Three-dimensional imaging from photographs

B.Sc. Year 4 Research Project

Heriot-Watt University

Submission date: 07/05/2021

Student: L. Hawkes (H00267382)

Supervisor: Dr. J. Leach

Heriot Watt University Physics Department – Own Work Declaration

This sheet must be filled in and included after the title page of your project report. Work will not be marked unless the following is included.

I confirm that all this work is my own except where indicated, and that I have:

· Clearly referenced/listed all sources as appropriate
· Referenced and put in inverted commas all quoted text (from books, web, etc)
· Given the sources of all pictures, data etc. that are not my own
· Not made any use of the essay(s) of any other student(s) either past or present
· Not sought or used the help of any external professional agencies for the work
· Acknowledged in appropriate places any help that I have received from others

Name: **Luke Hawkes**
Student Number: **H00267382**
Course/Programme: **Master of Physics in Physics**
Title of work: **Three-dimensional imaging from photographs**
Date: **07/05/2021**

Acknowledgements

I would like to thank Dr. Jonathan Leach for overseeing this project and providing guidance on how to approach the challenges presented in this report. I would also like to thank Alice Ruget for helping me debug particularly confusing code as well as teaching me many of the fundamentals required for creating neural networks. For proof reading and extra guidance I would like to thank my friends and family.

Contents

1 Introduction **6**
- 1.1 Aims . . . 6
- 1.2 Photogrammetry . . . 6
- 1.3 Neural Networks . . . 7
- 1.4 Choice of Point Clouds . . . 9

2 Related Work **9**

3 Materials: Data Set preparation **11**
- 3.1 Neural Network one - PointNet Structure . . . 11
- 3.2 Neural Network two - Autoencoder Structure . . . 11
- 3.3 Neural Network three - Image to Point Cloud Structure . . . 11

4 Designing the Network Structures **12**
- 4.1 Neural Network one - PointNet Structure . . . 12
- 4.2 Neural Network two - Autoencoder Structure . . . 13
- 4.3 Neural Network three - Image to Point Cloud Structure . . . 14
- 4.4 Loss Function . . . 15
  - 4.4.1 Loss Function for Network one . . . 15
  - 4.4.2 Loss Function for Networks two and three . . . 15
  - 4.4.3 Surface Penalty for Network three . . . 16

5 Experimental methods and results **16**
- 5.1 Neural Network one - Classification with PointNet . . . 16
- 5.2 Neural Network two - AutoEncoder . . . 17
- 5.3 Neural Network three - Image to Point Cloud . . . 18
- 5.4 Applying surface penalty . . . 21

6 Discussion of Results **21**

7 Conclusion **22**

8 Further Work **23**
- 8.1 Multi-image networks and semantic scene segmentation . . . 23
- 8.2 Alternative Loss functions . . . 23

9 Appendix **24**
- 9.1 More Results . . . 25
- 9.2 Pseudo-code . . . 25
- 9.3 Gantt Chart . . . 26
- 9.4 Risk Assessment . . . 27
- 9.5 Summary of Summer Project . . . 29

Abstract

Three-dimensional (3D) point clouds are a widely used file type made up of a set of coordinates defining where points should be placed in space to create an object. These point clouds can be generated by Light Detection and Ranging (LIDAR) sensors, from the depth images gathered by Single-Photon avalanche diode (SPAD) arrays and from deep neural networks. The generated data can then be used in applications such as autonomous vehicles, sensing for robotics and Virtual Reality (VR).

The goal of this project was to study 3D point clouds and their integration into neural networks for three major tasks: (1) categorise objects from their point cloud representation, (2) building a reconstructive point cloud autoencoder, (3) generate a 3D point cloud from a two-dimensional intensity image. All were tested using the same dataset.

For each task, one novel neural networks was coded in Python using Tensorflow API. The network of task (1) achieved 81.8% accuracy when classifying a validation dataset into 10 categories, compared to state-of-the-art 89.2% in the PointNet paper[1]. The reconstructive autoencoder of (2) was capable of reproducing an input point cloud. Finally, for task (3) different 2D to 3D transformation networks were coded and tested. It was necessary to render intensity images of the whole dataset using Blender for this task. A UNet was chosen as the preferred structure as it reproduced 3D objects that were closest to the real object (the ground truth), as measured by a low chamfer distance. This UNet was then applied to a second dataset of human point clouds, for easier visual identification of outlying points.

1 Introduction

1.1 Aims

This project investigates generating a 3D object given an input two-dimensional intensity image. An intensity image is a standard photographic image that could be taken with any camera. There are three main hurdles to overcome in order to reach the end goal:
1. Make a Neural Network capable of understanding 3D files.
2. Make a Neural Network capable of producing 3D files.
3. Make a Neural Network capable of transferring the main features of an image to a 3D file.

Different approaches for the transformation of 2D to 3D images are discussed below. The end goal is to recreate state-of-the-art techniques and if possible contribute to them through new suggestions.

1.2 Photogrammetry

Photogrammetry[2] takes a combination of input images and precise measurements and uses them to generate a 3D object or scene. In order to manually generate a 3D object, reference photographs are intensely studied to recreate an exact replica in a modelling software. This process can take hours[3]. Techniques to automate object generation have evolved using hand crafted algorithms such as Bayesian and other specialised networks for mapping urban areas[4]. Bayesian networks take a probabilistic approach to segmenting images and obtaining 3D data, they require contextual information provided by surrounding pixels to build a height-map. However these networks are not very precise nor are they very advanced.

More recently the affordability of Light Detection and Ranging (LIDAR) sensors has decreased the requirement for specialised networks by generating full point clouds of a given scene with sub-microsecond delays. Point clouds are a type of 3D file where information about the scene is stored as a set of 3D coordinates, sometimes including colour values as a 4th dimension. The points act like pixels in 3D space so that high resolution captures of objects can be viewed from all angles.

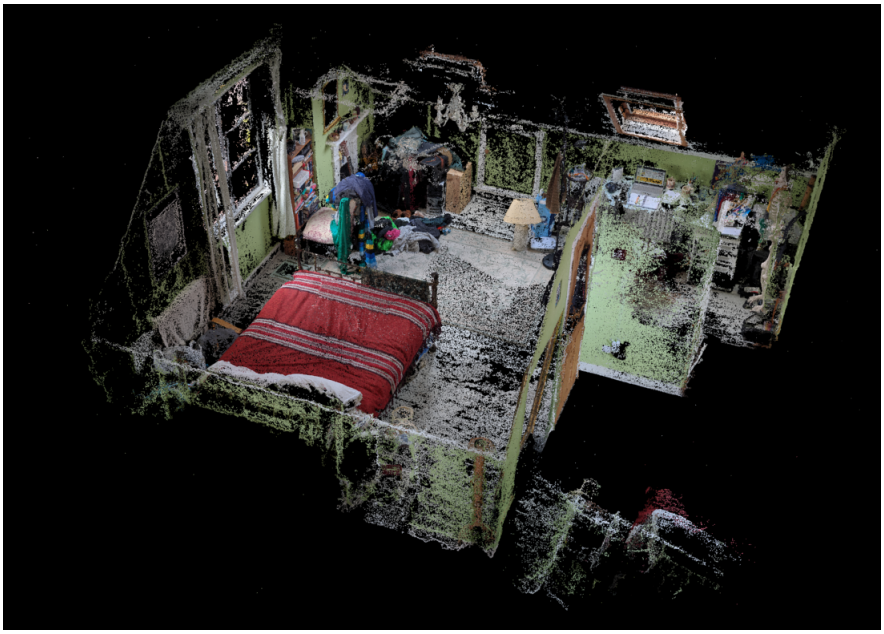

Figure 1: An example point cloud of a bedroom. Each 3D point is stored in the file alongside its colour value. This point cloud is not my own[5].

LIDAR generated point clouds are a crucial technology in the development of autonomous vehicles due to their high resolution and fast sampling rate. As the results could fundamentally

affect the safety of our future roads, the output must be in no way vague or prone to misinterpretation. However, LIDAR point clouds become increasingly sparse the further an object is from the sensor, potentially confusing classification algorithms in the vehicle. Consequently, intensity images have been reintroduced in an effort to identify far-off objects that are indistinguishable to the LIDAR sensor[6].

Another field where photogrammetry has been extensively used is for industrial and professional Virtual Reality (VR) applications such as producing realistic anatomical models for medical professionals. In this application LIDAR sensors produce large amounts of noise and bad polygons when a point cloud is turned into a polygonal mesh (a surface that can be interacted with). These interpretation issues due to insufficient number of LIDAR points presented here are mostly due to aliasing from an under-sampled LIDAR point cloud and can be smoothed out with post-processing[7]. Photogrammetry in the virtual reality industry is generally performed by expert model creators and artists making high quality meshes unavailable to standard consumers. Introducing a mesh creating software using only intensity images - which are available to anyone with a camera - would benefit the industry.

