Advanced Technologies Task 2 (DirectX11 Deep Learning Textures)

Drew Mayhew
17027805

University of the West of England

May 7, 2020

The second assigned task for Advanced Technologies was to create a Deep Learning Neural Network. Throughout development, the project changed scope and switched back and forth ideas several times, however the end goal of outputting the same product remained the same.

1 Introduction

The creation of a deep learning neural network was a challenge as this was completely new territory, however the prospect was exciting. The project's end goal was to successfully be able to read a set of training data, over time learn about the dataset and then output the end product into Unity.

2 Related Work

On 05/12/2017 DeepMind developed AlphaZero, a deep learning machine which taught itself from scratch how to master chess, shogi (Japanese chess), and Go, in under 24 hours beating world-champion programs in each case. The preliminary results showed a highly dynamic and unconventional style of play, defeating world-champion programs Stockfish, Elmo, and the 3-day version of AlphaGo Zero, as seen in Figure 1.

AlphaZero was trained solely via "self-play" using 5,000 first-generation tensor processing units (TPUs) to generate the games alongside 64 second-generation TPUs to train the neural networks. After four hours of training, DeepMind estimated AlphaZero was playing at a higher Elo rating than Stockfish 8; after 9 hours of training, the algorithm defeated Stockfish 8 in a time-controlled 100-game tournament (28 wins, 0 losses, and 72 draws). The trained algorithm played on a single machine with four TPUs. Sarah Knapton, 2017

The DeepMind team eventually want to use the algorithm to solve big health problems believing that one day the program could discover cures for major illnesses in a matter of days or weeks, which would have taken humans hundreds of years to find.

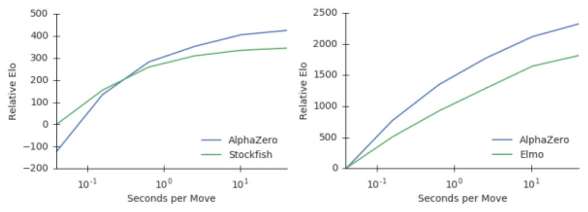

Figure 1: *Scalability of AlphaZero with thinking time, measured on an Elo scale. a Performance of AlphaZero and Stockfish in chess, plotted against thinking time per move. b Performance of AlphaZero and Elmo in shogi, plotted against thinking time per move. Silver and Hassabis, 2018*

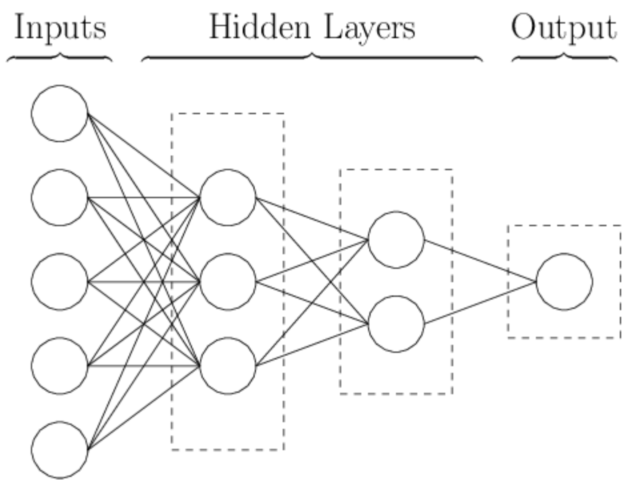

Figure 2: *Neural Network. Lima, 2018*

3 Method

How Neural Networks Work: A Neural networks are deep learning machines which are able to learn processes by reading in huge amounts of data, learning and advancing over time. Such systems learn to perform tasks without being programmed with task-specific rules. For example, in image recognition, a machine can be trained to identify images containing cats by first analyzing images manually labeled as "cat" or "not a cat", identifying characteristics from the examples as they process. After the machine has read through several thousand images of cats and not cats, the machine should have gained the ability to distinguish whether cats are present or not in selected images, to varying accuracy depending on how long it was trained.

Within a neural network there are three different types of layers, depicted in Figure 2:
1. **Input Layer**, receives input data.
2. **Hidden Layer(s)**, perform mathematical computations on our inputs.
3. **Output Layer**, returns the output data.

The “Deep” in Deep Learning refers to having more than one hidden layer which links to one of the most difficult challenges when creating neural networks, deciding upon the number of hidden layers, as well as the number of neurons for each layer. Neural Networks learn by streaming through large amounts of data thousands of times. Once this has been done a function needs to be created to show how wrong the AI’s outputs were from the original inputs. This function is called the Cost Function.

Reducing the cost function, leads to more accurate results so it is in the AI’s best interest to narrow these down to the best of its abilities, this is done by changing the weights between neurons. One option is to randomly change weightings manually until the cost function is low enough however, this is inefficient. Instead a technique called Gradient Descent is utilised. Gradient Descent is a technique which allows the AI to find the minimum cost function by changing the weights in small increments, after each data set iteration, as seen in Figure 3. By computing the derivative (or gradient) of the cost function at a certain set of weight, we’re able to see in which direction the minimum is.

