Learning Game Map Representations using Deep Convolutional Generative Adversarial Networks

Roshan Sridhar
Supervised by Philip Bontrager, Julian Togelius
Game Innovation Lab
New York University

December 2017

1 Introduction

This project aims to generate heightmaps for a popular real-time strategy game StarCraft II using deep convolutional generative adversarial networks (DC-GANs). The network is trained on existing maps and uses the generator to produce new maps. The reason for this approach is that DCGANs prove to be a simple and attractive way to learn good representations of images. In the future, this would also help improve resource locations predictions for AI-assisted tools during gameplay (Lee, Isaksen, Holmgård, and Togelius 2016).

2 Data

The data for this project consists of maps authored by the developers of the game Blizzard, since they would have a high inclination to design quality maps. This provides a good baseline and a stable dataset to work with. This motivation for my decision was influenced from [2].
There are 164 map files collected from both the base game and its expansions. The heightmaps were obtained from authors of [2]. These maps were then rotated by 90, 180 and 270 degrees and added to my dataset which resulted in a total 656 images of training data. Figure 1 shows a sample of the data.

It is worth mentioning that the maps were not all symmetric. To overcome this problem, the maps were centered and the sides were padded with untraversable terrain. The heightmaps files were downloaded, decompressed and made consistent by [2]. Figure 2 shows an example of this.

