The Garden Guardians

Ching Zheng Nimish Kapoor Xiang Chen Henry Nguyen Jackson Lim
University of Maryland Baltimore County
CMSC 491/691 Final Report
Spring 2020, Group 4

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 send an alert to the garden owner so that they can take action to protect their garden. 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.

1. INTRODUCTION

This project is the aggregation of our knowledge from machine learning and computer vision. While projects combining these two fields of computer science are 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 we 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.

2. RELATED WORKS

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.

- This project is very similar to ours that it’s about installing a camera in the background to detect animals. The difference is that it’s only trained on three general data classes: person, animal, and car. Also, we want to use it as a comparison for our end results.

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. It’s like a simple tutorial for how to implement object detection in opencv. It uses the MobileNets and Single Shot Detector.

Chen, K. (1993, November 02). *US5258899A - Motion sensors lighting control*. Retrieved from https://patents.google.com/patent/US5258899

- The inspiration for our project these lights are very common and are used to ward of pests and intruders but regardless of what they detect and and are not actually very effective in stopping animals

Norouzzadeh, Mohammad Sadegh, et al. “Automatically Identifying, Counting, and Describing Wild Animals in Camera-Trap Images with Deep Learning.” *PNAS*, National Academy of Sciences, 19 June 2018, www.pnas.org/content/115/25/E5716/.

- This project is aimed at using motion-sensor cameras to track animals in the wild and record their behaviors. The differences are that it’s counting the number of the same species and also assigning attributes to them, such as if they’re standing, eating, having babies, and etc. This project is certainly a lot more complicated than ours.

Milosevic, D. (2019, November 19). Working with Node.js Stream API. Retrieved May 21, 2020, from https://medium.com/florence-development/working-with-node-js-stream-api-60c12437a1be

- This project demonstrates the usage of Node.js Streaming. Parts of this project was used as examples of how to use the functions and how they could be built together to work as a streaming API.

3. METHODOLOGY

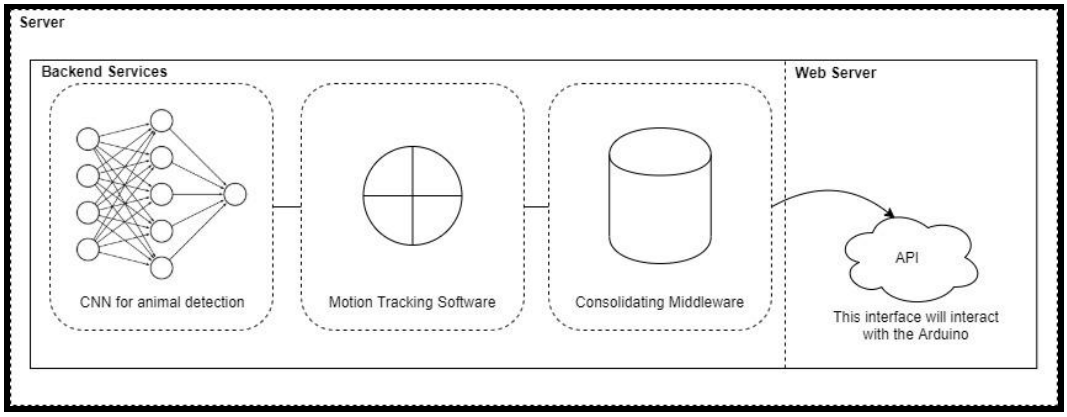

Figure 1. Flow diagram showing how all each individual components of our program will collaborate as a single application.

As the flow diagram in Figure 1 shows, the convolutional neural network (CNN) will be developed first, then the motion tracking software, followed by the webserver. We used Jupyter Notebook as our main developmental tool, and Pycharm as our secondary developmental tool. Jupyter was used to develop our CNN so that we could train the program to detect animals from images. We first started off by using a pre-made image set, named CIFAR-10, this image set contains 60,000 images divided into ten categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Not all images within the imageset pertained to our project; however, for the initial steps we just wanted to develop a functional CNN. Once the CNN was functional, the CIFAR-10 imageset was replaced with an image set created by ourselves. The second dataset contains only animals we deemed as pests, more details could be seen in the following table:

