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

Abstract

The Garden Guardian is a project to develop an application that will detect and identify mammalian pests and mitigate their presence in a home garden. The dangers to a home garden are not insectoid pests, but instead mammalian pests such as: rabbits, deers, raccoons, or potentially even your pets. Thus, the main focus of our application is to use live video feed to identify whether an animal is a pest, and if an organism is identified as a pest, our application will take defensive measures against it. Now the question, why should potential clients use our product? True, there are alternatives to our solution, ranging from spraying chemicals and erecting a protective structure around the perimeter. However, these solutions are either short-termed or labor-intensive/expensive. We hope that our project will be a superior approach, because the applications and devices are built for long-term use and therefore, the expenses are kept to a minimum. Thus, by applying our knowledge of machine learning, computer vision, and software development into a singular product we will be able to effectively protect everyone's produce.

Introduction

This project the aggregation of our knowledge from machine learning and computer vision. While projects combine these two fields of computer science as common these days, they tend to require large investments of cutting-edge technology, time, and funds. Thus, we wanted to find an idea more suitable to our scale, something niche, we thought of the one word: gardening. Much like how everything in the world has the potential to be automated by computers, even the task of gardening can be automated. Now our project will not plant, water, and harvest your produce for you, but will do our best to protect it. Relevant to many gardeners at home, and potentially new gardeners who feel the need to start after the quarantine due to the recent coronavirus, the product may help preserve other's efforts by saving their crops. As stated in the introduction sentence, the primary goal of this project is to apply our knowledge and bring our idea into fruition; perhaps this project can become a part of our everyday lives.

Related Work

- Chen, K. (1993, November 02). US5258899A - Motion sensor lighting control. Retrieved May 08, 2020, from https://patents.google.com/patent/US5258899
    - We have all seen these types of lights. Many of us may even have them but these lights work through rader and do not care for the subject and aren't likely to scare any pests. Our project on the other hand will only affect pests and is much more likely to produce results.

- Surma, Greg. (2019 Jan 17). “Image Classifier - Cats vs Dogs, Retrieved from https://towardsdatascience.com/image-classifier-cats-vs-dogs-with-convolutional-neural-networks-cnns-and-google-colabs-4e9af21ae7a8
    - Our project will work with a web server to lessen the stress put on the end users hardware.
- Rosebrock, Adrian. (2017, Sept 11). “Object detection with deep learning and OpenCV.” Retrieved from https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/
    - We used this as reference when implementing our own CNN.
- Baimuratov, Gaiar. (2020, Feb 15). “Detecting Animals in the Backyard - Practical Application of Deep Learning.” Retrieved from towardsdatascience.com/detecting-animals-in-the-backyard-practical-application-of-deep-learning-c030d3263ba8.
    - A project that we want to use as comparison for our end results
- (n.d.). RESTful Web Services Tutorial with Example. Retrieved May 08, 2020, from https://www.guru99.com/restful-web-services.html
    - We will be using the same technology, but with NodeJS and our own features.
- Sandoval, K. (2019, June 28). REST vs Streaming APIs: How They Differ: Nordic APIs |. Retrieved May 08, 2020, from https://nordicapis.com/rest-vs-streaming-apis-how-they-differ/
    - This will be used for investigating how we will transition into streaming from RESTful and applied into our implementation

Methodology

As stated in our initial project report, we used Jupyter Notebook as our main developmental tool, and we continued to develop the majority of our project in that environment. We successfully created our own data set, and replaced the CIFAR-10 image dataset that we initially used to train our CNN. In addition, for the CNN model, we had decided to use the MobileNet V2 as our base model. It would be more efficient to utilize a pre-trained model, instead of building our own and going through all the fine-tuning steps. Our old model was only able to achieve an accuracy rate of 70%. Other environments that we are using, include Pycharm to develop the motion and object detection portion of the project, because it is better suited for the various libraries we’re using. We have planned and successfully transitioned all of our code from Pycharm to Jupyter, so that we can compile and test our application as a whole.

