Innovation Center for Education

PHEMESOFT
Software That Matters
TRUSTED DELIVERY PARTNER for IBM ICE

University Of Petroleum and Energy Studies Dehradun

Final Project Report

on

Deep learning algorithm to generate realistic 3D models of objects from 2D images.

<u>Team members</u>:

ASHUTOSH SINGH KUSHWAH- R214220293

ASHNEET KAUR KOCHHAR-R214220289

ASHUTOSH TRIPATHI - R214220291

ASMITA PALAK LAL- R2142201720

ATEEV BAHUHKHANDI - R214220297

Industry Mentor:

SUMIT SHUKLA

Table of Contents

1. Background

A big difference between human beings and computers is that humans have developed some fundamental understanding of the world. Due to this understanding, it is easy to transfer a complex structure through a brief explanation. For example, telling a person to “think of a pink horse, with wings”, the person in question would have a pretty clear image of the creature you are talking about, without having seen it before. Generative artificial intelligence (AI) is a vastly growing area of research, and it is believed that Generative AI will enable computers to gain an understanding of the world [12]. This would enable computers to complete complex tasks, without someone controlling its every move. It would make artistic work much easier, a music producer could simply tell the computer to “increase the beat” or “add more drums”. A graphical designer could specify features (“cute dog”, “bigger head”, “more red color”) instead of spending a lot of man-hours on drawing. This could have a huge impact on the game creation industry. Machine learning (ML) is a subsection of artificial intelligence, it is used to enable systems to automatically learn from experience instead of being programmed exactly how to perform a task [8]. Meaning that it can be used to solve problems without the need to specify how to solve the problem, this is what makes ML incredibly powerful. This thesis will look into feature extraction from 2D images and reconstruction of 3D objects, using machine learning.

1.1 Aim

The following objectives will need to be fulfilled:

1. Decide how the machine learning model to extract features from a 2D image should be constructed.
2. Decide upon how the generative 3D machine learning model should be constructed.
3. Create and link the two machine learning models.
4. Tweak hyperparameters to give the best result
5. Draw a conclusion from the final model

When this project is complete, the intention is to enable a 2D-to-3D reconstruction of an object, from a mobile camera photo to a 3D object. The object should be able to be imported into a game engine and it will be limited to only reconstruct chairs.

1.2 Technologies

The project discussed above involves the implementation of a complex neural network-based system for tasks related to data analysis and 3D object generation. To achieve its goals, this project leverages a range of cutting-edge technologies and tools.

- Machine Learning Frameworks: Central to the project is the use of machine learning and deep learning frameworks. TensorFlow, in particular, stands out as a fundamental technology. TensorFlow, an open-source machine learning library developed by Google, provides a powerful platform for building and training neural networks. Its flexibility and scalability make it a popular choice for projects involving complex models like convolutional neural networks (CNNs) used for image analysis and generation.

- Python: Python serves as the primary programming language for this project. Python's versatility and extensive libraries for scientific computing and data analysis make it a natural choice for implementing machine learning algorithms and data processing tasks.

- Git and Version Control: For efficient collaboration and code management, Git, a distributed version control system, is used. Git enables the tracking of changes in code, datasets, and documentation throughout the project's lifecycle. This facilitates collaboration among team members and offers the ability to roll back to previous versions if necessary.

- Data Management and Storage: Given the significant data involved in the project, robust data management and storage technologies are essential. Secure repositories, databases, and structured data archiving mechanisms are likely employed to store and organize datasets. Additionally, data integration tools are used to align and format datasets from different sources for compatibility with the neural network and associated algorithms.

- Performance Optimization Tools: To ensure efficient resource utilization and minimize latency, performance optimization tools may be used. These tools could include profiling tools for identifying bottlenecks in code and caching mechanisms for enhancing data retrieval speed.

- GPU Acceleration: Given the compute-intensive nature of deep learning tasks, the project might take advantage of GPU (Graphics Processing Unit) acceleration. Technologies and libraries that enable GPU support, such as CUDA for NVIDIA GPUs, could be integrated into the project to expedite neural network training and inference.

1.3 Hardware Architecture

The architecture of this project is designed to encompass a comprehensive and versatile framework for data management, analysis, and neural network integration. It's crucial to provide an overview of this architecture in detail to understand how all the components work together harmoniously.

At its core, the architecture consists of three main layers: data management, neural network integration, and user interface, each playing a unique role in the system's functionality.

1. Data Management Layer: This foundational layer is responsible for handling all aspects related to data. It starts with data ingestion, where diverse datasets from multiple sources are collected and brought into the system. These datasets can vary in formats, sizes, and structures, making data normalization and preprocessing a vital component. This layer employs data migration strategies to align and format data, ensuring that it's compatible with the neural network models and analysis algorithms. Furthermore, it encompasses data archiving capabilities, ensuring that outdated or less relevant datasets are systematically stored and documented for future reference. Version control mechanisms, powered by Git, are employed throughout the data lifecycle to track changes, ensuring traceability and the ability to revert to previous states if needed. This layer forms the backbone of the entire project, as the quality and accessibility of data are fundamental to subsequent analysis and neural network operations.

2. Neural Network Integration Layer: Sitting on top of the data management layer, this segment is dedicated to the integration of neural networks. It involves the implementation, training, and fine-tuning of various neural network models tailored to specific tasks, such as image analysis and 3D object generation. One of the project's highlights is the utilization of an empirical binary search tree structure for hyperparameter tuning, allowing for efficient optimization while minimizing the number of tests. The integration layer also handles the experimentation process, systematically testing different configurations for neural networks, including dropout rates, activation functions, loss functions, and optimizers. The architecture ensures the seamless flow of data from the management layer to the neural networks and back, enabling iterative improvements and training.

3. User Interface Layer: The user interface layer provides a user-friendly gateway to interact with the system. It offers secure user authentication to ensure authorized access. The interface is designed to be intuitive, allowing users to employ data analysis tools for visualization and exploration of datasets. Users can monitor the training and performance of neural networks, visualize results, and interact with 3D object generation outputs through charts, graphs, and interactive interfaces. This layer also encompasses performance-related aspects, as users may have expectations regarding response times for data processing and neural network training.

The architecture signifies the orchestration of these layers, working in unison to create a robust and efficient system for data-driven analysis and neural network experimentation. By balancing data management, neural network integration, and user interface design, this architecture aims to facilitate seamless transitions between datasets, foster collaboration in development, and ensure the project's objectives are met effectively and securely.

2. System

2.1 Requirements

2.1.1 Functional requirements

Functional Requirements are at the core of the project's capabilities. They include the need for a robust data analysis module capable of efficiently processing and analyzing large datasets. This module should support various data manipulation and transformation operations, ensuring that the data is prepared for further analysis and integration into neural networks. Additionally, the project necessitates the integration of neural networks for specific tasks, such as image analysis and 3D object generation. This involves not only the implementation of these networks but also the ability to train, fine-tune, and deploy them effectively.

Version control is another crucial functional requirement. Throughout the project's lifecycle, a version control system like Git will be employed to meticulously track changes in code, datasets, and documentation. This facilitates collaborative development, ensures traceability, and allows for easy roll-back to previous versions if needed. Furthermore, data integration capabilities are essential to align and format datasets from diverse sources and formats. This ensures compatibility with the neural networks and associated algorithms, promoting the seamless use of data from various origins.

