Proceedings of Machine Learning Research 123:100–109, 2020 NeurIPS 2019 Competition and Demonstration Track

Evolution Algorithm and Online Learning for Racing Drone

Sangyun Shin KIMSHIN812@GMAIL.COM
Yongwon Kang KYW2539@SJU.AC.KR
Yong-Guk Kim* YKIM@SEJONG.AC.KR
Department of Computer Engineering, Sejong University, Seoul, Korea

Editor: Hugo Jair Escalante and Raia Hadsell

Abstract

Drone racing has become one of the challenging topics in robotics and machine learning because such a drone requires to equip with high performing modules that carry out demanding tasks, such as obstacle avoidance, mapping, and planning. In addition, one of the most crucial aspects of the racing drone is its speed. However, this is the somewhat less studied area compared to conventional topics such as obstacle avoidance and path-finding, probably because designing a loss function for the speed optimization with the gradient-based method is difficult. In this paper, we propose an evolutionary scheme for optimizing the speed-related parameters for shortening the travel time rather than using the gradient-based loss for them. For the planning part, we use an online learning method with the racing parameter optimization. Therefore, our approach is to combine evolutionary algorithms for speed optimization and gradient-based online learning, achieving first place in Tier 2 and Tier 3 in Game of Drones competition at NeurIPS 2019.

Keywords: Deep Drone Racing, Evolutionary Algorithm, Online Learning

1. Introduction

AI-based drone racing has gained popularity lately, mainly because of the challenges in operating the drone fast and accurately. For the successful drone racing, it requires high performing technologies that have been actively studied with deep neural networks. In particular, vision-based obstacle avoidance capability has become important in drone racing because conventional navigation algorithms, such as Simultaneous Localization and Mapping(SLAM) show its weakness in dealing with the change in scene, though it has met great needs on a static map without changes. This has led studies utilizing SLAM to apply a vision-based approach with deep neural network, *i.e.* object detection, for dealing with the change in the scene. On the other hand, instead of merging deep models into SLAM, studies developing a neural network to handle everything have shown promising results in drone navigation. To take advantage of the recent progress in deep learning for the task, these studies have diverged into either utilizing supervised learning or reinforcement learning. While deep neural network-based approaches have shown great performance in navigating drone by obstacle avoidance, however, gradient-based optimization of deep models have difficulties in optimizing certain conditions, such as maximizing the speed, which is the crucial condition for drone racing.

Recent studies have suggested that using evolutionary algorithms with neural networks could enhance optimization performances. Though there are many applications of evolu-

tionary algorithms for neural network-based systems, one of the crucial reasons for adopting the evolution scheme is that such algorithms do not require gradient during the optimization process. In the present study, we propose an evolution-based optimization for racing parameters with simultaneous gradient-based online learning for drone racing. Our contributions are:

1. In contrast to the conventional drone navigation problem where speed parameters are often designed by manual fine-tuning, we propose an evolution-based approach for optimizing the speed parameters.
2. A significant advantage of our proposed system is that the speed optimization for the racing does not require any gradient-based method, leading to the remarkably simple and intuitive design of the objective function, *i.e.*, travel time of the racing drone.
3. It is shown that two different optimization methods, such as gradient-based and evolution-based approaches, can successfully co-operate with each other to make a significant improvement of the system performance for drone racing.

2. Related work

2.1. Neural network based drone controller

Deep neural network-based methods have been proposed to enhance the ability to control the drone using computer vision for collision avoidance, navigation, *etc* (Carrio et al., 2017). Among many, supervised learning and reinforcement learning have been successful. First, supervised learning can be a useful approach if a large amount of labeled data is available. Recently, by utilizing a simulation environment during training, a CNN based model for real-time navigation in a drone-racing track is proposed using micro UAV (Kaufmann et al., 2018). Similarly, by using a large amount of data collected from the urban area, it has been shown that a drone can navigate robustly by avoiding obstacles appearing in a city (Loquercio et al., 2018).

Some researches try to overcome the issue of collecting and labeling a large number of a dataset with the Reinforcement Learning(RL) perspective. By using sensory data from an accelerometer, it is shown that the RL model can plan swing-free trajectories while carrying the suspended load task (Faust et al., 2017). By applying a segmentation model with actor-critic models, (Shin et al., 2020) has shown that an actor-network can successfully navigate through obstacles using only visual input, where they have used the pre-trained actor model and have trained it for the Game of Drones NeurIPS 2019 competition.

