Title: Garden Guardian
Group Number: 4
Group Members: Ching Zheng, Nimish Kapoor, Xiang Chen, Henry Nguyen, Jackson Lim

Abstract

When you see or hear the word "pest", you probably envision tiny insects that nibble on your vegetable garden. However, some of the more threatening pests to your garden may come in the shape of mammals - rabbits, deers, racoons or potentially even cats - that devour your ripe lettuce and tomatoes. There are many ways to prevent this from happening: with solutions ranging from spraying coyote urine around the area, to just simply erecting a fence around your garden. But, these solutions are costly, laborious, smelly, and short term fixes at best. To solve this problem, we will apply our knowledge of computer vision to create an aptly named product - "Garden Guardian". Garden Guardian will detect and identify mammalian pests and mitigate their presence in your garden with a little water gun. The plan for the project is to use live video footage to identify pests and shoo it away with a spray of water. The software will be able to distinguish whether the object is a "pest" or a human. Another feature can include selecting animals that you don't want the software to identify as pests. We hope that by applying our knowledge of machine learning, computer vision, and software development, we will be able to effectively protect everyone's produce.

Management Plan

Collaboration and organization

For a major part of our communication, we used a free VoIP application called Discord. Discord allows us to create a server for this project to chat, have group meetings, and link potential resources for our project. The plan is to have at least weekly meetings throughout the semester to collaborate on major deadlines and communicate problems and solutions about the project. Throughout the week, there will be occasional updates (similar to that of e-scrum) about our individual progress. We also created a Github repository and Google Drive shared to all the team members for this project. We've organized it so that any meeting notes and links will be posted on Discord, any text documents, datasets, and papers to be stored in Google Drive, and all the source code to be stored in Github.

Team work distribution and version control

Initially, we noted that the neural network part of this project was going to be the majority of the work, so we started off with everyone looking for datasets of animals as well as tutorials and code examples of exporting the model and weight values of a CNN. Once we were done collecting and organizing the datasets, the work distribution was made so that we have 2 people working on the neural net and three people working on miscellaneous software (see "Member Collaboration" section for more details). Our entire software package was stored in a private Github repository, organized by sections, along with documentation and software setup instructions. Github naturally managed everyone's code versions so that we would not have any conflicts when working at the same time. As noted in the collaboration section, we were always open to chat on Discord about shifting any efforts or helping each other out with our task.

Installing development tools and acquiring data

Many of the development tools have been already installed on some of our systems, thanks to other computer science courses such as CMSC471 (Introduction to Artificial Intelligence) and CMSC478 (Introduction to Machine Learning). These development tools consisted of Jupyter Notebooks, TensorFlow, Scikit learn, and opencv (Python) for computer vision applications. All tools and packages were installed through Anaconda, a data science platform and Python's package management system - pip. Many of the datasets were found in Kaggle, University of Toronto, and YouTube links during the collaborative team research described in the team work section. For the web API, we used NodeJS on express to manage the server routing and Postman for testing the RESTful calls.

