Food Quality Detection with Convolutional Neural Network

Abed Atassi[#1], Aurelia Haas[#2], Lukas Durand[#3], Michel Cantacuzene[#4]

[#]*Department of Electrical and Computer Software Engineering, McGill University*

845 Rue Sherbrooke O, Montreal, QC H3A 0G4, Canada

[1]abed.atassi@mail.mcgill.ca,

[2]aurelia.haas@mail.mcgill.ca,

[3]lukas.durand@mail.mcgill.ca,

[4]michel.cantacuzene@mail.mcgill.ca

Abstract

Food health and safety have always been a major concern for producers and distributors alike. From farm to table, nearly all store-bought fresh products are evaluated in multiple stages before reaching the end consumer. The industry has continuously adapted new technologies in order to increase efficiency and throughput while also emphasizing quality control. With the development of computer vision techniques and convolutional neural networks, many papers have investigated computerized identification and classification of produce freshness, but these mainly incorporated traditional RGB cameras with mixed results. Knowing that the agricultural industry has been embracing multispectral cameras with drones to survey crop health, we will identify and propose a ***method for the identification and classification of agricultural produce using images from multispectral sources.*** *This project is separated into two main sections. Firstly, the choice of the convolutional neural network model and an investigation into its scalability and ability to increase the range of produce handled, as the industrial application of computer vision for agricultural produce requires retraining and extension of models. Secondly, we investigated the grading of quality based on images from a multispectral camera source. The first semester goals were to secure funding for a multispectral research camera, and give the members of this team a thorough understanding of the different convolutional neural networks for image recognition, while also determining problems we would encounter further on; The second semester goals were to create a multispectral camera able to take pictures of produces with different wavelength lenses and, by using the CNN models previously studied, analyze these pictures to obtain a certain accuracy of image recognition. A significant amount of testing for both CNN and Multispectral phases was conducted in order to find the most efficient and precise algorithm and capture process to meet our project's needs.*

Introduction

Testing the freshness and health of agricultural produce is a cumbersome task, some damage and imperfections may be spotted quickly with the naked eye while other problems remain impossible to diagnose without specific tools or time-costly inspection. Indeed the supply chain logistics of fruits and vegetables is such that it may leave the field in perfect condition, but arrive at a distribution center damaged, rotten or mouldy. Thus, it is important for food companies to verify the freshness and health of their products before they make their way to the consumers.

This presents an engineering challenge: how to verify the freshness of agricultural produce in

an industrial context. To solve this challenge, our team has decided to leverage the power of computer vision coupled with multispectral imaging.

The goal of this project is to build a convolutional neural network (CNN) that first uses a standard RGB camera system to identify in-frame produce, and then performs an analysis of the frame using multispectral-imaging data to detect features such as freshness, colour, shape, and damage.

The system we are describing would constitute a radical step in regards to the quality-assurance standards of food distributors. The project is divided into two major parts, the first one is focused on determining what neural network model to choose, how to train it, and build a dataset with the minimum amount of data required to recognize a specific ingredient above a certain accuracy range. The second phase mainly consists of building and using a multispectral camera in order to get more details beyond the scope of visible light to identify potential internal damage in an ingredient. This implementation could lead to a more reliable, consistent, and automatic food inspection process that can be cost-saving and less time-consuming for companies.

Our project has been sponsored by *Les Recettes Cook It*, a Montreal-based meal kit company that will help us get more insight into the food industry and how machine vision can enhance food quality assurance. As engineers, it is essential for us to get some hands-on experience in the field. We were excited to work alongside a market-leading company to help in their quest towards further increasing their processing efficiency.

Background

This project involves a wide variety of tasks such as programming, hardware design, problem-solving, tuning, computer vision, and multispectral analysis integration.
As students in electrical, computer, and software engineering, we had a strong background in coding and hardware which prove useful in regards to our project. However, we had to learn and understand several new theories unrelated to our academic background in order to successfully complete our project goals.

1. Programming Language: Python

Python [1] is a programming language that is fast and integrates systems more effectively. This language is extremely instinctive and is one of the most common languages used in order to build and train neural networks, especially CNNs [2]. The availability of extensive platforms for machine, computer vision, and deep learning —such as PyTorch and TensorFlow—, makes Python the best programming language for our project. In addition, it is suitable for collaborative implementation which was needed in our case.

2. Hardware Background

A GPU is a graphics processing unit that is designed to quickly control and alter memory in order to accelerate the creation of images. Current GPUs are remarkably useful in image processing and prove to be very efficient in terms of image manipulation [3].

A Raspberry Pi V3-B chip is a small-sized single-board computer that is designed to be modular. This small computer allows people to learn to code electronics for physical projects as well as create their own automation projects. It has an ARMv6 700 MHz single-core processor, a VideoCore IV GPU, and 512MB of RAM. A micro SD card stores its operating system and data logger (Figure 1) [4].

Figure 1: Raspberry Pi 3B [5]

3. Transfer Learning

Transfer learning is a machine learning method in which a model previously developed for a specific task is reused as the starting model for another task [6]. This system allows the use of pre-trained models and then speeds up the performance and training of the deep learning model.
Transfer learning is particularly useful for CNNs since some pre-built models will save time and optimize the performance of future ones [7].

4. Convolutional Neural Network

A CNN is a deep learning algorithm that analyzes visual imagery by assigning different levels of importance to the diverse objects from the input picture, so it can differentiate them [8] (Figure 2). CNNs consist of different layers among which hidden ones can be located. These hidden layers which are not visible to the external systems and are private to the neural network, generally allow for the function of a neural network to be broken down into specific transformations of the data such as identifying faces in images, probabilities and more [9]. CNNs convolve with multiplication or dot product. The goal of CNN is to perceive what the image input represents with the best possible accuracy.

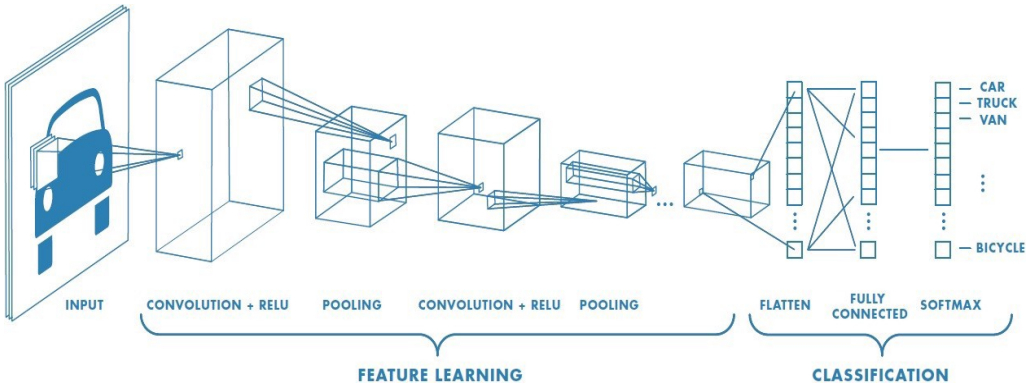


Figure 2: CNN system [8]

5. Multispectral Imaging

Multispectral imaging refers to a method in which images are captured by sampling a single discrete wavelength across the electromagnetic spectrum per image [10]. A method of sampling a discrete wavelength uses band-pass filters and in application wavelengths in the IR and UV ranges are sampled (Figure 3).

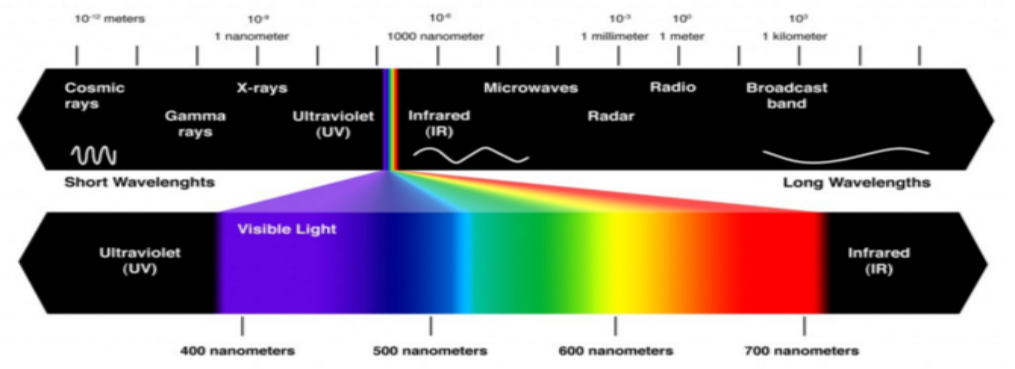


Figure 3: Multispectral Imaging Spectrum [11]

6. 3D Printing

3D printing refers to the practice of creating a desired object in a 3-dimensional reference. Objects are first modelled using a computer and Computer-Aided Design software. Many methods exist to 3D print objects, the most known being additive manufacturing such as Fused Deposition Modeling (FDM) where a print head will deposit a stream of material creating layers (Figure 4). Another approach is SLA where a vat of photocuring resin is used as the catalyst and layer by layer the final object is "pulled" out of the resin with the use of a light source that cures the resin bit by bit. Materials such as plastics and resins are often used due to their cost and ease of use with specific printing methods but printers can also print fibreglass, carbon fibre, metal and even ceramic.

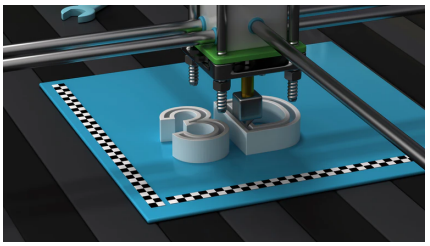


Figure 4:3D Printer [12]

Requirements

1. Problem statement

In general, the food industry still relies heavily on humans for inspection procedures. However, these methods are tedious and can lead to considerably high error margins. Quality assurance tasks are repetitive in nature, often resulting in fatigue, which can undermine the quality review process. For some products, a thorough inspection is critical in order to abide by health regulations. Inspection workers can also introduce more contaminants to the process, which often go unaccounted for [13]. One of the most distinct drawbacks of current procedures is the variability in human perception of quality. Eyes can identify surface damage or discoloration of an ingredient, but often fail to detect concerning factors such as fungal growth present within the item. . As the food industry expands to meet the needs of society, increasing the efficiency and throughput of quality assurance is essential. This requires a fast and dependable process to keep up with consumer demand [14].

Developments in computer vision, processing power, camera system affordability, and data storage options have resulted in an interesting opportunity to modernize food inspection.

Technological methods can be used to extract information from images, analyze them, and output reliable diagnostic data about the content found in an image. This is where our proposed solution comes in place to undertake the aforementioned issues that currently affect most of the food industry.

2. Requirements

Our proposed system will have five main requirements:

a. Camera system

Four CMOS cameras with resolutions of 8 megapixels. All cameras should have the same resolution but they will differ in their functions. The first is an RGB camera with an infrared filter that will capture regular images. These will be used for product identification by CNN. The remaining three will be equipped with custom filters in order to capture specific wavelengths of light (IR – 795 nm, NIR – 742 nm and UV – 550 nm). This camera system will be used to capture images of produce in the visible and non-visible wavelengths, allowing us to acquire more information for dataset creation. These will in turn be employed to train our neural network models and classify the type and condition of each ingredient correctly.

b. Computing system

This system will consist of a computer equipped with a discrete GPU that can handle ingesting all the images from our dataset and train our CNN. A GPU can greatly accelerate the process with parallelization. This system can either be local or provided on the cloud by a third party.

c. Image Dataset

The most important step of training a CNN is having sufficient labelled data of ingredients. This requires access to a large number of ingredients in various shapes, sizes, and conditions to have highly accurate results for our network. The taken images also need to be in good lighting conditions to ensure that every ingredient is recognizable by the network. In addition, all along our project, we decided to use a neutral background to facilitate the CNN analysis of the important part of the picture: the produce.

d. Machine learning libraries and programming language

These are software packages written in C or C++ that allow us to perform machine vision tasks, like extracting image data, preprocessing the captured images, and building neural network models like CNNs. The two most known libraries are Tensorflow and PyTorch which are supported in the Python programming language.

e. Dataset Creation System

In order to speed up the dataset creation phase, we decided to design a semi-autonomous turntable rig that rotates the produce item by a set amount between each capture. This system made it easier and faster to create quality datasets without straining the person in charge of doing so.

3. Constraints

The main constraint for this project is the dataset that we require to be able to train our CNN and get accurate results. There are very few publicly available datasets and practically none from multispectral sources that have sufficient and clear images of ingredients like fruits and vegetables. This led us to create our own datasets with RGB pictures and multispectral images, which is time-consuming and requires us to have access to a large number of ingredients in various shapes, sizes, and conditions. We are also constrained by the hardware that is available to us like GPU, processing power, and camera resolution for the first section. In the second phase, another constraint appeared, which was linked to the multispectral camera. Indeed, building the camera with the correct elements the first time is not achievable, we had to spend a huge amount of time fine-tuning the

camera and its setup. Moreover, taking pictures with a multispectral camera takes longer than taking RGB pictures, therefore this issue added to our constraints.

Design and Results

Phase 1:

While researching CNNs we discovered that models can be trained in two different ways, from scratch or from a pre-trained model by freezing a few layers and training the remaining unfrozen layers with the desired dataset (Transfer Learning). As we were not fully familiar with machine learning libraries in Python before implementing our models, we decided to use Tensorflow to build and train our model from scratch. PyTorch was then used to perform transfer learning. For the rest of the project, we opted to use PyTorch due to its ease of use.

