SnapAI Related Work

Kieran Kelly kell1170, Wyatt Kormick kormi001

April 30, 2018

1 Other Object Detection Approaches

1.1 Genetic Algorithm Based Optimization

This method uses only a genetic algorithm to find natural edges between objects in an image. The genetic algorithm takes in a gray-scale image and produces an image of the same size. The pixels of the output image have two values, 1 or 0 (white or black), corresponding to whether or not the pixel belongs to an edge. The fitness function used calculates probabilities of a pixels in the image belong to an edge based on dissimilarities between neighboring regions pixels and compares the output image to these values. The first generation were given an output image consisting of randomly selected values for pixels. Each generation used 512 images, and finished when there were 15 consecutive generations where the fittest image did not change. Reproduction involved trading regions of pixels between mates, and mutations involved flipping the values of pixels. The higher the fitness of an image, the higher its probability to reproduce. This method of edge detection was tested, along with simulated annealing and local search algorithms, on a series of natural and synthetic images, and was found to perform much better than both in terms of time to find the solution and robustness to noise.

1.2 Reinforcement Learning with Classifiers

This method uses a classifier, trained on a data set of 12,000 human-labelled images to detect boundaries between objects in an image. This classifier sampled circular regions of the input image, split them in half, checked the brightness, color, and texture in the regions, and if it found them to be sufficiently different, put a border of black pixels between the two regions in the output image [MFM04]. After the classifier put where it thought there would be boundaries in the image, it would check it against where the human place boundaries in the image. Depending on how far away the classifier's output image was from the human's the classifier would adjust the weights and biases of the its network through a process called back-propagation. The goal of this process is that after training itself on a large training set, it would be able to find the boundaries in images it had never seen before. This classifier was found to be able to detect objects in images with low noise levels at a rate of success of just above 90%, but as noise levels increased, the success rate declined exponentially.

A similar method was used in [Giu+16] to detect a usable path to traverse. This classifier implemented a deep neural network to tell the camera which direction to turn based on what the camera saw moment-to-moment. Like the other classifier, this neural network was trained on thousands of images. This method, however, used 17,000 images of trails, and the classifier's job was to say whether the camera needed to be turned left, turn right, or stay the same in order for the camera to be facing the trail straight on. In an experiment, the authors found that their deep neural network managed to classify a sample set of images just as well as human observers could, with a consistent 85% success rate.

1.3 Bayesian Modeling

This method of object detection computes probability models of the foreground objects and the background and the two compete over ownership of the pixels. The models are each 5 dimensional (x, y, red, green, blue) and consist of every pixel in the image. Instead of modelling each pixel independently, it models the entire background as one image. This is to show the dependability of pixels on each other, as pixels nearby to a high probability background pixel are going to have a higher probability of being a background pixel, likewise for foreground pixels. For every frame the probability of a pixel belong to the background or the foreground is calculated for every pixel. A sliding window of frames is kept for both the foreground and the background, which corresponds to the learning rate of the system.

The background of an image should have a set structure. It is either stationary or it moves in some sort of a pattern, so pixel regions that follow this structure have a higher probability of being in the background. The foreground calculates its probabilities based on the idea of temporal persistence [SS05]. Basically what this means is that objects tend to stay in the same general area, and stay the same general color between frames. Changes have to occur over time and aren't instant. At first, the probability of any pixel belonging to an object is uniform across all pixels and frames. Then, pixel regions that vary from other dominant pixel regions have a higher probability of belonging to a foreground object. Once a pixel is found to have a high probability of belong to a foreground object, nearby pixels have a higher probability of also belonging to an object than those further away. At the next frame, there is a high probability of a pixel region to be in the foreground if, in the previous frame, there was a pixel region in the foreground in the same general area. This strategy of object detection had an extremely high detection rate at 99.708%. In each scene nearly every object that was detected by a human observer was also detected by the algorithm.

1.4 CAMSHIFT

Continuously Adaptive Mean Shift (CAMSHIFT) is another probability-based approach to object detection and tracking [Bra98]. The original "mean shift" algorithm's use is to find peaks in probability distributions, and has been used previously to divide static images into regions. CAMSHIFT modifies the mean shift strategy to find these peaks in probability distributions that change in time along with a sequence of video frames, such that it instead tracks dynamic image regions. The algorithm chooses an arbitrary section of the image to search. It then calculates the mean pixel values in that region, and chooses a new subregion that contains those values, and repeats on the new region. It does this until is converges on what it decides is an image region. CAMSHIFT applies this strategy over a sequence of video frames to track the movement of image regions. CAMSHIFT ends up being a simple, efficient algorithm, but is found to still be able to handle noisy images, and objects moving in irregular ways.