Category of Animals	**Number of Images**
Squirrel	1862
Rabbit	731
Racoon	724
Bird	1334
Deer	1499
Cat	1300
Dog	1300

Figure 2. Table showing the categories of animals and number of images in each category. The splitting ratio for train/validation/test dataset is 0.7/0.1/0.2

**Most images are collected from various datasets found on Kaggle.com*

Alongside a new imageset, which contains a total of 8750 images, we had also decided to use the MobileNet V2 as our base model. Thus, instead of building our own network and going through all the fine-tuning steps, we would be utilizing a pre-trained model. MobileNet V2 is a light weight model developed at Google and trained on ImageNet dataset that consists of 1.4 million images and 1,000 classes (Transfer Learning). Because it's already well trained on a large and general enough dataset, it can effectively serve as a generic model for our problem. In order to use this pre-trained model, we had to remove the top layers of the model because they are usually concerned with high-level features to classify data and thus more specialized for a particular dataset. Whereas the bottom layers are generalized to detect low level features from an image. In addition, we freezed the convolutional base to prevent the weights from being updated during the training. Basically, it will act like a feature extractor. Even though we could enable weight-update, it will only likely to increase the chance of overfitting since we might be expanding our data classes in the future. At last, we added two layers on top of the base model, one is the globalAveragePooing to reduce the number of parameters and the other is softmax to generate predictions. Our previous model was only able to achieve an accuracy rate of 70%, while the current model was getting 90% and above. 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.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Model) (None, 7, 7, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 7)                 8967
=================================================================
Total params: 2,266,951
Trainable params: 8,967
Non-trainable params: 2,257,984
_________________________________________________________________
```

Figure 3. Table showing the structure of our CNN

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.

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.

If the CNN identifies the animal as an animal that falls under the pest category the alert system 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.

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 deemed as pests (refer to *Figure 2*). Thus, the overall idea of our project is to develop a CNN that will take in the frames from a video feed and use those frames to identify the animals captured in that frame. The CNN and object/motion detection algorithm will be stored on a remote server, and the camera essentially streams to the server, so that the frames of the video feed can be processed.

The server wrapping portion has shifted with trial and error. The initial plan was to use a RESTful API to save each image and forward it to our object tracking software. However, forwarding image by image was not a compatible option for our object tracking software. We have shifted over to streaming data to the tracker using file system read and write streams. The server reads streams and holds sessions with the tracker, forwarding all the stream data for the

tracker to detect and interpret. The server then responds with the appropriate detection response to the client side.

4. RESULTS

When developing the CNN we needed to set a goal for how high our training accuracy and validation accuracy should be in order for our CNN to be considered successful. If a model fails to reach that goal, we would then have to return to the CNN and adjust our hyperparameters accordingly until that accuracy rate is reached. We initially set a goal of at least 0.70 for both the training and validation accuracy. Once we set our goal we ran and fine-tuned our model until an accuracy of 0.70+ was achieved. The model in *Figure 2* was used to run for 10 epochs. *Figure 4* shows the results of the training.

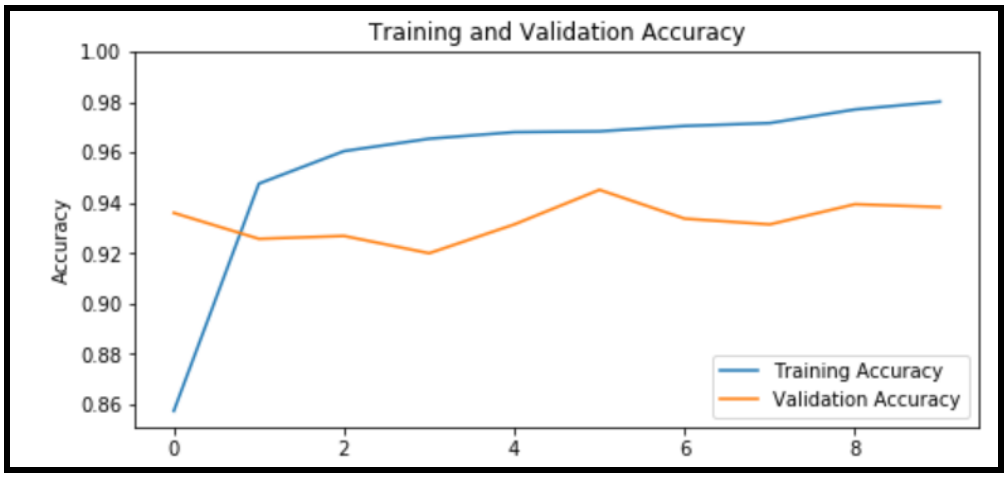

Figure 4. Display training accuracy and validation accuracy against epochs

According to the graph, at epoch 0 the training accuracy is already 0.86 while the validation accuracy is 0.94. The accuracies are maintained through the training process, as after epoch 0 neither metric dipped below 0.92, with training peaking at 0.98 and validation peaking at 0.95. Then with the testing set, the model achieved an accuracy rate of 0.94. These values exceeded our initial goal of 0.70 accuracy; thus, we considered our CNN implementation to be successful. After completing the CNN aspect of the application, we move on to the motion detection/object detection component of the project.

