Effect of Feature Map Resolution on Reproducing Results of DeepMind's StarCraft II Reinforcement Learning

Ali Abbas
University of Central Florida
Orlando, FL, US
ali.abbas@knights.ucf.edu

ABSTRACT

Google's DeepMind was hugely successful in applying Reinforcement Learning (RL) techniques to solve Go and Atari games, but StarCraft remained an open challenge due to its vast action and state spaces with imperfect information. However, DeepMind made some progress on StarCraft with their paper "Starcraft II: A new challenge for reinforcement learning", using Asynchronous Advantage Actor Critic (A3C) RL. I reproduced some of the results of DeepMind using a lower feature map resolution of 32*32 pixels, using Reaver, a modular deep RL open source framework, which uses Advantage Actor Critic (A2C) RL. I compared my results with Reaver's results which was obtained by using 16*16 pixels feature map resolutions and DeepMind's results which was obtained with 64*64 pixels feature map resolutions. I was able to get similar results to Reaver in most cases and similar results to DeepMind in a few cases. I found that the effect of increasing resolution was to make the algorithms more sample efficient but generally did not lead to better results.

Keywords

Reinforcement learning; machine learning; Starcraft ii; Advantage actor critic.

1. INTRODUCTION

StarCraft II (SC2) is a real-time strategy (RTS) game that has become an excellent testbed for developing new AI algorithms, due to its complexity while still being relatively easy to simulate many games in a short period of time. Some of the challenges posed by SC2 especially to Reinforcement Learning (RL) techniques include the huge number of possible actions and states, partial map observability, and credit assignment problem. Recently Google's DeepMind released a set of tools for tackling these problems using RL which made progress in this field [1]. However despite releasing their tools, they did not release the complete source code for the experiments they performed. This encouraged other researchers to try to reproduce DeepMind's results and release their own open source frameworks. In this paper, I will use Reaver [2], a modular deep RL open source framework that was created with DeepMin's tool sets, to reproduce their results while studying the effect of feature map resolution on the results.

The remainder of the paper is structured as follows. In section 2, there's a brief review of related work. Section 3 discusses the method. In section 4, experimental results are shown and discussed. Finally, section 5 concludes the paper and suggests some directions for future work.

2. RELATED WORK

Recently, RL methods have made significant progress on the difficult task of playing StarCraft II and Google's DeepMind group was at the forefront of this progress.

Google's DeepMind (Vinyals et al., 2017) released StarCraft II Learning Environment (SC2LE), an open source tool for developing RL methods for StarCraft II, written in python. They proposed and tested the Asynchronous Advantage Actor Critic (A3C) RL with different agent architectures including Random, Atari-net, FullyConv, and FullyConv LSTM agents. Their reported results on a series of mini games were comparable to a beginner-level human player [1].

3. METHOD

I used Reaver [2], a modular deep RL open source framework, which uses Advantage Actor Critic (A2C) RL to reproduce the results of [1]. The following sections will go into detail about the used methods.

3.1 Advantage Actor-Critic (A2C)

Advantage Actor-Critic (A2C) is a synchronous version of the Asynchronous Advantage Actor-Critic (A3C) that was used in [1]. Actor-Critic is a family of reinforcement learning (RL) algorithms that combine the benefits of both value-iteration and policy-gradient methods. The actor represents the policy-gradient part which takes the state as input and outputs the action to be performed. The critic represents the value-iteration part which takes the action and state as input and outputs the value function. Usually, neural networks are used for both actor and critic parts.

Actor: $\pi(a_t \mid s_t; \theta)$	π is a policy function. θ is the actor's neural network parameters
Critic: $A(s_t, a_t; \theta, \theta_v)$	A is the advantage function. θ_v is the critic's neural network parameters

Equation 1. Actor and critic definitions [3].