This project takes a step back from the use of LIDAR sensors or any laser scanning techniques and focuses instead on improving the quality of automated object generation from intensity images.

1.3 Neural Networks

Neural Networks have been suggested as an approach to photogrammetry, representing a compromise between requiring additional equipment such as LIDAR and the higher degree of error in the end results of Bayesian networks.

Neural Networks typically act as a single function on a statistical distribution by considering a set of linear functions (the weights and biases of the network) and applying a non-linear constraint to each of these functions (the activation function). The result is a function (the Neural Network) which can perform analysis or reconstruction by feature-extraction on the initial distribution. The function must then be "trained" such that the values of the weights and biases perform appropriate feature extractions on any given initial distribution. In the majority of Neural Networks this is completed through backpropagation where the output of the network is compared to a ground truth using a loss function. The loss function must compare the output and true distributions based on the context of the problem.

A network often consists of a set of layers between the input and output, each layer performs an additional aspect of feature extraction which becomes more complex further into the network. These layers are known as hidden layers because there is not always an apparent physical meaning to their numerical values, hence they are hard to interpret and are "hidden" in their functionality.

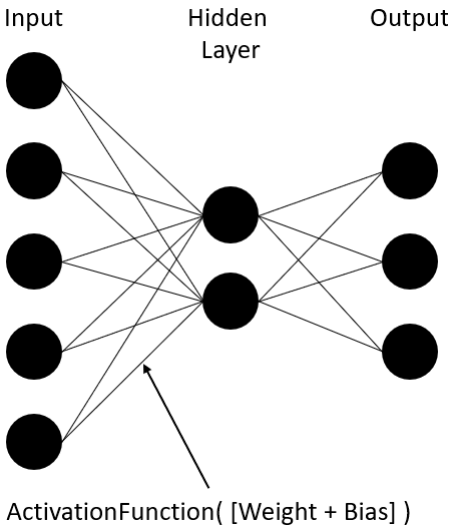

Figure 2: A schematic of a neural network depicting a hidden layer. The input nodes may represent a string of data, pixels in an image or even points in a point cloud. The output nodes must be of the correct shape and dimensions as the ground truth from a dataset.

The structure of a neural network is defined by the position of the layers that make it up, as well as how they are connected together. Choosing the correct structure is crucial to making a problem-specific neural network. An autoencoder is best for reconstructing missing information (see section 2), a UNet is better suited for deconstructing input data into parts, and a Generative Adversarial Network (GAN) is used for creating new, realistic information from a given description. The structures of the UNet and Autoencoder are discussed in depth in section 4. The GAN is briefly discussed in section 9.5.

The choice of layer type also plays a role in Neural Network design as they are the building blocks of any network. There are two main types of layer: Convolutional layers and Dense Layers which must be chosen based on the application. Dense layers work by taking in a weighted value from every node in the previous layer, a bias is added to the final value making each neuron equivalent to a linear function. The dense layer is able to pick out features by combining linear functions to create new unique functions. Convolutional layers work by scanning across input data and highlighting patterns that match the weights in the layer. As the network trains, the convolutional weights become more defined as they pick out more generalised patterns.

Another important aspect of neural networks is the activation function. The weights of both convolutional layers and dense layers represent linear functions which makes representing non-linear functions difficult. A good approximation to a non-linear function could be produced given enough layers but a better approach is to use activation functions. The concept is analogous to a synapse in a biological neuron, in this case the axon has transmitted an electrical signal which needs to be converted into an understandable format for the next neuron to receive. For this the chemical Acetylcholine is released when an action potential threshold is reached. Likewise an activation function provides boundary conditions through a non-linear function which acts on the whole layer. The output of a layer with an activation function is therefore a non-linear function and can converge to a non-linear output distribution with fewer layers.

A large number of activation functions are available, the oldest and best known being the sigmoid function which has been in use since the 1970s (and known for its first application in an 1838 study of population growth[8]). The sigmoid function benefits from being differentiable everywhere such that gradient descent through backpropagation can be calculated easily. Most networks have now replaced sigmoid with a hyperbolic tangent as it is centered on 0 which makes negative and positive gradient descent simpler. A common choice in image processing is the Rectified Linear Unit or ReLu which sets negative input values to 0 and any positive value returns itself. The ReLu is a relatively new activation function, first proposed in 2009 and is not differentiable at $x = 0$ meaning that algorithmic gradient descent needs to be used. The activation function used throughout this project was the Swish activation function which was

proposed in 2017 [9] and returns similar values to the ReLu but is continuously differentiable everywhere meaning that convergence to a solution is often smoother and faster.

1.4 Choice of Point Clouds

Prior to 2017 a popular technique in generating objects from images was by a mesh or voxel representation of the given image. A common issue that arises is that a mesh or voxel representation will create quantisation artefacts so the end result will not appear continuous, this can be seen in Figure 3.

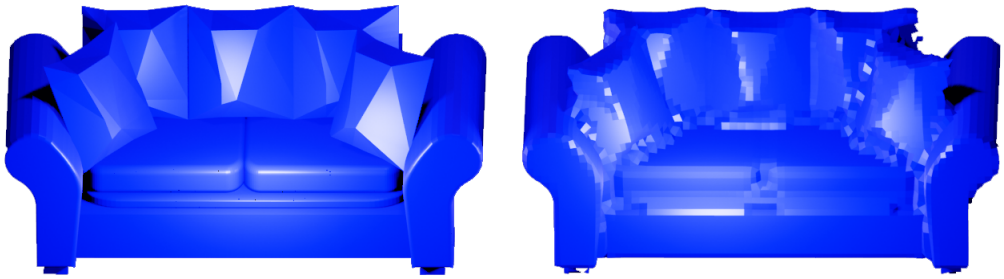

Figure 3: Left: A 3D mesh of a sofa from the ModelNet 10 dataset, Right: The same sofa re-meshed in Blender as a voxel representation, quantisation artifacts are present and edges are blended together where they should be sharp.

In 2017, PointNet[1] discussed the benefit of using a point cloud representation of the object, removing the issue of quantisation artefacts and creating a more malleable set theoretic interpretation. The paper focused on solving classification problems but lays the basis for generating any other network which has a 3D point cloud input or output. Throughout this project both input and output point clouds were required, PointNet was therefore a guiding light in how to address this problem.

2 Related Work

When making an image-to-object network it is necessary to ensure both image-to-image and object-to-object aspects are understood. Standard image based autoencoders[10] commonly use convolution layers for the encoder and decoder as this preserves the proximity between pixels needed for recognising shapes. In an image the pixels that are next to each other are spatially related which means comparing ground truth and prediction pixels can be used to assess how poorly a network is performing. An example of an image based autoencoder can be seen in Figure 4 where denoising has been trained into the network by learning the features which make up a dog. It is important to note that an image of another animal, object or scene would not be denoised as efficiently and may end up with distortions. When training a neural network the dataset used for training completely shapes how it reacts to new input images.

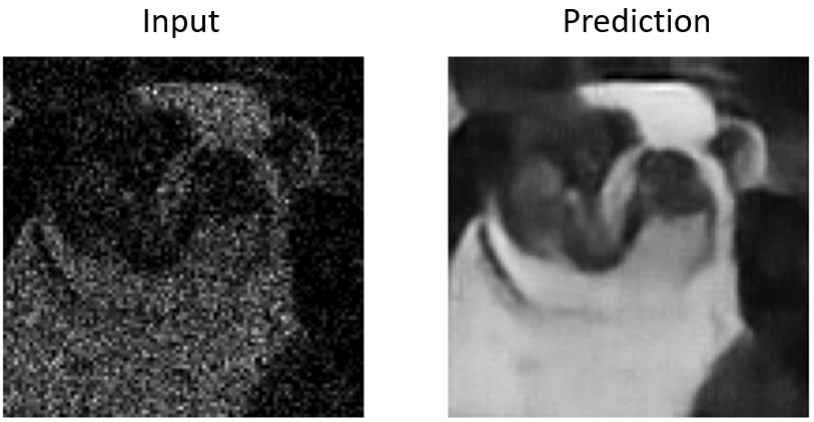

Figure 4: A convolutional image-to-image autoencoder capable of denoising an input image. This network has a latent space of 128 kernels of size 5x5 and has been trained for 100 epochs on the Dogs vs. Cats dataset available on Kaggle. Made during my summer project, see section 9.5.