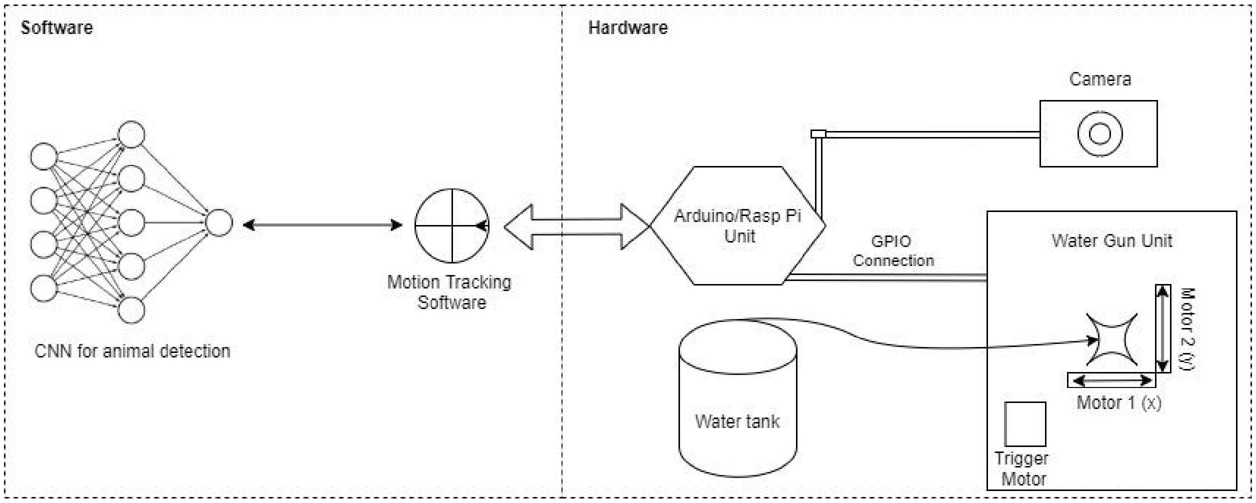
Literature and writing tasks

The majority of the writing tasks were done distributively and reviewed and revised by team members Ching Zheng and Jackson Lim. The sections relevant to the team member's task would write the bulk of the content and consolidated by the "editors" of the team. Then before submitting an assignment, we would meet up and go over the final edit to make sure that everything was documented properly.

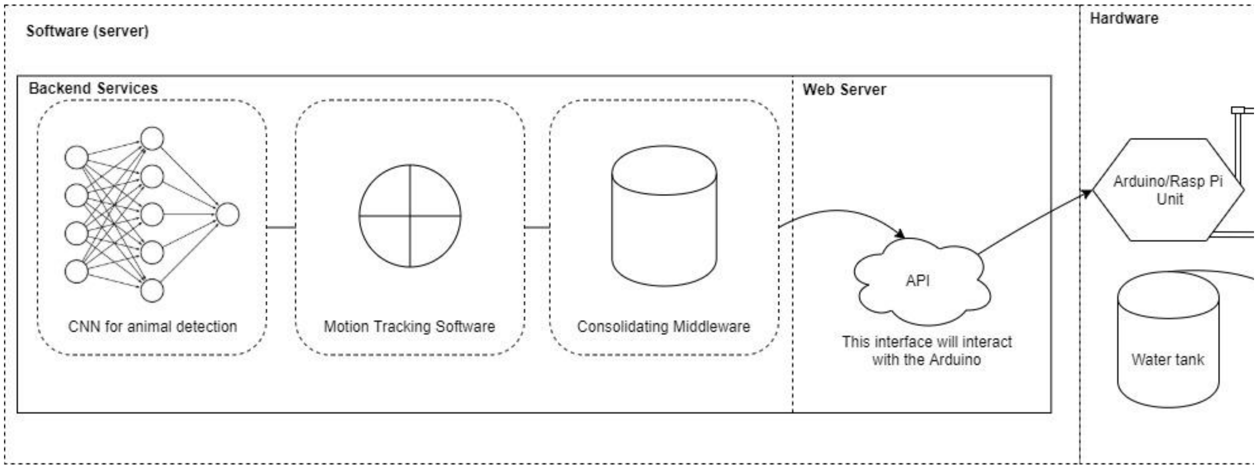
Change of Plans

Initial Plan

The initial design for Garden Guardian involved a hybrid of software and hardware work. The backbone of the project consisted of a convolutional neural network for animal detection and an Arudino unit, controlling a water gun using two motors (each motor controlling x and y coordinates to aim). The diagram below shows the outline of the initial project:

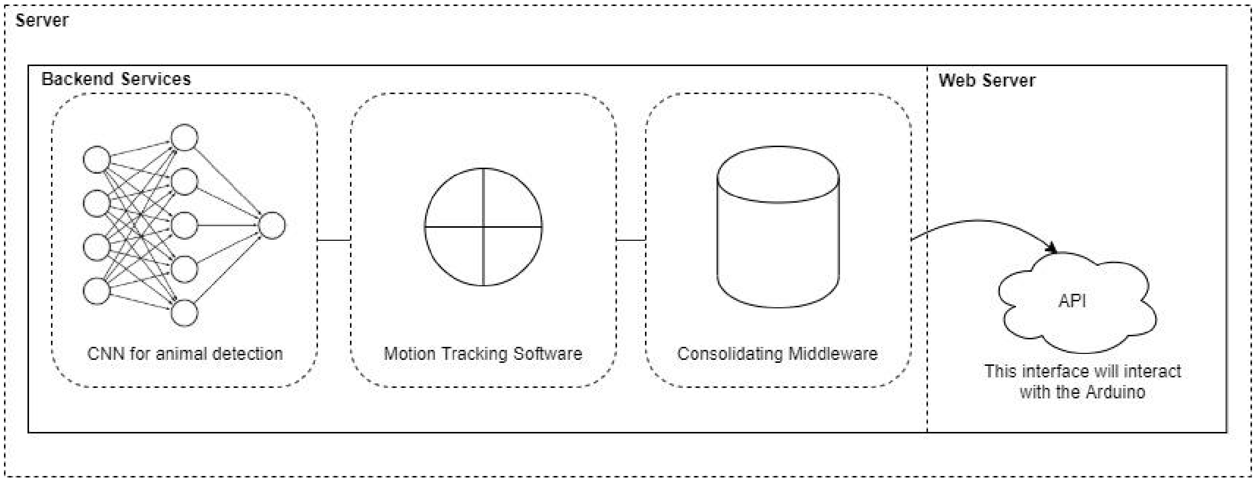


As the semester passed and we did more research, we realized that an Arduino alone does not support the use of opencv and even the use of Python is very limited. So we needed a server to receive and analyze the data and send the result back to the Arduino so it can control the hardware correspondingly. The full scale of the project now expanded to the diagram shown below:



This is a lot of work, and simultaneously due to the COVID19 quarantine, we were unable to meet in person to work on and interact with any of the hardware part of the project. We needed to scale down the project so that it would match the timeline for the rest of the semester and suit our quarantine situation.

New Plan



The project was scaled down only to the software portion of the product, which adds on the consolidating web API from the old plan's software portion. The final product will be this package, available to deploy and use to complement the hardware portion we unfortunately won't be working on this semester. If the project was to be extended, this API will be an end-point for the Arduino to interact with and it will deal with all the image computations necessary. The following list briefly describes the scope of the new plan:

- CNN for animal detection - this portion remains the same
- Motion Tracking Software - this portion remains the same
- Consolidating Middleware - this portion consists of extra bits of code to link the CNN, Motion Tracking Software, and the Web API, and other self implemented utility code
- Web API - this portion is an interface using a RESTful API implementation. This should easily be interfaceable with clients such as Postman or even web browsers

Member Contribution

Group Member	Contribution
Jackson Lim	Web API/Middleware (primary), Documentation/Editing
Nimish Kapoor	Motion Tracking Software (primary)
Henry Nguyen	Convolutional Neural Network (primary)
Xiang Chen	Convolutional Neural Network (primary)
Ching Zheng	Documentation/Editing (primary), Convolutional Neural Network

**primary indicates that majority of the group member's efforts were spent doing that task*

NOTE: as described above, everybody was involved in writing their own sections for the paper for their corresponding contribution.

