



School of Computing and Mathematical Sciences

BSc Games Design and Development

Academic Year 2019-2020

An investigation into the use of Neural Networks for procedurally generating images
based on real-world map data

Kaisei Fukaya ID:000979004-7

TOTAL NUMBER OF WORDS: 9516

A Dissertation submitted in partial fulfilment of the requirement for the degree of
Bachelor of Science

ABSTRACT

Procedural content generation (PCG) can be used in games to dynamically create content and reduce the workload of artists. Existing methods limit artistic control of the resulting content. To solve this issue PCG can instead be applied to the aesthetic of the content rather than the content itself, allowing for artists or designers to have full control over the properties of the content. This document will propose how this can be achieved using Machine Learning (ML).

This document proposes the use of real-world data as a source of characteristics that can be applied to the generation of map images. This can be achieved by applying the concept of neural style transfer to produce satellite style images from basic road map images. In having a tool that can generate satellite quality maps from a basic input, artists can have full control over the content of the map, with the styling process handled by the system. This concept could be extended to apply different styles other than that of satellite images such as the design style of games using this method.

ACKNOWLEDGEMENTS

I would like to thank my initial first supervisor Dr Damon Daylamani-Zad for helping to formulate the direction of this project, providing support in the early stages of research.

Additionally, at a time when in-person product testing cannot be conducted, I would like to thank my supervisor Mihai Polceanu for providing invaluable advice on how testing could be still be achieved.

TABLE OF CONTENTS

CHAPTER 1: Project Contextualisation	1
1.1 Introduction	1
1.2 Research aim and objectives	1
1.3 Research approach.....	2
1.4 Legal, Social, Ethical and Professional Issues and Considerations.....	1
1.5 Literature Review.....	1
1.6 Dissertation outline	2
CHAPTER 2: Data Sourcing.....	3
2.1 Access to satellite and road data through open source GIS software	3
2.2 Summary	5
CHAPTER 3: Development of convolutional autoencoder	6
3.1 Data Pre-processing	6
3.2 The Encoder	7
3.2.1 Conv2D	7
3.2.2 Maxpooling2D.....	9
3.2.3 Batch Normalization.....	9
3.2.4 Encoder structure	10
3.3 The Decoder.....	10
3.3.1 UpSampling2D	11
3.3.2 Output Node	11
3.3.3 Decoder structure	12
3.4 Training.....	13
3.5 Summary	14
CHAPTER 4: Serving on Unity	15
4.1 User Interface	16
4.2 Workflow for producing modular map pieces.....	18
4.3 Interfacing with Tensorflow in Unity.....	18
4.4 Converting between Unity Textures and Bitmap	19
4.5 Summary	19
CHAPTER 5: Testing and evaluation of the product.....	20
5.1 Evaluating Model Performance	20
5.1.1 Performance over the course of training	21
5.2 User Study.....	25
5.2.1 User study evaluation	27
5.3 User Privacy Considerations.....	31
5.4 Testing Conclusions.....	31
CHAPTER 6: Concluding Discussion	33

6.1 Dissertation Summary.....	33
6.2 Research contributions	33
6.3 Future research and development	33
APPENDIX A: PRIMARY DATA COLLECTION METHOD	37
Appendix A.1	37
Appendix A.2	40
Appendix A.3	41
APPENDIX B: SOURCE CODE	43
Appendix B.1:	43
Appendix B.2:	44
Appendix B.3:	45
Appendix B.4	46
Appendix B.5	47

CHAPTER 1: PROJECT CONTEXTUALISATION

1.1 Introduction

Procedural content generation (PCG) can be defined as the algorithmic generation of content. In the context of video games it is applied as a method of creating game content automatically, reducing a game artist's workload (Van Der Linden, Lopes and Bidarra, 2014). One area where PCG can be applied is the creation of maps. Maps in the real-world are used for navigation and are stored digitally in the form of tiles which are square images of a set resolution (Qiu and Chen, 2018). A large amount of real-world map data is freely available to the public. This real-world data could be used to explore patterns in how networks of roads and buildings throughout the world are structured.

Machine learning (ML) is an area of computing and data science that involves identifying and applying processes that are able to learn from data (Vasilev et al., 2019, p.6). One ML approach is the use of neural networks, these are algorithms modelled after brain functionality. This area is applied to image recognition among other high complexity tasks such as speech recognition and language processing (Nielsen, 2015).

Gatys, Ecker and Bethge discuss the concept of using a neural network to transfer image styles, applying a class of neural network called convolutional neural networks in obtaining content or characteristics of one image and applying the style or textural quality of another to create a new interpretation of the original image (2015).

The intent of this dissertation will be to explore the application of ML in producing content based on real-world data. Available map data repositories such as those owned by Google Maps Platform (Google, 2019) can be accessed to obtain satellite imagery as well as simplified maps that show the layout of roads. The project will involve the development of a convolutional neural network for use in a system that takes basic map imagery as an input and outputs a satellite style image. This can be achieved by training a neural network to associate the characteristics of a basic map, with the textural qualities of a satellite image. Applications of the result could include use as a tool for game designers in creating image assets from basic level plans, or in-game implementations that take the layout of a procedural level and create a dynamic mini-map on the user interface.

1.2 Research aim and objectives

The aim of this dissertation will be to investigate how machine learning can be applied to the procedural generation of maps using real-world data.

Objective 1. To research relevant academic literature pertaining to algorithms and techniques related to image processing using neural network implementations. Furthermore, investigating real-world data sources.

Objective 2. Design and development of a neural network structure, with decisions being made on algorithms and methodologies employed. Including how input data will be structured to best facilitate the neural network in producing the desired outputs. This process is to be documented and ideas will be iterated upon with functional prototypes.

Objective 3. Developing a suitable dataset for use in training and testing the

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

neural network. This objective will use the knowledge gained in objective 1 as well as decisions made in objective 2, to form a dataset of appropriate size and quality. The dataset will consist of basic map representations as well as satellite images of London, this will provide a set of consistent characteristics for training the neural network.

Objective 4. Create proof of concept prototype- implementing a final trained neural network, using the results of objectives 2 and 3. Additionally, a user interface will be created to act as medium that allows users to produce input data for the neural network, and to display the results.

Objective 5. Devising a set of metrics for testing the result of objective 4. Following this, the effectiveness of the product will be evaluated as well as decisions made throughout the project. Issues with the product will be discussed as well as how the concept may be built upon.

Table 1. Summary of objectives, methods, deliverables and expected duration

Objective	Method	Deliverable	Duration
Objective 1	<i>Reading and application of existing techniques displayed in academic papers.</i>	<i>Research documentation.</i>	4 weeks
Objective 2	<i>Designing and iterative prototyping of ideas using TensorFlow (Google Brain, 2019a).</i>	<i>Design documentation on chosen neural network structure and user interface.</i>	3 weeks
Objective 3:	<i>Storing data from map data sources.</i>	<i>A dataset of appropriate breadth and quality.</i>	2 weeks
Objective 4:	<i>Developing a user interface and neural network using existing platforms i.e. TensorFlow (Google Brain, 2019a) and web format or game engine.</i>	<i>A proof of concept prototype.</i>	6 weeks
Objective 5:	<i>A questionnaire will be used to receive feedback from users.</i>	<i>Documentation of testing and project evaluation.</i>	5 weeks

1.3 Research approach

The first objective in this project will be to investigate methods of machine learning as well as real-world data sources. Research papers in the area of machine learning will be accessed from databases such as IEEE (Institute of Electrical and Electronics Engineers) and ACM (Association for Computing Machinery). Similarly, books and articles written based within the field of machine learning will be accessed. The purpose of this investigation will be to identify key methodologies used in the creation of neural networks. In particular, techniques used in the creation of neural networks that centre around image interpretation and processing. Furthermore, a decision will be made on the data source most appropriate for use in training a neural network of this type. Sources will be compared based on the properties of their data and how such properties may affect the quality of the neural network's output. This objective will be achieved in the duration of 4 weeks.

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

Based on the outcome of research a neural network will be designed. The design process will involve the prototyping of ideas which will be iterated on. The machine learning framework TensorFlow (Google Brain, 2019a) will be used for this process, in order to have high level control over the structure of the neural network. This objective will take 3 weeks to achieve.

Objective 3 is to acquire a set of data to be used in training the neural network. The source of such data will be chosen based on the outcome of the first objective. A method of obtaining and storing this data will be formulated. The data may also be altered, for example- reduced in pixel resolution- to maximise the speed at which the neural network can process it. This objective will take 2 weeks to achieve.

Objective 4 is to implement the neural network designed in objective 2, training it on the data-set from objective 3. Additionally, a user interface will be implemented to allow users to create their own basic map designs to be used as an input to the neural network. This may take the form of a canvas where a user can draw in road shapes, or a system where modular pieces can be connected to create a map layout. The goal is for the user input to stylistically match the training data, with the intention being for the neural network to produce an image accurate to the characteristics of the user design. This user interface will be implemented on a platform with direct access to the TensorFlow (Google Brain, 2019a) library, such as an internet browser, or game engine. This objective will be achieved in the duration of 6 weeks.

Objective 5 will involve the testing of the finished project, with a set of quantitative metrics being devised and tested for in the form a questionnaire that will be completed by users. The results of this testing will be examined and discussed with the goal of ascertaining areas of success and failure. Additionally, the project will be evaluated as a whole, with further areas of development discussed. This objective will be achieved over the duration of 5 weeks.

1.4 Legal, Social, Ethical and Professional Issues and Considerations

An integral part of the project is sourcing real-world data. This will involve accessing existing databases. One such source- Google, provides an application programming interface (API) for accessing map images called Maps Static API. Use of this API will need to comply with terms of service set out by Google Maps Platform Terms of Service (2019). One notable condition is 3.2.4 (a) which states that the customer cannot save Google Maps content for use outside of the services. This means that storing a large number of images from Maps Static API is not an option. If necessary, the system would need to directly take in images from the API without storing them first. Another condition 3.2.4 (c) states that the customer cannot create content based on Google Maps content. This limits the usability of such content, potentially disallowing its usage in the case of this project for training a neural network. Therefore, another source will need to be used for this project. Further ethical considerations must be made in the testing of the product. Testing participants will need to have consented to the testing process. Questionnaire questions will be phrased objectively. The questionnaire will be filled in anonymously with no inclusion of personal data. Additionally, the individual answers will be kept confidential.

1.5 Literature Review

Van Der Linden et al. (2014) observed a multitude of possible procedural generation methods used in games that can be applied to the task of creating dungeon levels. Discussing the benefits and drawbacks of each method. It is stated that PCG can range from confined tools that can be used to aid development to full systems that form a large part of a game's content. Van Der Linden et al. make the assertion that PCG techniques are used less due to the lack of control that is available to designers, going on to conclude that some existing methods can fulfil designers' requirements. The benefit being that PCG systems can save time. One technique covered was the use of real-world architectural data being applied to Bayesian networks in research by Merrell et al. (2010). Although designers had control of general parameters such as the number of rooms or square footage, the results always appeared too similar as there was a lack of fine control.

Qiu and Chen (2018) discuss how real-world map data can be accessed and visualised. Explaining how Mercator projection is used to map the surface of Earth onto a 2-dimensional map, with it being commonplace for web services to separate data into square basemaps that are accessible based on a given coordinate and zoom level.

Gatys, Ecker and Bethge (2015) establish a method of employing machine learning in generating content. A system is created that attempts to compose images based on image style and content. This is achieved with the use of convolutional neural networks that abstract content and style information from given images- such that this information can be re-combined to create a new image. In doing so- Gatys, Ecker and Bethge present the novel concept of neural style transfer. Furthermore, general information on machine learning methodologies and high level techniques are provided by Vasilev et al. (2019), with autoencoders and variational autoencoders (VAEs) being discussed as a method of generating images through

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

capturing basic representations and altering how these are re-constructed. Where a VAE can produce unique variations of a given input, regular autoencoders have a simpler structure but can still be trained to associate certain input information differently by manipulating the data used as a target for optimisation.

This dissertation will investigate how the aforementioned concept of neural style transfer- in the use of neural networks- can be applied to the problems found in the intersection of design and PCG described by Van Der Linden et al.; applying the use of real-world map data as a basis for a system that can be used as a generative tool for designers.