1. Datasets

In order to train and test different models, which will be detailed below, we first needed to build large datasets of images to represent different produce. We opted for red apples and tomatoes as they have similar appearance characteristics. Figures 5 and 6 depict examples of pictures from our database. The idea was to be able to recognize and differentiate them through our models. We built a dataset of more than 2,500 images with different levels of difficulties. This means that we took photos with different angles, different backgrounds, and lights so our tests could be more accurate. Pictures of grapefruits and oranges were also added in order to accomplish more tests.

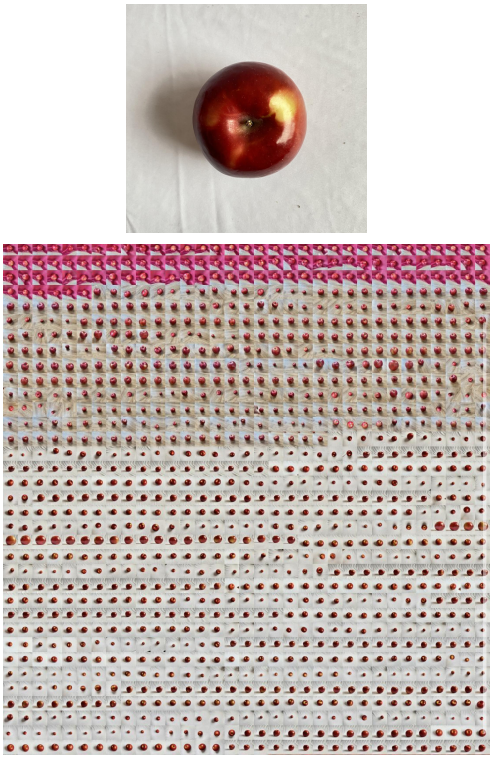


Figure 5: Red Apple Pictures from Dataset

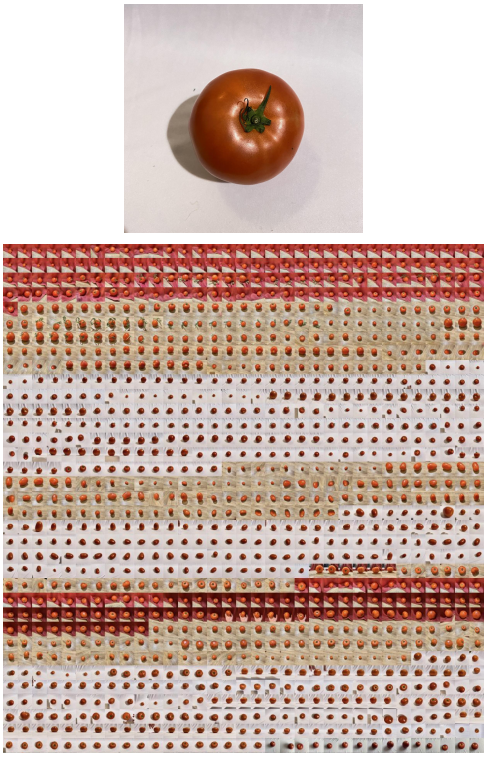


Figure 6: Tomato Pictures from Dataset

2. Training CNNs from scratch

We Started with 3 pre-existing models, notably VGG16 (K. Simonyan and A. Zisserman), Xception (Francois Chollet), and MobileNetV2 (Mark Sandler, Andrew Howard, Menglong Zhu,

Andrey Zhmoginov, Liang-Chieh Chen). Our aim was to use these models — which had proven accuracy on the ImageNet competition — to train on our datasets.
After an initial round of training, we removed the Xception model as its training times were too long and memory intensive for our application.
We then proceeded with training the MobileNetV2 and VGG16 models on increasingly larger data sets and reporting their loss. In order to do this, we created datasets of apples and tomatoes of varying sizes as mentioned above. These datasets contained 5 repetitions (5 datasets of the same size but different images to test for statistical accuracy on the loss). We then tested 3 repetitions of each size of the database, on an increasingly large number of epochs: 25, 50, 75, and 100 epochs (Appendices - Figures A.1 to A.16 and B.1 to B.20).
The aim of the tests was to identify a correlation between dataset size and epochs to identify a training regiment that could potentially be replicated that had the least amount of images and epochs.
We also have chosen 0.1 as the target loss based on the general naive assumption that the accuracy is inversely proportional to the loss which would give us ~90% accuracy. We will be testing this further during the second part of the project.

3. Transfer learning

As mentioned in the constraints and dataset sections, we had to create our own dataset as none satisfied the exact requirements we needed for the project. Due to the difficulties of building sufficiently large datasets, we decided to explore transfer learning and use pre-trained CNNs on our datasets for initialization or fixed-feature extracting.

The choice of which transfer learning variant to use was made based on the predefined dataset used to train the model and our own dataset. We decided to implement promising models that were used in the ImageNet competition as they were trained on a dataset which contains 1.2 million images with 1000 categories including some vegetables and fruits as they are also proven to give high accuracy [15]. The chosen models were VGG16 which is a 16 layer-deep CNN (Figure 7), ResNet50 which is 50 layers deep and MobileNetV2 (53 layers deep) which was designed for mobile vision applications while still providing high accuracy. Models with a considerable amount of layers were used because it was demonstrated that higher depth networks help increase classification accuracy. However, as models get deeper, they also get harder to optimize. ResNet50 tries to improve optimization by using network layers to fit a residual mapping scheme instead of directly trying to fit a desired underlying mapping [16] (Figure 8). While MobileNetV2 is based on an inverted residual structure where the residual connections are between the bottleneck layers [17] (Figure 9). All models were run on GPUs to accelerate the training process.

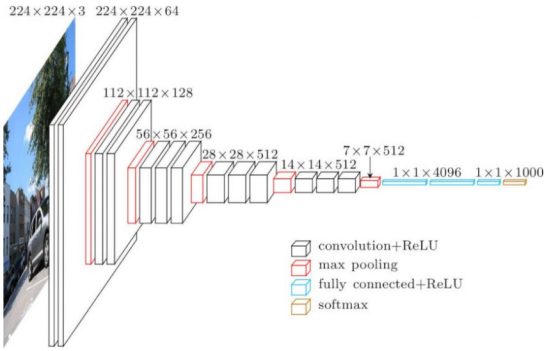


Figure 7: VGG16 architecture [16]

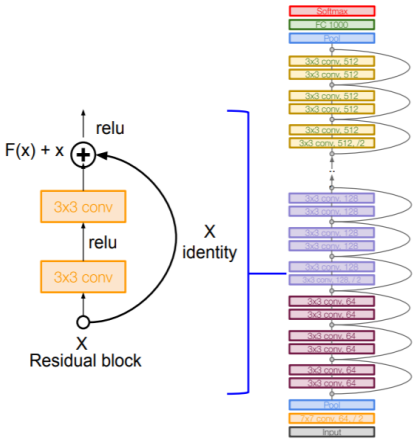


Figure 8: ResNet50 architecture [18]

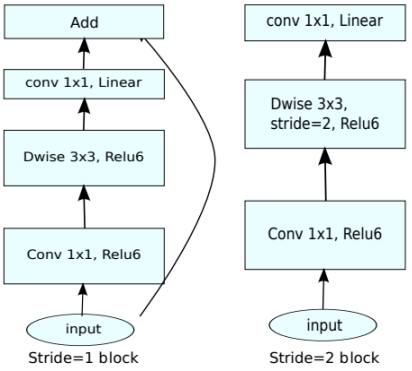


Figure 9: ResNet50 architecture [17]

As our dataset is considerably small compared to ImageNet, we decided to freeze all layers except the last classification layer and train it, as can be seen in Figure 10.

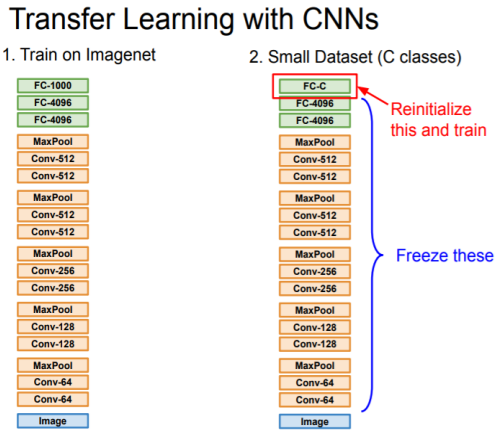


Figure 10: Transfer learning [19]

Our dataset was broken up into a training set (50% of the dataset), validation (25%), and a testing set (25%) for each class. As mentioned previously, because of the small size of the current dataset we implemented image augmentation where images were randomly cropped, rotated and flipped. Augmentation was only performed on the training set, the images were then normalized, and resized to 224x224 pixels to match the size of the ImageNet images used on the pre-trained models. Augmentation, as shown in figure 11, was only performed on the training dataset as we need to increase the number of images for training the classifier.

```
image_transforms = {
    'train':
    transforms.Compose([
        transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),
        transforms.RandomRotation(degrees=15),
        transforms.ColorJitter(),
        transforms.RandomHorizontalFlip(),
        transforms.CenterCrop(size=224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
```

Figure 11: Training images augmentation

Once augmentation has been performed, we place each of our training, validation and testing sets into data loaders so we could iterate through them with the specified batch size. Then we iterate through each layer of our model except the classification layer and set a boolean variable to false to indicate that gradient computation for parameters is not required thus freezing these layers. We then check if a GPU is available in the system where we are training and map the training process to take advantage of the parallel execution provided by the GPU. Our code does not necessarily require a GPU and can be run solely on the CPU as well. The final step before starting training is to indicate which optimizer the model should use in order to find the values of the weights to minimize our final loss value faster.

Training was done using 30 epochs, wherein each epoch we record the loss value. After each training iteration, we perform one validation iteration and check the loss value. If it is the lowest recorded yet, we save this iteration of the model so that it can be used for testing after training is done. We also implemented early stopping, where we specify a hyperparameter for the maximum number of epochs we want to have without a reduction in loss, and in case it happens we stop training early. This allows us to shorten training time by seeing when the loss stagnates and check how many epochs are needed to reach the minimum loss value the model can achieve with our dataset and avoid overfitting. During training, we compute the training and validation losses, accuracies at each epoch and print them so we could see how the model is performing as it trains. They are then plotted for comparison and to make it easier to check for overfitting and underfitting. Finally, after saving the model, we start the testing phase, where we compute the accuracies for each class and compare them to the number of images provided to determine the minimal amount of images required to achieve the accuracies we need.

4. Results

As mentioned above, we had to train different models: those from scratch (VGG16, MobilNetV2, and Xception) and those from transfer learning methods (VGG16, ResNet50 and MobileNetV2).

We then acquired a vast set of outputs from these models using small datasets with different levels of difficulties, developed by merging the ones found online and the ones we built. The final outputs allowed us to build tables with the loss of the models by using an increasing number of epochs on each model, as well as the accuracy of prediction. The graphs generated from these tables can be found in the Appendices (Appendices - Figures A.1 to A.16 and B.1 to B.20).

All the results obtained gave us a consequential insight on which model we will be using as well as the required size of the dataset we will collect in order to operate the best model we can.

Using our findings on the effect of data set size versus epoch quantity for training models from scratch, we are able to identify a testing range for us to test the relationship between loss and real-world accuracy.

As mentioned, for the two different types of models we have:

- **Models from scratch:**
  - VGG16

    Training the VGG16 model showed problems such as blocking on loss value indicating an overtraining, or in our case, blocking on local minima. This happened as our loss always converged to the same amount even with the change of dataset and the change in the number of epochs (Appendices - Figures A.1 to A.16). Furthermore, these training results proved that VGG16 was far too volatile and unpredictable to train from scratch and would not be applicable in our context. In addition, there is no proper observable correlation between epochs and dataset size due to the high volatility, and there was an observable and replicable way of getting a loss lower than 0.1 from our extensive testing. Finally, VGG16 took a considerable amount of time to train per epoch compared to the other model under investigation (MobileNetV2). Although this may not be a problem if the model is meant to be trained only once, it could cause problems in industrial settings where retraining of the model may be necessary due to external factors. For these reasons, we concluded that VGG16 would not function for our use case.
  - MobileNetV2

    Training the MobileNetV2 model yielded results that were closer to our expectations and allowed us to observe a trend that indicated a relationship between the loss, size of the dataset, and the number of epochs. Furthermore, our tests show that these results are reproducible and we can, in theory, extract a training program by selecting a minimum number of images for a dataset and the minimum number of epochs for training and result in an expected accuracy (Appendices - Figures B.1 to B.20). As well as yielding these results, the MobileNetV2 model was much quicker to train than the VGG16 model, making it more suitable for industrial applications.

    Based on these observations, we conjecture that a suitable training regimen for agricultural produce would be a minimum of 500 images per class and 60 epochs of training to obtain a loss value of 0.1 or lower (independently of what produce we are training).

- **Transfer Learning Models:**

  Transfer learning was evaluated using two separate tests for each model. The first test was to check the effectiveness of the model regarding fruit prediction accuracy. The goal of the second test was to see how the model's performance would change when more classes are present, thus image data was added to our dataset.

  - **VGG16**

    The losses and accuracies' resulting graphs for the VGG16 model can be observed in figures 12 and 13 for test 1 and in figures 14 and 15 for test 2. Tables 1 and 3 show the number of images used for our three steps: training, validation, and testing.

