the consumer. The system could provide users with the best storage locations within the fridge for a particular food, provide recipe suggestions based on the fridge's contents, or help the user make smart eating decisions based on what they have already eaten that day.

The Consumer Electronics Show (CES) is the annual preeminent showcase for manufacturers to debut their latest, cutting-edge solutions for consumer appliances and products [3]. Every company that makes appliances attends this convention to influence the marketplace for decades to come. At CES 2020, Samsung and LG both announced computer vision capabilities within some of their refrigerator units. Both companies promised their AI could automatically detect items in the fridge and build a virtual food

inventory. Samsung describes the technology on their website as, “[...] the new Samsung Family Hub with the ViewInside camera, where AI-powered image recognition is used to first understand what’s inside the fridge. Then, the fridge recommends a curated feed of recipes that incorporate the ingredients you already have with your preferences, desires and situational needs.” [4] Neither company has released these new features to consumers, so it is still unknown how accurate the AI is at detecting objects.

While having a constant inventory of items in the main refrigerator compartment is useful, both Samsung and LG’s solutions neglect the location where the most food is wasted: *the produce bins*. Studies have shown that vegetables and fruit are the top category of wasted food. According to the Natural Resources Defense Council (NRDC), Figure 1 represents the estimated total food loss per household in the United States in 2017 [5].

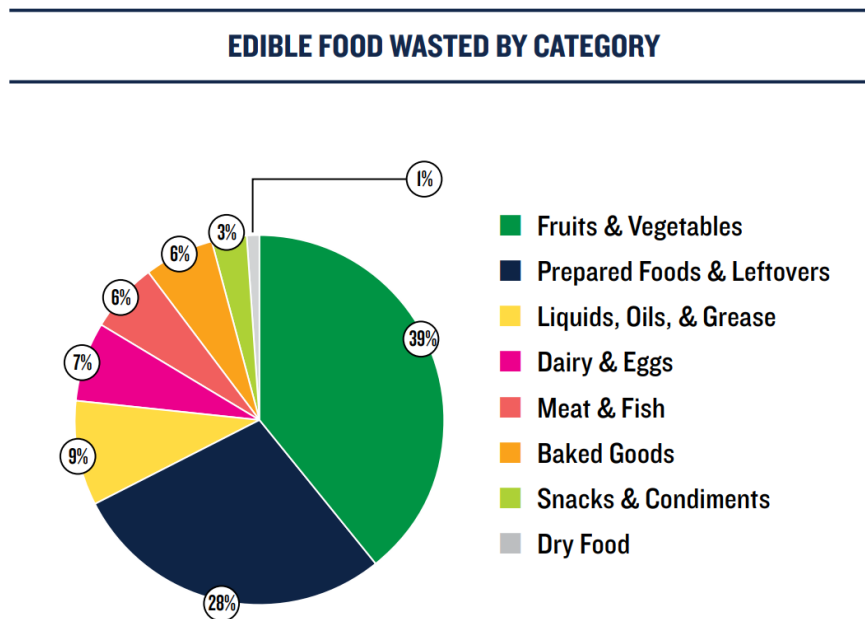


Figure 1. Household food waste by category.

Figure 1. shows that 39% of all food waste is fruits and vegetables. A large contributor to the waste is due to produce becoming inedible or spoiled. ReFED, an American non-profit that researches ways to reduce food waste, believes that fifty-nine thousand tons of food could be saved each year by improving “[...] the ability of retail inventory management systems to track an average product’s remaining shelf-life (time left to sell an item) and inform efforts to reduce days on hand (how long an item has gone unsold)” [6] Inventory management should not be limited to retail, and an automatic inventory system for the produce bins would allow users to always be aware of every produce item they have, how long it has been in the fridge, and how long it will stay fresh. Giving this information to the consumer will help them make better grocery-shopping decisions, and ultimately reduce produce waste at the consumer level.

Because the produce bins are deep and difficult to see inside, items can easily get lost and forgotten under layers of other produce. Figure 2, below, shows the contents of a typical consumer’s produce bin.



Figure 2. An image of a typical consumer’s produce bin.

As shown in Figure 2, the items on top are easily distinguishable, but the layers underneath are invisible. Typically, one would need to remove the top layer to see underneath, but doing so is cumbersome and inefficient.

What if the fridge could tell you what was underneath those layers without you needing to do a thing? This research aims to explore that question by observing if the hand can provide useful information. The thesis proposes that all items must go in or out of the fridge by the main door, and by using a single camera to observe the hand-object interactions, a more complete inventory list can be created.

BACKGROUND

The idea of inventory management in the refrigerator has been around almost as long as the refrigerator itself. Knowing what is in the refrigerator is helpful for making grocery lists, planning the dinner menu, and preventing food spoilage. Manually listing all fridge items is impractical. A simple solution is to keep a grocery list on the fridge and add to it when one uses the last of an item. This only works if all members of the household remember to add to the list when they finish an item, and does not address spoilage. An extension of this idea, utilizing the now ever-present virtual assistants such as Siri and Alexa, is to keep a virtual list by telling them when you use the last of a food product. Virtual assistants make tracking inventory somewhat easier and allow the list to be kept always accessible on a cell phone, but still relies on consumer action to add items to the list. Also, Siri and Alexa provide no information on the condition of the inventory and whether an item is past its prime [7] [8]. Apps like Fridge Pal allow users to scan the barcodes of food items to keep track of what is in the fridge or freezer [9]. Once again, this requires consumer action to remember to add new items and does not address food spoilage. The above solutions are a good start, but the products still rely heavily on input from the consumer. A better solution would be to leverage research from the field of computer vision and the rise of internet-connected appliances to implement a fully-autonomous system to identify and track inventory. The system would automatically share pertinent information to the user, and provide an up-to-date refrigerator inventory list accessible anywhere through a web or mobile application.

A. Trends in Computer Vision

1. Brief History of Computer Vision

Computer vision (CV) is a subset of artificial intelligence which aims to replicate the human vision system by using algorithms to gather meaningful information from images and video. Early research studied how the vision systems worked in mammals. The influential 1959 paper by Hubel and Wiesel, “Receptive fields of single neurons in the cat’s striate cortex”, studied cats to try and understand visual perception. The research found that individual neurons reacted to stimuli in different orientations and locations, with layers of neurons working together to aid in perception. The neurons connected most directly with the eyes first detected visual information like the orientation of edges which then allowed neurons with subsequent connections to extract higher-level information from those areas of interest [10]. The idea of using more general features like edges to then develop more complex ones is the basic idea behind most computer vision techniques used today. The computer vision field arguably began in the late 60s when a team of researchers at the MIT Artificial Intelligence Group believed that, over their summer break, they could create a vision system to recognize objects. Called the “Summer Vision Project,” the research was unable to meet the goal of an autonomous object recognition system, but paved the way for computer vision today [11].

Object recognition is a subset of CV that studies how to recognize objects in an image or video. The ImageNet Large Scale Visual Recognition Challenge defines the major tasks of object recognition as:

1. **Image classification** (2010-2014): Algorithms produce a list of object categories present in the image.

2. **Single-object localization** (2011-2014): Algorithms produce a list of object categories present in the image, along with an axis-aligned bounding box indicating the position and scale of *one* instance of each object category.
3. **Object detection** (2013-2014): Algorithms produce a list of object categories present in the image along with an axis-aligned bounding box indicating the position and scale of *every* instance of each object category. [12]

The CV techniques of classification, localization, and detection are studied and implemented in this work. Each of these techniques selects features from images and uses algorithms to predict the object (or class) label and bounding box (if required for the task). Learning or training are terms used in the field of CV, and indicate the parameter optimization process used to create useful algorithms for a particular task [13]

Early object recognition research focused on hand-engineering the features used for training. Techniques like Scale Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF), and Histogram Oriented Gradients (HOG) could extract edges, shapes, and other discriminating features [13]. The features could then be passed into an algorithm like Support Vector Machine (SVM) or Random Forrester (RF) to learn to identify the object. [13] These techniques required small datasets for training and could easily be run on the available computer technology. By 2012, progress on object recognition tasks had stalled. It was not until the resurgence of convolutional neural networks (CNN) that object recognition research was able to reach levels close to human-level accuracy [13]. Figure 3, below, shows how research plateaued, and then improved dramatically with the resurgence of CNN.

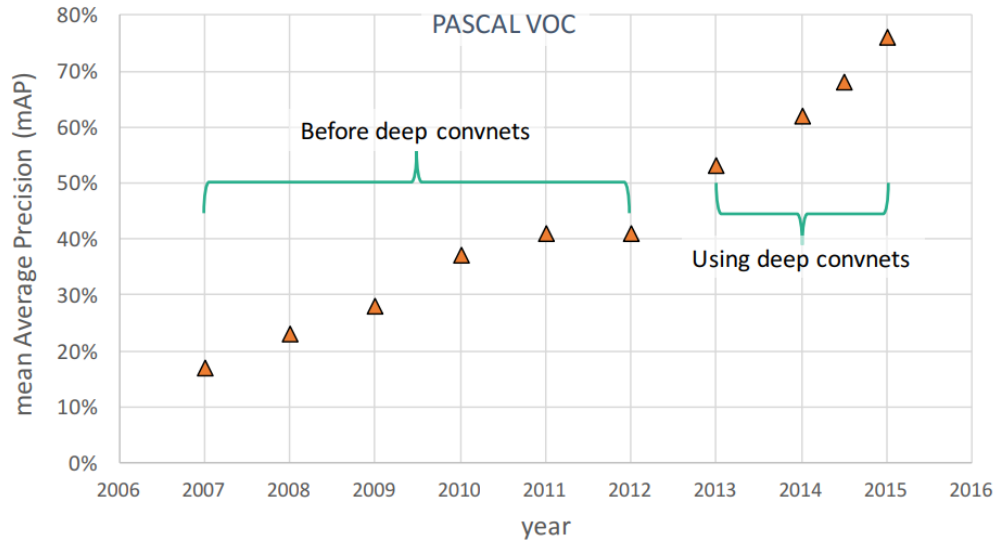


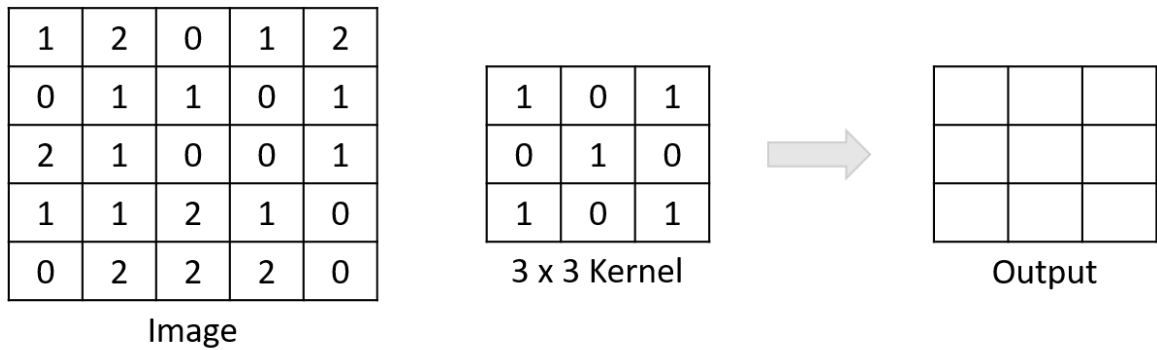
Figure 3. Object detector performance on the popular PASCAL VOC dataset over time [14].

Figure 3 shows that with the proper data, CNN models perform better on object detection tasks than prior techniques.

CNN were first introduced by Fukushima in 1988, but was limited in its usefulness because it required large datasets and powerful computers that were not available at the time [15]. One benefit of CNN over other techniques is that CNN learns the distinguishing features from the image dataset, whereas prior techniques used human-defined features. By allowing features to be learned from the images, CNN can be used to detect patterns and descriptors unique to the application. Another benefit of CNN over older architectures like SIFT and HOG is that the CNN maintains the spatial integrity of an image, whereas SIFT and HOG flattens the image into a 1D matrix. Spatial information is essential when localizing an object in an image.

2. Convolutional Neural Networks

Convolution is a mathematical process that takes two functions and calculates how one function will affect the other [16]. Convolutions are used extensively in signal processing, and can be used in image processing as a filter to blur, smooth, and otherwise alter the input image [17]. Convolution works on an image by passing a kernel of $N \times N$ size along each pixel of the image. At each pixel, the kernel is multiplied by the underlying portion of the image. The sum of all the multiplications is the value for that pixel location in the convolved output image. An example of the operation is shown Figure 4.



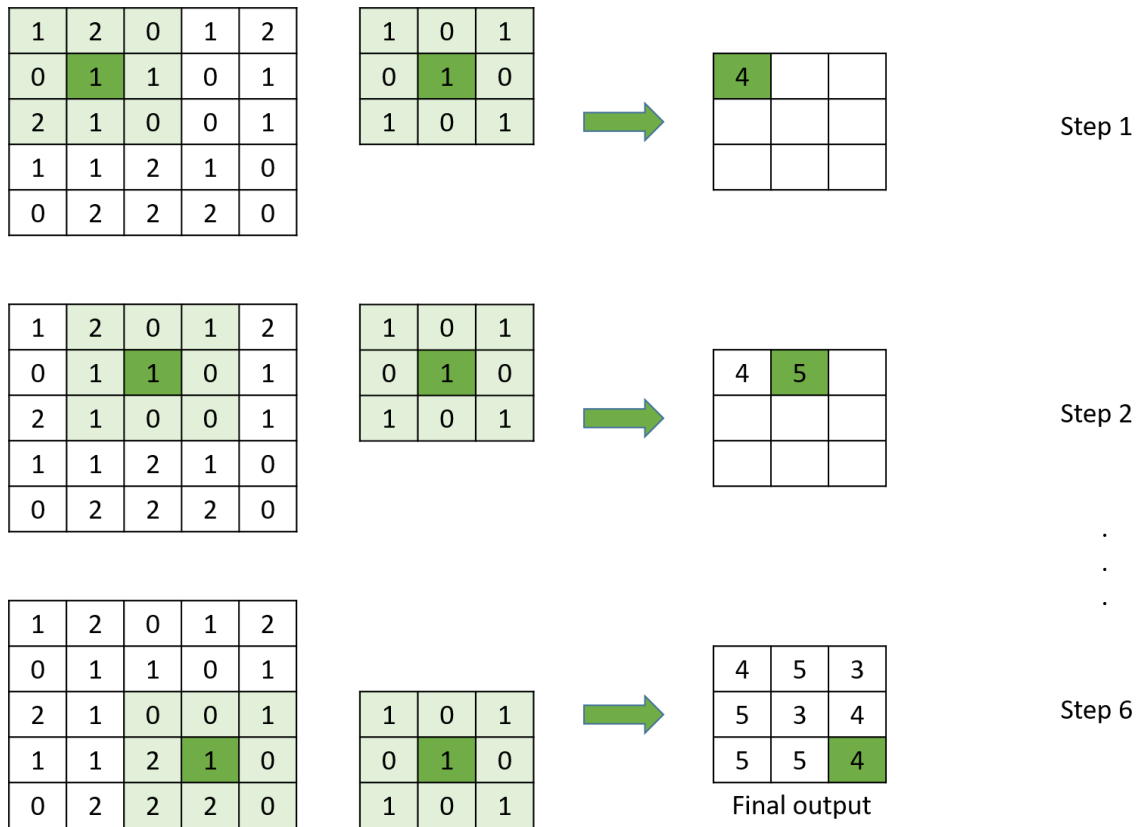


Figure 4. An example of a convolution with a 3x3 kernel, no padding, and stride of 1 [18].

Figure 4 shows a convolution of an image with a 3x3 kernel. The number of pixels the kernel moves per operation is called the stride. Figure 4 shows that the operation reduces the dimensions of the original image. A high value in the final output means that the feature for the kernel (such as a line or edge) was found in that location. A low value, or zero, means that feature is not present in that location [19]. Padding is the process of adding extra rows and columns around the border of the image to maintain the original image dimensions. The example above has a stride of 1 and no padding [18]. Padding, stride, kernel size, as well as number of kernels and resulting feature maps for each convolutional layer, are parameters that can be changed to optimize the CNN model [17]. The final output, or feature map, of the kernel can then be fed into another convolutional

layer. By stacking convolutional layers, the model is able to learn kernels that first detect edges and lines, then in the following layers learn kernels that combine those attributes to learn more complex features. There are many kernels per layer, with each kernel learning a different discriminating feature of the image. Because each kernel slides over each part of the image, only a single horizontal line filter is needed to detect all horizontal lines in the image. An example of how simple features can combine to detect more complex features is represented by Figure 5.

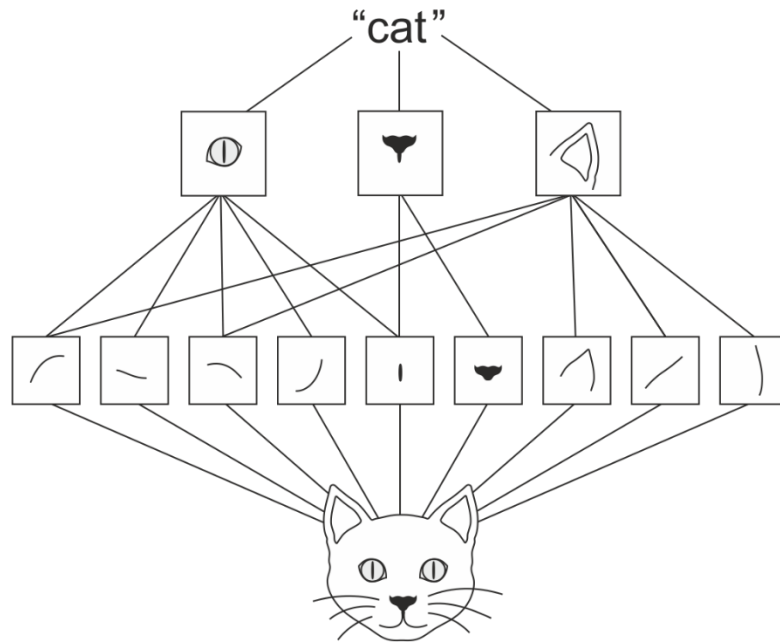


Figure 5. An illustration of how the early convolution layers allow further layers to create more complex features [20].

Figure 5 shows how edges and lines in the first convolutional layer can be combined in the next layer to detect eyes, noses, and ears. The combination of these features can then be learned by the model to represent a cat object [20].

Figure 6 shows which feature maps “activate” (or compute a large value when the kernel is passed over the region) as an image progresses through the convolution layers.

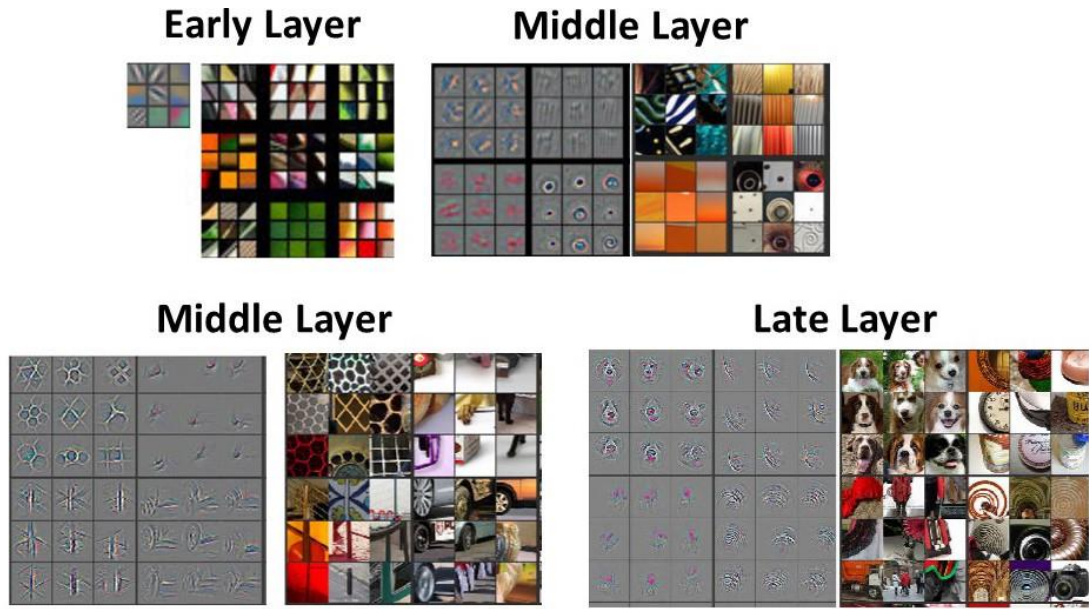


Figure 6. Activated feature maps after different convolutional layers (left), and the corresponding image patch the activated the feature map (right) [21] [22].

Figure 6 shows that early layers, or layers more directly connected to the input image, detect edges and colors, while later layers can detect complex features like dog faces. Not all feature maps will activate for a given image, and the unique combination of activated feature maps defines which object is present in the image. After the convolutional layers, the model becomes like a traditional neural network with one or more fully connected (FC) layers feeding into a function to determine probabilities for each class. Loss is calculated with respect to the ground-truth label versus the output class probabilities, and the error is back-propagated to the weights in the fully connected layers and the kernels in the convolution layers. [23] Essentially the training algorithm is used to reduce error in the same way that a gradient decent optimization scheme works, with larger error values leading to larger corrections in model weights.

The ability of an image classification model to learn discriminating features is reliant on massive amounts of quality labeled data. At least 1,000 images per class are

needed to get accurate and generalizable performance from a CNN [17]. Small datasets can impact not just the accuracy of the model, but can also cause overfitting. Overfitting occurs when the model models the training data too well, and thus performs well on the training set but is unable to generalize to new data [17]. Lack of data limited early CNN development, but more recently, large, high-quality datasets have been created to fill that gap. One of the earliest large scale image databases that was a benchmark for both traditional and CNN models is the Modified National Institute of Standards and Technology database (MNIST) handwritten digit database. Introduced in 1998, MNIST features 600,000 train and 10,000 test labeled images [24]. More complex datasets like PASCAL VOC (Currently 500,000 images, 20 classes, class and bounding box annotations) [25], ImageNet (Currently over 14 million images, 20,000 classes, class and bounding box annotations) [26], and Microsoft Common Objects in Context (COCO) (Currently over 2 million images, 91 classes, class, bounding box, and pixel-level annotations) [27] have become the benchmarks by which to measure new CV models. In addition to providing more data, Chen, Goodfellow, and Shiens showed in 2015 that the weights from a model trained on a large set of data can then be used as the starting weights for a new model for a different task. These pre-trained models have already learned many basic features from the other data that can then be used to better learn the new data [28]. Called transfer learning, the technique reuses convolutional layer feature maps from larger datasets. The early layers of less complex features are kept, while either the later convolutional layers or the fully connected layers are retrained by back-propagating the loss and only updating the desired kernels and weights. With transfer learning, *hundreds*, not thousands, of images per class can be used to train an accurate

classifier or object detector. Transfer learning can also reduce training time, and help the model generalize from the training data to real-world applications [13]. Transfer learning is used extensively in this thesis for both the classification and detection tasks.

Another reason for the resurgence of CNN is the introduction of highly optimized machine learning frameworks. TensorFlow, PyTorch, Keras, and Caffe are some popular open source machine learning frameworks [13]. The frameworks are optimized for speed and efficiency, and have ample documentation, tutorials, and large communities to make the complicated algorithms more straight-forward to implement. Keras is a high-level platform that runs on top of lower level libraries like TensorFlow. Keras plus TensorFlow are used in this research for building image classifiers. Keras has built-in functions to run popular model architectures and utilize pre-trained weights from ImageNet and other datasets. The TensorFlow Object Detection API is used for the object detector. The API is written in Python, and has libraries that implement state-of-the-art object detection architectures like Single Shot Detector (SSD) and Faster R-CNN [29] [30]. In addition to the frameworks, companies like Google and Microsoft now provide coding platforms that come with the popular machine learning frameworks already installed [31] [32]. Beyond reducing the learning curve that comes with setting up the frameworks, the platforms let users train their models in the cloud using powerful GPU and TPUs. Google Colaboratory (Colab) is used in this research to train all models.

a. CNN for Image Classification

Training a model utilizing transfer learning involves deciding which model architecture to use for the task. There has been extensive research into optimizing the structure of a convolutional neural network [13]. The research problem involves finding

the best combination of network layers and depth to give enough parameters for the model to learn distinguishing features, while balancing the size and computational power necessary to train all the parameters. LeNet, introduced in 1998, was one of the earliest examples of a neural network using CNN layers [24]. The structure of LeNet is shown in Figure 7:

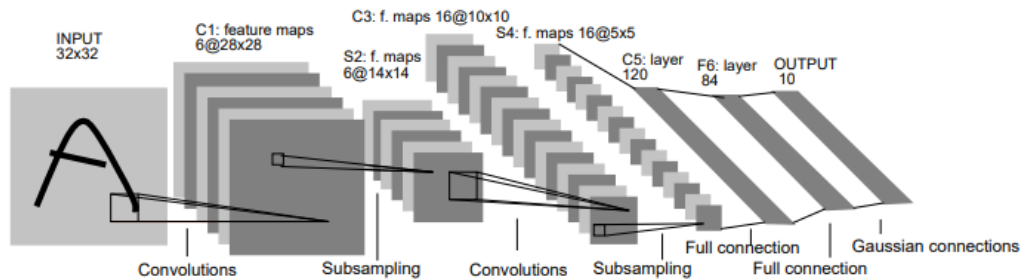


Figure 7. Architecture of LeNet-5 [24].

Today's networks have a similar structure to LeNet, with each convolutional layer producing a number of feature maps (a feature map for each convolutional kernel). Between the convolutions is a layer which reduces the dimensions the feature map, while maintaining the spatial integrity of the image. Reducing the size of the feature maps is important to reduce the number of computations required as the number of feature maps increases. Each subsequent convolutional layer increases the number of feature maps while decreasing the dimensions of the maps. In the final layers, the images are flattened and passed to fully connected layers which use the activated features from the CNN layers to determine the likely class [33]. Lastly, the output layer is assessed to classify the image based on the number of classes for the specific task. For the handwritten digit task, there are ten classes for the digits (0 to 9). LeNet had few layers compared to today's networks, but was able to perform better than any other model at that time on handwritten digit recognition [24]. Current networks have many more layers, or depth, than LeNet

and often handle 3-channel RGB images instead of a single channel for black and white images, but the basic structure of the network is still used. There was a lull in research using CNN until 2010 when it was shown that powerful graphical processing units (GPU) could be used to train the networks, allowing for much larger networks able to learn more complex features [34]. Building on the idea of deep networks with many layers, AlexNet, in 2012, was able to achieve the lowest ever error rate on the ImageNet dataset. AlexNet error was 37.5%, compared to the previous best of 45.7% using SIFT [12]. AlexNet's success on the ImageNet dataset led to the boom of CNN-based image classifiers seen today [35] [36].

Keras offers applications to easily implement many of today's popular architectures [37]. The applications allow for training the models from scratch, or using weights pre-trained on ImageNet. A list of models available in Keras is shown in Figure 8:

Documentation for individual models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Figure 8. Models available in Keras [37].

The models in Figure 8 can be used to classify images containing any of ImageNets 20,000 classes, or as a starting point for training a model to detect new classes.

b. CNN for Object Detection

One of the first instances of CNNs used for object detection is the OverFeat framework introduced by Sermanent et al in 2013 [38]. OverFeat uses multiscale sliding windows to detect and identify objects. Later in 2014, Girshick et al introduced Regions with CNN features (R-CNN) which improved upon the relative accuracy of the next best architecture on PASCAL VOC 2012 by 30% [39]. R-CNN obtained a mean average precision (mAP) of 31.4% on the ILSVRC2013 detection dataset, compared to 24.3% mAP for OverFeat [39]. Mean average precision, the ability of the model to correctly detect the desired object in an image, is a standard metric for measuring the accuracy of an object detection model. mAP is discussed in more detail in the Methodology. R-CNN uses the selective search algorithm to produce regions of interest (ROI) that are then passed to CNN layers to extract features. Selective search utilizes natural boundaries in an image, such as color and texture, to segment the image and use those segments as ROI [40]. Finally, the extracted features from the CNN are passed to SVMs to make a classification. The paper's researchers continued to improve on their framework, first with Fast R-CNN in 2015 [41], and finally with Faster R-CNN in later 2015 [30]. Faster R-CNN improved speeds by using a single CNN to generate region proposals (called the Region Proposal Network or RPN) and classify/detect the objects. A comparison of speeds is shown in Figure 9.

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	0.2 seconds
(Speedup)	1x	25x	250x
mAP (VOC 2007)	66.0	66.9	66.9

Figure 9. Speed and mAP comparison for each generation of R-CNN [42]

The purpose of the RPN is to generate good features that can be used to find potential object locations. Images are passed through a pre-trained CNN, up to an intermediate convolutional layer to create a feature map [43]. The original paper used VGG16 trained on ImageNet. Images are propagated up to the fifth convolutional layer to create a feature map proportional to the original image but with greater depth [30]. Potential regions are found by sliding anchor boxes of different sizes (0.5, 1, 1.5) and aspect ratios (1:1, 1:2, 2:1) over the output feature map [23]. The nine boxes at each point are used to both classify and predict bounding boxes. One FC layer examines each anchor box and scores it as either an object or not an object. A separate FC layer produces four offset values (x center offset, y center offset, width offset, and height offset) to predict how the anchor box needs to be offset to encompass the object [44]. Boxes with high object scores are then passed through processing to reduce the boxes to a predetermined max value of regions, and reshape each box to be a uniform size [45] [44]. The proposals are then passed to more FC layers, which classify the regions into the specified class or a background class to be discarded, and further improve the bounding box offsets. The basic structure of the Faster R-CNN is shown in Figure 10.

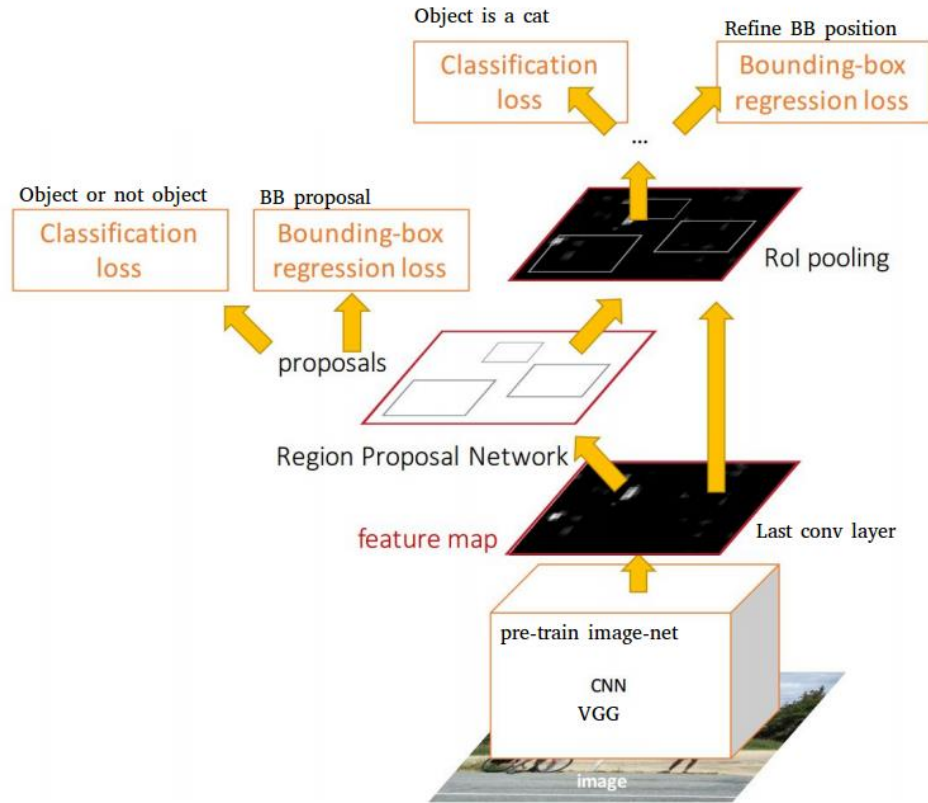


Figure 10. The Faster R-CNN Network [23].

Figure 10 illustrates that, unlike the prior frameworks which needed to be trained in parts, Faster R-CNN can be trained all at once. Faster R-CNN consists of four losses: RPN classification and bounding box localization losses, and R-CNN classification and bounding box localization loss [30]. Not only does end-to-end training speed up computation time, it also improves accuracy [42]. Faster R-CNN is considered a two-stage detector because there are two stages, the RPN and the fine-tuning stage, to the network. Single stage detectors, like Single Shot Detector (SSD) and You Only Look Once (YOLO), are faster than two-stage detectors, but are often less accurate [17]. More information on object detection frameworks can be found in the SSD and YOLO papers

[29] [46]. Experiments have been done to determine which of the frameworks is best suited to this research, and the results can be found in Section A of the Results.

The rise of CNN for object recognition tasks was driven by an increase in computing power and the introduction of large datasets. Optimized machine learning frameworks like TensorFlow and Keras allow state-of-the-art neural network research to be applied to real-world problems like automatic inventory management.