1.6 Dissertation outline

The following chapters in this dissertation will detail the techniques and methodologies used throughout the production of the product. This will be a proof of concept prototype, based on the initial research set out in the literature review, consisting of a trained autoencoder model and user interface created in the Unity game engine (Unity Technologies, 2020b).

In chapter 2 a method of sourcing an appropriate dataset will be established, with considerations for how the autoencoder will be trained. Chapter 3 will detail the structure and workflows employed in creating the autoencoder model and training it. In chapter 4 the design of the final Unity program will be covered, with consideration for how the trained autoencoder model is implemented and accessed by the user. Chapter 5 will cover the testing methodologies used to evaluate the finished product and chapter 6 will consist of concluding discussion.

CHAPTER 2: DATA SOURCING

In order to train a machine learning graph to produce a desired output, an appropriate data set is required. During the training process, the weights of the model are fit to this data. The more varied and expansive the dataset used the more it is expected that the fully trained model is able to produce the intended result (Vasilev et al., 2019, pp.48–55). The aim of this project is to produce a convolutional autoencoder that is able to learn the features of a set of roadmaps corresponding to the features of satellite imagery of the same location. This will require a dataset consisting of two parts, a set of roadmap images, and an identical set of satellite images. This chapter will explore how the dataset has been sourced for use in this project, covering how this data is accessed, and processed for use in training the autoencoder model.

2.1 Access to satellite and road data through open source GIS software

A geographic information system (GIS), is a type of software that is used for working with geographic data, they can be used to visualize and edit data for various uses ranging from designing maps to represent real locations to creating graphical representations of data combined with maps. QGIS (QGIS Development Team, 2020) is an open source GIS application that allows for loading existing map data and layering different representations and graphics.

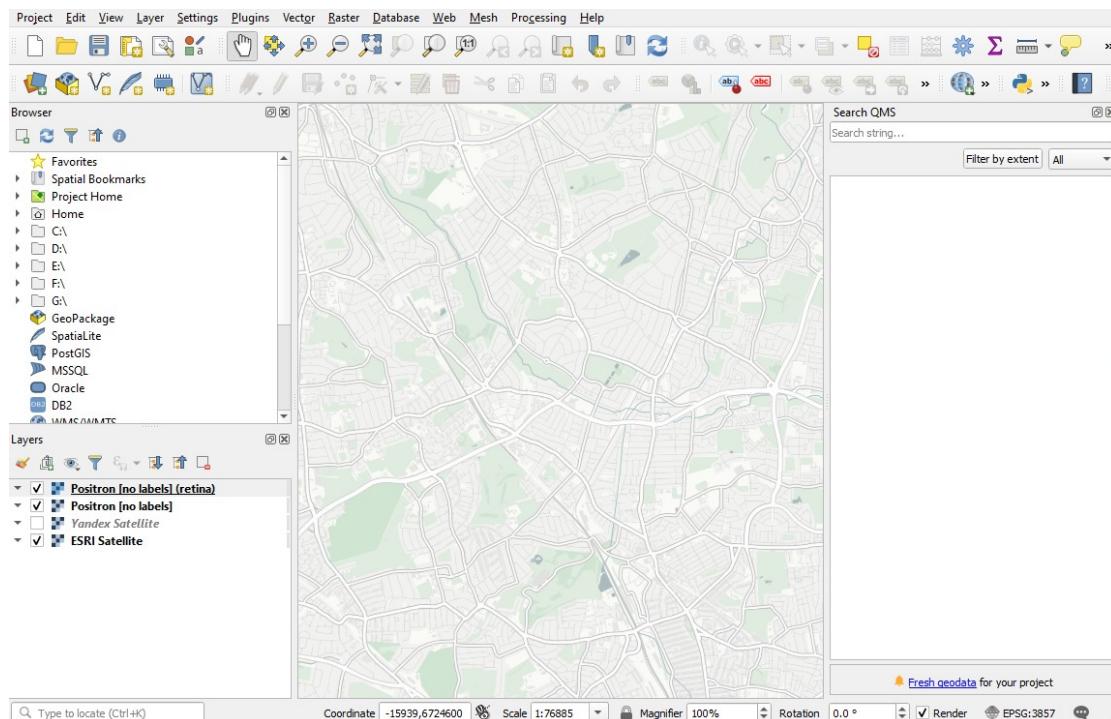


Figure 1: QGIS software's user interface (QGIS Development Team, 2020).

In this project QGIS is used for accessing map data for training. A plugin called QuickMapServices (NextGIS, 2019) is used to access basemaps from a selection of web based services, these are loaded as layers in QGIS that naturally align based on the coordinate and scale set in the software's viewport. The satellite image basemap selected is the ArcGIS satellite data (ESRI, 2020). The roadmap selected is the

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

CartoDB Positron basemap without labels, this was chosen for its simple clear visuals that would both act as consistent data for training the neural network as well as being suitable for creating modular pieces for the end user to create layouts with. This basemap is built using up to date information from OpenStreetMap which should match the details in the satellite basemap (CartoDB, 2014).

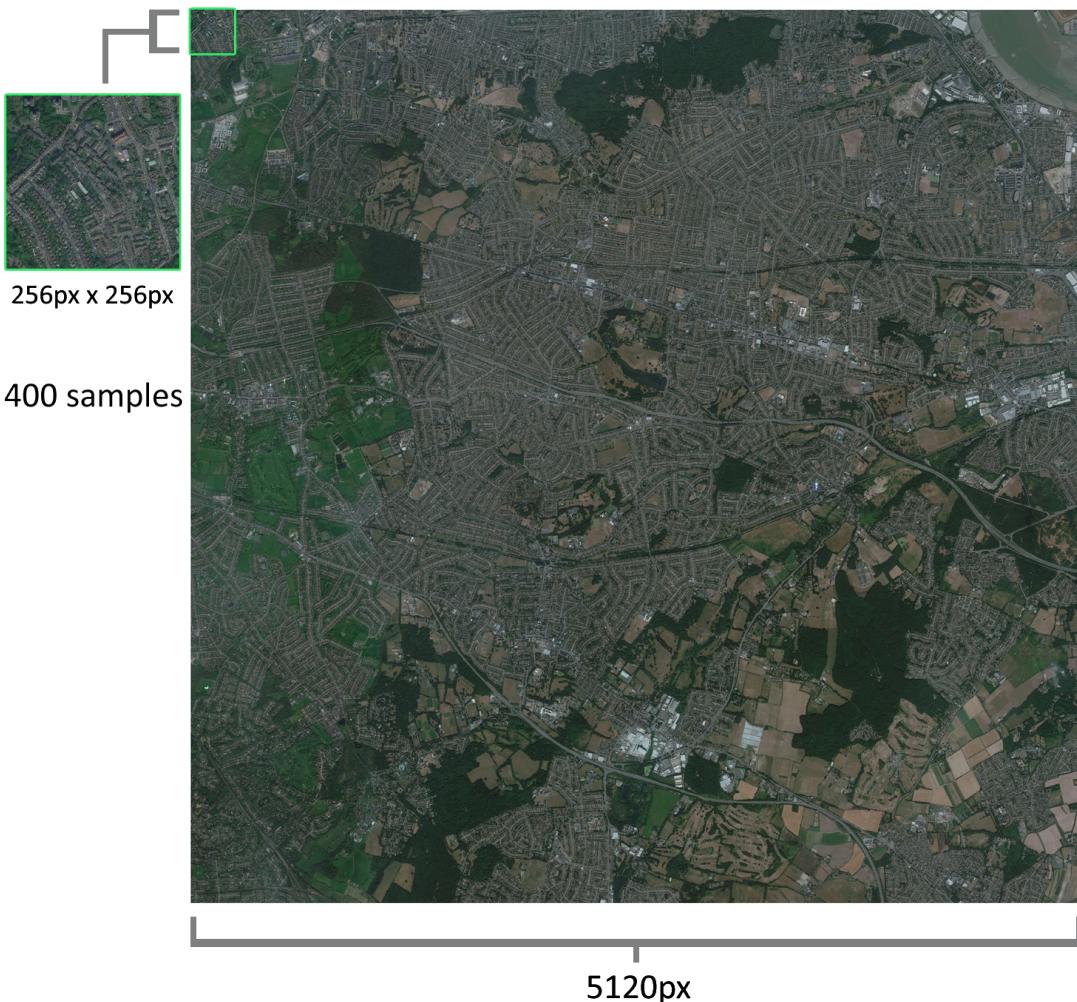


Figure 2: Diagram showing how a single 5120px map images is processed down to 400-256px samples.

Figure 2 shows part of the workflow for processing this basemap data into an appropriate dataset. A 5120 by 5120 pixel window is used to frame a location in or around London, which is then exported from QGIS in both roadmap and satellite form. This image is then processed into 400 equal samples 256 by 256 pixels in size. These samples are named and numbered appropriately such that satellite and roadmap samples match in location at any given index. This process is repeated 10 times resulting in a dataset of 4000 sample pairs.

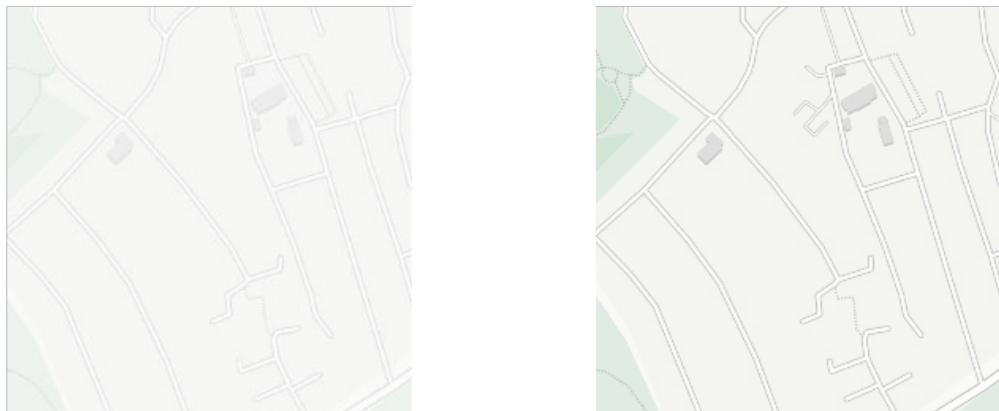


Figure 3: Comparison between the Positron basemap and the edited version used in the final training dataset.

One issue with the roadmap initially was that it lacked contrast and definition, which could be detrimental to training as well as making it difficult for users to see their layout as they create it. Therefore QGIS' layer system was utilised in layering the two copies of the basemap with a blend mode of multiply while increasing the saturation.

2.2 Summary

In this chapter a method of sourcing and processing data for use in training the autoencoder is explained. For legal reasons it is established (in chapter 1) that the Google static map apis are not appropriate for use in this project, instead an alternative method is suggested that involves the use of QGIS in accessing and rendering basemaps that can be processed down to an appropriate dataset of 4000 sample pairs. In the next chapter- chapter 3- the development of the autoencoder will be discussed, detailing how the dataset from this chapter is pre-processed and used to train a model.

CHAPTER 3: DEVELOPMENT OF CONVOLUTIONAL AUTOENCODER

This chapter will cover the development and design decisions behind the creation of the autoencoder. Section 3.1 will cover the pre-processing of training data. Section 3.2 and 3.3 will cover the structure of both the encoder and decoder respectively with the types of Keras layer used in each. Keras is a high-level library used for creating machine learning models, this is built on Tensorflow and utilises the Python scripting language (Chollet, 2019). This chapter will be summarised in section 3.4.

3.1 Data Pre-processing

To ensure that training data is presented to the neural network in a consistent way it must be pre-processed. This can involve re-sizing or reshaping an image. In the case of this project the data collection method allows for a dataset where each image has identical dimensionality. In this case the only pre-processing required is the normalization of the image data. This means scaling the data down from an integer value between 0-255 per colour dimension, to a float value between 0-1. By normalizing the image data, training speed and consistency can be improved (Vasilev et al., 2019, p.114).

In order to feed data into the neural network during the fitting process, the image data must be readily available. One way this can be delivered is by storing all this data in RAM for the duration of fitting. This can work for smaller datasets, however if the dataset exceeds the available system RAM this method is not possible. In this project generators are used instead. These are yieldable methods that can be used to sequentially present data in batches to the neural network. Rather than storing the entire dataset in RAM at one time, the generator accesses each batch of images individually from the directory, pre-processing the images and passing them to the neural network after each training step.