The final functional requirement is data archiving. As datasets become obsolete or are no longer needed for active development, they must be archived systematically. This process involves compressing and securely storing data in repositories, accompanied by comprehensive documentation and metadata management for future reference.

2.1.2 User Requirements

User Requirements are equally significant. Users of the system expect secure and user-friendly interactions. User authentication is vital to ensure that only authorized individuals can access specific system functionalities. The interface should be intuitive and user-friendly, enabling users to easily interact with data analysis tools and neural network functionalities. Additionally, users should have the capability to visualize data analysis results and 3D object generation outputs through charts, graphs, and interactive interfaces. They may also have performance expectations related to response times for data processing and neural network training.

2.1.3 Environmental Requirements

Environmental Requirements encompass the infrastructure needed to support the system. This includes hardware resources like GPUs for accelerated neural network training and high-capacity storage for managing extensive datasets. The system's software dependencies, such as machine learning frameworks and version control systems, must be compatible and well-integrated. Network infrastructure considerations are crucial to ensure seamless functioning within the existing environment, accounting for factors like network latency and bandwidth, especially when dealing with remote data sources. Last but not least, data privacy and security are paramount environmental requirements. Robust measures must be in place to protect sensitive data and enforce access controls.

These system requirements collectively form the backbone of the project, outlining what the system should do, how users will interact with it, and the environmental factors that will impact its performance and security. Adhering to these requirements is fundamental to achieving the project's objectives while ensuring its effectiveness and safety in operation.

2.2 Design and Architecture

The basic architecture consists of two convolutional neural networks (CNN), one called ’Convolutional Autoencoder’ and the second called ’Convolutional 3D Decoder’ see architecture in Figure 4.1 below. The task of the first CNN is to extract features from an image, this feature is then fed as input to the second CNN. The second CNN task is to generate an octree-structured voxel space from the latent space. This process is repeated to increase the resolution of the voxel-space, see the right iterative side of Figure 1.1. Due to the 3D object generator being harder to generalize, it will be the feature extraction job to make the latent space similar for the same object, independent of the angle of the object in the input image.

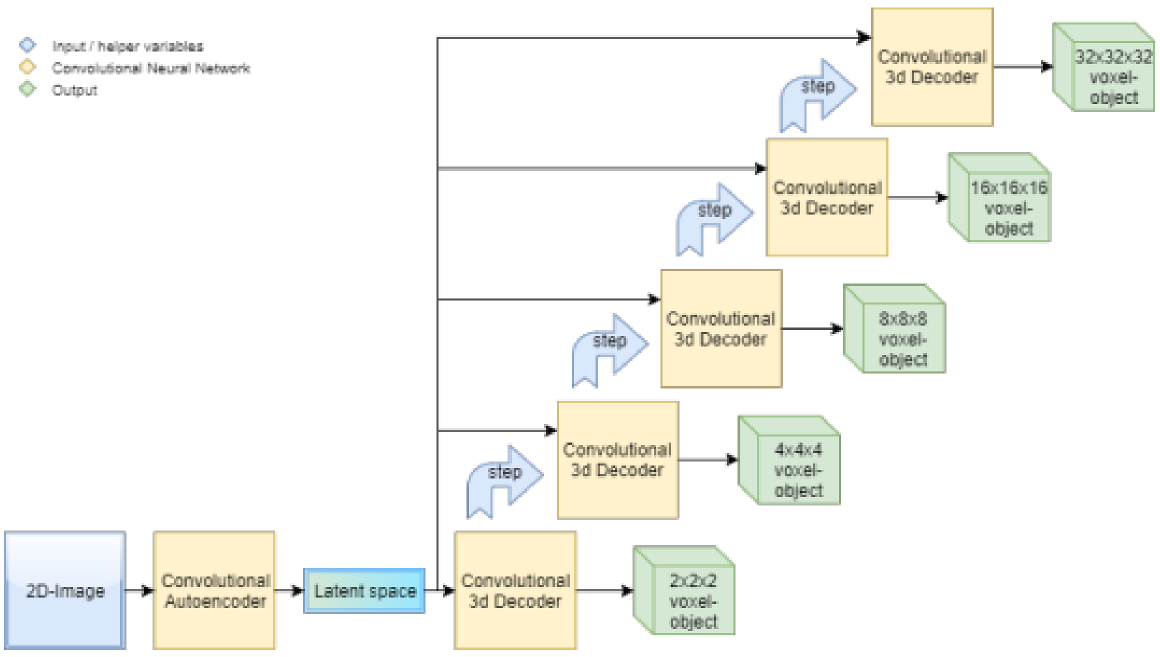


Fig 1.1 Architectural structure of neural networks used during generation

2.2.1 Convolutional Feature Extraction Architecture

Architectural structure of neural networks used during generation. The feature extraction is heavily based on the same model as in Jiajun Wu's paper in 2016 [23]. One key difference is in the data. To make it more generalized in recognizing objects from different angles, all images of the same object are reconstructed to the same output. This will also make it easier for the 3d generative NN since the angle of the input is irrelevant for a 3D object.

2.2.2 Convolutional 3D Generator

Generating 3D objects is not a trivial task, thus two different approaches were taken into consideration. Conditional GAN was the first approach, due to it being a hyped area in content generation and is proved to give an OK result [23]. The second approach is a conditional autoencoder that uses an octree structure. Early in the development of the Conditional GAN, it was shown that it had an exponential growth in memory usage while increasing the resolution. This is not a stable solution, thus approach Two is focused.

2.2.3 Approach One: Conditional Generative Adversarial Network

The idea of the conditional GAN is to create two competing neural networks. One of the networks tries to correctly classify which images come from the actual dataset, whilst the other neural network tries to fool the classifier that its generated content is from that same dataset. The purpose of doing this is to improve the quality of the generated objects without doing a direct comparison against an answer, making the NN more generalized. Some assumptions were made to make the training fairer whilst building the networks were:

Only training Generator while Discriminator accuracy is above 20%.

† Only train Discriminator while discriminator accuracy is below 80%.

† The Discriminator will get a head start of 10 training sessions.

The first two bullets make the network improve alongside each other, instead of one being the superior which will make it harder for the other neural network to improve. The 3rd bullet is so that the Discriminator will have a better starting point, It will also improve the Generator start since it is better to compete against a slightly better component**.**

2..4 Approach Two: Conditional Autoencoder

The idea of this conditional autoencoder structure is to make the NN generate an object in steps. The foundation is inspired by a sculpturing process, which in this case starts of with a single voxel that is split into smaller pieces that are being validated to remove unnecessary voxel/space, this results in enhancing the resolution (See Figure 1.2. Each step is trained individually, which will enable the NN to be able to train on creating as high resolution as possible.

The first step will generate 2x2x2 (8 voxels), for example, this step could find the proportions of the object. An object that is twice as high as its width would only need to use 4 voxels (one side of the cube).

To understand the power behind this, let us say that we would like to create an object with the quality of 64x64x64 (x,y,z) voxels. In this case, the first step would discard half of the voxels (64*64*64 /2 = 131072 voxels/permutations), which is a lot. Figure 4.3 shows how each iteration of the generation could improve the quality of a generated chair.