1.5 Comparison

Object detection and tracking doesn't have a definite solution yet. As such all available strategies for solving the problem are based in probability and learning, neither of which are 100% accurate. Of these methods CAMSHIFT provides high accuracy with high efficiency, and Bayesian Modeling provides near-perfect accuracy, but both are beyond our abilities to implement. Using only a genetic algorithm to solve our problem probably would take a lot of generations to accomplish, which would take a lot of time that we do not have. If we were to try this again using one of these methods, we would use a supervised neural network, as it is within our abilities to design, and can be done in a reasonable time frame while also likely providing a higher accuracy than our current method.

2 Refining / Structuring a Neural Network

2.1 Genetic Algorithm to Refine a Neural Network

[WL93] provides the basic strategy that we elected to use to solve our problem, a genetic algorithm used to optimize a neural network. More specifically the genetic algorithm decides the optimal amount of hidden layers, the number of neurons in each hidden layer, and the weights and biases connecting each. We use a simpler version that only optimizes the weights and biases, and our hidden layers maintain the same size. The algorithm, dubbed ”GANNet”, runs the problem on randomly generated neural networks. The highest scoring networks are then selected to breed together to create a new network using values from both parents. This is done by selecting values from one parent and overwriting the values in the other parent with them. These new individuals are then mutated by randomly changing randomly selected values in the neural network. This is done to provide diversity in order to search more of the possible options. The new generation that the problem is then run on is a selection of the parents and the new children. This strategy is always guaranteed to converge on a network that is better than a starting network, but it has the problem that it is possible for it to converge on a solution that is not a global solution. This is a problem that is common to most genetic algorithms, and is one that we are prepared to accept.

2.2 Genetic Algorithm Optimal Population Size

Finding what size of population to use with a genetic algorithm is tricky. It largely depends on the complexity of the problem [Ala92]. Low populations have a much more rapid mutation from the original, which has been proven in nature, while larger populations resist to mutation more. Time of execution is also a factor to consider, which is tied to the complexity of the problem. Seeing as our neural network has quite a high complexity, in order to reduce the time of execution we have limited our population size down to five. This also helps with a more rapid mutation which we predict to lead to faster results. This does run the risk of mutating quickly away from a potentially successful neural network structure, but it is a calculated risk we are willing to take.

2.3 Mutation in Genetic Algorithms

Looking at nature we can look and see there are various ways to create the next generation [SF96]. There is self mutation, crossover, crossover mutation to name a few, and most genetic algorithms only take advantage of self mutation. Using different or multiple types of mutation can be hugely beneficial to the algorithm to achieve an optimal solution more quickly. That is why we utilize all three of the methods mentioned above in our genetic mutation. This provides us the possibility of converging on an optimized solution more quickly.

2.4 Neural Network Ensembles

Using an ensemble of neural networks to improve training and performance can be very effective [HS90]. Using cross validation, Hansen and Salamon optimize using cross-validation in order to get a more generalized neural network that can work across multiple data sets instead of just keying in on the training data set. On top of that they use an ensemble because of the issue of optimizing weights being tied to local minima, not global, so multiple runs of the algorithm can give different results. We took the idea for an ensemble for training purposes and decided that since we only care about a single data set, that a genetic algorithm would be better suited on the ensemble of neural networks to try to optimize the best solution.

3 A Different Approach to the Problem

3.1 A Minimax Algorithm

Find a maximizing method for player 1 when player 2 isn't necessarily trying to minimize player 1's payoff in a a zero sum game. Creating a finite-state machine to mimic a nonminimax player is the problem with finding an algorithm to maximize player 1's payout accurately. They use an evolutionary programming technique to try to achieve this. It takes the finite-state machine used to mimic the maximizing player and randomly creates a mutation from one of four choices [Bur69]. The resulting child finite-machine's average payoff is then analyzed and compared with the parent's average payoff. If the child has a worse average payoff than the parent then it is discarded and another mutation is chosen. They use this technique with several children which are evolved in parallel. They conclude that the experiments they run indicate success with this method.

This approach is quite comparable to the method we chose to use. They use a genetic algorithm very similar in how we use ours, the subject of their genetic algorithm however is different. While we are trying to find an optimized neural network to play the game, they are trying to find an optimized finite-state machine to imitate the player. While it seems this would create the same result, we decided a neural network better covers our specific problem. With the minimax algorithm we would have to treat the game itself as a player not as the environment it is.