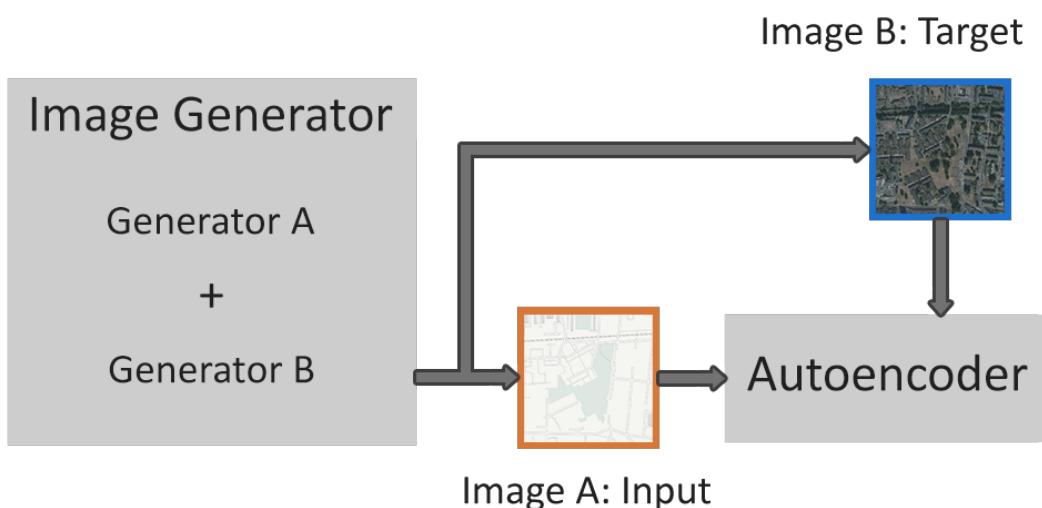


Figure 4: A diagram showing the flow of how a generator produces an input image and target image upon request.

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

To train the autoencoder, a generator that outputs batches of both roadmap inputs and satellite image targets is required. With the main specification being that both images match at any given inference. This is achieved by combining the results of two generators.

The training method in Keras requires a two-dimensional array that takes a batch of input images and a batch of target images respectively. In the case of training an autoencoder to reconstruct an identical image to its input, both of these dimensions will consist of identical data (Vasilev et al., 2019, pp.168–170). This is so that the error can be calculated based on the likeness of the output data to the original image. However, in this use case, where the aim is to train the autoencoder to construct the input image in a different style, the target values must consist of the target style, in this case the satellite variant of the given roadmap. Two separate generators are used to achieve this, with one yielding the input and another yielding the target. These are then combined to create a single two-dimensional array that can be passed to the training method, see Appendix B.3.

3.2 The Encoder

The purpose of the encoder within an autoencoder is to encode the essential details of an input such that it can be reconstructed, ideally identical to the input. In this case however, the autoencoder is being used to reconstruct the image in a different style to that of the input. In either case the encoder serves the same role- extract key features of the data and remove anything else.

3.2.1 Conv2D

Conv2D is a Keras layer that implements convolutions over a two-dimensional image space. In this case it is primarily used for extracting information from an input. A convolutional layer works by taking the input data and passing a kernel over it, convolving at each point where an input point aligns with a point in the kernel. This results in a feature map as an output. Figure 5 shows an example of this process with the blue grid representing an input, the darker cells representing the kernel, mapping onto an output in green.

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

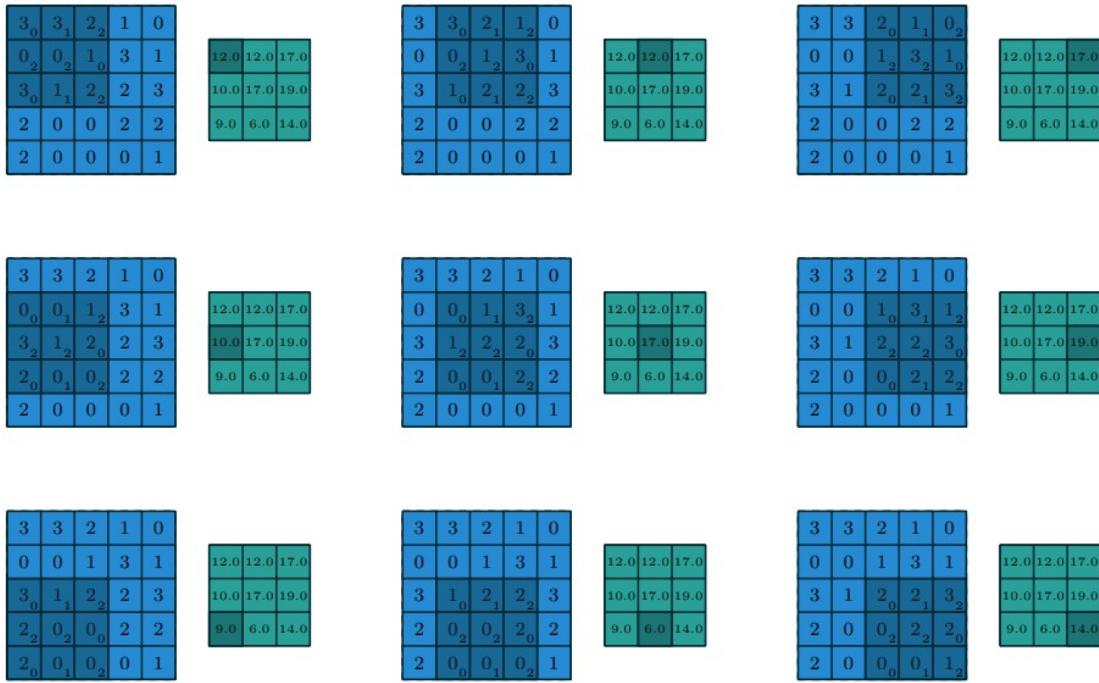


Figure 5: Example of convolution operation, a 3 x 3 kernel moves over a 5 x 5 input with a stride of 1 x 1. Provided in *A guide to convolution arithmetic for deep learning* (Dumoulin and Visin, 2018).

In images, the structure of the data is important. The relationship between a pixel and its neighbours is what makes an image visually identifiable, additionally in the case of images with colour- each channel relates to the other in the same way. The use of convolutional layers in this case has the benefit of making use of this relational information (Dumoulin and Visin, 2018).

In the Keras implementation a number of parameters can be adjusted, one such parameter is the number of filters, which is the number of kernels that are convolved over the input- increasing this will result in more output feature maps- each of which will abstract the input data slightly differently. Another is the stride, this is a value by which a kernel is moved each time, for example a stride of 1 will shift the kernel by one value or pixel in the grid- increasing the stride will resultantly reduce the output dimensions as values will be skipped over.

The kernel size value controls how large each kernel is, this can be increased to obtain feature maps based on larger areas of an image, allowing it to learn features or relationships over a larger space.

Padding adds values of zero around the input data, this is done to control the output dimensions of a given feature map. In Keras this has two settings, with the intricacies being handled by the library itself. These are “valid” and “same”, “valid” results in no padding, therefore potentially resulting in an output with different dimensions to the input- depending on how the kernel maps onto the input as it moves. Padding set to “same” results in Keras adjusting padding such that the output always has the same dimensions as the input (Rosebrock, 2018).

3.2.2 Maxpooling2D

Maxpooling2D is a Keras layer that implements max pooling over two-dimensional data. Max pooling is a method of down sampling data, reducing each dimension based on the size of the filter kernel. For example, an input with two dimensions of 256 and a filter size of 2 by 2 will result in an output with two dimensions of 128, or one quarter of the input data. It does this by taking the highest or “Max” value from each filter and discarding the rest (Dumoulin and Visin, 2018).

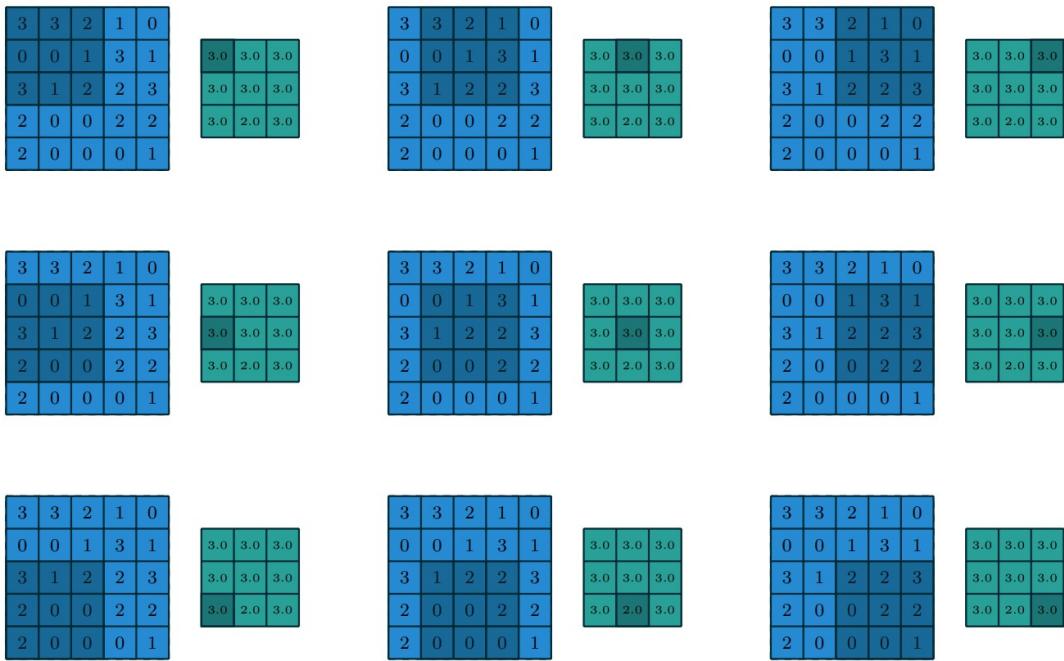


Figure 6: Example of max pooling operation with a 5×5 input and 3×3 output with a stride of 1×1 . Provided in *A guide to convolution arithmetic for deep learning* (Dumoulin and Visin, 2018).

Within the context of the convolutional autoencoder this is used as the primary method of bottlenecking the input image- discarding low value data and keeping the most prominent features of the data. Maxpooling2D layers are used a total of three times in the final architecture, resulting in a reduction from an input of shape of “(256,256,3)” to an encoded shape of “(32,32,256)”, where the third dimension is the number of filters resulting from Conv2D layers.

3.2.3 Batch Normalization

When training a neural network, the parameters of each layer are adjusted with each iteration. Each layer’s input consists of the output of the layer prior, therefore parameter changes in layers can have a knock-on effect for proceeding layers which amplifies the deeper the network is. This issue is identified as internal covariate shift and is often countered by using a lower learning rate. Batch normalization however, mitigates internal covariate shift while allowing for a higher learning rate, therefore increasing training speed (Ioffe and Szegedy, 2015). It achieves this by normalizing input values over the course of a mini-batch. The Keras implementation of batch normalization allows for batch normalization layers to be placed after other layers to produce this effect.

3.2.4 Encoder structure

Layer (type)	Output Shape	Param #
input_node (InputLayer)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 256, 256, 64)	9472
batch_normalization (BatchNo	(None, 256, 256, 64)	256
max_pooling2d (MaxPooling2D)	(None, 128, 128, 64)	0
conv2d_1 (Conv2D)	(None, 128, 128, 64)	36928
batch_normalization_1 (Batch	(None, 128, 128, 64)	256
conv2d_2 (Conv2D)	(None, 128, 128, 128)	73856
batch_normalization_2 (Batch	(None, 128, 128, 128)	512
max_pooling2d_1 (MaxPooling2	(None, 64, 64, 128)	0
conv2d_3 (Conv2D)	(None, 64, 64, 256)	295168
batch_normalization_3 (Batch	(None, 64, 64, 256)	1024
conv2d_4 (Conv2D)	(None, 64, 64, 256)	590080
batch_normalization_4 (Batch	(None, 64, 64, 256)	1024
max_pooling2d_2 (MaxPooling2	(None, 32, 32, 256)	0

Figure 7: Encoder structure summary in Keras.