We have tested the accuracy of the CNN with our new dataset, and it was to our satisfaction. Yet, the accuracy of our trials means nothing if it does not perform well on the frames we pass from the live video feed. We would of course like to maintain our accuracy rate of 0.90 and above from our test cases, but we also understand that images/frames from videos

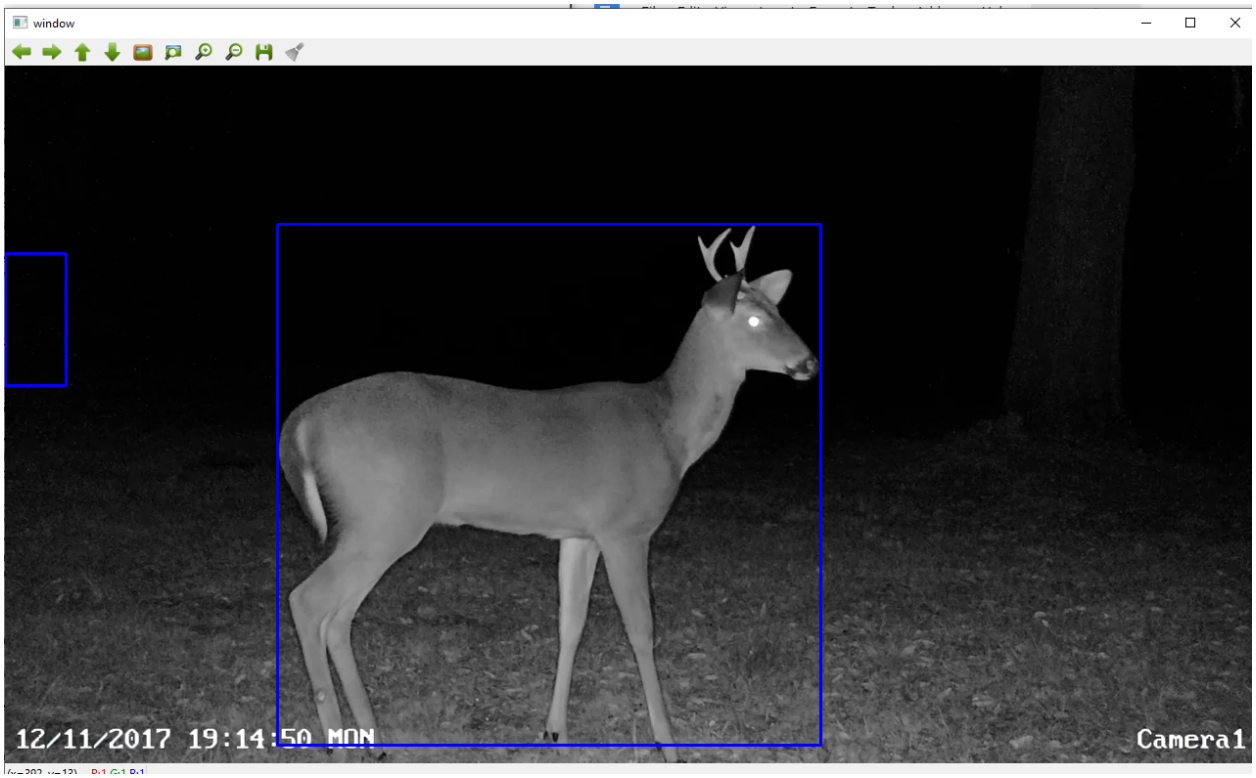
may not always be clear, especially if the subject is at a distance away from the camera. Thus, we decided that an acceptable accuracy rate for the animal classification in the frames is 0.70, of course anything higher would be gladly accepted. However, if the resulting accuracy is not satisfiable, we will fine-tune our CNN hyperparameter to improve the result. Possible ways are like modifying the layers (number of hidden layers, activation function, dropout, etc), changing the optimizer and loss function, and image preprocessing. By this stage of the project, we would prefer to not make any significant changes, but we can still consider using a pre-trained CNN as the base model. The reason why we need a high accuracy rate from the image classification is because our application identifies pests, and if the CNN misidentifies an animal the garden would be in danger or the wrong animal would be identified as a pest.

The object detection aspect of the project was also improved so that once a motion was detected, the software will highlight the subject, and avoid anything else in the environment that is not moving. Additional improvements include, the program will only focus on the highlighted portion of the image, meaning that anything outside of the region will be cropped off. The reason for this is so that once we sent the frame to our CNN, the network would be able to detect the animal more easily, which would lead to a higher classification accuracy. In order to test our application, we have found various videos of animals on the internet and we will be feeding that into our application to test the accuracies of various animals. Then once those tests are completed, we do further testing by using webcams to detect group member pets in real time.