B. Internet-Connected Appliances and Inventory Management

The current proliferation of wireless, internet-connected appliances has opened up new solutions to inventory management. Often given the moniker “smart,” the connected devices give consumers unprecedented control over their products [47]. Smart appliances allow users to control their units remotely, get instant software updates, and allows appliance makers to add innovative new technology to their products. Internet-connected refrigerators have allowed researchers to install barcode scanners, RFID readers, scales, cameras, and other sensors into refrigerators to develop inventory management systems. The sensor information was used to track inventory within the refrigerator, alert users of items close to their expiration date, suggest recipes and provide nutritional information, create grocery lists, or identify the refrigerator contents. [48] A limitation of barcode or scale-based inventory systems is that they place the burden on the consumer to scan the barcode or ensure the item is placed on the scale [49]. RFID systems do not need to be manually scanned, but still require the consumer to add tags to items. Some food manufacturers are adding RFID to their packaging; but many items, specifically *produce*, would need to be manually tagged by the consumer in order to work in the systems mentioned above [48] [50]. One study used a 360-camera within the produce bin to

automatically identify items. The research used multiple photos of the bin to extract the food region and identify the object using histogram matching. The research achieved 96.5% accuracy on four produce items, but identification could only occur if a single item was present in the bin. [51] Another study used a consumer's mobile phone as the inventory management system. Through a phone application, the user was able to scan barcodes, identify items by taking a picture, and get nutritional information. [52] Utilizing mobile phone technology has the potential to give smart refrigerator capabilities to anyone with a phone, but the system is still dependent on the user actively operating and updating the system every time the refrigerator inventory changes. The 2012 paper, *Negotiating Food Waste: Using a Practice Lens to Inform Design*, explored the causes of food waste in a consumer home by studying the behaviors of 14 households. The researchers installed a camera inside each of the participant's refrigerator that would take pictures inside the unit every time the door was opened. The pictures would be uploaded to a website, and could be accessed by the household at any time. The research showed that while the participants found the camera images useful when they remembered to look at them, most users simply forgot the images were available. The study indicates that it is important for an inventory management system to proactively inform the user instead of relying on the consumer to get the information from the system on their own. [53] A more comprehensive list of refrigerator inventory management research can be found in Appendix A.

Appliance companies are taking advantage of the new connectivity by installing cameras in the refrigerator to allow users to see inside even when they are away from home. Samsung's Family Hub™ Side-by-Side refrigerator has internal cameras that

connect to an app to give users a constant video stream inside their fridge [54]. LG's ThinQ smart fridge offers a panoramic camera inside the fridge, as well as Amazon Alexa integration [55]. Both products are high-end, costing at least double what a typical refrigerator would cost. Aside from the cost, a limitation of these products is that even the best organized fridge will have items hidden behind other items. Also, there are areas in the fridge, like the fresh food bins, that are not visible unless opened.

A better solution would be an automatic inventory management system. The system would identify and track all items within the refrigerator and how long they have been there; automatically adding and removing items from the inventory. The technology could alert the user when something is about to go bad, provide an up-to-date inventory list on the go through a mobile application, and suggest recipes based on items that are nearing their best-by-date. Research has shown promising results using machine learning techniques to automatically identify items within the refrigerator. The introduction of large produce databases like VegFru, with over 160,000 images of fruits and vegetables, and Fruit-360, 90,483 images of fruits and vegetables, have enabled CNN models to be applied to produce classification [56] [57]. A challenge specific to fruits and vegetable is there is a lot of variability even within the same class of food item. Research has shown that models trained on the large produce databases perform well on the training set, but are much less accurate in applied settings due to the produce variability. A study found that supplementing the data with application specific images improved accuracy. [58] The studies have shown that CNNs are successful at accurately identifying produce items, but much of the research remains academic and has not been applied to the

development of consumer products. A table of selected research for fruit and vegetable identification can be found in Appendix A.

Both Samsung and LG announced at CES 2020 new capabilities using AI to automatically identify objects in the fridge [59]. The company FridgeEye has developed a standalone camera system that adds fridge-viewing capabilities to any refrigerator.



Figure 11. FridgeEye, the battery-operated camera and automatic detection system to make any refrigerator a smart refrigerator.

Figure 11 shows the FridgeEye camera housing can suction to the wall of the refrigerator. This system advertises that it provides users an up-to-date camera stream of their refrigerator's contents from a phone app. FridgeEye also claims it can automatically detect and list objects in the fridge to provide users with a virtual inventory. The FridgeEye product has not been released so it remains to be seen how well the automatic detection works. [60] There are many downfalls to these products, especially because the technology is so new. Companies lack the extensive application-specific product and food data required to make successful AI systems that can detect and track all possible food items. Additionally, AI models require significant on-board and cloud storage space, and AI systems like those above still struggle with occluded items.

The biggest limitation with research into automatic inventory management using computer vision is very few systems have been deployed in an actual consumer refrigerator. Field studies of this nature are important. They can further highlight the shortcomings of a technique, prove the potential of promising research, provide valuable application specific data, and ask new questions to shape the next wave of research.

C. Trends in CV Applied to Hand Analysis

Hand analysis for this research consists of detecting the hand within the frame, tracking hands from frame to frame, and determining what the hand is holding. Computer vision research has focused on hands since the 1990s. The research initially focused on recognizing hand gestures, which could then be used for human-computer interaction, sign language recognition, and hand-object interaction for virtual reality systems [61]. Less work has been done to use the hand as an anchor to detect objects, but Amazon is researching this currently as a way to implement grocery stores that automatically know what a user has purchased [62]. The recognition research, while not entirely applicable to this study, can be used as a starting point for hand identification and tracking. Early hand gesture recognition systems used either sensors to relay location information or skin-based thresholding to detect and extract the hand and arm region [63]. The sensor-based system is limited because the user must wear a glove with the sensors embedded within. Skin-based thresholding uses the color of the skin to extract skin regions from an image. Research has shown that transforming images to different color spaces can help extract skin in images even under different illumination conditions, and increase the similarity between different skin tones [64]. One study found that the optimum thresholds for skin detection are [85, 85, 85] for the RGB color space, and [180, 50, 33] for HSV [65]. The

major drawback of using skin color for identification is that it is less accurate when the background is similarly skin-colored. The introduction of large hand datasets like Oxford and EgoHands have allowed CNN to be used for hand detection [66] [67]. The creators of the EgoHands dataset were able to get over 80% accuracy using a CNN trained on EgoHands. Unlike the color-based skin thresholding, CNN models are not dependent on color differences within an image and can detect hands even when the background is similarly skin-colored. Another technique for detection that is not limited to hands is background subtraction. Background subtraction algorithmically creates a model of the background (i.e., stationary) area. The background area is used as a mask to remove the stationary information from a sequence of images and keep only what is moving [68]. In the case of hands moving in and out of a refrigerator, the extracted foreground would consist of the hand and the object it is holding. Research has shown that a Gaussian mixture model performs well at modeling the static background scene [69]. Background subtraction is also not limited by color similarity, but does become less precise in dynamic environments when the background is constantly changing [70]. Research has shown that background extraction is also useful for produce recognition [71].

Hand tracking, or more broadly, object tracking, is the process of detecting objects in an image frame, assigning each object a unique ID, and tracking the objects in subsequent frames while maintaining the associated IDs. Object tracking is important for extracting context from sequences of movements. Object tracking is a common research topic in the field of computer vision, and many of the state of the art tracker codes are publicly available [72]. Hand tracking is an especially difficult subset of object tracking, because hands often overlap and switch places in a scene. Sensor based hand detection

techniques can easily track hands by identifying the unique sensors, but purely vision based techniques struggle with maintaining the correct object IDs [73]. One of the simplest solutions for a CV based hand tracking system is to use the center of the detected hands bounding box, or centroid, to correlate hands between frames based on the distance between old and new centroids [74]. Tracking via centroids does not handle maintaining the correct object IDs for multiple hands within a frame, but is fast and simple to implement. One solution to distinguish between multiple hands and the difficulty with skin-colored backgrounds is to use a camera that can detect the depth of objects. A depth-based camera can easily differentiate between foreground and background due to the extra depth information. Using the extracted foreground region and hand segmentation would provide a clear representation of the object being held by the hand. The major limitation of depth-based detection is the hardware (camera) cost is more than double the cost of a traditional three-channel color camera. [75]

Much research has gone into hand recognition, but there remain few real-world applications outside of gesture recognition. This thesis aims to explore hand-object interaction methods to assist in an inventory management system for fresh food in produce bins.

METHODOLOGY

Multiple experiments were designed to study whether it was possible to determine what a user is holding as their hand moves in and out of the refrigerator, and if that information can be used to automatically add and remove items from the refrigerator inventory list.

The work conducted for this hand analysis study consisted of three main parts. The first was to create a model that could identify hands within the refrigerator. The model needed to be robust enough to detect different hand sizes, colors, orientations, and partially-occluded hands. The accuracy of the model was determined by comparing ground truth detections for 185 new images to the model versus the detections made by the model.

The second was to track the hand and determine if it was moving in or out of the refrigerator. The tracker needed to differentiate between the left and right hand, as well as different users' hands. The accuracy of the tracker was determined by visually inspecting and comparing the ground-truth hand location with the output of the hand tracker. Visual inspection involved viewing recorded video of hands within the refrigerator and noting how well the tracking algorithm marked hands and tracked them within the appliance.

The third part was to analyze the detected hand to determine if it was holding something or was empty. Multiple experiments were tried and validated through visual inspection by noting how well an experiment accomplished the task. The accuracy of the hand analysis using an image classifier was determined by observing its performance on a validation video consisting of 161 frames with ground-truth labels.

Finally, the detector, tracker, and hand analysis components were combined to see if the program could autonomously determine and log if an apple or orange was added or removed from the refrigerator. The accuracy of the add and remove logic was determined by comparing the programs performance to the ground truth labels of eleven add and remove interactions in a validation video.

A. Data Collection

Data collection is one of the most important aspects of training an effective machine learning model [76]. A model is only able to learn from the data it sees, so making sure the images the model is trained on are indicative of what the model will see in the application is vital.

1. Camera Setup and Specifications

The image capture setup was meant to replicate a production unit with a camera. The setup consists of a single camera at the threshold of the refrigerator door, shown in Figure 12.



Figure 12. Camera location in the refrigerator and corresponding field of view.

The camera in Figure 12 can view the produce bins when they are open, and the threshold of the refrigerator.

The Logitech C920 webcam was used for the experiments [77]. The Logitech webcam was selected because it has a USB connection, three-channel red-green-blue (RGB) image sensor, and 1080p resolution up to 30 frames-per-second (FPS) [78]. The mounting location of the webcam was chosen because the central location provided the best view of the produce bins, and the entrance to the refrigerator. An example image from the webcam is shown in Figure 13.



Figure 13. Example image from the webcam used for data collection.

Figure 13 shows that the camera gives a clear view of the opening of the refrigerator, and hopefully the best view of a user's hands entering and leaving the appliance. The Logitech Webcam Software Application Version 2.51, an application provided by the webcam manufacturer, was used to set the focus, frame rate, gain, contrast, brightness, and exposure parameters before each camera use [79]. It was important to manually set the parameters to ensure each frame is in focus and uniform from frame to frame. Frames with varying contrast, exposure, or other parameters could decrease the accuracy of the CV models. It may be possible to use OpenCV to manually set the camera parameters in the code [80]. The only illumination for the camera is the lighting already included within the unit. Additional lighting was not considered as the production setup was sufficient.

2. Collecting Images to Develop the Dataset

Image collection was done on both a Raspberry Pi B+ and a Dell laptop. For both the Pi and laptop, the webcam was connected via the USB port. The webcam video stream was accessed using Python and OpenCV [80]. The images were collected by saving video frames from the refrigerator video feed. Images were saved to a file at 30

frames per second, and an image size of 1920 by 1080. There was no difference in images collected by the Pi or the laptop, and the computers were used interchangeably depending on which was available at the time.

Initial image collection consisted of a single user unloading several typical produce items. “Typical produce” was determined from the USDA top produce shown in Figure 14.

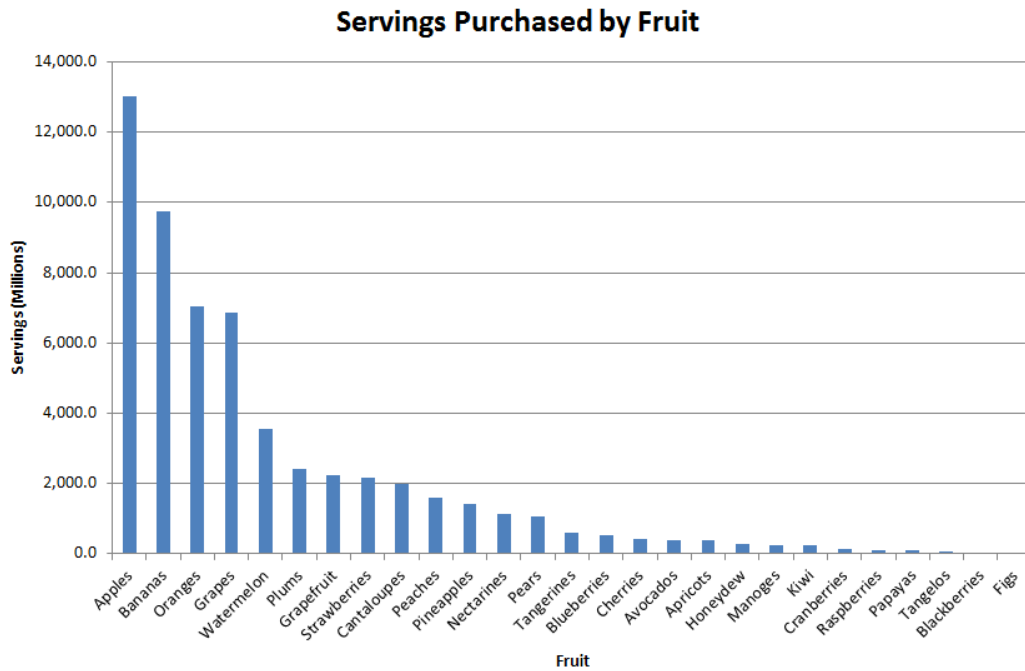


Figure 14. USDA Data [81].

Figure 14 shows that apples and oranges are the top refrigerated produce items purchased by US consumers. Apples and oranges were picked as the produce to be detected based on the data in Figure 14. The single user, a right-handed white male in his late-20’s, was used to simulate a consumer reaching in and out of the refrigerator. The user placed items in the refrigerator as he would when he unloaded his grocery bags, usually one item at a time. Once all items were in the unit, the user began taking items out of the produce bins

and putting them back in different locations in the refrigerator, such as the bottom shelf. The moving of items ensured a diversity of hand images holding different items.

3. Dataset Annotation

Annotation of the image dataset is the step that tells the computer vision model what objects are important, and characterizes what the model will be able to learn [82]. Rectangular bounding boxes with labels were used most often because the goal was to train a model for object detection. As much information as possible was included in the annotations. For example, left and right hands were distinguished, as were empty and non-empty hands. Produce items in bags were noted versus loose items. An example annotated image is shown in Figure 15 (the annotation labels have been overlaid for ease of reading).



Figure 15. Example of bounding box annotations for an image.

Figure 15 illustrates the many challenges posed by detecting object in the produce bins. Items are hidden in the back of the bin and under other items. The same items can be

bagged, cut up in containers, or placed in the bin individually. For this reason, as much information as possible was included in the annotations to ensure flexibility when training models in the future [56]. For example:

mfruit_rgrapes_blueberries_cantaloupe_bc_qc_h stands for mixed fruit of red grapes, blueberries, and cantaloupe, in a plastic container (bc), quantity is cut fruit (qc), and h for held in a hand. The annotation is flexible enough that it could be used to create a mixed fruit class, cut fruit, blueberries, or held items class. A complete list of items in the annotated dataset and an explanation of the annotation conventions can be found in Appendix B.

a. LabelImg

The image dataset was annotated using the open-source annotation software, LabelImg [83]. LabelImg was selected because it was free, easy to use, and recommended by multiple TensorFlow Object Detection API tutorials [84] [82]. LabelImg saves each image annotation in an XML file. Each file contains the image filename and image path, width, height, and depth of the image, or number of color channels, and the xmin, ymin, xmax, and ymax coordinates of the bounding box for each object in the image. The bounding box coordinates specify the top left and bottom right coordinates of the box, shown in Figure 16.

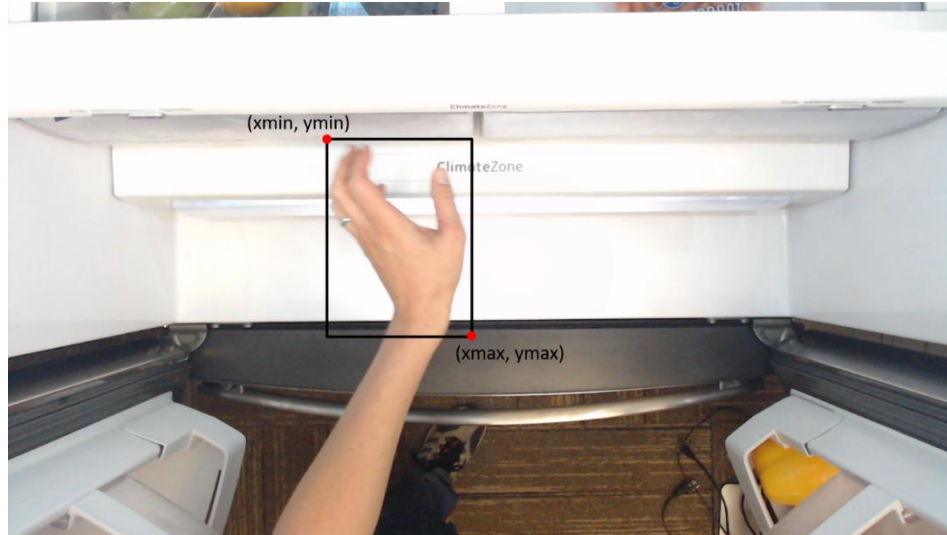


Figure 16. Bounding box coordinate convention.

The final LabelImg dataset consisted of 487 labeled images with 18 different item labels, see Appendix B for more details. The images were from a single user unloading a bag of groceries. Experiments were tried using all the object labels, just hands, just hands and arms, hands, arms, and other objects combined into a single object class, left hand/right hand classes, and empty hand/not empty hand. The results from selected training experiments can be found in the Results.

b. Anno-Mage

More hand annotations were made using Anno-Mage, a semi-automatic image annotation tool [85]. Anno-Mage uses the weights of a trained TensorFlow model to automatically produce bounding boxes on new images for the training dataset. The bounding boxes can then be manually adjusted to best fit the object. The Anno-Mage code was adapted to use a TensorFlow model trained on the data from Section 3 above. The code was further adapted to store all annotations in a csv file based on the format required by the TensorFlow Object Detection API [84].

326 more images from the dataset were annotated for the hand class using Anno-Mage. The final dataset, with both the Anno-Mage and LabelImg annotations, contained a total of 1,294 hand annotations (this number will not match with the data in Appendix B because during processing it was found some of the annotations were duplicates and were removed). The model was trained with a 75/25 train/test split for a total of 937 hands in the train set, and 312 in the test set. One image can contain multiple annotations, so care was taken to ensure all annotations for a single image were contained within either the train or test set and not split between the two, thus the split may not be exactly 25% [86].

4. EgoHands dataset

The EgoHands dataset is the largest publicly-available dataset of hand images at the time of this writing [67]. EgoHands consists of 15,053 bounding-box labelled hand annotations, compared to the prior Oxford hand dataset of 13,050 labeled hands [26]. The EgoHands dataset was created in 2015 by researchers at the Indiana University Computer Vision Lab. The dataset was collected from users wearing Google Glass smart glasses. The glasses were used to record 48 videos of the users (either one or two users per video) doing complex actions such as playing Jenga, chess, playing cards, or putting together a puzzle. The dataset consists of both bounding box and pixel-level segmentation of the hands, and are labeled as own or other hand and left or right hand. [67] Example annotated images from the dataset are shown in Figure 17.



Figure 17. Images from the EgoHands Dataset [87].

The paper that introduced the EgoHands dataset used a novel sliding window approach using the Caffe library to get an average precision of 0.807 for detecting hands in an image. In 2017, Victor Dibia used the EgoHands dataset along with the TensorFlow Object Detection API to get a 0.9686 average precision on hand images where the detection overlaps the ground-truth label by at least 50% [87]. The author Dibia provided code on his GitHub repository to convert the initial paper's Matlab code into python code; the python code was used in this research to convert the Matlab-encoded annotations into csv files. [87] The EgoHands dataset was used to supplement the training data for the hand detection model and improve accuracy through transfer learning [28].

B. Hand detection

The first step of the research was to use the annotated dataset to build a model that could accurately identify human hands within the refrigerator. A CNN was chosen for the task over other techniques because CNN have consistently performed better on image classification problems over other algorithms [14].

Initially, the research focused on creating a framework from scratch to classify and detect the hands. Detection is important as it provides information about where within the refrigerator the hand is, and presumably where an object is being placed. The

detection information can be used to track the hand, which is the second goal of this thesis. Detection was to be done by sliding a window over the image and running each cropped image section through the trained classifier [17]. Implementation of the sliding window method was slow and inaccurate. It became clear that, to stay within the time constraints of the thesis, making the detection framework from scratch was not a good use of time.

1. TensorFlow Object Detection API

The TensorFlow Object Detection API is a framework created by the open-source machine-learning platform TensorFlow [88]. The framework provides tools to implement many state-of-the-art object detectors like Faster R-CNN and Single-Shot Detector. A flowchart showing all the steps to prepare the data for the API is shown in Figure 18.

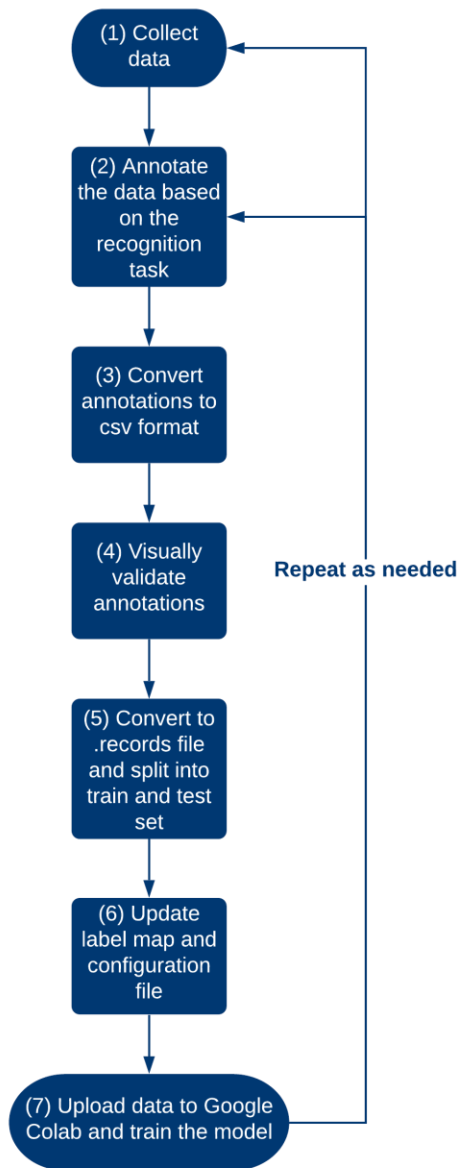


Figure 18. Flowchart for preparing data for the TensorFlow Object Detection API. As the flowchart illustrates, the first step is to collect and annotate the data (1) (2), as discussed in Section A. The images and annotation data were then converted into TFRecord files. The files are binary representations of the data which allow for quicker processing without taking up valuable memory space. The LabelImg XML annotation files were converted into a csv annotation file using a modified `xml_to_csv.py` (3) [89].

The code was modified to create a single csv file of all annotations, unlike the original code which creates a file for each the train and test datasets. The format of the csv is required to be “filename”, “image width”, “image height”, “class”, “xmin”, “ymin”, “xmax”, “ymax”. Filename is the entire path to where the image is stored, and is needed so the code can locate and open each image. A section of the csv file is shown in Figure 19.

	A	B	C	D	E	F	G	H
	filename	width	height	class	xmin	ymin	xmax	ymax
9	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_46_610834.png	1280	720	hand	418	613	602	720
0	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_46_967109.png	1280	720	hand	497	381	782	720
1	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_47_301019.png	1280	720	hand	563	656	743	720
2	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_48_643946.png	1280	720	hand	348	280	501	494
3	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_48_915105.png	1280	720	hand	280	201	458	399
4	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_49_324824.png	1280	720	hand	278	403	509	572
5	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_49_596368.png	1280	720	hand	293	483	496	628
6	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_49_954820.png	1280	720	hand	312	524	479	674
7	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_50_271636.png	1280	720	hand	303	547	506	719
8	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_51_386971.png	1280	720	hand	456	417	591	554
9	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_51_632900.png	1280	720	hand	423	344	554	504
0	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_52_251762.png	1280	720	hand	382	274	496	447
1	C:\Users\212488697\Desktop\OneDrive - GE Appliances\data\images\2019_08_12_10_29_52_251762.png	1280	720	hand	61	607	309	720

Figure 19. Example rows from the csv file used to convert the dataset to the format required for the TensorFlow Object Detection API

The figure above shows the csv file with the file location for each image, image size, class information, and bounding box coordinates. The annotations from Anno-Mage were pasted into the file with the LabelImg annotations. For early experiments, generate_tfrecord.py, provided by TensorFlow, was used to convert the images and annotations from the csv file into the TFRecords .record file (4) [82]. The Python code scales each bounding box for an image by the image size so that all values are between zero and one. Scaling the bounding boxes normalizes the values and allows the boxes to be used at any scale or aspect ratio of the image [86]. The image pixels are sorted as an array, along with all bounding box and class labels for that image. All information in the .records file is numerical; a label map file must be provided to map the class string to an integer (6). An example label map is shown in Figure 20.

```
item {
  id: 1
  name: 'hand'
}

item {
  id: 2
  name: 'arm'
}

item {
  id: 3
  name: 'mfruit'
}

item {
  id: 4
  name: 'apple'
}

item {
  id: 5
  name: 'orange'
}
```

Figure 20. Example label map to use for the TensorFlow Object Detection API.

The code uses the file above to map the class *hand* to be represented by 1, *arm* by 2, and so on. The label map is created automatically when running the `xml_to_csv.py` file. The code needs a separate csv file for the train and test images, and from them, two `.record` files are created, one for the training set and one for the test set. For this research, a 25% test split was used [17]. Later experiments used a modified `build_lisa_records.py` file [86]. The code produces the same output `.records` files as `generate_tfrecord.py`, but uses a single csv annotation file and splits the data into train and test sets within the code. The `build_lisa_records.py` also displays a subset of images plus their bounding boxes and class information to ensure the annotations are correct (4). All images and annotations were visually verified before writing the information to the `.records` files. Duplicate bounding boxes and mislabeled items were fixed or discarded. Visually validating the dataset is essential to creating a robust object recognition model [86]. Lastly (6), a

configuration file was generated to tell the API which model to use, where the TFRecord files are located in the file structure, how many classes, and other important training parameters. Each pre-trained model provided by the API has different configuration files. An example configuration file highlighted to show which lines need to be changed can be found in Appendix C. The configuration file also allows for data augmentation to be applied, although for this research no additional data augmentation was used. Because the configuration files use model parameters that have already been optimized for top performance, no parameters were changed except for those necessary for the API to function. The model parameters in the configuration file (learning rate, loss function, etc.) have also been optimized for top performance based on the various state-of-the-art models, and thus were not be altered for this experiment. [90] Figure 21 shows an example of the API code running in Google Colab.

```

!python /content/models/research/object_detection/model_main.py \
--pipeline_config_path={pipeline_fname} \
--model_dir={model_dir} \
--alsologtostderr \
--num_train_steps={num_steps} \
--sample_1_of_n_eval_examples=1

index created!
...
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=12.33s).
Accumulating evaluation results...
DONE (t=0.98s).
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.695
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.966
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.851
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.040
Average Precision (AP) @[ IoU=0.50:0.95 | area= medium | maxDets=100 ] = 0.374
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.726
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.258
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.742
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.749
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.158
Average Recall (AR) @[ IoU=0.50:0.95 | area= medium | maxDets=100 ] = 0.511
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.777
INFO:tensorflow:Finished evaluation at 2020-04-13-14:36:19
I0413 14:36:19.838330 140638190106496 evaluation.py:275] Finished evaluation at 2020-04-13-14:36:19
INFO:tensorflow:Saving dict for global step 5589: DetectionBoxes_Precision/mAP = 0.6950702, DetectionBoxes_Precision
I0413 14:36:19.838590 140638190106496 estimator.py:2049] Saving dict for global step 5589: DetectionBoxes_Precision
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 5589: /Object-Detection/training/model.ckpt-5589
I0413 14:36:19.848144 140638190106496 estimator.py:2109] Saving 'checkpoint_path' summary for global step 5589: /Ob
INFO:tensorflow:global_step/sec: 0.466979

```

Figure 21. Training the model using TensorFlow Object Detect API in Google Colab. Once all files are updated with the proper information, the command shown in Figure 21 can be run to train the model (7). The code for training was modified from code in Chengwei Zhang’s GitHub Repository [91] [92].

Multiple experiments were done to find the best model for the hand detector. The first task was to determine which of the model architectures from the TensorFlow model zoo was best-suited for the task [92]. The most important metrics for selecting a model were inference speed and mean average precision, or mAP. See Section 3 below for more information on mAP. The size of the model (amount of bytes for the model weights) was also a consideration, but less so than speed and mAP. The top three candidates are shown in Table 1.

Table 1. TensorFlow detection model zoo metrics [92].

Model name	Speed (ms)	mAP on COCO dataset
ssd_mobilenet_v2_coco	31	22
faster_rcnn_inception_v2_coco	58	28
rfcn_resnet101_coco	92	30

The three models in Table 1 were selected because they were the most common models used in the literature, and offer a good tradeoff between speed and accuracy [88].

2. Python Environments

All code was written in the Python language. Python is ideal because it comes with many libraries optimized for image processing and matrix multiplication and manipulation [93]. Many of the machine learning frameworks are written for Python.

Google Colaboratory (Colab) is an open-source, web-based platform for writing and executing Python. Colab environments come pre-installed with many machine learning specific libraries like TensorFlow and Keras, making it ideal for quickly building and testing various machine learning models. The greatest advantage of Colab is that it allows the code to be run in the cloud on a graphical processing unit (GPU). [31] GPUs can train models in hours versus days required if using a standard central processing unit (CPU) [94]. All models in this paper were trained using a Colab environment with a GPU hardware accelerator, running Python 3, TensorFlow 1.15.0, and Numpy 1.17. [31]

The Scientific Python Development Environment (Spyder) for Windows 10 is used for all other development, from testing the trained models to creating the algorithms for hand analysis. Spyder version is 3.3.6, running Python 3.6.10, TensorFlow 1.15.0, Numpy version 1.18.1. [95] The laptop is running Windows 10 Pro, 24 GB RAM, 64-bit operating system.

OpenCV is an open source computer vision library that can be used to load, display, and write image files, as well as contain functions to implement hundreds of image processing and computer vision algorithms. OpenCV was used extensively in this thesis to display, read, write, and manipulate images. Many of the algorithms in OpenCV were tested and utilized in various parts of the research. OpenCV loads all images as blue-green-red (BGR) as opposed to RGB, so all images and frames in the research were converted to RGB before any further processing was done. OpenCV version 4.1.2 was used. [96]

3. Metrics for Evaluating the Models and Algorithms

Precision and recall are common metrics for measuring the quality of an image classifier, and can be adapted to evaluate an object detector as well [97]. Precision is the ability of the model to correctly detect the desired object in an image and is defined mathematically as:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (1)$$

where TP is true positive, which is a bounding box that correctly overlaps a ground-truth bounding box of the same class, and FP is false positive, which is a bounding box that incorrectly detects an object that is not there [97]. Recall is the ability of the model to detect all of the desired objects in an image, and is defined as:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2)$$

where FN is false negative, which is a desired object that is not detected by the model. TN or true negative, not in either equation, is a portion of an image that is correctly not

identified as an object. [98] Recall and precision can be combined into a single metric called the F1-Score, calculated using equation (3).

$$\text{F1-Score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (3)$$

With object detection, there is one more factor necessary to quantify the ability of the model: how well the predicted bounding boxes match the ground-truth boxes.

Intersection over union (IoU) is used to determine how much a detection needs to overlap the ground-truth box to be considered a true positive. Intersection is the area where the predicted box and the ground-truth box overlap, and union is the entire area encompassed by both boxes. [99] A visual example plus the IoU equation is shown in Figure 22.


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figure 22. Intersection over union (IoU) is an important metric for evaluating an object detection model [99].

The IoU calculation is used to determine a true positive versus a false positive. IoU is also used in training as part of the loss function for refining the bounding box. An IoU of 0.5 is commonly considered a good prediction [99]. A visualization of TP, FP, and FN and IoU is shown in Figure 23.