A problem for the basic policy gradient algorithms (Algorithm 1) is the high variance introduced by using only the return value R_t. Actor-Critic solves this problem by subtracting a baseline value $b(s_t)$ to decrease this variance and improve the gradient search [4]. In A2C and A3C the advantage estimate is given by Equation 2.

$$\hat{A}_t = A(s_t, a_t; \theta, \theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v)$$

Equation 2. Advantage estimation function [3]

Where γ is the discount factor, k is how many future states should be considered, and V is the value of the state estimated by the critic using θ_v parameters of its neural network [3]. The advantage estimate tells us how much better or worse an action is compared to the expected value.

Initialize policy parameter θ, baseline b
for iteration=1, 2, . . . **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$.
   (Plug $\hat{g}$ into SGD or ADAM)
end for

Algorithm 1. The basic policy gradient algorithm [5].

The difference between A3C and A2C is that A3C uses parallel agents, each with its own network and environment that updates the global network independently, but in A2C there are multiple copies of the same agent with added noise and the updates generated by the agents are aggregated and the global network is updated once by this aggregation of updates. One advantage of A3C is that it allows a better exploration of the environment since each agent can have different network parameters and policies. A2C achieves a similar exploration goal by adding noise to each copy of the agents, so each agent explores a different part of the environment [5].

3.2 StarCraft II Learning Environment (SC2LE)

SC2LE released by Vinyals et al. (2017) has three main components: a Linux StarCraft II binary, the StarCraft II API, and PySC2. The overall structure is shown in Figure 1. The StarCraft II API provides a programming interface to StarCraft II binary for controlling the game, performing actions, and reading game states. PySC2 is a set of Python tools that makes it easier for RL agents to deal with the API. Each RL agent receives four set of features: Non-spatial features such as available actions or resources, screen features such as units and structures displayed in main screen, minimap features such as units and structures of the whole map that are displayed in the minimap, and the reward such as enemy units destroyed or collected resources. Then the agent learns the proper action through A2C and then sends the selected action (for example building a structure) to PySC2 API component [1].

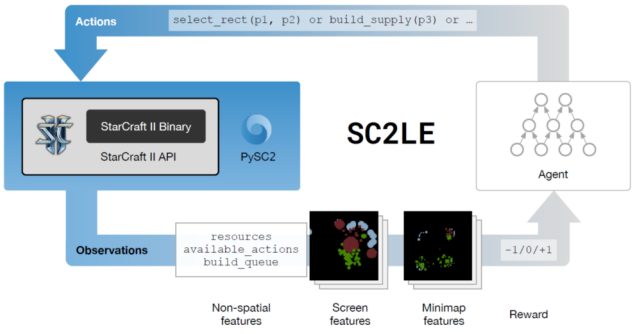


Figure 1. SC2LE structure [1].

PySC2 provides visualization of the features as they are seen by the agent. Figure 2. Shows an image of a PySC2 StarCraft viewer used to display this information for the "CollectMineralsAndGas" mini game. The left panel is only meant for a human understandable overview of the game. The right panel is the screen and minimap features used by the RL agent.

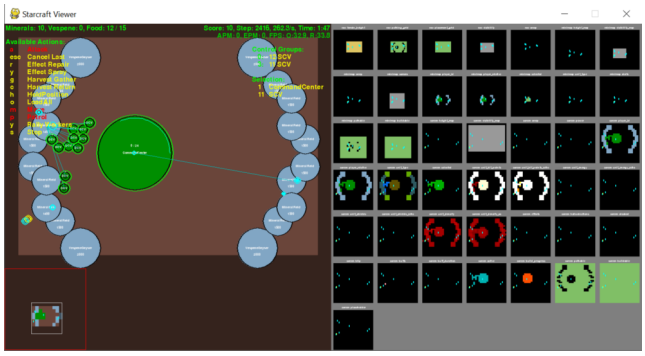


Figure 2. PySC2 StarCraft viewer for the CollectMineralsAndGas mini game.