```
55/55 [==============================] - 47s 856ms/step - loss: 0.2174 - accuracy: 0.9366

[0.21742934038523923, 0.9365714]
```

Figure 5. Display training accuracy and validation accuracy against epochs

For the object detection and motion detection software, like the methodology section mentioned, we downloaded a pre-recorded video from online and loaded it into our detection program. As you can see from the image below, the frame of the video successfully detected the deer and the area is highlighted in a blue box.

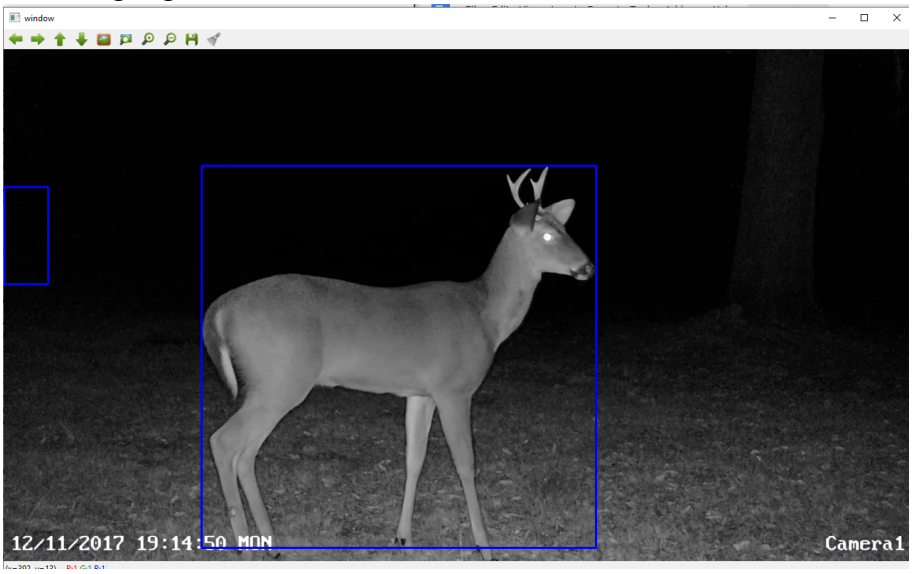

Figure 6. An example of a frame with an animal being highlighted through the use of motion detection and object detection

Once the animal in the image is detected, the frame is resized and cropped, then it is transferred to the CNN to be 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. 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.

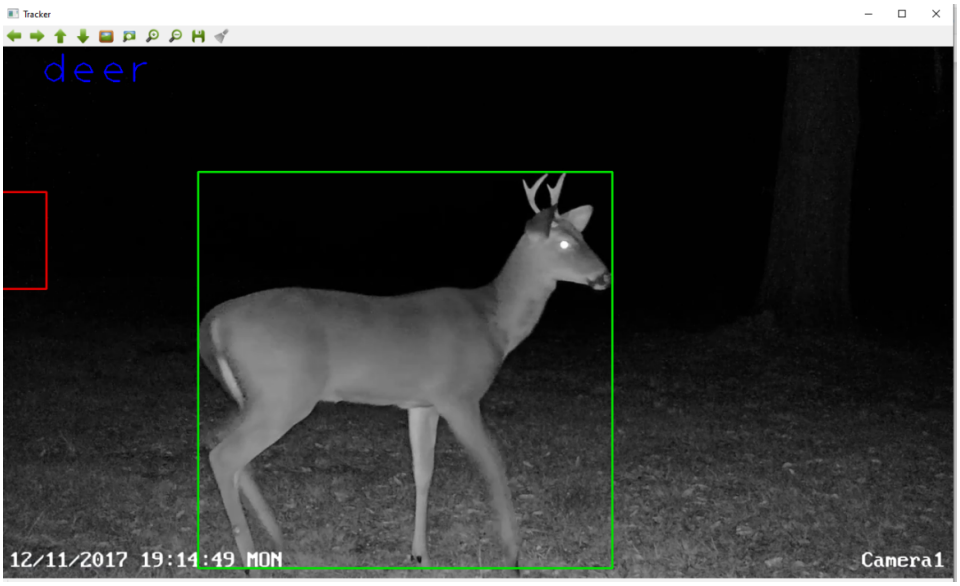

Figure 7. An example of a frame with the animal identified (in this case a deer) as denoted by the green colored box, indicating that the animal is identified.

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
```