3.2 Alpha-Beta Pruning

The concept of alpha-beta pruning is similar to an algorithm called ”branch and bound.” The main difference between the two algorithms is that ”branch and bound” doesn't have both a lower and upper bound so it cannot make ”deep-root cutoffs”[KM75] like alpha-beta pruning can. The most notable possible applications of alpha-beta pruning is chess. It analyzes the best case with a proof that they are dealing with the optimal case. It then deals with a non-optimal case with no ”deep cutoffs” before finishing talking about the model of the algorithm.

Alpha-Beta pruning would've been something to try to add to the minimax algorithm above. The specifics of how to accurately apply pruning to a scenario where one player in a non-minimax player proved very difficult which is ultimately why we decided against using a minimax algorithm with or without pruning.

3.3 State Space Search

State space search (SSS*) [Sto79] proposes a better strategy for the minimax algorithm than alpha-beta pruning. They achieve this by traversing the tree in parallel instead of a left-to-right strategy that alpha-beta is forced to do. They then delve into the definition of state space search in general and explain how it applies to create SSS*. Following with a proof of correctness to back their claim. Then they compare the efficiency of SSS* to alpha-beta by running an extensive simulation in which every test they made SSS* performed strictly better.

This strategy is something that could've been used in tandem with the minimax algorithm above. We ran into similar struggles as we did with the Alpha-Beta pruning, how do we apply it to the minimax algorithm with a non-minimax player? This proved to be beyond our scope of understanding so we chose not to go this way with our approach.

References

[Bur69] G Burgin. “On playing two-person zero-sum games against nonminimax players”. In: *IEEE Transactions on Systems Science and Cybernetics* 5.4 (1969), pp. 369–370.

[KM75] Donald E Knuth and Ronald W Moore. “An analysis of alpha-beta pruning”. In: *Artificial intelligence* 6.4 (1975), pp. 293–326.

[Sto79] George C. Stockman. “A minimax algorithm better than alpha-beta?” In: *Artificial Intelligence* 12.2 (1979), pp. 179–196.

[HS90] Lars Kai Hansen and Peter Salamon. “Neural network ensembles”. In: *IEEE transactions on pattern analysis and machine intelligence* 12.10 (1990), pp. 993–1001.

[Ala92] Jarmo T Alander. “On optimal population size of genetic algorithms”. In: *CompEuro'92.'Computer Systems and Software Engineering', Proceedings.* IEEE. 1992, pp. 65–70.

[WL93] David White and Panos Ligomenides. “GANNet: A genetic algorithm for optimizing topology and weights in neural network design”. In: *International Workshop on Artificial Neural Networks.* Springer. 1993, pp. 322–327.

[BZP94] Suchendra M Bhandarkar, Yiqing Zhang, and Walter D Potter. “An edge detection technique using genetic algorithm-based optimization”. In: *Pattern Recognition* 27.9 (1994), pp. 1159–1180.

[SF96] Jim Smith and Terence C Fogarty. “Self adaptation of mutation rates in a steady state genetic algorithm”. In: *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on.* IEEE. 1996, pp. 318–323.

[Bra98] Gary R Bradski. “Real time face and object tracking as a component of a perceptual user interface”. In: *Applications of Computer Vision, 1998. WACV'98. Proceedings., Fourth IEEE Workshop on.* IEEE. 1998, pp. 214–219.

[MFM04] David R Martin, Charless C Fowlkes, and Jitendra Malik. “Learning to detect natural image boundaries using local brightness, color, and texture cues”. In: *IEEE transactions on pattern analysis and machine intelligence* 26.5 (2004), pp. 530–549.

[SS05] Yaser Sheikh and Mubarak Shah. “Bayesian modeling of dynamic scenes for object detection”. In: *IEEE transactions on pattern analysis and machine intelligence* 27.11 (2005), pp. 1778–1792.

[Giu+16] Alessandro Giusti et al. “A machine learning approach to visual perception of forest trails for mobile robots”. In: *IEEE Robotics and Automation Letters* 1.2 (2016), pp. 661–667.

4 Group Contributions

4.1 Kieran Kelly

- 6 References
- Subsections 2.2, 2.3 and 2.4
- Section 3

4.2 Wyatt Kormick

- 6 References
- Section 1
- Subsection 2.1