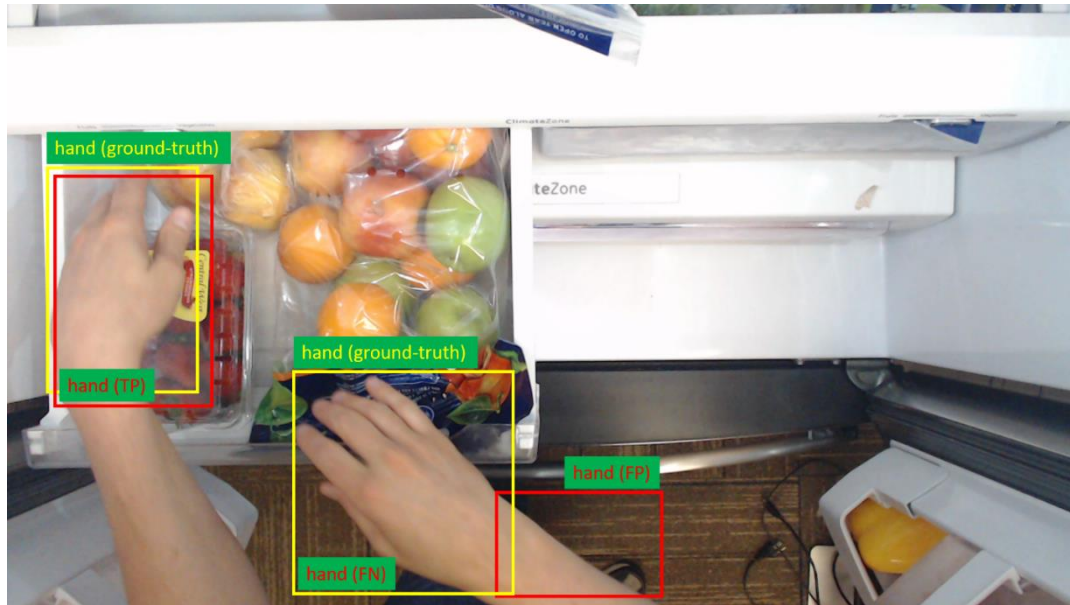


Figure 23. Image showing visualization of TP, FP, FN, and IOU. Red boxes are what the model predicts, yellow are ground-truth labels.

For the left hand in Figure 23, the model prediction overlaps the ground-truth by an IoU greater than 50% making it a true positive prediction. The right hand prediction overlaps the ground-truth by less than 50%, making the prediction a false positive. The ground-truth for the right hand is a false negative because it is not properly detected. For the predictions in Figure 23, $TP = 1$, $FP = 1$, and $FN = 1$. Precision would be $1 / 2$ or 0.5, and recall would be $1 / 2$, also 0.5. The F1-score would be 0.5 as well.

The standard metrics used to evaluate an object detection model is average precision, AP, (or mean average precision, mAP, when a model is detecting multiple classes) which is the average precision and recall values varied over a confidence interval. AP is the most common metric used by benchmark datasets like MS COCO and Pascal VOC to evaluate object detection models [100] [101] [97]. To determine the average precision, a precision-recall curve is created by varying the confidence threshold of the model. Once the curve is created, the area under the curve is the average precision.

The TensorFlow Object Detection API calculates the mAP automatically from the data in the test set. The API defaults to the COCO evaluation protocol, so that is what is used in this research to evaluate the models during training steps. The COCO protocol gives mAP at IoUs from 0.05 to 0.95, and precision and recall values for small, medium, and large detections [100]. mAP@0.5IoU expresses how well the model predicts bounding boxes that have an IOU of at least 50% with the ground-truth. The mAP metrics for the data in this thesis are presented in the Results.

The overall accuracy of the hand detector, image classifier, and add and remove logic was calculated using equation (4).

$$\text{Accuracy} = \frac{\text{total correct predictions per class}}{\text{total number of ground – truths per class}} \times 100\% \quad (4)$$

Equation (4) gives the ratio of correct predictions to the ground-truth labels, and is used to give information on the quality of each model and logic component applied to images not in the training set [102].

4. Model Training and Real-Time Metrics

TensorBoard is TensorFlow’s machine learning metric and visualization toolkit. TensorBoard automatically calculates and provides real-time metrics like loss and mAP, and displays images of ground-truth versus what the model is detecting at a certain learning step. TensorBoard was used alongside the API’s mAP metrics to observe the progress of model training. [103] Model training was stopped when the overall loss or mAP values stagnated for consecutive model weight update steps, or epochs. An epoch is complete once all training images of the dataset have passed through the model layers

and the losses have been propagated back to update the model weights. On average, stagnation occurred around 10,000 steps [17].

5. Hand Detector Experiments

Multiple experiments were done to develop the best hand detector for the task. The first experiment was to compare the accuracy of a model only trained using the dataset collected in Section A, versus one first trained on the EgoHands dataset and then trained on the dataset from Section A. Both experiments started with pre-trained weights from the COCO dataset. Next, the annotations were divided into different classes to see what worked best. The experiments for the hand detector are shown in Table 2.

Table 2. Hand detector experiments and corresponding classes.

Model Task	Classes
Detect all hands in an image	Hand
Detect all hands and arms	Hand, arm
Detect all hands and differentiate between left and right	Left hand, right hand

Because of the way the annotations were structured, creating the classes in Table 2 involved simply searching through the annotation strings for keywords like lhand for left hand, or hand if using a single hand class. The modified `xml_to_csv.py` code used the desired class keywords to write to the csv file all bounding boxes corresponding to the keywords, and ignore all other annotations and bounding boxes. The results of the hand detector experiments can be found in the Results.

Finally, to verify the model in a production setting, a video was recorded of a user adding and removing produce from the refrigerator. The user was a left-handed, mid-30's white female. The hand detection models were run on each frame, and through visual inspection, TP, FP, and FN were tallied for the entire video. An IoU calculation was not

implemented for validation; the author used her best judgement to visually validate that a predicted bounding box overlapped a hand by at least 50%. A confusion matrix of the precision and recall values will be reported. Visually inspecting these values is beneficial in seeing how and where the model is making mistakes. Also, for the tracking and analysis sections, it is more important to reduce false negatives than false positives because missing a hand means missing valuable data, whereas falsely identifying a hand can usually be ignored in the code. AP calculations do not show that distinction. A confusion matrix is not typically used for object detection metrics, but is mentioned as a good additional metric for ensuring robust performance in production applications [104].

C. Hand tracking

The second portion of the research involved developing an algorithm to track the hand as it moved through the refrigerator.

1. Centroid Tracking

The first tracking method used the center, or centroid, of the bounding boxes from the hand detector. The centroid of the bounding boxes was calculated using:

$$cX = x_{\min} + \frac{x_{\max} - x_{\min}}{2} \quad (5)$$

$$cY = y_{\min} + \frac{y_{\max} - y_{\min}}{2} \quad (6)$$

where cX and cY are the respective x and y centroid coordinates, x_{\min} and y_{\min} are the upper left coordinates of the bounding box, and x_{\max} and y_{\max} are the lower right coordinates of the bounding box, see Figure 16. Bounding box coordinate convention. The centroid for each frame is calculated and stored in an array. For each subsequent

frame, the Euclidean distance is calculated between the old and new centroids, using equation (7).

$$\text{dist}((cX_{prev}, cY_{prev}), (cX_{new}, cY_{new})) = \sqrt{(cX_{prev} - cX_{new})^2 + (cY_{prev} - cY_{new})^2} \quad (7)$$

If the distance is below some set minimum distance parameter, the new centroid is assumed to belong to the same object as the old centroid. By calculating the differences from frame to frame, the hand can be tracked throughout the refrigerator. If there are multiple bounding boxes from frame to frame, the centroids with the smallest Euclidean distances are assumed to be the same object. The code used for centroid tracking is modified from Adrian Rosebrock's blog, which is an online repository of computer vision tutorials and open-source software. [105] Limitations to this approach are the tracking is dependent on a good detection model to constantly feed in new bounding boxes. Also, running the detector on every frame can be computationally expensive. Another limitation is when more than one hand is in the frame. If the centroids of the different hands are too close or cross, the algorithm is unable to differentiate between the hands.

2. dlib Correlation Tracker

To address the limitation posed by the computationally-expensive process of running the hand detector on every frame, a tracking algorithm within the image processing library dlib was tested [106]. The correlation tracking algorithm implemented in dlib is based on the 2014 paper, Accurate Scale Estimation for Robust Visual Tracking. The paper uses scale pyramids to estimate the scale and track an object as it changes throughout a scene. The paper shows that the correlation tracker is much faster than running a detector on every frame [107] The tracker within dlib is initiated by

passing in the initial bounding box of the hand. The tracker then tracks the hand throughout subsequent frames, automatically producing new bounding boxes. The hand detector runs periodically to validate the tracker and detect other objects that have moved in or out of the frame. The dlib tracker was implemented using code modified from Rosebrock's online tutorial. [108]

3. Determining Direction of Movement

Knowing if the hand is moving in or out of the appliance is important for knowing if an object is being added or removed from the refrigerator inventory. Determining the direction the hand was moving, whether in or out, was found using information from the centroid array. Direction was determined from the sign of the delta between the old and new centroid y-coordinate, with a positive delta signifying moving out and negative moving in. The equation is shown in (8).

$$dY = cY_{\text{new}} - cY_{\text{prev}} \quad (8)$$

A visualization is shown in Figure 24.

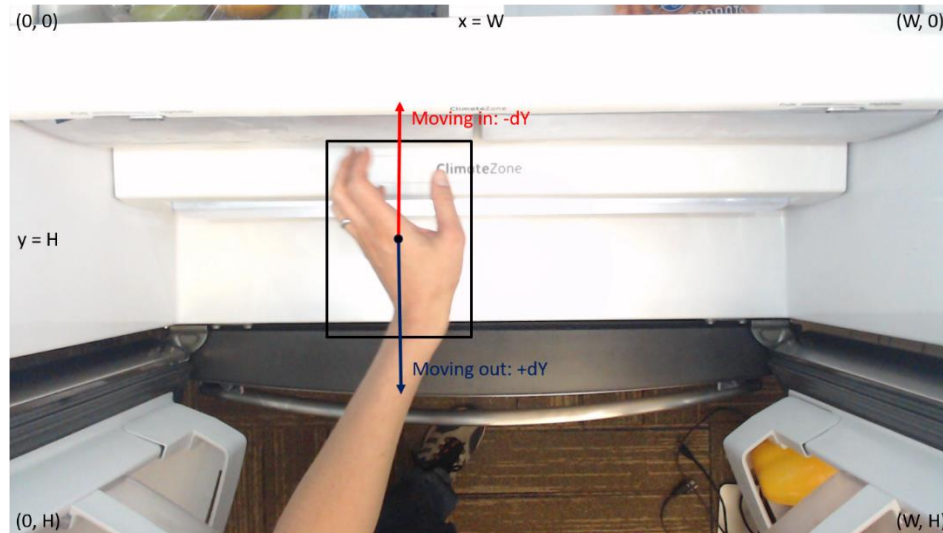


Figure 24. Direction is determined by looking at the sign of the delta between the y centroid of the current frame versus that of the previous frame. W is the total width of the image frame, and H is the total height.

Figure 24 shows how direction is determined. The origin for the image is at the upper left corner, so a negative dY means the hand is moving in to the refrigerator, while a positive dY means the hand is moving out. The code for determining direction was based on Rosebrock's online tutorial [108]. The code was modified to use only the previous centroid y value, whereas the original code used the average of all prior centroid y values. The original code used the average y value to make the code more robust, but for the refrigerator case where hands are changing direction quickly, looking at the last y centroid gave better results than an average.

4. Distinguishing Between Hands

Distinguishing between different hands is important for knowing what items are added or removed from the refrigerator. One hand can add an item, while at the same time the other hand removes one. Being able to correctly attribute the actions to the

corresponding hand is necessary for accuracy. However, distinguishing between different hands in the refrigerator proved to be not a trivial problem. The centroid tracker used the distance between centroids to distinguish between different hands, but failed if hands crossed over one another or if both hands were close. Ideas for distinguishing between hands include:

- Assume a left hand will most likely be on the left side of the screen, and right on the right side, and use that information to hard code rules to distinguish hands
- Train a model to learn left versus right hands

The different techniques were implemented in the code, and then observed by playing back a video to visually observe performance. No technique performed well, therefore it was decided to limit the research to observing a single hand to ensure the thesis could be completed within the given timeframe.

5. Determining if a Hand is Inside the Refrigerator

Knowing if a hand is inside the refrigerator is important not only to get information on where and what the hand is doing, but also to decide when an interaction, or single add or remove event, has started and ended. Defined interactions drive the logic to update the inventory list. Experiments to determine if the hand was in the refrigerator include:

- Assume if there is no hand in the frame then a hand is not within the appliance
- Assume if there is no hand or arm in the frame then a hand is not within the appliance
- Use centroid and direction info to set in and out boundaries. For example, if the hand leaves the frame, but the last centroid coordinate was near the top of the

frame and moving in, assume the hand is still inside the refrigerator. Likewise, if the last centroid crosses the coordinates of the bottom of the frame and is moving out, assume that the interaction is over and the hand has left the refrigerator if the next few frames do not show a hand

- Use the door open and close as the trigger for an interaction
- Use a motion sensor to detect if a hand is inside the appliance

The different techniques were implemented in the code, and then observed by playing back a video to visually observe performance. Detecting multiple hands entering and leaving the refrigerator increases complexity, and was decided to be outside of the scope of this research. Only a single hand was used to develop the logic for determining if a hand is inside the appliance.

D. Hand-Object Interaction

The final objective of the thesis was to determine what the hand was holding, specifically if it was empty or holding a produce item. Analyzing hand-object interactions in the refrigerator is beneficial mostly if one is able to detect what the hand is holding. In order to limit the scope of the thesis, detecting an empty hand versus holding an apple or orange is studied as a proof of concept. Determining if a hand is not empty without identifying what it is holding was also an area of research. The advantage of knowing a hand is holding an unidentified object is unclear, but the information may be useful in future projects. The ability to scale the solution to detect all possible objects was considered, and is discussed in the Future Works section. The information from the hand-object interaction will be used to determine what item was added or removed from the refrigerator.

1. Hand Holding Item or Not, using TensorFlow Model

The first experiment to automatically detect an empty or not empty hand involved training the TensorFlow Object Detection API on images of hands holding things versus empty. The experiment was quick to implement as the dataset was already collected and annotated. The process was the same as the process of training the hand detection model, but the data was modified to learn to distinguish empty and non-empty hands. The initial annotations only noted if an object was being held, not if a hand was empty or not. A GUI was developed using the Python GUI toolkit PyQt to quickly re-annotate the hand bounding boxes [109]. Some of the GUI code was based on code from Chang Luo's website [110]. The GUI is shown in Figure 25.



Figure 25. GUI to create holding/not holding annotations.

Each image was displayed in the GUI, as seen in Figure 25, with the corresponding bounding boxes. The status of each hand bounding box was initially set to holding an item, but could be changed to not holding item by clicking on the corresponding button

above the image. In Figure 25, Hand 1 has been unchanged and is set to holding an item, while the button for Hand 2 has been pressed to change to not holding an item. Pressing the Next button saves the current image filename, bounding box coordinates, and updated classes to a new csv file. Pressing Skip skips the image, and is used in the case of an image with no hands present.

2. Hand/Object Segmentation

The next experiment for hand-object interaction was to try and extract the foreground, presumably the hand and object, from the background. The camera only produces 2-dimensional images, so there is no way to visually determine the depth of the objects in the frame. Extracting the foreground from the background would be useful when dealing with instances where the hand is over a produce bin like the right hand in Figure 25. In the figure it is difficult to tell if the hand is holding the item or if the item is within the bin.

a. OpenCV's GrabCut

GrabCut is an interactive foreground extraction algorithm included in OpenCV [111]. GrabCut was designed by Carsten Rother, Vladimir Kolmogorov and Andrew Blake from Microsoft Research Cambridge, UK. and is based on their paper, "GrabCut": interactive foreground extraction using iterated graph cuts [112]. The GrabCut algorithm takes a rectangle specifying the foreground object to extract, and considers everything outside of the rectangle to be background. An example image and rectangle is shown in Figure 26, plus the output of the algorithm.



Figure 26. Left: The input image and blue rectangle provided to the program to designate the foreground. Right: The output of the algorithm. [111]

Figure 26 shows the input image and blue rectangle on the left, and corresponding output image on the right. GrabCut uses a Gaussian Mixing Model to create color models for the foreground and background pixels, and uses an optimized loss function to split the foreground and background boundaries [112]. For this research, the bounding box from the hand detector was passed to the algorithm to specify the foreground.

b. OpenCV's Background Subtractor MOG2

Another approach to segment the hand and object from the background was to use the OpenCV BackgroundSubtractorMOG2 class [113]. The algorithm works by building up a buffer of images from a video stream. From these images, a model of the static background is created. The model is created by using a Gaussian distribution to measure how long a background pixel color stays in the frame. If the colors stay in the frame longer, they are assumed to be background. Using a stream of images allows the model to update with the slight variations in lighting and shadow which change constantly over time. The model is then applied to further images, updating the background model as well

as displaying anything that is in the foreground. Parameters such as how many past images the background model uses to create the model, and how strictly the background model is applied to new images allow for fine-tuning the algorithm's performance on the specific use case. [114] An example input and output image is shown in Figure 27.

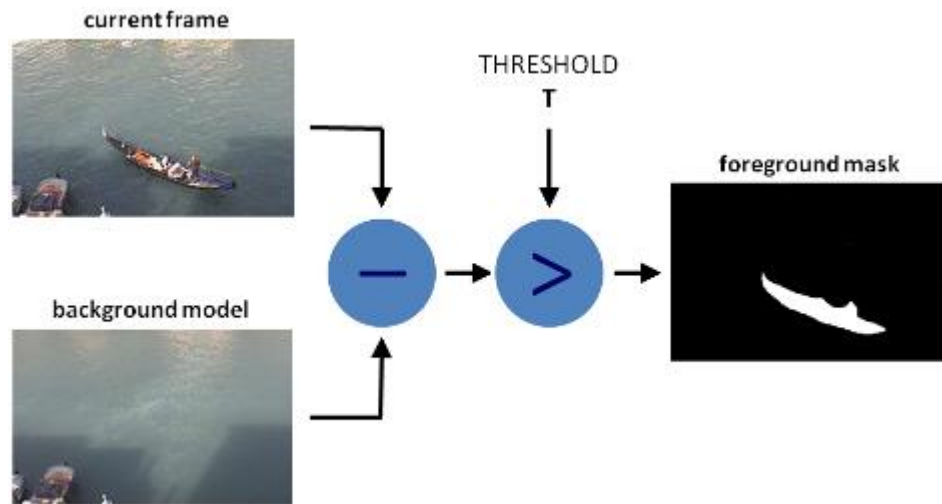


Figure 27. Background subtraction works by developing a background model from prior frames, and then comparing that model to the current frame. The pixels that are different from the background model, with respect to a specified threshold, are considered foreground. The mask is created by setting all pixels above the threshold to white, and all others to black. [113]

Figure 27 shows an example background model and current frame. The pixels that are different enough from the background model are set to white in the foreground mask, and all others are set to black. The mask can be applied to the current frame to extract the foreground, which is the boat in the above example. Experiments were tried to find the best value for how many prior images to use to create the background model, and what was a good threshold value.

c. Color Thresholding for Skin Segmentation

Color thresholding was another technique tried to extract the foreground from the background. Color thresholding involves creating color ranges, and then using those ranges to determine which pixels to keep. A mask can be created, similar to the foreground mask in Figure 27, where the white area could be the pixels that were within the threshold. Applying the mask to the original image would extract all pixels within the color range. [115]

d. Color Spaces

RGB is the most common color space, and is used for most digital images and displays. RGB is an additive color model, where different amounts of red, green, and blue can be added to create different colors. The problem with RGB for color thresholding is that under different lighting, the RGB value will change even for the same color. Hue (H), saturation (S), and value (V), or HSV, separates the illumination portion of a color's appearance to the value variable. Thus, different lighting conditions will impact the value, but hue and saturation will remain the same for the same color. [116] The HSV color space was used for the thresholding experiments. Converting to HSV is a straightforward computation, and can be done in OpenCV using the function `cvtColor(image, cv2.COLOR_BGR2HSV)` [117].

e. Thresholding Implementation

Thresholding experiments were done using a video stream of hands, both empty and holding things, moving in and out of the refrigerator. Experiments were done both on the entire image, and only on the portion of the image within the bounding box detected by the hand detector. Before applying the threshold, the images were normalized so that the 0 to 255 pixel values became between zero and one. Normalizing ensures a high pixel

value does not exert undue influence just because it is a large number. Normalizing was done using OpenCV's `normalize()` function. [96] Gaussian blur was applied to the image to reduce noise in the final image. The Gaussian blur uses a kernel like those used in convolutions. The kernel blurs the boundaries between pixels, which helps reduce the influence of large variations in pixel values that would otherwise cause noise. [118] Figure 28 shows an image before and after the Gaussian blur is applied.

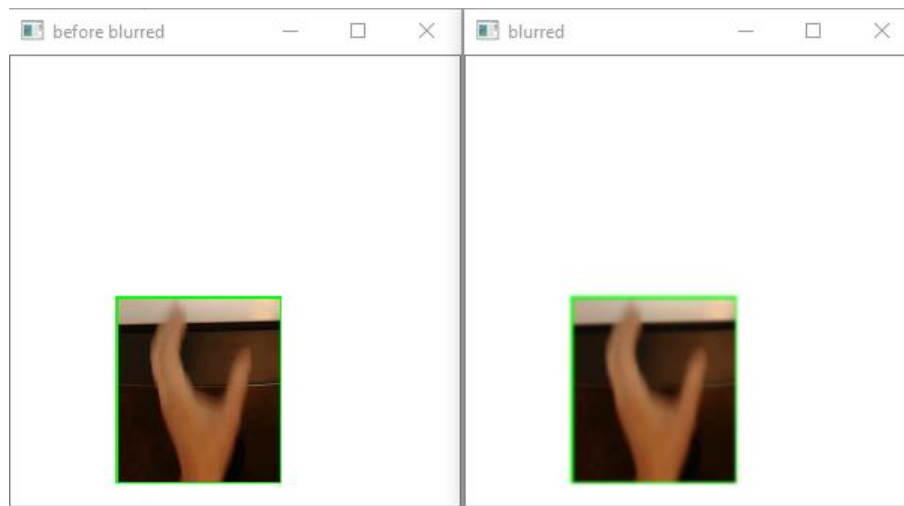


Figure 28. Left: Image before the Gaussian blur is applied. Right: After the image is blurred.

Figure 28 shows that before the blur is applied (left), the image appears crisper with sharper edges around the hand due to the movement. The blurred image (right) is much smoother around the edges, and produces a more uniform thresholded image. [119] The image was then converted from the standard OpenCV color space, BGR, to HSV. A skin mask was created by setting all pixels within the skin threshold range to white, and all pixels outside of the range to black. An object mask was created using the same technique above, just with a different threshold range. Each mask was combined with the original cropped image to create a segmented image of the hand and the object.

3. Hand Analysis from Segmented Images

Multiple techniques were tried to extract useful information from the segmented images. One experiment used the K Nearest Neighbor (KNN) algorithm on the extracted foreground regions to see if it could determine if a hand was empty or not, based on color clusters. KNN was implemented using the Python machine learning library scikit-learn. The function `MiniBatchKMeans` clusters similar pixels into K different clusters by color [120]. The center coordinates of the clusters can be extracted, and used to distinguish between different colors in an image. Possible ways to analyze the extracted foreground images include:

- Use a single cluster (plus a background all black cluster) and assume a hand would give one average color value, while a hand holding an object would give a different value
- Train a classifier on the segmented images – using a dataset with segmented fruit to easily scale up the classifier
- Create histogram templates to classify hands and objects

4. Video for Testing

Most of the experiments, models, and logic were tested and validated on videos recorded within the refrigerator. The video sequences were designed to mimic how a typical user would interact with their refrigerator. To collect videos, a program was developed in Spyder using OpenCV that accesses the webcam stream. The code was based on a tutorial provided by OpenCV [121]. The program records video until the user ends the program. The video is saved in .avi format to a folder on the author's laptop. The dimensions of the frames in the video were determined by the capabilities of the webcam,

and was usually 640 by 480 [77]. The final validation video consisted of 14 complete hand in and out interactions, with 301 frames with hands and/or objects present.

Frames per second (fps) was found to be an important variable for the performance of the entire system. The webcam defaults to 30 fps, and experiments were done to determine the ideal frame rate for the application. The frame rate needed to be fast enough to catch several frames of both the hand moving in and out of the refrigerator, but also balance disk memory and the capacity of the camera. The fps code was modified from the top response from a Stack Overflow post, and works by only saving video frames at certain time intervals to produce the desired fps [122].

Determining the ideal frame rate was done by visually inspecting data at various frame rates, and picking the rate that balanced having enough data for the add/remove algorithm, with not so much data that it took up too much memory.

To reduce the complexity and variables so the research could be completed in the given time frame, the following assumptions were made:

- Test videos will be of a single hand moving in and out of the refrigerator
- The hand will hold a single item
- One item can be added, and another can be removed during a single interaction.
- An interaction is defined as the entire period for a hand to move into the refrigerator, add or remove an item, and exit the refrigerator
- Other than the few produce items added into the compartment, the refrigerator will be empty
- Closing and opening the refrigerator door between interactions has no impact on the experiment

During playback for code development, sections of video when the door is open or closed are skipped over by skipping all frames where the average color of all pixels is less than a set threshold. When the door is closed, all pixels in the frame are black or close to black, so the threshold value is set to know when a majority of the pixels are black. To speed up development, frames between interactions where no hand was present were skipped. The frames were skipped by using a counter to count the number of frames, then using an if statement to only run the detection code if the frame number was within the range of frames with interactions.

5. Image Classifier using CNN

Another technique that was used to understand the hand-object interaction was to pass the cropped image from the hand detector to an object classifier. The model was trained using the Keras VGG16 model architecture mentioned in Figure 8.

a. Dataset Collection

The dataset for training the image classifier is entirely separate from the dataset for the object detector. The training images were gathered using the webcam in the refrigerator and a program in Spyder. When the hand detector detected a hand in the webcam frame, it cropped the hand image, added an offset of ten pixels to ensure the entire hand was captured, and saved the file to a local folder. Using this method, hundreds of images were gathered in a matter of minutes. Because the classifier only needs a label with the image and not a bounding box, sorting the images into the different classes was as straightforward as looking through the image folder and moving the empty hand images into an empty folder, and the apple images into an apple folder. [17]

Example hand and apple training image are shown in Figure 29.



Figure 29. Training images for the object classifier. The left image is an example in the class for Empty and the right is an example from the Apple class.

Figure 29 shows images used to train the image classifier. The image on the left is an example from the empty class, and the right is an example from the apple class. Three different users, one right-handed male in his early 20's, one left-handed female in her early 20's, and one left-handed female in her early 30's, with three different skin tones were used to build the dataset. The hand detector was accurate at detecting all three skin tones. Building the dataset with multiple users ensures the model will be robust at identifying different users and ways of holding produce items. Because it was so quick and straight-forward to add a new class, an orange class was added to test how well the model could differentiate between produce items. The number of images in the final dataset can be found in Appendix B.

b. Preprocessing the Dataset

All preprocessing and training was done in Google Colab, with the dataset stored within Google Drive so that the images could be accessed within Google Colab. The code for preprocessing and training was modified from Adrian Rosebrock's book [17]. A flowchart of the process from collecting data to training the model is shown in Figure 30.

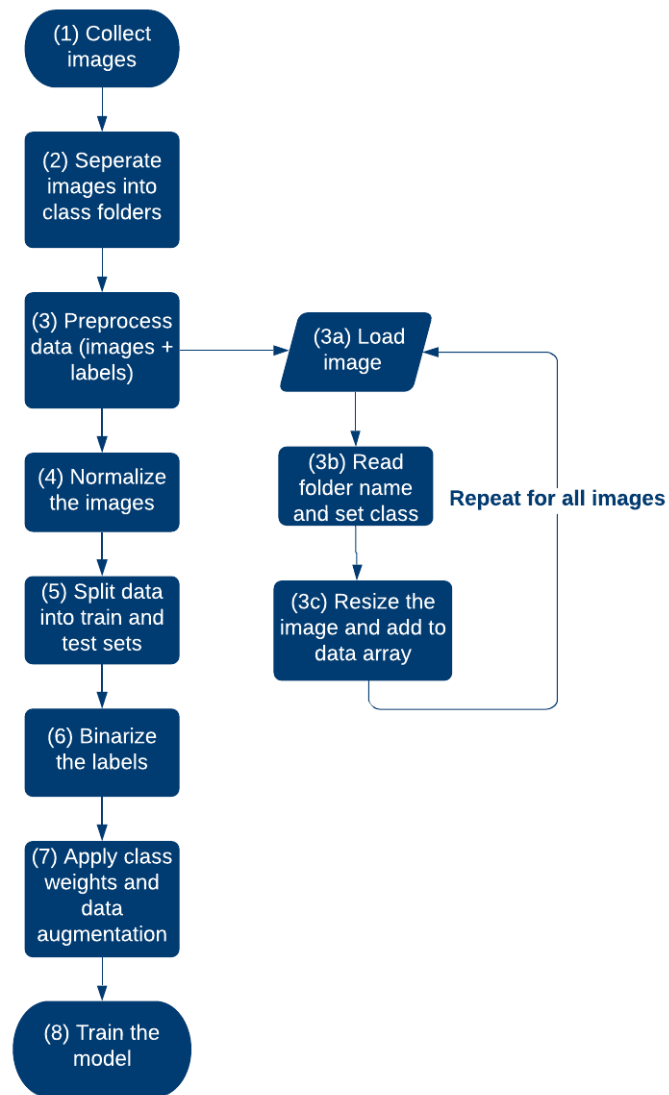


Figure 30. A flowchart showing the process of collecting data to training the classifier model.

Figure 30 shows that the first step, after collecting the images, was to read in the images and their file paths (3a). The file structure of the dataset, shown in Figure 31, was done in a way so that the class could be set by reading in the folder name (3b).

```

|--- Images
|   |--- Apple
|   |   |--- 00000000.png
|   |   |--- 00000001.png
|   |
|   |...
|   |   |--- 00000448.png
|   |   |--- Empty
|   |   |--- 00000000.png
|   |   |--- 00000001.png
|   |
|   |...
|   |   |--- 00000564.png
|   |   |--- Orange
|   |   |--- 00000000.png
|   |   |--- 00000001.png
|   |
|   |...
|   |   |--- 00000314.png

```

Figure 31. Folder structure for image dataset within Google Drive.

The code begins at the highest folder level (“Images”), and iterates through each subfolder. Each image is resized to 224 x 224 with the aspect ratio maintained (3c), and then converted to an array using the Keras `img_to_array` function [123]. Next, each image array is appended to a data list that will eventually store all images in the dataset. Similarly, the class, determined by the subfolder name (ex. “Apple”), is appended to a list of labels. Once all images and labels are added to their respective lists, the images are normalized by dividing each image pixel by 255 to ensure each pixel value is between zero and one (4). Normalizing the pixel values ensures a pixel with a large value does not have a greater contribution than a smaller pixel value. [17] Once all images had been read in, the data was split into train and test (25% test split) sets using scikit-learn’s `train_test_split` function (5). The `train_test_split` function randomly splits the data and labels into the train and test subsets, but ensures the data stays matched with the corresponding label. [124] The data labels were converted from strings to binary values using scikit-learn’s `LabelBinarizer` function (6) [125]. For example, after the function, the orange label was expressed as [0, 0, 1], empty is [0, 1, 0], and apple is [1, 0, 0]. The

classes in the dataset were slightly imbalanced, with both the Apple and Empty class having over 100 more images than the orange class. A class imbalance could cause the model to skew towards predicting one class more than the others just because the model sees it more often in training. The imbalance was dealt with by dividing the total images in the largest class (“Empty”) by the total images in each other class (7). [17] The equation is shown in equation (9):

for each class:

$$\text{class weight} = \frac{\text{max of largest class}}{\text{class total for class in question}} \quad (9)$$

For example, based on the data in Table 3, the max of largest class is the Empty class with 455 images in the training set. The total images in the Apple class are 361, so the class weight for the Apple class would be $455/361 = 1.26$. The final class weight breakdown is shown below:

Table 3. Class weights for each class to ensure a balanced dataset.

Class	Total images per class in training set	Balanced Class Weight
Apple	361	1.26
Empty	455	1.00
Orange	236	1.93

Keras automatically weights each class according to the values provided by passing the values in Table 3 to the `class_weight` parameter in the training function [126]. After the data was preprocessed and split into train and test, it was passed to the Keras data augmentation class, `ImageDataGenerator` (7) [127]. The data augmentation used was rotation, shifting width and height, sheering, zooming, and horizontal flips. An example of the augmented images is shown in Figure 32.