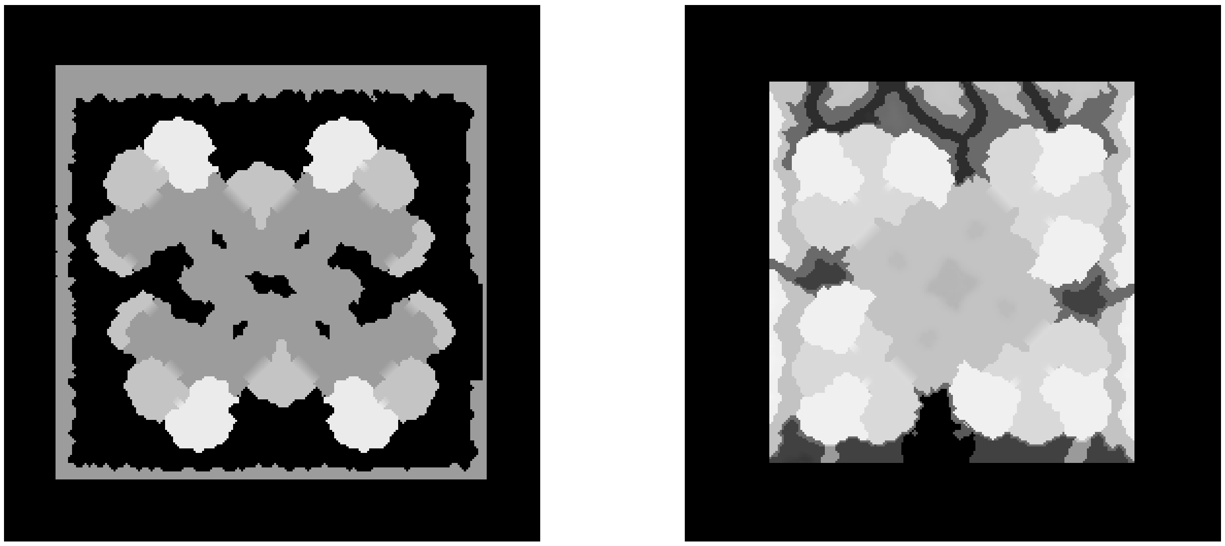

Figure 1: StarCraft II heightmaps

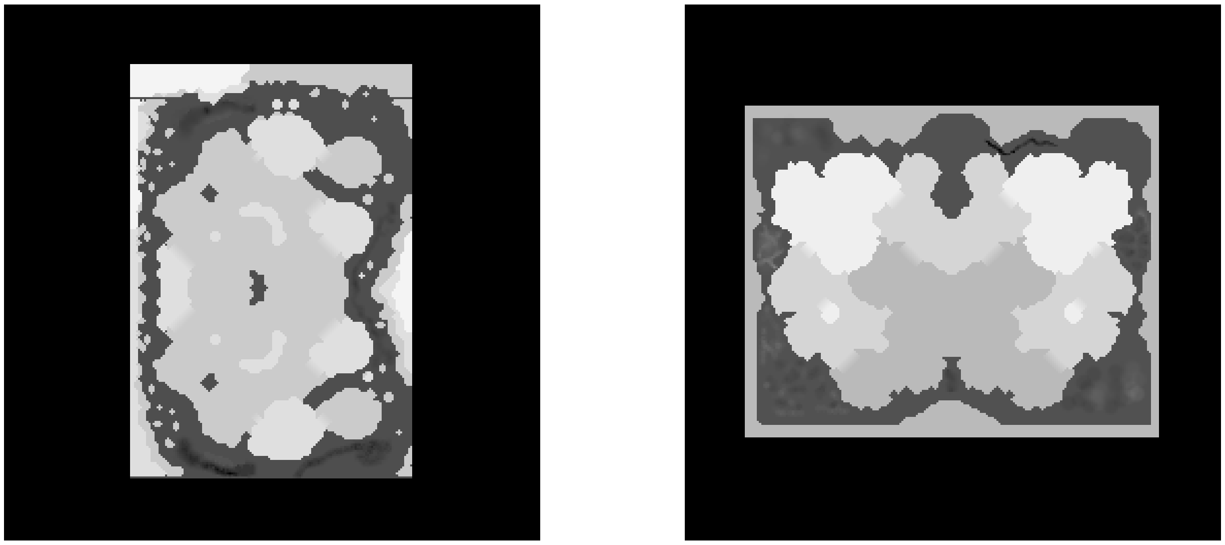

Figure 2: Examples of map modifications

3 Method

The network employed to generate new maps is a class of CNNs called deep convolutional GANs (DCGAN). DCGANs demonstrate good ability in learning hierarchical representations of images [3].
This system contains two networks, a generator and a discriminator. The discriminator takes in real and generated images, performs a series of convolutions with multiple filters, to provide a single scalar output that states the probability estimate of whether the fed image is real or fake. The generator network takes a noise distribution as an input and tries to replicate the training data. During training, the generator tries to maximize the probability of the discriminator classifying the generated image as real. This is analogous with the system playing a two-player minimax game. [1].

3.1 Architecture

The system takes images and a noise vector as the discriminator and generator inputs respectively. A 64x64 image of a map is fed as the input to the discriminator network. I decided to work with downsized images to improve memory management during training. The images would be resized up later to portray full maps.
The discriminator network consists of five convolutional layers with a kernel size of 4x4. The first four layers have a stride of 2 and a padding of 1, while the last layer has a stride of 1 and no padding. A *LeakyReLU* activation function follows all the layers except the output, which uses a *Sigmoid* activation function after it. Details are found in Table 1.

Table 1: Discriminator Network Architecture

Layer	Activation	Kernel Size	Output Channel(s)
Convolution	LeakyReLU	4x4	64
Convolution	LeakyReLU	4x4	128
Convolution	LeakyReLU	4x4	256
Convolution	LeakyReLU	4x4	512
Convolution	Sigmoid	4x4	1

Similarly, the generator network also consists of five layers and a kernel size of 4x4. The first layer has a stride of 2 and no padding while the rest of the layers have a stride of 2 and a padding of 1. *ReLU* activation follows all the layers except the output, which uses *Tanh*. This is described in Table 2.

Table 2: Generator Network Architecture

Layer	Activation	Kernel Size	Output Channel(s)
Transposed Convolution	ReLU	4x4	512
Transposed Convolution	ReLU	4x4	256
Transposed Convolution	ReLU	4x4	128
Transposed Convolution	ReLU	4x4	64
Transposed Convolution	Tanh	4x4	3

3.2 Training

Layer parameters like kernel size were chosen experimentally. The system was trained for 10000 epochs with a learning rate of 0.0005.

4 Results

The following images were output by the generator after training the system on 656 samples. Figure 3 shows some of the best results obtained. Figure 4 shows an example of the output batch.