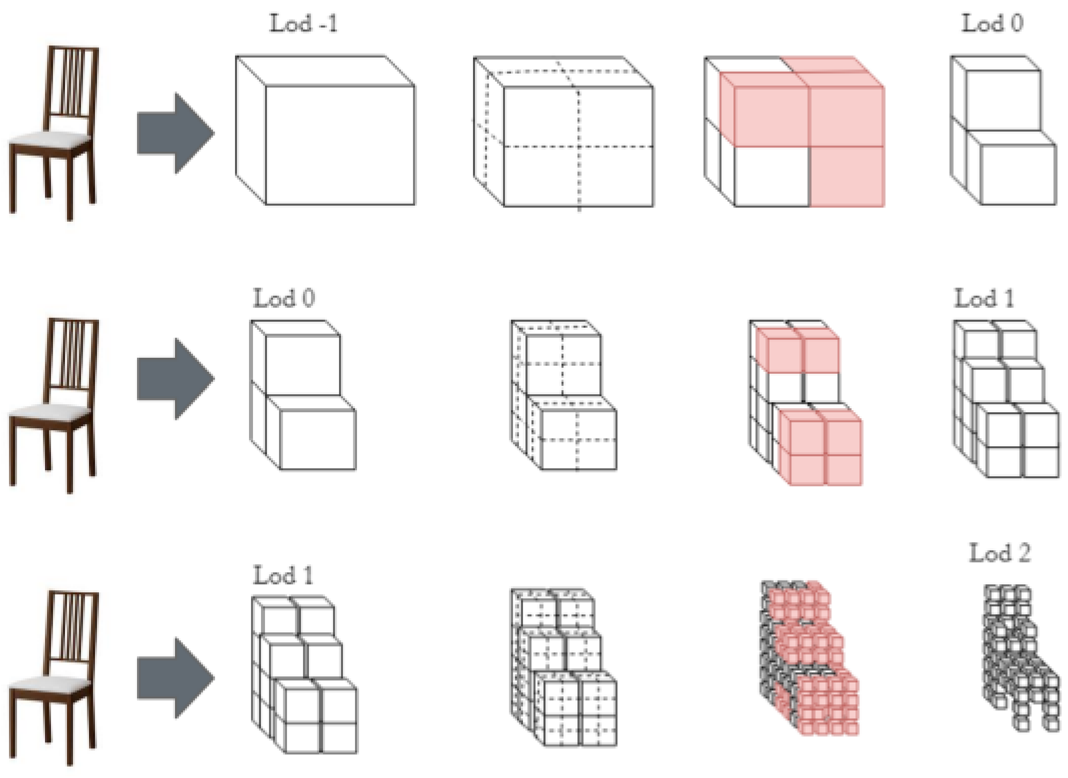


Figure 1.2: Visual representation of how each voxel is split and validated between each LoD, in the process of increasing the resolution.

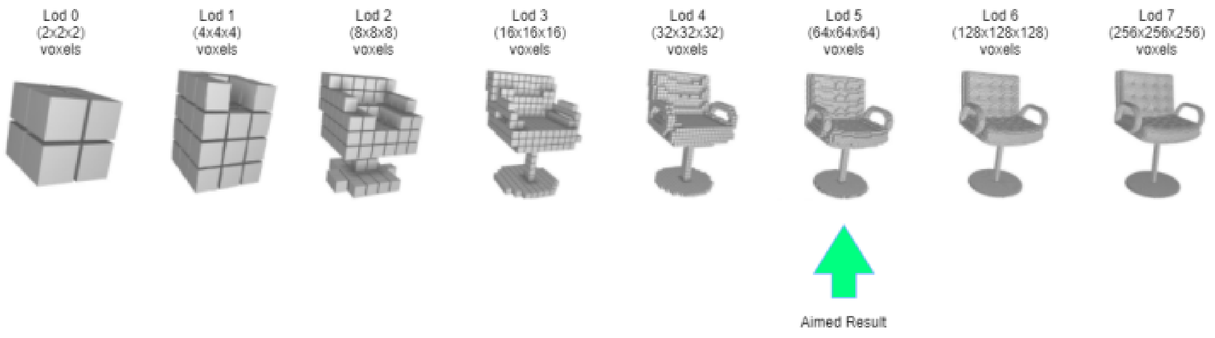


Figure 1.3: Shows an example of the iterative process of generating a chair. The green arrow is pointing at the quality that this thesis is aiming for.

The total amount of voxels required to represent a chair in the training data set, corresponding to a 64x64x64 quality, is between ~ 3000 and ~ 20000 voxels. Comparing the worst case (~ 20.000) against permutations for each voxel in the space, would reduce the total number of predicted voxels to ~ 20000 from 262144 (~ 7.6%) and the best case is ~ 3000 predictions down from 262144 (~ 1.1%). The fact is that the percentage of the total voxel-space decreases with each LoD (More gain from increased quality), see the percentage of total voxel used in Table 4.1 for different LoD.

2.3 Implementation

Environment

The software and hardware used in this project:

† Python 3.5

† Tensorflow 1.3

† Windows 10

† GPU: NVIDIA GeForce GTX 1070 8GB (1.7715GHz)

† 16GB RAM

† Dataset Size: input images ~ 1500, output objects ~ 30

2.4 Testing

2.4.1 Test Plan Objectives

- Data Integration Testing: Verify the effectiveness of data migration and integration strategies. Ensure that datasets from different sources and formats can be seamlessly used in the project without compromising data quality or model performance.

- Security Testing: Assess the security aspects of the project, especially data security. Perform penetration testing and vulnerability assessments to identify and address potential security risks. Ensure that sensitive data is protected from unauthorized access.

- Performance Optimization Testing: Evaluate the project's performance optimization mechanisms. Measure the system's speed, resource utilization, and responsiveness. Optimize the code and algorithms to reduce latency and enhance overall performance.

- Version Control Testing: Verify the effectiveness of the version control system, such as Git. Test the ability to track changes in code, datasets, and documentation. Ensure that collaborative development is efficient, and the rollback to previous versions is possible when needed.

- Data Archiving and Retrieval Testing: Test the data archiving mechanisms to ensure that obsolete datasets can be effectively compressed, stored securely, and retrieved for future reference. Verify that metadata is maintained for archived data.

- GPU Acceleration Testing: If GPU acceleration is utilized, assess its impact on performance. Measure the speedup achieved by offloading compute-intensive tasks to GPUs.

2.4.2 Data Entry

Data is one of the most important aspects of dealing with artificial intelligence. To reduce the scope of this thesis, the dataset is only looking into chairs that can be found in the IKEA dataset [14]. The IKEA dataset is chosen due to its availability and that it is relatively easy to modify and enhance. To make sure that no incomplete data is corrupting the neural network, multiple datasets are considered, using both existing data and generating a new dataset. The IKEA dataset was collected from Joseph J. Lim et al paper from 2013 [14]. The three approaches described below are only used to modify the input, the 3d output is the same for all image datasets. Object files (3D) from the IKEA dataset is converted into a voxelized Python format, using a web-browser converter by Arjan Westerdiep [6]. It uses a simple GUI that is easy to understand and the local computer GPU which makes the conversion quick and 28 painless.