3.3 Reaver, a Modular Deep RL Open Source Framework

Reaver [2] was developed using SC2LE to reproduce the DeepMind's results reported in [1], and also to provide an easy to extend framework for developing different RL methods for SC2. Reaver uses the FullyConv architecture that was used in [1] as its main agent architecture. Unlike most convolutional neural networks, FullyConv preserves the input resolution in order to not lose any spatial information since each pixel in an SC2 frame can contain valuable information such as location of enemy units or their movements. FullyConv is used to extract the spatial and non-spatial features and also used by A2C (or other RL agents) as function approximator to find the best policy. Due to computational power restrictions, Reaver uses 16 pixel resolution for screen and minimaps features, while DeepMind used 64 pixels. Thus in some mini games, their results were worse than DeepMind.

4. EVALUATION

In this section, I describe the results I obtained using Reaver using a different resolution for feature maps and compare it to DeepMind results reported in [1] and Reaver results reported in [2]. The results were obtained on these DeepMind mini games: MoveToBeacon, CollectMineralShards, DefeatRoaches, DefeatZerglingsAndBanelings, and CollectMineralsAndGas [1].
I use the same hyperparameters such as learning rate, loss weights and batch size, as used by Reaver's default values except for one important parameter: the screen and mini map resolution. DeepMind used 64*64 pixels resolution [1], while Reaver used 16*16 pixels resolution [2] due to limited computational resources. While Reaver was able to get close to DeepMind results in some mini games, it performed much worse on others. To get a better understanding of the impact of the resolution, I

Map	My Experiments	Reaver (A2C)	DeepMind	Human Expert
MoveToBeacon	25.5 (2) [22, 30]	26.3 (1.8) [21, 31]	26	28
CollectMineralShards	102.1 (8.1) [49, 117]	102.8 (10.8) [81, 135]	103	177
DefeatRoaches	72.6 (54.9) [11, 294]	72.5 (43.5) [21, 283]	100	215
DefeatZerglingsAndBanelings	57 (19.2) [16, 113]	56.8 (20.8) [21, 154]	62	727
CollectMineralsAndGas	3,070.3 (274.5) [2,250, 3,555]	2,267.5 (488.8) [0, 3,320]	3,978	7,566

Table 1. Best average score comparison between My Experiments, Reaver, DeepMind, and Human Expert. Numbers in parentheses are standard deviation, and numbers in square brackets are [min, max] [2].

	My Experiments		Reaver (A2C)	
Map	**Samples**	**Episodes**	**Samples**	**Episodes**
MoveToBeacon	1,049,600	4,372	563,200	2,304
CollectMineralShards	43,750,400	182,279	74,752,000	311,426
DefeatRoaches	157,235,200	1,386,618	172,800,000	1,609,211
DefeatZerglingsAndBanelings	4,582,400	117,240	10,496,000	273,463
CollectMineralsAndGas	27,929,600	33,248	16,864,000	20,544

Table 2. Comparison between My Experiments and Reaver in terms of training details [2].

used 32*32 pixels resolution to understand the impact of resolution and to see if I could improve on Reaver's results while still requiring much less computational power than DeepMind.

For my experiments, SC2 4.1.2.60604, Python 3.6, Tensorflow 2.1, CUDA 10.1, PySC2 3.0.0, and Reaver 2.1.9 were used. I rented two servers from Vast.ai to run the experiments, with the following specifications: Ubuntu 18.04 server with Xeon E5-2680 v2 CPU, 16 GB RAM, and NVIDIA GeForce GTX 1080 Ti GPU. Adam optimizer with learning rate 0.0007 with batch size 16 were used in both Reaver and my experiments. Both I and Reaver, truncate the trajectories after 16 steps, while DeepMind truncates it after 40 steps. Deepmind ran 64 parallel threads for agents and ran 100 experiments with random hyperparameters for 600 million game steps (samples) and reported their best of the run results. Doing a fair comparison with DeepMind requires huge computational resources, so instead Reaver trains agents in 32 parallel environments and reports their average score while the run continued until it was close to DeepMind's results or a set maximum time was reached. I followed the Reaver approach (except for 32*32 resolution instead of 16*16) and Table 1. compares my experiments' results to Reaver, DeepMind, and Human Expert. Table 2 compares the number of episodes and samples used in training between my experiments and Reaver's experiments as reported in [2]. As it can be seen in Table 2, except for MoveToBeacon and CollectMineralsAndGas, I used less episodes and samples than Reaver. Table 1 shows that for all experiments except CollectMineralsAndGas, the results are not significantly different from Reaver. Also we can see that compared to DeepMind, Reaver gets similar results for MoveToBeacon and CollectMineralShards, and close results for DefeatZerglingsAndBanelings, but significantly worse results for DefeatRoaches and CollectMineralsAndGas. My results follow the same general trend as Reaver, except for CollectMineralsAndGas, which I got significantly better results than Reaver and much closer to DeepMind. Human Expert does significantly better than all other methods which show the enormous challenge that SC2 poses to RL methods.