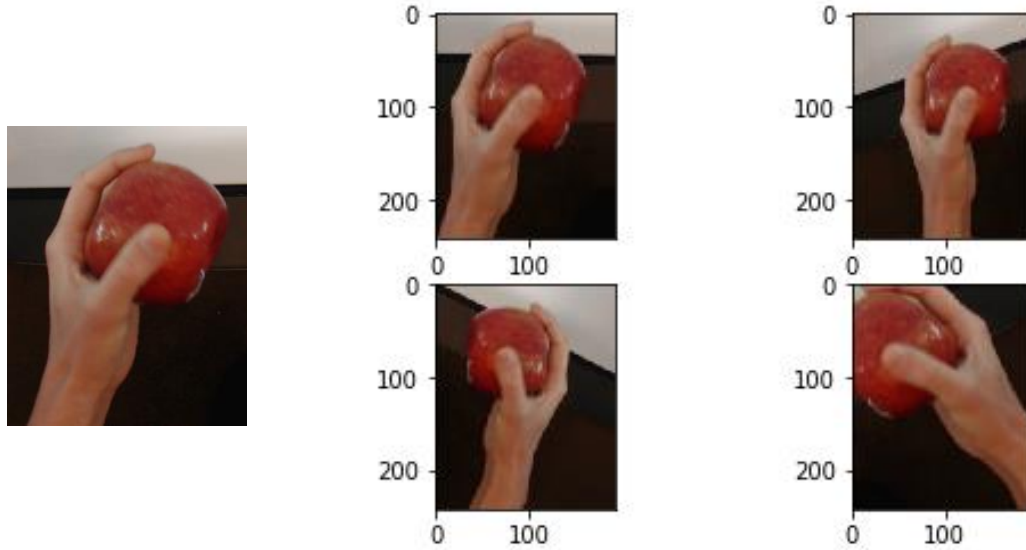


Figure 32. Left: the original training image for class Apple. Right: Example random augmentations using the ImageDataGenerator class.

Figure 32 shows the original image on the left. On the right, random augmentations are applied to rotate, shift, flip, shear and zoom in on the original image. Data augmentation is useful to prevent overfitting and ensure the model is able to generalize and perform well on new images [128]. Data augmentation using the ImageDataGenerator class does not create new images for the training set, the augmentation function augments the image before passing it to the model to train, thus providing more variety in the training set but not more data. The augmented images were then passed to the model to train (8). [127]

c. Training the Model

Stochastic gradient descent was used as the optimizer with a learning rate of 0.001. The Keras fit_generator function was used to train the model [126]. The model was trained using the VGG16 within Keras. The model structure used for training is shown in Figure 33.

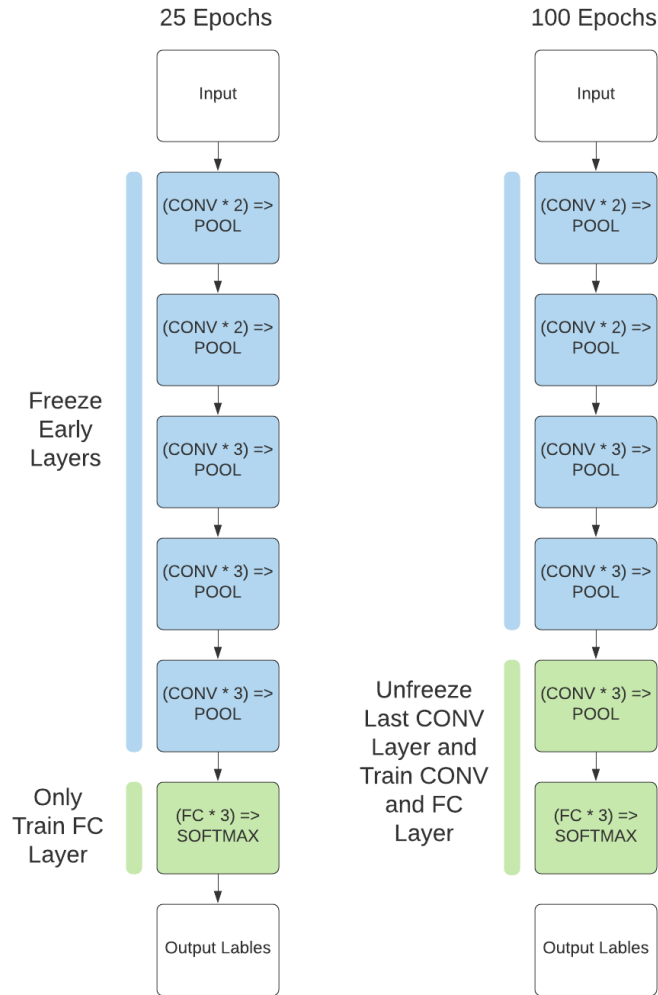


Figure 33. VGG16 Model Structure [129].

For the first 25 training epochs, Figure 36 left, the images were passed through all layers but weights were only updated for the fully connected layers. This technique allows the model to “warm up” to the new data. Typically, fine tuning struggles on smaller datasets because there is not enough data to make large changes in the weight values. Initially training the fully connected layers allows for those weights to begin learning from the data versus being randomly initialized. [17] After 25 epochs, the last convolutional layer was unfrozen and the weights of that layer were allowed to be updated as well. Only the final layer was unfrozen because the first few layers have learned simple features such as

edges and shapes that generalize well to most objects. The last few layers typically learn the more complex features specific to a particular dataset, and are thus the best candidates for training via fine tuning. The model was trained for 100 epochs. [17]

6. Inventory Detection Pipeline

Once the object identification model had acceptable accuracy, it was added to the overall detection pipeline. In order to verify the usefulness of exploring hand-object interaction, a program was developed to try to track the addition and removal of apples and oranges. To simplify the problem, single apples or oranges were used as opposed to a bag of fruit. Initial algorithm development had the hand adding the fruit by moving straight inside the refrigerator and placing the item on the lowest shelf above the produce bins. The empty hand removing the fruit moved straight in, grabbed the item, and moved straight out with the item. Only one item was added at a time, and only one hand was inside the refrigerator at a time. It was important to limit the variables for the initial program to prove feasibility. The items were not initially placed within the produce bins as the opening and closing of the bins added more complexity. Once the algorithm was accurate at detecting the fruits added the lower shelf, more complicated scenarios of adding an item into an empty bin were added. The program was tested on videos of a single user adding and removing the fruit, and closing the refrigerator door between each interaction. A flowchart of the logic for running the hand detector is shown in Figure 34.

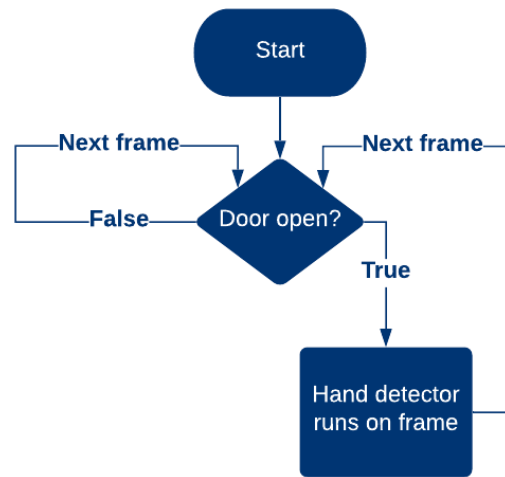


Figure 34. Flowchart for detection model applied to a video sequence.

The system begins by first loading the test video and both the detector and classifier models. Once loaded, the program begins analyzing the video frame by frame. The program checks if the refrigerator door is closed by comparing the average pixel values with a threshold value. Frames with the door closed are skipped. When the program detects the door open, it begins running the hand detector model. For development, the code ran in real-time, but in the application, the program can take more time to analyze the images. To reduce the issue of objects in the background interfering with the object classifier, only hand detections within the designated “loading zone” are passed to the classifier. An image of the loading zone is shown in Figure 35.

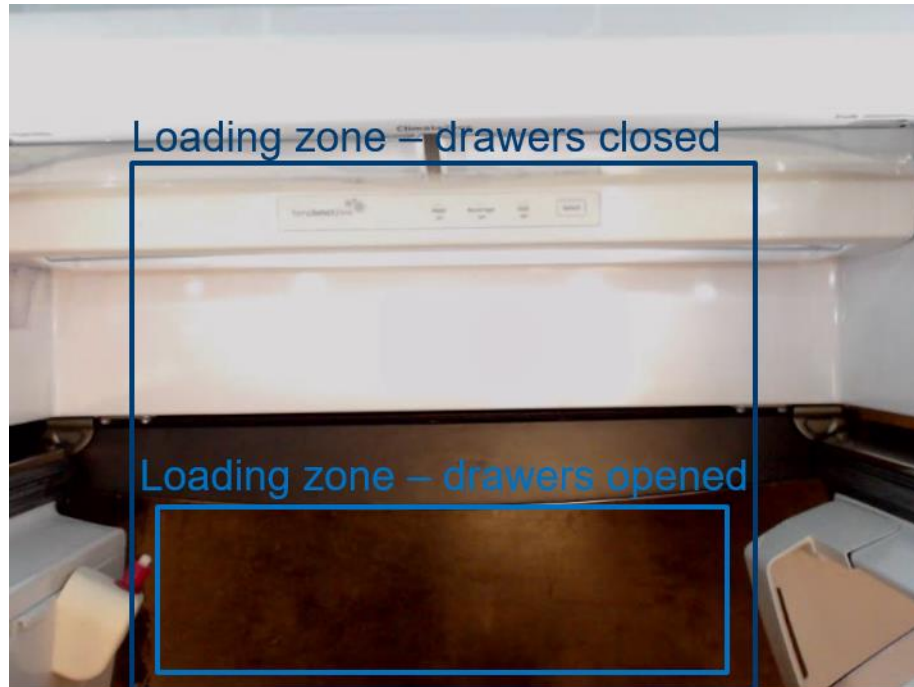


Figure 35. “Loading zone” area within which the classifier will run to detect what a hand is holding.

The loading zone area was chosen as it encompasses the largest region that is not likely to be full of food items. For example, the refrigerator doors are likely to be full of bottles and jars that could cause misidentification if visible in the cropped box passed to the classifier. When the drawers are open, the loading zone is smaller. The smaller loading zone ensures the classifier does not run when the hand is over the drawer and could predict false positives. The IoU calculations from Figure 22 are used to determine if the hand is within the loading zone. Because the area of the loading zone is so much larger than the hand bounding box, the area of union was much larger than the area of overlap. The large denominator meant the overall IoU value was very small. Because of this, only the numerator, or area of overlap was used to determine if a hand is within the loading zone. An area of overlap greater than 5000 was considered within the loading zone. When a hand is detected within the zone, it is cropped with an offset of 10 pixels and

passed to the classifier. The classifier makes a prediction on whether the hand is empty or holding an apple or orange. Possible ways to determine if an item is added or removed are:

- Use the hand tracking data to determine if the hand is moving in or out, and thus if an item is being added or removed
- Run the classifier on the first and last frame of the interaction and use that information to say what is added or removed (for example, Apple is first frame, Empty is last, so can assume an apple was added)
- Create a list to store the classifications and update each frame where the hand stays within the refrigerator. The identified object most common at the beginning of the list was most likely added, and the object most common at the end of the list is most likely removed
- Examine produce drawer images to see what has changed (added or removed) and use that information to validate the information from the hand

The different techniques were implemented in the code, and then observed by playing back a video to visually observe performance.

Knowing when an interaction was complete and when to update the inventory list was another challenge. Ideas include:

- Update the inventory list every time the door is closed
- Update the list every time a hand leaves the refrigerator
- Update the list every time the status of a hand changes (e.g., Empty to Orange)

Each idea was tested in the code to find the best solution.

7. Storing Inventory Information

An Excel spreadsheet was created to hold the inventory list. The current inventory in the Excel sheet is read into a local array when the program begins running, and is updated as the inventory changes. The inventory list is split into three storage locations: shelf, left produce bin, and right produce bin. In a production application, all unique storage areas of the refrigerator would be represented in the inventory list, but the areas were reduced in this research for ease of implementation. The centroid information from the hand tracking section is used to determine where each item was added or removed. The Excel spreadsheet is updated every time the door is closed. For the production solution, the inventory will be stored on a phone or web application. Development of the phone app was not started in time to test for this research.

RESULTS

A. Hand Detection

Experiments were done to train a TensorFlow Object Detection API model to accurately detect hands moving in and out of the refrigerator camera feed.

1. Determining the Best Model for Hand Detection

An initial experiment was done to compare the different models available in the TensorFlow Object Detection API, see Table 1, and determine which model to use to train the hand detection model. The test was done by training each model using the EgoHands dataset, and then comparing the results to published results on the EgoHands dataset [87]. The results of the initial experiment compared with published results is shown Table 4, the mean average precision for an IoU of 50% or greater is reported.

Table 4. Performance results for various models, compared with published results.

	Model name	mAP @0.50IOU on EgoHands
Experimental results	ssd_mobilenet_v2_coco	0.961
	faster_rcnn_inception_v2_coco	0.975
	rfcn_resnet101_coco	0.976
Published results	ssd_mobilenet_v1_coco [87]	0.969
	Sliding window using CaffeNet [67]	0.807*

*original paper did not use TF API so only have a single mAP value [67].

The top three models in Table 4 are all trained on the EgoHands dataset for this research, while the bottom two models are the results from published papers [87] [67]. Table 4 shows that the results are similar to published results, thus validating the training pipeline. Additionally, the table shows that each of the models performs similarly well on the data.

2. Supplementing the Dataset with the EgoHands Dataset

The next experiment was to see if a model would perform better if it was first trained on the large EgoHands dataset and then on the local dataset, or only trained on the local dataset. The experiment was only tried using the `faster_rcnn_inception_v2_coco` in the interest of time. The results are shown in Table 5.

Table 5. mAP results for each model on the local dataset.

Model name	mAP @0.50IOU local dataset only	mAP @0.50IOU pre-trained EgoHands to train local dataset
<code>faster_rcnn_inception_v2_coco</code>	0.965	0.962

Table 5 shows that the models perform similarly on the test dataset when looking at an mAP for 50% IOU.

Results for the two models on the validation video are shown in Table 6 and Table 7.

Table 6. Precision and recall (left) and confusion matrix for EgoHands then local dataset (`faster_rcnn_inception_v2_coco`).

Class	Precision	Recall	F1-Score	Support
Hand	0.90	0.94	0.92	185

True Class	Hand	173	12
	No Hand	19	-
		Hand	No hand
		Predicted class	

Table 7. Precision and recall (left) and confusion matrix for local dataset (`faster_rcnn_inception_v2_coco`).

Class	Precision	Recall	F1-Score	Support
Hand	0.98	0.85	0.91	185

True Class

Hand	158	27
No Hand	3	-
	Hand	No hand

Predicted class

Table 6 and Table 7 show that, despite similar training metrics, the model trained on the EgoHands dataset first and then on the local dataset detects 15 more hands in the validation video. The F1-scores are similar for both models, but recall is 0.94 for the EgoHands then local compared to just 0.85 for the local trained model.

3. Results for Each Model in the Real-World Application

Each model at training time had similar metrics, Table 4, but it was observed in the application that some models performed better than others for the task of tracking a hand in the refrigerator. The most important metric for the hand tracking was a high TP rate. FP were not as important because they could be easily ignored in the code. Results for `ssd_mobilenet_v2_coco` and `rfcn_resnet101_coco` on the validation video are shown in Table 8, results for `faster_rcnn_inception_v2_coco` are shown above in Table 6.

Table 8. Precision and recall (left) and confusion matrix for models: (a)

`ssd_mobilenet_v2_coco` (b) `rfcn_resnet101_coco`.

Class	Precision	Recall	F1-Score	Support
Hand	1.00	0.26	0.42	185

True Class

Hand	49	136
No Hand	0	-
	Hand	No hand

Predicted class

(a) `rfcn_resnet101_coco`

Class	Precision	Recall	F1-Score	Support
Hand	0.99	0.87	0.93	185

True Class	Hand	161	24
	No Hand	2	-
	Hand		No hand
Predicted class			

(b) `ssd_mobilenet_v2_coco`

Table 8 shows that while `ssd_mobilenet_v2_coco` had a high mAP, in practice the model only detected 49 of the total 185 hands. Model `rfcn_resnet101_coco` had the highest mAP after training, but only detected 161 of the 185 hands. The best model was `faster_rcnn_inception_v2_coco`, which detected 173 hands out of 185. The `faster_rcnn_inception_v2_coco` model had many more FP than the other models.

The disk size of each model file is shown in Table 9.

Table 9. Disk size of each hand detection model.

Model Name	Size (MB)
<code>ssd_mobilenet_v2_coco</code>	54.1
<code>faster_rcnn_inception_v2_coco</code>	148
<code>rfcn_resnet101_coco</code>	600

Table 9 shows that the `ssd_mobilenet_v2_coco` is the smallest model, at 54.1 MB. `Faster_rcnn_inception_v2_coco` is roughly three times larger at 148 MB, and `rfcn_resnet101_coco` is by far the largest at 600 MB.

In addition to the mAP, precision/recall, confusion matrix results, and size, it was observed that the `rfcn_resnet101_coco` model took almost 11 seconds to infer a result between frames. Both `ssd_mobilenet_v2_coco` and `faster_rcnn_inception_v2_coco` took about two seconds to make a prediction. The time difference would not matter in the application as there is no need to run the code in real-time. For development purposes, a

faster inference time was essential to quickly validate and test the logic for the next parts of the research.

4. Model Trained with Left and Right Hand Class

Faster_rcnn_inception_v2_coco was used to train a model to distinguish between the left and right hand. The model was first trained on the EgoHands dataset with a left and right hand class. The training results are shown in Figure 36.

Average Precision	(AP)	@[IoU=0.50:0.95	area= all	maxDets=100]	= 0.673
Average Precision	(AP)	@[IoU=0.50	area= all	maxDets=100]	= 0.913
Average Precision	(AP)	@[IoU=0.75	area= all	maxDets=100]	= 0.819
Average Precision	(AP)	@[IoU=0.50:0.95	area= small	maxDets=100]	= 0.038
Average Precision	(AP)	@[IoU=0.50:0.95	area=medium	maxDets=100]	= 0.332
Average Precision	(AP)	@[IoU=0.50:0.95	area= large	maxDets=100]	= 0.708
Average Recall	(AR)	@[IoU=0.50:0.95	area= all	maxDets= 1]	= 0.476
Average Recall	(AR)	@[IoU=0.50:0.95	area= all	maxDets= 10]	= 0.761
Average Recall	(AR)	@[IoU=0.50:0.95	area= all	maxDets=100]	= 0.766
Average Recall	(AR)	@[IoU=0.50:0.95	area= small	maxDets=100]	= 0.204
Average Recall	(AR)	@[IoU=0.50:0.95	area=medium	maxDets=100]	= 0.523
Average Recall	(AR)	@[IoU=0.50:0.95	area= large	maxDets=100]	= 0.795

Figure 36. Precision and recall training metrics for EgoHands dataset with left and right hand class.

The training results in Figure 36 are promising, with 91.3% average precision at an IoU of 50%.

The weights from the EgoHands model were then used to train the local dataset.

Results from the training are shown in Figure 37.

Average Precision	(AP)	@[IoU=0.50:0.95	area= all	maxDets=100]	= 0.548
Average Precision	(AP)	@[IoU=0.50	area= all	maxDets=100]	= 0.836
Average Precision	(AP)	@[IoU=0.75	area= all	maxDets=100]	= 0.631
Average Precision	(AP)	@[IoU=0.50:0.95	area= small	maxDets=100]	= -1.000
Average Precision	(AP)	@[IoU=0.50:0.95	area=medium	maxDets=100]	= 0.244
Average Precision	(AP)	@[IoU=0.50:0.95	area= large	maxDets=100]	= 0.565
Average Recall	(AR)	@[IoU=0.50:0.95	area= all	maxDets= 1]	= 0.639
Average Recall	(AR)	@[IoU=0.50:0.95	area= all	maxDets= 10]	= 0.710
Average Recall	(AR)	@[IoU=0.50:0.95	area= all	maxDets=100]	= 0.717
Average Recall	(AR)	@[IoU=0.50:0.95	area= small	maxDets=100]	= -1.000
Average Recall	(AR)	@[IoU=0.50:0.95	area=medium	maxDets=100]	= 0.404
Average Recall	(AR)	@[IoU=0.50:0.95	area= large	maxDets=100]	= 0.729

Figure 37. Precision and recall training metrics for local dataset with left and right hand class.

The metrics for the trained model decrease slightly, with 83.6% average precision at an IoU of 50%. The model performs poorly on the validation videos. Hands are labelled as both left and right, left hand is marked right hand and vice versa, and non-hands are labelled hands. An example image with the hand predictions is shown in Figure 38.

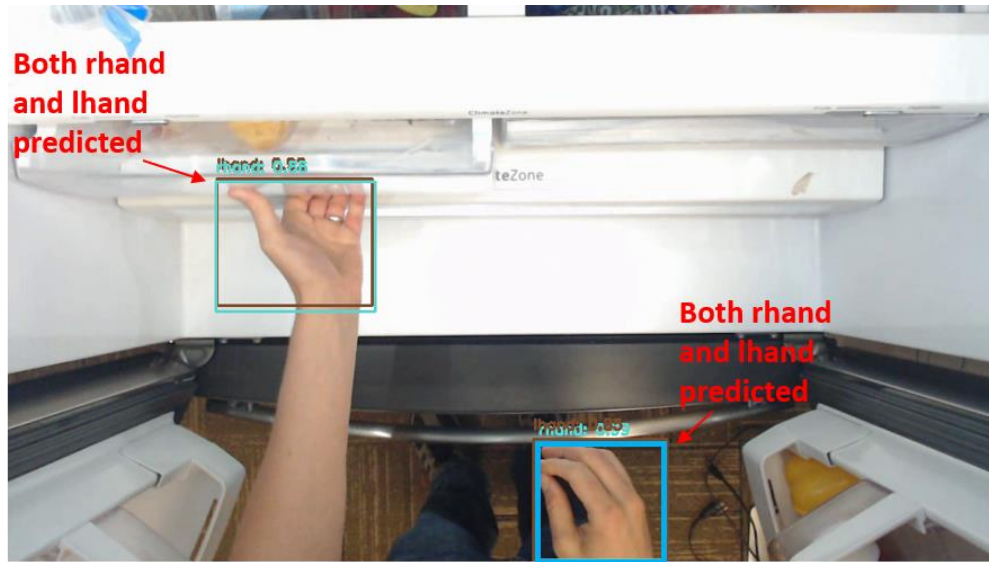


Figure 38. Prediction on a validation video for the left/right hand model.

The labels are difficult to see in the image, but each hand is labeled both lhand (left hand) and rhand (right hand). The model was unable to learn enough discriminating features from the data to reliably differentiate between the left and right hand.

5. Hand Detector Limitations

Overall, the hand detector with a single hand class performed well on new data, but the model struggled to detect hands at the edges of the frames. An example of a missed hand is shown in Figure 39.

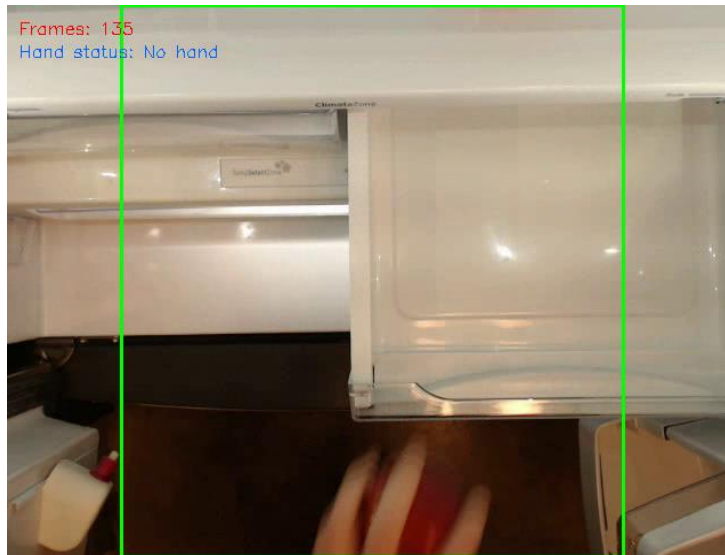


Figure 39. The hand detector did not detect the hand on the edge of the frame.

Figure 39 shows that the hand was not detected. The hand is blurry and holding an object, which could be why the hand was missed.

Based on the above data and observations, the `faster_rcnn_inception_v2_coco` model trained on the EgoHands dataset and then the local dataset and a single hand class was determined to be the best choice to maximize accuracy, speed, and model size. The model was 93% accurate on the limited validation data. The experiments in the following sections use the `faster_rcnn_inception_v2_coco` model.

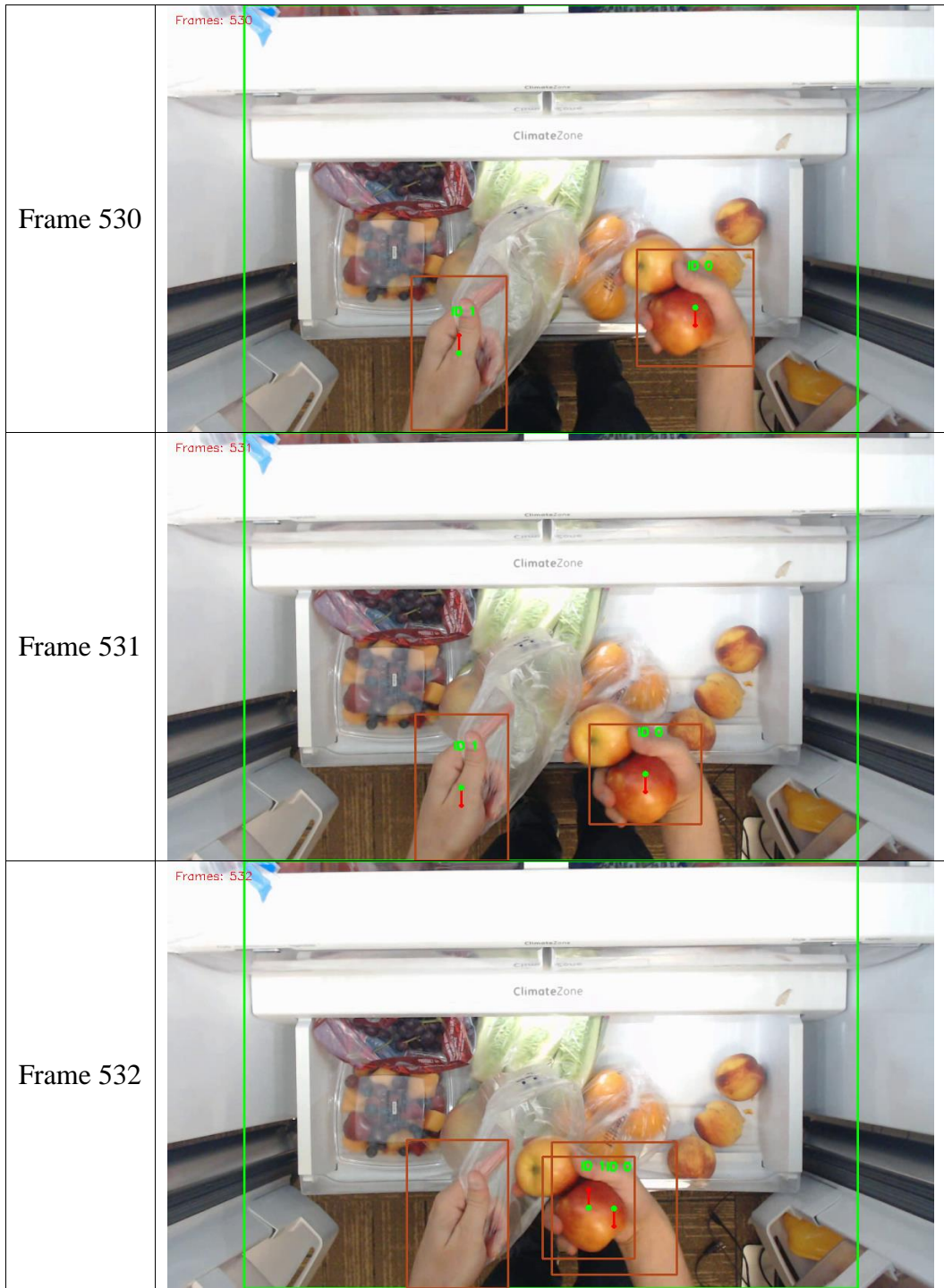
B. Hand Tracking

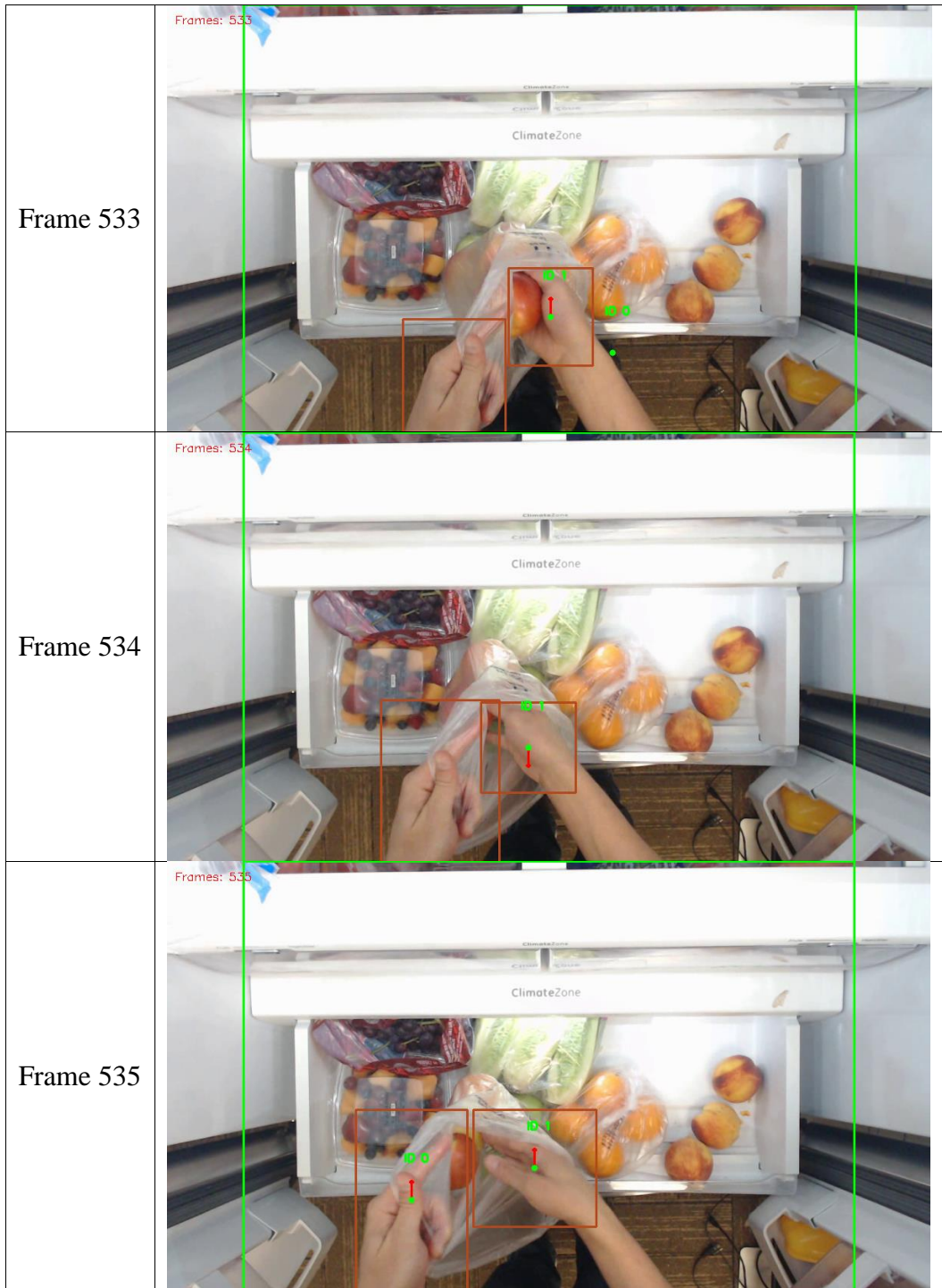
The second portion of the research focused on tracking the hand within the camera frame. The detector detects the hand, and the tracker determines if the hand is the same as in the previous frame, or a new hand. The tracker also needs to be able to track the left and right hand separately without confusing the two hands.

1. Centroid Tracking

Centroid tracking uses the center of the detected hand bounding box as an object anchor. Centroids for objects in new frames are compared to centroids from prior frames. If the distance between the centroids is less than the max distance threshold, the new and old object can be considered the same object, otherwise the new object is considered a new hand within the frame. Through visual inspection, the best max distance threshold for the application was found to be 150 pixels. The 150-pixel threshold means that if the distance between a centroid from frame-to-frame is less than 150 pixels, the two objects are the same. A large distance was needed because tracking experiments were carried out on video with a lower frame rate, thus the hand moved far between frames and detections. If the previous and new centroid were both on the same half of the camera frame, the distance threshold was increased to 300. The higher distance allowed for the tracker to continue tracking the hand even after a missed detection, the hand traveling a large distance between frames, or when the hand momentarily moves further into the refrigerator and out of the frame. The sequence in Table 10 shows the hand tracker in action. The left hand begins as ID 1, the right ID 0. The hands are very close in Frame 532, which causes the hand tracker to swap the hand IDs. Frame 535 and 536 shows that the left hand is now ID 0, the right ID 1.

Table 10. Example hand tracking sequence where the tracker swaps the left and right hand.