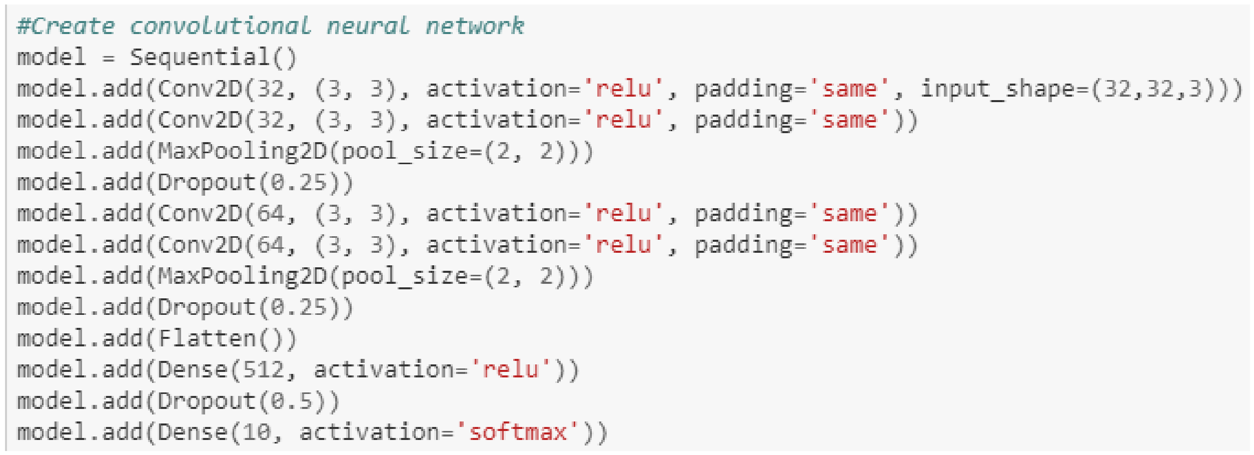
Initial Progress

CNN for animals

For the project we plan to first use a convolutional neural network (CNN) to train images sets to differentiate between animals. We came with a list of animals that we considered to be "pests" for a garden, the following animals are: deers, birds, raccoons, squirrels, and rabbits. The images sets are separate so we plan on combining them into a singular image set and essentially use the neural network for muti-class classification. Our initial goal is to simply have a program that can classify our test image sets with a fairly high accuracy. And once we have achieved that accuracy, we want to be able to use object detection to focus on the animal and have it identified on a live feed.

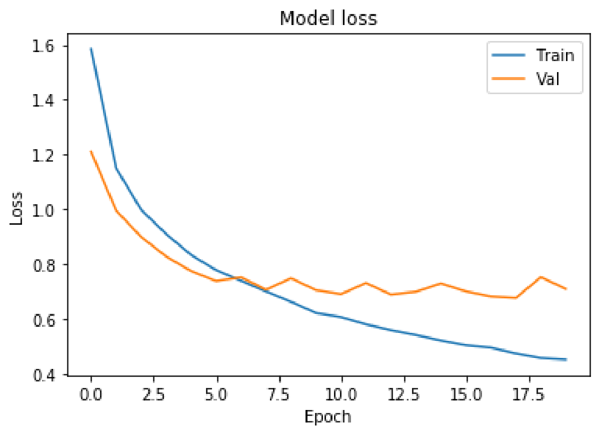
To begin the project we selected the CIFAR-10 dataset which contains 60000 images divided into 10 categories, meaning there are 6000 images for each category. Of the 60000 images, 50000 are used for training and the remaining 10000 are used for testing. The 10 categories of the image set are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Not all images within the image set pertains to our project; however, for now we are just developing our neural network and testing its accuracy on image classification. We will replace the CIFAR-10 imageset with our own set in the future.