Figure 3. Shows the average training rewards per sample for the MoveToBeacon mini game. For the DeepMind results, we're only interested in the blue line which is a FullyConv agent that also is used in my experiments and Reaver's experiments. Unfortunately since DeepMind's sample size of 600 million is huge compared to 1 million samples in my experiments, we can't see the effect of early samples in DeepMind and we only see a quick rise and a flat line. Comparing my results with Reaver, it seems that increasing the resolution made it harder for the agent to find the best policy quickly since there are more possible actions to be considered, as can be seen in the lagged rise in my experiment. Also from Table 2 we can see that my experiment needed more samples to get a comparable score to Reaver. In both cases, initially the agent performs poorly due to random actions but once it finds a good move, it quickly optimizes that action in a few steps. In this mini game higher resolution had a negative effect on the convergence speed but no significant effect on the final results.

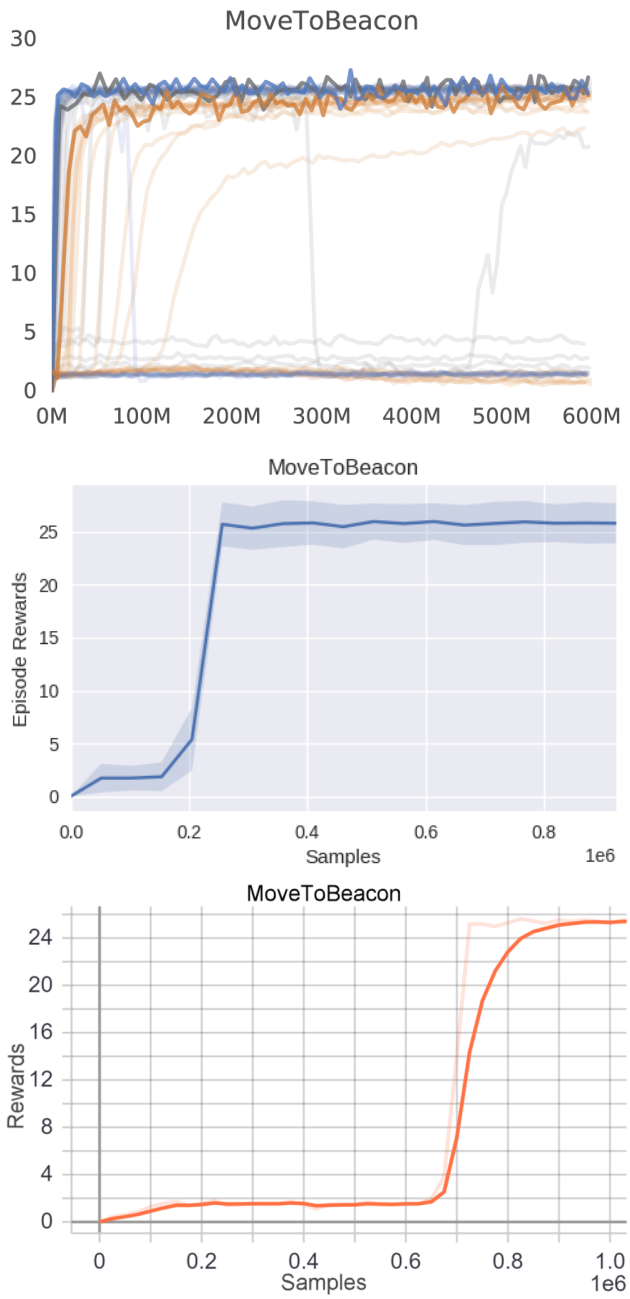


Figure 3. MoveToBeacon average training rewards per sample. Top graph is DeepMind (blue line) [1], the middle is Reaver [2], and the bottom is my experiment.