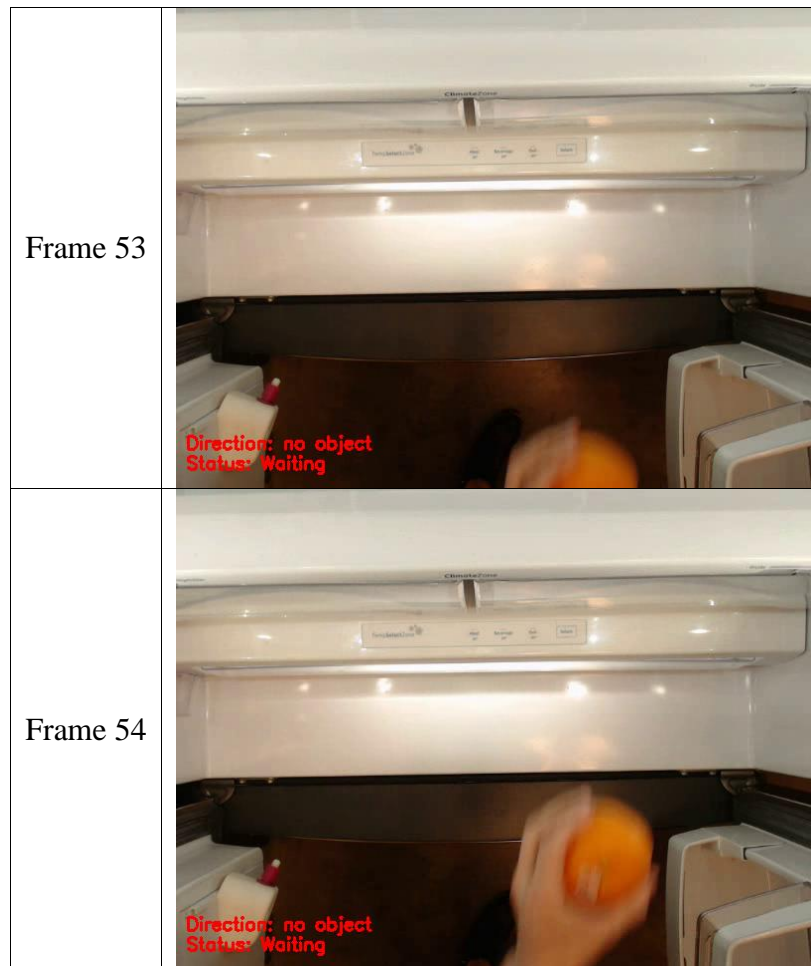
Fine-tuning the max distance parameter was important to ensure the left and right hand are recognized as two different objects. Despite much tweaking of the distance parameter, the centroid tracker was never able to accurately distinguish between the two hands when the hands crossed over or were close together within a frame, like shown in Table 10. For a single hand, the tracker was able to accurately track and determine the direction of the hand.

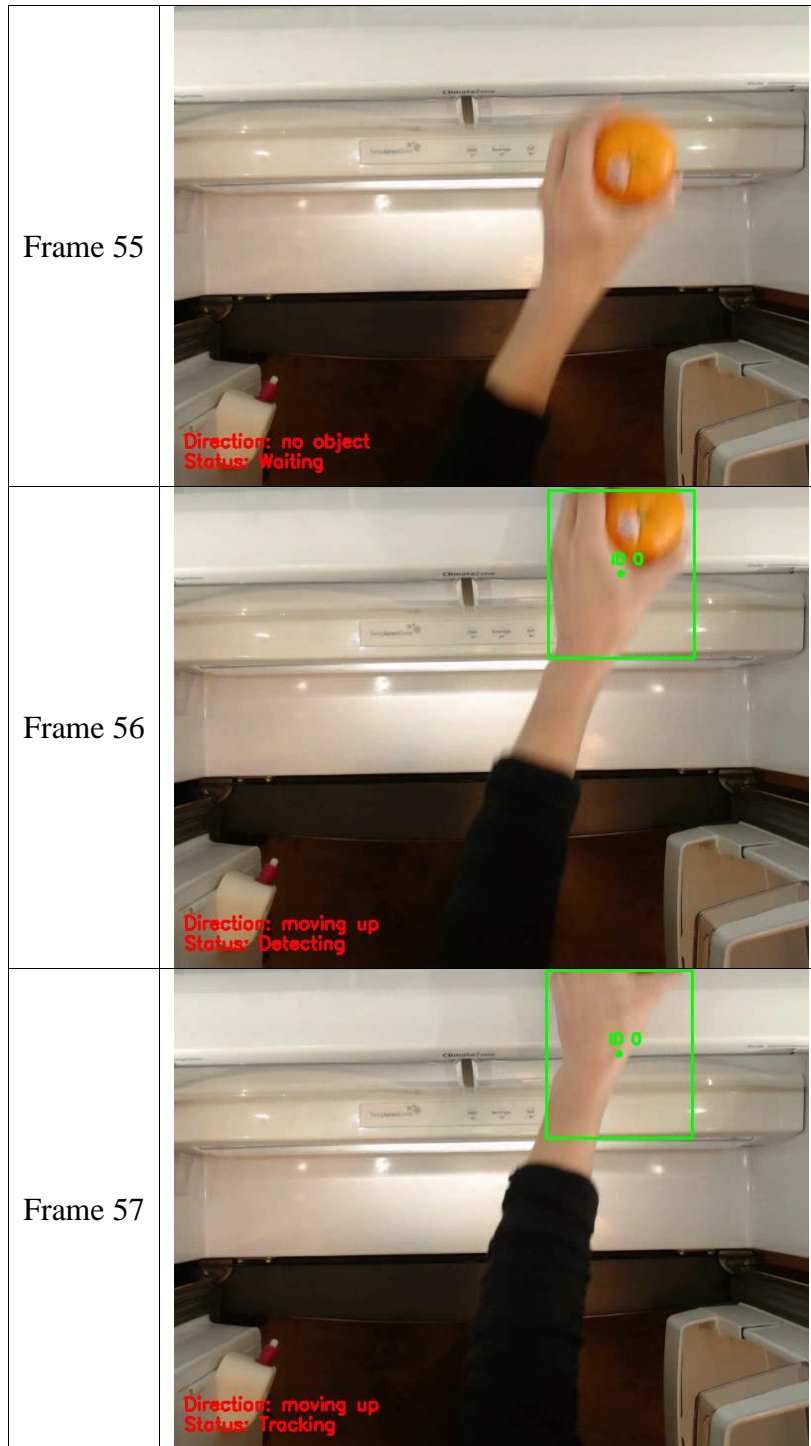
2. Correlation Tracker

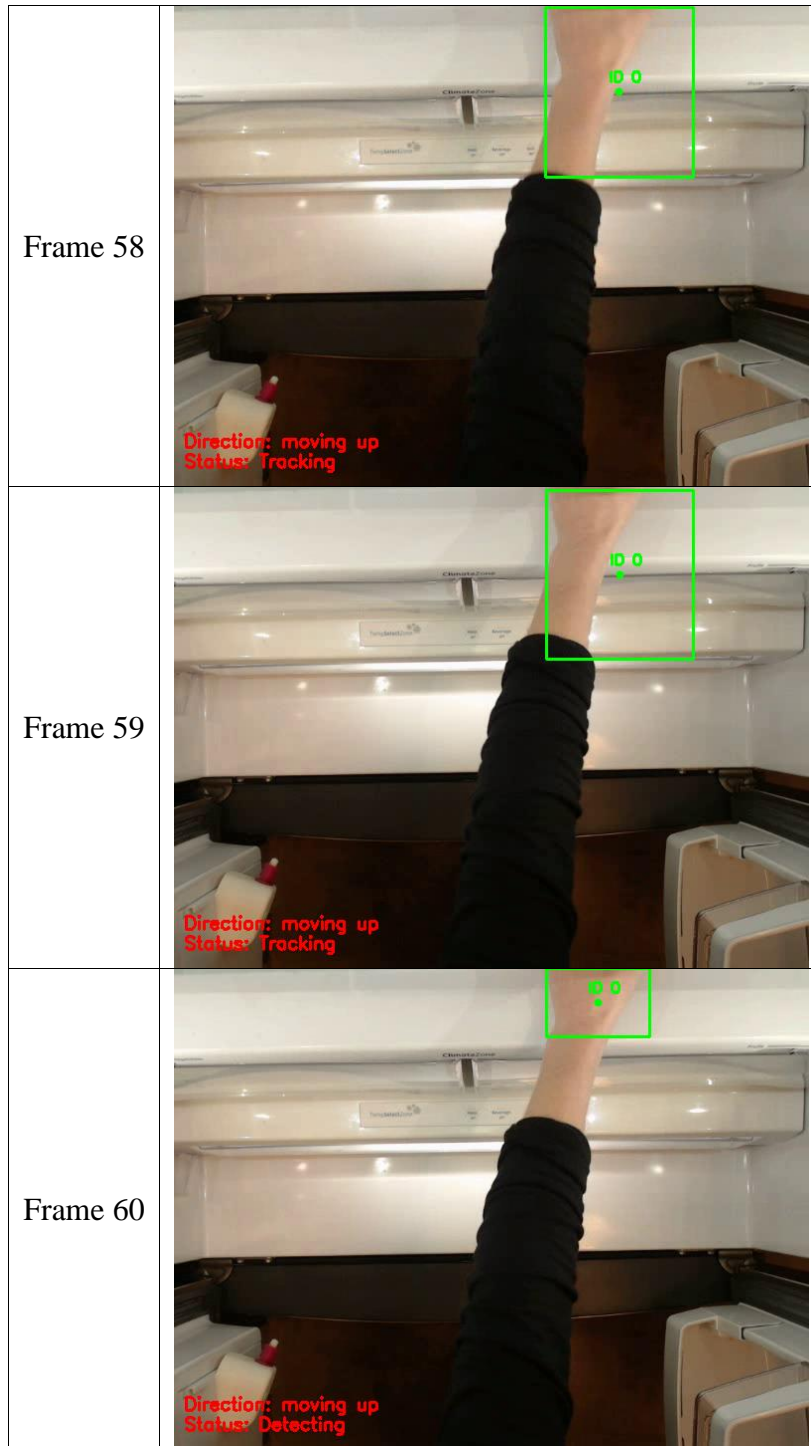
The dlib correlation tracker was supposed to be an improvement over the centroid. Unlike the centroid tracker method, the hand detector would not need to run every frame, thus speeding up the entire process. The hand detector would run, pass the detected bounding box to the tracker, and the tracker would use a faster algorithm to track the hand in subsequent frames. Because the hands within the refrigerator move quickly between frames, the correlation tracker would lose the hand almost immediately. Once the hand was lost, nothing could be done to re-track the hand until the hand detector ran again. The important value for the correlation tracker is how often to run the detector to update the bounding box used by the tracker. Run the detector too few times and the hand

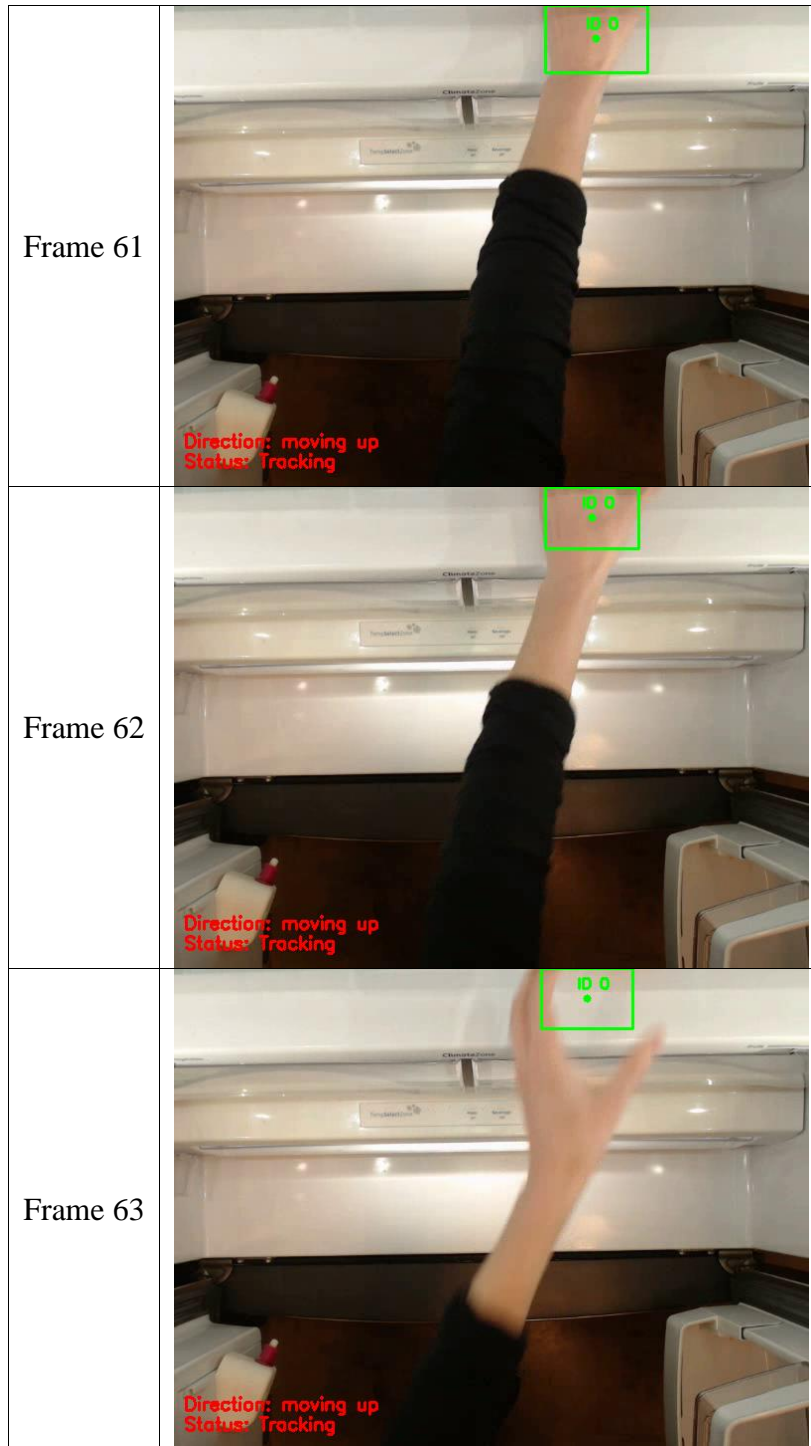
will never be detected during an interaction. Run the detector too often, and the speed benefits of the correlation tracker are cancelled out. It was found, through visual inspection, that running the hand detector every fourth frame worked the best to balance detecting the hand early in an interaction, and still speeding up the detections. An example video sequence using the correlation tracker and running the detector every fourth frame is shown in Table 11.

Table 11. dlib correlation tracker applied to a video sequence, detector running every fourth frame.









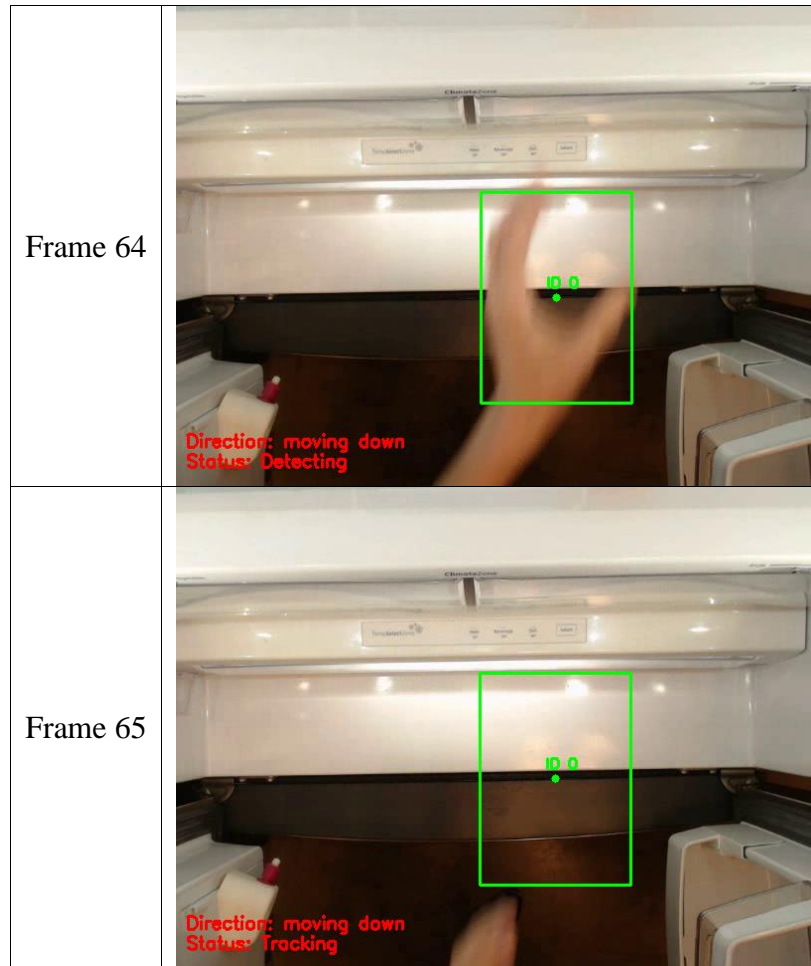


Table 11 shows that the detector does not run until four frames after the hand enters the frame (frame 56). The bounding box does not resize for the smaller partial hand until the detector runs again in frame 60. In frame 63, the tracker is unable to track the hand as it moves outside the original area of detection. Frame 64 shows the detector running again, but the hand is immediately lost in frame 65.


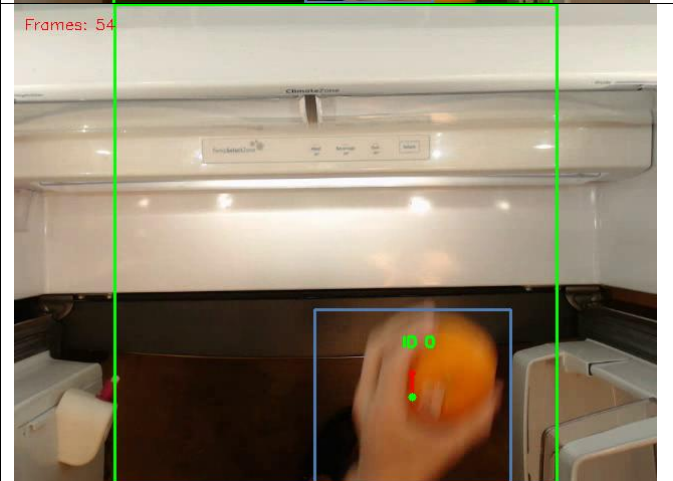
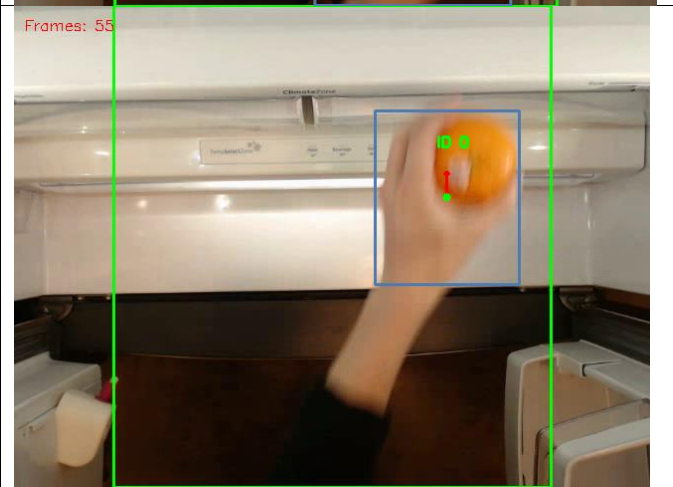
The correlation tracker was much faster than running the hand detector on each frame. The correlation tracker updated on each new frames almost instantly, compared to the two second lag each time the detector runs. To get the correlation tracker to track the hand in every frame, the detector needed to run on every frame, negating the benefits of the correlation tracker. While it may be possible that spending more time fine-tuning the

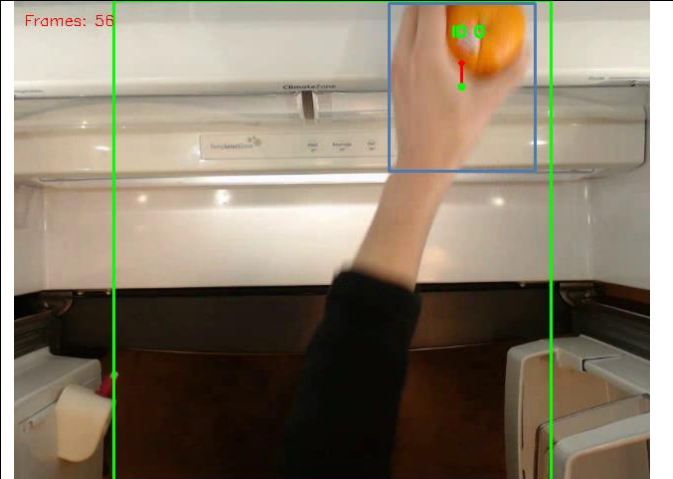
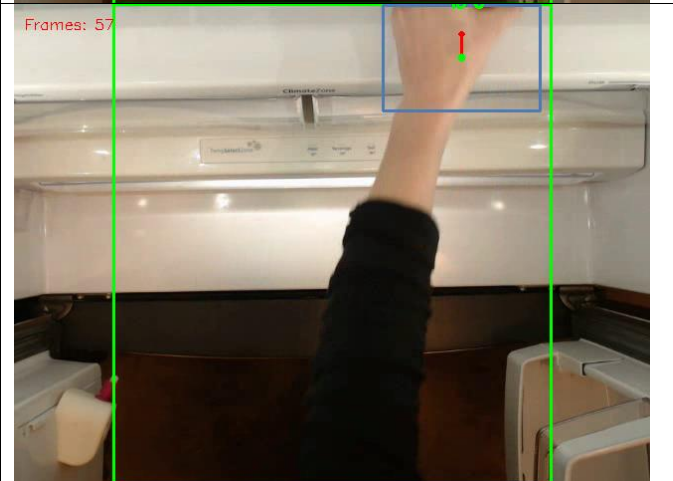
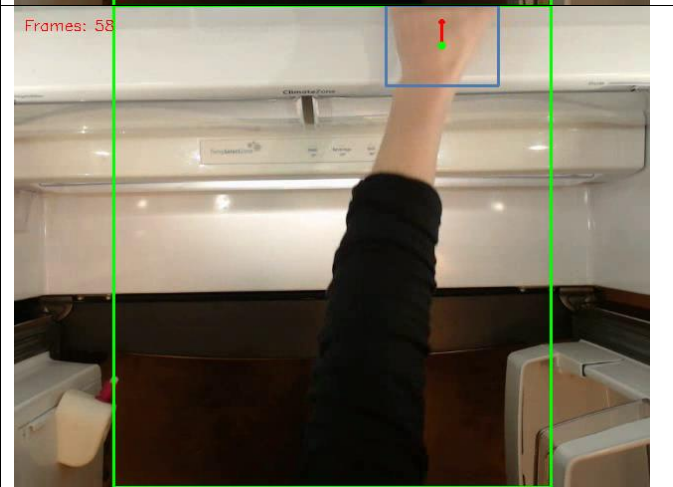
parameters could provide a good result with the correlation tracker, the time saved per frame was not worth the effort in this case. For the application in this research, the correlation tracker did not perform well.

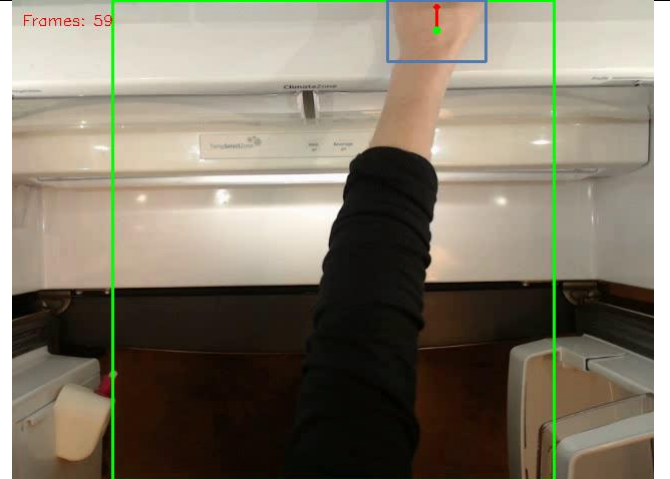
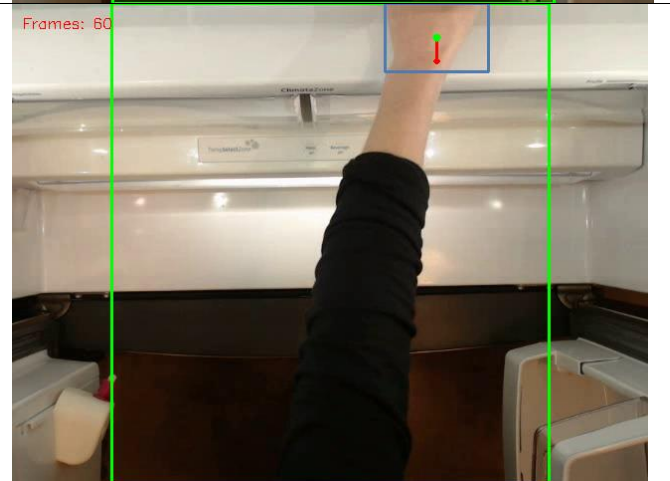
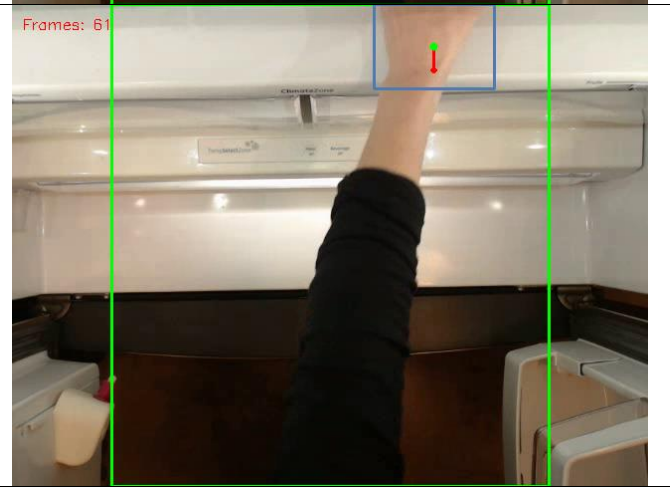
3. Direction of Movement

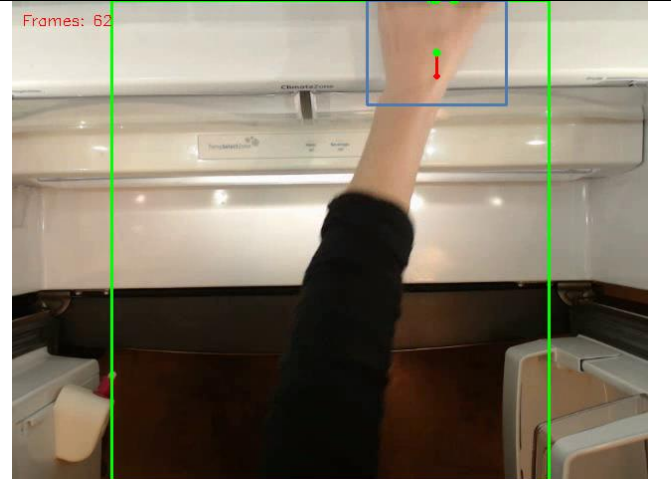
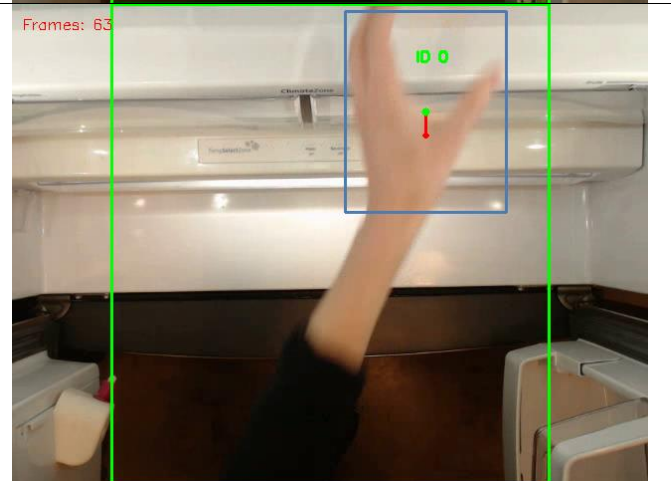
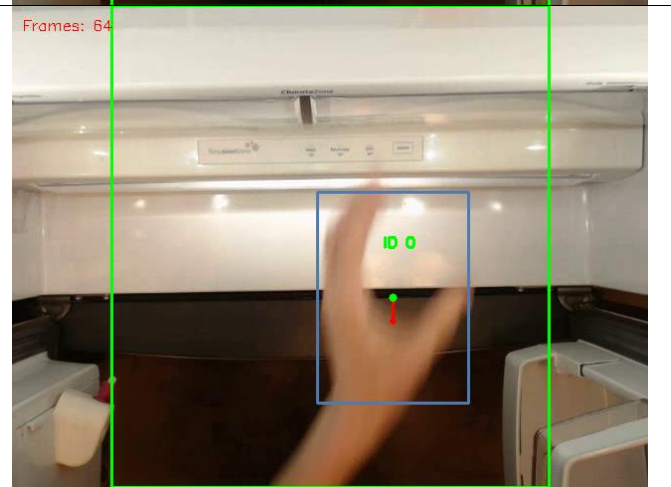
Determining the direction of the hand was as straight-forward as looking at the sign on the distance calculation between the new and old frame. The program the direction code was based on used an average of old distances to determine direction [108]. Using an average is useful if the tracked objects are moving large distances in the frame in one direction, but was not accurate for this research. The hands change direction quickly between frame, and using an average of prior centroid locations caused errors. Comparing the new centroid to only the prior centroid gave better results. The video sequence in Table 12 shows the movement direction of a hand during one interaction. The direction determined from comparing only the last Y centroid and the direction determined from comparing to an average of prior centroids are listed in the first column of the table. The red direction arrow in the frame is based on the last Y centroid information.

Table 12. Movement direction, shown by the red arrow, for one interaction.

<p>Frame 53</p> <p>First detection</p>	<p>Frames: 53</p> 
<p>Frame 54</p> <p>Last cY_{prev}: "Moving in"</p> <p>Average cY_{prev}: "Moving in"</p>	<p>Frames: 54</p> 
<p>Frame 55</p> <p>Last cY_{prev}: "Moving in"</p> <p>Average cY_{prev}: "Moving in"</p>	<p>Frames: 55</p> 

<p>Frame 56</p> <p>Last cY_{prev}: "Moving in" Average cY_{prev}: "Moving in"</p>	
<p>Frame 57</p> <p>Last cY_{prev}: "Moving in" Average cY_{prev}: "Moving in"</p>	
<p>Frame 58</p> <p>Last cY_{prev}: "Moving in" Average cY_{prev}: "Moving in"</p>	

<p>Frame 59</p> <p>Last cY_{prev}: "Moving in" Average cY_{prev}: "Moving in"</p>	<p>Frames: 59</p> 
<p>Frame 60</p> <p>Last cY_{prev}: "Moving out" Average cY_{prev}: "Moving in"</p>	<p>Frames: 60</p> 
<p>Frame 61</p> <p>Last cY_{prev}: "Moving out" Average cY_{prev}: "Moving in"</p>	<p>Frames: 61</p> 

<p>Frame 62</p> <p>Last cY_{prev}: "Moving out" Average cY_{prev}: "Moving in"</p>	<p>Frames: 62</p> 
<p>Frame 63</p> <p>Last cY_{prev}: "Moving out" Average cY_{prev}: "Moving in"</p>	<p>Frames: 63</p> 
<p>Frame 64</p> <p>Last cY_{prev}: "Moving out" Average cY_{prev}: "Moving out"</p>	<p>Frames: 64</p> 

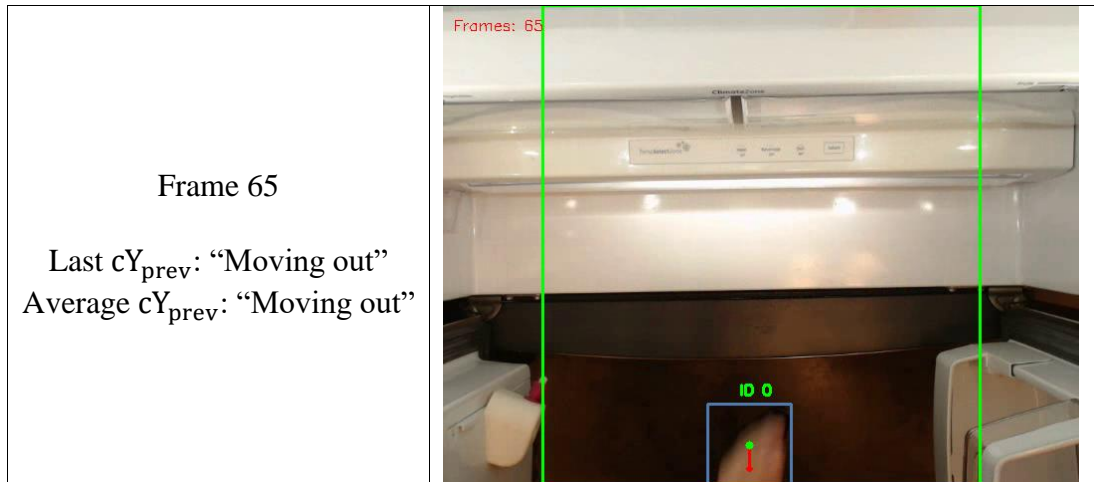


Table 12 shows that using only the last previous Y centroid value can detect a change in direction four frames faster than observing the average of all previous Y centroids. Table 12 also shows that the centroid tracker plus direction is very accurate when tracking a single hand. Because a hand in the refrigerator is constantly changing direction and moving in and out (like to open a bin), knowing what direction the hand was moving was not as useful as hypothesized. Using the direction to make decisions for the add/remove logic presented unnecessary complexity, thus was not used for the add and remove logic.