2.2. Neural network optimization with evolutionary algorithm

Recently, evolutionary algorithms have gained popularity in assisting the optimization process for deep neural network based system. (Miikkulainen et al., 2019) has shown that evolutionary optimization for finding the best architecture enhances the performance of the network without any delicate tuning. Instead of finding the architecture, (Young et al., 2015) has shown that an evolutionary algorithm could be used to optimize the hyper-parameters of the network. In this study, a genetic algorithm (Davis, 1991) among many evolutionary algorithms has been used for the hyper-parameter optimization for the racing drone.

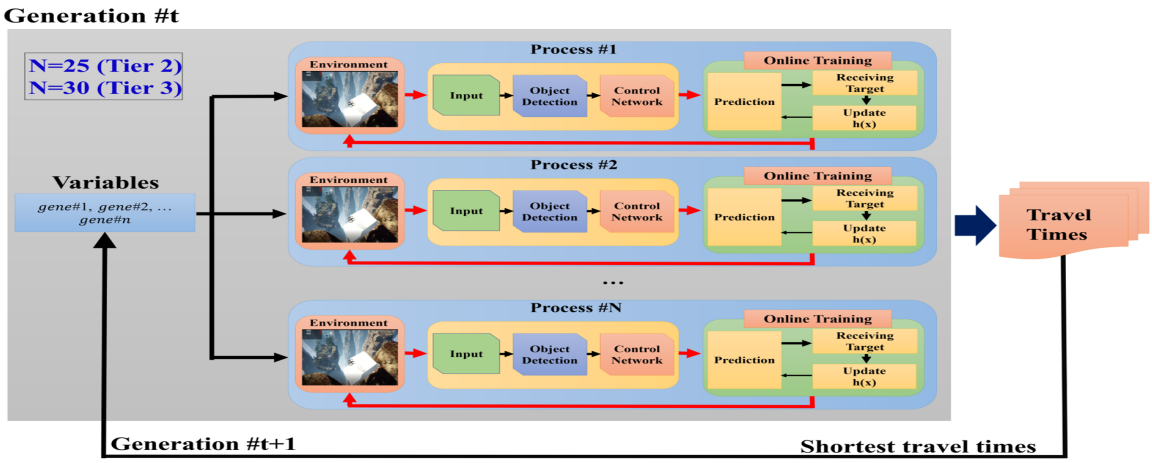

Figure 1: Flow diagram of the racing parameter optimization with online training for each process. Using the genetic algorithm, t th generation consisting of N population optimizes racing parameters such as maximum velocity, acceleration, and *dist*. Each *gene* consists of racing parameters, and their corresponding process runs the racing with the copied object detection model and control network f. Once the t th generation finishes, the best performing genes are chosen, and their parameters are combined together to make $t+1$ th generation. Note that the online training for f is performed during these racings.

3. Method

In this section, we describe our learning process for simultaneous online training of neural network as well as optimizing the latent vectors consisting of racing parameters for the racing. Object detection models are adopted to detect the gates in the given monocular First Person View(FPV) image.

3.1. Online training of neural network for controlling the drone

To make a neural network learn to carry out sophisticated navigation against challenging objects and tracks, the online training scheme is crucial throughout the multiple drone racings. We have adopted a pre-trained actor-network f (Shin et al., 2020) for the online training with the object detection model, as shown in Figure 2. f outputs linear velocities in X, Y, and Z, which are later converted to the global coordinates through the post-processing described in the latter of this section. Therefore, the training of f requires target values corresponding to X(horizontal), Y(forward), and Z(vertical) movements, respectively. The loss function L_o for online learning of f, given the processed data FPV as shown in Figure 2, is defined as:

$$L_o = \frac{1}{n}\sum_{i}^{n}(C_i - f(FPV_i))^2 \tag{1}$$

where C^{xz} for horizontal and vertical targets are :

$$C_t^{xz} = f(x, z|FPV_{t-1}) + \frac{img_{center} - Gate_t}{img_{width,height}}. \tag{2}$$

