DEEP DRONE RACING

DEEP LEARNING, TAU, 0510-7255, FALL 2019

Idan Basre
304834492
idanbasre@gmail.com

Alon Kurzweil
303093819
alonkur@gmail.com

March 10, 2020

ABSTRACT

Fully autonomous drones can be valuable for different tasks. Developing them has many challenges, such as accuracy, quick response for dynamic environment and computational cost on board. Drone racing is a platform for testing autonomous drones, and it became popular competition for academic robotic groups. DroNet is a network that has been used by UZH robotic and perception group. This network has shown good performances. It is based on resnet8 architecture. Residual networks (ResNet) are neural networks in which each layer consists of a residual convolutional layers and a skip connection bypassing the residual path. Multi-residual networks (Multi-ResNet) are created by adding residual connections to each layer thus increases the multiplicity and complexity of the network, while keeping its depth fixed. In this project we tried improving DroNet performances with limited additional computations. We examined the implication of changing the network's residual blocks to MultiRes blocks, and the implication of using a deeper network. Our results show that the MultiRes approach does not improve DroNet performances (even makes them worse), and that deeper DroNet converges faster but does not improve performances.

***K*eywords** DroNet · Drone Racing · Multi-Residual

1 Introduction

Nowadays drones became cheaper and popular, and are being used for simple tasks because they need to be controlled manually. Transforming these platforms to be autonomous can be valuable to perform different kind of tasks, such as disaster response or structure inspection. Developing a fully autonomous drones that can quickly and safely fly through complex and dynamic environments is a difficult problem. In order to do so, the research community has a major interest in drone racing. Drone racing is a popular sport in which professional pilots fly small quadrotors through complex tracks at high speed. Drone pilots undergo years of training to master the sensorimotor skills involved in racing. Developing a fully autonomous racing drone has challenges such as onboard perception, localization and mapping, trajectory generation and optimal control.

One approach to autonomous is to fly through the course by tracking a precomputed global trajectory. However, global trajectory tracking requires to know the race layout in advanced, along with highly accurate state estimation – both are not necessary the situation in our case, since gates can be moved from lap to lap, and since drifts in the state estimation are not neglectable.

Another approach, the approach that we choose, is to identifies dynamically waypoints in local body-frame coordinates. These waypoints are computed locally on the drone, and there is no need to know the track in advanced. In order to calculate these waypoints, we use convolutional neural-network that its inputs are pictures that are being captured by the drone itself. After a waypoint is being calculated, it is being fed to a short-term trajectory planner, that plans the trajectory to this single waypoint. This process repeat itself from one waypoint to another, until the race is finished.

Our challenge is to build and train a convolutional neural-network that support the racing drone to succeed in its race – via all race's complexities (different gates positioning, size, texture and shape, in addition to different light conditions).

We have used an existing network named DroNet (described in [2]). This network is a successful network for autonomous drone racing. We have tested two approaches for improving this network performance and optimization process. We have succeeded improving the optimization process by increase the convergence rate.

2 Related Work

A wide variety of techniques for drone navigation and obstacle avoidance can be found in the literature. This is more complex when the only inputs are pictures that are being captured by the drone itself and that all the computation is done locally on the drone. One approach is to train a neural network policy by imitating expert behavior. This is done by supervised learning using captured picture labeled with waypoints selected by an expert policy [3]. Learning navigation policies from real data is difficult and has high cost of generating training data. Recent works suggests that this data can be simulated and still enable high navigation performances in real life. This is done by generating an abundance of simulated data via domain randomization that makes the neural network robust to changes [1].

In order to reduce image processing time a simple convolutional neural network (CNN) was proposed by [2] and was named DroNet. DroNet predicts a goal direction in local image coordinates, together with a desired navigation speed, from a single image collected by a forward-facing camera. The main part of the net consists of a ResNet-8 architecture. The residual blocks of the ResNet, were proposed by [6].