2.4.3 Security

Ensuring the security of a project during the testing phase is paramount. The project's security measures encompass several crucial facets. Data security involves safeguarding sensitive training and testing data through encryption, access controls, and secure storage mechanisms. Code security is achieved through static code analysis, vulnerability scanning, and penetration testing to identify and rectify potential code vulnerabilities. Secure data transfer protocols like HTTPS and VPNs are used for data transmission between systems. User authentication, authorization, and multi-factor authentication protocols are implemented to restrict unauthorized access. Secure development practices, including regular code reviews, are integral in detecting and mitigating security issues early. Network security measures like firewalls and intrusion detection systems protect against external threats. Robust logging and monitoring systems help detect and respond to security incidents, while compliance with relevant regulations and standards ensures legal and security requirements are met. An incident response plan outlines procedures to mitigate the impact of security breaches, and third-party components are regularly monitored for vulnerabilities. In summary, security remains a continuous, iterative process during the testing phase, with regular assessments, audits, and updates to safeguard the project against evolving threats.

2.4.4 Test Strategy

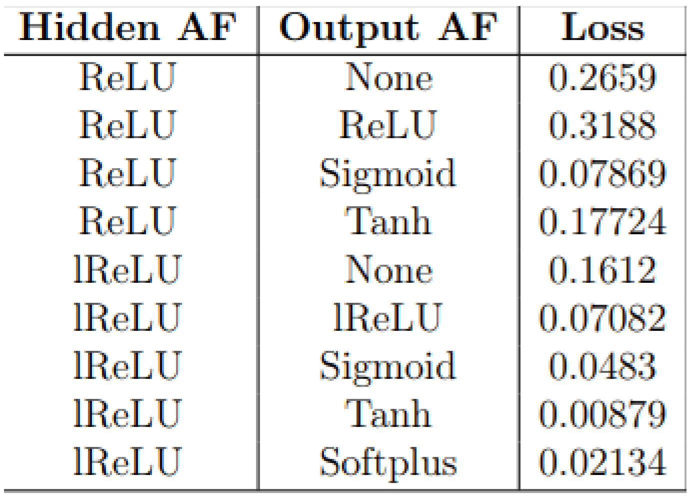
The test strategy for the project involves a multifaceted approach to ensure the reliability and effectiveness of the neural network-based systems being developed. Beginning with a comprehensive requirement analysis, the project's testing phase encompasses various critical aspects. The process commences with unit testing, scrutinizing individual components to verify their functionality. Integration testing follows, focusing on the seamless interaction between these components and data flows. Functional testing is central to the strategy, validating core tasks such as image classification, 3D object generation, and feature extraction. Performance testing evaluates the system's response time and resource utilization, while security testing ensures data protection and identifies vulnerabilities. Usability testing, if applicable, examines the user interface and experience, while load and stress testing assesses the system's robustness under demanding conditions. Regression testing is continuously applied, and documentation is thoroughly verified. Cross-platform, cross-browser, and data quality testing guarantees compatibility and data accuracy. Compliance with legal and regulatory requirements is a priority. User acceptance testing involves end-users in validating the system's alignment with their needs. A strong bug-tracking system is in place, and automation is leveraged to expedite repetitive tests. Test data management strategies maintain data integrity, and continuous monitoring and reporting facilitate informed decision-making. The strategy culminates in stakeholders' sign-off, ensuring the project is ready for deployment. This holistic approach to testing ensures the project's quality, security, and performance meet or exceed expectations.

2.4.5 System Test

System testing is an indispensable phase in the project's development cycle, serving as the ultimate validation of the integrated neural network-based system. This comprehensive testing process scrutinizes the entire system's functionality and performance. It ensures that all individual components, algorithms, and modules function seamlessly when combined, delivering the expected outcomes. System tests encompass functionality verification, performance evaluation, and security assessment. They also assess the user interface's usability, test external integrations, and validate the system's compatibility across different platforms. Load and stress tests gauge the system's capacity, while regression tests verify issue resolutions without introducing new defects. Data quality is closely examined, and compliance with legal and regulatory requirements is confirmed. User acceptance testing, involving stakeholders, ensures the system aligns with end-users' expectations. Overall, system testing is a pivotal step in achieving a reliable, robust, and user-friendly neural network-based solution.

Activation Function

The testing of different setups for activation functions is done by training each setup on 50 epochs. The tests use the same activation function for input layer and all hidden layers, the output layer may differ. As can be seen in Table 3.4 below, leaky ReLU for hidden layers and Tanh for the output layer gave the best (lowest) result.



Hidden AF	Output AF	Loss
ReLU	None	0.2659
ReLU	ReLU	0.3188
ReLU	Sigmoid	0.07869
ReLU	Tanh	0.17724
lReLU	None	0.1612
lReLU	lReLU	0.07082
lReLU	Sigmoid	0.0483
lReLU	Tanh	0.00879
lReLU	Softplus	0.02134

Fig 3.4 MSE result after 50 training epochs (Low is good), using different activation functions (AF). Hidden AF is used for input & hidden layers, output AF is only used for the output layer.

2.4.6 Performance Test

Performance testing is a critical aspect of ensuring the reliability and efficiency of the project. This phase is dedicated to evaluating the system's responsiveness, scalability, and overall speed under various conditions. Performance tests assess how well the neural network-based system functions when subjected to different levels of workload, from average usage to peak demand. This includes load testing, which measures the system's performance under expected and extreme loads, stress testing to determine its breaking point, and endurance testing to ensure stability over extended periods. Scalability tests help predict the system's behavior as the user base or dataset size grows, ensuring it can handle increased data and traffic without compromising performance. Response time, throughput, and resource utilization are meticulously analyzed to identify bottlenecks and areas for improvement. By conducting performance tests, the project can optimize resource allocation, enhance user experience, and guarantee that the neural network functions efficiently even during high-demand scenarios, contributing to the project's success and user satisfaction.

2.4.7 Security Test

Security testing is a fundamental aspect of ensuring the robustness and integrity of the project, particularly given its sensitive nature. To safeguard the project, a comprehensive security testing strategy is implemented during its various phases.

One crucial step is Vulnerability Assessment, where the project's code, including neural network algorithms, is thoroughly examined for potential vulnerabilities. This includes identifying and addressing any weaknesses in the code that could be exploited by malicious actors, such as input validation issues or vulnerabilities related to third-party libraries.

Penetration Testing is also conducted, simulating real-world attacks on the system to assess its resistance to intrusion. This involves attempting to breach security measures, gain unauthorized access, or manipulate data. Any vulnerabilities discovered are promptly remedied to enhance the project's overall security posture.

Data Security is a paramount concern, and encryption techniques are employed to protect sensitive data both in transit and at rest. Access controls and user authentication mechanisms are rigorously tested to ensure that only authorized personnel can access critical components of the project.

To counter Adversarial Attacks, where malicious actors attempt to manipulate the input data to deceive the neural network, robustness testing is performed. This involves assessing the network's ability to recognize and reject adversarial inputs.

Moreover, continuous Monitoring and Intrusion Detection are vital. Security mechanisms are implemented to detect any suspicious activities in real time and trigger alerts. Regular security audits and code reviews are conducted to maintain the highest standards of security throughout the project's lifecycle.