The target value C^f for moving forward using the distance d between a current position to a next gate is:

$$C_t^y = f(y|FPV_{t-1}) + \frac{d_{t-1} - d}{d_{t-1}} \tag{3}$$

Here, $Gate_t$ stands for the center of a detected gate in X and Y from the raw image plane when the t_{th} action is made, and f indicates a neural network that receives the location of the gate detected in an image plane. Note that X in image plane corresponds to the horizontal movement in control output of f, whereas Y in image plane corresponds to the vertical movement in f. img_{center} is the center of the image plane in X and Y. img_{width} and img_{height} are the width and height of the image, respectively. Note that L_o is designed to make the drone passing through the center of the detected gate by considering the previous outputs with regard to the center of the gate, as shown in Figure 3.

The control function we have used is *moveBySplineAsync*, as the function provides a reliable control for Yaw. That is, the Yaw of the drone is automatically changed following X and Y movement in global coordinate. Since our control network f produces X, Y, and Z linear velocities in the local frame, we have used the rotation matrix to covert the linear velocities into the global coordinate. The equation we used to calculate X, and Y global position with the linear velocities from f is as follow:

$$\begin{aligned} q_x &= o_x + cos(Yaw) * (p_x - o_x) - sin(Yaw) * (p_y - o_y) \\ q_y &= o_y + sin(Yaw) * (p_x - o_x) + cos(Yaw) * (p_y - o_y) \end{aligned} \tag{4}$$

where p_x and p_y stand for the point that x and y vectors were initially pointing, and o_x and o_y are the origin of the vector. q_x and q_y are the rotated vector given yaw angle Yaw. For 4 directions, such as Forward, Backward, Left, and Right, q_x and q_y have been calculated for every direction each time a control command has to be made. A combination of these vectors is used to make the drone move. Since controlling roll and pitch leads to make the drone move to either forward/backward or left/right, respectively, we were able to convert the pre-trained actor model's output into the command for *moveBySplineAsync*. Once the drone moves by setting a position in X, Y, and Z, we have set a threshold parameter *dist* for the termination of the command. That is, the movement command stops when the distance between the drone's current position and destination is less than *dist*. Figure 2 shows the process of online training for making the drone pass through the center of the gate.

3.2. Racing parameter optimization with evolutionary algorithm

Racing parameters considered for the optimization are maximum velocity *vel*, acceleration *acc*, and a distance threshold *dist* that determines whether or not the drone arrives at a location where it was previously produced by the neural network f. The three parameters are set according to the number of sections in the track. That is, given the track is divided into n number of sections, the three parameters belonging to the section is applied to the

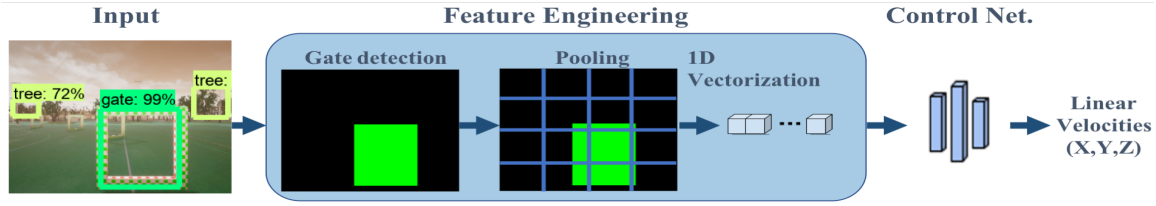

Figure 2: Illustration of the flow from detecting the object to making an input for Control Net f. An array with the same size as the image is first initialized with zero, and the region where the gate is detected is filled with value one. The array is divided into width w and height h for averaging the value inside each of them. Lastly, these $w \times h$ number of average values are fed into the control network f for producing the linear velocities of the drone.