Residual networks are neural networks in which each layer consists of a residual blocks that consist of convolutional layers and an identity function skip connection. This architecture enables very deep networks that converge without accuracy degradation or high training error [6]. Deeper neural networks often have better performances but every percentage of improvement requires increasing the number of layers, which linearly increases the computational and memory costs.

A study of deep residual networks showed that those networks act like ensembles of relatively shallow networks and that only the shallow networks are needed during training [5]. Using residual blocks with multiple parallel layers as suggested by [4], creates ensembles of shallow networks that achieves performances of deeper residual network (with regular residual blocks) but with less computational cost and faster convergence.

3 Data

In contradiction to other works, the network training is done by simulated data (and not like other works that use real pictures from a real racing drone). We use the simulated data from [1]. The simulation randomized the texture of the scene background, floor and gates, the gates shape, and the lighting condition at the scene. For background and floor textures, the simulation used 40 different textures. The gate texture is drawn from a pool of 10 textures (mainly red/orange). The gates shapes are generated from a pool of 6 shapes, and for illumination conditions the background, floor and gates textures properties are being perturbed uniformly.

As can be seen in [1], despite the large appearance differences between the simulated environment to the real world, a policy that was trained in the simulation has the ability to control the drone in the real world. Thanks to the abundance of simulated data, this policy can not only be transferred from simulation to the real world, but is also more robust to changes in the environment than the policy trained with data collected on the real track.

Once a race-track was generated (from all the different possible properties that was described above), a global trajectory is being defined for the track (This will be the expert policy trajectory). This global trajectory is generated by providing a set of waypoints to pass through and a maximum velocity to achieve. When a drone is racing in the track, it is not necessary following the desired expert trajectory. Therefore, for each image the drone's forward-facing camera is taking, a label is being defined. This label is a comparison between the current drone's trajectory and desired (expert) trajectory as shown in Figure 1. The calculation of the label is done by the following procedure.

Given a drone position $p_c \in \mathbb{R}^3$, the simulation computes the closest point $p_{c'} \in \mathbb{R}^3$ on the global reference trajectory (that is denoted by t_g). The desired position $p_g \in \mathbb{R}^3$ is defined as the point on the global reference trajectory that has

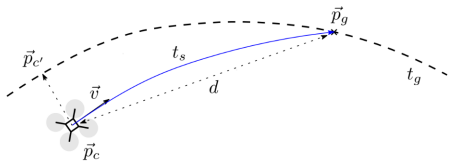

Figure 1: Computing labels by expert policy

the distance d from $\vec{p_c}$. Then, the simulation projects the desired position $\vec{p_g}$ onto the image plane of the forward-facing camera in order to generate the ground truth normalized image coordinates $\vec{x_g} \in \mathbb{R}^2$ (corresponding to the goal direction). The desired speed $v_g \in \mathbb{R}$ is defined as the speed of the reference trajectory at $\vec{p_{c'}}$ normalized by the maximum speed achieved along t_g. Eventually, each image that was captured by the drone's forward-facing camera, has a label of the form (x_g, y_g, v_g) where (x_g, y_g) are the normalized image coordinates that symbolize the goal direction on the current image, and v_g is the normalized goal speed.

We use the date that was generated by the simulation in [1]. It has in it 146 different tracks. Inside every track there are approximately 415 labeled images. We divided the data into training data and testing data. We took the first 110 tracks to be the training data, and the last 36 tracks to be the testing data. In this way, our testing data is new tracks that the drone has yet seen in its training. To summary, the training data consists of $110 * 415 \approx 45480$ labeled data, while the testing data consists of $36 * 415 \approx 13750$ labeled data (note that every track does not have precisely 415 images).