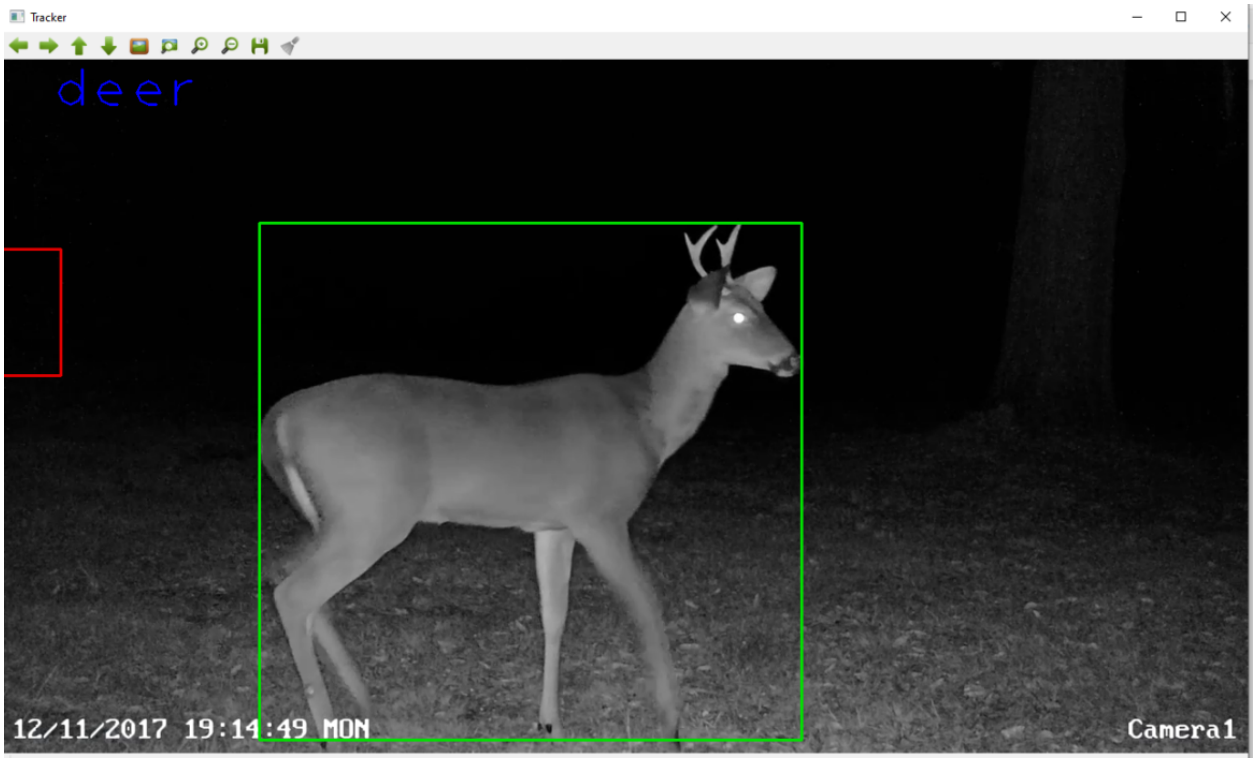
If the CNN identifies the animal as an animal that falls under the pest category the protective measures will activate. Since we do not plan to implement the hardware aspect of this project, a web API will take its place in the end point and send mocked command data to the client-side. The web API will be built on express, a NodeJS package typically used as a web-server. We plan on adding image streaming to the server's end-points and feed it into the internal image processing software.

Results

When testing the CNN and object detection program together, we first converted the CNN file into a .h5 to be loaded into the obj_tracking.ipynb file. Once loaded in, we loaded a short video into the obj_tracking file and took the first frame of that video and formatted it to be used as a template to be used as comparison for the successive frames, this frame is known as the delta frame. The delta frame is converted into grayscale and we used gaussian blur to remove any details from the background. Then for the rest of the video, we essentially compare every frame of the video against the delta frame, with the function contourArea() and if any change is detected, the object is highlighted by a box. An example could be seen below.