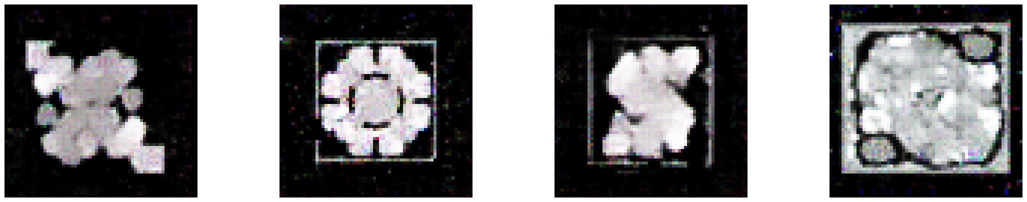

Figure 3: Select outputs

See Figure 4 below for the batch output.

5 Conclusion

Although some results are satisfactory, there still exist areas for continued development.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

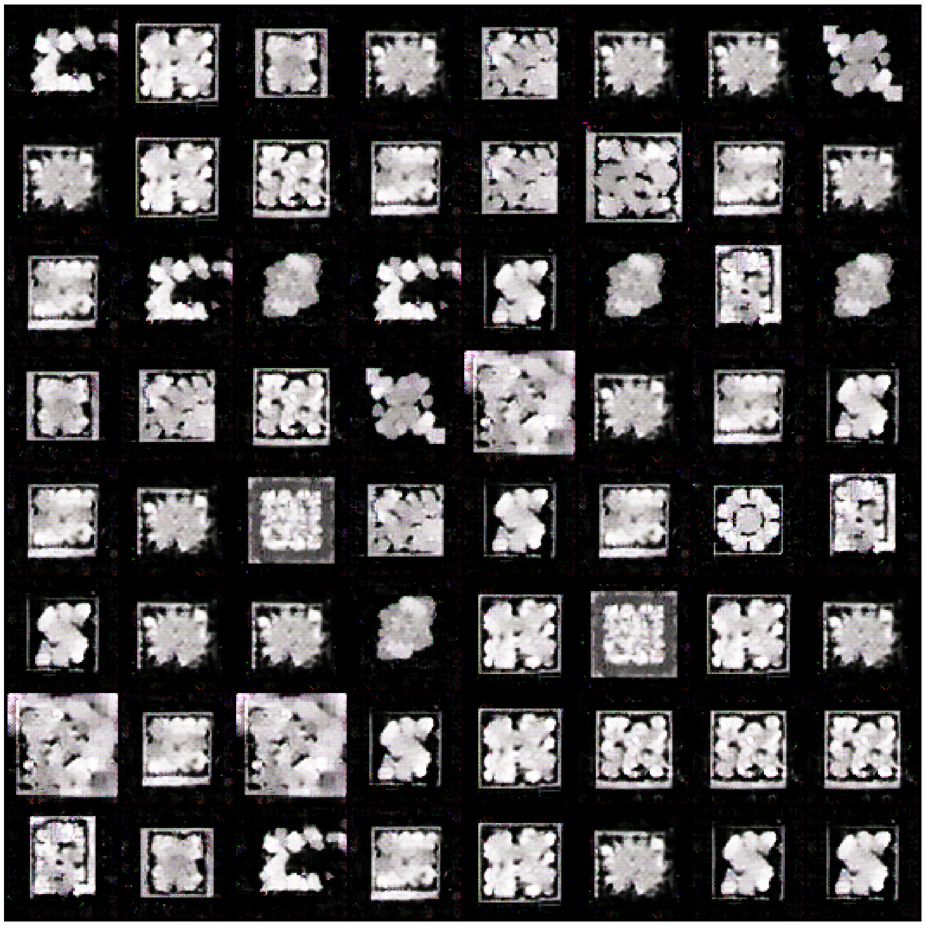

Figure 4: Batch of outputs

[2] Scott Lee, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. Predicting resource locations in game maps using deep convolutional neural networks. In *The Twelfth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. AAAI*, 2016.

[3] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.