4. Distinguishing Between Hands

Distinguishing between the left and right hand was a very challenging problem. The centroid tracker was not able to differentiate between hands when the two hands crossed over or came close together, as shown in Table 10. Some cases could be dealt with by hard coding rules, such as whichever hand centroid was closest to the right side of the frame was assumed to belong to the right hand and vice versa, but there was no way to use logic to solve all cases. Observing the arms and the hands could be a way to distinguish the hands, but was not tried in this research.

An accurate way to distinguish between hands was not found during the many experiments, thus the scope of the research was limited to focus on a single hand within the refrigerator.

C. Hand Analysis

The final part of the research was to analyze the detected hands to determine if they were empty or holding an apple.

1. Hand Empty or Not

Initially, all hand examples were used to train a model to detect if a hand was empty or not. After several epochs with no change in training loss or accuracy, it was clear that the data was insufficient at providing the model with enough information to distinguish between the two classes. Looking into the data showed edge cases like the image in Figure 40, below, where it is difficult for even a human to clearly determine if the hand is empty or not.



Figure 40. An example image of an edge case where it is difficult to definitively determine if the left hand is empty or not.

Figure 40 demonstrates the challenge of analyzing hand-object interaction, especially in and around the bin. A simple solution for this problem was to remove all edge case images and only provide the model with clearly empty and not empty hand data.

However, removing the difficult images did not solve the problem that in the real-world application the model would see difficult images. The model trained on clear-cut empty or not cases performed acceptably on those images, but predictably performed poorly on less-obvious instances. For example, should the image in Figure 41, where the left hand is opening the bin, label the hand as empty or not?



Figure 41. An example image where it is unclear whether the left hand opening the bin should be considered empty or not.

Questions such as how to classify the hand in Figure 41, which are challenging even for a human, become exceedingly difficult to train a computer to interpret. The experiment showed that other methods were needed to extract useful information from the hand-object interactions.

2. TensorFlow Object Detection API for Produce Detection

The first experiment done to identify objects in the hand was to use the TensorFlow Object Detection API to train a model to detect all annotated objects in the dataset. If an object bounding box overlapped a hand bounding box, it could be assumed that the hand was holding the detected object. An apple bounding box overlapping a hand bounding box by at least 50% would be considered a hand holding an apple. Using object/hand overlap would only work if there was not a bin full of produce underneath to negate the overlap logic. The Faster R-CNN model trained on Inception V3 was used to train the model. The results of the training are shown below:

Average Precision	(AP)	@ [IoU=0.50:0.95	area= all	maxDets=100]	= 0.515
Average Precision	(AP)	@ [IoU=0.50	area= all	maxDets=100]	= 0.762
Average Precision	(AP)	@ [IoU=0.75	area= all	maxDets=100]	= 0.593
Average Precision	(AP)	@ [IoU=0.50:0.95	area= small	maxDets=100]	= -1.000
Average Precision	(AP)	@ [IoU=0.50:0.95	area=medium	maxDets=100]	= 0.421
Average Precision	(AP)	@ [IoU=0.50:0.95	area= large	maxDets=100]	= 0.518
Average Recall	(AR)	@ [IoU=0.50:0.95	area= all	maxDets= 1]	= 0.504
Average Recall	(AR)	@ [IoU=0.50:0.95	area= all	maxDets= 10]	= 0.603
Average Recall	(AR)	@ [IoU=0.50:0.95	area= all	maxDets=100]	= 0.608
Average Recall	(AR)	@ [IoU=0.50:0.95	area= small	maxDets=100]	= -1.000
Average Recall	(AR)	@ [IoU=0.50:0.95	area=medium	maxDets=100]	= 0.443
Average Recall	(AR)	@ [IoU=0.50:0.95	area= large	maxDets=100]	= 0.615

Figure 42. Precision and recall for the model trained on all categories.

The data from Figure 42 is after about 13,000 training steps, and shows an mAP of 0.762. Training was stopped because the loss began to increase consistently. Overall loss at this point was 0.20. The results are less than 80% mAP for an IoU of 50%, which is not enough to be robust in a production setting. Reasons for the low mAP could be due to not enough data for each category and inconsistent bounding box annotations. An example of a bad annotation is shown in Figure 43.

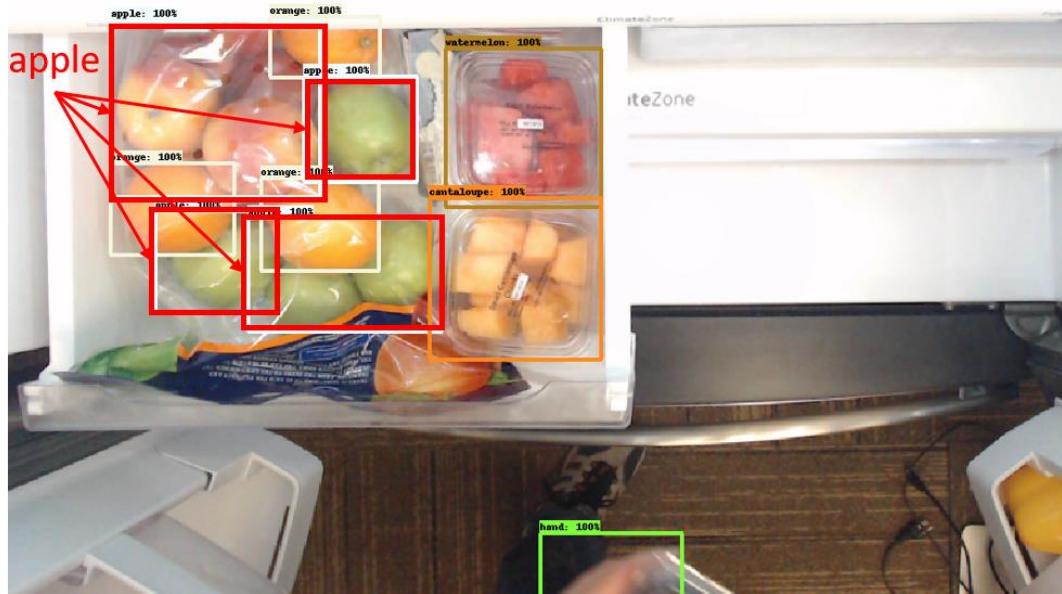


Figure 43. Inconsistent annotation where the apple bounding box covers both single and multiple apples. Each red box denotes the *apple* class.

Each red box in Figure 43 denotes an instance of the *apple* class. The apple bounding boxes in the upper left of the bin includes both single and multiple apples. At annotation time, multiple apples were put into one *apple* bounding box to save time. A quantity label was added to the annotation to distinguish between one or many apples. Training using the quantity label would need to produce a different class for each quantity of apples, or a class of a single apple and multiple apples. Even a class of multiple apples might not be distinct enough for the model to learn enough features to accurately detect the class in a general setting. Experiments on the validation video show that the trained model is unable to reliably detect a single apple in a frame.

Performance could be improved by re-annotating individual produce like apples and oranges, as could adding more images. Bounding box annotations are time

consuming, therefore other routes were researched to more easily solve the problem of detecting produce in the hand.

The model trained on all objects is both less accurate than the hand model trained using the EgoHands weights, as well as not accurate on the produce items.

3. Image Background Removal

Experiments were done to try and remove the background and extract the foreground. Because the refrigerator camera only provides a 2-dimensional view, it is difficult, even for humans, to know the difference between an item in the produce bin with an empty hand above it, versus a hand holding an apple over a full produce bin. An example of this challenge is shown in Figure 44.



Figure 44. A challenging image to determine if the apples are in the right hand or within the bottom produce bin.

Many of the bounding boxes of the produce items in Figure 44 would overlap the hand bounding box, but not all of them are in the hand. One idea to solve this problem would be to extract only the moving or foreground parts of the image (the hands) by removing the static background.

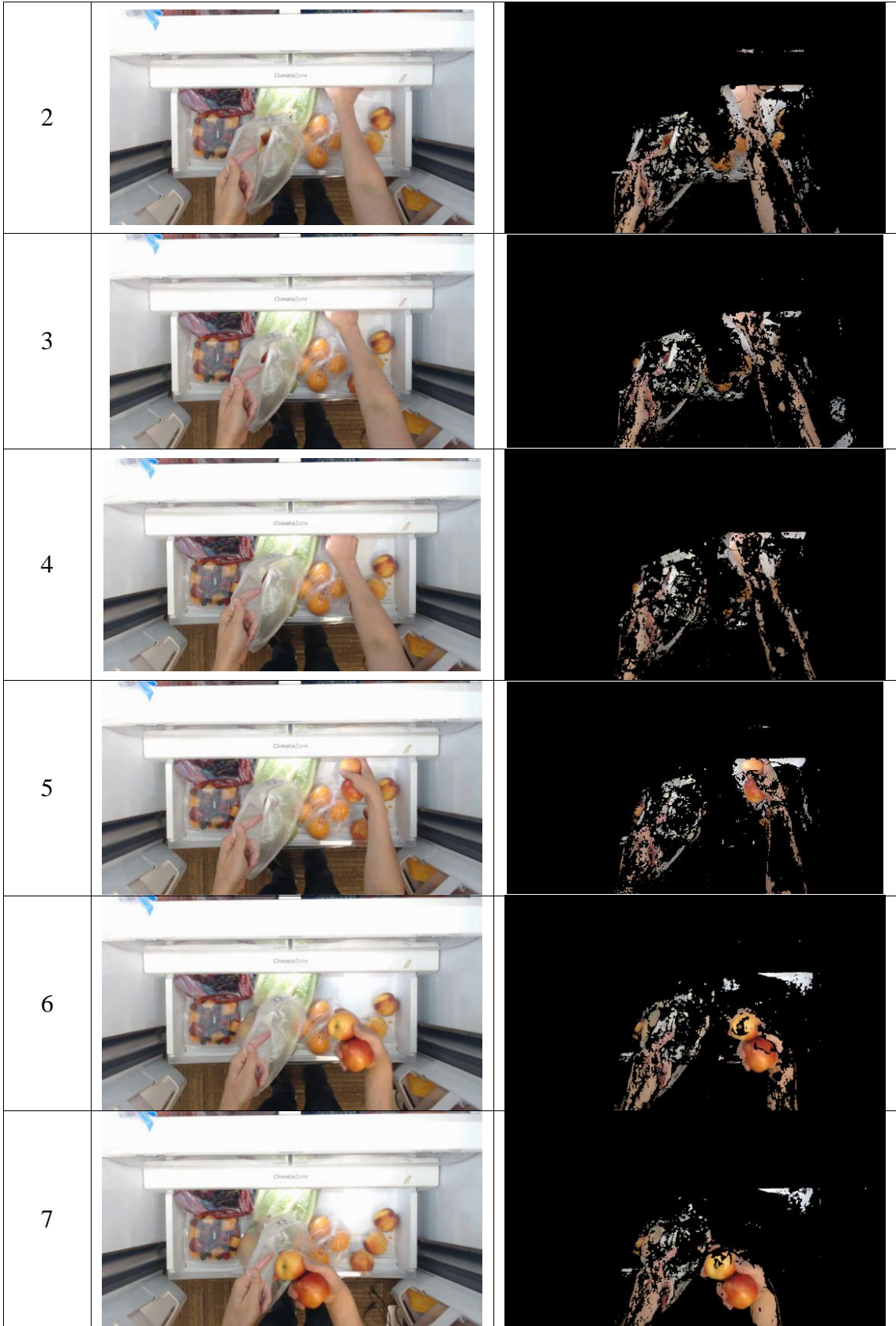
a. MOG2 Background Subtractor in OpenCV

There are two variables in the `createBackgroundSubtractorMOG2` class: `history` and `varThreshold`. `history` is how far back in the history of video frames the background subtractor will go to create the background model. The `history` variable defaults to 500. The `varThreshold` value determines what threshold to use when comparing background pixels to foreground pixels. If the distance between the background model and the current frame is greater than the threshold, that pixel is considered foreground. The default `varThreshold` value is 16. [130] Trials were done to try and optimize the `history` and `varThreshold` values, but the difference between the performance using the default values and trial-and-error values was not enough to warrant the effort to fine-tune the parameters.

The output of the subtractor on a subset of images corresponding to the interaction in Figure 44 is shown in Table 13.

Table 13. Video sequence showing the original frame and the extracted foreground using MOG2.

Frame #	Frame	Extracted Foreground
1		



Frame 5 in Table 13 is the same image shown in Figure 44. The extracted foreground image shows clearly that two apples are in the user’s hand and not in the bin. An object recognition system could run on this image, and would only see the items in the hand. With the background removed, the detector would not be challenged by bin items.

The background subtractor does not produce crisp images of the extracted foreground. Filters and other image processing techniques can be used to get a more clear image, but those experiments are outside of the scope of this research [131].

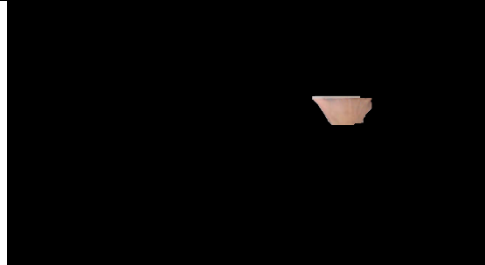
b. GrabCut













The hand detection model was used to pass rectangles into the GrabCut function. GrabCut in OpenCV has a single parameter that can be tweaked, iterCount, or the number of iterations the algorithm runs [111]. For this experiment, it was determined through trial and error that there was no discernable difference between the default iterCount of five and other values. The results shown here use an iterCount of five.

The output of GrabCut on the same subset of images from Table 13 and corresponding to the interaction in Figure 44 is shown in Table 14.

Table 14. Video sequence showing the original frame and the extracted foreground using

GrabCut

Frame #	Frame	Extracted Foreground
1		

2		
3		
4		
5		
6		
7		

For the sequence in Table 14 only one of the two hand bounding boxes was passed into the GrabCut function. Only the right hand was examined to make the code easier to implement. As with the MOG2 function, GrabCut is able to show that two of the apples in Figure 44 are in the hand and not in the bin. Frame 6 shows that some of the produce in the bin was left in the frame. Bin items in the extracted foreground frame could cause false positives by an object recognition system. GrabCut provides a much cleaner output image, but only operates within the bounding boxes. This is unlike MOG2 which works on any pixel in the image that is substantially different than the previous pixels.

c. Color Thresholding

Color thresholding to extract skin has been a popular avenue for hand detection and produce classification. Experiments were done to see if thresholding could be used to extract the foreground, as well as classify the fruit in the hand.

Through trial and error, the ideal skin thresholds were found to be:

Table 15. HSV color space values for skin thresholding.

Threshold	Hue, Saturation, Value
Lower skin	0, 75, 100
Upper skin	30, 255, 255

Two trials were run: first, applying the threshold to the entire frame, second, thresholding only the bounding box area found by the hand detector, and setting all other pixels to black. For the hand detector trial, only the bounding box for the right hand was used.

Table 16 shows thresholding applied to the entire frame.

Table 16. Skin threshold applied to frames corresponding to Figure 44 interaction.

Frame #	Frame	Skin Threshold Applied to Frame
---------	-------	---------------------------------



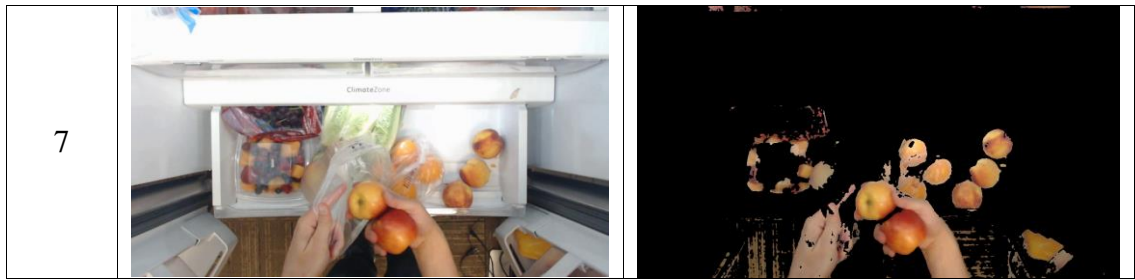


Table 16 shows that the skin threshold extracts not only the skin area, but also skin colored objects like the apples and peaches. It is difficult to tell the location of the apples in Frame 5 because the processing is color based and does not take background or foreground into account. The threshold nicely segments all items of a color within the range, and set all other items (like the refrigerator and bag of lettuce) to black.

Once again, the images are noisy. Filters and other image processing techniques can be used to get a more clear image, but those experiments are outside of the scope of this research [131].

Table 17 shows the threshold applied to the bounding box from the hand detector.

Table 17. Skin threshold applied to the detected hands corresponding to Figure 44 interaction.

Frame #	Frame	Skin Threshold Applied to Hand Detector ROI
1		



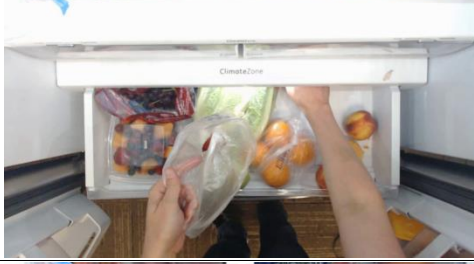









2		
3		
4		
5		
6		
7		

Table 17 shows that when the skin thresholds are applied only within the detected hand bounding box, much of the noise is removed. However, the output is limited to produce items that are small enough to fit within the hand bounding box. Like the threshold on the entire image, the thresholding applied to the box is color based and does not take background or foreground into account. Frame 6 in Table 17 shows that the items within the bin are still visible in the thresholded image.

d. Empty or Not using Extracted Foreground

Using the extracted foreground to identify produce was a challenge. Training a classifier on the images could give good results, but annotating a sufficient amount of segmented images was too time consuming for this research. Without training a classifier to detect different produce, the best outcome was to determine if the hand was empty or not. One idea was to use the color of the extracted foreground to determine if the hand was empty or not. Using scikit-learn's MiniBatchKMeans function, the extracted foreground was converted to two colored clusters ($n_clusters = 2$, one for black background and the other for the hand and object color) [132]. The pixel color of the resulting non-black area was analyzed to see if it was unique for empty and non-empty hands. The analysis was done by splitting each cluster into its three color components, H, S, and V. It was hypothesized that an empty hand would give H or S values within a standard range, and a non-empty hand would fall outside of that range. Observing the color values could then tell if the hand was empty or not. Examples of an empty hand and hand holding an orange are shown in Figure 45 and Figure 46.

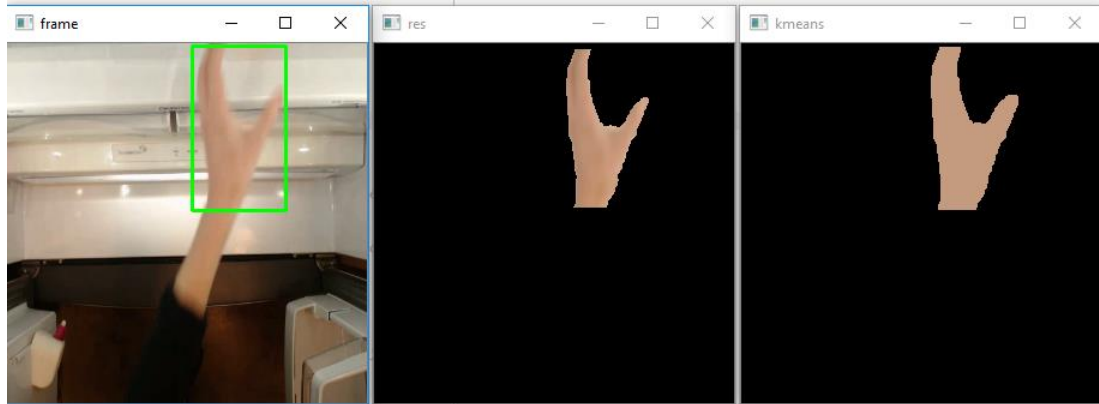


Figure 45. Original, thresholded, and k-means cluster for empty hand.

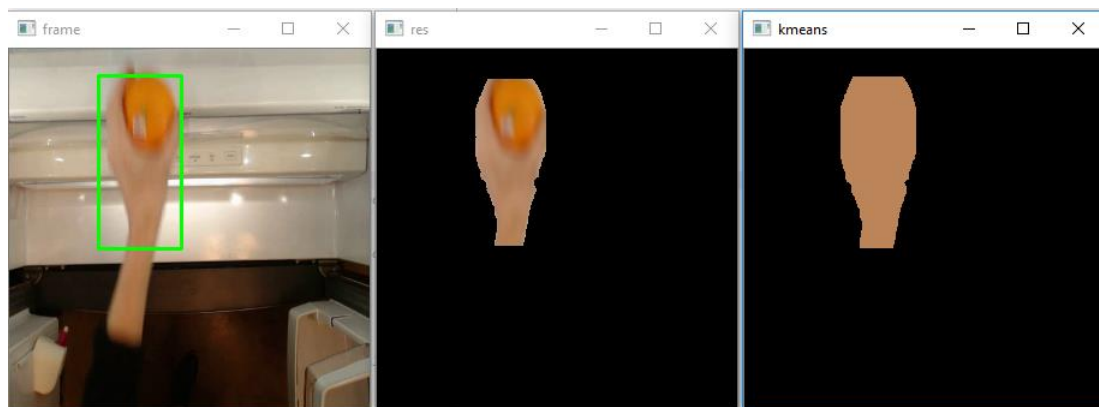


Figure 46. Original, thresholded, and k-means cluster for hand holding an orange.

The clusters in Figure 45 and Figure 46 are slightly different color, possibly indicating the difference between an empty and not empty hand. Trial and error was tried to find a threshold between empty and not average pixel values of the clusters, but none of the thresholds gave consistent accurate responses over multiple frames and test videos.

Because many produce items are a similar color to skin, monitoring the average pixel value of the extracted foreground was not useful to differentiate between an empty and not-empty.

4. Image Classifier

The hand detector is used as an anchor to find regions of interest to send to the image classifier. The classifier was trained on images of empty hands and hands holding apples or oranges. The precision and recall data and training loss and accuracy curves are shown in Figure 47 and Figure 48.

```
[INFO] evaluating after fine-tuning...
precision  recall  f1-score  support
apple      0.98   0.98   0.98     133
empty      0.97   0.96   0.97     79
orange     0.97   1.00   0.99     76
accuracy   0.98
macro avg  0.98   0.98   0.98     288
weighted avg 0.98   0.98   0.98     288
```

Figure 47. Precision and recall data for image classifier.

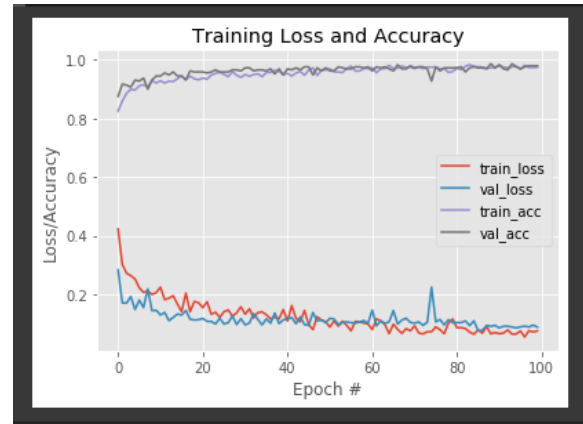


Figure 48. Training loss and accuracy curve for image classifier.

The plots above are after 25 epochs to warm up the fully connected layers, followed by 100 epochs to fine-tune the last convolution layer. Figure 47 shows that the overall F1-score for the model is 98%. Figure 48 shows that the model is slightly overfitted to the data, with the validation loss starting to increase after about 40 epochs.

The model was also tested on a video simulating the production application. The precision and recall data for the classifier is shown in Table 18.

Table 18. Precision and recall data for image classifier in production application.

Class	Precision	Recall	F1-Score	Support
Apple	0.50	0.85	0.63	20
Empty	0.99	0.87	0.93	118
Orange	1.00	0.74	0.85	23
<i>Overall</i>	0.83	0.82	0.80	161

Table 18 shows that precision and recall decrease for all classes when applied to new data. The apple class decreased the most from the training metrics, dropping from an F1-score of 0.98 down to 0.63. The orange class dropped from 0.99 to 0.85, and the empty class went from 0.97 to 0.93. The average F1-score across all classes dropped from 0.98 to 0.80. The confusion matrix for the classes is shown in Table 19.

Table 19. Confusion matrix for image classifier in production application.

True Class	Apple	17	1	0	2
	Empty	12	103	0	3
	Orange	5	0	17	1
	Unsure	0	2	0	1
		Apple	Empty	Orange	Unsure
		Predicted Class			

The “Unsure” class in Table 19 was added only for the confusion matrix calculations. The class of some of the detected hands was difficult to determine even by a human, thus the “Unsure” class was created to not penalize the model unnecessarily. In the code, the “Unsure” class is set when none of the other class predictions are above 50%. The “Apple” class was the most common false positive class. “Empty” was most frequently mistaken for “Apple”, followed by “Orange”. “Orange” was incorrectly classified as “Apple” 41% of the time. “Apple” had the least amount of false negatives.

The image classifier was only 80% accurate in the actual application, but is the best performing of the experiments. The hand detector plus image classifier is used to develop the logic for determining if an item is being added or removed.

D. Object Add or Remove Logic

The add/remove logic was developed only for the case of a single hand adding or removing one item (either apple, orange, or nothing) at a time.

1. Frames Per Second

Experiments showed that ten to twenty fps was ideal. Ten to fifteen fps were used to develop the add/remove logic because development on video playback was faster with fewer frames, but twenty fps offered a good tradeoff between data and memory. The number of frames per interaction (adding an item to the shelf) for various frame rates is shown in Table 20.

Table 20. Number of frames per interaction (adding an item to the shelf) for different fps.

Frame per second (fps)	Frames per interaction
10	20
20	35

Table 20 shows that, as expected, the number of frames per interaction almost doubles when fps is doubled. The data above is for one of the quickest interactions, just adding or removing an item from the shelf, and illustrates the importance of finding the proper frame rate. Too few fps and the hand can be missed going in or out, too high and the number of frames to process will become unnecessarily high and could cause errors.

2. Hand within the Refrigerator

Many experiments were tried to get the program to accurately detect when a hand is within the refrigerator, and when there is no hand within the refrigerator. Originally, observing if a hand detection was made within the camera frame was tried. The problem with that solution is that the camera frame only covers the entrance to the appliance. Any item that is added or removed beyond the entrance will cause the hand to go out of frame, although the hand should still be considered within the refrigerator. Next, rules were

hard-coded into the logic to try and deal with hands that go off frame as they move further into the appliance. The rules included:

- If the hand bounding box was last seen at the top of the frame and the next frame has no hand, assume the hand moved further within the unit and is still within the refrigerator
- If the last hand bounding box was moving into the appliance but the next frame has no hand, assume the hand moved further within the unit and is still within the refrigerator
- If the hand was last seen at the bottom of the frame near the refrigerator entrance and moving out, and next frame has no hand, assume hand left the appliance

The rules were able to correctly determine if a hand was within the refrigerator some of the time, but hard-coding rules to cover every edge case became complicated and unreliable.

Observations of video and images from within the refrigerator showed a simple solution: if the hand moves further within the refrigerator and out of frame, the arm almost always remains within the frame. Figure 49 shows an example where the hand has moved further within the appliance and it out of frame. The arm is still in frame, and can be used to determine that the user interaction with the refrigerator is still in progress.



Figure 49. An example where the hand is out of frame, but the arm can be seen.

Thus, a hand is said to be within the refrigerator if a hand or an arm, as in Figure 49, is present in the frame. A TensorFlow Object Detection API model was trained on all objects, including the hand and arm. The model was less accurate than the hand model trained from the EgoHands dataset, but was accurate enough to prove that using the hand and arm was the simplest and best way to determine if a hand is within the refrigerator.

3. The Algorithm

Creating the machine learning models was only half the challenge of this research. Once the hand detection, tracking, and object identification models were reasonably accurate, the task shifted to combining the models in a way to allow the computer to automatically extract useful information from the predictions and data. The proposed algorithm logic is shown in the flowchart in Figure 50.

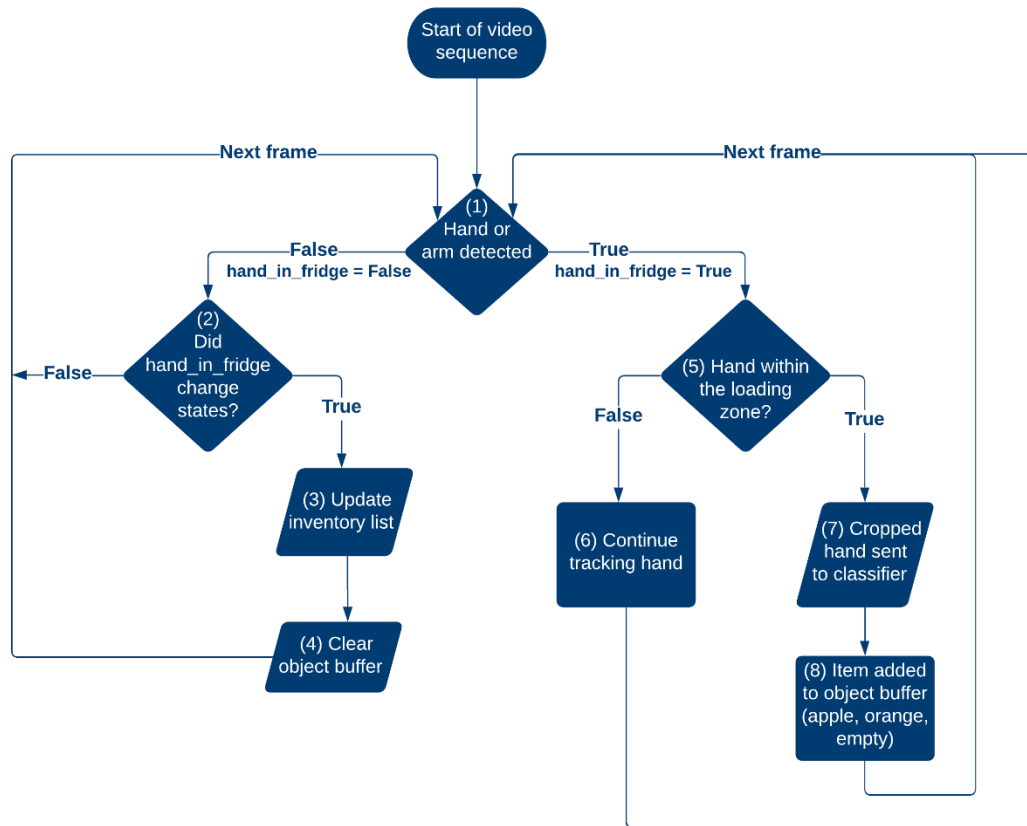


Figure 50. Flowchart of hand analysis logic.

Shown in Figure 49, the algorithm begins (1) by running the hand detector on each new frame where the door is open. The `hand_in_fridge` variable is essential for telling the program when an interaction is complete, and when decisions should be made to update the inventory. If a hand or arm is not detected (2), the `hand_in_fridge` variable remains false, and the next frame is processed. If a hand or arm is detected (5), the `hand_in_fridge` variable becomes true. If the detected hand is not within the loading zone (6), the centroid coordinates are recorded and the algorithm moves to the next frame. If the hand is within the zone (7), the cropped bounding box image is passed to the image classifier. The prediction from the classifier is added to the object buffer (8). The object buffer tracks the object classifications for each frame. Each time the `hand_in_fridge` variable changes

states, the inventory is updated and object buffer is reset. The algorithm continues until the `hand_in_fridge` variable changes to false (2). At this point, the algorithm makes decisions on what and where the object was added and/or removed (3). The object buffer is split in two, with the first half of the list representing the added objects, and the second half representing the removed objects. The logic assumes that the identified objects at the beginning of the object buffer specify what is being added as the hand moves into the refrigerator, while the identified objects at the second half of the buffer specify what is being removed as the hand moves out of the refrigerator. Experiments showed that the algorithm was more accurate when only the first and last three items of the buffer were used to determine the added or removed objects. For an object buffer with less than seven items, the floor division (divided by two) is used to split the buffer. For example, a buffer of length five divided by two would be 2.5. The floor of 2.5 is two so the first and last two items of the buffer are used. Next, the most frequent item in each the add and remove buffer was found using code modified from the GeeksforGeeks website [133]. The most frequent item in each buffer was taken to be the item added or removed. At the same time as the object buffer is being updated, the centroid of each hand detection is added to a centroid buffer. After the interaction is complete, the centroid buffer is split similar to how the object buffer is split. The location of the item is determined based on the thresholds in Figure 51.