    Test 1:

    - Best epoch: epoch 16
    - Total epochs: 22
    - Early stopping: Yes
    - Loss: 0.003
    - Accuracy: 100.0%
    - Training time: 432 seconds (20 seconds per epoch)

Class	Training images number	Validation images number	Testing images number
Apples	534	354	353
Tomatoes	460	223	225

Table 1: Transfer learning - VGG16 Dataset split

Figure 12: VGG16 training and validation losses

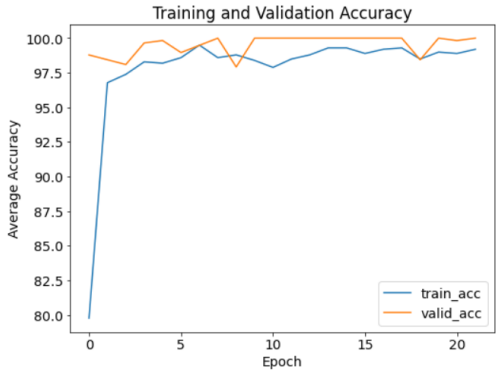


Figure 13: VGG16 training and validation accuracies

Class	Testing accuracy	Loss
Apples	100.0 %	0.000385
Tomatoes	100.0%	0.005338

Table 2: VGG16 prediction (testing) results

Test 2:

- Best epoch: epoch 6
- Total epochs: 13
- Early stopping: Yes
- Loss: 0.01
- Training accuracy: 99.4%
- Training time: 378.27 seconds (32 seconds per epoch)

Class	Training images number	Validation images number	Testing images number
Apples	534	354	353
Tomatoes	460	223	225
Orange	513	84	101
Grapefruit	420	54	63

Table 3: Transfer learning - VGG16 Dataset split

Figure 14: VGG16 training and validation losses

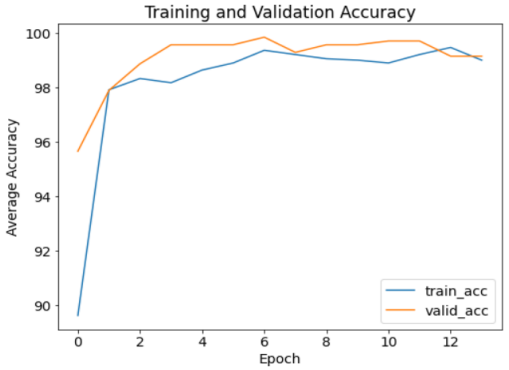


Figure 15: VGG16 training and validation accuracies

Class	Testing accuracy	Loss
Apples	100.0 %	0.000854
Tomatoes	99.1%	0.018453
Orange	100.0%	0.007094
Grapefruit	80.95%	0.665899

Table 4: VGG16 prediction (testing) results

- **ResNet50:**

The losses and accuracies' resulting graphs for the ResNet50 model can be observed in figures 16 and 17 for test 1 and in figures 18 and 19 for test 2. Tables 5 and 7 show the number of images used for our three steps: training, validation, and testing.

Test 1:

- Best epoch: epoch 26
- Total epochs: 30
- Early stopping: No
- Loss: 0.002
- Accuracy: 99.83%
- Training time: 494 seconds (17 seconds per epoch)
- Training dataset: own
- Testing dataset: own

Class	Training images number	Validation images number	Testing images number
Apples	534	354	353
Tomatoes	460	223	225

Table 5: Transfer learning - ResNet50 dataset split

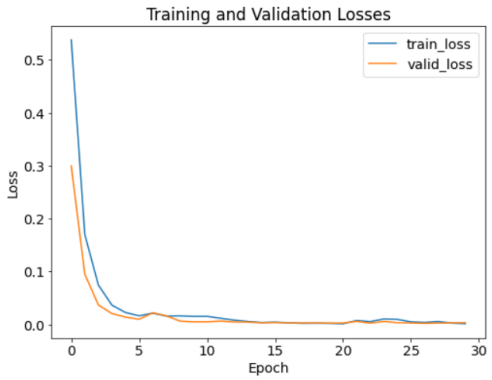


Figure 16: ResNet50 training and validation losses

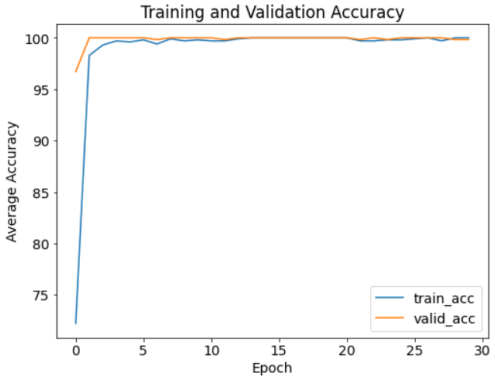


Figure 17: ResNet50 training and validation accuracies

Class	Testing accuracy	Loss
Apples	100.0 %	0.000540
Tomatoes	100.0%	0.001251

Table 6: ResNet50 prediction (testing) results

Test 2:

- Best epoch: epoch 8
- Total epochs: 11
- Early stopping: Yes
- Loss: 0.02
- Training accuracy: 98.46%
- Training time: 669.27 seconds (28 seconds per epoch)

Class	Training images number	Validation images number	Testing images number
Apples	534	354	353
Tomatoes	460	223	225
Orange	513	84	101
Grapefruit	420	54	63

Table 7: Transfer learning - ResNet50 Dataset split

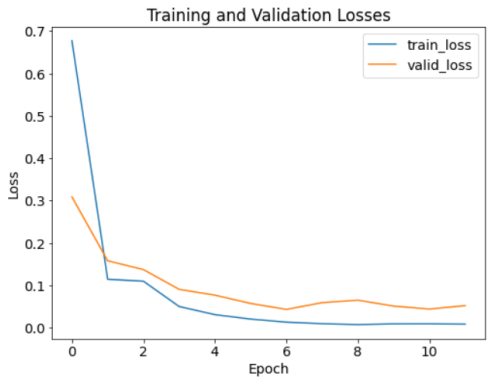


Figure 18: ResNet50 training and validation losses

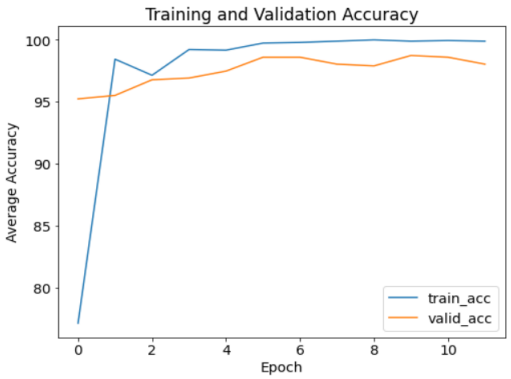


Figure 19: ResNet50 training and validation accuracies

Class	Testing accuracy	Loss
Apples	99.71 %	0.004535
Tomatoes	99.56%	0.015037
Orange	97.03%	0.106293
Grapefruit	65.08%	0.943615

Table 8: ResNet50 prediction (testing) results

- **MobileNetV2**

  The losses and accuracies' resulting graphs for the MobileNetV2 model can be observed in figures 20 and 21 for test 1 and in figures 22 and 23 for test 2. Tables 9 and 11 show the number of images used for our three steps: training, validation, and testing.

  Test 1:

  - Best epoch: epoch 10
  - Total epochs: 15
  - Early stopping: Yes
  - Loss: 0.001
  - Accuracy: 100.0%
  - Training time: 302 seconds (19 seconds per epoch)

Class	Training images number	Validation images number	Testing images number
Apples	534	354	353
Tomatoes	460	223	225

Table 9: Transfer learning - MobileNetV2 Dataset split

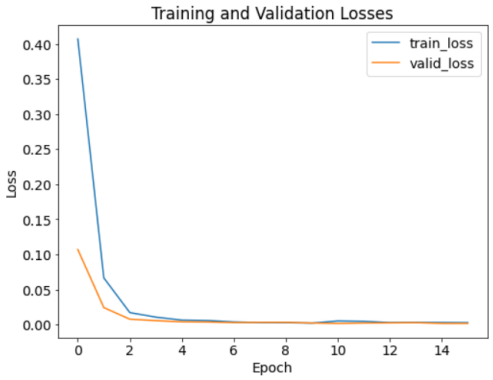


Figure 20: MobileNetV2 training and validation losses

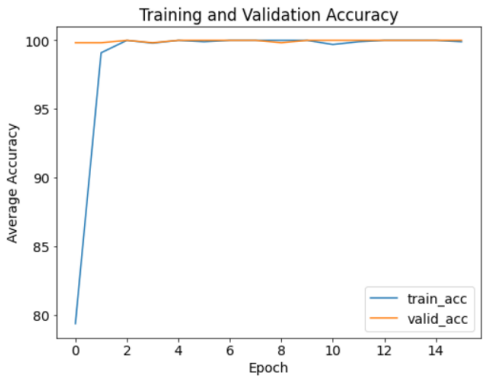


Figure 21: MobileNetV2 training and validation accuracies

Class	Testing accuracy	Loss
Apples	100.0 %	0.000816
Tomatoes	100.0%	0.001069

Table 10: MobileNetV2 prediction (testing) results

Test 2:

- Best epoch: epoch 4
- Total epochs: 9
- Early stopping: Yes
- Loss: 0.04
- Training accuracy: 99.33%
- Training time: 202 seconds (21 seconds per epoch)

Class	Training images number	Validation images number	Testing images number
Apples	534	354	353
Tomatoes	460	223	225
Orange	513	84	101
Grapefruit	420	54	63

Table 11: Transfer learning - MobileNetV2 Dataset split

Figure 22: MobileNetV2 training and validation losses

Figure 23 MobileNetV2 training and validation accuracies

Class	Testing accuracy	Loss
Apples	100.0 %	0.000537
Tomatoes	100.0 %	0.010234
Orange	100.0%	0.029857
Grapefruit	90.47%	0.271179

Table 12: MobileNetV2 prediction (testing) results

Test 1 results:

Our testing illustrated that using around 400-500 images was enough to get 100% accuracy in recognizing apples and tomatoes for all models (Tables 2, 6 and 10). ResNet50, VGG16 and MobileNetV2 had no overfitting (getting trained on training image noise) as loss curves for training and validation are similarly changing. Setting only 30 training epochs has also been enough to get results this high as early stopping was triggered as loss started to stagnate earlier. VGG16 had a lower loss for apples than ResNet50, while the latter had a lower loss for tomatoes. Training time has also been very similar. However, MobileNetV2 had the lowest overall loss (0.001) and required the least amount of epochs (15) and training time to achieve the same accuracies. Thus, we can conclude that

all networks performed well with MobileNetV2 having an edge in results with lower loss and faster training times.

Test 2 results:

While still getting high accuracy results for fruits like apples, tomatoes, and oranges, grapefruits had a prediction accuracy of 80.95% as seen in table 4 for VGG16, only 65.08% as seen in table 8 for ResNet50 and 90.47% for MobileNetV2 as seen in table 12. This can be explained through figures 14, 18 and 22 where we can see that while the loss for training and validation are decreasing till the 10th epoch, there is a gap between the two curves. This gap may indicate that there are too few dataset images for training as compared to the validation dataset. Increasing the training images for grapefruits and oranges to make them match the number of images for tomatoes and apples could improve accuracy and avoid the model from mistaking grapefruits for oranges. MobileNetV2 performed the best out of all the models when it comes to grapefruit recognition as well as having the lowest loss values in each category. However, it did have some overfitting while training as can be seen in figure 22.

We can also notice that training times are relatively fast compared to training from scratch (15 to 20 minutes faster than training from scratch as all layers are frozen and only the classification layer is trained) as we are only training the last layer. It is especially the case of MobileNetV2 where it only required 9 epochs while having the fastest training time to achieve the best results out of all three models. This would be beneficial when adding new ingredients to the dataset and would allow a model to adapt faster while in use in the food industry. Another benefit seen through transfer learning is that through freezing layers, we went from having to train 2,552,836 parameters from scratch to only having to train 328,964 parameters for MobileNetV2.

Thus concluding from our results, we opted for MobileNetV2 to use for the second phase of our project as it was the most accurate to train, as well as choosing transfer learning as our training technique which not only greatly shortens training times, but also achieved high accuracy scores for fruit recognition.

Phase 2:

The second phase of our project was centred on multispectral imaging analysis and the CNN model based on said multispectral images. Given that multispectral imaging could capture image data within discrete spectral ranges, this technique has now been acknowledged as well suited for the safety and quality evaluation of food and produces [20]. Building an effective multispectral imaging camera involves a lot of theoretical and practical aspects. Factors that were considered for using multispectral imaging include spectral image acquisition methods, components for building spectral imaging system, methods for calibrating spectral imaging systems, and techniques for analyzing spectral images

This phase was divided into 4 main parts that were dependent on one another.

1. Building and Calibrating the Multispectral Camera

We decided to build our own MS camera since commercially available hardware was too expensive and would have exceeded the sponsorship resources given for our project. Building our multispectral camera was a long and tedious aspect of our project, but the second phase of this one depended on it. We first focused on the acquisition of materials to build our own multispectral camera. This camera, crucial to our project, also needed to be made by us since the current offering is tailored towards high altitude agriculture analysis generally meant to be placed on drones (far-field focal point). Due to this, the camera and lenses are themselves optimized for such conditions. Furthermore, the multispectral bands on these cameras are preset and cannot change, removing flexibility in our testing. As such we opted to use a Raspberry-Pi with NoIR cameras and high-grade filters that will allow our setup to be more flexible to use as we will only need to change the filters (and optimize the lighting source for those filters) for it to function properly.

We acquired all the necessary material and tested the use of all of the components independently first. Using Fusion 360, we modelled a first case that allowed us to use the multispectral camera with ease.

Our first case was made using the Ender 3 FDM 3D printer with PLA filament (biodegradable) which is generally used for preliminary print tests. Another common plastic that we could have used for prototyping is ABS. However, we opted for PLA as it is derived from renewable resources and it will biodegrade within 50 days in industrial composters and 48 months in water whereas ABS is not biodegradable.

The first camera we built was as follows (Figure 24).