Figure 8. An example of what the CNN outputs when classifying the frames imported from the video.

These of course are the best case scenario, not all of our tests were accurate as mentioned before, but we also have some conducted tests that do not behave as intended. For example, in another test, we have a basic video with one animal that stands still for a majority of the time, with a poorer picture quality. Then we tried another video to test how our application behaves when we have animals that do not belong to any of the pest categories that we trained our CNN with. As you can see from the results below, with *Figure 9* the software detected the animal within the image; however, it did not successfully identify the animal. In this case the image of the animal (a possum) is rather poor, and possum was not in part of the seven categories of pests. but the program still should have attempted to identify the animal. With Figure 10, the bear is pretty obvious to us humans, but since the program was not trained to identify one it cannot tell that the object in the animal is a bear, it made a wrong prediction. In the same example, you can see that part of the image was also highlighted on the lower left, but that highlighted area is

empty. Furthermore, since there are more than one predictions being made on the same image the name on the top-left corner overlapped, making it difficult to read.

Figure 9. Another night time video showing how an animal (possum) that we didn't train with on the CNN is treated.

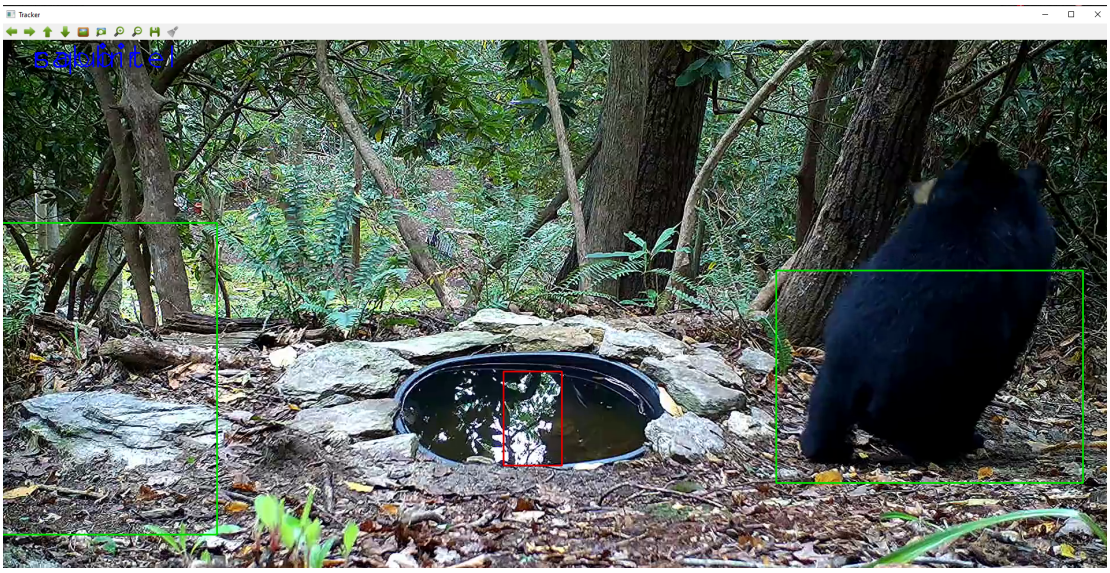

Figure 10. We can see a bear and an empty space being identified as a squirrel and a rabbit and we can see that with 2 identified objects the labeling in the upper left hand corner is displaying on top of itself and is hard to read.