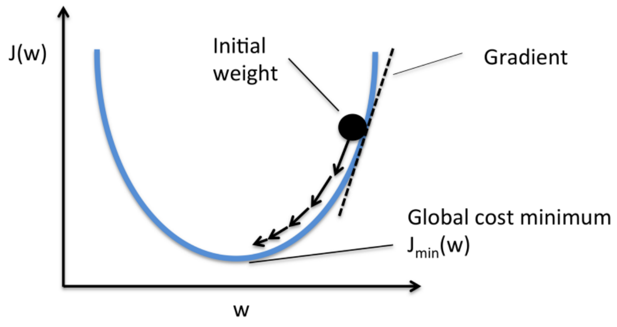

Figure 3: *Reducing Cost. Raicea, 2017*

Gradient descent is also used in backpropagation, which calculates the gradient of the error function with respect to the neural network’s weights. The name stems from the fact that gradient calculations proceed backwards through the network, with the gradient of the final layer of weights being calculated first, and the first layer calculated last. This backwards flow of error information allows for efficient computation of the gradient at each layer, compared to calculating the each layers gradient separately.

Backpropagation’s popularity has experienced a recent resurgence given the widespread adoption of deep neural networks for image recognition and speech recognition. It is considered an efficient algorithm, and modern implementations take advantage of specialized GPUs to further improve performance.

Implementation Within Various Networks

In total 4 separate neural networks were tested with 4 completely different data sets and methods. The first network started off simple with cats and dogs. The first dataset trained through thousands of pictures of cats and dogs running over a couple thousand epochs. When the training data had all been read, an image of a cat or a dog was input into the network which in turn would output whether the AI thought the image was of a cat or a dog. The result of which relied solely on how well the AI was trained prior to the input.

The second network revolved around Satellite Imagery. To get started similar to the Cats and Dogs the network had to be taught a dataset. However this wasn’t provided and had to be assembled from scratch. In order to effectively collect a dataset satellite images were streamed from a website, accessed within a written script called ImageGrabber. A grid would then be marked on the image, which a separate script would chop up evenly into manageable chunks. These chunks were then sorted into folders each relative to the area it best depicted, for example a forest, road, water, etc. This helped to teach the system the different types of areas it would be looking at.

A slight issue was run into when obtaining the satellite images. It appeared when grabbed the images would be greyscale and have UI markers covering them, as seen in Figure 4. After a couple hours of debugging, going through the code line by line it was discovered the whole issue stemmed solely off of one mistake within the code. "Centre" had to be spelt the American way of "Center". After this was fixed the images output colourful and clear, as seen in Figure 5.

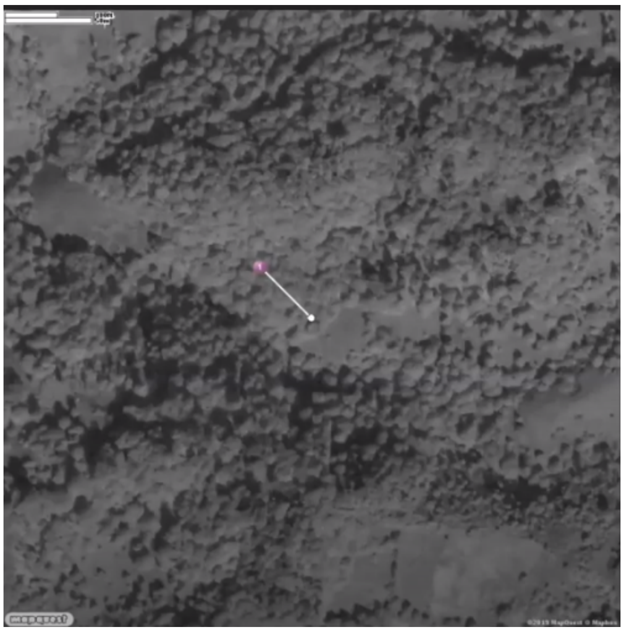
Figure 4: *Incorrect Satellite Grab*

The third neural network tested took a different approach by using faces. The 2 previous networks output a result of what the AI thought the image was of. The main difference between this network and the 2 prior is this one aims to recreate the faces to a degree which it they could pass off as the original images and not generated by AI. After a single run of 10,000 epochs there were clear results. Figure 6 shows the first output after 100 passes were made by the AI, with little more than a grey smear to show for it. However, after 10,000 epochs the results somewhat resembled faces, albeit very pixelated and distorted, seen in Figure 7.