The encoder consists of three max pooling passes, reducing the input dimensions from 256 to 32 with 256 filters, between which are convolutional layers followed by batch normalization. Padding is set to “same” on each convolutional layer to ensure that the layer outputs a tensor of the same dimensions as the input. This ensures that throughout the encoder the shape remains proportional, such that the correct dimensions can be achieved once up sampled in the decoder. Each convolutional layer has a kernel size of 3 by 3 with the exception of the first layer which has a kernel size of 7 by 7, this is done to extract larger features early on in the process. The number of filters is increased throughout the encoder to abstract the data more the further into the graph it gets, see appendix B.4.

3.3 The Decoder

The purpose of the decoder within an autoencoder is to interpret data from the bottleneck and produce a reconstructed image of the desired shape and quality. It does this by building up the data’s dimensions using a method of up sampling, while also interpolating information to form an image with more definition than could be found within the bottleneck.

3.3.1 UpSampling2D

UpSampling2D is a Keras layer that is used for up sampling two-dimensional data, by default it doubles each dimension using nearest neighbour interpolation. Within the decoder portion of the convolutional autoencoder these layers are used to reverse the effect of max pooling layers, increasing the dimensions of the data to match the desired output shape.

Up scaling can also be achieved using Conv2DTranspose layers which, with strides of 2 by 2 can double both input dimensions. Transpose convolutions make use of an inverted form of regular convolutional computation. This can be used in much the same way as nearest neighbour up sampling in the sense of increasing the dimensions of data, and can result in a higher quality output, however it introduces more parameters to a graph (Dumoulin and Visin, 2018).

In the autoencoder model up sampling layers are used in combination with standard convolutional layers to both increase dimensions and learn a method of introducing characteristics of the target output.

3.3.2 Output Node

The output node is a Conv2D node with only three filters, this will result in a desired output of “(256,256,3)”, with each filter representing the red, green and blue values respectively. A sigmoid activation function is used to ensure an output of values between 0 and 1, this will match the format of the normalized target data on which the output will be compared.

3.3.3 Decoder structure

up_sampling2d (UpSampling2D) (None, 64, 64, 256)	0
conv2d_5 (Conv2D) (None, 64, 64, 256)	590080
batch_normalization_5 (Batch (None, 64, 64, 256))	1024
conv2d_6 (Conv2D) (None, 64, 64, 256)	590080
batch_normalization_6 (Batch (None, 64, 64, 256))	1024
up_sampling2d_1 (UpSampling2 (None, 128, 128, 256))	0
conv2d_7 (Conv2D) (None, 128, 128, 128)	295040
batch_normalization_7 (Batch (None, 128, 128, 128))	512
conv2d_8 (Conv2D) (None, 128, 128, 64)	73792
batch_normalization_8 (Batch (None, 128, 128, 64))	256
up_sampling2d_2 (UpSampling2 (None, 256, 256, 64))	0
conv2d_9 (Conv2D) (None, 256, 256, 64)	36928
batch_normalization_9 (Batch (None, 256, 256, 64))	256
output_node (Conv2D) (None, 256, 256, 3)	195

Figure 8: Decoder structure summary in Keras.

The decoder consists of three up sampling passes, increasing the bottlenecked data's dimensions from 32 to 256, matching the input shape. Between up sampling layers are conv2d and batch normalization layers mirroring those found in the encoder portion of the graph.

3.4 Training

An optimisation algorithm is an algorithm that is used to optimise the parameters of the model to achieve the desired output, usually by minimising a loss function. One key parameter of an optimisation algorithm is the learning rate, this is a value used to specify the magnitude at which parameters in the model are adjusted (Vasilev et al., 2019, pp.48–58). This project uses the Adam optimisation algorithm, with an initial learning rate of 0.002, that is decayed to 0.0006 over the course of training.

Prior to training the model, the number of epochs must be specified. An epoch is defined as a single pass over the training dataset, in this case 4000 sample pairs. A value of 500 is used, however as the final model was trained over multiple sessions, the total number of epochs is far greater at 2,789- which is largely a result of experimentation, adjusting hyper-parameters and batch sizes between training runs. If training were to be repeated- fewer epochs should be required to achieve the same result.

Each epoch is divided into steps at which optimisation occurs, in this case the number of samples divided by the batch size is used to define the number of steps per epoch. This is done to ensure that there is an optimisation step for every batch of images.

Keras has a feature built into its training methods that allows for call backs between training epochs, in this project two have been used (Keras, 2020). The first is a call back that outputs checkpoints at set intervals, this was configured for every five epochs in this case. A checkpoint is a file that stores the weights of the model at the time it is created, this acts as a backup, also allowing training progress to be resumed between runtime sessions. The second is a method that monitors a specified value, if the value does not change over the course of a set number of epochs it reduces the optimizer's learning rate, in this case loss is monitored. Reducing the learning rate when the model is not improving helps to gradually approach the ideal weights.

Training was performed using Google Colaboratory (Google, 2019), a web based service that provides access to high performance GPUs (Graphical Processing Units), which are able to perform the repetitive calculations involved in training at a high rate. One downside to using this service is that access to such resources is limited and inconsistent. When training for a long period of time it is possible that the runtime will be stopped and access to GPU usage will be temporarily revoked, this is done to prevent users from using resources for too long such that other users cannot access them (Google, 2020a). Additionally, the hardware capability can vary depending on the resources available, this resulted in having to adjust training batch sizes depending on resources at a given time, as larger batches require more memory to store.

3.5 Summary

In this chapter the autoencoder structure has been examined, detailing each of the four Keras layer types used: Conv2D, Maxpooling2D, BatchNormalization and UpSampling2D. It is established that a generator is used instead of loading the entire dataset into RAM at one time, this consists of two directory sources from which data is normalised, combined and batched as a single data object that is retrieved during training. Within the encoder portion of the model- convolutional layers are used to abstract feature maps from the input, with max pooling layers being used to reduce the dimensions of the data down to 32 by 32 pixels. Within the decoder portion of the model- operations are mirrored such that the output shape matches the initial input shape. This is done by using up sampling layers in place of max pooling and convolutional layers with the number of filters in a given layer descending as opposed to the layers in the encoder portion. Batch normalization layers are placed after convolutional layers which allows for faster more consistent training.

The limitations of working with Colaboratory (Google, 2019) are discussed with batch sizes being adjusted based on the resources provided at a given time.

The next chapter- chapter 4- will cover the process of serving the model within Unity, explaining the design of the user interface as well as the method of running the model within a Unity runtime- discussing how a user generated layout is presented to the model, and how the output is extracted and stored.

CHAPTER 4: SERVING ON UNITY

In this chapter the final implementation of the neural network will be covered, with the creation of an appropriate Unity environment for serving and presenting the trained model. The purpose of the Unity environment is to provide wider access to the model, allowing it to be tested as a tool, as well as providing insight into how the model responds to inputs similar but not matching the data it is trained on. This helps to explore how a model such as this can be used as a tool or implemented into a game as a feature.

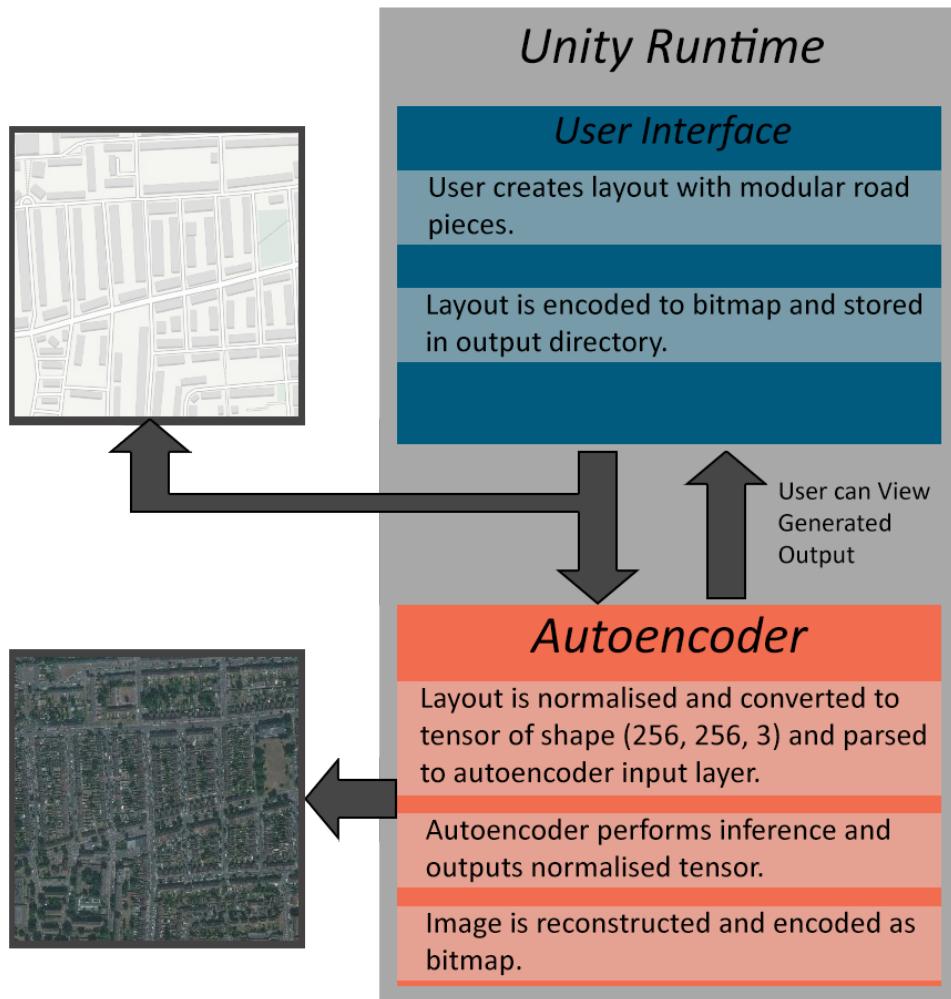


Figure 9: Diagram of the project structure, showing how data from the user interface flows into the autoencoder.

Figure 9 shows how this environment is structured to facilitate interfacing with the model. The user is given the tools to create a map layout, which is then fed into the trained model- the output of which is presented back to the user. Both the input and output are stored for later reference.

The rest of this chapter will consist of section 4.1 which will cover the design of the UI, section 4.2 which will detail the creation of modular map pieces and section 4.3 and 4.4 covering both how the model interfaces with Unity and how image data is restructured throughout the process respectively.

4.1 User Interface

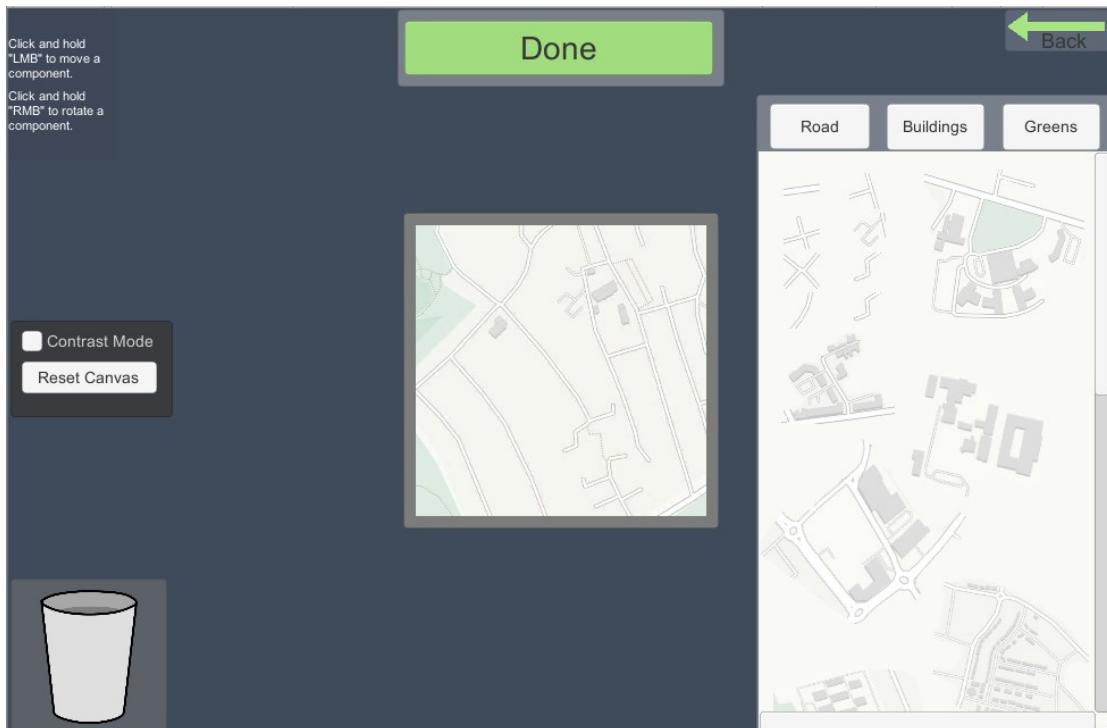


Figure 10: The user interface for creating new layouts.