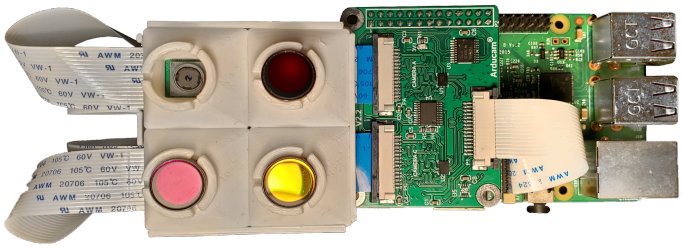


Figure 24: Previous PLA printed casement

After discussion and research, we also decided to build a rotating platform linked to the camera and launchable through a usable coding interface. We also decided to use another case for the camera as the one mentioned above had large tolerances leading to the filters being misaligned and moving from the inside. Therefore, with the purchase of an SLA 3D printer, we were able to design and print a casing to align the cameras and filters precisely. This build allowed us to take high-resolution pictures with our cameras without noise or blur as the cameras and filter are very precisely aligned thanks to the multiple design iterations (Figure 25).

Figure 25:Final multispectral camera

Once the MS camera system hardware was calibrated and finalized, we integrated the downsampling of the images.

2. Multispectral camera software and dataset creation

With our finalized MS camera design, we then began writing the software to interface with the camera as well as fine-tuning all aspects such as lighting intensity, lens focus point, background, and reflection-reduction in order to ensure all of the ensuing datasets have the same exact conditions throughout. We also opted to design, integrate and tune an automated turntable in order to further improve our capture mechanism.

We decided to interface with the camera system using a wireless connection. We wrote a web server in Go which allowed us to take pictures with each of the cameras as well as create a batch of pictures through a URL. All the photos taken were stored on the Raspberry-Pi and we were able to automate the taking and downloading of the pictures using a Bash script which would use the C-URL program to access the web URL triggering the capture of the images and once completed we would copy the images to our local computer using SCP. Due to the web-based interface, we could in theory trigger dataset creation from anywhere with an internet connection and download the images to any file server allowing for added flexibility when implementing this set up in an industrial setting.

To fine-tune the camera for our use case, we first tested it in various lighting conditions and realized that the best results could only be obtained in natural lighting from a window. A few sun-lit sample images taken of the same avocado are provided as follows (Figure 26-29) which show that our camera is capable of producing accurate spectral images.

Figure 26: RGB image in natural lighting conditions

Figure 27: 550nm UV filter in natural lighting conditions

Figure 28: 742nm NIR filter in natural lighting conditions

Figure 29: 795 IR filter in natural lighting conditions

The issue we now faced was figuring out how to artificially reproduce natural light in a closed environment. Standard LED lighting does not provide a wide enough light spectrum to target each bandpass filter (from 550nm UV to 795nm IR). Relying on natural light creates issues as it varies significantly throughout the day, affecting the dataset exposure levels. In order to rectify this, we created a light rig using 7 Philips Hue colour light bulbs which we calibrated to provide the best conditions for the narrow-band filters. On the software side, significant improvements to the image capture system were made simultaneously in order to speed up dataset creation.

With all the lighting tests in mind, we then proceeded in building a fully enclosed lightbox that would keep exposure levels constant throughout. Each of the four cameras was then manually configured for the lighting conditions. Through countless hours of testing, we finely tuned our capture methods (hardware and software) to produce very professional and reproducible images. Another significant improvement was background-omittance. We discovered that darker backgrounds fade to black for both the UV and NIR filters while not disturbing the RGB and or IR. This would allow us to perform combinational indexing more easily in future testing.

Figure 30: RGB image in natural lighting conditions

Figure 31: 550nm UV filter in natural lighting conditions

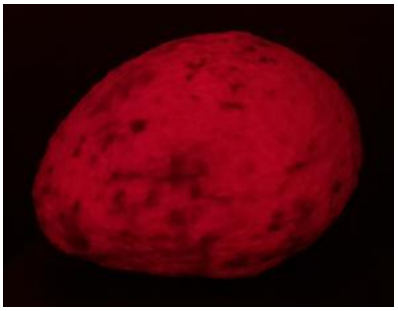


Figure 32: 742nm NIR filter in natural lighting conditions

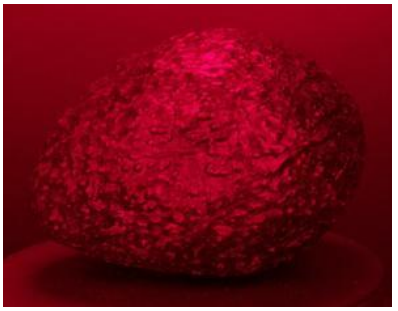


Figure 33: 795 IR filter in natural lighting conditions

Another very important part of dataset creation was having a functional autonomous rotating platform as previously mentioned. The engineering diagram of the turntable can be found in the Appendices - figure C.4. We managed to integrate the servo motor's python script with the capture code which allows us to dramatically speed up the dataset creation phase. The final dataset creation box is shown in figure 34.

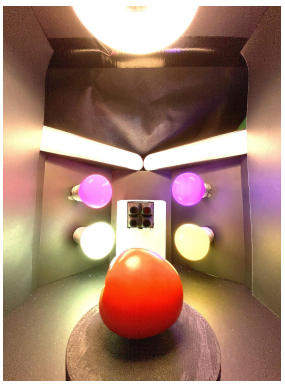


Figure 34: Lightbox design with Hue colour lights and autonomous turntable

We interfaced our MS camera with the turntable by directly connecting it to the GPIO ports of our Raspberry Pi. Then, a Python script is executed from our Go server to rotate the platform by the specified number of degrees per image capture. We chose Python for this interface due to its readily available GPIO library designed specifically for the Raspberry Pi.

3. Dataset Creation

Once all the hardware, software and setup issues were resolved, we were able to build large datasets of images to represent different produce. We opted for red apples as our initial dataset as we used them in the previous phase and wanted to compare our results. We also decided to widen our scope by visually testing other produce items such as bell peppers, lemons, oranges, tomatoes, and avocados. Through this exploration, we noticed that the avocado MS pictures had the most promising visual results. Unfortunately due to the long capture time for each dataset (about 14 hours), we decided that only two could be prioritized within this semester. Therefore, we focused on avocados and apples for the datasets. large multispectral datasets were then created of apples (6550 images) (figure 44), and avocados (7640 images) (figures 35-42). In addition, thanks to our setup and our autonomous turntable linked to the camera, we were able to take pictures of the same produce with 10° rotations between each capture.

Figure 35: Good and Bad Avocado - RGB filter

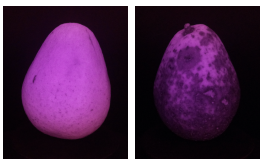


Figure 36: Good and Bad Avocado - UV filter

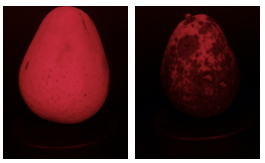


Figure 37: Good and Bad Avocado - NIR filter

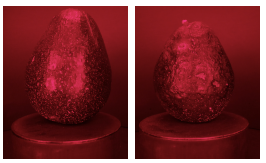


Figure 38: Good and Bad Avocado - IR filter

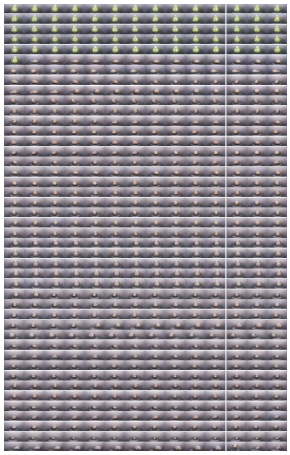


Figure 39: Avocado dataset - RGB filter

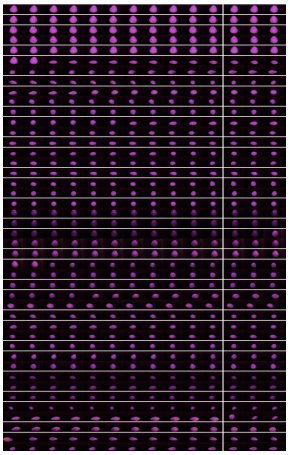


Figure 40: Avocado dataset - UV filter

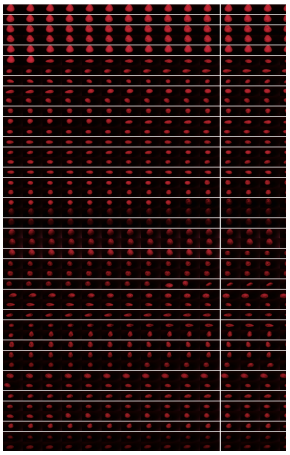


Figure 41: Avocado dataset - NIR filter

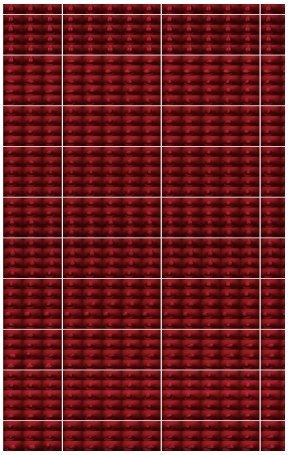


Figure 42: Avocado dataset - IR filter

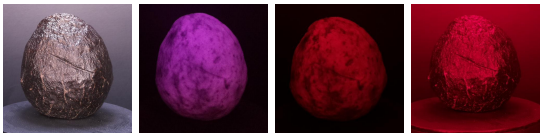


Figure 43: Avocado with all 4 filters

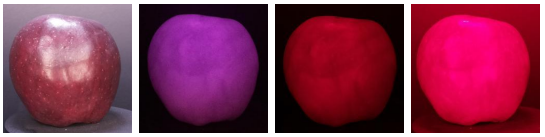


Figure 44: Apple with all 4 filters

The lightbox (enclosed in a dark black surface) creates images with omitted background noise. These example pictures are raw captures before any cropping or image manipulation. The time required for each dataset is approximately 14 hours but would be significantly longer if done manually without the turntable. Despite the fact that the turntable speeded up the process of taking pictures, the system still was really slow to take all the pictures, and creating the dataset was extremely time-consuming. Therefore we had to wait to pass the threshold of pictures that we had estimated in Part 1 of this project of produce in good and bad shape in order to implement CNN analysis correctly.

4. CNN Based on the Multispectral Images

Once the dataset is composed of enough pictures with different quality of produce, we were able to focus on implementing, testing, training and validating CNNs on the multispectral images collected.

As a reminder, our results from the previous semester showed us that even small neural networks like MobileNet were able to achieve high accuracies with image recognition, thus we based our networks on MobileNetV2. We also used transfer learning again as it allowed us lower training times as fewer parameters are trained while maintaining the required high accuracy for fruit recognition.

We considered two potential approaches:

- The first approach was to keep feeding the network three channelled images, however, this time as we had additional bands for each image and in order to maintain the three-channel limit, we would have to combine different single banded images that were captured from our multispectral camera into one false-colour image composite. This technique is used for satellite imagery to get more information about vegetation health and combines bands including ultraviolet (UV), infrared (IR) and near-infrared (NIR). Our false colour images would be used for ingredient recognition as well as determining if it is healthy or not. This approach would allow us to use a similar network to MobileNet, but all the extra data we collected would not be fully used. However as images are taken from different angles from each camera filter, images would need to be perfectly aligned using keypoint detection, feature mapping and homography estimation as images need to be matched exactly. Due to the time restrictions on our project, this approach has been deemed not optimal.

- The second approach we had in mind would take advantage of all the extra data that we collected with our new camera. Combining certain one-band images into a three-channel image would not fully utilize all the collected multispectral data from our new dataset. This approach would take all six bands (R, G, B, UV, IR and NIR) and combine them into a single 6 channel image to preserve all data. However, this would require considerable restructuring of the network as our network was designed for 3 channel images and not six. This approach would be also more computationally intensive as the number of parameters increases with the input images having more channels and some parameter reduction using pointwise convolutions would be necessary to achieve faster training times.

Inspecting our multispectral images, we noticed that some individual filters were giving us enough data to distinguish between a healthy and unhealthy fruit as well as provide enough data for fruit recognition. Thus we decided to input each individual band image separately into the network to test health assessment on each band individually.