The final neural network attempted to replicate a pattern. The pattern chosen was of tiger stripes. This process required gathering another dataset from scratch which would have been ideally as simple as the satellite network. After downloading a couple of

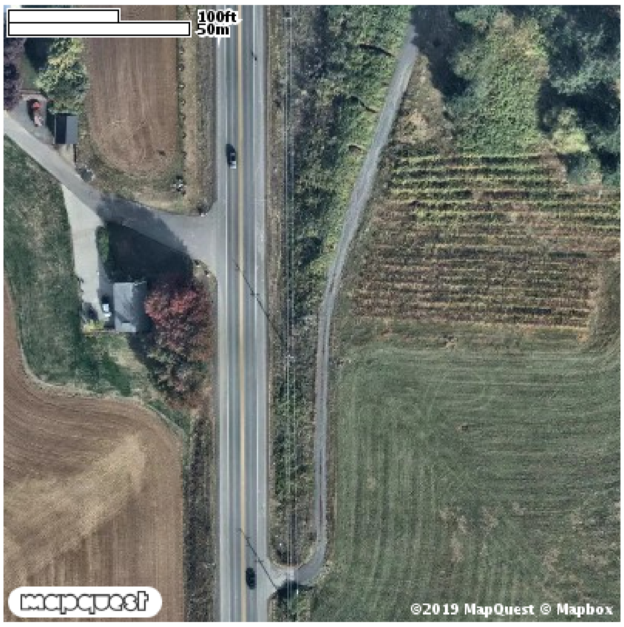

Figure 5: *Correct Satellite Grab*

hundred images off of the internet, issues soon started to rise to the surface in the form of the system not accepting the new dataset due to some format conflicts.

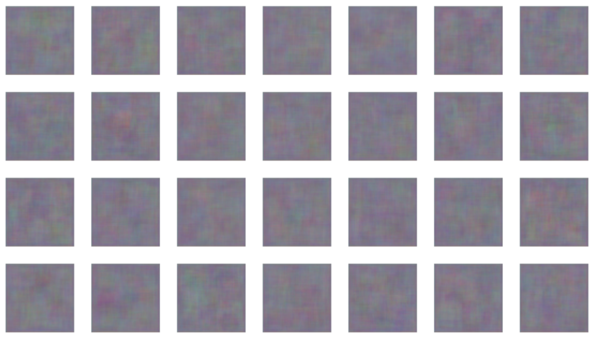
Figure 6: *Neural Network's First Output*

The images output by each network replicating textures can be easily exported into Unity or other engines, and used for materials or groundwork for future developments, as seen in Figure 8.

4 Evaluation and Conclusion

Deep Learning algorithms are extremely intricate and hold a huge amount of power when programmed correctly. The ability for the AI to learn and perfect tasks completely by its own methods is truly amazing, ranging from mastering chess or discovering cures for diseases, the possibilities are endless. There are still many ways in which these systems can be developed and improved upon in future, with refined coding and more powerful equipment the process of

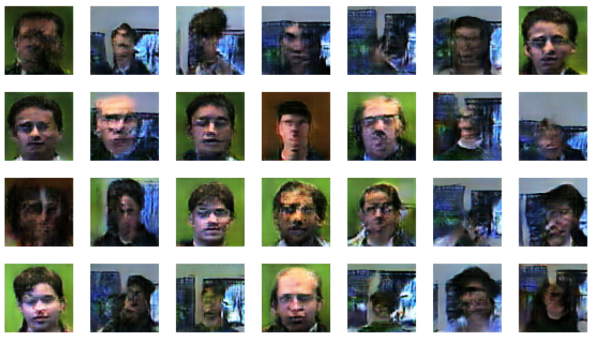

Figure 7: *Neural Network's 100th Output*

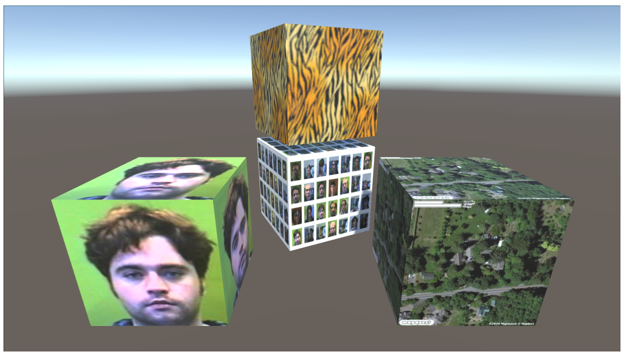

Figure 8: *Textures Exported To Unity*

creating realistic copies could be sped up exponentially.

Bibliography

Lima, Markus V. S. (2018). *Vessel Classification through Convolutional Neural Networks using Passive Sonar Spectrogram Images*. Federal University of Rio de Janeiro.

Raicea, Radu (2017). "Want To Know How Deep Learning Works? Here's A Quick Guide For Everyone." In:

Sarah Knapton, Leon Watson (2017). "Entire Human Chess Knowledge Learned And Surpassed By Deepmind's Alphazero In Four Hours". In:

Silver D., Hubert T. Schrittwieser J. Antonoglou I. Lai M. Guez A. Lanctot M. Sifre L. Kumaran D. Graepel T. Lillicrap T. Simonyan K. and D. Hassabis (2018). *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*. DeepMind.