output of the neural network f in Equation 1. For example, if the drone is flying in section 1, then the velocity, acceleration, and the distance threshold set for section 1 are applied to the output of neural network. By defining *Airsim* as a function running the racing and returning the termination time, the objective function we define for the minimization using a genetic algorithm is as follow:

$$Airsim(w^1_{vel}, w^1_{acc}, w^1_{dist}, .., w^n_{vel}, w^n_{acc}, w^n_{dist}) \tag{5}$$

where n is a total number of gates, and w is variable that the genetic algorithm has to optimize. Note that there are $3n$ variables in total. Using w, function *moveOnSplineAsync* is called with its parameter set to w_{vel} and w_{acc}. For instance, if the drone is moving just passed first gate and is going toward the second gate, all of *moveOnSplineAsync* call is carried out with the parameter w^2_{vel} and w^2_{acc} with the threshold set to w^2_{dist} until the drone passes gate 2. In the experiment, we have set n as the number of gates, so that the section is divided by the gate. For instance, if the drone passed through the gate 1 with then the w^1_{vel}, w^1_{acc}, and w^1_{dist} then section becomes 2 and w^2_{vel}, w^2_{acc}, and w^2_{dist} for section 2 are applied for the control. The process was terminated either when the drone arrives at the last gate or when the travel time exceeds the shortest travel time, which leads each generation to evolve faster.

4. Experiment

In this section, we describe the experimental design for the competition and demonstration of the results for Tier 2 and Tier 3 that we attended. First, we show the training and testing performances for the object detection models, followed by the online training process for passing through the centers of gates. Lastly, a detail evolutionary scheme we used for the racing parameter optimization is illustrated.

4.1. Software and hardware specifications

A workstation equipped with Intel Xeon E5-2600 v4 CPU and 16×Nvidia Titan X with 256 Gb RAM were used for optimizing both racing parameters and L_o. Racing environments

	FPS	Complete/ Non-complete	Missing	Collision	Lag
SSD Mobilenet	27	9/1	2.3	5.4	138
SSD Inception	21	7/3	4.1	7.2	172
FRCNN Resnet	9	9/1	4.72	2.4	220
RFCN Resnet	8	5/5	8.4	2.2	254

Table 1: Success ratio, average numbers of the missing gate, collision, and lag time when using each object detection model for Tier 2 in **qualification round**. Success ratio was measured by calculating whether or not the drone could reach the goal during 10 trials. Note that the experiments for the success ratio were carried out after the first reach to the goal point during online learning.

	FPS	Complete/ Non-complete	Missing	Collision	Lag
SSD Mobilenet	26	9/1	1.7	3	183
SSD Inception	21	8/2	3.4	5.2	214
FRCNN Resnet	9	9/1	2.2	2	275
RFCN Resnet	9	7/3	4.2	1.4	290

Table 2: Success ratio, average numbers of the missing gate, collision, and lag time when using each object detection model for Tier 3 in **qualification round**.

were provided by Microsoft Airsim (Madaan et al., 2020) for Game of Drones competition at NeurIPS 2019. Python 3.6, Tensorflow 1.10.0 (Abadi et al., 2015), and OpenCV 3.4.1 (Bradski, 2000) were used for our network programming in Ubuntu 16.04 OS.

4.2. Object detection models

In this experiment, we wanted to see which object detection model performs the best for the task. Since the task required a detection model to not only accurate but also fast, we carried out experiments with various models using Tensorflow Object detection model zoo(Pkulzc et al., 2019). For the training, we collected 987 images from training maps and qualification maps. These data were labeled into four classes, consisting of tree, sky, gate, and gate entry. Table 1 and Table 2 show the detection models' performances for Tier 2 and Tier 3. By considering both the accuracy and the execution speed, $SSDMobileNet$ was selected for the final round and remaining experiments.