To further evaluate our model, we automatically computed and generated the confusion matrix for each band filter after the testing phase. This step was added in our testing code pipeline to automatically generate the confusion matrix, plot it and compute all the required values for monitoring the model's performance. Accuracy by itself is not good enough of an indicator to give the full picture of the performance of our model. The confusion matrix allows us to get the number of true positives/negatives and the number of false positives/negatives for each category of images. This will allow us to compute the accuracy, recall, precision, false positive rate and false negative rate. Recall is used to measure the capability of our model to correctly identify true positives and determine what proportion of actual positives was identified correctly. Precision tells us what proportion of positive identifications was actually correct. Having a recall value close to 1 means our model produces very few false negatives and having a precision close to 1 produces very few false positives.

$$Recall = \frac{True\ positive}{True\ positive + false\ negative} \qquad Precision = \frac{True\ positive}{True\ positive + false\ positive}$$

NIR:

Class	Training images number	Validation images number	Testing images number
Bad Apple	501	126	133
Bad Avocado	504	247	252
Healthy Apple	434	219	221
Healthy Avocado	455	225	224

Table 13: NIR Dataset split

- Best epoch: epoch 2
- Total epochs: 7
- Early stopping: Yes
- Loss: 0.18
- Training time: 308.27 seconds (39 seconds per epoch)

Figure 45: NIR training and validation losses

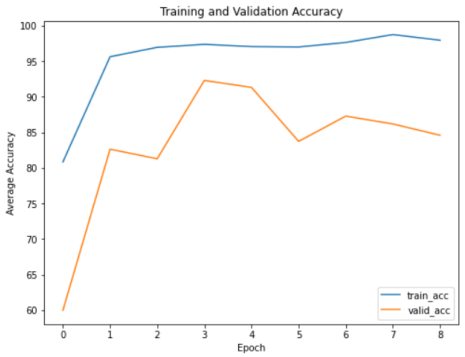


Figure 46: NIR training and validation accuracies

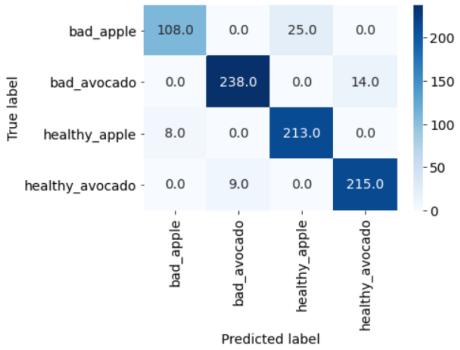


Figure 47: Confusion matrix NIR

	Bad Apple	Bad Avocado	Healthy Apple	Healthy Avocado
Accuracy	0.96024096	0.97228916	0.96024096	0.97228916
Recall (true positive rate)	0.81203008	0.94444444	0.9638009	0.95982143
Precision	0.93103448	0.96356275	0.89495798	0.93886463
False positive rate	0.01147776	0.01557093	0.0410509	0.02310231
False negative rate	0.18796992	0.05555556	0.0361991	0.04017857

Table 14: NIR model results

UV:

Class	Training images number	Validation images number	Testing images number
Bad Apple	403	175	182
Bad Avocado	504	247	252
Healthy Apple	434	224	216
Healthy Avocado	455	225	224

Table 15: UV Dataset split

- Best epoch: epoch 13
- Total epochs: 18
- Early stopping: Yes
- Loss: 0.22
- Training time: 776.28 seconds (41 seconds per epoch)

Figure 48: UV training and validation losses

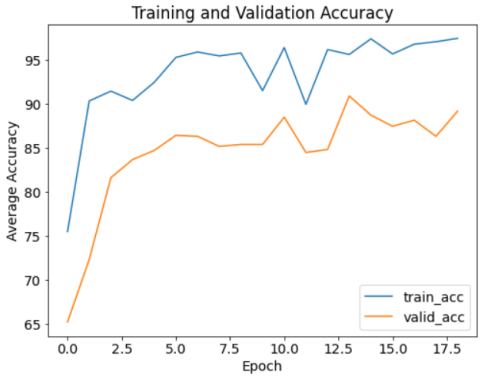


Figure 49: UV training and validation accuracies

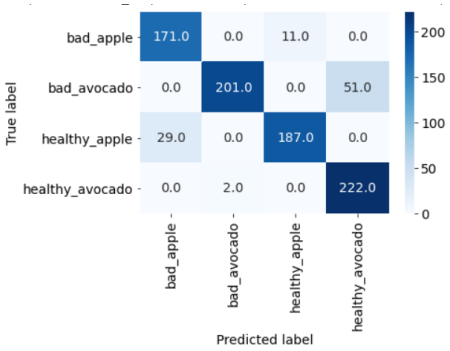


Figure 50: Confusion matrix UV

	Bad Apple	Bad Avocado	Healthy Apple	Healthy Avocado
Accuracy	0.95423341	0.93935927	0.95423341	0.93935927
Recall (true positive rate)	0.93956044	0.79761905	0.86574074	0.99107143
Precision	0.855	0.99014778	0.94444444	0.81318681
False positive rate	0.04190751	0.00321543	0.01671733	0.07846154
False negative rate	0.06043956	0.20238095	0.13425926	0.00892857

Table 16: UV model results

IR:

Class	Training images number	Validation images number	Testing images number
Bad Apple	410	175	259
Bad Avocado	504	247	252
Healthy Apple	399	216	175
Healthy Avocado	448	225	231

Table 17: IR Dataset split

- Best epoch: epoch 2
- Total epochs: 7
- Early stopping: Yes
- Loss: 0.83

- Training time: 482.15 seconds (60 seconds per epoch)

Figure 51: IR training and validation losses

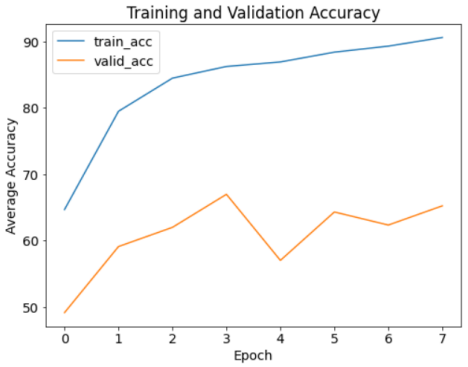


Figure 52: IR training and validation accuracies

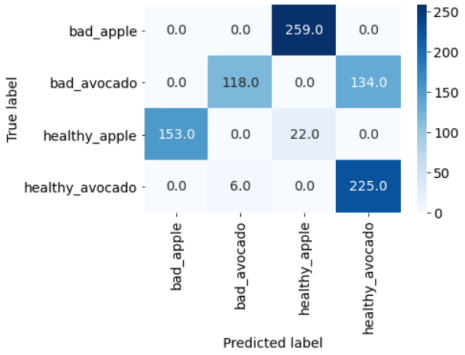


Figure 53: Confusion matrix IR

	Bad Apple	**Bad Avocado**	**Healthy Apple**	**Healthy Avocado**
Accuracy	0.55070883	0.84732824	0.55070883	0.84732824
Recall (true positive rate)	0	0.46825397	0.12571429	0.97402597
Precision	0	0.9516129	0.07829181	0.62674095
False positive rate	0.2325228	0.00902256	0.3490566	0.19533528
False negative rate	1	0.53174603	0.87428571	0.02597403

Table 18: IR model results

RGB:

Class	**Training images number**	**Validation images number**	**Testing images number**
Bad Apple	384	192	184
Bad Avocado	504	247	252
Healthy Apple	440	218	216
Healthy Avocado	455	225	252

Table 19: RGB model results

- Best epoch: epoch 5
- Total epochs: 10
- Early stopping: Yes

- Loss: 0.70
- Training time: 663.31 seconds (60 seconds per epoch)

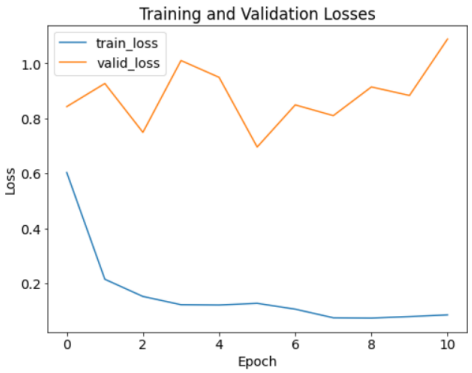


Figure 54: RGB training and validation losses

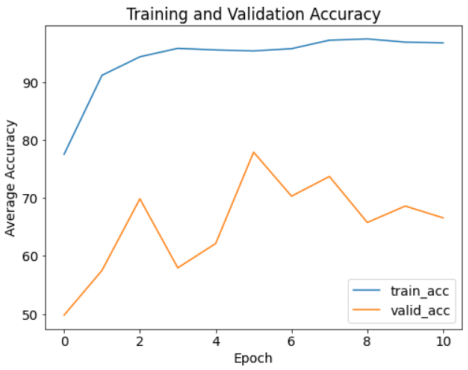


Figure 55: RGB training and validation accuracies

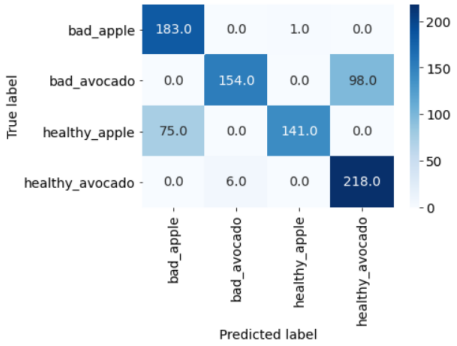
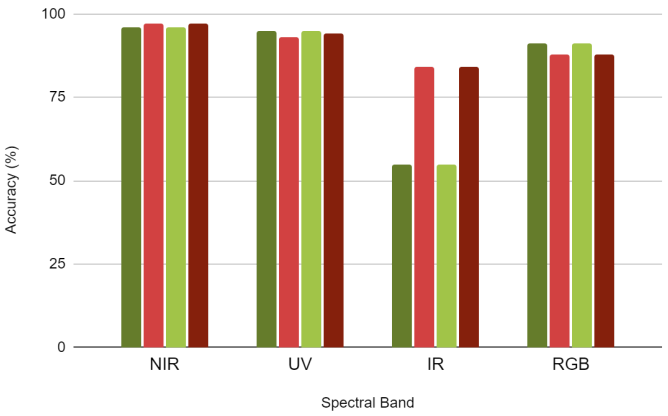


Figure 56: Confusion matrix RGB

	Bad Apple	Bad Avocado	Healthy Apple	Healthy Avocado
Accuracy	0.91324201	0.88127854	0.91324201	0.88127854
Recall (true positive rate)	0.99456522	0.61111111	0.65277778	0.97321429
Precision	0.70930233	0.9625	0.99295775	0.68987342
False positive rate	0.1083815	0.00961538	0.00151515	0.15030675
False negative rate	0.00543478	0.38888889	0.34722222	0.02678571

Table 20: RGB model results



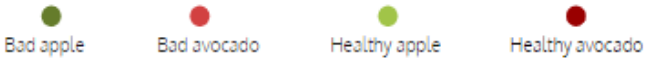


Figure 57: Recognition accuracy per fruit class on MobileNetV2

Band Filter	Average recognition accuracy
NIR	96.5%
UV	94.0%
IR	69.5%
RGB	89.5%

Table 21: Recognition accuracy percentage per fruit class on MobileNetV2

As can be seen from the results above, our model was not only able to distinguish between avocados and apples when it comes to using different bands, it was also able to correctly assess the health of each individual fruits with high accuracy for the Near Infrared band as well as UltraViolet with Infrared performing the worst out of all bands. NIR images used as input performed the best in our model as it had the lowest loss (0.18), the least overfitting as can be seen in each training and validation loss graphs above. It also gave the highest accuracy scores, precision and recall (values were close to 1) while having the lowest false positive and false negative rates. UV provided good results as well, however, it had more overfitting while training with slightly lower accuracy scores and higher false negative rate for bad avocados. IR performed the worst out of all, especially struggling with the distinction between healthy and bad apples. It had the highest training loss (0.83), overfitting and the lowest accuracy scores. The IR images for apples did not provide enough data to allow the model to correctly assess its health by only using images from the IR camera filter. While RGB provided us with good results for fruit recognition, it struggled with the health distinction. Training on RGB images also resulted in considerably more overfitting as can be seen in figure 54 and false negative rates were the highest for bad avocados and healthy apples as can be seen in table 20 with high false positive rates for bad apples and healthy avocados as well. Thus as can be seen from the previous results, combining and using all Bands from all our cameras was not strictly necessary to achieve high accuracies in fruit recognition and health assessment. NIR performing best could be used on its own, closely followed by UV to recognize and distinguish between healthy and unhealthy fruit.

Impact on Society and the Environment

As engineers, we must continuously assess the adverse effects of our actions on the world around us. Food processing is one of the largest industries in modern society. Processing facilities perform numerous stages of quality-assurance inspections on their produce before it reaches end consumers. Manual inspection is still a common practice in performing these mundane tasks, but this entails human error and time constraints. The role and fundamental goal of this project are to determine the advantages and limitations of embedding machine-learning processes in the food industry. Integrating new technology in this field has the potential to drastically improve efficiency, costs, and accuracy which in effect means fresher produce available to an ever-growing population.

1. **Use of non-renewable resources**

Throughout the development stage, we became confident that our project did not have a significant adverse impact on the environment. Most of our initial concerns, such as power usage and hardware materials, were alleviated by using environmentally conscious alternatives.

Indeed, our project did not have much of an impact in regards to the final product. Throughout this project, and especially the first phase, we expressed concerns about the electricity requirements that would be needed to train and test the CNN models. This process is fundamentally power-heavy as the computations are complex and entail the use of GPUs. To address this, our software engineers decided to use cloud GPU services provided by Google Colab hosted on Google Cloud Platform which offsets its electricity and is considered carbon neutral.

2. **Environmental benefits**

As mentioned above, we succeeded in reducing the energy consumption from the CNN training and deployment. In addition, for the prototyping phase of our hardware casement, we utilized PLA filament materials. PLA is a biodegradable thermoplastic that enters a soft and moldable state when heated and then returns to a solid when cooled. It is derived from renewable resources such as corn starch or sugarcane and will biodegrade within 50 days in industrial composters and 48 months in water.

The use of PLA filament for prototyping was important as we had to print numerous designs in order to troubleshoot lens placement. Once settled on a final design, we switched to an SLA resin printer as it has better precision and is more structurally rigid. The rest of the hardware equipment like the camera sensors and the Raspberry Pi can last numerous years without having to be changed and with minimal degradation. For the final product, we anticipate that our system can actually create a positive impact in terms of food waste. Manual produce inspection works in a binary fashion; either the produce is ‘satisfactory’ (1) or ‘unsatisfactory’ (0). Having a computerized grading system would allow for more classification groups based on shape, ripeness, and damage. Items that are considered ‘too ugly’ for supermarket selection could still get classified as edible and set aside in another subgroup for soup kitchens and public welfare initiatives.

3. **Safety and risk**

As aforementioned, we decided to use at the end an SLA 3D printer as it allowed for more precise printing for both the filter casing and the rotating table that was used to take pictures. However, we had to respect all the necessary safety precautions with SLA since the resin it uses can be toxic.