The result was improved as the dropout was increased (see Table 3.3), due to the generalization part being in the feature extraction, this was an expected result

Dropout	0.5	0.75	0.90	0.95	0.99	1.0
Loss	0.27399	0.1710	0.06546	0.0405	0.02028	0.0094869

Fig 3.3 MSE from the empirical approach to optimize the dropout.

2.4.8 Basic Test

The testing process begins with Unit Testing, where individual components and functions are rigorously examined to ensure they perform as expected. This includes testing data preprocessing modules, neural network layers, and data storage functions to verify their correctness.

Following Unit Testing, Integration Testing is conducted to validate the seamless interaction between different components. This phase ensures that data flows smoothly from preprocessing to the neural network and that results are correctly stored and retrieved.

The core functionality of the project is then subjected to Functional Testing, where the complete system is evaluated. This involves feeding sample images into the system and verifying that the generated 3D models align with the expected output.To safeguard against regressions, Regression Testing is an ongoing process, ensuring that existing functionalities remain intact as the project evolves and new features are introduced.Performance Testing assesses the system's efficiency, responsiveness, and overall performance. It gauges how well the project handles varying workloads and stress conditions, confirming that it can generate 3D models efficiently.

2.4.9 Stress and Volume Test

Stress Testing is designed to push the system beyond its normal operational limits. This involves subjecting the project to heavy workloads, often far beyond what it would encounter in typical usage scenarios. The goal is to identify its breaking points and vulnerabilities. For instance, in this project, Stress Testing would involve inputting an exceptionally large number of high-resolution images to assess how the system copes with excessive computational demands. By doing so, weaknesses and potential bottlenecks can be uncovered, allowing for necessary optimizations to be made.

Volume Testing, on the other hand, focuses on evaluating the system's performance when dealing with a vast amount of data. In the context of this project, it entails feeding a substantial volume of images into the system to ensure it can process and generate 3D models efficiently without slowing down or crashing. This phase helps in determining whether the project can handle the expected data growth over time without compromising its performance.

The results of Stress and Volume Testing are invaluable for identifying scalability issues, memory leaks, or processing bottlenecks that may arise as the project encounters larger datasets or increased user demands. By conducting these tests, the development team can fine-tune the system to ensure it remains stable, responsive, and reliable even when subjected to challenging conditions, thereby enhancing its overall quality and resilience.

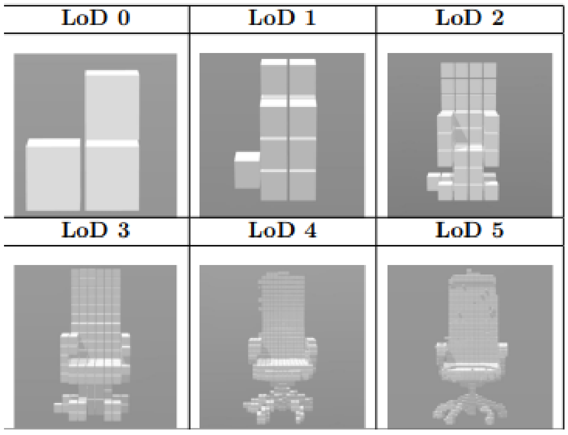


Fig2.3 Visual result for each iteration in the generation process.

2.4.10 Recovery Test

Recovery Testing is a critical component of ensuring the reliability and resilience of the project. This type of testing evaluates how well the system can recover from various types of failures or unexpected events, such as crashes, data corruption, or hardware malfunctions.

In the context of this project, Recovery Testing would involve deliberately inducing failures or disruptions to the system during its operation. For instance, simulating a sudden power outage, intentionally corrupting a portion of the dataset, or causing a crash in one of the system's components. The primary goal is to assess how the project handles these adverse situations and whether it can return to normal operation smoothly.

This testing phase helps in identifying weaknesses in the project's error-handling mechanisms and its ability to maintain data integrity. It also ensures that, in case of failures, the system can recover gracefully without losing critical data or compromising the user experience.

Recovery Testing is essential to guarantee that the project meets its non-functional requirements, especially in terms of reliability and fault tolerance. By systematically testing its ability to bounce back from adverse scenarios, the development team can implement necessary improvements to enhance the system's robustness and ensure uninterrupted functionality, even in the face of unexpected challenges.

2.4.11 User Acceptance Test

User Acceptance Testing (UAT) plays a pivotal role in the project's lifecycle, serving as the final frontier before a system is deployed for production use. During UAT, end-users and stakeholders take center stage in evaluating the project's readiness and functionality. The output of a well-executed UAT phase is multifaceted. It encompasses the validation of project requirements, ensuring that the system aligns with initial expectations and objectives. UAT also delves into the user experience, shedding light on the system's usability and any interface issues that may hinder user interaction. This phase is instrumental in bug detection, where users, interacting with the system under real-world conditions, can unveil issues that prior testing phases might have missed. Moreover, UAT serves as a performance litmus test, evaluating the system's responsiveness, scalability, and overall efficiency. Additionally, UAT often reveals the need for user training and documentation enhancements, ensuring users are well-prepared for the system's deployment. Ultimately, a successful UAT results in user acceptance, marking a crucial milestone for project closure, mitigating post-production risks, and providing a clear pathway for the system's transition to live operation.

2.4.12 System

This encompasses a range of vital elements, including sophisticated software applications like neural network models and data processing algorithms, along with the underlying hardware infrastructure that facilitates their execution. The system also involves secure data repositories for storing and managing datasets, which are essential for training and validating neural networks. It entails a meticulously designed hardware setup comprising powerful GPUs and servers to handle resource-intensive tasks such as training complex models. Security measures like encryption protocols and access controls are embedded to safeguard sensitive data. Furthermore, the system includes rigorous testing mechanisms with quality assurance tools to ensure that the software functions as intended, adheres to performance benchmarks, and meets security standards. Documentation, encompassing installation guides, usage instructions, and troubleshooting resources, is intrinsic to aiding users and maintaining the system. Collaboration tools, version control systems, and monitoring solutions are intertwined to facilitate team coordination, track progress, and ensure seamless operation. Altogether, the "system" embodies a holistic infrastructure that empowers the project to accomplish its goals effectively, efficiently, and securely.

3 Snapshots of the Project

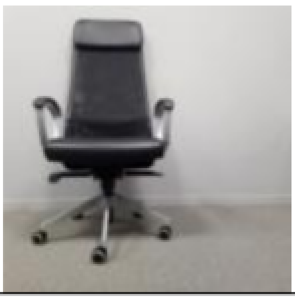
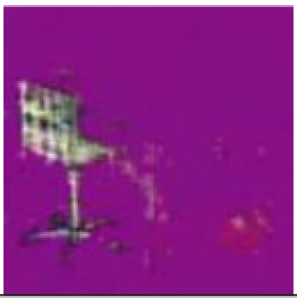
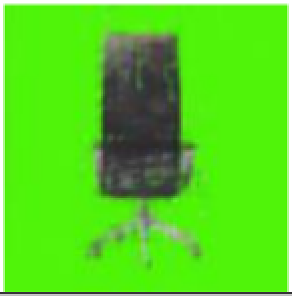
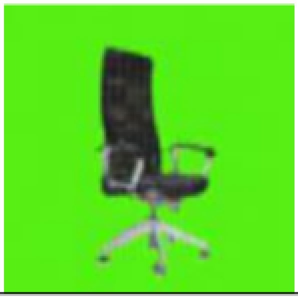
Input	Dataset One	Dataset Two	Dataset Three
Loss	0.31739	0.254156	0.165701
			

Table 3.1 Result MSE from training on different datasets, The bottom row is a 2D-image reconstruction, the purpose of those is only to give the reader an understanding of the numbers. (Low loss is good).