The user interface (UI) of the product has been designed to allow users to create their own map layouts. It does this by providing a canvas on which the user can drag and drop modular pieces- derived from the initial dataset. Some utilities are included in the UI such as a bin that acts as a region where items are deleted if placed, a contrast mode and an option to reset the canvas. The contrast mode changes the colour of the canvas background from the light colour derived from the training data to black, this is included as an option to help visibility of the items placed on the canvas. When a user has finished creating a layout, the “Done” button can be pressed, at which point the layout is encoded, processed then sent to the autoencoder running on Tensorflowsharp (De-Icaza, 2019). The output from the autoencoder is then loaded as a texture, alongside the input and presented to the user.

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

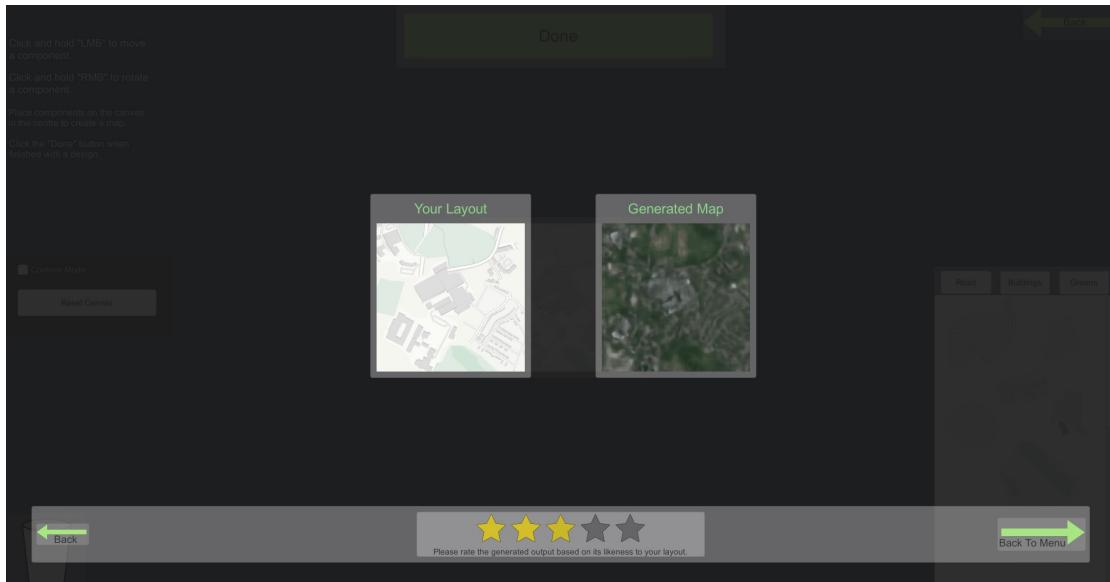


Figure 11: The user interface showing the generated output alongside the input.

Figure 11 shows an example layout presented on the output screen. The user has the option to close this window and reset or edit their design or go back to the main menu. Also, for the purposes of testing, the generated result can be given a star rating as to how well it matches the design.



Figure 12: Library page UI, listing each sample pair.

Additionally, there is a library page accessible via the main menu, this presents all layout and output pairs, along with their session sample IDs. This can be used to view and compare all designs within the samples directory.

4.2 Workflow for producing modular map pieces

The aim was to have user generated layouts match the original training dataset as closely as possible. This would give the neural network the best chance at producing the desired output. The user could be given the ability to freely draw a layout, but the likelihood that the input would have no resemblance to the training data would be high- resulting in poor results. Layouts would instead need to be derived from this initial data. This is done by taking parts from these images and allowing the user to arrange and compose a new layout using them.

Unique parts from the dataset are taken and edited within an image editing tool as PNGs with an alpha channel, allowing them to be overlapped. Once imported into the Unity project, they are made available on a UI panel where they can be dragged onto a canvas.

4.3 Interfacing with Tensorflow in Unity

The machine learning framework used for this project- Tensorflow (Google Brain, 2019a)- primarily supports interaction within the Python scripting language, however the Unity engine primarily supports C# scripting. To import and serve a Tensorflow graph within Unity an implementation called Tensorflowsharp (De-Icaza, 2019) can be used. This along with the ML-Agents plugin (Juliani et al., 2018).

The use of Tensorflowsharp in the current version of Unity ML-Agents has been replaced by a bespoke implementation that directly interacts with the rest of the ML-Agents plugin. Therefore, to run the graph for inference within a Unity environment an older version must be used. The latest accessible version of Tensorflowsharp for Unity works using version “1.7.0” of Tensorflow. Up until this point, the version of Tensorflow being used to create and train the model has been Google Colaboratory’s version “1.x” which equates to the newest version of Tensorflow 1. This means that version “1.7.0” must be installed within the Colaboratory instance so that the exported model is compatible.

To serve the model in Unity the model must be first frozen. Freezing is the process of generating a single file that contains the graph definition and trained weight values converted to constants (Morgan, 2016). This can be saved as a “.bytes” file that can be loaded using Tensorflowsharp. This frozen model is then accessed within Tensorflowsharp, where it can be restored for inference, see appendix B.1. At this point the input data can be passed to the input node, which is accessed by name, and then the output node is fetched also by name.

For this to be successful when using batch normalization layers, the model’s phase must be set to non-learning. This will disable the function of batch normalization which would otherwise cause errors during inference, see appendix B.5. To freeze the model a saved TensorFlow model is passed to the freezing method, this consists of a proto buffer (pb) file and associated weight data stored in an adjacent directory named “variables”.

4.4 Converting between Unity Textures and Bitmap

Textures in Unity can be implemented using the Texture2D class, with the functionality of getting and setting colours at any given pixel. Images can also be imported in the form of a byte array or exported to common compressed image formats such as JPEG and PNG (Unity Technologies, 2020a).

In order to run inference on the layouts that the user generates, they will need to be flattened out and exported in the form of a JPEG file in this case. This has two uses, the first being that the image can then be decoded and fed into the autoencoder at any time and the second being that the image can be stored alongside the output result from the autoencoder for comparison and analysis.

Importing and preparing a given layout for inference involves mirroring the pre-processing steps used in the training process. This includes extracting pixel data normalizing it and then reshaping it into the correct tensor shape.

4.5 Summary

In this chapter a structure for serving the autoencoder within Unity has been established, with simple editing tools that allow the user to create their own map layouts and view the output results. This is achieved with the use of Tensorflowsharp, a library that is able to load frozen Tensorflow models and run inference within a C# environment. The user is given a set of modular pieces derived from the dataset that the model is trained on to ensure that layouts stylistically match what the model is fit to.

The next chapter will cover the methodologies employed in testing and evaluating the product.

CHAPTER 5: TESTING AND EVALUATION OF THE PRODUCT

In this chapter both the trained autoencoder and the Unity based system are evaluated. The autoencoder is evaluated based upon objective training and testing data, detailing its statistical performance and how this compares to the visual result. Additionally, this is evaluated subjectively within the user study where participants have tested the system and rated generated images. Finally, the usability of the Unity program is assessed using the system usability scale.

5.1 Evaluating Model Performance

To evaluate the final model's performance a set of testing data is used. This consists of 200 image pairs (inputs and targets) that have not already been provided to the model throughout training. This is done to test how the model responds to new data.

Table 2: Final model loss and accuracy from testing run. Loss is rounded to 4 decimal places and accuracy is rounded to 2 decimal places.

Model Loss	Model Accuracy
0.0118	0.85

Table 2 shows the resulting loss and accuracy of the model from testing the model on the aforementioned 200 image pairs.



Figure 13: Final trained output image alongside the input and target images.

Observing the output image in comparison to the target (figure 13), certain similarities are visible such as the road structures and colours in general. It is clear from this example that the trained model is able to encode general characteristics of an input and replicate the colours of roads and foliage, however, fails at defining individual buildings and appears blurry. This could suggest that more layers should be added to the architecture to abstract more information from the input and produce a result with more definition.



Figure 14: Output when model is overfit to one sample.

Figure 14 shows an example of a case where the same model is trained on one sample. The output appears almost identical to the target image, as would be expected. This shows that the model is at least capable of producing the aforementioned fine details but is unable to generalise well enough to create a convincing result when provided with more data.

5.1.1 Performance over the course of training

Tensorboard (Google Brain, 2020b) is a graphing system created for visualising Tensorflow models as well as plotting scalars and other trackable data. The following Tensorboard graphs represent the accuracy, loss and learning rate of the model throughout training. These are tracked throughout the first 200 epochs of training. The initial run consisted of many more epochs of training 2,789 to be specific; however, this data had not been tracked. Training had followed a similar trend after 200 epochs.

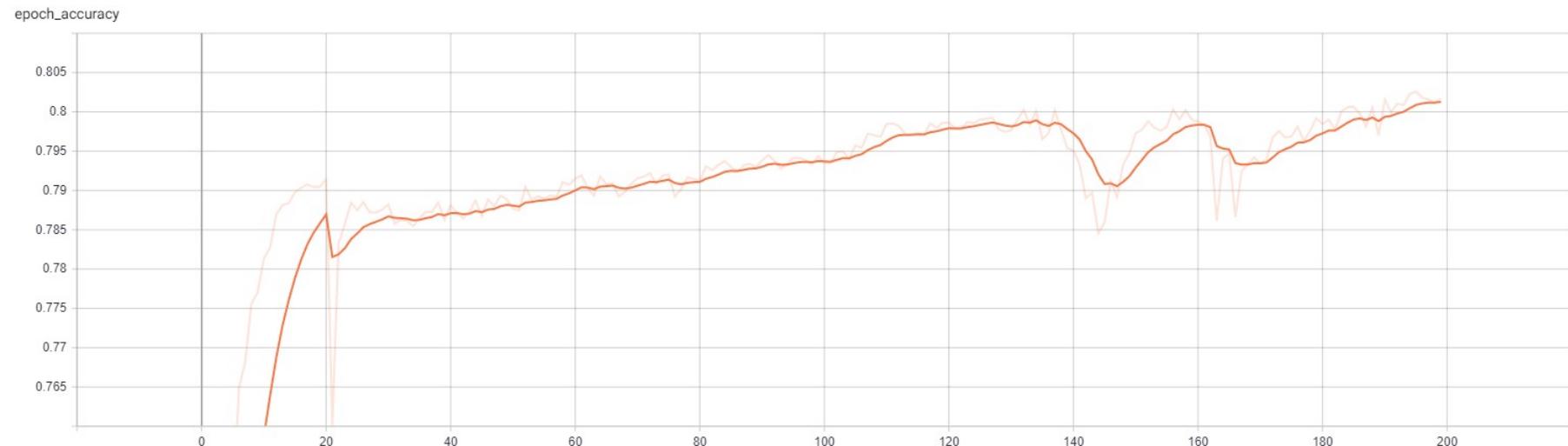


Figure 15: Model accuracy over the first 200 epochs of training.

The accuracy of the model mirrors the loss value throughout training. This value gives a percentage of output values that match their target values. Figure 15 shows the accuracy increasing rapidly within the first 20 epochs before increasing more slowly throughout the rest of training. At epoch 200 the accuracy value exceeds a value of 0.8 or 80%. Although this number may seem high, for a model that outputs images, a much higher accuracy would be required for images to be perceived as similar to their targets.