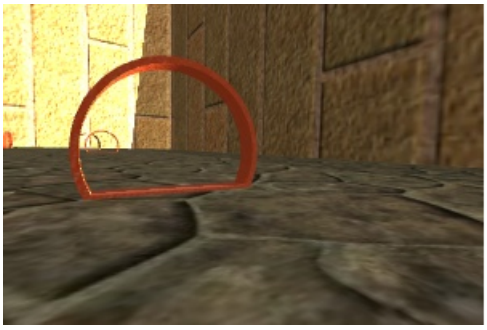

Figure 2: Example for an image that was generated in the simulation. We can see the background, floor and gate texture, the gate shape, and the different lighting condition along the track. This image has a label of the form (x_g, y_g, v_g) where (x_g, y_g) are the normalized image coordinates that symbolize the goal direction on the current image, and v_g is the normalized goal speed.

4 Method

4.1 Residual networks and MultiRes architecture

Deep neural network have shown great performances at image processing and recognition. In general deeper network are better at more intricate computational tasks. There are a couple of problems that arise when using a deeper network such as vanishing/exploding gradients that prevents convergence and training accuracy degradation that happened after convergence and results in higher training error.

Residual networks (ResNet) are neural networks in which each layer consists of a residual convolutional layers and a skip connection bypassing the residual path. These networks are simply designed and allow convergence and high accuracy even at very deep networks. We refer to the structure of the convolutional layers in each residual layer as a residual block.

Let us consider $H(x)$ as the mapping that we're trying to fit by a nonlinear architecture (such as convolutional layers with nonlinear functions), with x denoting the inputs to the first of these layers. Rather than expect the architecture to approximate $H(x)$, we explicitly let these layers approximate a residual function $F(x) = H(x) - x$. The original mapping $H(x)$ can be fitted by the residual function plus the identity function. Although both forms should be able to

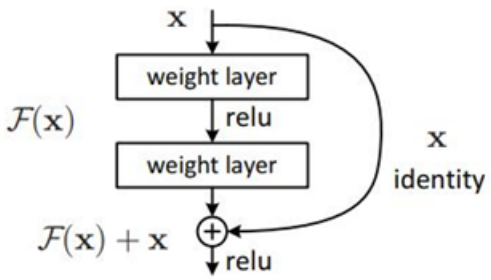

Figure 3: Residual block

asymptotically approximate the desired functions the ease of learning is different.

The degradation problem suggests that the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers. With the residual blocks structure, if identity mappings are optimal, the solvers may simply drive the weights of the multiple nonlinear layers toward zero to approach identity mappings. Thus allowing deeper networks to be at least as good as shallower networks (just approximate the deep layers to the identity function and have a network that's equivalent to the shallow network) [6].

A study by [5] shows that deep residual networks act like ensembles of relatively shallow networks. In each residual network there are exponential paths from the output layer to the input layer that gradient information can flow through while training (shown in Figure 4). Moreover the study shows that removing a layer during the test time has a modest effect on its performance. This shows that most of the gradient updates during optimization come from ensembles of relatively shallow paths.

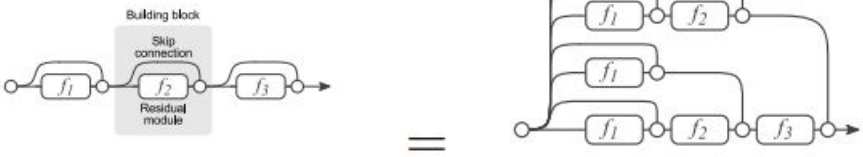

Figure 4: Conventional 3-block residual network unraveled

That motivated a new approach for residual architectures [4]. Multi-residual networks (Multi-ResNet) which increase the multiplicity of the network, while keeping its depth fixed. This is achieved by increasing the number of residual functions in each residual block, thus creating an ensemble of shallow paths.

4.2 DroNet Improvements

In order to estimate waypoints and velocity for Drone navigation using the forward-facing camera a simple convolutional neural network (CNN) was proposed by [2]. The network named DroNet predicts a goal direction in local image coordinates, together with a desired navigation speed. The main part of the network consists of a ResNet-8 architecture.