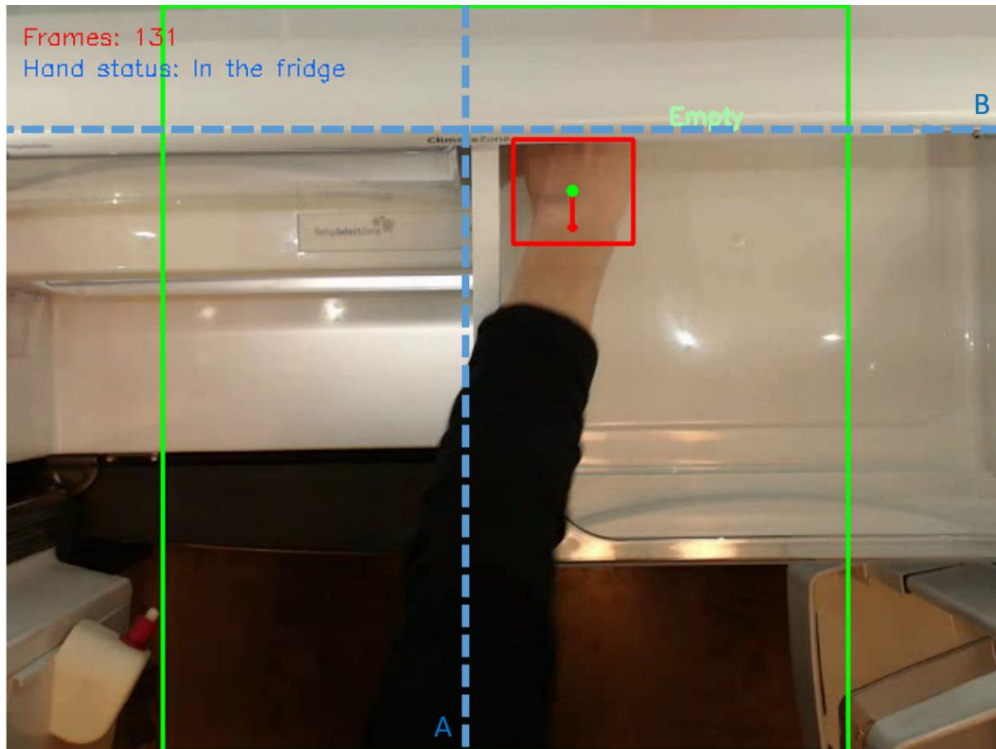


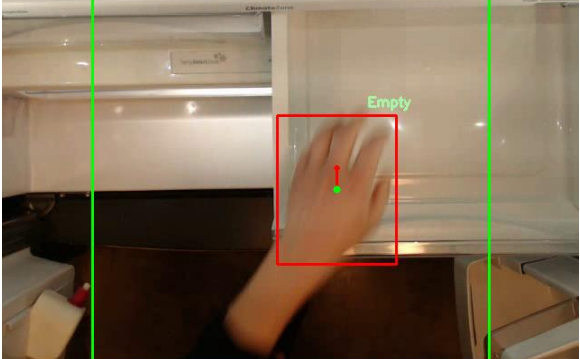
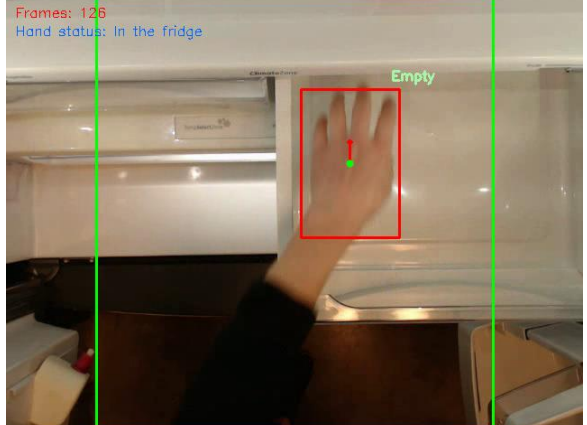
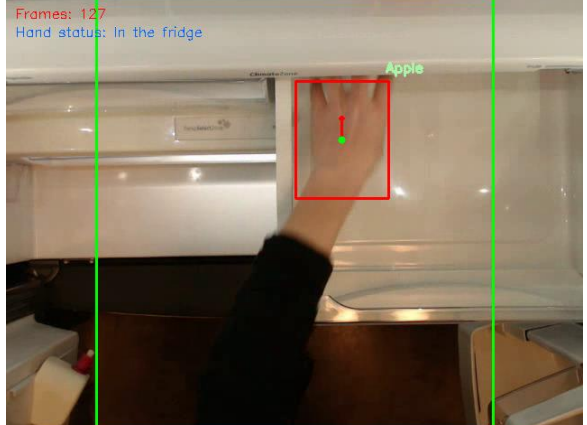
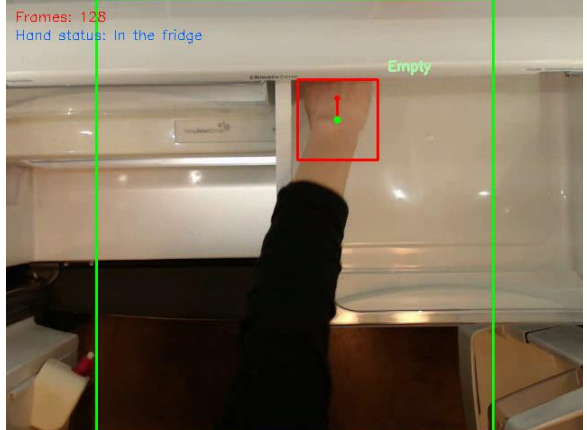


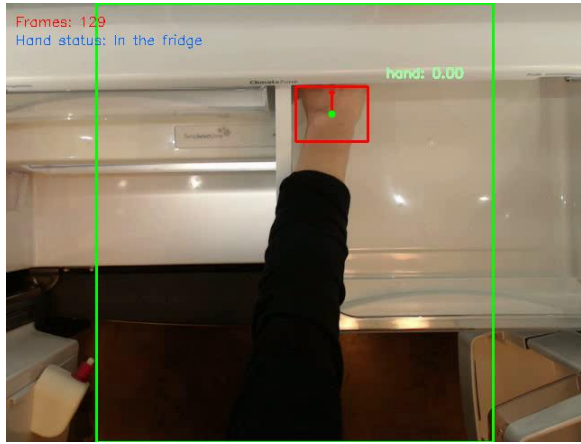
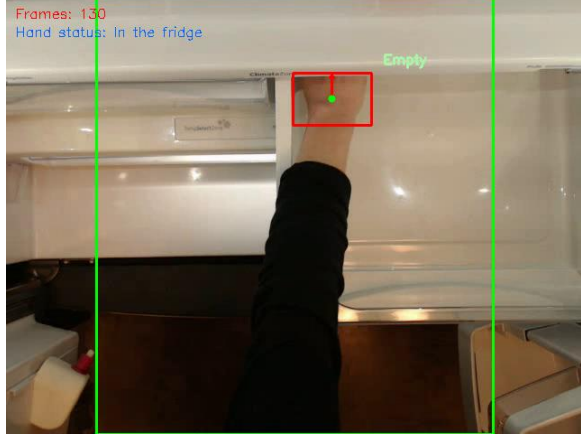
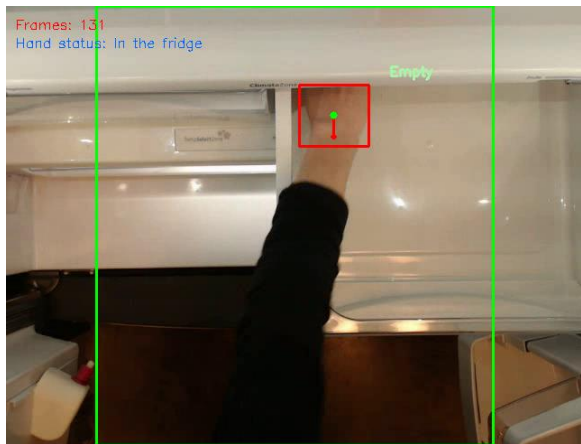
Figure 51. Thresholds for determining item location within the refrigerator.

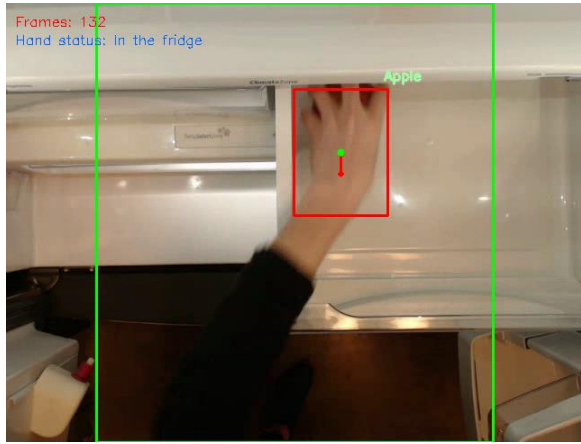
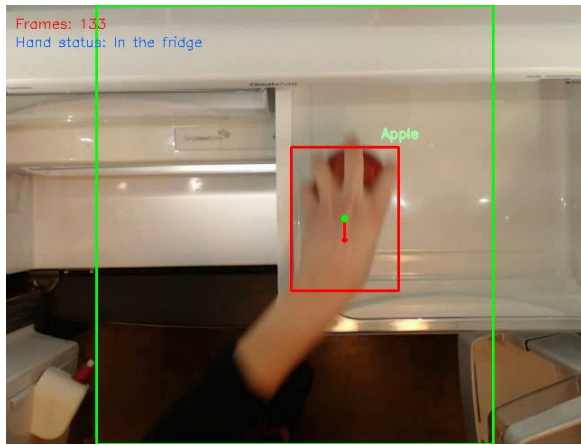
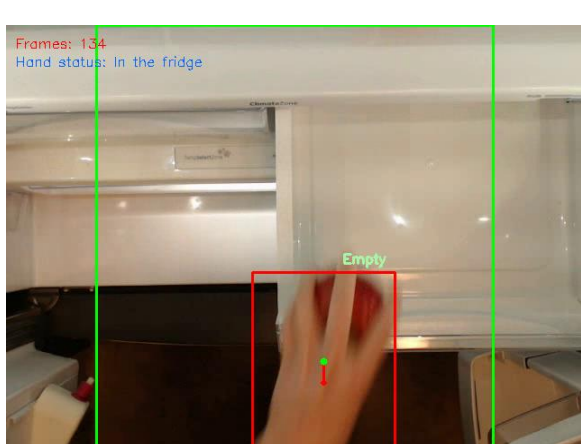
The centroid buffer is split in half, with the first half representing what was added, and the second what was removed. For both the added and removed centroid buffer section, the maximum x and y coordinate, and the minimum x coordinate are sent to a function to determine the maximum hand location. If both xmin and xmax are less than the middle refrigerator threshold (line A), the item location is “left side”, otherwise the location is “right side”. If ymax is below the shelf threshold (line B), the item location is “bin”, otherwise the location is “shelf”. If the detected item is “Empty”, no location information is provided. Finally, all object buffers are reset for the next interaction (4). An entire add and remove interaction with the logic outputs per frame is shown in Table 21.

Table 21. Example interaction and add/remove logic

<p>Frame 123</p> <p>Hand_in_fridge = False Hand within loading zone = False Object buffer = [] Centroid buffer = []</p>	<p>Frames: 123 Hand status: No hand</p> 
<p>Frame 124</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty'] Centroid buffer = [[334, 424]]</p>	<p>Frames: 124 Hand status: In the fridge</p> 
<p>Frame 125</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty', 'Empty'] Centroid buffer = [[334, 424], [371, 285]]</p>	<p>Frames: 125 Hand status: In the fridge</p> 

<p>Frame 126</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty', 'Empty', 'Empty'] Centroid buffer = [[334, 424], [371, 285], [381, 185]]</p>	<p>Frames: 126 Hand status: In the fridge</p> 
<p>Frame 127</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty', 'Empty', 'Empty', 'Apple'] Centroid buffer = [[334, 424], [371, 285], [381, 185], [372, 158]]</p>	<p>Frames: 127 Hand status: In the fridge</p> 
<p>Frame 128</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty', 'Empty', 'Empty', 'Apple', 'Empty'] Centroid buffer = [[334, 424], [371, 285], [381, 185], [372, 158], [367, 133]]</p>	<p>Frames: 128 Hand status: In the fridge</p> 

<p>Frame 129</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty', 'Empty', 'Empty', 'Apple', 'Empty'] – bounding box area too small, no change Centroid buffer = [[334, 424], [371, 285], [381, 185], [372, 158], [367, 133], [361, 121]]</p>	
<p>Frame 130</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty', 'Empty', 'Empty', 'Apple', 'Empty', 'Empty'] Centroid buffer = [[334, 424], [371, 285], [381, 185], [372, 158], [367, 133], [361, 121], [361, 113]]</p>	
<p>Frame 131</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty', 'Empty', 'Empty', 'Apple', 'Empty', 'Empty', 'Empty'] Centroid buffer = [[334, 424], [371, 285], [381, 185], [372, 158], [367, 133], [361, 121], [361, 113], [363, 119]]</p>	

<p>Frame 132</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty', 'Empty', 'Empty', 'Apple', 'Empty', 'Empty', 'Empty', 'Apple'] Centroid buffer = [[334, 424], [371, 285], [381, 185], [372, 158], [367, 133], [361, 121], [361, 113], [363, 119], [371, 163]]</p>	
<p>Frame 133</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty', 'Empty', 'Empty', 'Apple', 'Empty', 'Empty', 'Empty', 'Apple', 'Apple', 'Apple'] Centroid buffer = [[334, 424], [371, 285], [381, 185], [372, 158], [367, 133], [361, 121], [361, 113], [363, 119], [371, 163], [375, 233]]</p>	
<p>Frame 134</p> <p>Hand_in_fridge = True Hand within loading zone = True Object buffer = ['Empty', 'Empty', 'Empty', 'Apple', 'Empty', 'Empty', 'Empty', 'Apple', 'Apple', 'Apple', 'Empty'] Centroid buffer = [[334, 424], [371, 285], [381, 185], [372, 158], [367, 133], [361, 121], [361, 113], [363, 119], [371, 163], [375, 233], [352, 370]]</p>	


<p>Frame 135</p> <p>Hand_in_fridge = False Hand within loading zone = False Object buffer = [] Centroid buffer = []</p>	
<p>In_buffer = ['Empty', 'Empty', 'Empty'] Most_frequent(in_buffer) = 'Empty' Confidence factor = 100% (3 / 3) In_loc = [[334, 424], [371, 285], [381, 185], [372, 158], [367, 133]] Empty so no location calculated Prediction: Nothing added</p>	<p>Out_buffer = ['Apple', 'Apple', 'Empty'] Most_frequent(out_buffer) = 'Apple' Confidence factor = 67% (2 / 3) Out_loc = [361, 113], [363, 119], [371, 163], [375, 233], [352, 370]] Xmin = 352 Xmax = 375 Ymin = 113 Prediction: Apple removed from right bin</p>

Table 21 shows the logic along with an entire add/remove interaction. The remove interaction, specifically frame 127 and 134, shows why it was important to use multiple frames and average the predictions to determine what the hand is holding. The predictions in both frames are incorrect, but because the algorithm is looking at the average of the last three frames, the add/remove predictions are still correct. Frame 135 shows why it is important to have a hand detector that maximizes true positive predictions. The model missed the hand, thus there is one less data point for the algorithm. The interaction in Table 21 shows that the algorithm is able to detect objects hidden from view within the bin. Only the first and last three items in the object buffer are examined because, from visual inspection, that seems to be the average amount of frames it took for a hand to move through the entire loading zone. More data and

experiments are needed to verify that looking at only the first and last three frames gives the right amount of data. In the event of a tie in the buffer, the prediction is “Unsure”. Using an odd number of values for the in and out buffer ensures a tie is unlikely. It was found that the predictions at the top of frame are not as accurate because the hand is usually out of frame so the predictions are not reliable. The confidence factor (CF) was calculated using equation (10).

$$CF = \frac{\text{total number of the most frequent prediction}}{\text{total number of predictions}} \times 100\% \quad (10)$$

CF was initially used for development purposes only to quickly test different lengths of the in and out buffers. However, the CF may be useful in production to give more information on how confident the logic is in the prediction it makes.

The loading zone (the lime-green box) in Table 21 extends farther than the original loading zone in Figure 35. It was found that because the in and out buffers only see the first or last three or so predictions, the information near the top of the frame would be ignored automatically, and there was no reason not to extend the zone to the edge of the frame. The larger loading zone is used for all cases, with and without the drawers open. The smaller loading zone for when the drawers are open was not used because the object classifier is not robust enough to make an accurate prediction from a single frame. Also, the hand detector was less accurate around the bottom of the frame. In some instances within the area of the smaller loading zone, like frame 135 in Table 21, the classifier did not get a chance to make a prediction because the detector did not detect the hand. Using the smaller loading zone in that instance would mean no remove data for the code, and would result in an error.

Despite the decreased accuracy for both the hand detector and image classifier in a real-world application, Table 22 shows that the overall logic performed well on a video simulating the production application.

Table 22. Predicted and ground-truth classes for 14 interactions. Incorrect predictions are highlighted.

Interaction	Ground-Truth		Predicted			
	Add	Remove	Add	CF	Remove	CF
1	Orange	Empty	Orange	100%	Empty	100%
2	[Empty]	[Empty]	[Empty]	67%	[Empty]	100%
3	Empty	Apple	Empty	100%	Apple	67%
4	[Empty]	[Empty]	[Empty]	100%	[Empty]	67%
5	Empty	Orange	Empty	67%	Orange	100%
6	Orange	Empty	Orange	67%	Empty	100%
7	Apple	Empty	Apple	100%	Empty	100%
8	Empty	Orange	Empty	100%	Orange	67%
9	Orange	Apple	Orange	100%	Apple	67%
10	Apple	Empty	Apple	67%	Empty	100%
11	Empty	Orange	Empty	100%	Orange	100%
12	Orange	Empty	Apple	67%	Empty	100%
13	Empty	Apple	Empty	100%	Apple	67%
14	Apple	Empty	Apple	67%	Inconclusive	33%

Table 22 shows the predictions made by the program versus the ground-truth. Interaction 12 was incorrect, with an orange being wrongly classified as an apple. Interaction 14 was inconclusive, with the object buffer showing ['Apple', 'Empty', 'Unsure'] so there was no most frequent item to predict. Interaction two and four represent an empty hand opening and closing the produce bin. The location predictions are not shown in the table, but the program correctly predicted all item locations. The overall classification accuracy for the video was 93%.

4. Storing Inventory Information

For development, the inventory list was stored in an Excel spreadsheet, shown in Figure 52.

	A	B	C	D
1	Left Bin	Right Bin		Fridge
2		Apple		Orange
3	Bottom Bin			
4				
5				

Figure 52. Excel spreadsheet storing refrigerator inventory.

Figure 52 shows how each refrigerator compartment, bins and main compartment (“Fridge”) takes up a different cell. The code reads in each cell and corresponds that inventory list with the specific storage location. A flowchart of the logic is shown in Figure 53.

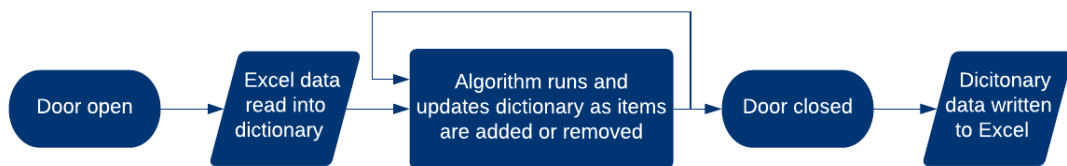


Figure 53. Logic for updating the Excel spreadsheet storing the inventory information.

Each time the refrigerator door opens, the data from the spreadsheet is read into a dictionary to store the information for each bin or refrigerator location. As the algorithm runs, the dictionary is updated according to what is added or removed. When the door is closed, the dictionary information is written back to the Excel file.

ANALYSIS

The beginnings of an inventory management system using the hands and an image classifier has been described from concept to realization. The objective of this thesis was to see if the hand can be used to identify objects being added and removed from the refrigerator produce bins. The research was successful at the objective, and presented a system that detects and tracks hands, identifies what a hand is holding, and automatically updates the inventory list based on what and where an item was added or removed. The research began by collecting images to use to train machine learning models. The images were then annotated, some with bounding boxes and another group labelled a single class per image. For the bounding box annotations, it was determined that by spending more time upfront including more information in the annotations, more flexible models could be trained down the line. Cutting corners during annotation by grouping together single objects under one class proved to make it difficult for a model to ever learn the class in a meaningful way. If both a group of apples and a single apple represent the class apple, the model will not give good results for either. To avoid annotation flaws, annotating a small subset of the data and then training and testing could be a good way to save development time. Seeing how the small subset performs could validate the annotation strategy, or illuminate issues. Bounding box annotations is a huge bottleneck in creating a CV system, but it was shown that using a semi-automatic annotation tool, like Anno-Mage, can speed up the process. One problem with Anno-Mage is that the bounding boxes are not always precise and need to be adjusted. The adjustments can sometimes make the predicted bounding boxes more time consuming to adjust than just drawing them from scratch. The final detection dataset consisted of over 3500 annotations on about 900

images, with over 20 different classes. The bounding box annotations were flexible enough that different classes could be developed just by filtering out different information in the annotations. The final image classifier dataset contained over 1000 images with three different classes. Larger application-specific datasets (i.e., datasets with thousands of images per class) would be needed to develop a more robust detector and classifier for a production application, but the small datasets proved feasibility. More image data from different users would help to develop machine learning models that generalize well to all use cases.

The hand detection model was successful at detecting hands in almost every frame in the validation videos. It was shown that, while the training metrics of different models may be similar, each model will have strengths and weaknesses when applied to new data. The best performing model, `faster_rcnn_inception_v2_coco`, detected 93% of hands in the validation video, but struggled with partial hands at the edges of the frame. The research showed that training the model first on a large dataset like EgoHands and then on the local dataset could improve performance by increasing the true positive detections. Research from later in the thesis showed that the arm was also important to detect, because in some cases the hand may be out of frame. Training the hand detector to distinguish between the left and right hand was not successful. One reason why could be that because the EgoHands dataset has two sets of hands, the model was never able to learn that the left hand would most likely be on the left side of the screen and vice versa. More experiments should be done to train a model with a left/right hand class, as differentiating between hands is important for getting accurate results from the add/remove logic. Not enough testing was done to ensure the detector was robust to

different hand sizes and skin tones, but initial testing showed positive results in identifying different types of hands.

Hand tracking was found to be challenging in the application because the hands moved quickly and sporadically in and out of frame, often crossing over each other or becoming hidden under food items or the produce bins. Due to the challenge, the tracking was limited to a single hand. The centroid tracker performed the best at tracking the hand. Comparing the outputs in Table 10 and Table 11 clearly show that the centroid tracker was able to track the hand as it moved through the video frames better than the dlib correlation tracker. As illustrated in Table 12, the distance information between old and new centroid was accurate at determining direction. Later research in the thesis showed that knowing when a hand was within the refrigerator, and recording the centroids at each detection, was more important than tracking the hand frame-by-frame. This approach (i.e., simple detection of hand vs. no hand and then tracking the centroid) led to the development of the add and remove logic. In comparison, creating an algorithm based on frame-by-frame hand-tracking and direction proved to be too complex to create an accurate algorithm. The hand tracking research showed the importance of implementing the simplest solution for the task. A simple solution means increased robustness of the overall system, algorithms that are easier to debug, and less risk of unexpected behavior from the algorithm.

Initial research focused on training a detection model to identify all annotated objects, but it became clear that because the camera only gave a 2D view, detections over the bins and other areas where food was present would interfere with detections. Multiple methods were tried to extract the foreground from the background, hypothesizing that the

extracted foreground would show both the hand and object of interest. The background subtractor method produced an output with a lot of noise, but was the best at extracting only the foreground. Other methods produced good images, but background objects often remained in the final output. Using the HSV color space to extract skin regions was successful, but the skin threshold also extracted produce items that were similar to skin color. No unique threshold was found to distinguish between the color of an empty hand, and a hand holding an item. Combining the background subtractor to extract the foreground, and using a color threshold to remove the hand and leave only the object of interest could be a solution to the problem of background objects interfering with the object detector or classifier. The segmented object would then be passed to an image classifier to be recognized. Within the scope of this research, the problem was solved by creating a loading zone within which the image classifier would run. The developed hand detector and classifier were not accurate enough to identify objects within the smaller open bin loading zone. Only empty bins were used for the development of the logic when items were being added or removed from the bin area. The loading zone was useful to filter out false positive hand detections at the edges of the refrigerator around the doors. The developed classifier was accurate enough for proof of concept. The classifier showed that an object could be detected even when held, and using the hand as an anchor to focus the classifier worked well. The small dataset for the classifier meant that there were many false positives, with the apple class often being predicted for the other two classes. Weighting the class weights for the three items equally could also be a cause of the false positives. In the application, the hand is much more common than the produce items, and the training data should reflect that. Passing the hand bounding box to the classifier is

limited to only objects that are contained within the box. Larger items, or items held by both hands at once, would not be correctly classified by this approach. Dynamic bounding box areas able to adjust to larger items, or using the background subtractor to extract a clean foreground image could solve the problem of larger items.

The add and remove logic, while limited to a single hand, performed well at automatically recognizing when and where an item was added or removed. The logic was able to use the centroid information to determine where within the unit the object was stored. The buffer holding the object classifications was able to make accurate predictions, despite the trained image classifier not being very robust. Averaging the items in the buffer to find the most common object accounted for incorrect classifications, and also provided information on how confident the algorithm was for the prediction. Only using the first and last few predictions to make decisions lessened the impact of the poor classifier performance at the top of the frame. The method illustrated the importance of a hand detector that detected every hand in every frame as a lost hand meant lost object information. If the hand is missed in any of the first or last few frames, dividing the object buffer in half to determine what was added and removed becomes less accurate. Setting an optimal fps rate is also important to ensure there are enough frames to have at least three images of the hand moving both in and out. Updating the inventory list every time a hand exits the refrigerator was the best way to ensure that each add and remove interaction was captured. Detecting both the hand and the arm ensured that even when the hand moved out of the frame and deeper into the refrigerator, there was still an arm to let the system know the interaction was still in progress. A major limitation to the logic is that it was only developed for a single hand. The problem stems largely from the

challenge of being able to differentiate between two hands within the refrigerator. Once an accurate way to distinguish hands is discovered, updating the add and remove logic would be straight-forward. Separate buffers per hand would track the objects in the hands separately. The logic to determine when a hand left the refrigerator would need to be updated to be two distinct variables, one to update the inventory when the left hand exited the appliance, and one for the right. The timestamp information tied to each added object could be used to alert the user when a food is about to go spoil. The timestamp plus the centroid location information would allow the system to pinpoint where within a cluttered bin the item is located. Figure 54 in the next section shows an example of how this information could be displayed to the user.

CONCLUSIONS AND FUTURE WORK

The research contained in this thesis showed that a user's hand is a useful tool in identifying objects as they are added and removed from the refrigerator. Analyzing the hand provides another layer of information for the complex overall automatic inventory management system. Observing the hand-object interaction is especially useful to identify objects that would otherwise be hidden within the bin, under other items, or occluded from the camera view. Recording when and where the hands enter the refrigerator provides a timestamp along with an inventory location information that can be used to alert the user to items approaching their best-by-date.

The developed hand detector was 93% accurate on the small validation video, detecting 173 of the 185 hands. The tracker, through visual inspection, was shown to be able to accurately provide centroid information to detect where within the refrigerator the item was added or removed. The image classifier correctly identified 17 of the 20 apples in the validation video, 17 of the 23 oranges, and 103 of the 118 empty hands. The add and remove logic correctly identified and updated the inventory information for 26 of the 28 add or remove cases in the validation video. The logic was 100% accurate on determining the location from where the item was added or removed.

Future Work

The research above represents phase 1 of the hand analysis system. Phase 2 of the research should focus on the following:

- Extend the add and remove logic to work for two or more hands

- Collect a larger and more diverse (e.g., different users of various ages, skin tones, hand dominance, etc.) dataset of application specific images and videos to improve the models and add more produce classes
- More testing to ensure the system is accurate for all users and is able to identify a wide range of produce items

The goal of phase 2 should be to develop robust models and algorithms based on application specific images.

One of the biggest limitations of this research is that the add and remove logic is limited to a single hand. Future work on the problem could include researching how to distinguish between hands as they move within the refrigerator. The creators of the EgoHands dataset were able to train a machine learning model to distinguish between the left and right hand within the dataset they developed [67]. Their success suggests that a large dataset can provide a CNN with enough information to learn to distinguish between the two hands. Adding more application specific hand data within the refrigerator could increase the results of the left and right hand detector.

Furthermore, future work would include researching if there are better ways to solve the problem of tracking hands within the refrigerator. Newer CNN models are 3-dimensional, having a third dimension that uses time. The models are trained on sequences of images, with each sequence labelled as a single class. A 3D CNN could learn entire behaviors, like adding and removing items, or opening a drawer. These models would be more robust than hand-designed logic as their distinguishing features would be learned from the data.

Future work could include improving the background subtraction method and skin thresholding. Being able to extract only the things that are changing within the refrigerator would be useful not just for the hand analysis portion of the inventory management system, but also detecting objects within the bins and other refrigerator locations.

More image data in general is needed to improve the system. All computer vision applications have unique challenges, but based on current successes/advancements, a realistic goal for inventory management would need at least 1,000 annotated images per class to detect between classes with enough accuracy to deploy in a consumer application [17]. Datasets like VegFru and Fruit-360 could be used to supplement a produce dataset. A larger and more diverse (e.g., different users of various ages, skin tones, hand dominance, etc.) dataset of application specific images and videos would not only improve the models, but would also give more insight on how consumers use their refrigerators. Many of the assumptions and decisions in this research were based on limited user examples, and may not fully reflect the majority of users' behaviors.

More work needs to be done to make the system production ready. An automatic inventory system needs to be highly accurate to be adopted by consumers. Collecting more data will improve the recognition models. Hardware needs to be selected to balance processing power with cost. More research should be done to explore other camera options like a depth-based camera. Camera specs must be finalized and analyzed to observe any impact the camera choice has on the detection algorithm. Fps or low quality images or low-light conditions may impact the overall inventory management system – issues that cannot be studied fully, in detail, until hardware is selected/finalized. As

discussed in the Background, on-board or cloud storage issues will have to be addressed to make a viable smart refrigerator system. Future work can begin to face these issues by exploring techniques to reduce model size so the entire system can exist locally on-board the unit. A local system would help with reducing privacy concerns. A system on the cloud could be more powerful because of more processing power, and the models could constantly be improved. Cloud-based systems are more expensive and require stricter security measures, but could provide new revenue streams, like subscription services or premium updates, for appliance makers. A website or phone application needs to be developed to give consumers an easy way to access the inventory information. The application could give the user notifications if an item is about to go bad, suggest recipe ideas based on items that need to be used up, or suggest a grocery list based on items that have been used throughout the week. An example of how the information could be displayed to a user is shown in Figure 54.



Figure 54. Example mobile or web app for displaying inventory information to the consumer.

The image on the left in Figure 54 is the initial application view, with the colored circles in the top left corner of each bin representing the overall freshness of the items inside. Clicking on a bin would show the middle image, and highlight the location of the item(s) that has been in the bin for an extended period of time. The application could also

proactively notify the user of an item that has been inside the refrigerator for a long time, and show the user a picture of the item when it was put into inventory. A screen on the refrigerator, or voice-control system could also be used to receive and provide information to the consumer. Continuing to explore these areas on how to create advanced inventory management within a refrigerator will make for a more accurate system from a cutting-edge engineering perspective, and a more attractive product for consumers. Automatic inventory management has the capability to transform the user experience by giving the user proactive data to better guide their food consumption decisions. The benefits of the technology could include happier and healthier consumers, new revenue streams for appliances makers, and less food waste overall.

REFERENCES

- [1] J. Hyman, H. F. Wells and J. C. Buzby, "The Estimated Amount, Value, and Calories of Postharvest Food Losses at the Retail and Consumer Levels in the United States," *Economic Information Bulletin*, vol. 121, February 2014.
- [2] C.-C. Liang, "Smart Inventory Management System of Food-Processing-and-Distribution Industry," *Procedia Computer Science*, vol. 17, pp. 373-378, 2013.
- [3] Consumer Technology Association (CTA), "About CES," 2020. [Online]. Available: <https://www.ces.tech/About-CES.aspx>. [Accessed 6 April 2020].
- [4] Samsung Newsroom U.S., "New Food AI Looks Inside Your Fridge To Help You Find The Perfect Things To Cook With What You ALREADY Have," Samsung, 7 January 2020. [Online]. Available: <https://news.samsung.com/us/new-food-ai-looks-inside-fridge-help-find-perfect-things-cook-already/>. [Accessed 5 December 2019].
- [5] J. Berkenkamp, D. Hoover and Y. Mugica, "NRDC," The Rockefeller Foundation, October 2017. [Online]. Available: <https://www.nrdc.org/sites/default/files/food-matters-ib.pdf>. [Accessed 23 January 2020].
- [6] ReFED, "Improved Inventory Management," ReFED, 2020. [Online]. Available: <https://www.refed.com/solutions/improved-inventory-management/>. [Accessed 8 February 2020].
- [7] "Siri - Apple," Apple Inc., 2020. [Online]. Available: <https://www.apple.com/siri/>.
- [8] "Amazon Alexa Official Site: What is Alexa?," Amazon, 2020. [Online]. Available: <https://developer.amazon.com/en-US/alexa>. [Accessed 2020].
- [9] "Fridge Pal," Apple iTunes Store, 2014. [Online]. Available: <https://apps.apple.com/us/app/fridge-pal-shopping-lists/id496451091>. [Accessed 2020].
- [10] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of Physiology*, vol. 148, no. 3, pp. 574-591, 1959.
- [11] S. Papert, "The Summer Vision Project," 1 July 1966.
- [12] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems*, pp. 1097-1105, 2012.
- [13] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. V. Esesn, A. A. S. Awwal and V. K. Asari, "The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches," *arXiv preprint*, 12 September 2018.
- [14] R. Girshick, "rgb's home page: Fast R-CNN - Slides," 2015. [Online]. Available: <https://www.rossgirshick.info/>. [Accessed 2020].