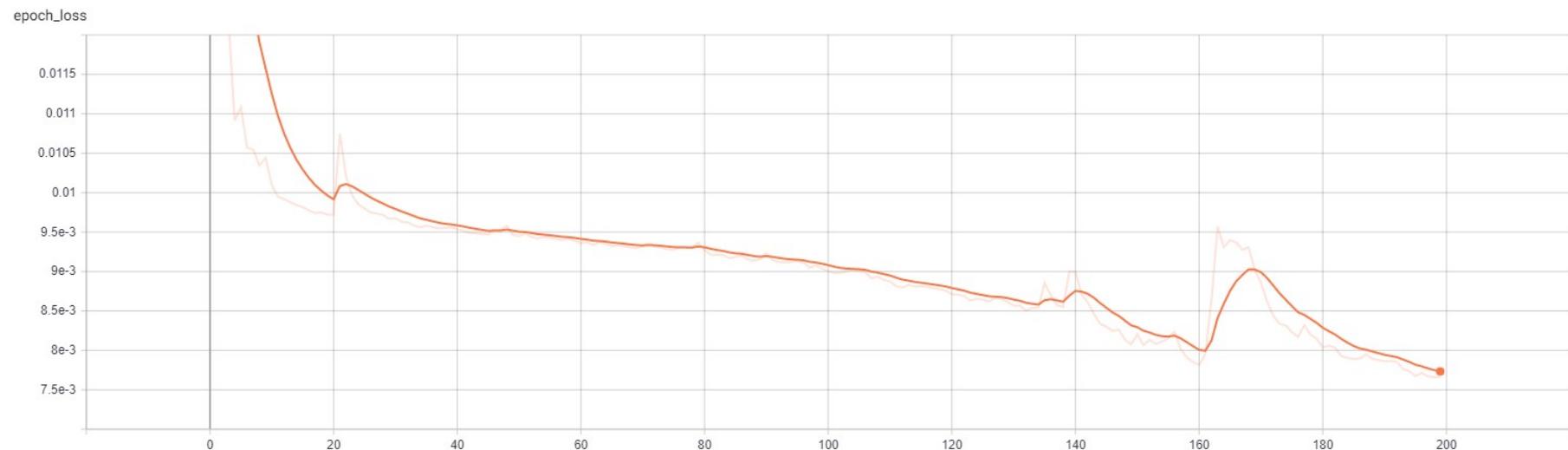


Figure 16: Model loss over the first 200 epochs of training.

Figure 16 shows the calculated loss throughout the first 200 epochs. Loss in this case is used to measure the difference between output and target data, therefore the lower the value the closer the model has come to matching the target image (Vasilev et al., 2019, p.30). The graph shows the loss rapidly decreasing within the first 20 epochs, then decreasing at a lower rate before jumping up between epochs 160 and 180. This is likely related to the learning rate, as changes to the learning rate (seen in figure 17) are reflected in the loss decreasing again at epoch 170.

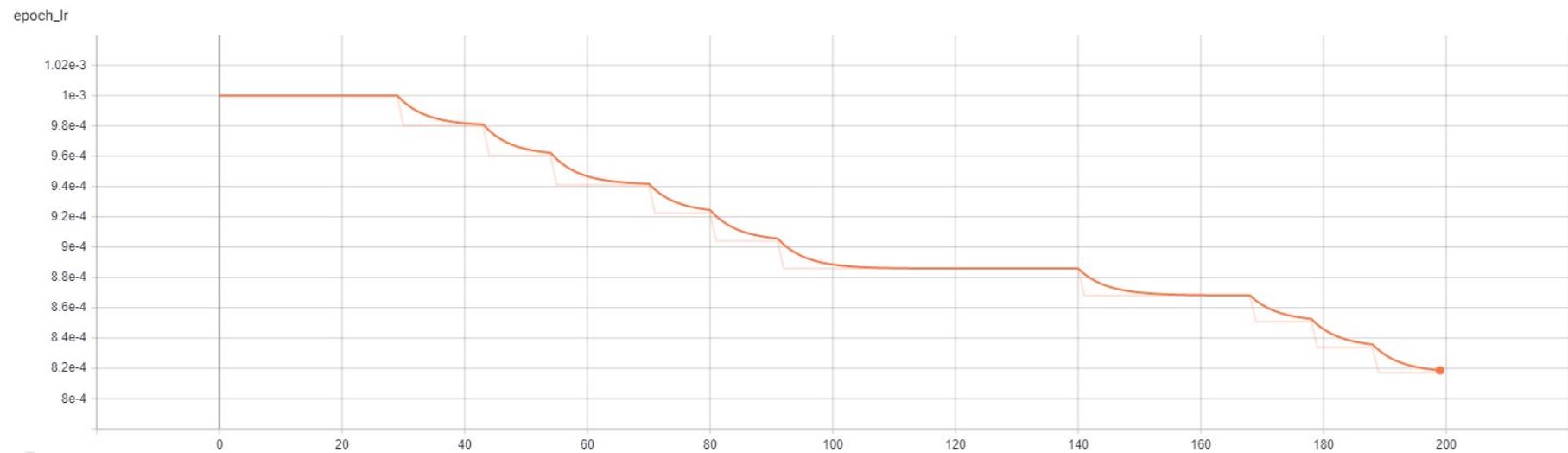


Figure 17: Model learning rate over the first 200 epochs of training.

Throughout training the learning rate was adjusted to improve model loss. This was done by tracking loss between epochs and reducing the learning rate after 10 consecutive epochs where the loss had not decreased. In this case learning rate begins at a value of 0.001 and is reduced by a factor of 0.98 at each reduction. This allows loss to continue to decrease as smaller parameter changes are made at each interval, narrowing down to the ideal parameters over time.

5.2 User Study

To test the usability of the system and gain a subjective view of the autoencoders efficacy, the product was presented to users. Unmoderated remote testing involves having participants complete testing without a moderator present, often manifesting as an application or web form that the participant accesses remotely. The benefit of this is that testing can be done faster and conducted regardless of location (Whitenton, 2019). When using this method, it must be ensured that users are able to complete given tasks based on the instructions provided. Due to current circumstances, testing can only be done remotely.

Using an unmoderated remote approach, participants were asked to complete a web form of three parts. The first section asked participants to rate the similarity between pairs of images using a Likert scale with values ranging between “very dissimilar” and “very similar”. This is done to subjectively ascertain the ability of the autoencoder to produce a likeness of the target images. This is done before the user has access to the product such that their opinions of the system or the results of their own designs do not affect how they perceive the images presented.

The Likert scale (Likert, 1932) is a common method of collecting the opinions of participants. This works by asking the participant to rate their level of agreement to a given statement out of 5 or 7 possible scale degrees (Brooke, 1996).

The second section asked participants to download the product and complete the task of first creating a design using the UI, then rating it using a Likert scale in the form of a star rating. Participants are asked to do this at least once but encouraged to do this as many times as they wish beyond that. Created designs, generated outputs and ratings are stored within a folder that the participant is asked to compress and upload to the web form.

The third section asked participants to complete the positive version of system usability scale (SUS) with the first question removed. The SUS is a standardised questionnaire that is used for assessing the usability of a product (Lewis, 2018). This was first established as a list of 10 Likert scale based questions consisting of alternating tonality (Brooke, 1996). Using the original version of the SUS, all 10 questions must be answered by a respondent, for any questions un-answered the centre point must be checked. However, it has been discovered that any one question can be removed from the SUS with little affect to the validity of the score. This can be done in cases where a question does not match the system being tested, as leaving it in could result in confusing the respondent. Calculating the SUS in this case means negating 1 from the scores of positively toned questions and negating 4 from negatively toned questions. The sum is then multiplied by 100/36 rather than 2.5 (100/40) used in the standard version (Lewis and Sauro, 2017). Additionally, it is established that the alternating tonality of the SUS can be removed, such that all questions have positive tonality. This can be employed in the case of remote unmoderated tests to avoid errors in answers, as the user may perceive the opposite meaning of a given question- an issue that can normally be addressed by a moderator (Sauro and Lewis, 2011).

An investigation into the use of Neural Networks for procedurally generating images
based on real-world map data

As the study uses an unmoderated remote approach the positive SUS is employed, with the first question- 'I think that I would like to use this system frequently' (Brooke, 1996), removed as the system is designed for one off testing of the autoencoder, where it is unlikely that a user would find a frequent use for it.

5.2.1 User study evaluation

A total of 4 participants took part in the study. These were all Games Design and Development students. The following will discuss results from this user study.

Table 3: Average user responses to image comparisons.

Question number	Image comparisons		Average scores
1			2.5
2			3
3			3.25

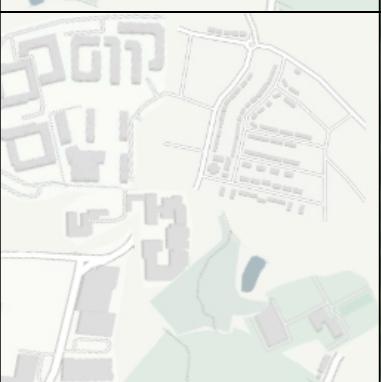
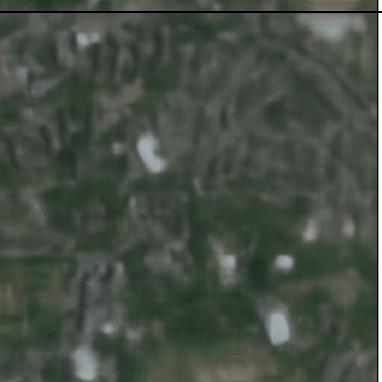
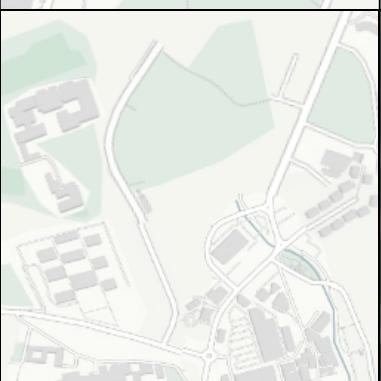
As shown in table 3, the third pair of images received the highest similarity rating with an average of 3.25. One feature of the images labelled "B" (the generated outputs), is that they appear undefined, with roads being more identifiable than specific features

An investigation into the use of Neural Networks for procedurally generating images
based on real-world map data

such as buildings. It could be proposed that the reason for the third pair being rated highest and the first pair being rated lowest is the identifiability of the roads. The third pair contains a large road that passes through the centre with sparse foliage running adjacent- a feature that the generated version replicates. Whereas the roads in image “B” of the first pair- although being visually distinct, have a uniform outline that appears to unsuccessfully represent buildings.

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

Table 4: Examples of user made designs compared to their generated outputs and user ratings. Ratings range between values of 0 and 4 with 4 being a high perceived likeness to their design and 0 being a low perceived likeness to their design.

User made sample	Generated Output	User Rating
		4
		2
		1
		2

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data



The raw value is from 1 to 5 with 0 being a non-rating, a value of 1 is negated from all results to form a scale of 0 to 4 for consistency with other scoring systems used in this study. In this case a non-rating, or value of 0 where the user had not rated the result remains a 0 to fit within the scale.

Of the examples shown, a notable trend between lower rated images is that they contain blank space. As the model had been trained on real-world map data, there had been no instances where it could have learnt to work with blank areas within inputs. Therefore, the output reflects this lack of information, which is noticeable to the user. Additionally, the model appears to affectively reproduce the shapes of roads, particularly those that are larger. But does not perform as well when reproducing buildings. In general, as observed in the initial comparisons between real data and the generated counterparts, the outputs are undefined and blurry. This can contribute to the smaller details such as buildings blending together.

The results of section 3- the positive 9 question SUS- are mostly positive with the lowest score being 88.9. This is calculated by negating 1 from the score of each question and summing the results. The resulting value is then multiplied by 100/36, approximately 2.8 to produce a number within the range of 0 to 100.

Table 5: SUS scores given by users.

User	Raw Score	Score	Average
User 1	36	100.0	95.1
User 2	32	88.9	
User 3	34	94.4	
User 4	35	97.2	

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

The results- shown in table 5- indicate an average score of 95.1. This value is likely biased due to the selected participants not being diverse in terms of ability (all participants were Games Design and Development students), coupled with the limited number of participants. To improve the accuracy, more participants with different levels of experience in computer systems would be required (Nielsen, 1996).