A good example of an image based network that also demonstrates knowledge of three dimensions is Generative Radiance Fields (GRAF)[11]. This approach uses a GAN and applies it to solve interpolation of 3D rotation. This benefits from the sharp results that GANs produce and generates new intensity images as if they were taken from a different angle. The GAN generates and then renders a voxel representation of the input image in order to rotate it, meaning the network steps from 2D to 3D and then back to 2D in order to create a prediction. The resulting image is heavily distorted by this transformation which limits how permanently a rotation can be held in place. For example any camera movement in the input images would be replicated in the output meaning that a video sequence will not track an object. If the voxel or point cloud representation was used instead then the camera would have no issue tracking the object as the 3D space would be pre-defined. Nevertheless, this work offers a good starting point in learning about 3D imaging from photographs. Other work taking similar approaches to GRAF include PLATONICGAN[12] and Hologan[13] which all present novel networks capable of rotating an image in 3D.

Point cloud autoencoders are required to be invariant under permutation, a problem that is encountered in GRAF, PLATONICGAN and Hologan. The position of a point can be written anywhere in the set and therefore adjacent points in a set are not necessarily spatially related. The solution to this issue is to use a metric known as the chamfer distance which is discussed in more depth in section 4.4. A more robust metric is the Wasserstein metric or Earth Movers distance which has been applied to multiple point cloud networks[14] as it performs more intuitive transformations in 3D space resulting in both quantitative and qualitative improvements. The Wasserstein metric has become a popular metric to use in machine learning due to its use in a range of state of the art GANs based on the Wasserstein Gan paper[15]. The Wasserstein metric compares distributions by determining how much "work" needs to be done in order to transform one set to another. This creates a solution for the question how much work is needed to move one pile of dirt to another configuration and so is called the Earth Movers distance. The Wasserstein metric can be expressed as:

$$W(S_1, S_2) = \inf_{\gamma \in \Pi(S_1, S_2)} \mathbb{E}_{(x,y)\ \gamma}[|| \ x - y \ ||] \tag{1}$$

Where S_1 and S_2 represent the point sets, $\mathbb{E}$ represents Euclidean space (which is used in the majority of cases) and γ represents the amount of the distribution which must be transported. As this is a problem of finding the global minimum or infimum of the set there is no definitive solution until all possibilities have been exhausted making the metric very computationally expensive. However, a number of approximations can be made to decrease the scale of the problem as presented by Wasserstein Gan.

3D point cloud reconstruction from intensity images has been applied as early as 2016[14] before the PointNet paper was released. It is shown that a full 3D reconstruction can be achieved from a single image after a segmentation mask is applied to separate the object from the background. Segmentation networks require a fully labelled and masked dataset alongside the 3D model dataset. The scale of producing such a dataset did not lie within the scope of this project.

Not all previous work has been based on point clouds or voxel representations. AtlasNet[16], which the authors describe as a Papier-Machê approach to learning 3D surfaces, predicts projections from the initial intensity image and applies them to a malleable mesh. The mesh is then created out of multiple segments which gives it the name Papier-Machê as the algorithm "glues" these segments together. A limitation to this work is that the final product is a collection of meshes rather than a seamless model.

3 Materials: Data Set preparation

For all approaches the Princeton ModelNet 10 dataset was used[17]. The ModelNet 10 dataset contains 10 simple labelled categories to train on and a large dataset for each category. This dataset contains 5000 files and every file is in the .off format.

For the final 2D image to 3D point cloud network a human dataset was also used, generated by the Semantic Parametric Reshaping of Human Body Models - the authors of SPRING[18]. This dataset was chosen for qualitative purposes, a mistake on a 3D human point cloud is easier to spot than an error on a chair or a bathtub. The SPRING dataset contains 3000 files split evenly between gender and comes in .obj file format.

Trimesh Python library was used to obtain a uniformly distributed 3D point cloud from every mesh in the datasets[19].

3.1 Neural Network one - PointNet Structure

The aim here was not to create a one-shot classification network but instead to optimise the network based on a larger dataset. Labels were generated dependent on the category to which the point cloud belonged.

3.2 Neural Network two - Autoencoder Structure

The Princeton ModelNet 10 dataset was also used in the Autoencoder experiments. In these experiments only the 3D point clouds were required.

3.3 Neural Network three - Image to Point Cloud Structure

Every mesh had to be imported into a 3D renderer, resized and photographed with the camera at various distances and angles. Blender software was chosen for this process so Python automation could be run on the large dataset. Every picture contained the whole mesh with no obscuring masks in front, three different randomly assigned angles were taken of each mesh with only one angle being used per training epoch. Blender does not allow imports of .off files so an extension (the Blender OFF Addon[20]) was required to make this dataset compatible with the Princeton ModelNet 10.

When importing the images for training, a small amount of Gaussian noise was added on top of the images for augmentation. The point sets were also augmented by a small amount of Gaussian noise without disturbing their general shape. All point clouds were normalised to fit within a 1x1x1 cube independent of what percentage of the intensity image they occupied.

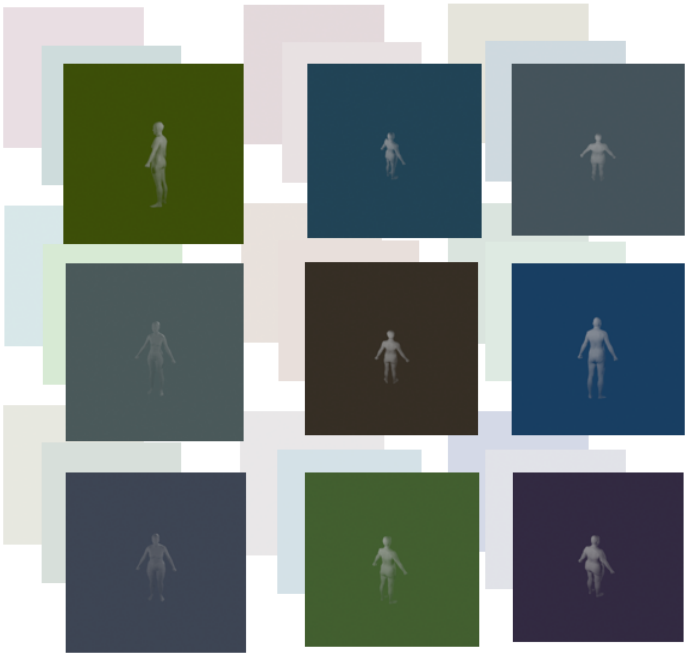

Figure 5: Examples from the final dataset of randomly generated images produced using Blender. The coloured background adds some additional augmentation to the dataset.

4 Designing the Network Structures

Three neural networks are proposed to solve the objectives stated in 1.1.

The first approach aims to prove that 3D files can be interpreted using a neural network.

The second approach aims to re-create an input 3D file, this will prove that 3D files can be produced by neural networks.

The third approach aims to combine all of this knowledge and generate a full 2D to 3D neural network.

4.1 Neural Network one - PointNet Structure

A basic neural network classifier will take any given picture or object and return a set of probabilities. The highest probability is the given label. For a network to be useful it must be able to sort through large amounts of data with very high accuracy. Networks have become very adept at recognising handwritten numbers such as those in the MNIST dataset, the aim of PointNet is to reproduce a similar accuracy with point clouds.

The full source-code for PointNet can be found at [21], this was extensively re-written to fit the scenario.

PointNet acts as a classification tool for unordered 3D point sets and works by either segmentation of whole scenes or through classification of separated individual objects. When working on a whole scene the network will return labels on each point individually, often given as a single colour such that some parts of the scene are coloured in to represent the objects within them. When working on a single object the network can be used to generate labels. The requirement for this project did not involve analysing whole scenes, consequently the structure of PointNet was reduced.

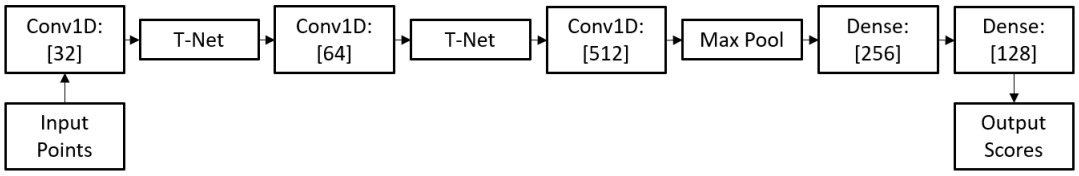

Figure 6: The PointNet structure is made up of a symmetrical max pooling function as well as "T-Nets" which perform feature extraction by attempting to create an affine transformation matrix

The PointNet structure shown in Figure 6 is reduced considerably from the structure given in the original PointNet paper. Before the maxpool function 1 dimensional convolutional layers with a 1x1 kernel size were used to change the filter dimension and perform faster than dense layers. The dense layers after the maxpool function were used to reshape the data into a coherent labelling system where the Conv1D layers would have output an incorrect shape. The maxpool function reduces the latent dimension by pooling together nodes and averaging their value.

The aim in the experimental section was to show that this structure can identify point clouds and therefore also show that point clouds are a reasonable approach for future work.