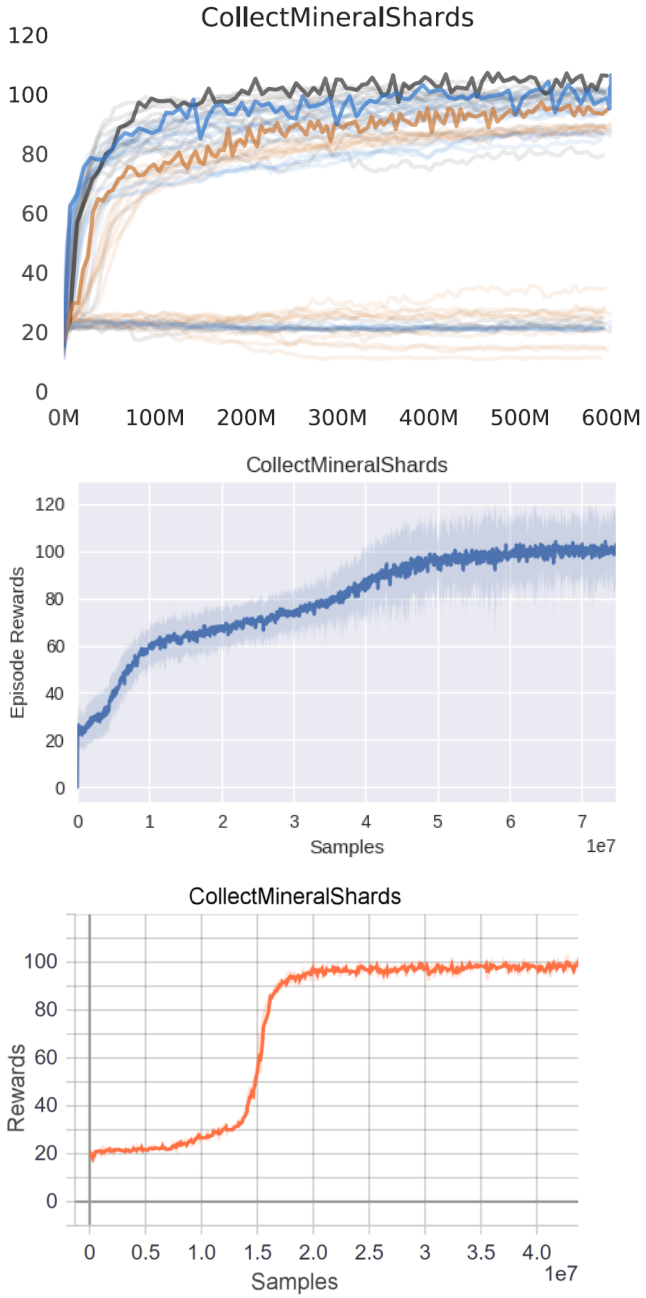
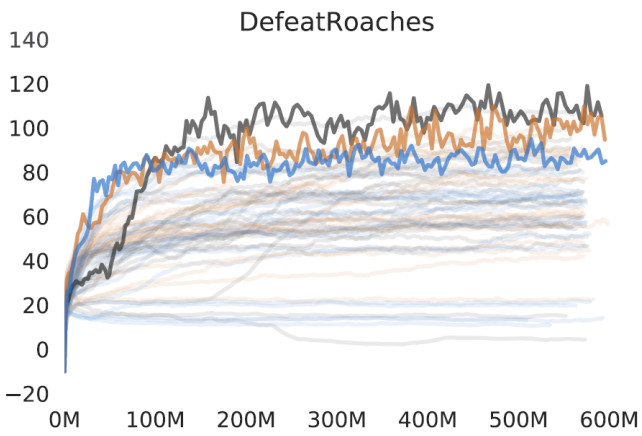
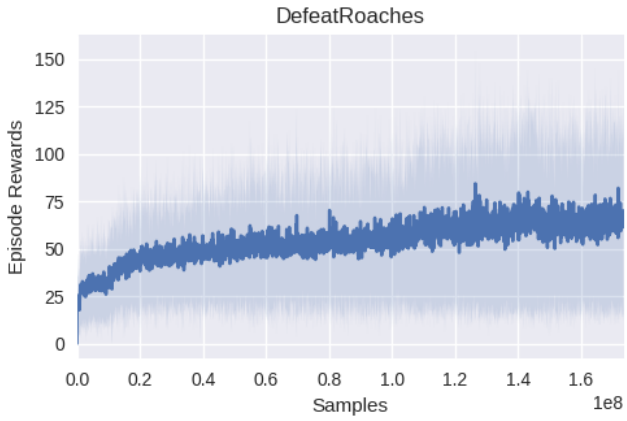