The DroNet architecture is shown in Figure 6. The Input image is forward through a convolutional layer with kernel size 5X5 and stride 2X2 and max pooling of 3X3. Then through 3 residual blocks each consist of 2 convolutional layers of kernel 5X5 and 2 ReLu function with an identity skip connection implemented by a 1X1 convolution. The output of

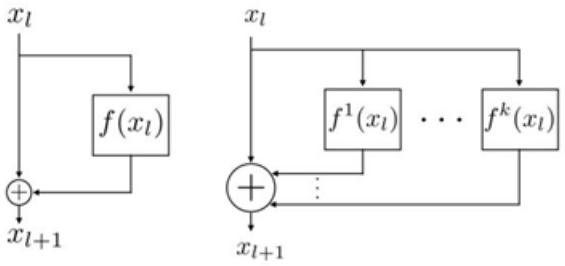

Figure 5: A residual block (left) versus a multi-residual block (right)

the Res blocks is then going through 2 fully connected layers with an additional ReLu function. The output of the net is a vector with 3 dimensions (X, Y, V).

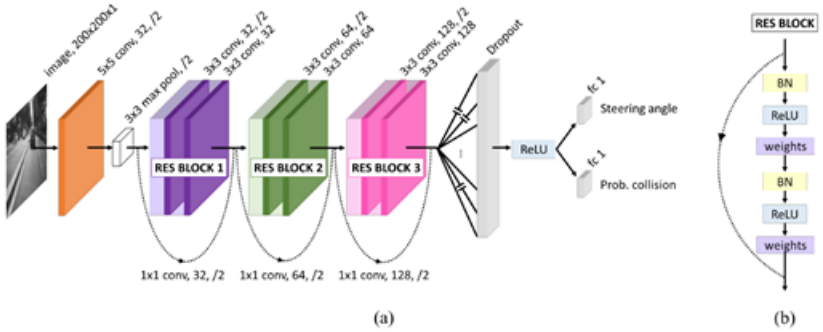

Figure 6: DroNet architecture

As discussed before, one of the ways to improve residual network performances is to add more paths for the input to go through and for gradient information back-propagate through. That can be achieved by adding more layers to the network or by changing the architecture to a MultiRes architecture. We examined both of this approaches by adding residual block in depth or in parallel to the existing architecture. We calculated the number of learnable parameters and possible paths for each architecture as shown in Table 1. The number of parameters added for a parallel res connection in all 3 res block is almost the same as the number of parameters for adding another res block in depth.

Table 1: Characteristics of different DroNet architectures

Architecture	# of paths	# of learnable parameters
DroNet	$2^3 = 8$	$\sim 164K$
MultiRes Dronet (with n parallel res connection)	$(2+n)^3$	$\sim 164K + 18,500n$
Deeper DroNet (with n additional res blocks in depth)	$2^{(3+n)}$	up to $\sim 164K + 19,500n$

5 Experiments

First, we implement in Pytorch the regular DroNet network as described in Figure 6. We use the same starting learning rate as described in DroNet training in [2], that is 0.001. In addition, we use the same batch size (200) and optimizer (ADAM). Furthermore, we tested different learning rate. As can be seen in Figure 7, different starting learning rates result in higher loss and slower convergence rate.

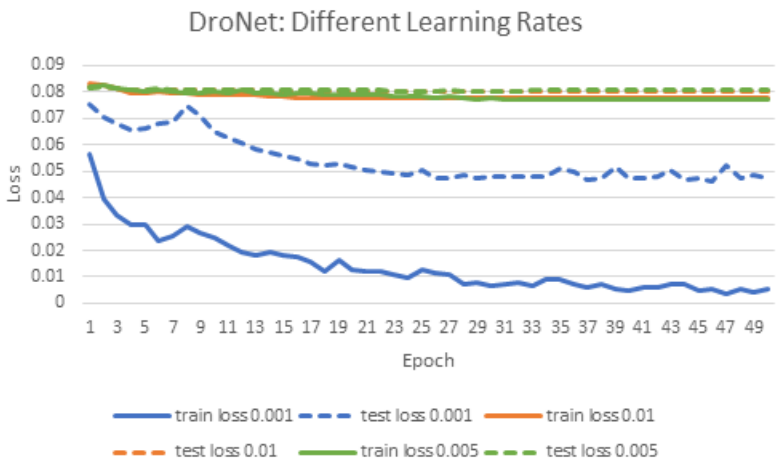

Figure 7: DroNet convergence graphs (train and test) for different starting learning rates. Note that the lower loss is for LR=0.001

