Neural Network Image Recognition Applied to Autonomous Quadcopters

Asher Mancinelli
Sophomore

Davis Mariotti
Junior

Whitworth University
Department of Computer Science
CS 457 - Artificial Intelligence

May 15, 2018

Abstract

Neural networks and autonomous vehicles have seen a large spike in developement in the last 10 years, and are often used in tandem. We attempted to merge the two fields through an image classification neural network used to control a quadcopter autonomously.

Nomenclature

Keras Library built to run with a backend of Tensorflow or another low-level matrix-multliplecation library. Has higher level abstractions, such that the arcitecture of the neural network can be more consisely built.

NN Neural Network: mathematic concept through which perceptrons are layered together and adjusted with sample data.

TF TensorFlow: a library built by Google for in-house neural network computations and large matrix calculations. Written in C, but most often used via a Python wrapper.

1 Goal

To create a program to interface a neural network (NN) such that our quadcopter will take off and fly in the direction of our target. The NN will take images from the quadcopter and classify them into the correct command to make the quadcopter follow the cube as it moves. The classification will then be converted into commands that the quadcopter can interpret and sent to the quadcopter.

2 Classification Methods

One of the primary methods employed by software programmers to control vehicles autonomously is artificial intelligence, a subfield of which is neural networks. To autonomously control a quadcopter, we used a specific NN framework, Tensorflow (TF).

The most notable drawback of using a NN is that it requires a very large data set for training. In order to create a dataset large enough to effectively train the NN, we wrote a script to take all the image data from the quadcopter while in flight. We then took our target object, a neon yellow-green cube, and walked it back at varying places relative to the quadcopter as we flew the quadcopter a rough 6 feet away from the target (see Figure 1: Training Method).

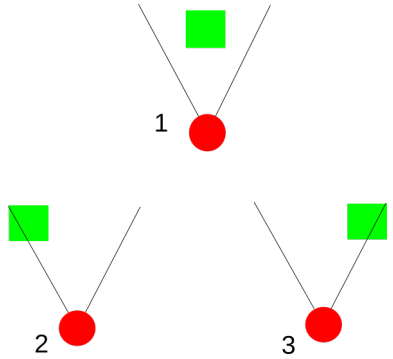

Figure 1: Training Method

By varying the locations the cube is held at relative to the quadcopter,

we obtained image training data assorted into the following categories representing the intended course of action the quadcopter should take:

- Forward
- Left
- Right
- No Action

Element 1 of Figure 1 shows the position relative to the quadcopter (red circle) that the cube (green square) is held when we obtain the training data for the Straight category. This is because the course of action the quadcopter should take is determined by where the cube is in the quadcopter's field of vision. Thus, the category for element 2 will be Right, element 3 is Left, and the No Action category is just a quadcopter view without the target at all. The images will be downsized to $150x150px$, and used to train the NN to classify images from the quadcopter's feed.

In production, the process will work as follows:

1. Quadcopter takes image
2. Image is sent to paired computer
3. Image is downsized to $150x150px$
4. Image is passed through the trained NN
5. Resulting classification is translated to quadcopter instruction
6. Quadcopter instruction is sent to and executed by quadcopter
7. Return to step 1

In this way, the quadcopter will act autonomously, although paired with another computer. However, given the small file size of the project, it would be feasable to port the NN and scripts to run it onto the board of a quadcopter, such that it could drive itself autonomously, without being paired with another computer.

3 Arcitecture

The arcitecture of our NN is built on the keras.models.Sequential model, with three sections of layers, followed by a final section which converts the network to output and introduces dropout. The arcitecture of our NN is as follows:

- Section 1:
  - 2 Dimensional Convolutional layer
  - Relu Activation layer
  - 2 Dimensional Max Pooling layer
- Section 2:
  - 2 Dimensional Convolutional layer
  - Relu Activation layer
  - 2 Dimensional Max Pooling layer
- Section 3:
  - 2 Dimensional Convolutional layer
  - Relu Activation layer
  - 2 Dimensional Max Pooling layer
- Section 4:
  - Flatten layers
  - Densely connected layer
  - Relu Activation layer
  - Dropout with 50% dropout
  - Densely connected layer
  - Sigmoid Activation layer

4 Problems We Faced

We ran into a countless number of problems with both ends of this project, and each of us had to work through elements on our own and as a team. One of the biggest problems we ran into was the image handling system of the library we used. OpenCV2 for Python only allows you to take a buffer of images, and does not allow any skips. Davis overcame this by placing all the images into a last-in-first-out queue to grab the latest image. I also ran into problems trying to get the model to compile and train efficiently , and how to save the model such that it can be loaded and used quickly enough for the controller to classify and send instructions to the drone. Parrot, the manufactorer of our drone, also pushed a firmware update which essentially broke the entire library which we used for controlling our drone, which has understandably caused lots of problems. In order to overcome this, we got a spare drone which didn't have the update and we were careful not to connect to another controller.

5 Conclusions

I personally found this project to be really interesting as I have always liked flying drones and neural networks seemed like a mysterious foreign concept before this. I enjoyed learning more about them and seeing how they work. On the neural network side of the project, things seemed to come together very quickly and seamlessly. I believe in the future, we would want more training data as the classifier wasnt perfect, but it could classify images with somewhere around 60% success rate in practice. With our training set, it could classify images with a 98% success rate, which showed us that if we had gathered more data, we would have seen better results. On the drone control side of the project, I encountered numerous difficulties. While it was pretty simple to use the PyParrot API to control the drone, I had many problems downloading images from the drones camera. To fix this, I ended up writing new code to download images instead of using the API. The second major drone problem stemmed from the fact that OpenCV, the image processing package we used, buffers images as they come in, and since we only needed images every half second, we needed a way to pull from the back of the buffer instead of the front which would become very delayed. To fix this problem, I implemented a last-in-first-out queue on a separate thread which would

constant add new images from the camera. The main thread could then pull an up-to-date image from the camera at will. The last problem went unsolved. The day before presentations, the drone forced a firmware update which made the camera and drone controls unreliable. Since downgrading would have been a dangerous option, we did not have enough time to develop a fix for the issue and were no longer able to test the drone.

6 Contributions

For this project, Asher handled much of the neural network design as he had much more experience with neural nets. He also helped gather the training data as that was a two person job. My role in the project was to design the software that would control the drone, and make use of the neural net. This involved multithreaded python programming as well as writing the software that would translate the output of the neural net to drone commands. I would say that the group contributions were about equal overall.