- [15] K. Fukushima, "capable of visual pattern recognition capable of visual pattern recognition," *Neural networks*, vol. 1.2, pp. 119-130, 1988.
- [16] P. D. Steven W. Smith, "The Scientist and Engineer's Guide to Digital Signal Processing," San Diego, California Technical Publishing, 1997, p. 107.
- [17] D. A. Rosebrock, "Deep Learning for Computer Vision with Python Volume 1," PyImageSearch, 2019, p. Chapter 11. Convolutional Neural Networks.
- [18] R. Yamashita, M. Nishio, R. K. G. Do and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, pp. 611-629, 2018.
- [19] A. Deshpande, "A Beginner's Guide To Understanding Convolutional Neural Networks," Github, 20 July 2016. [Online]. Available: <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>. [Accessed 2020].
- [20] F. Chollet, Deep Learning with Python, Manning Publications Co., 2018.
- [21] P. Zach Monge, "Does Deep Learning Really Require “Big Data”? — No!," Medium, 19 April 2018. [Online]. Available: <https://towardsdatascience.com/does-deep-learning-really-require-big-data-no-13890b014ded>. [Accessed April 2020].
- [22] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," *arXiv*, 2013.
- [23] saagie, "Object Detection (Part 2)," Saagie, 8 December 2017. [Online]. Available: <https://www.saagie.com/blog/object-detection-part2/>. [Accessed 2020].
- [24] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.
- [25] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn and A. Zisserman, "The PASCAL Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303-338, 2010.
- [26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "ImageNet: a Large-Scale Hierarchical Image Database," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Miami, 2009.
- [27] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollar, "Microsoft COCO: Common Objects in Context," in *Computer Vision—ECCV 2014.*, Springer International Publishing., 2014, pp. 740-755.
- [28] T. Chen, I. Goodfellow and J. Shlens, "Net2Net: Accelerating Learning via Knowledge Transfer," *arXiv*, 2015.
- [29] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single Shot MultiBox Detector," *ECCV 2016*.
- [30] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *Advances in Neural Information Processing Systems*, pp. 770-778, 2016.

- [31] Google, "What is Colaboratory?," [Online]. Available: <https://colab.research.google.com/notebooks/intro.ipynb>. [Accessed 10 July 2019].
- [32] Microsoft, "Cloud Computing Services | Microsoft Azure," [Online]. Available: <https://azure.microsoft.com/en-us/>. [Accessed 2020].
- [33] "Fully Connected Layers in Convolutional Neural Networks: The Complete Guide," missinglink.ai, [Online]. Available: <https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/>. [Accessed 2020].
- [34] D. C. Ciresan, U. Meier, L. M. Gambardella and J. Schmidhuber, "Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition," *Neural Computation*, vol. 22, no. 12, 2010.
- [35] E. Culurciello, "A BRIEF HISTORY OF NEURAL NETWORK ARCHITECTURES," TopBots, 9 June 2017. [Online]. Available: <https://www.topbots.com/a-brief-history-of-neural-network-architectures/>. [Accessed 2020].
- [36] A. Canziani, A. Paszke and E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications," 14 April 2017. [Online]. Available: <https://arxiv.org/abs/1605.07678>. [Accessed 2020].
- [37] "Applications," Keras Documentation, [Online]. Available: <https://keras.io/applications/>.
- [38] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," *arXiv*, 2013.
- [39] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *arXiv*, 22 October 2014. [Online]. Available: <https://arxiv.org/pdf/1311.2524.pdf>. [Accessed 2019].
- [40] J. R. R. Uijlings, K. E. A. v. d. Sande, T. Gevers and A. W. M. Smeulders, "Selective Search for Object Recognition," *International Journal of Computer Vision*, 2013.
- [41] R. Girshick, "Fast R-CNN," *arXiv*, 27 September 2015.
- [42] F. F. Li, A. Karpathy and J. Johnson, "CS231n Winter 2016 Lecture 8 Slides, Slide 84," Stanford, 1 February 2016. [Online]. Available: http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf. [Accessed 2019].
- [43] H. P. Kim, "Tutorial on Object Detection (Faster R-CNN)," SlideShare, 2018. [Online]. Available: <https://www.slideshare.net/hpkim0512/tutorial-of-faster-rcnn>. [Accessed 2020].
- [44] J. Rey, "Faster R-CNN: Down the rabbit hole of modern object detection," tryolabs, [Online]. Available: <https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/>. [Accessed 2020].

- [45] Alegion, "Faster R-CNN," Using Region Proposal Network for Object Detection, 2019. [Online]. Available: <https://www.alegion.com/faster-r-cnn>. [Accessed 2020].
- [46] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *arXiv*, 2015.
- [47] M. Rouse, "internet of things (IoT)," TechTarget, 2017. [Online]. Available: <https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT>. [Accessed 2020].
- [48] S. Miniaoui, S. Atalla and K. F. B. Hashim, "Introducing Innovative Item Management Process Towards Providing Smart Fridges," in *2nd International Conference on Communication Engineering and Technology*, Nagoya, Japan, 2019.
- [49] S. Luo, H. Xia, Y. Gao, J. S. Jin and R. Athauda, "Smart Fridges with Multimedia Capability for Better Nutrition and Health," in *International Symposium on Ubiquitous Multimedia Computing*, Hobart, ACT, Australia, 2008.
- [50] B. Son, C.-S. Han, Y.-T. Jeon and D.-H. Lee, "A RFID/NFC Fusion based Smart Refrigerator for Wellness Service," in *Sensors*, 2014.
- [51] S. A.S, "Intelligent Refrigerator Using Artificial Intelligence," in *11th International Conference on Intelligent Systems and Control*, Coimbatore, India, 2017.
- [52] J. Rouillard, "The Pervasive Fridge: A Smart Computer System Against Uneaten Food Loss," in *The Seventh International Conference on Systems*, 2012.
- [53] E. Ganglbauer, G. Fitzpatrick and R. Comber, "Negotiating Food Waste: Using a Practice Lens to Inform Design," *ACM Transactions on Computer-Human Interaction*, vol. 20, no. 2, 2013.
- [54] "Side-by-Side Refrigerator with Family Hub," Samsung, 2020. [Online]. Available: <https://www.samsung.com/us/home-appliances/refrigerators/side-by-side/26-7-cu-ft--large-capacity-side-by-side-refrigerator-with-touch-screen-family-hub--in-black-stainless-steel-rs27t5561sg-aa/>. [Accessed 2020].
- [55] C. Gartenberg, "LG's new ThinQ smart fridge has a transparent 29-inch touchscreen and runs webOS," *The Verge*, 7 January 2017. [Online]. Available: <https://www.theverge.com/circuitbreaker/2018/1/7/16860260/lg-instaview-thinq-smart-refrigerator-webos-alexa-home-ces-2018>. [Accessed 2019].
- [56] S. Hou, Y. Feng and Z. Wang, "VegFru: A Domain-Specific Dataset for Fine-Grained Visual Categorization," in *IEEE International Conference on Computer Vision*, Venice, 2017.
- [57] H. Muresan and M. Oltean, "Fruit recognition from images using deep learning," *Acta Univ. Sapientiae, Informatica*, vol. 10, no. 1, pp. 26-42, 2018.
- [58] C. Liu, X. Wang, J. Ni, Y. Cao and B. Liu, "An Edge Computing Visual System for Vegetable Categorization," in *18th IEEE International Conference On Machine Learning And Applications*, Boca Raton, FL, USA, 2019.

- [59] D. Lee, "Samsung and LG go head to head with AI-powered fridges that recognize food," *The Verge*, 2 January 2020. [Online]. Available: <https://www.theverge.com/2020/1/2/21046822/samsung-lg-smart-fridge-family-hub-instaview-thinq-ai-ces-2020>. [Accessed 2019].
- [60] "Turn Any Fridge Into a Smart Fridge," *FridgeEye*, 2020. [Online]. Available: <https://fridgeeye.com/>. [Accessed 2020].
- [61] J. D. a. M. Shah, "Recognizing Hand Gestures," in *ECCV-94*, Stockholm, Sweden, 1994.
- [62] N. Wingfield, "Amazon Moves to Cut Checkout Line, Promoting a Grab-and-Go Experience," *The New York Times*, 5 December 2016.
- [63] G. Dong, Y. Yan and M. Xie, "Vision-Based Hand Gesture Recognition for Human-Vehicle Interaction," in *the Proceedings of the International conference on Control, Automation and Computer Vision*, 1998.
- [64] M. C. Shin, K. I. Chang and L. V. Tsap, "Does Colorspace Transformation Make Any Difference on Skin Detection?," in *Sixth IEEE Workshop on Applications of Computer Vision*, Orlando, 2002.
- [65] A. Albiol, L. Torres and E. J. Delp, "Optimum color spaces for skin detection," in *Proceedings 2001 International Conference on Image Processing*, Thessaloniki, Greece, 2001.
- [66] A. Mittal, A. Zisserman and P. Torr, "Hand detection using multiple proposals," *Proceedings of the British Machine Vision Conference 2011*, 2011.
- [67] S. Bambach, S. Lee, D. J. Crandall and C. Yu, "Lending A Hand: Detecting Hands and Recognizing Activities in Complex Egocentric Interactions," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.
- [68] S. Usmanhujaev, S. Baydadaev and K. J. Woo, "Real-Time, Deep Learning Based Wrong Direction Detection," *Applied Science*, vol. 10, no. 7, p. 2453, 2020.
- [69] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *In Proceedings of the 17th International Conference on Pattern Recognition 2004*, Cambridge, 2004.
- [70] D. Das and D. S. Saharia, "Implementation And Performance Evaluation Of Background Subtraction Algorithms," *International Journal on Computational Sciences & Applications (IJCSA)*, vol. 4, no. 2, April 2014.
- [71] R. M. Bolle, J. H. Connell, N. Haas, R. Mohan and G. Taubin, "Veggievision: a produce recognition system," in *Proceedings of the 3rd IEEE Workshop on Applications of Computer Vision*, Sarasota, USA, 1996.
- [72] Y. Wu, J. Lim and M.-H. Yang, "Online object tracking: A benchmark," *CVPR*, 2013.
- [73] L. Jiang, H. Xia and C. Guo, "A Model-Based System for Real-Time Articulated Hand Tracking Using a Simple Data Glove and a Depth Camera," *Sensors*, vol. 19, no. 21, 2019.
- [74] S. S. Kakkoth and S. Gharge, "Real Time Hand Gesture Recognition & Its Applications In Assistive Technologies For Disabled," in *Fourth International*

Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 2018.

- [75] B. Kang, K.-H. Tan, N. Jiang, H.-S. Tai, D. Tretter and T. Q. Nguyen, "Hand Segmentation for Hand-Object Interaction from Depth map," *arXiv*, 2016.
- [76] J. Hui, "Build a Deep Learning dataset (Part 2)," Medium, 1 March 2018. [Online]. Available: https://medium.com/@jonathan_hui/build-a-deep-learning-dataset-part-2-a6837ffa2d9e. [Accessed 2020].
- [77] logitech, "C920 HD PRO WEBCAM," 2020. [Online]. Available: <https://www.logitech.com/en-us/product/hd-pro-webcam-c920>.
- [78] W. TREINEN, "GE APPLIANCES SHOWCASES ITS LEADERSHIP IN COOKING AT KBIS 2020," GE Appliances, 21 January 2020. [Online]. Available: <https://pressroom.geappliances.com/news/ge-appliances-showcases-its-leadership-in-cooking-at-kbis-2020>. [Accessed 2020].
- [79] "Logitech Webcam Software for Windows 10," CNET, 31 July 2017. [Online]. Available: https://download.cnet.com/Logitech-Webcam-Software-for-Windows-10/3000-2348_4-77592932.html. [Accessed 2019].
- [80] OpenCV, "Reading and Writing Images and Video¶," 31 December 2019. [Online]. Available: https://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html#videocapture-get. [Accessed 2019].
- [81] USDA. [Online]. Available: <http://www.ers.usda.gov/media/184291/ap032.pdf>.
- [82] Python Programming, "Introduction and Use - Tensorflow Object Detection API Tutorial," 2018. [Online]. Available: <https://pythonprogramming.net/introduction-use-tensorflow-object-detection-api-tutorial/>. [Accessed 4 April 2020].
- [83] Tzutalin, "LabelImg," Git Code, 2015. [Online]. Available: <https://github.com/tzutalin/labelImg>. [Accessed 3 August 2019].
- [84] L. Vladimirov, "TensorFlow Object Detection API tutorial," Read the Docs!, 2018. [Online]. Available: <https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/>. [Accessed 4 April 2020].
- [85] V. Mavani, "Anno-Mage: A Semi Automatic Image Annotation Tool," Github Repository, 13 May 2018. [Online]. Available: <https://github.com/virajmavani/semi-auto-image-annotation-tool>. [Accessed 2019].
- [86] D. A. Rosebrock, "Chapter 15: Training a Faster R-CNN From Scratch," in *Deep Learning for Computer Vision with Python ImageNet Bundle, 3rd Edition*, PyImageSearch, 2019, pp. 261-290.
- [87] V. Dibia, "HandTrack: A Library For Prototyping Real-time Hand TrackingInterfaces using Convolutional Neural Networks," *GitHub repository*, 2017.
- [88] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama and K. Murphy, "Speed/accuracy trade-offs for

- modern convolutional object detectors," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [89] D. Tran, "Raccoon Detector Dataset," GitHub Repository, 2017. [Online]. Available: https://github.com/datitran/raccoon_dataset. [Accessed 2019].
 - [90] S. Bianco, R. Cadene, L. Celona and P. Napoletano, "Benchmark Analysis of Representative Deep Neural Network Architectures," *arXiv*, 2018.
 - [91] C. Zhang, "How to train an object detection model easy for free," GitHub Repository, 2019. [Online]. Available: https://github.com/Tony607/object_detection_demo.
 - [92] Tensorflow, "Tensorflow detection model zoo," Tensorflow, [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. [Accessed 4 April 2020].
 - [93] "python," Python Software Foundation, 2001. [Online]. Available: <https://www.python.org/>. [Accessed 2020].
 - [94] Y. E. Wang, G.-Y. Wei and D. Brooks, "Benchmarking TPU, GPU, and CPU Platforms for Deep Learning," *arXiv*, 22 October 2019.
 - [95] The Spyder Website Contributors, "Spyder," 2018. [Online]. Available: <https://www.spyder-ide.org/>. [Accessed 1 July 2019].
 - [96] "Introduction," OpenCV, [Online]. Available: <https://docs.opencv.org/3.4/d1/dfb/intro.html>. [Accessed 2020].
 - [97] TensorFlow, "Supported object detection evaluation protocols," Github repository, 15 July 2019. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/evaluation_protocols.md. [Accessed 6 April 2020].
 - [98] W. Koehrsen, "Beyond Accuracy: Precision and Recall," Medium, 3 March 2018. [Online]. Available: <https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c>. [Accessed 2020].
 - [99] A. Rosebrock, "Intersection over Union (IoU) for object detection," PyImageSearch, 7 November 2016. [Online]. Available: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. [Accessed 11 October 2019].
 - [100] "Detection Evaluation," COCO Common Objects in Context, [Online]. Available: <http://cocodataset.org/#detection-eval>. [Accessed 6 April 2020].
 - [101] M. Everingham and J. Winn, "The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Development Kit," 8 May 2010. [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2010/devkit_doc_08-May-2010.pdf. [Accessed 6 April 2020].
 - [102] "Accuracy Metrics," Humboldt State University, 2019. [Online]. Available: http://gis.humboldt.edu/OLM/Courses/GSP_216_Online/lesson6-2/metrics.html.
 - [103] TensorFlow, "TensorBoard: TensorFlow's visualization toolkit," [Online]. Available: <https://www.tensorflow.org/tensorboard>. [Accessed 7 April 2020].
 - [104] C. U. P. Malla, "How to Improve Object Detection Evaluation," Medium, 3 April 2019. [Online]. Available: <https://medium.com/moonvision/smart-object->

- detection-evaluation-with-confusion-matrices-6f2a7c09d4d7. [Accessed 6 April 2020].
- [105] A. Rosebrock, "Simple object tracking with OpenCV," pyimagesearch, 23 July 2018. [Online]. Available: <https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/>. [Accessed 2019].
 - [106] D. E. King, "Classes - dlib documentation," dlib, 2013. [Online]. Available: http://dlib.net/python/index.html#dlib.correlation_tracker. [Accessed 2020].
 - [107] M. Danelljan, G. Häger, F. S. Khan and M. Felsberg, "Accurate Scale Estimation for Robust Visual Tracking," *Proceedings of the British Machine Vision Conference*, September 2014.
 - [108] A. Rosebrock, "OpenCV People Counter," pyimagesearch, 13 August 2018. [Online]. Available: <https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/>. [Accessed 2019].
 - [109] "What is PyQt?," Riverbank Computing Limited, 2018. [Online]. Available: <https://riverbankcomputing.com/software/pyqt/intro>. [Accessed 2020].
 - [110] C. Luo, "PyQt5," 10 May 2019. [Online]. Available: https://www.luochang.ink/posts/pyqt5_layout_sidebar/. [Accessed 13 December 2019].
 - [111] "Interactive Foreground Extraction using GrabCut Algorithm," OpenCV, [Online]. Available: https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html. [Accessed 2020].
 - [112] C. Rother, V. Kolmogorov and A. Blake, "GrabCut: Interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics*, vol. 23, pp. 309-314, 2004.
 - [113] "How to Use Background Subtraction Methods," OpenCV, [Online]. Available: https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html. [Accessed 2019].
 - [114] L. A. Marcomini and A. L. Cunha, "A Comparison between Background Modelling Methods for Vehicle Segmentation in Highway Traffic Videos," *arXiv*, 5 October 2018.
 - [115] Harrison, "OpenCV with Python Intro and loading Images tutorial," PythonProgramming.net, 2019. [Online]. Available: <https://pythonprogramming.net/loading-images-python-opencv-tutorial/>. [Accessed 2019].
 - [116] A. Elgammal, C. Muang and D. Hu, "Skin Detection - a Short Tutorial," 2009.
 - [117] "Color Space Conversions," OpenCV, [Online]. Available: https://docs.opencv.org/trunk/d8/d01/group_imgproc_color_conversions.html#ga397ae87e1288a81d2363b61574eb8cab. [Accessed 2019].
 - [118] "Smoothing Images," OpenCV, 2020. [Online]. Available: https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html. [Accessed 2019].

- [119] Harrison, "Blurring and Smoothing OpenCV Python Tutorial," Python Programming, [Online]. Available: <https://pythonprogramming.net/blurring-smoothing-python-opencv-tutorial/>. [Accessed 2019].
- [120] "sklearn.cluster.MinibatchKMeans," scikit-learn, 2019. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MinibatchKMeans.html>. [Accessed 2019].
- [121] "Getting Started with Videos," OpenCV, [Online]. Available: https://docs.opencv.org/master/dd/d43/tutorial_py_video_display.html. [Accessed 2019].
- [122] J. Mr. C, "change frame rate in opencv 3.4.2," Stack Overflow, 29 August 2018. [Online]. Available: <https://stackoverflow.com/questions/52068277/change-frame-rate-in-opencv-3-4-2>. [Accessed January 2020].
- [123] "tf.keras.preprocessing.image.img_to_array," TensorFlow, 31 March 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/img_to_array. [Accessed 2019].
- [124] scikit-learn, "sklearn.model_selection.train_test_split," 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. [Accessed 2019].
- [125] scikit-learn, "sklearn.preprocessing.LabelBinarizer," 2019. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html?highlight=labelbinarizer#sklearn.preprocessing.LabelBinarizer>. [Accessed 2019].
- [126] MkDocs, "The Sequential model API," Keras Documentation, 2019. [Online]. Available: <https://keras.io/models/sequential/>. [Accessed 2019].
- [127] MkDocs, "https://keras.io/preprocessing/image/," Keras Documentation, [Online]. Available: <https://keras.io/preprocessing/image/>. [Accessed 2019].
- [128] Y.-D. Zhang, Z. Dong, X. Chen, W. Jia, S. Du, K. Muhammad and S.-H. Wang, "Image based fruit category classification by 13-layer deep convolutional neural network and data augmentation," *Multimedia Tools and Applications*, vol. 78, pp. 3613-3632, 2019.
- [129] A. Rosebrock, "Fine-tuning with Keras and Deep Learning," pyimagesearch, 3 June 2019. [Online]. Available: <https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/>. [Accessed 2019].
- [130] "OpenCV: Motion Analysis," OpenCV, [Online]. Available: https://docs.opencv.org/master/de/de1/group__video__motion.html#ga2beb2dee7a073809ccec60f145b6b29c. [Accessed 2020].
- [131] I. Setitra and S. Larabi, "Background Subtraction Algorithms with Post-processing: A Review," *Proceedings - International Conference on Pattern Recognition*, pp. 436-2441, 2014.

- [132] "sklearn.cluster.MinibatchKMeans," scikit-learn developers, 2019. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MinibatchKMeans.html>. [Accessed 2020].
- [133] GeeksforGeeks, "Python | Find most frequent element in a list," [Online]. Available: <https://www.geeksforgeeks.org/python-find-most-frequent-element-in-a-list/>. [Accessed 2020].
- [134] "PyTorch," [Online]. Available: <https://pytorch.org/>.
- [135] "TensorFlow," [Online]. Available: <https://www.tensorflow.org/>.
- [136] G. Appliances, "GE Café™ Series ENERGY STAR® 27.8 Cu. Ft. French-Door Refrigerator with Keurig® K-Cup® Brewing System," 2020. [Online]. Available: <https://products.geappliances.com/appliance/gea-specs/CFE28USHSS>. [Accessed 2020].
- [137] "CIELAB color space," Wikipedia, 19 April 2020. [Online]. Available: https://en.wikipedia.org/wiki/CIELAB_color_space. [Accessed 2020].
- [138] K. Kim, S. Hong, S. Kwon, I. Lee, M. Lee and J. Kim, "REFRIGERATOR WITH CAMERA AND CONTROL METHOD FOR THE SAME, Patent #US 10 , 036 , 587 B2," United States Patent, 31 July 2018. [Online]. Available: <https://patentimages.storage.googleapis.com/46/d9/96/c18463ad950b61/US10036587.pdf>. [Accessed 2019].
- [139] J. G. Abdo, M. P. Ebrom, N. Giacomini, D. J. GILMORE, B. N. Radford, A. E. Showers and C. A. Stipe, "Interaction recognition and analysis system," Whirlpool Corp, 21 March 2019. [Online]. Available: <https://patents.google.com/patent/US20190087966A1/en?oq=US20190087966A1>. [Accessed 2020].
- [140] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, p. 211–252, 11 April 2015.
- [141] T. MA, M. Tanabian, L. Chunkwok and S. ADHIKARI, "Object Recognition for a Storage Structure, Patent #US 2016/0217417 A1," United States Patent Application Publication, 28 July 2016. [Online]. Available: <https://patentimages.storage.googleapis.com/08/82/c7/f7297fa65062f6/US20160217417A1.pdf>. [Accessed 2020].

APPENDIX A

Table 23. Selected papers on research into inventory management systems in the refrigerator.

Title	Task	Sensors
IoT Based Smart Kitchen Inventory Management System for Kitchen (Rezwan et al, 2018)	Track inventory, order groceries on a web/mobile app, and create a monthly grocery list	Scales and photoresistors within 9 “smart compartments”
Smart Fridges with Multimedia Capability for Better Nutrition and Health (Luo et al, 2008)	Track inventory, provide nutritional information, alert when a food item is about to go bad, create a shopping list	Barcode scanner
The Pervasive Fridge: A Smart Computer System Against Uneaten Food Loss (Rouillard, 2012)	Track inventory, provide recipes, alert the user if an item is about to go bad	Utilize an external food information database and the users phone camera to collect photos, do speech recognition, and scan barcodes
A RFID/NFC Fusion based Smart Refrigerator for Wellness Service (Son et al, 2014)	Track inventory and provide customized food suggestions based on nutritional needs	RFID and NFC
Smart Refrigerator Using the Internet of Things (Prapulla SB et al, 2015)	Track inventory and send a text/email to user if an item is running low	Pressure sensor, photoresistor, and barcode
Introducing Innovative Item Management Process Towards Providing Smart Fridges (Miniaoui et al, 2019)	Track inventory, browse and search refrigerator inventory using a web app, automatically order inventory items that are running low	RFID
An AI driven approach for Smart refrigerator to enhance family diet and sustainability (Kumar et al, 2019)	Cloud database with machine learning algorithm tracks inventory and learns user behaviors to suggest recipes, create grocery lists	Barcode, camera, and scale
Intelligent Refrigerator Using Artificial Intelligence (Shweta A.S, 2017)	Track inventory within the produce bins and tell user what vegetables they have,	360-degree camera uses histogram matching to identify bin contents for a

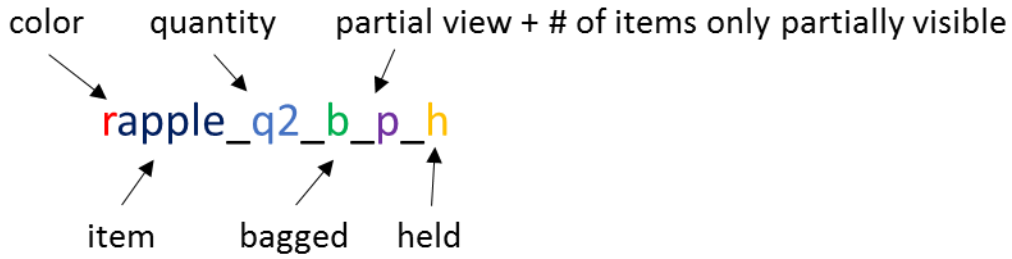
	and alert them when they will go bad	bin filled with a single type of produce
SMART HOME APPLIANCES: CHAT WITH YOUR FRIDGE (Gudovskiy, et al, 2019)	Allow users to “text” their refrigerator and ask questions like “are there any pears?” and “is any of my food about to spoil?”	CNN and natural language processing technique visual question answering

Table 24. Selected papers on research into produce classification.

Title	Task	Identification Technique
Recognition of Edible Vegetables and Fruits for Smart Home Appliances (Buzzelli et al 2018)	Classification Use VegFru dataset to train a model to identify fruit and veg that are very similar (like different types of apples)	Used a CNN image classifier, NasNet. First fine-tuned on super-classes, then sub-classes. Used data augmentation to improve accuracy
VegFru: A Domain-Specific Dataset for Fine-grained Visual Categorization (Hou et al 2017)	Classification Developed a food dataset specific for cooking. Specified the principles they used to building the dataset	Used a CNN image classifier – HybridNet
An Edge Computing Visual System for Vegetable Categorization (Liu et al, 2019)	Classification Trained then did work to reduce model size and deploy on mobile	Train a model on vegfru, used mobilenet
Classification of Vegetables using TensorFlow (Patil et al, 2018)	Veg classification	TensorFlow, CNN inception and transfer learning
DeepFruits: A Fruit Detection System Using Deep Neural Networks (Sa et al, 2016)	Classification and detection to detect fruits on the vine	Faster R-CNN
Fruit recognition from images using deep learning (Muresan et al, 2019)	New fruit database, trained using multiple color spaces	CNN using different color spaces
A Vision-Based Method Utilizing Deep Convolutional Neural Networks for Fruit Variety Classification in Uncertainty Conditions of	Use two phased classification with certainty factor taking in the two predictions	CNN – Yolo, run classifier over entire image, use yolo to detect, then pass those to antoehr classifier

Retail Sales (Katarzyna et al, 2019)		
---	--	--

APPENDIX B



qc – cut fruit

bc – bagged in plastic container

mfruit – mixed fruit (followed by list of the mixed fruit)

For rhand, rarm, the r_ specifies right. For lhand, larm, the l_ specifies left.

Figure 55. Annotation naming conventions.

Table 25. Annotation count by item for LabelImg annotations.

Item name	Annotation count
lhand	301
larm	215
rhand	250
rarm	208
lbin	53
rbin	114
rapple	237
orange	177
watermelon	20
pear	122
blueberries	107
carrot	277
tomato	77
broccoli	69
grapes	236
eggs	96
mfruit	143
cantaloupe	104

hand	196
Total annotations	3002
Total images	487

Table 26. Annotation count by item for Semi-Automatic Image Annotation tool, Anno-Mage.

Item name	Annotation count
hand	547
Total images	326

Table 27. Class breakdown for object classifier dataset.

	Class name	Image count
Training set	Apple	361
	Empty	455
	Orange	236
Test set	Apple	127
	Empty	145
	Orange	79
	Total images	1403

APPENDIX C

Sample configuration file for TensorFlow Object Detection API, highlighted portions must be updated for each training session.

```
model {
  ssd {
    num_classes: 22
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.9999999e-05
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.029999999
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.99970001
          center: true
          scale: true
          epsilon: 0.001
          train: true
        }
      }
    }
  }
  box_coder {
    faster_rcnn_box_coder {
      y_scale: 10.0
      x_scale: 10.0
      height_scale: 5.0
      width_scale: 5.0
    }
  }
  matcher {
    argmax_matcher {
      matched_threshold: 0.5
      unmatched_threshold: 0.5
      ignore_thresholds: false
      negatives_lower_than_unmatched: true
      force_match_for_each_row: true
    }
  }
}
```

```

}
similarity_calculator {
  iou_similarity {
  }
}
}
box_predictor {
  convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 3.9999999e-05
        }
      }
    }
    initializer {
      truncated_normal_initializer {
        mean: 0.0
        stddev: 0.029999999
      }
    }
    activation: RELU_6
    batch_norm {
      decay: 0.99970001
      center: true
      scale: true
      epsilon: 0.001
      train: true
    }
  }
  }
  min_depth: 0
  max_depth: 0
  num_layers_before_predictor: 0
  use_dropout: false
  dropout_keep_probability: 0.80000001
  kernel_size: 1
  box_code_size: 4
  apply_sigmoid_to_scores: false
}
}
anchor_generator {
  ssd_anchor_generator {
    num_layers: 6
    min_scale: 0.2
    max_scale: 0.94999999
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    aspect_ratios: 3.0
    aspect_ratios: 0.33329999
  }
}
}
post_processing {
  batch_non_max_suppression {
    score_threshold: 9.9999999e-09
    iou_threshold: 0.60000002
    max_detections_per_class: 100
    max_total_detections: 100
  }
}
score_converter: SIGMOID

```

```

}
normalize_loss_by_num_matches: true
loss {
  localization_loss {
    weighted_smooth_l1 {
    }
  }
  classification_loss {
    weighted_sigmoid {
    }
  }
  hard_example_miner {
    num_hard_examples: 3000
    iou_threshold: 0.99000001
    loss_type: CLASSIFICATION
    max_negatives_per_positive: 3
    min_negatives_per_image: 3
  }
  classification_weight: 1.0
  localization_weight: 1.0
}
}
}
train_config {
  batch_size: 12
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
}
optimizer {
  rms_prop_optimizer {
    learning_rate {
      exponential_decay_learning_rate {
        initial_learning_rate: 0.0040000002
        decay_steps: 800720
        decay_factor: 0.94999999
      }
    }
    momentum_optimizer_value: 0.89999998
    decay: 0.89999998
    epsilon: 1.0
  }
}
}
fine_tune_checkpoint: "/content/models/research/pretrained_model/model.ckpt"
num_steps: 200000
fine_tune_checkpoint_type: "detection"
}
train_input_reader {
  label_map_path: "/content/drive/My Drive/thesis/Object-Detection/data/object-
detection.pbtxt"
  tf_record_input_reader {
    input_path: "/content/drive/My Drive/thesis/Object-
Detection/data/train.record"
  }
}

```

```
}
eval_config {
  num_examples: 8000
  max_evals: 10
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "/content/drive/My Drive/thesis/Object-Detection/data/object-
detection.pbtxt"
  shuffle: false
  num_readers: 1
  tf_record_input_reader {
    input_path: "/content/drive/My Drive/thesis/Object-Detection/data/test.record"
  }
}
```

CURRICULUM VITA

NAME: Sarah Virginia Morris

ADDRESS: 1314 Everett Avenue
Louisville, KY 40204

DOB: Louisville, Kentucky – September 7, 1985

EDUCATION & TRAINING:

B.S., Electrical Engineering
University of Louisville
2012-2017

B.S., Mechanical Engineering
University of Louisville
2012-2017