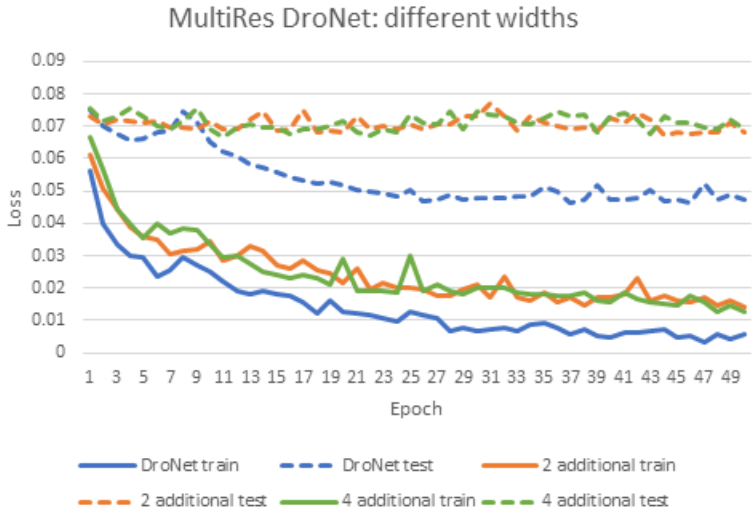

Figure 8: MultiRes DroNet convergence graphs for 2 additional residual connections and 4 additional connections compared to regular DroNet. Adding residual connections makes the network performance worse

In addition, we can see that after approximately 30 epochs the loss is entering to a plateau and decreasing rate is very low. This is important because in [1], the training procedure consists of 10 epochs only.

In order to see the influence of adding multi-residual connections, first we test adding only 2 more residual connections, adding 4 more residual connections. The results are shown in Figure 8. As one can see, adding additional residual connections does not help at all, and even make our network loss much higher.

After seeing that adding residual connections does not improve DroNet performance, checking the influence of making DroNet deeper is inevitable. In Figure 9, one can see the influence of making the DroNet deeper by 2 more Res blocks and by 4 more Res blocks. It can be seen that the convergence rate is higher as we make the network deeper. Note that the final loss is still the same, i.e performance does not change, only the optimization is faster.

Taking these two approaches in our case - additional width and additional depth, we see that additional width results with difficulties in optimization, that prevents the network from converging. Additional depth did improve the optimization and made the convergence rate faster. Also both approaches increased the number of paths the input go through and the gradient information back-propogate it did not improved the network performances. The optimization problem in the additional width approach can be explained by insufficient depth of DroNet. By [4], every different architecture has a