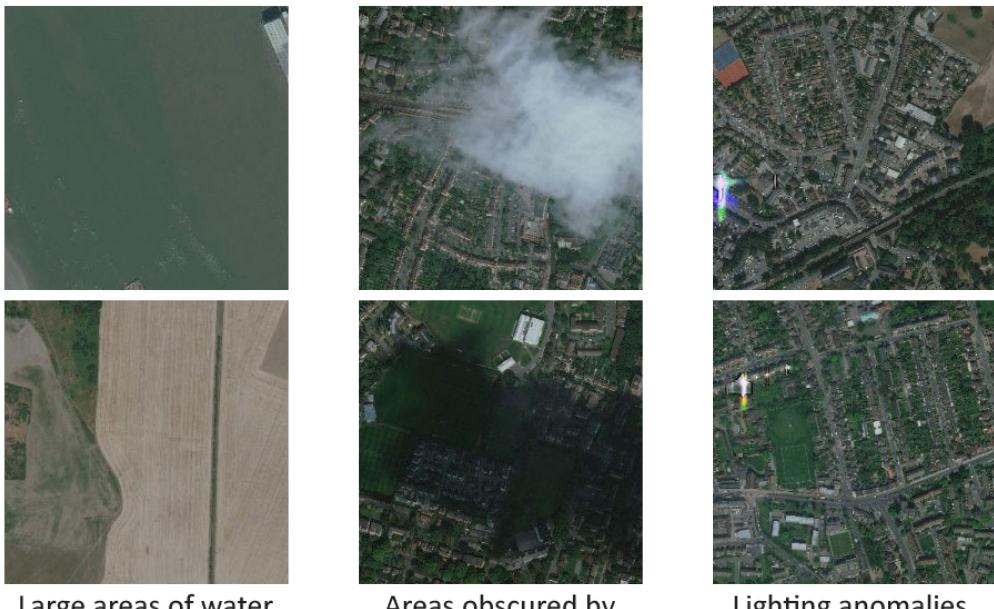
5.3 User Privacy Considerations

When carrying out the user study it is important to consider user privacy and data protection. The platform used for presenting and handling user testing is Google Forms(Google, 2020b), this provides features such as the ability to collect and store answers from users and exchange files. Google Forms requires participants to log in to upload files. As testing requires users to upload their generated images and ratings the user accounts of participants are recorded in the name of the files that they upload. This is of no use within the study as the particular user has no relevance to the results obtained. Nonetheless, results are anonymised when testing and evaluating- see table 4 and appendix A.3. Additionally, users are given explicit instructions on what files to upload to the form, ensuring that only the intended data is sent, see appendix A.1. All stored files uploaded by the users will be deleted as of the 30th of May 2020, with those presented in table 4 being anonymised.

5.4 Testing Conclusions

Results from the model testing show that although the model has been trained for 2,789 epochs, it has reached an accuracy value of 85% which is not ideal, this is reflected in user examples as well as examples taken from the training dataset. This results in blurry outputs in which roads can be identified but individual buildings and details are not. Observing the change in loss and accuracy over the first 200 epochs, it is clear that improvement is slow after the first 20 epochs. This could potentially be improved by changing the architecture (increasing the number of layers or convolutions) or improving the dataset.

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data



Large areas of water or empty land.

Areas obscured by shadows or clouds.

Lighting anomalies.

Figure 18: Anomalies found within the dataset used

Some anomalous data exists within the training dataset, as shown in figure 18. The dataset could be improved by removing such anomalies and also increasing the dataset size to provide more consistent data for the model to be trained on. Testing could also have been improved by performing validation testing at the end of each epoch using different data, a method that was unfortunately overlooked during training. It could be proposed that the use of a variational autoencoder (VAE) would result in more convincing results due to the level of abstraction that it would introduce, particularly in cases where blank space is included on input images.

The user study results reflect the aforementioned limitations, however results from the SUS show that users had relative success with using the Unity implementation. Although the SUS is limited to the 9 possible closed questions and it would have been more beneficial to include open ended questions that could have specified areas of improvement. The section of the form where participants were asked to rate the similarity between target and generated images did provide some insight into the weaker aspects of the generated results, however more examples should have been provided to participants- in order to gain further understanding.

CHAPTER 6: CONCLUDING DISCUSSION

In this chapter the research and topics explored throughout the document will be summarised (section 6.1), with research contributions and future development being discussed in sections 6.2 and 6.3 respectively.

6.1 Dissertation Summary

In this dissertation an attempt is made at applying an autoencoder to the task of neural style transfer. An autoencoder model has been developed and trained on 4000 image pairs consisting of roadmaps and satellite images of locations in and around The City of London. These are 256 pixels wide and tall. The resulting model has been evaluated based on its accuracy and loss values during and after training, the latter of which involving the use of 200 sample pairs not used in training prior. The model has also been tested by 4 users, using a program developed in the Unity engine that allows new roadmaps to be designed by the user. Users rated this system highly on a modified version of the SUS, this included 9 of the 10 original questions and was phrased with positive tonality, the average score received was 95.1 out of 100. Users were asked to rate how each of the generated results compared to the designs they created, results from this gave insight into weaker aspects of the models output, such as its lack of definition in smaller details. The resulting trained model is capable of producing vague approximations of inputs which is likely due to limitations in its architecture and the dataset.

6.2 Research contributions

The research in this dissertation provides one specific example of applying machine learning as a tool for creating content. This is done in the form of an autoencoder that re-interprets designed layouts. It is proposed that this concept be applied more generally with the aim of aiding designers in the creation of content for games. An application has been tested with an average rating of 95.1 on the SUS from a total of 4 participants. This application exemplifies one way that an end user can meaningfully interface with a machine learning model- to potentially generate new content with a high degree of control.

6.3 Future research and development

The implementation presented in this dissertation confirms that machine learning techniques can be applied to design tasks in a way that designers have a more direct form of control. The result, however limited, could be improved upon by developing the model further, improving datasets or using an alternate form of machine learning model. The core limitation is that this has only been applied to a single task- converting roadmaps into satellite style images. Using a VAE or the methodologies described by Gatys, Ecker and Bethge (2015), a more generalised approach could be taken. This dissertation explores how an end user could potentially interface with a machine learning model as a tool. Further user testing would be required in order to fully ascertain whether general users find such an interaction beneficial and reliable.

REFERENCES

- Brooke, J., 1996. SUS - A quick and dirty usability scale.
- CartoDB, 2014. *Introducing Positron and Dark Matter basemaps from CartoDB*. [online] Available at: <<https://carto.com/blog/getting-to-know-positron-and-dark-matter>> [Accessed 2 May 2020].
- Chollet, F., 2019. *Keras: The Python Deep Learning Library*. [online] Available at: <<https://keras.io/>> [Accessed 9 Dec. 2019].
- De-Icaza, M., 2019. *TensorFlowSharp*. [online] Available at: <<https://github.com/migueldeicaza/TensorFlowSharp>> [Accessed 19 May 2020].
- Dumoulin, V. and Visin, F., 2018. A guide to convolution arithmetic for deep learning. [online] pp.1–31. Available at: <<http://arxiv.org/abs/1603.07285>>.
- ESRI, 2020. *Spatial Analysis and Data Science*. [online] Available at: <<https://www.esri.com/en-us/arcgis/products/spatial-analytics-data-science/capabilities/machine-learning-ai>> [Accessed 19 May 2020].
- Gatys, L.A., Ecker, A.S. and Bethge, M., 2015. *A Neural Algorithm of Artistic Style*.
- Google, 2019. *Geo-location APIs | Google Maps Platform | Google Cloud*. [online] Available at: <<https://cloud.google.com/maps-platform/>> [Accessed 25 Oct. 2019].
- Google, 2019. *Google Colab*. [online] Available at: <https://colab.research.google.com/notebooks/basic_features_overview.ipynb> [Accessed 9 Dec. 2019].
- Google, 2020a. *Colaboratory Frequently Asked Questions*. [online] Available at: <<https://research.google.com/colaboratory/faq.html>> [Accessed 2 May 2020].
- Google, 2020b. *Google Forms*. [online] Available at: <<https://www.google.com/forms/about/>> [Accessed 14 May 2020].
- Google Brain, 2019a. *TensorFlow*. [online] Available at: <<https://www.tensorflow.org/>> [Accessed 25 Oct. 2019].
- Google Brain, 2020b. *Tensorboard*. [online] Available at: <<https://www.tensorflow.org/tensorboard>> [Accessed 14 May 2020].
- Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1, pp.448–456.
- Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M. and Lange, D., 2018. Unity: A General Platform for Intelligent Agents. [online] Available at:

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

- <<http://arxiv.org/abs/1809.02627>> [Accessed 18 Oct. 2019].
- Keras, 2020. *Callbacks API*. [online] Keras API Documentation. Available at: <<https://keras.io/api/callbacks/>> [Accessed 19 May 2020].
- Lewis, J.R., 2018. The System Usability Scale: Past, Present, and Future. *International Journal of Human-Computer Interaction*, 34(7), pp.577–590.
- Lewis, J.R. and Sauro, J., 2017. Can I Leave This One Out? The Effect of Dropping an Item From the SUS. *Journal of Usability Studies*, 13(1), pp.38–46.
- Likert, R., 1932. A technique for the measurement of attitudes. *Archives of Psychology*, 22 140, p.55.
- Van Der Linden, R., Lopes, R. and Bidarra, R., 2014. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1), pp.78–89.
- Merrell, P., Schkufza, E. and Koltun, V., 2010. Computer-Generated Residential Building Layouts. *ACM Transactions on Graphics*, 29.
- Morgan, 2016. *TensorFlow: How to freeze a model and serve it with a python API*. [online] Metaflow. Available at: <<https://blog.metaflow.fr/tensorflow-how-to-freeze-a-model-and-serve-it-with-a-python-api-d4f3596b3adc>> [Accessed 19 May 2020].
- NextGIS, 2019. *QuickMapServices*. [online] Available at: <https://plugins.qgis.org/plugins/quick_map_services/> [Accessed 2 May 2020].
- Nielsen, J., 1996. Putting the user in user-interface testing. *IEEE Software*, 13(3), pp.89–90.
- Nielsen, M.A., 2015. *Neural Networks and Deep Learning*. [online] Determination Press. Available at: <<http://neuralnetworksanddeeplearning.com/index.html>>.
- QGIS Development Team, 2020. *QGIS*. [online] Available at: <<https://qgis.org/en/site/index.html>> [Accessed 2 May 2020].
- Qiu, G. and Chen, J., 2018. Web-based 3D map visualization using WebGL. In: *Proceedings of the 13th IEEE Conference on Industrial Electronics and Applications, ICIEA 2018*. Wuhan, China: IEEE. pp.759–763.
- Rosebrock, A., 2018. *Keras Conv2D and Convolutional Layers*. [online] pyimagesearch. Available at: <<https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/>> [Accessed 2 May 2020].
- Sauro, J. and Lewis, J.R., 2011. When designing usability questionnaires, does it hurt to be positive? *Conference on Human Factors in Computing Systems - Proceedings*, (May 2011), pp.2215–2223.

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

- Unity Technologies, 2020a. *Texture2D*. [online] Unity Documentation. Available at: <<https://docs.unity3d.com/ScriptReference/Texture2D.html>> [Accessed 19 May 2020].
- Unity Technologies, 2020b. *Unity Engine*. [online] Available at: <<https://unity.com/>> [Accessed 19 May 2020].
- Vasilev, I., Slater, D., Spacagna, G., Roelants, P. and Zocca, V., 2019. *Python Deep Learning*. Second Edi ed. Birmingham: Packt Publishing.
- Whitenton, K., 2019. *Unmoderated User Tests: How and Why to Do Them*. [online] Nielsen Norman Group. Available at: <<https://www.nngroup.com/articles/unmoderated-usability-testing/>> [Accessed 13 May 2020].

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

APPENDIX A: PRIMARY DATA COLLECTION METHOD

The primary data collection method for this research consists of a web form of three parts: a comparison between generated and target images, direct product testing with ratings and individual samples saved, and a positively toned 9 question SUS.

Appendix A.1

Web form for user testing- image comparisons and product testing.

Section 1 of 3

User testing form- Map Generating Autoencoder

X ::

Thank you for taking the time to complete this form.

In this section you will be asked to compare the similarity between images.

How similar are the following two images?



Image A

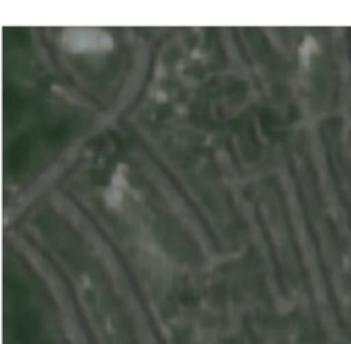


Image B

1

2

3

4

5

Very dissimilar

Very similar

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

How similar are the following two images?



Image A



Image B

1 2 3 4 5

Very dissimilar

Very similar

How similar are the following two images?