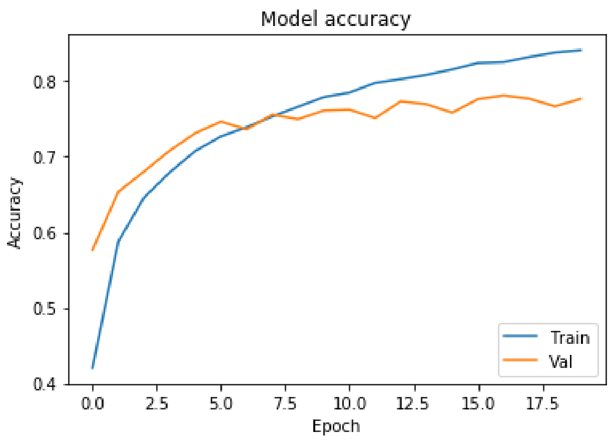
As stated in our management plan we are using Jupyter Notebook as our main developmental tool; thus, after loading in CIFAR-10 into a Jupyter workbook we started to create the CNN. For the neural network six dense layers were used with the first five layers using the relu activation function and the last layer using the softmax activation function to output the probability that an image falls into a particular category.



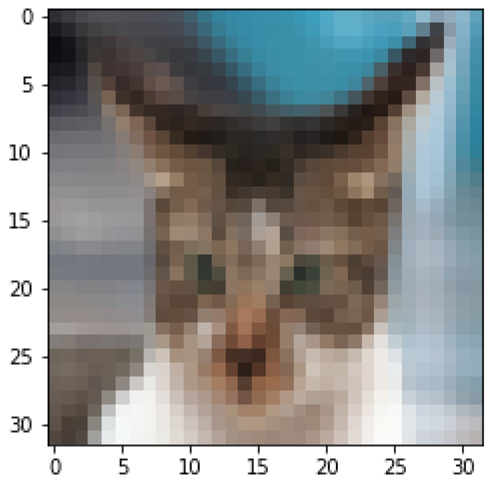
```
#Create convolutional neural network
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32,32,3)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

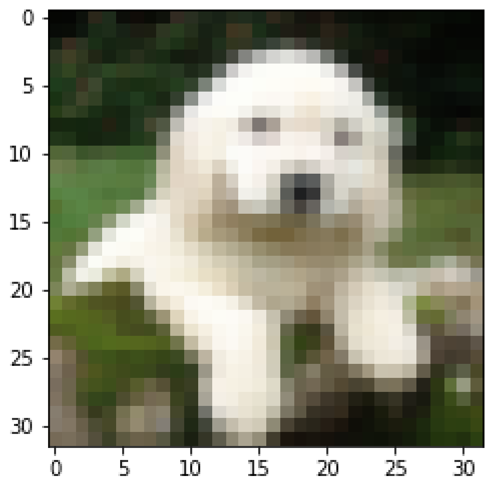
After creating the CNN, the training batch is then fitted onto a model utilizing the categorical cross entropy, the adam optimizer, and the accuracy metric. The model is set to run for 20 epochs and validation split is set to 0.2 meaning that the training batch of size 50000 images is split into 40000 images used for training and 10000 used for testing. Once the training process was completed, we plotted the loss (loss referring to bad predictions) against the epoch and we also plotted the accuracy against the epochs.





From the model loss plot we can see that as the model continues to train it makes less mistakes on its predictions, and from the second plot we can see that the model becomes more accurate in its prediction as the training progresses. The objective of our CNN implementation is to obtain an acceptable accuracy metric for our test cases. For our test cases, we uploaded two images, one of a cat and the other a dog into our CNN. Below are the results:





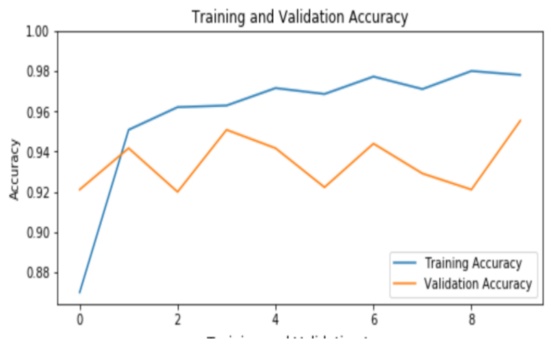
```
Most likely class: cat -- Probability: 0.98304635
Second most likely class: dog -- Probability: 0.015086506
Third most likely class: deer -- Probability: 0.001473558