4.2 Neural Network two - Autoencoder Structure

Autoencoders are able to pull the features out of images even when they aren't apparent to our eyes. A good test to check if an autoencoder is working is to see if it can reproduce an exact copy of whatever was put in.

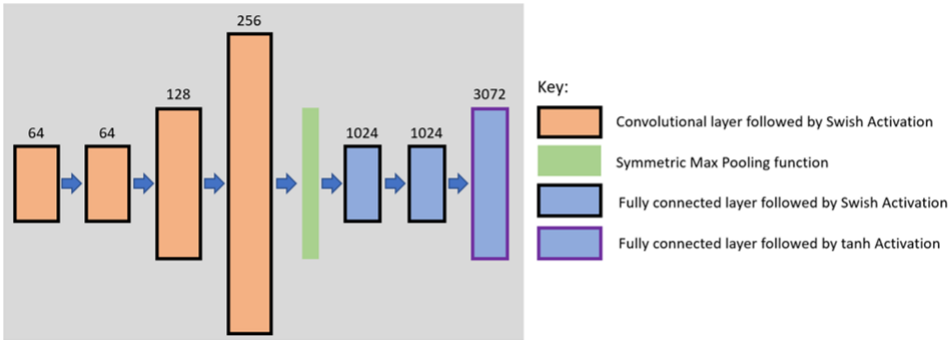

Figure 7: Point Cloud Autoencoder for a 1024 point output. The activation functions here are represented by the colour of the outline - black is a swish activation function, purple is a hyperbolic tangent activation function.

The code was based loosely on[22], although the structure was changed to reduce the time taken for execution.

The model follows the principles behind a traditional autoencoder, creating a latent vector (represented by the output of the max pool function) which stores the features of any input. The model differs from most existing autoencoders by using Swish activation functions[9] as opposed to Rectified linear units (ReLu) such that gradients can be differentiated continuously at every point. Importantly the network also contains a symmetric function:

$$f(x_1, x_2, x_3, ..., x_n) = f(x_2, x_n, x_3, ..., x_1) = f(x_1, x_3, x_2, ..., x_n) \tag{2}$$

which has been chosen as the Maxpool function based on the work of the authors of PointNet[1] and is demonstrated experimentally by an up-sampling network[23]. The Maxpool function ensures the set is invariant under permutations and therefore only considers the magnitude of given points within the set.

4.3 Neural Network three - Image to Point Cloud Structure

This network was a direct continuation of the autoencoder network in section 4.2. The progression from autoencoder to UNet is displayed by the dotted red line in Figure 8. Above the red line represents a standard autoencoder setup, with a similar function to the final UNet. Below the dotted red line, more layers perform extra feature extraction on the input but also transfer some data to layers further into the structure. The question then arises, "Why not add more layers to a simple autoencoder setup instead of transferring some data prematurely?"

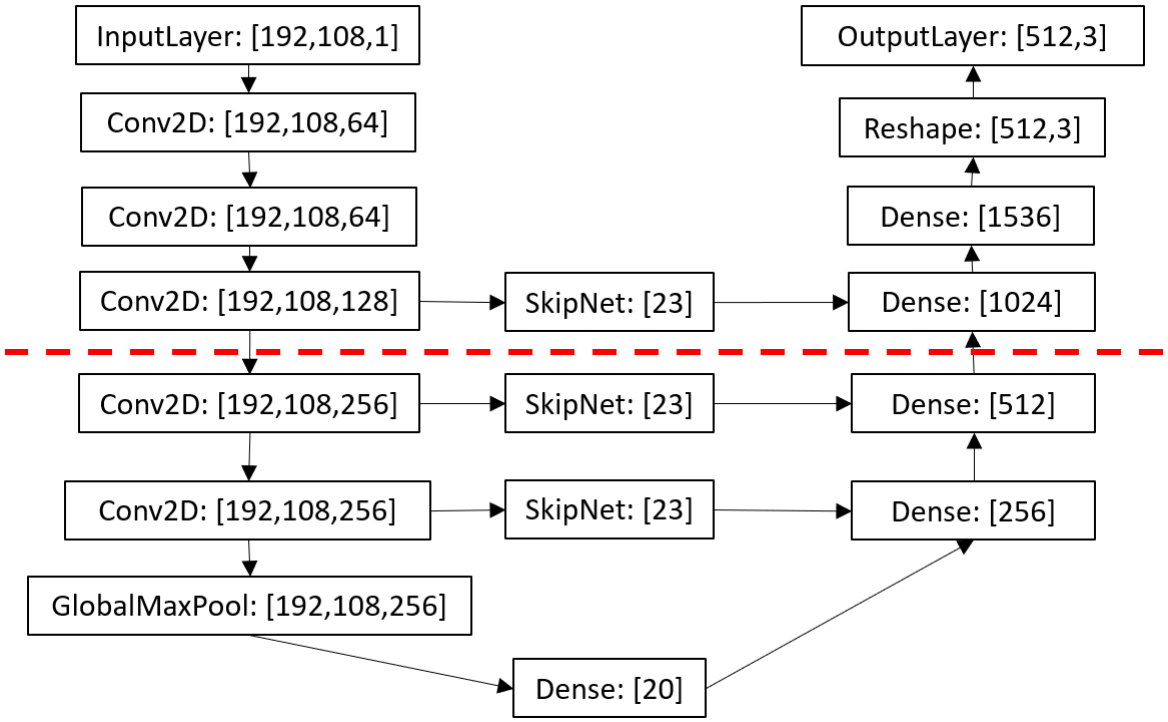

Figure 8: The final UNet structure, resembling the letter U. This structure was chosen for 2D to 3D deep learning because it had the most success at reducing the chamfer loss. Originally, only the upper half of the network was used (as shown by the dotted red line). As the network was improved upon more layers were added and the structure developed into a UNet.

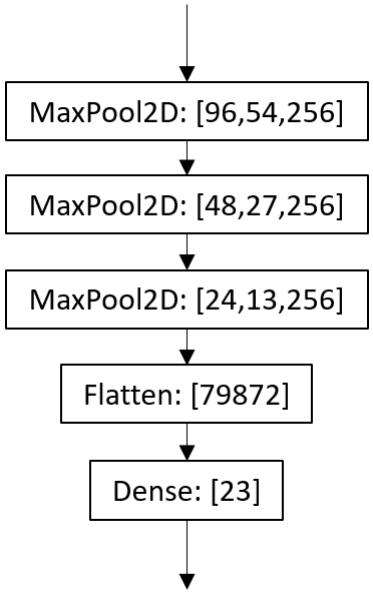

Figure 9: The SkipNet structure. Instead of transferring the data directly between layers in the "U" the SkipNet reduces and processes some of the features first.

To answer this, the UNet structure shown in Figure 8 can be viewed as 4 autoencoders placed in parallel, analogous to a parallel resistor network (where the SkipNet is a resistor). Each "resistor" is then tuned by the network during training, ensuring that important layers impact the final distribution to a larger extent. UNet structures typically show greater performance in image segmentation tasks[24], and successfully segments 3D LIDAR sensor data[25]. This model was chosen despite not being required to perform any segmentation tasks because it still outperformed standard autoencoder losses when experimented with.

The SkipNet structure was necessary to rapidly reduce the size of features being passed through the network. Without the reduction in size any subsequent dense layers would exceed memory capabilities of most modern computers and the large number of parameters would risk over-fitting of the entire network through the first SkipNet - ignoring lower SkipNets and not properly extracting features. The SkipNet then ensures that the data is flattened - a reduction in dimension of rank 1 - and then a single layer of feature extraction is applied before it is concatenated with the network.

4.4 Loss Function

The loss function of a neural network is a way to define how accurately the output predictions match the true values. This is then used to adjust the weights and biases such that the network will perform better in its next pass through. Loss functions can be unique to the problem at hand because comparing two sets of data can be done in many different ways.

The surface penalty is a novel loss function proposed in this project designed specifically to improve the results from the third - 2D to 3D - network.

4.4.1 Loss Function for Network one

Tensorflow's mean squared error loss function was used for training PointNet. The output labels are a 1D array with the largest output being the predicted label, the ground truth is compared against this by taking the average of the square of these differences.

4.4.2 Loss Function for Networks two and three

The loss function used is known as the Chamfer distance. For a given set of points $S_1, S_2 \subseteq \mathbb{R}$

$$d_{CD}(S_1, S_2) = \sum_{x \in S_2} min_{y \in S_2} \mid x - y \mid^2 + \sum_{y \in S_2} min_{x \in S_2} \mid x - y \mid^2 \tag{3}$$

The main benefit of this quantity is that it does not require an ordered set of points. As most point sets are not ordered within a set based on the coordinates of their points, this metric helps to differentiate two point sets based on their spatial variance. For each point in the first set S_1, the algorithm searches for the closest point in the second set S_2. Ideally if the point sets were identical these points would lie on top of each other and there would be no distance between them. If there is a difference between S_1 and S_2 then the chamfer distance will return a value based on how different the two sets are. This value increases if there are a greater number of points that do not overlap and it decreases if there are only a few outliers. The result is a function which works based on spatial dependence rather than position within the file. To explain this loss function more intuitively I have made a short explanatory video: https://www.youtube.com/watch?v=P4IyrsWicfs.