After the motion and object are detected from the frame, the frame is resized to only focus on the object within the highlighted area. Once the frame is resized and cropped, it's passed into the CNN model to have the animal in the image classified. The CNN model then returns an accuracy measure of the likelihood that the object in the image falls into any of the seven categories of animals we trained the CNN with. The seven species of animals that our CNN recognizes are: birds, cats, deers, dogs, rabbits, raccoons, and squirrels. Once the CNN results with the results the highlighted box will change its color to green to signify that the identification process has been completed. The video will also output the name of the animal that it believes the animal to be on the top left corner of the frame. However, we do run into some issues with the frame rates when the CNN is running for every single frame of the video, and we do intend to solve this issue if possible, but the application is functional and we are satisfied with the results.



The last function that the program does is output the probabilities of each of the frames sent to the CNN to be classified. And as you can see below, the results can be a bit more accurate given that in the first few frames, the deer was classified as a cat. But, it quickly adjusted and reclassified the object as a deer for the remaining duration of the video.

```
Most likely class: cat -- Probability: 0.9327883
Most likely class: cat -- Probability: 0.9815688
Most likely class: cat -- Probability: 0.96701247
Most likely class: cat -- Probability: 0.95582
Most likely class: deer -- Probability: 0.9157627
Most likely class: deer -- Probability: 0.94777185
Most likely class: deer -- Probability: 0.98153645
Most likely class: deer -- Probability: 0.99555343
Most likely class: deer -- Probability: 0.9966378
Most likely class: deer -- Probability: 0.9925483
Most likely class: deer -- Probability: 0.97736025
Most likely class: deer -- Probability: 0.9609883
Most likely class: deer -- Probability: 0.97514886
Most likely class: deer -- Probability: 0.9526301
Most likely class: deer -- Probability: 0.9859869
Most likely class: deer -- Probability: 0.9937551
Most likely class: deer -- Probability: 0.9925034
Most likely class: deer -- Probability: 0.9800511
Most likely class: deer -- Probability: 0.9533101
Most likely class: deer -- Probability: 0.9568546
Most likely class: deer -- Probability: 0.9703935
Most likely class: deer -- Probability: 0.9495926
Most likely class: deer -- Probability: 0.94441134
Most likely class: deer -- Probability: 0.9432843
Most likely class: deer -- Probability: 0.9483292
Most likely class: deer -- Probability: 0.9517114
Most likely class: deer -- Probability: 0.9543497
```

Plan for Completion:

The demo from our results shows that applying the CNN is working successfully and is pairing well with our object tracking software. However, there are a few key components that need to be completed, as well as a few minor issues we can fix to improve the results of our project. The major missing component is the web API. Because of our traditional familiarity and usage of the HTTP, using RESTful methods for our API seemed like a convincing choice. Using such methods, we would take snapshots of an image and make *static* comparison from one image to another. However, since our object tracking software needs *continuous* streams of frames to compare against the initial frame of the video footage for tracking, making 24 synchronous requests per second (if we were to seek footage at a *low* 12 fps) could not be done because it was not code compatible, neither was it a effective implementation plan due to network latencies. So to complete this project, we will be changing to a streaming API request, sending continuous image data from the client to the server, along with asynchronous responses from server-side, handled by an event handler on the client-side. Switching to this architecture will be attempted with a goal of successful implementation by the project final submission.

Another minor issue was in the lag induced by our object tracking. Sometimes the lag is minor, other times it can be a major delay, as our algorithm executes on each frame of the video footage. There needs to be further investigation of handling potential frame lag when we switch from feeding pre-recorded videos to the tracker, to streaming live data. We plan to handle that issue if it comes across our implementation process. We will continue to adjust different hyperparameters of our neural network in hopes of yielding more effective results. When these issues are resolved, we hope to launch this program as a packaged application to a server so that it can be accessed remotely. Finally, along with the web server work, the major goal to accomplish is testing - testing on more prerecorded video feeds, live footage via webcam, and any other sources of image streaming we can find that suits the detection and tracking of our targets. We will continue working within our assigned segments of the software package in hopes of fulfilling our mission statement and completing our project goals.