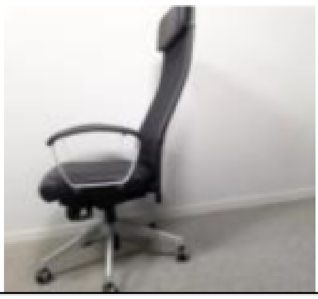
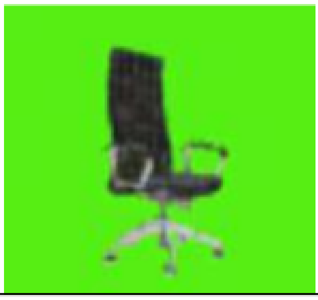
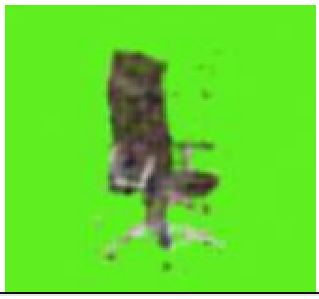
Input	Output dropout=0.5	Output dropout=1.0
		

Table 3.2 Graphical result to visualize the improvement from different dropout fractions.

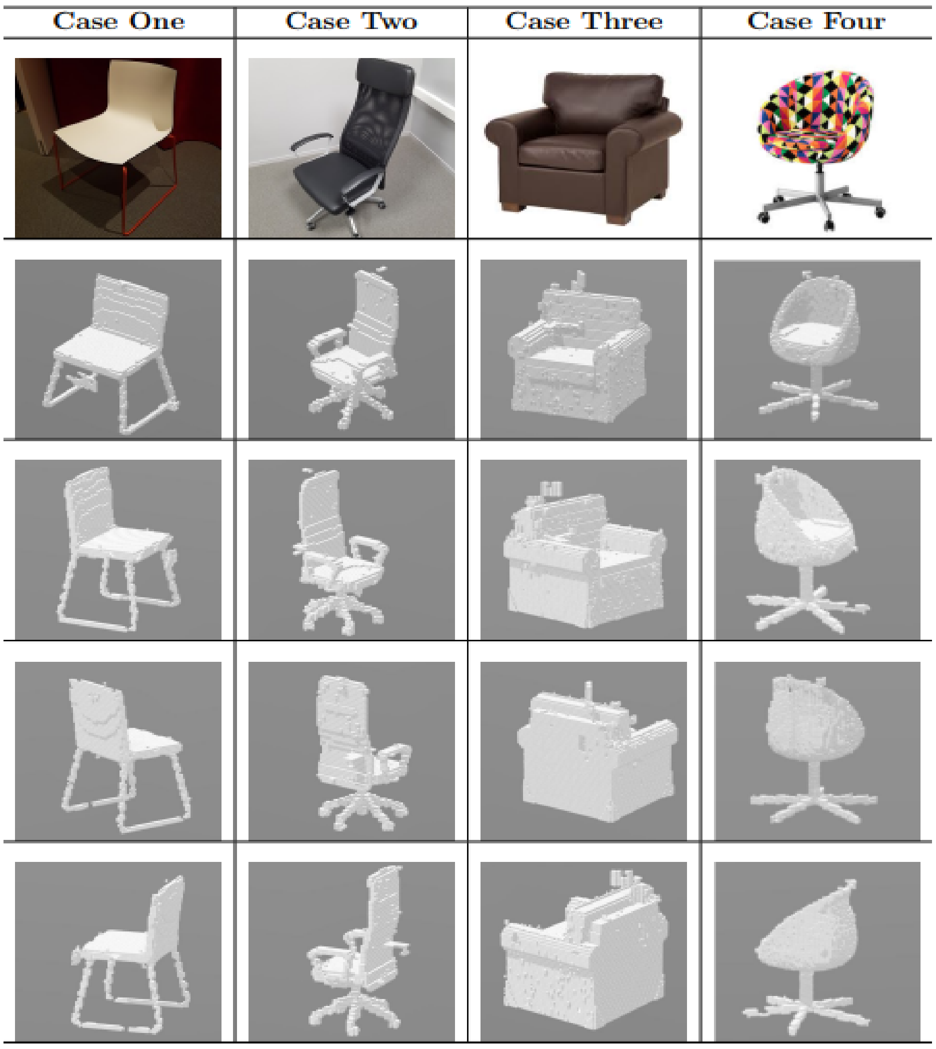


Fig2.1 Visual result of a generated object from 4 different angles.

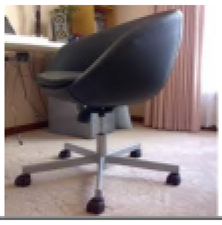
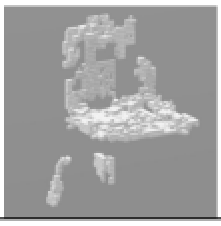
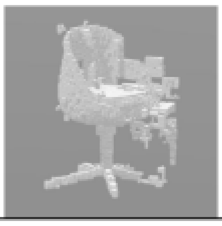
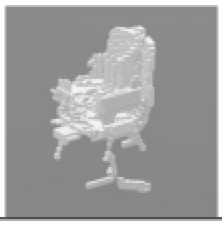
Unknow model	Unknow model	Unknow angle of model
		
		

Table 2.2 Visual result of three cases that gives bad output.

4 Conclusions

By looking at the result, it is possible to see a future where a similar system could be utilized for 3D generation. The key contribution of this project the gained knowledge about different problem areas for generating 3D voxel-based objects from 2D images. However, with the current implementation, the output is too unreliable and slow for high-resolution

Comparison against purpose

† How significant is the trade-off between generation speed and output resolution when generating 3D voxelized objects from a 2D RGB-image, using convolutional neural networks? The trade-off is in our favor. As shown in Section 5.3 the gained resolution grows exponentially by each LoD, whilst the generation time is not exponential. Since the speed has a correlation to the amount of permutation needed to go from one LoD to another, and the permutation needed is getting a lower percentage value with each increasing LoD. In conclusion, increasing the resolution is also increasing the effectiveness. However, the total number of permutations is still growing rapidly, which means that even though a higher quality is technically 58 more effective due to the trade-off, it is still not viable to generate high-resolution objects, since the total generation time will get too high (30 minutes on LoD 6).