Figure 4. CollectMineralShards average training rewards per sample. Top graph is DeepMind (blue line) [1], the middle is Reaver [2], and the bottom is my experiment.

Figure 4. Shows the average training rewards per sample for the CollectMineralShards mini game. All graphs show a slowly increasing curve with somewhat staircase shaped increase. Although in DeepMind, the staircase shape is barely recognizable in sub 100M part. This is mostly due to getting trapped in local optima for some time. Reaver's graph shows two local optima around 20 and 60 scores and a slow increase toward the 100 score, while my experiment stays longer in 20 score local optima, it skips the 60 local optima and converges to the 100 score quickly. Also from Table 2. We can see that my experiment required far less samples than Reaver to achieve similar results. In this mini game, higher resolution had a positive effect on the convergence speed but small effect on the final results.

Figure 5. Shows the average training rewards per sample for the DefeatRoaches mini game. All graphs show a slowly increasing curve per sample that flattens at the end. There are no significant differences between my results and Reaver, which could suggest that the better DeepMind results are either due to increased number of samples or better hyperparameters and not mainly due to higher resolution. Also from Table 2. We can see that my experiment required less samples than Reaver to achieve similar





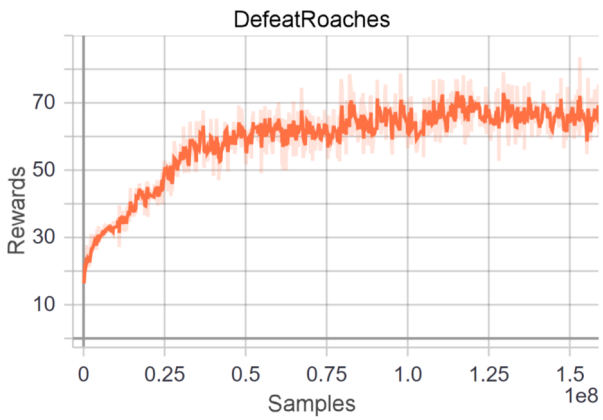
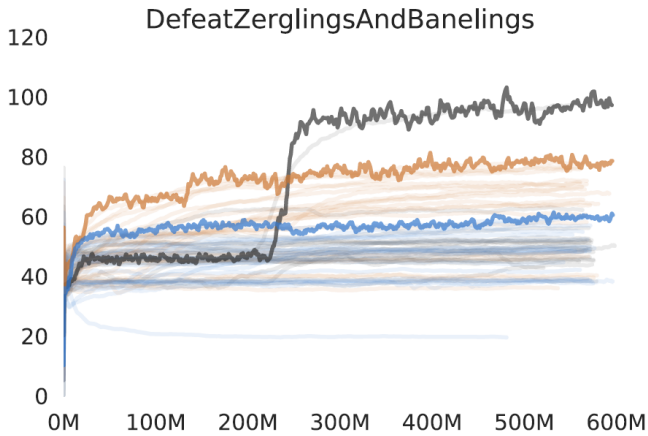
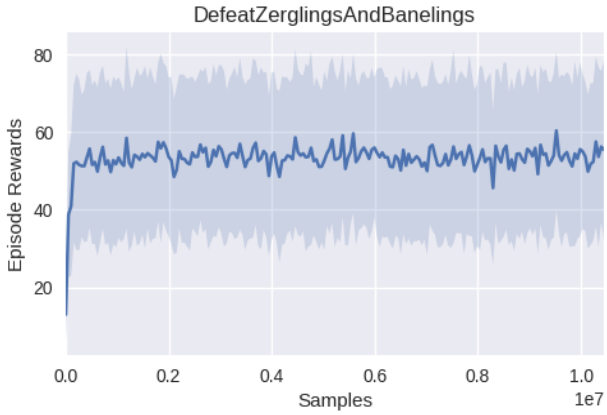