Alongside the modernization efforts felt throughout society, technological automation has systematically replaced the need for humans across numerous industries. This shift led to dramatic efficiency and precision improvements, but also negatively impacted the need for certain unskilled

workforces. In Canada, the food and beverage processing industries provide employment for 290,000 Canadians [21]. While these jobs are spread out over numerous fields such as food delivery logistics, a large portion are involved with the end-to-end inspection. Implementing more automation technology would lead to less reliance on unskilled labour for repetitive tasks. Many ethical issues arise with this trend as unskilled jobs would become more scarce. One argument is that this advancement would allow managers to hire more employees for marketing purposes, but these new jobs would not translate well to previous employees. Adapting to technology is a continuous uphill battle which can leave many behind. From an ethics standpoint, automation can drastically alter the social fabric of society.

4. Benefits to society

In mid-November, we visited our sponsor's processing facility to gain more insight into the industry. Our visit highlighted how computerized inspection methods could potentially help reduce food waste. Currently, employees sift through tens of thousands of produce items a day in order to separate out those with defects. This extensive process was noticeably variable in some regards as it is heavily based on human perception and the worker's attention to detail. Repeating the same task for several hours at a time can understandably lead to careless mistakes. The issue is, if a compromised vegetable gets packaged alongside healthy produce, the entire meal-kit box would be jeopardized. Computer automation on the other hand would not showcase the same pitfalls for the same repetitive tasks. With proper implementation, we believe computer vision systems could positively impact the inspection process for food safety while also ensuring that only truly defective produce gets deemed illegible. Furthermore, we believe that installing a similarly automated system throughout the supply chain would enable the composting of the defective produce, as often, these produces are thrown out as they are too close to the end consumer and too far from the producer. This application would reduce the use of chemical fertilizer and increase the use of natural compost-based fertilizer.

References

[1] Python, "Applications for Python," *Python™, 2001-2020*. [Online] Available: https://www.python.org/about/apps/ [Accessed: Nov. 23, 2020].

[2] MissingLink.Ai, "Python Convolutional Neural Network: Creating a CNN in Keras, TensorFlow and Plain Python," in *missinglink.ai*. [Online]. Available: https://missinglink.ai/guides/convolutional-neural-networks/python-convolutional-neural-network-creating-cnn-keras-tensorflow-plain-python/. [Accessed Nov. 22, 2020].

[3] A. Shepherd, "What is a GPU?," *ITPRO.*, October 5, 2020. [Online]. Available: https://www.itpro.co.uk/hardware/30399/what-is-a-gpu. [Accessed Nov. 22, 2020]

[4] Raspberry Pi Foundation, "What is a Raspberry Pi?", 2012-2020. [Online]. Available: https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/. [Accessed Dec. 3, 2020]

[5] "Raspberry Pi 3 Model B," *Génération Robots*. [Online]. Available: https://www.generationrobots.com/en/402366-raspberry-pi-3-model-b.html. [Accessed: 16-Apr-2021].

[6] J. Brownlee, "A Gentle Introduction to Transfer Learning for Deep Learning," in *Deep Learning for Computer Vision*. Machine Learning Mastery, [Online Document], December

20, 2017. Available: https://www.itpro.co.uk/hardware/30399/what-is-a-gpu. [Accessed: Nov. 22, 2020].

[7] R. Gour, "Transfer Learning for Deep Learning with CNN," in *7 min read*. Medium, November 28, 2018. [Online]. Available: https://medium.com/@rinu.gour123/transfer-learning-for-deep-learning-with-cnn-afa1fe0aa7e2. [Accessed: Nov. 22, 2020].

[8] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks – the ELI5 way," in *7 min read.* Towards Data Science, December 15, 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: Nov. 22, 2020].

[9] DeepAI, "Hidden Layer," *DeepAI*, 17-May-2019. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning#:~:text=In%20neural%20networks%2C%20a%20hidden,inputs%20entered%20into%20the%20network. [Accessed: 15-Apr-2021].

[10] RP Photonics Encyclopedia, "Multispectral Imaging," *RP Photonics Encyclopedia.*[Online]. Available: https://www.rp-photonics.com/multispectral_imaging.html#:~:text=Multispectral%20imaging%20means%20methods%20for,the%20infrared%20and%20ultraviolet%20region. [Accessed: Nov. 22, 2020].

[11] G. Miller, "Multispectral Imaging and the Mona Lisa," Teledyne Imaging, February 2, 2016. [Online]. Available: https://possibility.teledyneimaging.com/mona-lisa/. [Accessed: Nov. 23, 2020].

[12] Jeff Kerns, "3D Printing Saves the World, Part 1. How one process may single-handedly solve some of society's greatest problems.,", March 24, 2018. [Online]. Available: https://www.machinedesign.com/3d-printing-cad/article/21836554/3d-printing-saves-the-world-part-1. [Accessed: 04-Apr-2021].

[13] CDC, "How Food Gets Contaminated - The Food Production Chain," *Centers for Disease Control and Prevention*, 05-Sep-2017. [Online]. Available: https://www.cdc.gov/foodsafety/production-chain.html. [Accessed: 06-Dec-2020].

[14] J.P Chan, B.G. Batchelor, "Machine Vision for the Food Industry," *Food Process Monitoring Systems*, pp58-101, 1993. [Abstract]. Available: LinkSpringer, https://link.springer.com/chapter/10.1007/978-1-4615-2139-6_4. [Accessed: Nov. 23, 2020].

[15] Jason Brownlee, July 5 2019, "A Gentle Introduction to the ImageNet Challenge (ILSVRC)," https://machinelearningmastery.com/introduction-to-the-imagenet-large-scale-visual-recognition-challenge-ilsvrc/. [Online]. [Accessed: Nov. 23, 2020].

[16] M.U. Hassan, "VGG16 – Convolutional Network for Classification and Detection," in *Popular Networks*. Neurohive, [Online], November 20, 2018. Available: Neurohive, https://neurohive.io/en/popular-networks/vgg16/. [Accessed: Nov. 23, 2020].

[17] M. Sandler, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *The latest in machine learning*, 21-Mar-2019. [Online]. https://paperswithcode.com/method/mobilenetv2. [Accessed: 16-Apr-2021].

[18] M.U. Hassan, “ResNet (34, 50, 101): Residual CNNs for Image Classification Tasks,” in *Popular Networks*. Neurohive, [Online], January 23, 2019. Available: Neurohive, https://neurohive.io/en/popular-networks/resnet/. [Accessed: Nov. 23, 2020].

[19] F. Li, R. Krishna, D. Xu. CS231n. Class Lecture, Topic: "CNN Architectures.” School of Electrical and Computer Engineering, Stanford University, Stanford, CA, April 2020. [Online] Available: http://cs231n.stanford.edu/. [Accessed: Nov. 23, 2020].

[20] J. Qin, K. Chao, M. S. Kim, R. Lu, and T. F. Burks, “Hyperspectral and multispectral imaging for evaluating food safety and quality,” *Journal of Food Engineering*, 11-Apr-2013. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0260877413001659. [Accessed: 04-Apr-2021].

[21] A. A. F. C. Canada, “Government of Canada / Gouvernement du Canada,” *Agriculture and Agri-Food Canada (AAFC)*, July 16, 2020. [Online]. Available: https://www.agr.gc.ca/eng/food-products/processed-food-and-beverages/overview-of-the-food-and-beverage-processing-industry/?id=1174563085690. [Accessed: Nov. 23, 2020].

Appendices

Graphs - VGG16 from Scratch:

The graphs below represent the loss (Vertical axis) vs. epoch (Horizontal axis) for different number of images