This loss function can be computationally expensive $O(N^2)$ because it must loop through the whole of S_2 for each point in S_1 and then repeat the process for S_2. Luckily this can be run in parallel on a GPU which greatly speeds up results. Alternatively tensorflow's API contains the function einsum which provides another method to calculate the chamfer distance in parallel as seen in the Appendix section 9.2.

4.4.3 Surface Penalty for Network three

In an effort to further reduce the loss of the network a surface penalty was added. This worked on the assumption that all 3D models have surfaces which are reasonably flat and any point which doesn't lie on that surface should be penalised. In practice this was realised by creating a plane from 3 points using the equation for a plane:

$$ax + by + cz + d = 0 \tag{4}$$

The coefficients for this plane is found by first taking the determinant of the 3 points:

$$D = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} \tag{5}$$

And then the coefficients are given by:

$$a = \frac{-d}{D}\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix}, b = \frac{-d}{D}\begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}, c = \frac{-d}{D}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \tag{6}$$

The distance from the next closest point to the plane is then measured. If the distance is very large then the surface is naturally bumpy, contrariwise if the distance is very small then the surface is smooth.

$$S = \frac{\mid ax_4 + by_4 + cz_4 + d \mid}{\sqrt{a^2 + b^2 + c^2}} \tag{7}$$

Every nearest neighbour has a surface penalty taken, returning a sum of all the penalties for a given point cloud. The surface penalty does not compare two point sets but rather gives a single value describing how bumpy or smooth any given object is. In order to turn this into a loss function the overall surface penalty of both the ground truth and the prediction cloud are compared.

5 Experimental methods and results

The results from the three neural networks are presented here. All generated visualisations were rendered in Blender or plotted using the Python library matplotlib. Each blue dot in the Python plots represents a point in the point cloud, some points are obscured by those in front of them.

5.1 Neural Network one - Classification with PointNet

Testing the classification ability of the PointNet variant on a point cloud had 81.8% success rate in labelling shapes from ModelNet 10. This differs from the result stated in the PointNet paper of 89.2%. Training was proportional to the number of categories included, there were 10 different categories in the dataset taking approximately 9 minutes to fully train. A sample of predictions was taken every 10 epochs to manually assess how the network appeared to be performing. An epoch is an arbitrary unit of measure representing the training of a batch of input data. Towards the end of the training a sample has been selected and is shown in Figure 10.

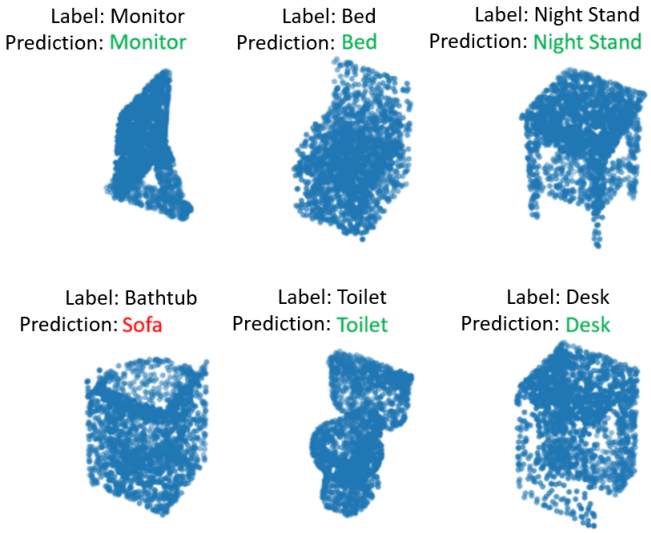

Figure 10: PointNet Results, correct predictions are in green, incorrect are in red. A bathtub is mislabelled as a sofa giving this sample a score of 5/6.

PointNet performed well with 81.8% accuracy on the ModelNet 10 dataset, following this success the network was challenged with an input that had not been used before. Two simplistic chairs of different styles were constructed in Blender using the modelling tools available, the chairs were then converted into point clouds and fed through PointNet. Figure 11 shows how both chairs were labelled as monitors rather than chairs which suggests one of three things: (a) chairs and monitors are geometrically alike, (b) the network is overfitting to the dataset, (c) monitors in the ModelNet 10 dataset contain many flat surfaces whereas ModelNet 10 chairs are more rounded, the chairs created here contain flat surfaces and therefore appear to be monitors.

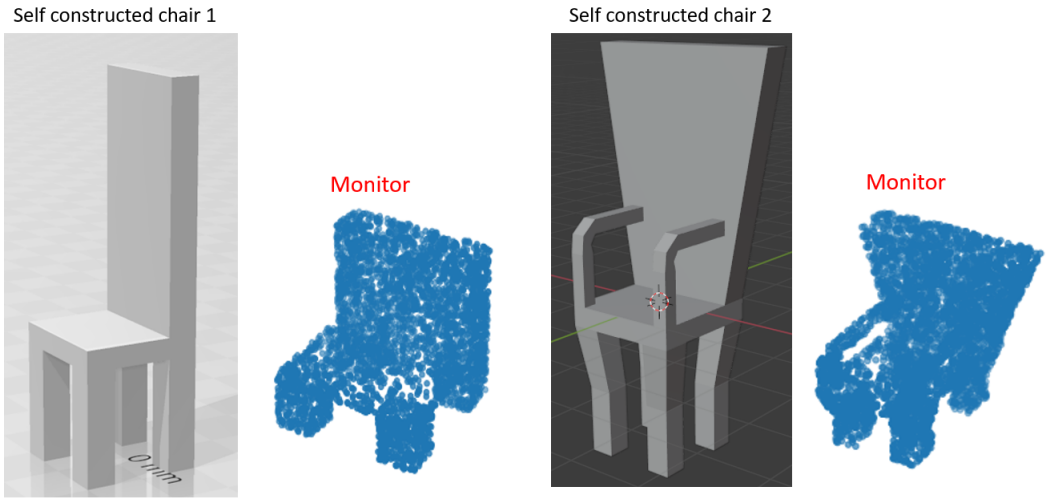

Figure 11: Incorrect labelling of a chair generated as tests

5.2 Neural Network two - AutoEncoder

With a classifying network complete, the first half of the autoencoder has been created. Figure 7 shows how the PointNet structure has had the T-Net's removed to further simplify the structure and speed up computation time. This is also the first experimental use of the chamfer distance where the prediction cloud is invariant under transformation. The results from the autoencoder shown in Figure 12 show 1000 Epochs of training with a batch size of 32 and 1024 points per point cloud. The chamfer loss is 0.007 which gives an intuitive description of how low the chamfer loss must be for reasonable results, this can be used as a goal in the photograph to 3D model.

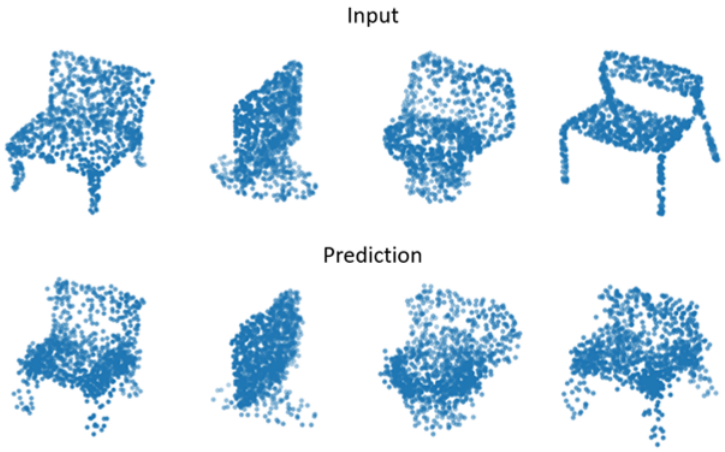

Figure 12: Autoencoder Results. Input point clouds displayed above are replicated by the network and the output is displayed below.

Whereas in network 1 and 3 there is only one point cloud at either end of the network, the reconstructive autoencoder both takes in and outputs a point cloud. Point clouds contain a large amount of data which added strain to the GPU as there were many parameters to compute. The training took over 5 hours for a 1000 epoch run. This was noticeably longer than the two other networks. When training the autoencoder, large batch sizes of 8 or more training examples exceeded the RAM of the GPU (an NVIDIA Quadro RTX 6000). The solution was to use small batch sizes of 2 or 4 training examples per epoch.