Figure 5. DefeatRoaches average training rewards per sample. Top graph is DeepMind (blue line) [1], the middle is Reaver [2], and the bottom is my experiment.





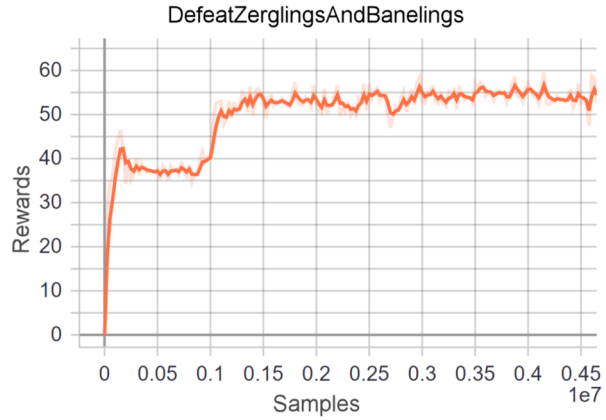


Figure 6. DefeatZerglingsAndBanelings average training rewards per sample. Top graph is DeepMind (blue line) [1], the middle is Reaver [2], and the bottom is my experiment.

results. In this mini game higher resolution doesn't seem to have a significant effect apart from requiring less samples.

Figure 6. Shows the average training rewards per sample for the DefeatZerglingsAndBanelings mini game. All graphs show a quick rise to their best score quickly except for my experiment where it gets trapped in 40 score local optima for a short time but then rises quickly to its best score. Also from Table 2. We can see that my experiment required far less samples than Reaver to achieve similar results. In this mini game, higher resolution didn't have a significant effect apart from the possibility of getting stuck in local optima for a short time.

Figure 7. Shows the average training rewards per sample for the CollectMineralsAndGas mini game. DeepMind graph shows a quick rise to a local optima around 3500 and then a quick rise to its best score but showing some back and forth between local optima and best score. Reaver quickly rises to 2000 score but then oscillates between 0, 500, 1500, and its best score. My results oscillate between 0, 1000, and best score while it spends a significant amount of time at 0 which was not the case for others.