```

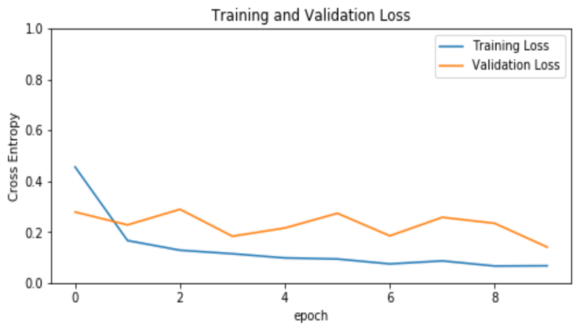
```
Most likely class: dog -- Probability: 0.99999726
Second most likely class: bird -- Probability: 1.9462584e-06
Third most likely class: horse -- Probability: 8.778759e-07
```

Given that the cat image achieved an accuracy rate of 98.3% (.983) and the dog image achieved an accuracy of 99.9% (0.999), we concluded that our CNN was successfully implemented.

After we tested the CNN using CIFAR-10, we began building our own dataset that will contain the list of animals that serve the purpose of this project. We searched through different datasets available online and only extracted the ones we need. Our current dataset consists of seven classes (dog, cat, bird, deer, rabbit, raccoon, and squirrel), each with about ~1000 images. We applied the similar CNN architecture and steps described above to the dataset, however, the accuracy rate dropped to only about 70%. We did try some parameter tuning methods to improve the accuracy, such as changing the optimizer, loss, and modifying the layers. Unfortunately, none of them really improved the result. Then, we decided to implement transfer learning by using a pre-trained network, instead of making our own one. Since a pre-trained model is a saved network that was previously trained on a large dataset, it will effectively serve as a generic model for many image classification problems.

The pre-trained model we selected is MobileNet V2. It's a model developed at Google, and trained on a large dataset consisting of 1.4M images with 1000 classes. The first step was to exclude the top layers of the model, because they are usually specialized for a particular dataset. The bottom layers are more generalized for detecting features from image. In addition, we also freeze the convolutional base to prevent the weights in layers from being updated during training. Basically, the pre-trained model will act like a feature extraction. At last, we added an extra layer on top of it and only trained the top-level layer.