5.3 Neural Network three - Image to Point Cloud

The UNet structure discussed in section 4.3 is used as the final network for generating point clouds from intensity images. The weights were set randomly by tensorflow at the beginning of each experiment and no weights were transferred prior to training. As with the Autoencoder results the Swish activation function gave the lowest loss performing significantly better than the ReLu function. The number of nodes - 23 - of the dense layer in each SkipNet was chosen by comparing integer values from 1 to 100 and again choosing the lowest loss. This approach was repeated when choosing the dense layer at the bottom of the UNet with 20 nodes.

Initial experiments were based on the ModelNet10 dataset as discussed in section 3 before the SPRING dataset was used as seen in Figure 13.

Figure 13: Two examples of the meshes that were used as ground truth, rendered in Blender with a single light source. The renders were used in earlier tests of the network before a more challenging dataset was created as discussed in section 3

Computation time scales quadratically with the number of points in the point cloud, so during training either 512 or 1024 points were used to determine the best solutions. Figure 14 shows an example of a more sparsely distributed point cloud containing 1024 points. In this result a single image has been used after training the network for 150 epochs. The chamfer loss converged to 0.006 as seen in Figure 15. In order to obtain the lowest possible loss a three main variables were changed.

Firstly, the number of layers in the model was increased from 3 Conv2D layers to 7 (which meant 1 Dense layer to 12 respectively to ensure the UNet structure stayed consistent). Results consistently improved as more layers were added but computation time increased rapidly too, a compromise of 5 Conv2D layers with 8 Dense layers gave results in under 20 minutes with a chamfer loss as low as 0.004.

Secondly, the filter size of the dense layers was experimented with so that the middle layer (and the SkipNet dense layers) formed a bottle neck on the amount of information able to pass through them. The aim was to constrain the network as much as possible before the loss started to decrease, this would then provide parameters that could be tweaked manually to change the outcome. The network was able to reduce its dimension to 89 parameters (3x23 in the SkipNet and 1x20 for the central dense layer) meaning all the information required to create a 512 point 3D model can be defined by 89 parameters.

Finally the Adam optimiser's learning rate was varied between 0.1 and 0.00001 with non-uniform step sizes that reduced in size as the value decreased. An optimal learning rate of 0.0001 was chosen to best reproduce the ground truth.

Figure 14: Sparse point cloud results - only 512 points. On the left is the output of the network which can be compared to the expected distribution on the right. A rotating 3D representation can be accessed in section 9.1.

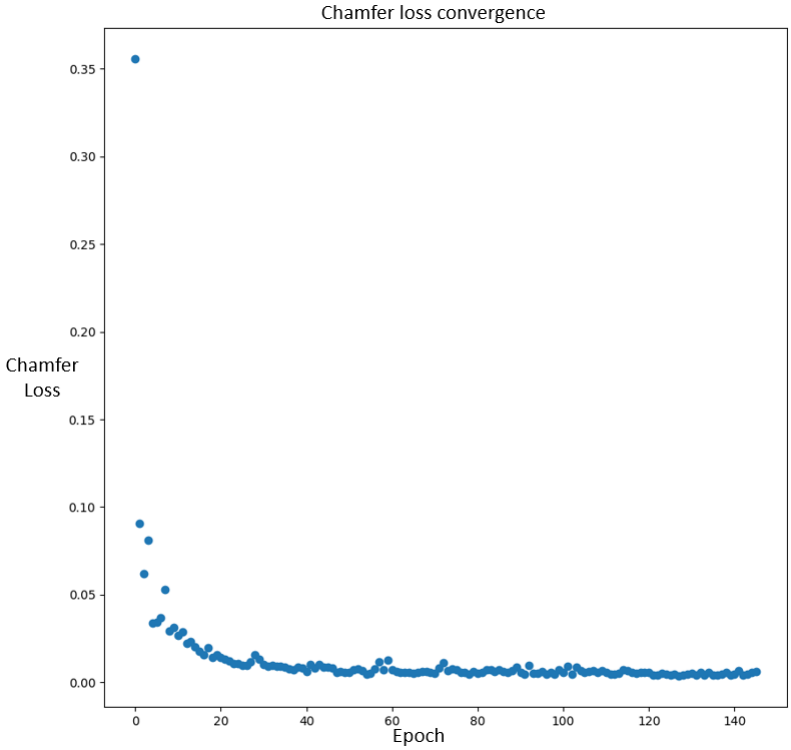

Figure 15: The initial training converges at a very fast rate, the convergence is smooth thanks to the Swish activation function.

The second experiment was repeated for a 2056-point point clouds with a total of 104 parameters (3x27 in the SkipNet and 1x23 for the central dense layer) being required before the loss function became overly constrained. The number of layers and the learning rate were not changed due to the increased training time required for such a large number of points.

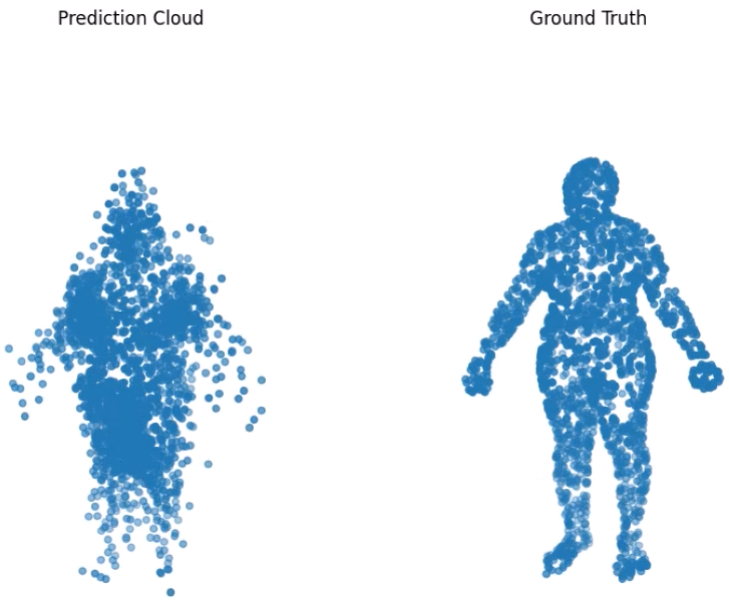

Figure 16: Dense point cloud results - 2056 points within the cloud. On the left the result shows how the arms and other finer details are missing crucial parts or not present at all. A rotating 3D representation can be accessed in section 9.1.

Training the 512-point network took 5 seconds per epoch and fully converged after around 150 epochs which meant 15 minutes per training run. The 2056-point network took 30 seconds per epoch and converged at the same rate taking nearly an hour and a half to train. For

this reason the majority of the experiments were completed on the 512-point network and the 2056-point was only used as a guide on occasions.

5.4 Applying surface penalty

The surface penalty works on the assumption that 3D models contain surfaces which are reasonably flat. An object such as a chair with many flat surfaces should have a low surface penalty compared to random noise. When applying the surface penalty to a variety of different SPRING models and then comparing to random noise:

Point Cloud	Surface Penalty				Mean
Person	0.2534	0.2256	0.2012	0.2087	**0.2028**
Random Noise	0.4439	0.4580	0.5068	0.4621	**0.4428**

Table 1: The surface penalty is able to clearly distinguish random Gaussian noise from a 3D point cloud of a person. A person is made up of more flat surfaces and therefore has a lower surface penalty. 4 example values are shown here on 4 separate 3D models.

It is clear that the surface penalty of a person is lower than that of random noise as expected. This indicates that adding the surface penalty as another constraint is a reasonable assumption, the penalty is able to distinguish between models and should be able to benefit the loss function.

Figure 17: Sparse point cloud results with the surface penalty applied. The prediction cloud on the left shows no resemblance to a human figure.

The result of applying the penalty to the model is shown in Figure 17. This is an example of adding a constraint which is too strong, the loss function acts similar to a gravitational force on a large body by pulling the points together into a ovoid with a smooth surface.

6 Discussion of Results

The replica PointNet performing with 81.8% accuracy despite being a smaller network than the one described in the PointNet paper was surprising. It is unclear exactly how a reduction in the number of parameters within the network changes the final accuracy. In other networks such as the UNet reducing the number of parameters during testing significantly increased the accuracy of the final results.

The reconstructive autoencoder appeared to be generating "noisy" versions of the input data. This was a bit unexpected as a neural network usually selects a particular shape or distribution and over-fits to it. A possible explanation is that the autoencoder was not fully trained and had not converged. Another explanation could be that the apparent "noise" was in fact an average distribution between two categories. For example the average between a chair and a monitor may not look like any recognisable 3D object. If an input chair had been mislabelled as half chair/half monitor then this distribution could be generated and look like a "noisy" chair.

In many ways the UNet performed better than expected, vague shapes such as limbs and a head are clearly visible in the results. Moreover areas behind the person have been filled in despite being hidden from view in the pictures which suggests that the network has learnt what these areas are expected to look like. It is unclear if the network has learnt how to interpret 3D space or if there has been overfitting during training. Human point cloud outputs did appear different depending on the input intensity images which suggests that the model is not severely overfitting and has some 3D awareness.