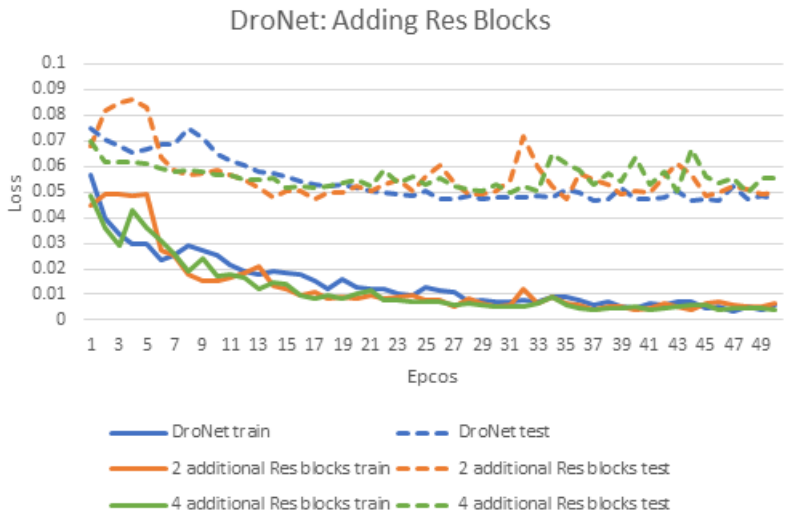

Figure 9: DroNet with additional Res blocks convergence graphs for 2 additional Res block and 4 additional Res blocks compared to regular DroNet

minimal depth $n0$ that beneath it, additional width would not improve performance at all, and even makes it worse. By [4], *"one might need to choose a suitable number of residual functions, and depth to achieve the best performance"*. Therefore, we conclude that resnet8 is not deep enough for improvement by additional width, and width of one is the optimal case for this network.

6 Conclusions

In our work, we test the implications of MultiRes architecture on residual networks, and in our case - DroNet. We implemented the basic DroNet architecture in Pytorch, and extended this implementation for MultiRes DroNet. In addition, we test the implications of additional layers for DroNet (adding ResBlocks). We have not been able to improve the performance with these two approaches. The multiRes approach made the performance worse. Probably because DroNet is too shallow. Possible way to improve DroNet is adding width and depth together. In this way, the network will have sufficient depth for additional residual connections. The disadvantage of this possible way is the high computational cost, that does not suit for on board computation in drone racing. The main reason is that drone racing requires high speed flight and quick decision making. Therefore, we see in simple DroNet a good network that has good accuracy while maintaining cheap computational cost.

A completely different approach for improving on board network is using reinforcement learning. This was our original goal for this project, but it required online training (i.e. drone simulation or real drone flight). The UZH robotic and perception group simulation can be very useful for training by this approach. We corporated with the drone lab in TAU, and even connected the UZH lab in order to use the simulation on TAU servers. Unfortunately, we have not succeeded using the simulation online, only offline - therefore we could not simulate each policy result, and therefore could not use reinforcement learning. We recommend testing this approach, that may result in better performances (without high computational cost).

7 Appendix

The original DroNet network was implemnted in Tesorflow and can be found at: `https://github.com/uzh-rpg/sim2real_drone_racing`.

Our code was uploaded to a Github repository, and can be found at: `https://github.com/idanbasre/DroNet.git`. The repository contains a Pytorch implementation of DroNet network (with additional Res blocks and without additional Res blocks), and MultiRes network.

References

[1] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun and D. Scaramuzza. Deep Drone Racing: From Simulation to Reality With Domain Randomization, In *IEEE Transactions on Robotics 36.1 (2020): 1–14. Crossref. Web.*

[2] A. Loquercio, A. I. Maqueda, C. R. del-Blanco and D. Scaramuzza. DroNet: Learning to Fly by Driving, In *IEEE Robotics and Automation Letters, vol. 3, no. 2 on*, 1088-1095, April 2018

[3] Elia Kaufmann and Antonio Loquercio and Rene Ranftl and Alexey Dosovitskiy and Vladlen Koltun and Davide Scaramuzza. Deep Drone Racing: Learning Agile Flight in Dynamic Environments, *arXiv:1806.08548*, Conference on Robotic Learning (CoRL), 2018.

[4] Masoud Abdi and Saeid Nahavandi. Multi-Residual Networks: Improving the Speed and Accuracy of Residual Networks, *arXiv:1609.05672*, 2016.

[5] Andreas Veit and Michael Wilber and Serge Belongie. Residual Networks Behave Like Ensembles of Relatively Shallow Network, *arXiv:1605.06431*, 2016.

[6] Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun. Deep Residual Learning for Image Recognition, *arXiv:1512.03385*, 2015.