Image A

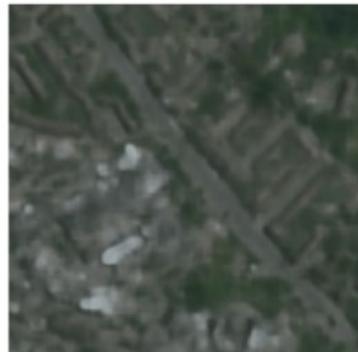


Image B

1 2 3 4 5

Very dissimilar

Very similar

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

Section 2 of 3

Product testing

In this section you will be asked to download and test a piece of software, you will also be required to upload a file.

Files you upload will be used to assess the product, and will not be tied to any other information you provide.

Getting set-up

Please download the supplied "ProductTestingBuild.zip" file and extract the folder within it.

FILE DOWNLOAD: <https://drive.google.com/open?id=105CUsZBZrdDH48vZ4tuBU6AQ2bwiWB6k>
(If the link doesn't let you download try it in incognito mode- for some reason the page just refreshes)

Run the "MapGenAutoencoder.exe" file found within.

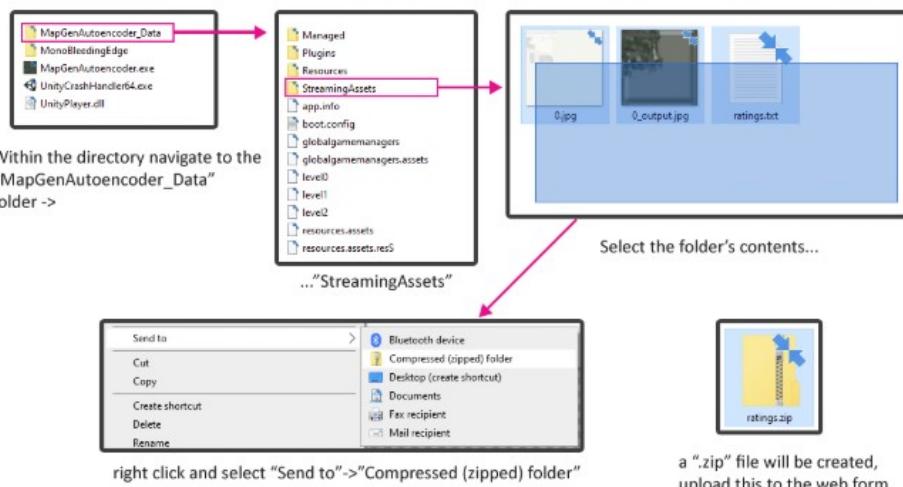
Please use the software to create a minimum of one design. You can create as many as you want beyond this.

Be aware that your designs are not being examined but rather the effectiveness of the system.

After creating a design you will be asked to rate the result, this is done by clicking on the stars. Clicking the "Back to menu", or "Back" buttons will save the result.

Then you can exit the software and complete the following steps to upload your test results.

How to provide testing results



Please upload compressed ".zip" file here *

Add file

An investigation into the use of Neural Networks for procedurally generating images
based on real-world map data

Appendix A.2

Version of the system usability scale (SUS) used. Consisting of 9 questions (Lewis and Sauro, 2017) each written in positive tone (Sauro and Lewis, 2011).

In this section you will be asked a set of questions about the product you have just used. Please answer these questions on a scale of 1-5 with 1 being strongly disagree and 5 being strongly agree.

Question 1. I found this system to be simple.

Question 2. I thought this system was easy to use.

Question 3. I think that I could use the system without the support of a technical person.

Question 4. I found the various functions in this system were well integrated.

Question 5. I thought there was a lot of consistency in this system.

Question 6. I would imagine that most people would learn to use this system very quickly.

Question 7. I found the system very intuitive.

Question 8. I felt very confident using the system.

Question 9. I could use the system without having to learn anything new.

Appendix A.3

Results from user study- file links are omitted.

Timestamp	How similar are the following two images? Question 1.	Question 2.	Question 3.	Please upload compressed ".zip" file here	1. I found the system to be simple.	2. I thought the system was easy to use.	3. I think that I could use the system without the support of a technical person.	4. I found the various functions in this system were well integrated.	5. I thought there was a lot of consistency in this system.	6. I would imagine that most people would learn to use this system very quickly.	7. I found the system very intuitive.	8. I felt very confident using the system.
15/5/2020 16:12:26	4	4	4		5	5	5	5	5	5	5	5
15/5/2020 16:29:29	3	4	4		5	5	5	4	4	5	4	4
15/5/2020 20:54:50	3	4	5		5	5	5	4	4	5	5	5
16/5/2020 17:25:21	4	4	4		5	5	5	5	4	5	5	5

An investigation into the use of Neural Networks for procedurally generating images
based on real-world map data

APPENDIX B: SOURCE CODE

Appendix B.1:

Loading input data and performing inference in Unity with Tensorflowsharp.

```
void RunNN(string fileName)
{
    //Set up Input
    var file = System.IO.File.ReadAllBytes(fileName);
    Texture2D inputImage = new Texture2D(256,256);
    inputImage.LoadImage(file);
    inputImage.Apply();
    TestImage.texture = inputImage;

    float[,,,] input_data = new float[1,256,256,3];

    for (int i = 0; i < inputImage.width; i++)
    {
        for (int j = 0; j < inputImage.height; j++)
        {
            Color colour = inputImage.GetPixel(i, j);
            input_data[0, i, j, 0] = colour.r;
            input_data[0, i, j, 1] = colour.g;
            input_data[0, i, j, 2] = colour.b;
        }
    }

    TFTensor input_tensor = input_data;

    Debug.Log(input_tensor.TensorType + " : tensor dtype");
    var ipshape = input_tensor.Shape;
    foreach (var item in ipshape)
    {
        Debug.Log(item);
    }

    //Run Inference
    using (var graph = new TFGraph())
    {
        graph.Import(AutoencoderModel.bytes);
        var session = new TFSession(graph);
        var runner = session.GetRunner();

        runner.AddInput(graph["input_node"][0], input_tensor);

        runner.Fetch(graph["output_node/Sigmoid"][0]);

        output_data = runner.Run()[0].GetValue() as float[,,,];
        readyToEncode = true;
        Debug.Log(output_data);
        session.Dispose();
        graph.Dispose();
    }
}
```

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

Appendix B.2:

Encoding the output from the autoencoder in Unity, with converting normalized data to 0-255 values.

```
void EncodeOutput(float[,,,] o_data)
{
    //Receive and return Output
    //Convert normalized output to rgb
    byte[,,,] output_denormalized = new byte[1, 256, 256, 3];

    for (int i = 0; i < 256; i++)
    {
        for (int j = 0; j < 256; j++)
        {
            //Red Channel
            output_denormalized[0, i, j, 0] = DenormalizeColourValue(o_data[0, i, j, 0]);
            //Green Channel
            output_denormalized[0, i, j, 1] = DenormalizeColourValue(o_data[0, i, j, 1]);
            //Blue Channel
            output_denormalized[0, i, j, 2] = DenormalizeColourValue(o_data[0, i, j, 2]);
        }
    }

    //Export image
    Texture2D outputImage = new Texture2D(256, 256);
    for (int i = 0; i < outputImage.width; i++)
    {
        for (int j = 0; j < outputImage.height; j++)
        {
            byte r = output_denormalized[0, i, j, 0];
            byte g = output_denormalized[0, i, j, 1];
            byte b = output_denormalized[0, i, j, 2];
            Color32 colour = new Color32(r, g, b, 255);
            outputImage.SetPixel(i, j, colour);
            if(i == 6 && j == 12)
            {
                Debug.Log(colour);
            }
        }
    }

    //Debug.Log(outputImage.GetPixel(150, 150).ToString());
    outputImage.Apply(true, false);

    byte[] byteArrayOutput = outputImage.EncodeToJPG(100);
    System.IO.File.WriteAllBytes(Application.dataPath +
    "/samples/" + sessionID.ToString() +
    "_output" + ".jpg", byteArrayOutput);

    TestImage.texture = outputImage;
}

byte DenormalizeColourValue(float normalized_colour)
{
    float val = normalized_colour * 255;
    byte outputVal = (byte)val;
    return outputVal;
}
```

An investigation into the use of Neural Networks for procedurally generating images
based on real-world map data

Appendix B.3:

Autoencoder training generator code.

```
#Import training data from google drive
import os
os.environ['KERAS_BACKEND'] = 'tensorflow'
from keras.preprocessing.image import ImageDataGenerator

rootDir = 'gdrive/My Drive/training data/'

batchSize = 1
listImgs = os.listdir(path=rootDir + 'a_data/a/')
numOfImgs = len(listImgs)

gen = ImageDataGenerator()

#Training Data Generator
train_datagen_a = ImageDataGenerator(rescale=1./255)
train_datagen_b = ImageDataGenerator(rescale=1./255)

def train_images():
    train_generator_a = train_datagen_a.flow_from_directory(
        rootDir + 'a_data',
        target_size=(256,256),
        batch_size=batchSize,
        shuffle=False,
        class_mode='input',
        color_mode='rgb')

    train_generator_b = train_datagen_b.flow_from_directory(
        rootDir + 'b data',
        target_size=(256,256),
        batch_size=batchSize,
        shuffle=False,
        class_mode='input',
        color_mode='rgb')

    while True:
        a = train_generator_a.next()
        b = train_generator_b.next()
        #print(a[0])
        yield a[0], b[0]
```

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

Appendix B.4

Model code in Keras.

```
#ENCODER
inp = Input((256, 256, 3), name="input_node")
e = Conv2D(64, (7, 7), padding='same', activation='relu')(inp)
e = BatchNormalization()(e)
e = MaxPooling2D((2, 2))(e)

e = Conv2D(64, (3, 3), padding='same', activation='relu')(e)
e = BatchNormalization()(e)

e = Conv2D(128, (3, 3), padding='same', activation='relu')(e)
e = BatchNormalization()(e)
e = MaxPooling2D((2, 2))(e)

e = Conv2D(256, (3, 3), padding='same', activation='relu')(e)
e = BatchNormalization()(e)

e = Conv2D(256, (3, 3), padding='same', activation='relu')(e)
e = BatchNormalization()(e)
e = MaxPooling2D((2, 2))(e)

#DECODER
d = UpSampling2D((2,2))(e)
d = Conv2D(256, (3, 3), padding='same', activation='relu')(d)
d = BatchNormalization()(d)

d = Conv2D(256, (3, 3), padding='same', activation='relu')(d)
d = BatchNormalization()(d)

d = UpSampling2D((2,2))(d)
d = Conv2D(128, (3, 3), padding='same', activation='relu')(d)
d = BatchNormalization()(d)

d = Conv2D(64, (3, 3), padding='same', activation='relu')(d)
d = BatchNormalization()(d)

d = UpSampling2D((2,2))(d)
d = Conv2D(64, (3, 3), padding='same', activation='relu')(d)
d = BatchNormalization()(d)

decoded = Conv2D(3, (1, 1), activation='sigmoid', padding='same', name="output_node")(d)
ae = Model(inp, decoded)
ae.summary()
```

An investigation into the use of Neural Networks for procedurally generating images based on real-world map data

Appendix B.5

Freezing keras model for serving in Unity.

```
tf.keras.backend.set_learning_phase(0)

tf.keras.backend.get_session().run(tf.global_variables_initializer())

ae.load_weights(rootDir + "checkpoints/cp13p14-0495.ckpt")

tf.saved_model.simple_save(tf.keras.backend.get_session(),
                           outFolder + "/simple_6",
                           inputs={"input_node": inp},
                           outputs={"output_node": decoded})
```



```
#Freeze graph

#The first two arguments are not necessary as a saved model is instead being passed.

freeze_graph.freeze_graph(input_graph=("./" + outFolder + GRAPH_NAME + ".pbtxt",
                                       input_checkpoint="./" + outFolder + GRAPH_NAME + ".ckpt",
                                       input_saver=None, input_binary=False,
                                       output_node_names=output_node_name,
                                       restore_op_name="./" + outFolder + "restore_all",
                                       filename_tensor_name="save/Const:0",
                                       output_graph="./"+outFolder+"frozen_"+GRAPH_NAME+".bytes",
                                       clear_devices=True,
                                       initializer_nodes="",
                                       checkpoint_version=saver_pb2.SaverDef.V1,
                                       input_saved_model_dir= outFolder + "/simple_6")
```