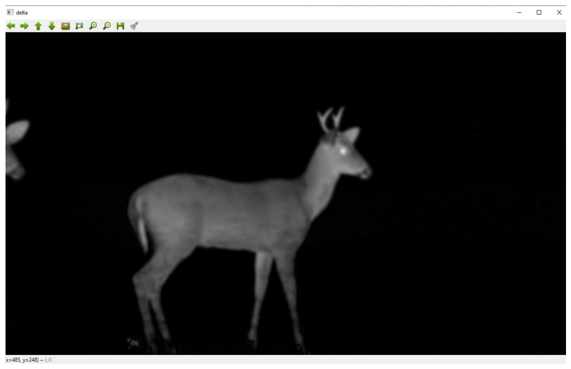


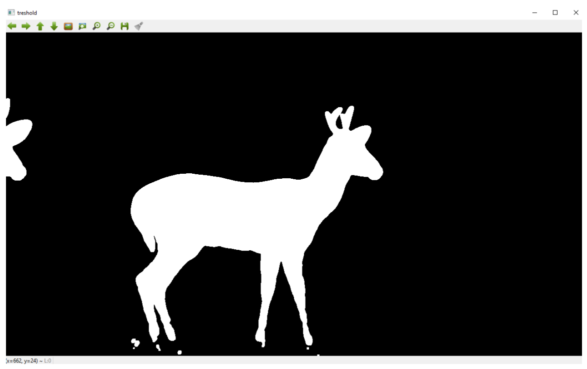


Given our image dataset, it achieved an accuracy rate of 95%, which is a fairly good result compared to using the CNN created by ourselves. From the accuracy and loss plots, we can see there isn't much improvement after a few epochs. Despite no improvement being made, the accuracy is still in an acceptable range for this project. Thus, we didn't do further tunings for this model.

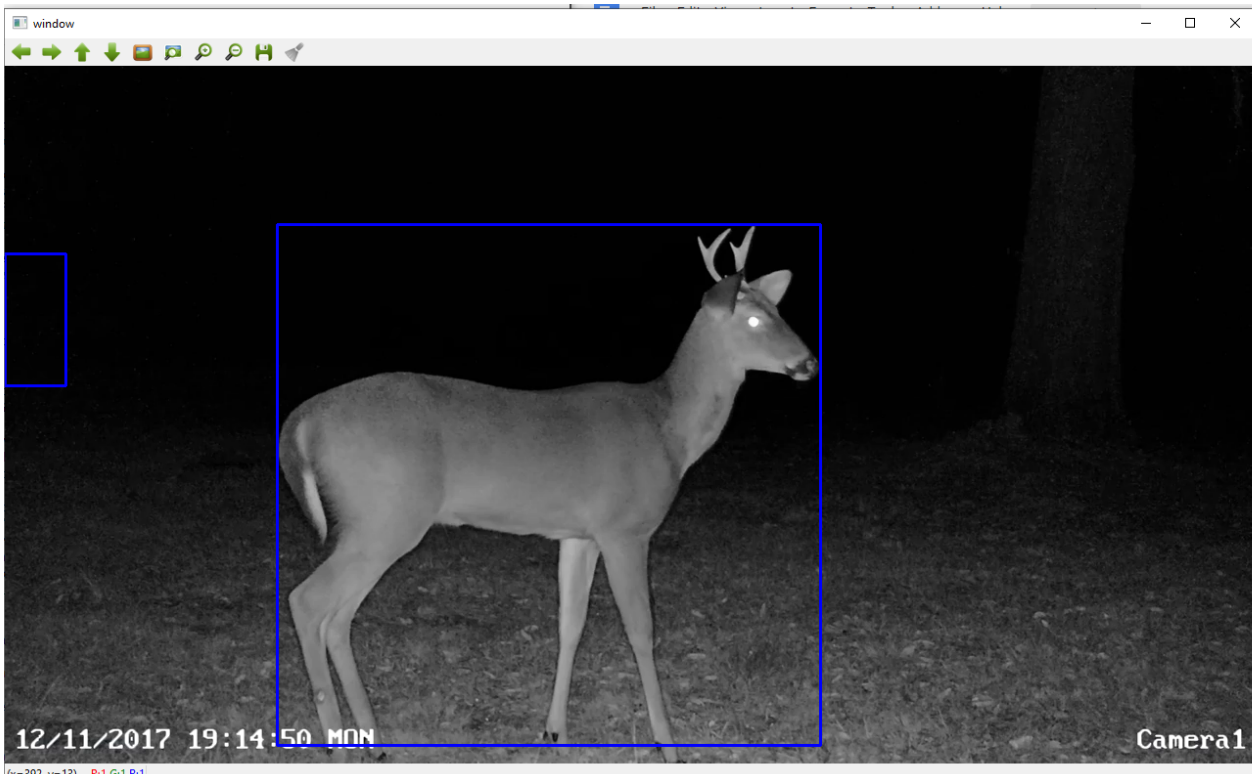
Motion Tracker

In addition to the CNN we have implemented a seperate program to detect motion in either a live video or a pre-recorded one. The purpose of the motion tracker is to detect motion in the garden and then send the detected object to the CNN to determine if the object is a pest or not. The motion tracker works by using the first frame and formatting it to be used as a template to be used as comparison for the successive frames. Once the delta frame has been established we used cv2.threshold() to limit the amount of information and to make sure that small changes in light and shadows don't get viewed as objects in motion. This creates the final frame that will be used to determine motion. An example of the first frame and its details removed are seen below:





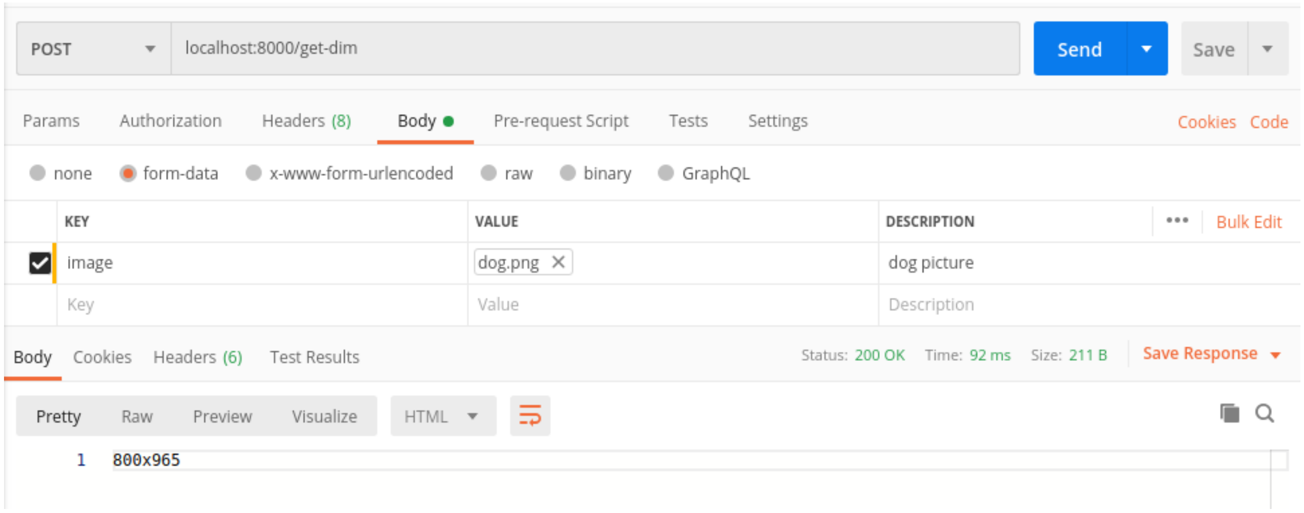
Then we find the contours in the threshold frame and if the area of the contour is above a predefined level we can consider the object a large enough change to determine motion. Finally we draw a box around the object to track its movements.