One decision that I think has had a negative impact on this project is that the bad result from the first NN, due to dataset One, forced an adjustment on the training. This adjustment was that the output of all images on the same object should be similar. I think this may have been a bad decision since it creates a bottleneck for the second NN, this is why the second NN gained better results without any dropout. Which can be seen as overfitting it in order to work well with the first NN. Due to the time constraint of this project, there was not enough time to remake the first NN when better input data was available. The pros of the solution created in this project are that it does not have any restrictions on how good a resolution it can generate. The output is completely generated by the AI which is good since there is no need for an asset storage for every 3D object that is possible to create. Also, A model like this could be utilized in a Minecraft environment since it does not require a high-quality output. The cons of the solution created in this project is that the generation time is slow for high-resolution output (643 $\leq$ Voxels). Also that the NN is fragile to unrecognized angles of objects and that this NN is only trained to be able to generate objects that it has previous knowledge about

5 Further development or research

This project has improvements that could be done in a few different areas. The first area I will call near-future work, which is to make it more generalized. This could be achieved by modifying the first neural network to compare the reconstruction to the input instead of the fixed image, thereafter increasing the dropout in the second neural network. The second area I will call the experimental-future work, which is another approach to gain a more generalized AI. As mentioned in Section 6.3, Hinton has released a paper on CapsNet [20]. I think it would be an interesting task to replace the first NN with a capsule network since it is shown to be less fragile to unseen variation in the data, in some cases. The third area, technical-future work. This is a problem area that exists but will need some research to possibly find a solution. Since the current model will only activate the weights that are needed to generate each LoD and the needed weights is decided from the previous LoD, there would be a huge improvement to find a faster way to load weights into memory

6 References

[1] Autodesk [Accessed 2017-01-17], 3ds Max [Software], Autodesk.

URL: https://www.autodesk.se/products/3ds-max/overview

[2] Brownlee, J. [2015 (Accessed 2018-01-05)], Basic Concepts in Machine Learning

[bloggpost], Machinelearningmastery.

URL: https://machinelearningmastery.com/basic-concepts-in-machinelearning/

[3] Chioka [2013 (Accessed 2017-11-28)], Differences between L1 and L2 as Loss

Function and Regularization [blogpost], Chioka.

URL: http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-andregularization/

[4] Deshpande, A. [2016 (Accessed 2017-11-01)], A Beginner’s Guide To

Understanding Convolutional Neural Networks [blogpost], Github, URL:

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner’

[5] Despois, J. [2017 (Accessed 2017-10-18)], Autoencoders Deep Learning bits [blogpost], Hackernoon.

URL: https://hackernoon.com/autoencoders-deep-learning-bits-1-11731e200694

[6] Drububu [Accessed 2017-11-06], voxelizer, convert your 3D model to voxels online [software], Drububu.

URL: http://drububu.com/miscellaneous/voxelizer/

[7] Encyclopedia.com [Accessed 2018-01-22], "octree." A Dictionary of Computing., Encyclopedia.

URL:http://www.encyclopedia.com/computing/dictionaries-thesaurusespictures-and-press-releases/octree

[8] expertsystem.com [Accessed 2018-01-20], What is Machine Learning? A definition, Expertsystems.

URL: http://www.expertsystem.com/machine-learning-definition/

[9] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. [2014], Generative Adversarial Networks, arXiv.

URL: http://adsabs.harvard.edu/abs/2014arXiv1406.2661G

[10] Häne, C., Tulsiani, S. and Malik, J. [2017], Hierarchical Surface Prediction for 3D Object Reconstruction, arXiv.
URL: http://arxiv.org/abs/1704.00710

[11] Kafunah, J. [2017 (Accessed 2017-11-01)], Backpropagation in Convolutional Neural Networks [blogpost], Jefkine,

URL:http://www.jefkine.com/general/2016/09/05/backpropagation-inconvolutional-neural-networks/

[12] Karpathy, A. [2016 (Accessed 2017-10-18)], Generative Models [blogpost], Openai.

URL: https://blog.openai.com/generative-models/

[13] Krizhevsky, A., Ilya, S. and Hinton, G. E. [2012], 'Imagenet classification with deep convolutional neural networks', Advances in Neural Information Processing Systems 25 pp. 1097–1105.

[14] Lim, J. J., Pirsiavash, H. and Torralba, A. [2013], 'Parsing ikea objects: Fine pose estimation', 2013 IEEE International Conference on Computer Vision pp. 2992– 2999.

[15] Nielsen, M. [2015 (Accessed 2017-10-18)], Neural networks and deep learning, Determination Press. URL: http://neuralnetworksanddeeplearning.com/

[16] oxforddictionaries.com [Accessed 2018-01-22], Voxel, Oxforddictionaries. URL: https://en.oxforddictionaries.com/definition/voxel

[17] Report of the Secretary-General [2017-05-11], Progress towards the Sustainable Development Goals, UN, URL: http://www.un.org/ga/search/view_doc.asp?symbol=E/2017/66&Lang=E.

[18] Rouse, M. [2017 (Accessed 2017-10-18)], machine learning, Whatis. URL: http://whatis.techtarget.com/definition/machine-learning

[19] Russakovsky, O. [2015], ImageNet Large Scale Visual Recognition Challenge, International Journal of Computer Vision (IJCV).

[20] Sabour, S., Frosst, N. and Hinton, G. E. [2017], Dynamic Routing Between Capsules, arXiv.

URL: http://arxiv.org/abs/1710.09829

7 Appendix

- Files
- data.py
- Helper class, used to parse input data into a usable format
- voxels.py
- Helper class, used to interpret voxel object.
- cube_helper.py
- Helper class, used to interpret cube (ex. converting octree-index into binary-index).
- AIFeatureExtraction.py
- Convolutional Neural Network, used to train and test AI for extracting features from
- 2d RGB-images.

- AIVoxelReconstruction.py

- Convolutional Neural Network, used to train and test AI for reconstructing 3d voxel