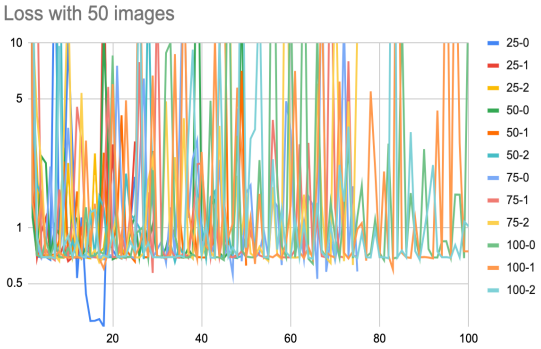


Figure A.1: Loss with 50 images

Figure A.2: Loss with 100 images

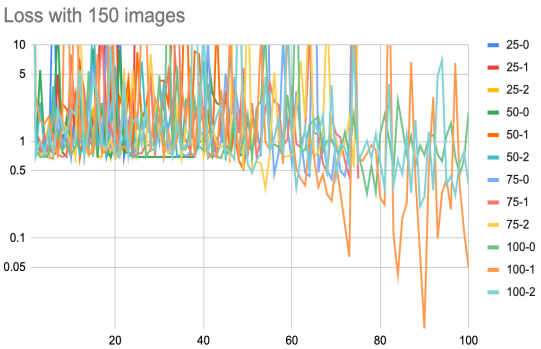


Figure A.3: Loss with 150 images

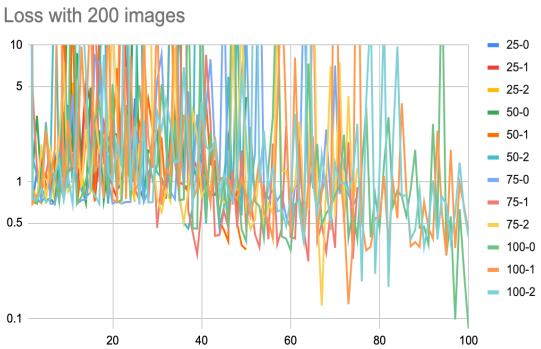


Figure A.4: Loss with 200 images

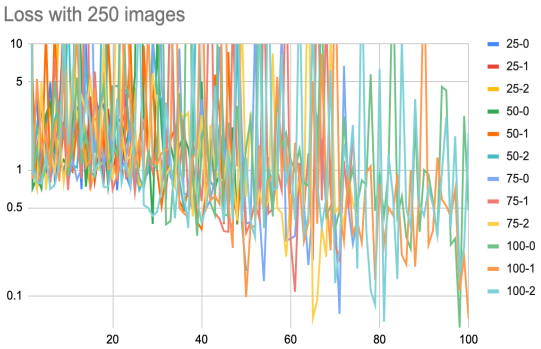


Figure A.5: Loss with 250 images

Figure A.6: Loss with 300 images

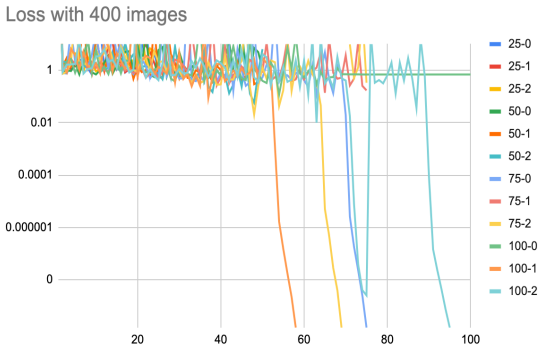


Figure A.7: Loss with 400 images

Figure A.8: Loss with 500 images

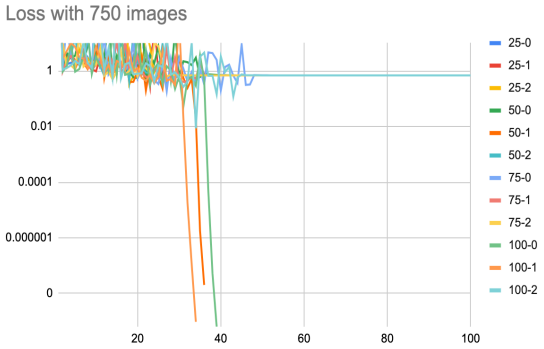


Figure A.9: Loss with 750 images

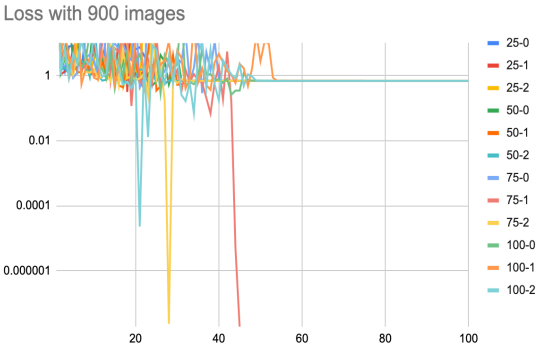


Figure A.10: Loss with 900 images

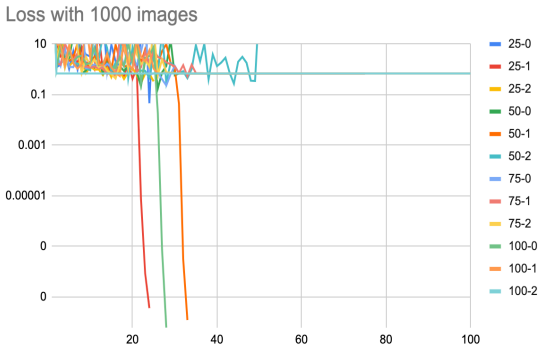


Figure A.11: Loss with 1000 images

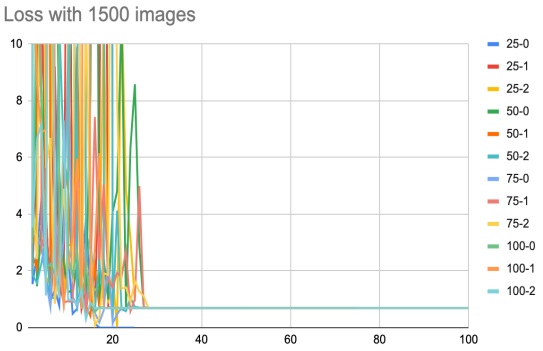


Figure A.12: Loss with 1500 images

Figure A.13: Loss with 2000 images

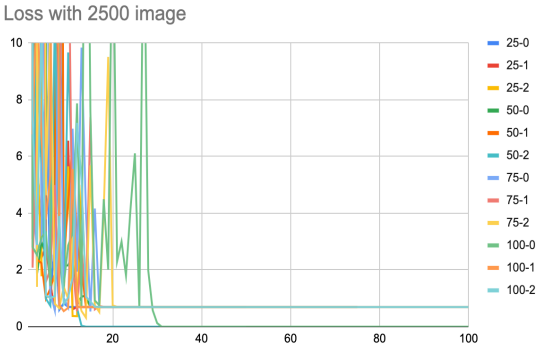


Figure A.14: Loss with 2500 images

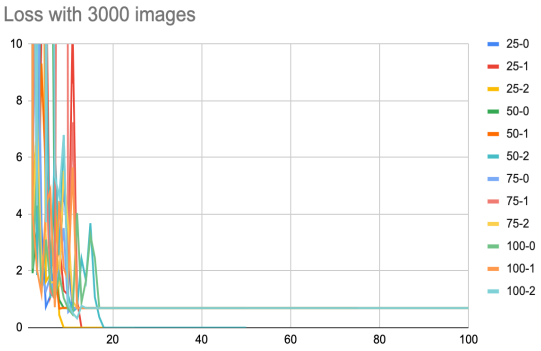


Figure A.15: Loss with 3000 images

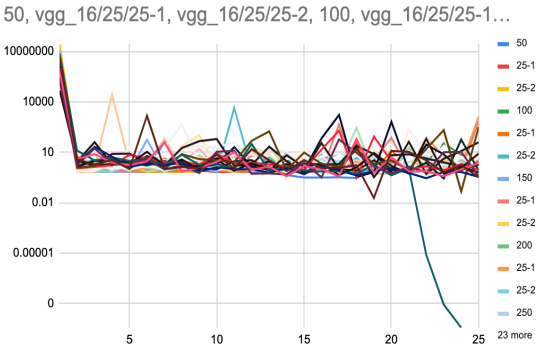


Figure A.16: Details for epoch 25

Graphs - MobileNet-V2 from Scratch

The graphs below represent the loss (Vertical axis) vs. epoch (Horizontal axis) for different number of images

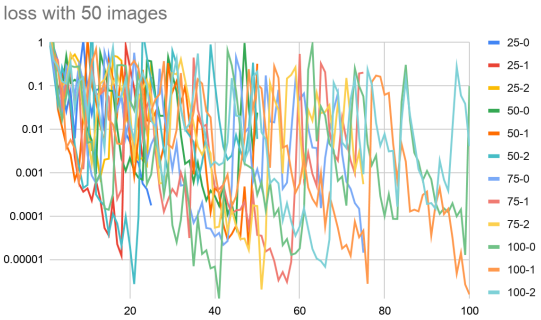


Figure B.1: Loss with 50 images

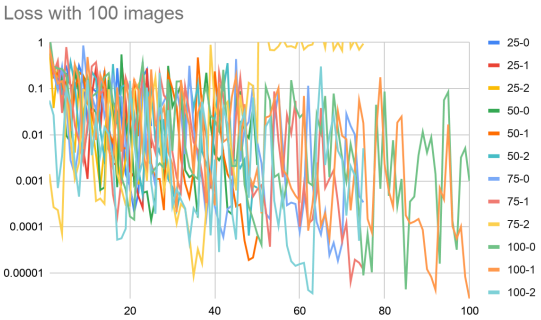


Figure B.2: Loss with 100 images

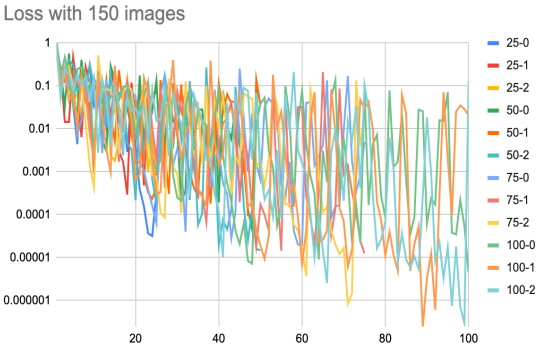


Figure B.3: Loss with 150 images

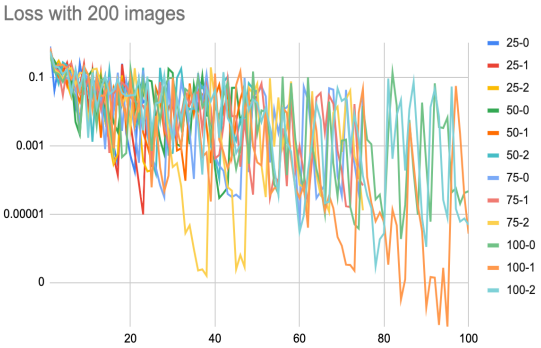


Figure B.4: Loss with 200 images

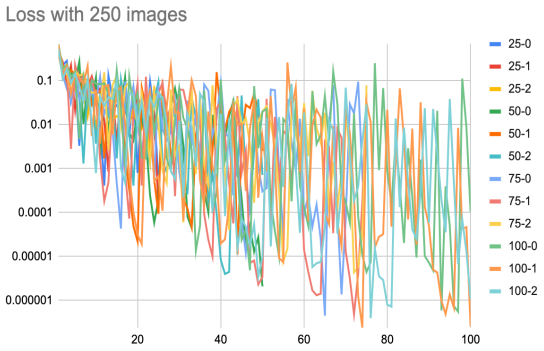


Figure B.5: Loss with 250 images

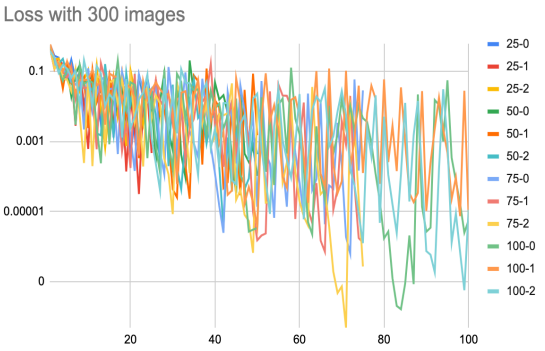


Figure B.6: Loss with 300 images

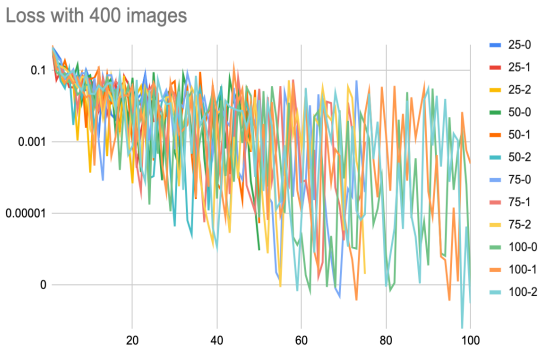


Figure B.7: Loss with 400 images

Figure B.8: Loss with 500 images

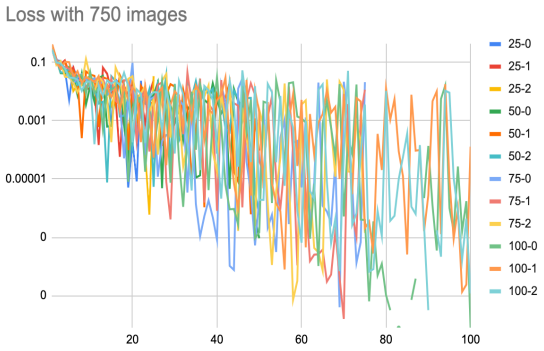


Figure B.9: Loss with 750 images

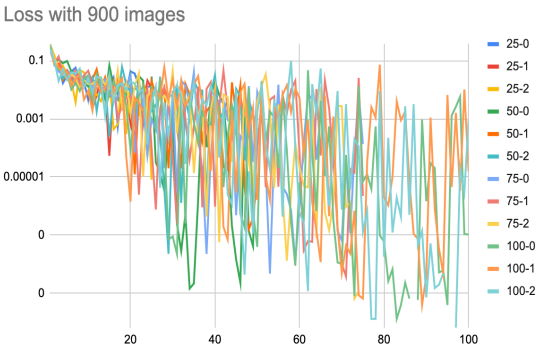


Figure B.10: Loss with 900 images

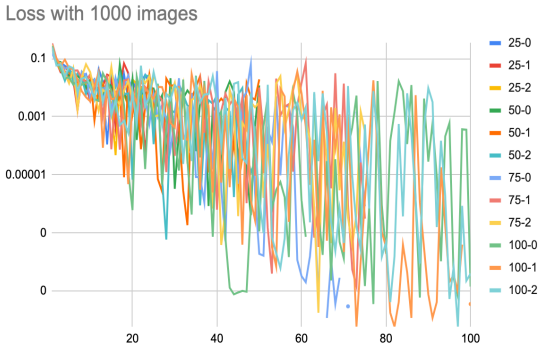


Figure B.11: Loss with 1000 images

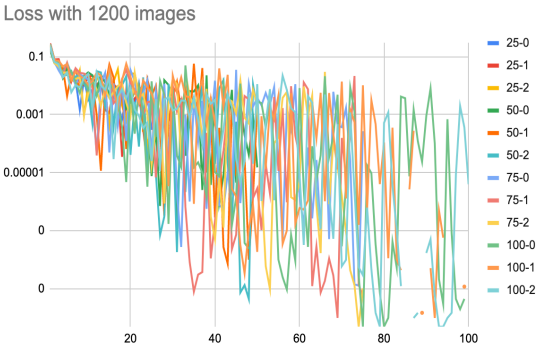


Figure B.12: Loss with 1200 images

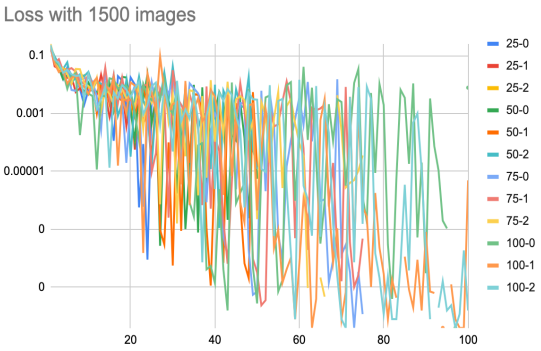


Figure B.13: Loss with 1500 images

Figure B.14: Loss with 2000 images

Figure B.15: Loss with 2500 images

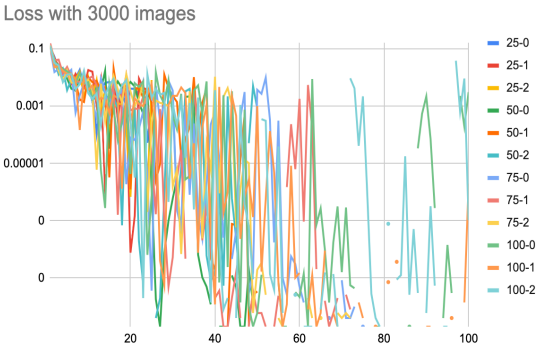


Figure B.16: Loss with 3000 images

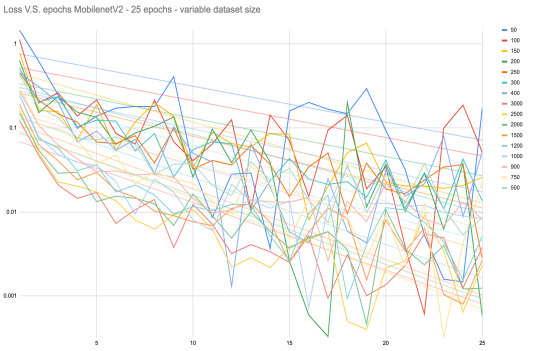


Figure B.17: Details for epochs 25

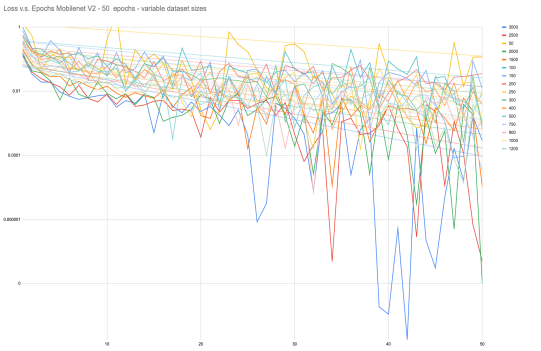


Figure B.18: Details for epochs 50

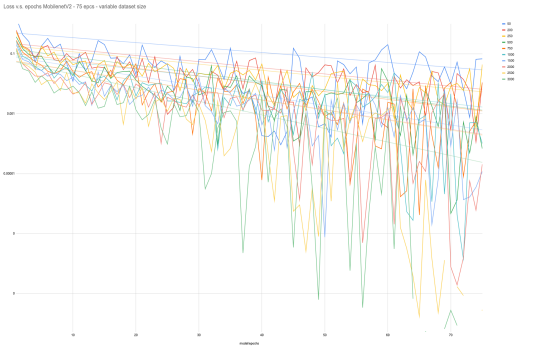


Figure B.19: Details for epochs 75

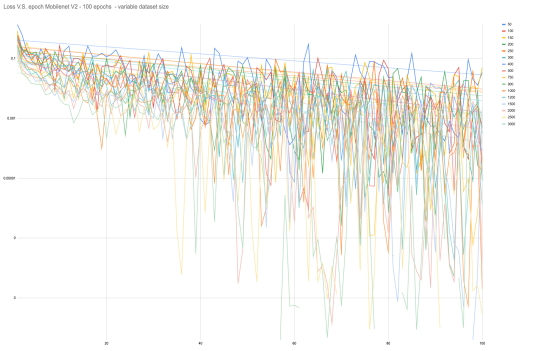


Figure B.20: Details for epochs 100

Engineering Diagrams for the Camera holder and the Turntable:

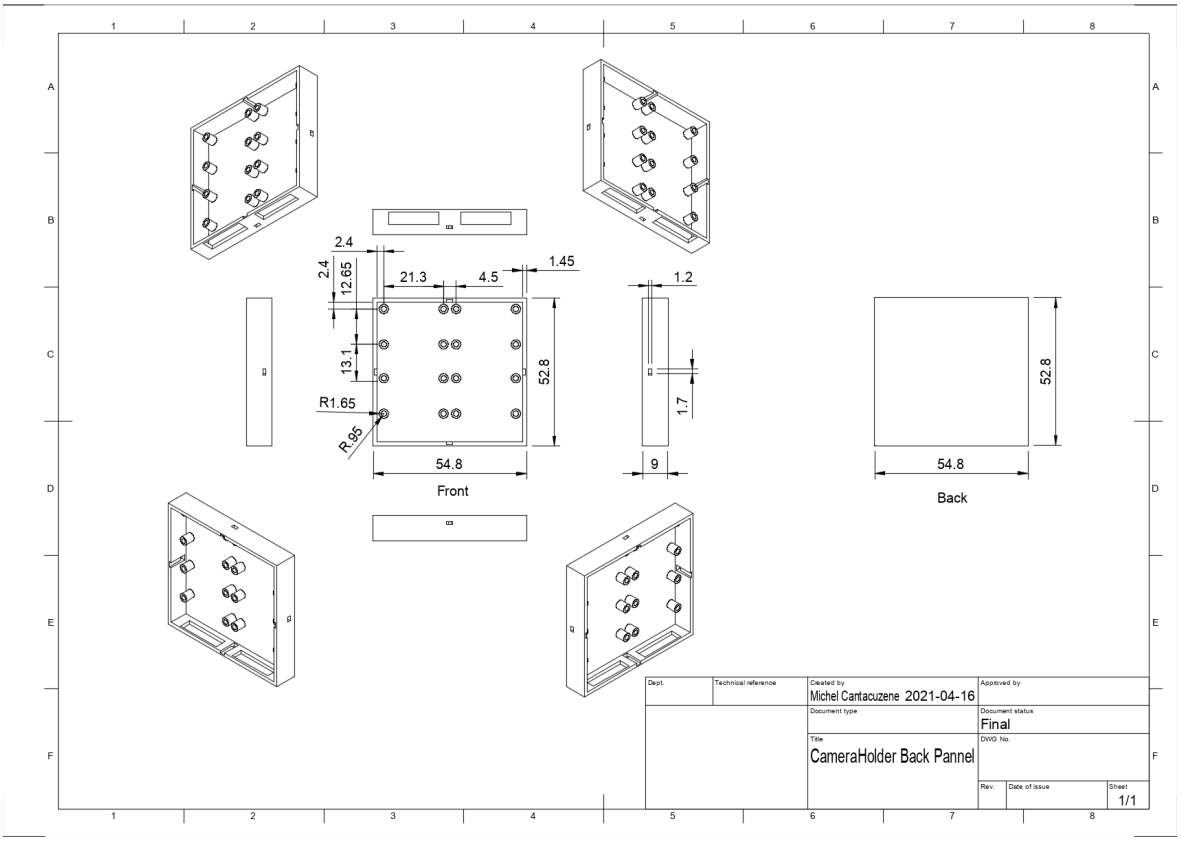


Figure C.1: Engineering Diagram for the Camera holder

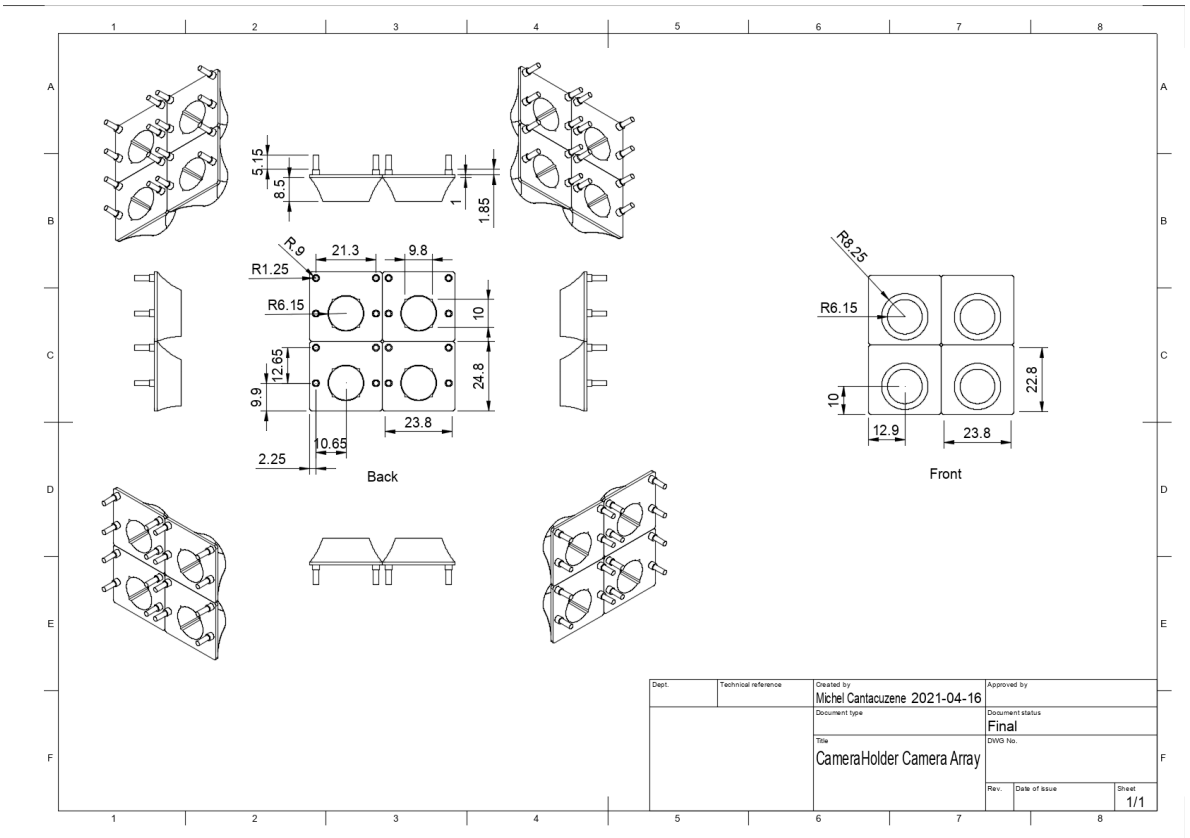


Figure C.2: Engineering Diagram for the Camera holder

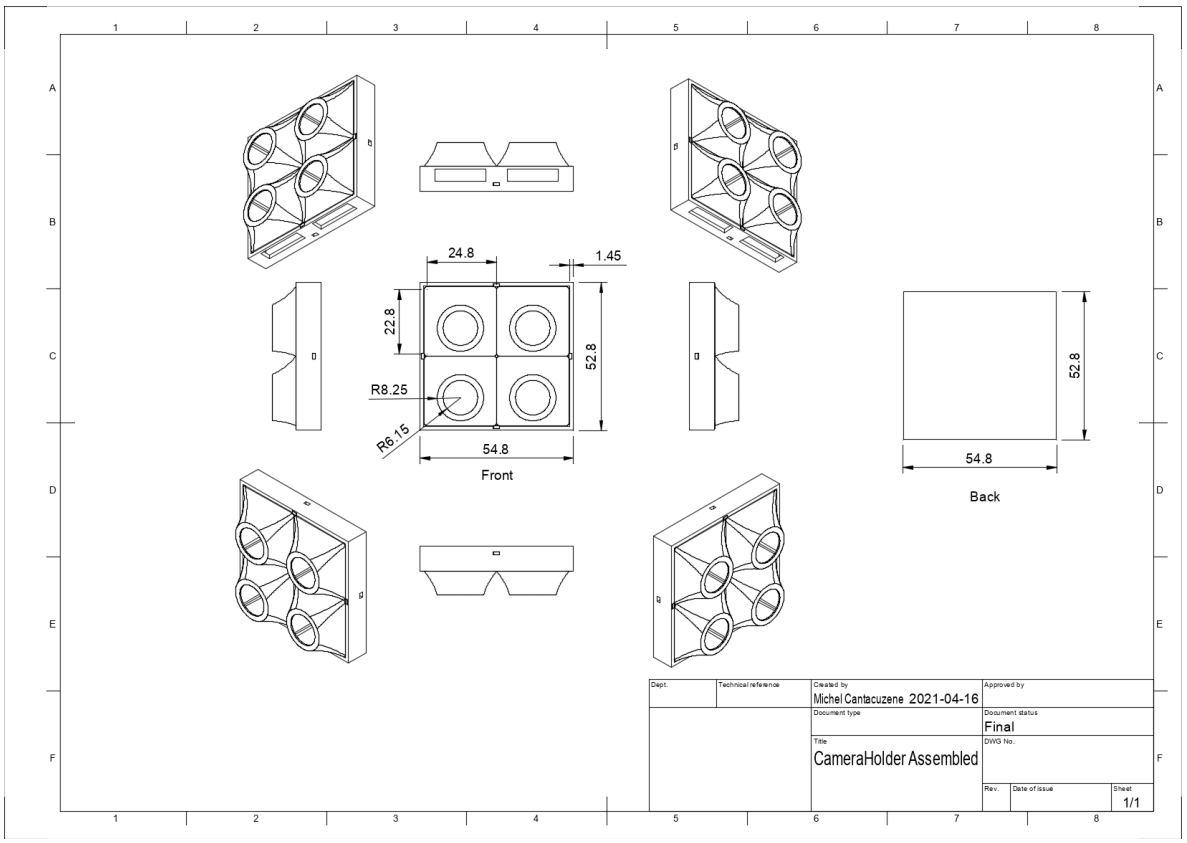


Figure C.3: Engineering Diagram for the Camera holder

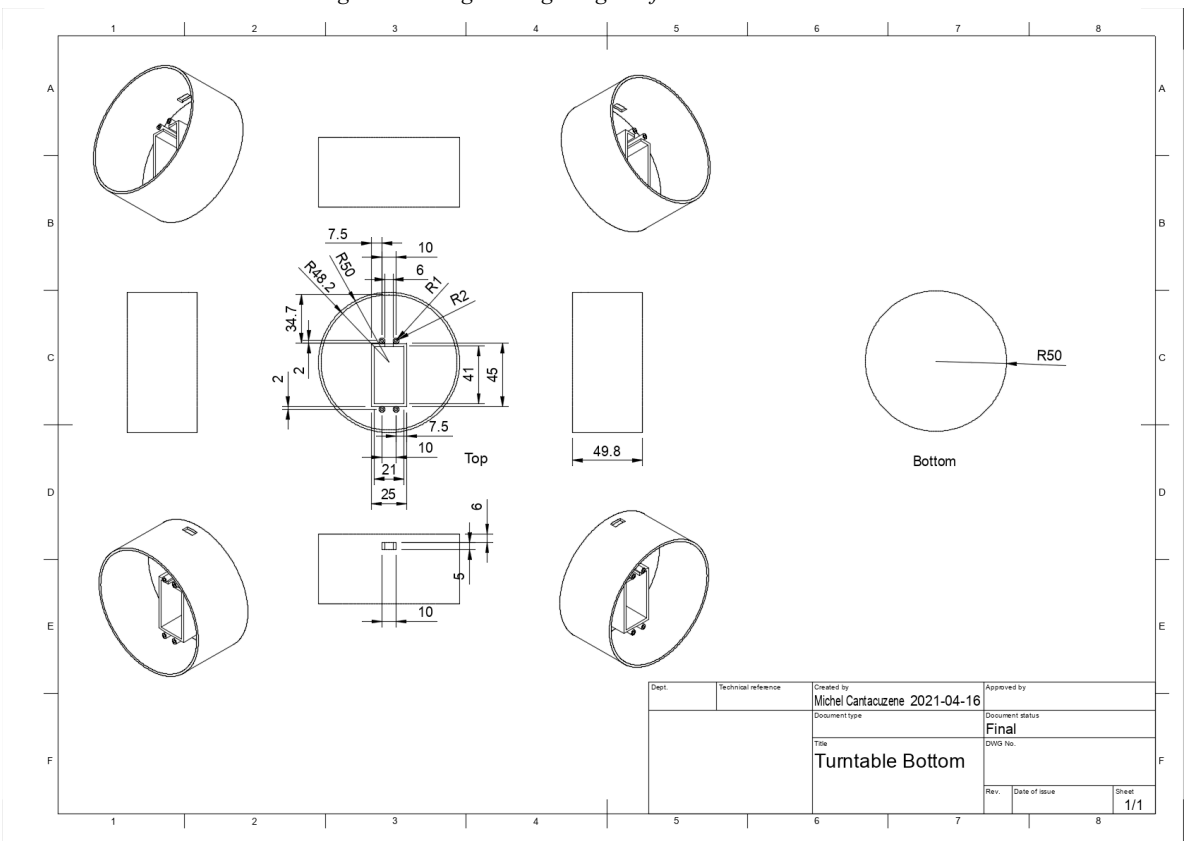


Figure C.4: Engineering Diagram for the Turntable