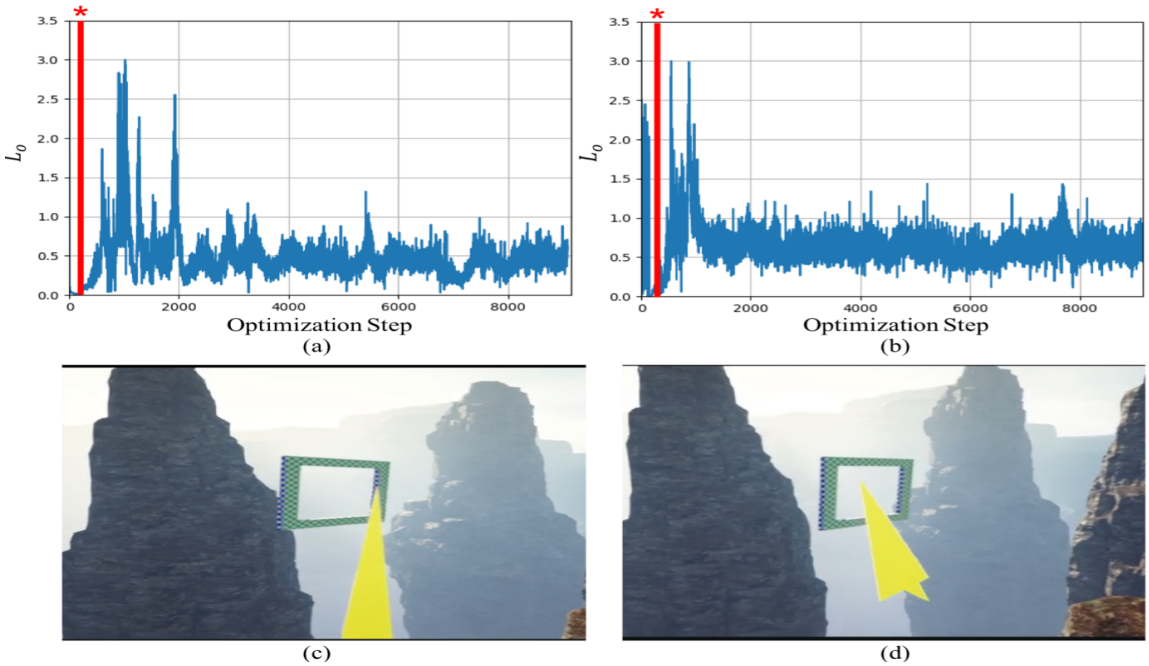

Figure 3: Illustrations of the average loss values L_o throughout the training for Tier 2 (a), and Tier 3 (b). Here, an optimization step means a racing from the start to the terminal state. The red lines in each (a) and (b) indicate the steps when the evolution-based optimization started. The yellow lines in (c) and (d) point the direction produced by the control network f at the 11th gate in Tier 3. Note that the control network f outputs the direction with the possible collision toward the gate at the optimization step 120 in (c), whereas it outputs the direction towards the center of the gate at step 160 in (d).

4.3. Online training for control network f

Training the control network f was carried out by online training for Tier 2 and Tier 3 individually. Before the training, we have adopted a pre-trained control network f from the study (Shin et al., 2020), where a model f was trained to maneuver the drone towards the indicated direction from a segmentation model. However, since the environment for the competition is much more challenging with diverse objects, we have used the object detection model and systematically label the gate's position with higher values as similar to the segmentation model indicated where to go, as shown in Figure 2.

Figure 3 shows the process of online training and the loss values L_o calculated by Equation 1. Before the racing parameter optimization, we have trained the network by minimizing L_o until the drone was able to reach the destination. For Tier 2, 153 racings in total were needed for the initial reach to the destination, and it was 189 racings for Tier 3. After the initial arrivals to the goal point, the optimization of the racing parameters started. For the learning, Adam optimizer with the learning rate $5e^{-3}$ was used.

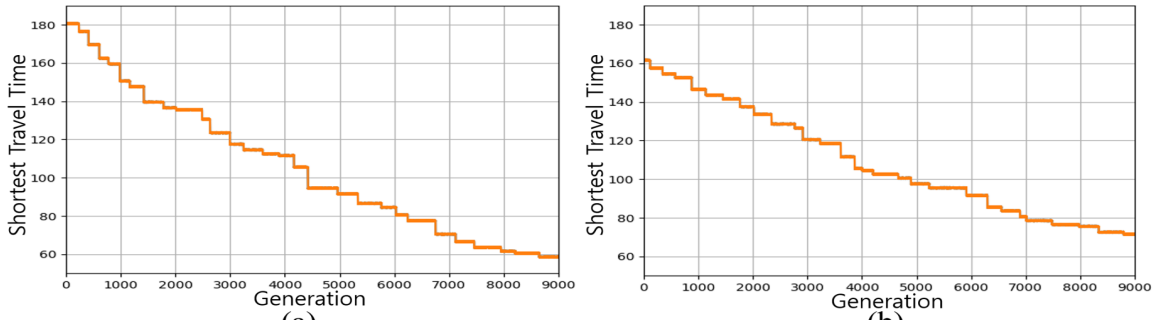

Figure 4: Illustration of travel times for Tier 2 (a) and Tier 3 (b) during the racing parameter optimization using the genetic algorithm for the final round. Here, the shortest travel times among genes of each generation are chosen. Our demo video can be found at https://youtu.be/I8z1Ebs6eoY