Another issue that plagues the application is that when used on a video feed, the video suffers from frame rate issues due to the fact we are sending over a significant portion of the video's frames to be processed by the CNN. This doesn't affect the output version of the test video; however, this does render using live-feed video nearly impossible due to the low frame rates.

The final streaming API section is relatively simple. Our initial approach with a REST API was not suitable for video use, so we have shifted the API scheme to streaming. Since the

web server just has to receive and forward the stream, the results will demonstrate the forwarding of a video to OpenCV. OpenCV supports streaming video capture from a URL source, which in this case is “localhost” at port 8000. The results show a file stream of the deer tracking example shown in one of our results.

5. CONCLUSION

Our goals for the project, as stated in the introduction, is to develop an application that will take video feed and identify the animals within the video frame. If the animal in the image falls under one of our pest categories, it will be identified as such and the user will be alerted. The purpose of all of this is to protect gardens from mammalian pests: squirrel, rabbit, raccoon, bird, deer, cat, and dog. Now as for our results, the results can be considered somewhat successful. The methodology used to create our program was pretty onpoint, as individually the CNN worked fine on its own and the motion detection worked fine on the most part. We developed a CNN that can classify images at an exceptionally high accuracy rate, and our motion detection software functions albeit there are bugs to work out. The CNN and motion detection can function together as seen with *Figure 7*, and the object detection can function correctly given the right conditions. Our software is uploaded to a server where the animal detection and classification can be done remotely.

6. DISCUSSION

As you can conclude by now, the project did not completely achieve its goals. There are many cases of failures as seen with *Figure 9* and *Figure 10*. Of course in both of these cases, the CNN only works correctly if we use images of animals that we learned the CNN on, so any animals outside of that group will give us wrong classifications. The frame rate issues were never resolved and we also never implemented our alert system due to the lack of time and poor time management, so is also a failure of the project. Perhaps we could have trained more categories of images, and the user could decide on which categories they want to be pests. With this method, we can cover more classifications and users can avoid having their pets identified as pets. Humans would of course be included, so that users don’t get alerted when they’re in the garden themselves. In order to resolve the frame issue, we can send less frames to the CNN so that the program doesn’t slow down as much due to every frame being processed by the neural network.

REFERENCES

Baimuratov, Gaiar. (2020, February 15). *Detecting Animals in the Backyard - Practical Application of Deep Learning* . Retrieved from https://towardsdatascience.com/detecting-animals-in-the-backyard-practical-application-of-deep-learning-c030d3263ba8

Bradski, G., & Kaehler, A. (2012). *Learning OpenCV:* Beijing: OReilly

Chapman, D(2020), Neural Nets for Image Classification, Retrieved from blackboard.umbc.edu

Chen, K. (1993, November 02). *US5258899A - Motion sensors lighting control*. Retrieved from https://patents.google.com/patent/US5258899

Géron Aurélien. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: OReilly Media, Inc.

How to Use Background Subtraction Methods. (n.d.). Retrieved from https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html

Node.js v14.3.0 Documentation. (n.d.). Retrieved May 21, 2020, from https://nodejs.org/api/stream.html

Rosebrock, A. (2015, May 25). Basic motion detection and tracking with Python and OpenCV. Retrieved from https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/

Rosebrock, Adian. (2017, September 11). *Object detection with deep learning and OpenCV*. Retrieved from https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv

Sandoval, K. (2019, June 28). *REST vs Streaming APIs: How They Differ: Nordic APIs*. Retrieved from https://nordicapis.com/rest-vs-streaming-apis-how-they-differ/

Spínola, D. (2019, July 09). Video Stream With Node.js and HTML5. Retrieved May 21, 2020, from https://medium.com/better-programming/video-stream-with-node-js-and-html5-320b3191a6b6

Surma, Greg. (2019, January 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?gi=179edc908dac

Transfer learning with a pre-trained ConvNet: TensorFlow Core. Retrieved May 21, 2020, from https://www.tensorflow.org/tutorials/images/transfer_learning