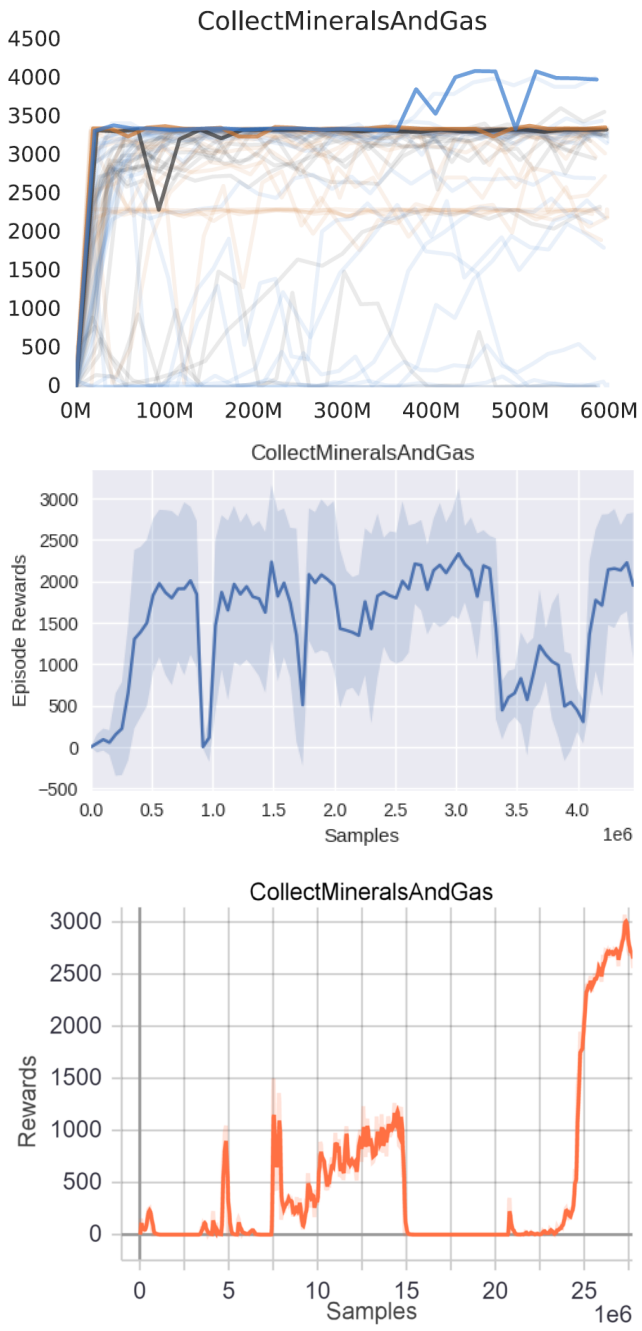


Figure 7. CollectMineralsAndGas average training rewards per sample. Top graph is DeepMind (blue line) [1], the middle is Reaver [2], and the bottom is my experiment.

For both Reaver and my experiment, it seems challenging to hold on to the progress made, although the same behavior is to a small degree is seen in DeepMind when it falls from near 4000 to 3500. My experiment required more samples than Reaver to reach a significantly higher score, which makes it hard to attribute this success to the higher resolution rather than higher samples. Although it's also not that simple to say my experiment did worse for the same amount of samples, since higher resolution introduces larger action and state spaces which might require higher samples. Also it could be the case that the lower resolution destroys some important information such that no amount of samples can lead to the best possible solution.

In general, higher resolution effects varied depending on the mini game, which is to be expected since each mini game has various levels of dependency on spatial resolution. Simpler mini games such as MoveToBeacon don't seem to benefit from higher resolutions and even require more samples to get good results. Higher resolution in most mini games required less samples, since it has more information in each feature map and the game is complex enough that useful information can be extracted from the higher resolution feature maps, but it did not have a significant effect on the best scores. One notable exception was CollectMineralsAndGas but as was discussed, it's hard to say if its better performance under higher resolution was due to higher number of samples or not.

5. CONCLUSIONS

DeepMind provided the AI research community with an excellent set of tools to test their RL methods for SC2. I used the Reaver framework to test the effect of feature map resolution on reproducing the DeepMind results. I used a 32*32 pixels resolution and compared my results with the Reaver which used 16*16, and DeepMind which used 64*64 resolutions. The results showed that in general, higher resolution can lead to good results while requiring less samples but did not have a significant effect on getting better scores. Requiring less samples is a highly desirable feature, especially in areas where getting samples are difficult, for example in applying RL to real world robotics.
For future work I suggest using 600 million samples with different hyperparameters for each resolution to be able to get a more accurate picture of the effect of resolution in solving SC2 mini games.

6. REFERENCES

[1] Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., ... & Quan, J. (2017). Starcraft ii: A new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782.

[2] Ring, R. (2018). Reaver: StarCraft II Deep Reinforcement Learning Agent. GitHub repository. https://github.com/inoryy/reaver.

[3] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937).

[4] Seita, D. (2018). Actor-Critic Methods: A3C and A2C. https://danieltakeshi.github.io/2018/06/28/a2c-a3c.

[5] A Clearer and Simpler Synchronous Advantage Actor Critic (A2C) Implementation in TensorFlow. https://github.com/MG2033/A2C.