When the density of points is increased there is a clear problem with how uniform the distribution of points are in the prediction cloud. The points are generally clumped together at the waist and around the shoulders, presumably this is due to the majority of point clouds in the dataset containing points at these locations. Again this could be an overfitting problem unique to these areas. The point cloud as a whole may not be overfitting but the points within these specific areas may have learnt a distribution that is repeated regardless of the input intensity image.

Using the surface penalty was not effective even though it performed as expected during the testing stage of the experiments. The issue could be that the solution to a surface penalty of a person is not unique. If this is not a unique solution then there will be no benefit to recreating the desired distribution, the prediction distribution is likely to just look like random noise that has been smoothed out. The surface penalty may also be acting as a repulsive force which helps the distribution become more uniform: if the ground truth has a particularly high surface penalty then points will learn to be pushed away from each other so that they don't lie on the same plane. This could be why the ovoid cloud seen in Figure 17 does not have any excessive clumping of points unlike the cloud seen in Figure 16.

7 Conclusion

In this project three algorithms capable of processing 3D point clouds for different tasks were implemented.

(1) **A Point cloud classification network was capable of sorting 3D objects.** The results can be used in conjunction with or as an alternative to current image recognition techniques, removing the need to render a 3D scene before labelling it.

(2) **Point cloud autoencoders using the Chamfer distance as a loss function were able to generate reasonably accurate reconstructions of the initial point cloud**, this shows potential for compressing large data files into feature maps allowing more objects to be placed in a scene without excessive memory usage.

(3) **Three dimensional imaging from photographs has been demonstrated using a novel dataset and network structure.** A visual understanding of 3D space has been extracted from a combination of non-linear functions, this is knowledge that two decades ago computers would have struggled to extract from a scene at all. A UNet is chosen as the preferred structure with the Chamfer distance acting as a loss function. A novel surface penalty is proposed as an additional constraint. The surface penalty is rigorously tested and experimental evidence proves that it should not be included as a constraint. Experiments on the dimension of the latent space reveal that fewer than 89 parameters for a 512-point point cloud create an overly constrained network.

8 Further Work

8.1 Multi-image networks and semantic scene segmentation

Generating information about hidden, masked or otherwise obscured areas of the image relies on the network's current dataset of where points should be placed. Given a set of more than one input image the network would theoretically rely less heavily on its own intuition and be able to pull 3D features from the set of images by performing comparisons. In practice this has been problematic due to large memory consumption from the increased number of parameters. A potential future addition to the project may include using smaller input images from multiple directions as a substitute for the higher resolution, single direction that is currently used.

Individual objects and human models were used as part of the dataset, with the background changing colour but never texture or appearance. In order to fully recreate a lifelike scene a more diverse dataset is necessary. Potential for recreation of whole scenes would mean a photo of any scene could become a 3D environment without the need to mask out any background. PointNet has demonstrated segmentation of 3D scenes implying that if a point cloud scene is generated from an intensity image then classification of that scene could be achieved. Image to point cloud followed by PointNet classification may create a more powerful classifier for scene based images compared to solely intensity image classification.

8.2 Alternative Loss functions

Many state-of-the-art intensity image to three dimensional object networks use the Wasserstein Metric in order to achieve lower loss and higher accuracy. This metric (the Earth Movers distance) has also been used in GANs and achieves high fidelity when training the discriminator. Replacing the Chamfer distance with this metric as an alternative loss function could significantly improve results.

Repulsion Loss is proposed for point cloud upsampling [23] based on the sampling techniques used in constructing a point cloud from a mesh. Repulsion loss works by penalising any points which are located too close to each other within a point cloud, hence achieving a more uniform distribution in the end result. The repulsion decays rapidly as distance from the point increases. Rapid decay is necessary to retain the model's shape as it would be counterproductive to push all points to a position where they are exactly equal distances from each other. By including this in the current work the issue of non-uniform point distribution seen in Figure 16 could be solved.

9 Appendix

Glossary

Adam optimiser An optimisation algorithm to manage gradient descent in a network. 19

Blender An open source 3D rendering software. 5, 9, 11, 12, 16–18

epochs An arbitrary unit of measure representing the training of one single batch of data. 10, 16

GAN A generative adversarial network. These networks contain a generator which produces predictions and a discriminator which rates the generated predictions based on how closely they resemble the ground truth. 8, 10, 23, 29

ground truth The data provided before any augmentation or algorithms are applied to it. 5, 7–9, 15, 16, 18, 19, 22

learning rate The step size taken in a gradient descent algorithm. 19, 20

LIDAR Light Detection and Ranging, used for gathering depth information from a scene. 5

mesh A collection of edges, vertices and faces made up of polygons defining a surface. 7, 9, 11, 23

MNIST Modified National Institute of Standards and Technology database. A standard dataset used to test many machine learning applications. 12, 29

SPAD Single-Photon avalanche diode, often arranged as an array to form a SPAD array. 5

T-Net A transformation network attempting to recreate an affine transformation. 13, 17

UNet A neural network shaped like a U. These networks are often used for image segmentation. 5, 8, 14, 15, 18, 19, 21, 22, 25

voxel A value on a 3D grid, acting like a pixel in a 2D image. 9–11

9.1 More Results

For more results follow this link: https://github.com/leh115/Image-to-3D-Project.git.

9.2 Pseudo-code

This code has been written in pseudo-python based on the Keras API. The full final network is displayed here, illustrating how a UNet may appear when written down. The chamfer distance is also included to give a rough description of how it may be implemented practically.

```
    def chamferDistance(PointSetA,PointSetB):
        difference = (tf.expanddims(PointSetA, axis=-2) - tf.expanddims(PointSetB, axis=-3)
)
        squareDistances = tf.einsum("...i,...i-...", difference, difference)
        minSquareAtoB = tf.reduceMin(squareDistances, axis=-1)
        minSquareBtoA = tf.reduceMin(squareDistances, axis=-2)
        return tf.reduceMean(minSquareAtoB) + tf.reduceMean(minSquareBtoA)
    def skipNet(net):
        skipnet = layers.MaxPool2D()(net)
        skipnet = layers.MaxPool2D()(skipnet)
        skipnet = layers.MaxPool2D()(skipnet)
        skipnet = layers.Flatten()(skipnet)
        return layers.dense(23)(skipnet)
    def buildModel():
        skips = []
        inputs = keras.Input(shape=(imgRows,imgCols,channels))
        net = conv2D(inputs,64)
        net = conv2D(net,64)
        net = conv2D(net,128)
        skipnet = skipNet(net)
        skips.append(skipnet)
        net = conv2D(net,256)
        skipnet = skipNet(net)
        skips.append(skipnet)
        net = conv2D(net,256)
        skipnet = skipNet(net)
        skips.append(skipnet)
        pool = layers.GlobalMaxPooling2D()(net)
        net = dense(pool,20)
        net = tf.concat([net, skips.pop()],axis = 1)
        net = dense(pool,256)
        net = tf.concat([net, skips.pop()],axis = 1)
        net = dense(pool,512)
        net = tf.concat([net, skips.pop()],axis = 1)
        net = dense(pool,1024)
        net = dense(pool,1536)
        outputs = layers.Reshape((NumberOfPoints, 3))(net)
        model = keras.models.Model(inputs=inputs, outputs=outputs, name="Model")
        return model
    def importDataset(batchSize):
        for file in range(batchSize):
          pictures.append(load(pictureFile))
          pointClouds.append(sample(load(ObjectMesh))) return pictures,pointClouds
    def train():
        for epoch in range(epochNum):
          pictures,points = importDataset(batchSize)
```

model.trainOnBatch(pictures,points)

9.3 Gantt Chart

Semester 1

Week Number	3	4	5	6	7	8	9	10	11	12
Literature review			2 ■							
Risk assessment										
Working Environment		1 ■				5 ■				
Build/Find Data Set			3 ■							
Python Coding				4 ■					6 ■	
Code Optimisation										7 ■

■ - Milestones:

1 – Decide on best working environment e.g. Google Colab/ University Computers/ a mixture

2 –The core papers on computer vision/machine learning have been read and notes made

3 – Comfortable with 3D object files, point clouds and working in a 3D environment

4 – A working, very basic network that can export 3D obj. files or point clouds

5 – Re-assessment of working environment, question if there are any better solutions

6 – The main part of the code should be working; decisions can be made on what to look at in semester 2

7 – Without adding any more features, the code should be optimised in its final state

Semester 2

Week Number	3	4	5	6	7	8	9	10	11	12
Literature review						3 ■				
Python Coding					1 ■					
Code Optimisation						2 ■				
Write Report						4 ■	5 ■			
Write/Present talk									6 ■	
Make/Present Poster										7 ■