It was shown that the loss value went down stable before the racing parameter optimization began. However, with the racing parameter optimization, the loss value fluctuated as the hyper-parameters, such as *vel*, *acc*, and *dist*, changed through the optimization. Based on this, it was found that those parameters could make a difference in resulting action, though the network f produces the same value in the identical situation. This indicates that changing the racing parameters causes a significant effect in racing.

4.4. Racing parameter optimization

To see how much optimizing the racing parameter could help the overall performance, we have adopted the genetic algorithm, which is one of the evolution-based optimizations. A number of mutations were set to 2 in each generation, and the number of parents for the next generation was set to 4 for both Tier 2 and Tier 3. Initial parameters were generated from random uniform distributions. For *vel*, the range of initial values was from 30 to 200, and it was from 30 to 100 for *acc*. Lastly, it was from 2 to 32 for *dist*. As shown in Figure 4, optimizing the racing parameters have significantly improved the travel time. As shown in Figure 1, we adopted parallel processes for this optimization, where each process played a role as a gene. The shortest travel time for Tier 2 among 225,000 racings during 9000 generation was 56 sec, and it was recorded 71 sec for Tier 3 among 270,000 racings in total. During the generations, the travel time became shorter without fluctuating, because every racing was terminated when its travel time exceeds the shortest travel time of the moment. However, there were periods when the algorithm had difficulty finding better racing parameters. For example, around generation 4400 in Figure 4 (a), it took about 600 generations to find the better racing parameters that make the travel time shorter.

5. Conclusion

In this study, we propose a learning scheme where multiple processes are individually dedicated to finding optimum racing parameters while each of them simultaneously carries out online learning for vision-based drone racing. Within very challenging environments

where high-performance obstacle avoidance and navigation abilities are necessary, we have made use of a single network dealing with such challenging surroundings. We found that an evolution-based approach for racing parameter optimization can play a crucial role in shortening the travel time, which led us to achieve first place in the competition. In contrast to the conventional method for drone navigation, such as SLAM, our approach requires a single neural network maneuvering the drone without expensive computation for post-processing. Such an approach makes our drone free from the lag caused by heavy post-processing for the control command, which in turn makes it reach the destination fast.

Acknowledgments

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2020-2016-0-00312) supervised by IITP.

References

Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

Adrian Carrio, Carlos Sampedro, Alejandro Rodriguez-Ramos, and Pascual Campoy. A review of deep learning methods and applications for unmanned aerial vehicles. *Journal of Sensors*, 2017, 2017.

Lawrence Davis. Handbook of genetic algorithms. 1991.

Aleksandra Faust, Ivana Palunko, Patricio Cruz, Rafael Fierro, and Lydia Tapia. Automated aerial suspended cargo delivery through reinforcement learning. *Artificial Intelligence*, 247:381–398, 2017.

Elia Kaufmann, Antonio Loquercio, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In *CoRL*, 2018.

Antonio Loquercio, Ana I Maqueda, Carlos R del Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

Ratnesh Madaan, Nicholas Gyde, Sai Vemprala, Matthew Brown, Keiko Nagami, Tim Taubner, Eric Cristofalo, Davide Scaramuzza, Mac Schwager, and Ashish Kapoor. Airsim drone racing lab. *arXiv preprint arXiv:2003.05654*, 2020.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

Pkulzc, Vivek Rathod, Mark Sandler, and Neal Wu. Tensorflow detection model zoo. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md, 2019.

Sang-Yun Shin, Yong-Won Kang, and Yong-Guk Kim. Reward-driven u-net training for obstacle avoidance drone. *Expert Systems with Applications*, 143:113064, 2020. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2019.113064. URL http://www.sciencedirect.com/science/article/pii/S095741741930781X.

Steven R Young, Derek C Rose, Thomas P Karnowski, Seung-Hwan Lim, and Robert M Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, pages 1–5, 2015.