Some current issues with the system which are currently being worked on: firstly, if there is an object in the template frame that moves its absence is considered an object as it appears as a major change in the delta. Along with this if there is a major change in lighting such as turning on a light or the transition from day to night or vice versa. The plan to combat these issues is to maybe refresh the template frame after a certain amount but this solution has other complications as well. Once the motion detection and object detection aspects of the implementation are working correctly, we plan to combine this software to our CNN implementation. Our plan is to use the initial frame from the live captured feed and use that as a test case for our CNN. After our CNN has identified the animal, the application will continue to track the animal by highlighting the area.

Web server

The general idea for the web server portion is to be able use a HTTP client (such as a web browser, or in this case - Postman) and send an image to the server. Then the server will save the image to the file systems. It then calls an executable with the file path, which then processes that image and sends the output to stdout. Finally, the server will return that result to the client as a HTTP response. Since the image processing service (CNN and Motion Tracking) are not ready for deployment, we used a simple opencv executable that returns the width and height dimension of a given path to an image (see “get-dim.cpp”). A demonstration of get-dim is shown below with the dog example we used in class.



```
localhost:8000/get-dim
```

KEY	VALUE	DESCRIPTION
✓ image	dog.png	dog picture
Key	Value	Description

Status: 200 OK Time: 92 ms Size: 211 B

```
800x965
```

We expect that there will be further complications as we add the two services such as response delays caused by computational time and more services will be added, with consideration of utilizing this API with an actual Arduino in the future.