Activation Function Structure

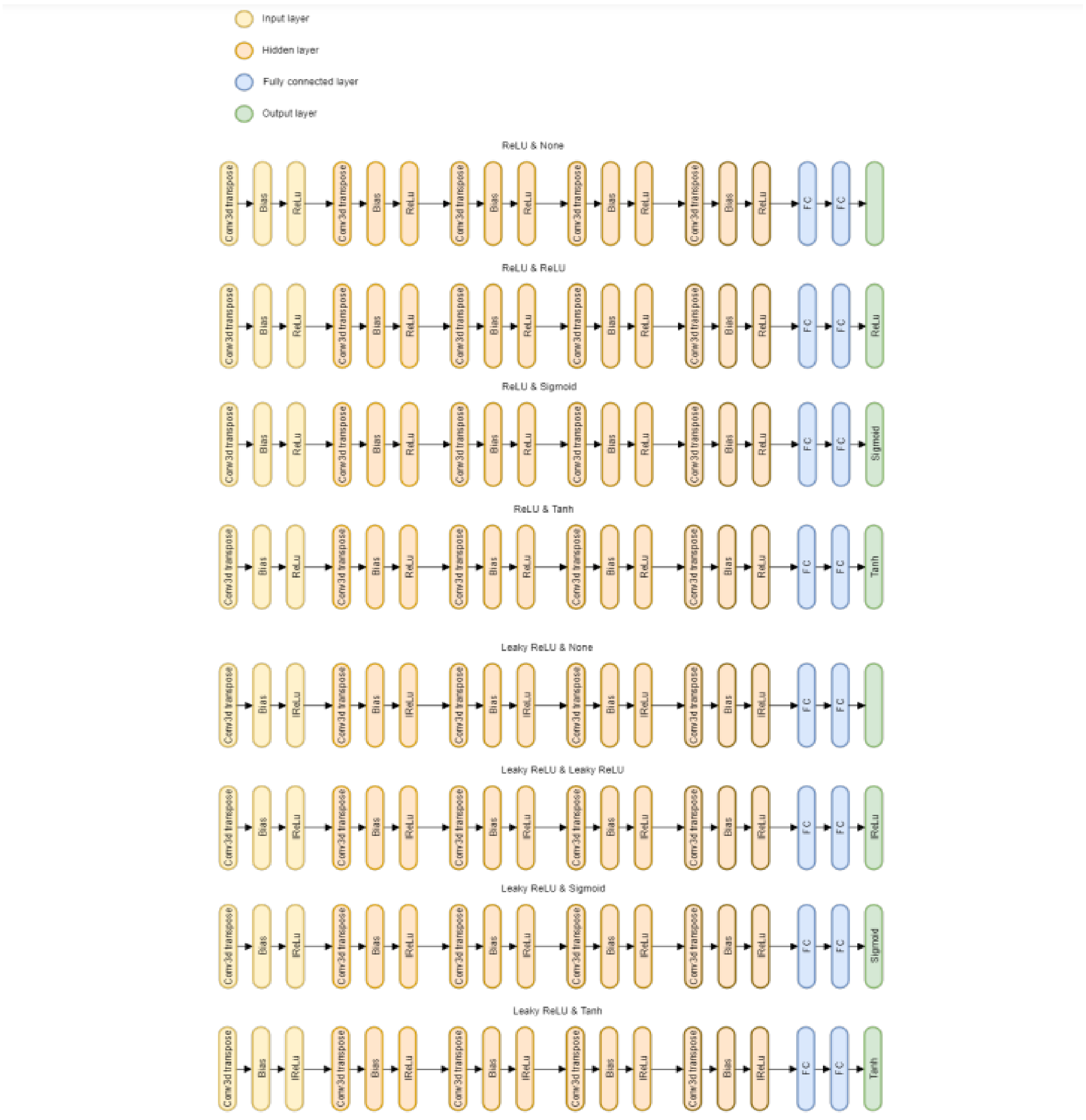


Figure 7.1: Different architecture tested to find optimal structure of activation functions.

Class Hierarchy

Class hierarchy is shown. AIVoxelReconstruction contains the main functionality for training and generation. The generation function is testAI which takes 3 input parameters, the first is the location of where the model is stored; the second is which chair to convert into 3D; the third is which Level of Detail.

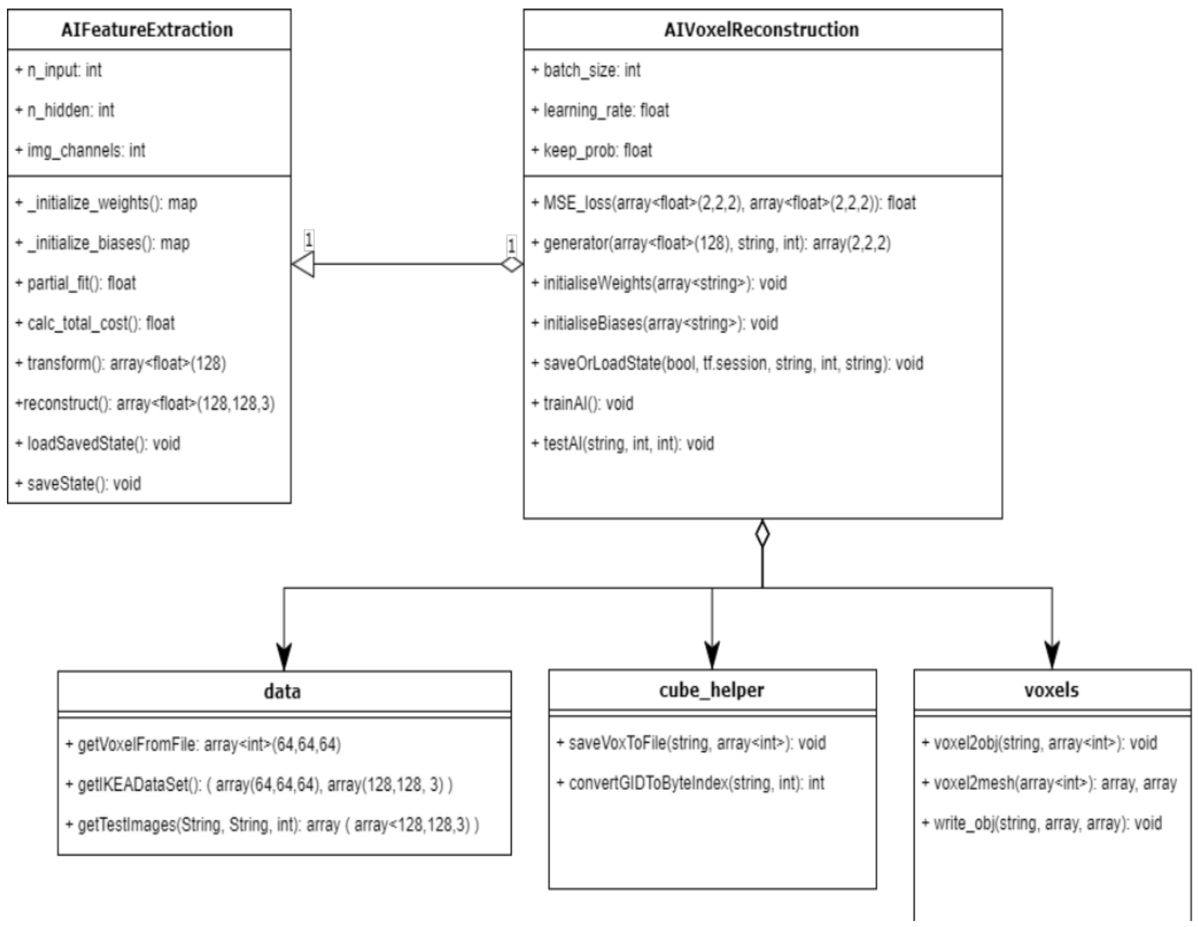

AIFeatureExtraction
+ n_input: int
+ n_hidden: int
+ img_channels: int
+ _initialize_weights(): map
+ _initialize_biases(): map
+ partial_fit(): float
+ calc_total_cost(): float
+ transform(): array<float>(128)
+reconstruct(): array<float>(128,128,3)
+ loadSavedState(): void
+ saveState(): void
1
1
AIVoxelReconstruction
+ batch_size: int
+ learning_rate: float
+ keep_prob: float
+ MSE_loss(array<float>(2,2,2), array<float>(2,2,2)): float
+ generator(array<float>(128), string, int): array(2,2,2)
+ initialiseWeights(array<string>): void
+ initialiseBiases(array<string>): void
+ saveOrLoadState(bool, tf.session, string, int, string): void
+ trainAI(): void
+ testAI(string, int, int): void
data
+ getVoxelFromFile: array<int>(64,64,64)
+ getIKEADataSet(): (array(64,64,64), array(128,128, 3))
+ getTestImages(String, String, int): array (array<128,128,3))
cube_helper
+ saveVoxToFile(string, array<int>): void
+ convertGIDToByteIndex(string, int): int
voxels
+ voxel2obj(string, array<int>): void
+ voxel2mesh(array<int>): array, array
+ write_obj(string, array, array): void

Fig 7.2 Class diagram, visualizing the relation between classes.