■ - Milestones:

1 – Code should be complete/ to a point where I can finish

2 – Fully comment finished code and remove unnecessary lines/functions

3 – Should finish with a much deeper understanding of computer vision and machine learning both from practical work and literature review

4 – The report should be finished - need to discuss with supervisor

5 – Submit report

6 – Present the talk

7 – Present the poster

9.4 Risk Assessment

Risk Assessment

(This is for example/training use only and is NOT a formal risk assessment document)

Activity title :	**Three-dimensional imaging from photographs**
Location :	**Home**
Assessor :	
Date Assessed :	

Risk Analysis Matrix Level of Risk					Likelihood	Severity
Likelihood	4	8	12	16	1. Unlikely	1. Insignificant/No Injury
	3	6	9	12	2. Possible	2. Minor Injury
	2	4	6	8	3. Likely	3. Moderate Injury
	1	2	3	4	4. Certain	4. Major Injury/Fatality
×		1	2	3	4	
		Severity			*Score likelihood*	*Score severity*

Hazard	Persons Affected	**Likelihood** of incident	Potential **Severity**	**Risk control already in place**	**RISK ASSESSMENT**	**Further Action required to control risk**
Electrical wires could be tripped over	People near computer	3	4	No wires are on floor, all managed safely and out of sight.	Minimal Risk with control in place	Check daily that wires are in place
Eye strain from prolonged computer use	Computer user	5	4	Intermittent breaks after every 20 minutes of screen time. Screen overlooks garden giving a range of depths to focus on.	High potential risk requires constant monitoring	Ensure risk is not overlooked with reminders
Back strain from sitting for too long	Computer user	4	4	Intermittent breaks and regular exercise can reduce strain.	High potential risk over longer timeframes. No definite prevention	Reminders to take breaks is again important.
Data stolen	Researcher/Data owner	2	1	Data is not stored on cloud computing; passwords are set up for computer.	Low risk with low consequences	None, unless research project advances rapidly to cutting-edge models.

Data corrupted	Researcher/Data owner	3	2	Warning - No back-up in place currently. No risk control in place	Requires action when coding section begins	Create a copy of project files on a separate hard drive/SSD
Heating in winter months	Everybody in UK	5	5	Radiators on timer	Potential High risk, but low risk with measures in place	If timed radiators are not enough use electric heater

9.5 Summary of Summer Project

The goal of the project was to create a working Generative Adversarial Network (GAN) capable of removing fog or mist from the foreground of a picture. A secondary goal was to develop personal confidence in writing and training machine learning models.

By studying and modifying known autoencoder networks I learnt how to create a working machine learning model which could partially remove fog from an image. To train the model a set of randomly generated images were created with Unreal Engine software to provide examples of pre/post-fog images.

With this training in machine learning I went on to create a working model based on a GAN structure. This model was based off previous examples[https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-an-mnist-handwritten-digits-from-scratch-in-keras/] and could re-create MNIST-style images of numbers using a convolutional generator and discriminator.

Figure 18: Example of generated MNIST numbers using GAN technique. The numbers should look as if they have been handwritten despite being created entirely by the computer.

Then an attempt at creating an autoencoder style GAN proved too complex so to improve my skill at writing GAN models I made a cat generator, capable of producing lifelike images of cats from a dataset.

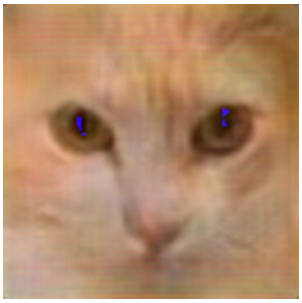

Figure 19: An example of a cat generated using the GAN

As the goal of this project was to see how well a computer could see through fog with appropriate machine learning techniques. The highest quality image produced was using an autoencoder, not a GAN although I believe this can be changed if I had more experience.

Figure 20: Using an autoencoder to defog an image

References

1. Qi, C. R., Su, H., Mo, K. & Guibas, L. J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. arXiv: 1612.00593 [cs.CV] (2017).
2. Baltsavias, E. P. A comparison between photogrammetry and laser scanning. *ISPRS Journal of Photogrammetry and Remote Sensing* **54,** 83–94. ISSN: 0924-2716. http://www.sciencedirect.com/science/article/pii/S0924271699000143 (1999).
3. Remondino, F. & El-Hakim, S. Image-based 3D Modelling: A Review. *The Photogrammetric Record* **21,** 269–291. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1477-9730.2006.00383.x. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1477-9730.2006.00383.x (2006).
4. Brunn, A. & Weidner, U. Hierarchical Bayesian nets for building extraction using dense digital surface models. *ISPRS journal of photogrammetry and remote sensing* **53,** 296–307 (1998).
5. vuplicrts. *Point Cloud* https://skfb.ly/6VDBY. (accessed: 16.10.2020).
6. Zhao, X., Sun, P., Xu, Z., Min, H. & Yu, H. Fusion of 3D LIDAR and Camera Data for Object Detection in Autonomous Vehicle Applications. *IEEE Sensors Journal* **20,** 4901–4913 (2020).
7. Fabio, R. *et al.* From point cloud to surface: the modeling and visualization problem. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* **34,** W10 (2003).
8. Nouveaux memoires de l'Academie royale des sciences et belles-lettres de bruxelles (1845).
9. Ramachandran, P., Zoph, B. & Le, Q. V. Searching for Activation Functions. arXiv: 1710.05941 [cs.NE] (2017).
10. Schroff, F., Kalenichenko, D. & Philbin, J. FaceNet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. http://dx.doi.org/10.1109/CVPR.2015.7298682 (June 2015).
11. Schwarz, K., Liao, Y., Niemeyer, M. & Geiger, A. GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis. *CoRR* **abs/2007.02442.** arXiv: 2007.02442. https://arxiv.org/abs/2007.02442 (2020).

12. Henzler, P., Mitra, N. J. & Ritschel, T. Escaping Plato's Cave using Adversarial Training: 3D Shape From Unstructured 2D Image Collections. *CoRR* **abs/1811.11606.** arXiv: 1811.11606. http://arxiv.org/abs/1811.11606 (2018).

13. Nguyen-Phuoc, T., Li, C., Theis, L., Richardt, C. & Yang, Y.-L. HoloGAN: Unsupervised learning of 3D representations from natural images. *CoRR* **abs/1904.01326.** arXiv: 1904.01326. http://arxiv.org/abs/1904.01326 (2019).

14. Fan, H., Su, H. & Guibas, L. J. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. *CoRR* **abs/1612.00603.** arXiv: 1612.00603. http://arxiv.org/abs/1612.00603 (2016).

15. Arjovsky, M., Chintala, S. & Bottou, L. Wasserstein GAN. arXiv: 1701.07875 [stat.ML] (2017).

16. Groueix, T., Fisher, M., Kim, V. G., Russell, B. C. & Aubry, M. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. *CoRR* **abs/1802.05384.** arXiv: 1802.05384. http://arxiv.org/abs/1802.05384 (2018).

17. Wu, Z., Song, S., Khosla, A., Tang, X. & Xiao, J. 3D ShapeNets for 2.5D Object Recognition and Next-Best-View Prediction. *CoRR* **abs/1406.5670.** arXiv: 1406.5670. http://arxiv.org/abs/1406.5670 (2014).

18. Yang, Y. *et al. Semantic Parametric Reshaping of Human Body Models* in *2014 2nd International Conference on 3D Vision* **2** (2014), 41–48.

19. mikedh. *Trimesh* https://github.com/mikedh/trimesh. (accessed: 13.10.2020).

20. Alextsui05 xuser86, x. *Blender OFF Addon* https://github.com/alextsui05/blender-off-addon. (accessed: 29.09.2020).

21. charlesq34. *PointNet* https://github.com/charlesq34/pointnet. (accessed: 14.10.2020).

22. charlesq34. *PointNet Autoencoder* https://github.com/charlesq34/pointnet-autoencoder. (accessed: 09.11.2020).

23. Yu, L., Li, X., Fu, C.-W., Cohen-Or, D. & Heng, P.-A. PU-Net: Point Cloud Upsampling Network. arXiv: 1801.06761 [cs.CV] (2018).

24. Ronneberger, O., Fischer, P. & Brox, T. *U-Net: Convolutional Networks for Biomedical Image Segmentation* in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (eds Navab, N., Hornegger, J., Wells, W. M. & Frangi, A. F.) (Springer International Publishing, Cham, 2015), 234–241. ISBN: 978-3-319-24574-4.

25. Biasutti, P., Bugeau, A., Aujol, J.-F. & Brédif, M. RIU-Net: Embarrassingly simple semantic segmentation of 3D LiDAR point cloud. *CoRR* **abs/1905.08748.** arXiv: 1905.08748. http://arxiv.org/abs/1905.08748 (2019).