Tribhuvan University

Institute of Science and Technology

MODELLING MORPHOGENESIS IN 3D: NEURAL CELLULAR AUTOMATA

A FINAL YEAR PROJECT SUBMISSION IN
PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF
BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND INFORMATION
TECHNOLOGY

UNDER THE SUPERVISON OF

Sumanta Silwal

Lecturer

SUBMITTED BY

Aaditya Jha, 21138/075

Srija Uprety, 21178/075

Shreeyesh Neupane, 21177/075

SUBMITTED TO

TRINITY INTERNATIONAL COLLEGE

Department of Computer Science and Information Technology

Dillibazar Height, Kathmandu, Nepal

May 2023

Tribhuvan University

Institute of Science and Technology

MODELLING MORPHOGENESIS IN 3D: NEURAL CELLULAR AUTOMATA

A FINAL YEAR PROJECT SUBMISSION IN

PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND INFORMATION

TECHNOLOGY

UNDER THE SUPERVISON OF

Sumanta Silwal

Lecturer

SUBMITTED BY

Aaditya Jha, 21138/075

Srija Uprety, 21178/075

Shreeyesh Neupane, 21177/075

SUBMITTED TO

TRINITY INTERNATIONAL COLLEGE

Department of Computer Science and Information Technology

Dillibazar Height, Kathmandu, Nepal

May 2023

DECLARATION

A project entitled "MODELLING MORPHOGENESIS IN 3D:NEURAL CELLULAR AUTOMATA" is being submitted to the Department of Computer Science and Information Technology, Trinity International College, Dillibazar, Kathmandu, Nepal for the fulfilment of the seventh semester under the supervision of Mr. Sumanta Silwal.

This project is original and has not been submitted earlier in part or full in this or any other form to any university or institute, here or elsewhere, for the award of any degree.

BY

Aaditya Jha, 21138/075

Srija Uprety, 21178/075

Shreeyesh Neupane, 21177/075

RECOMMENDATION

This is to recommend that Aaditya Jha, Srija Uprety and Shreeyesh Neupane have carried out research titled *"MODELLING MORPHOGENESIS IN 3D:NEURAL CELLULAR AUTOMATA"* for the fulfilment of the seventh semester in Bachelor's degree of Computer Science and Information Technology under Tribhuvan University under my supervision. To my knowledge, this work has not been submitted for any other degree.

They have fulfilled all the requirements laid down by the Trinity International College Department of Computer Science and Information Technology, Dillibazar, Kathmandu, Nepal.

Sumanta Silwal

Project Supervisor,

Department of Computer Science and Information Technology,
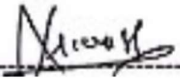
Trinity International College
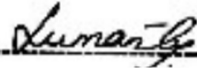
Dillibazar, Kathmandu, Nepal

LETTER OF APPROVAL

17th April 2023

On the recommendation of Mr.Sumanta Silwal, this Project Report submitted by Aaditya Jha, Srija Uprety and Shreeyesh Neupane entitled **"MODELLING MORPHOGENESIS IN 3D:NEURAL CELLULAR AUTOMATA"** in partial fulfilment of the requirement for the award of the bachelor's degree in Computer Science and Informational Technology is a bona fide record of the work carried out under our guidance and supervision at Trinity International College, Kathmandu, Nepal.

EVALUATION COMMITTEE

Satya Bahadur Maharjan

Program Coordinator,

Department of Computer Science

Trinity International College,

Dillibazar Height, Kathmandu, Nepal

Sumanta Silwal

Project Supervisor,

Trinity International College,

Dillibazar Height, Kathmandu,Nepal

External Examiner

Date: ----2080|04|07----

v

ACKNOWLEDGEMENT

ABSTRACT

The use of Neural Cellular Automata (NCAs) has been proven useful in simulating the morphogenetic process. NCAs has found its application primarily in the 2D domain, however, this project expands the use of NCAs to the 3D domain by incorporating 3D convolutions into the neural network architecture. This would enable the simulation of complex 3D structures and spatial dependencies in all three dimensions. The project create a CA model that defines global coordination out of local level interactions. The research findings highlight two significant contributions, namely, an expansion of NCA to 3D voxels and the development of a cellular automation technique for producing voxel structures with different levels of complexity. The project explores multiple aspects, including the structural decay of the system after running for additional iterations beyond the training phase. It also showcases the regeneration property of the system. Additionally, the training behavior of the model was analyzed by varying the number of channels in different datasets.

Keywords: *Convolutional Neural Network Neural Cellular Automata, voxels, morphogenesis, Stochastic update, activation function, gradient optimization*

TABLE OF CONTENT

LIST OF ACRONYMS AND ABBREVIATIONS

3D Three dimension

CA Cellular Automata

CNN Convolutional Neural Network

GNCA Growing Neural Cellular Automata

NCA Neural Cellular Automata

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Morphogenesis is the process through which an organism develops its shape, whereby cells communicate with one another to determine organ and body structure [2]. The creation of tissues and organs from a single cell, has been of interest in regenerative medicine[15][16].However, the process of emergence of complex outcomes from simple rules and feedback loops in morphogenesis is not yet fully understood, and the biggest mystery in this field is how cell collectives know when to stop building. A new area of research at the intersection of developmental biology and computer science involves discovering subroutines and mapping out developmental logic to help understand this puzzle. Cellular Automata (CA) is one approach used to understand morphogenesis by emulating local interactions of cells. Neural Cellular Automata (NCA) is a more sophisticated version of CA that uses neural networks to replace functions that equations cannot encapsulate.

The latest advancements in Neural Cellular Automata (NCAs) have primarily focused on 2D techniques, which involve generating target images from a single pixel [1] or creating endlessly expanding 2D textures [14]. Recent progress has also demonstrated use of NCA for growing Minecraft Entities and regenerating functional machines [7]. However, this approach limits the type of cell into fixed classes of minecraft block types. Our study suggests expanding NCAs into the 3D domain and allowing the cell to range into any of the RGB values by incorporating 3D convolutions into the neural network architecture.

Cellular automata (CA) are mathematical models that are used to simulate the behavior of systems. They consist of a grid of cells that are updated iteratively according to a set of rules. Each cell can have a certain number of possible states, which are typically represented by a set of discrete values. The state of a cell at a particular time step is determined by the states of the cells in its immediate neighborhood at the previous time step. Despite their simplicity, CAs can exhibit complex and interesting behaviors. Although they define a simple set of rules that determine how cells in a grid evolve over time, they can be used to simulate a wide range of phenomena, including patterns of growth, the spread of diseases, and the behavior of complex systems. One of the key features of cellular automata is that they can produce complex and seemingly intelligent behavior from a very simple set of rules. This makes them a useful tool for studying the principles of self-organization and emergent behavior in complex systems.

1.2 Problem Statement

Modelling biological morphogenesis in computation can allow us to gain a better understanding of how the shape and form of an organism develops. Such models both enhance our understanding of biology and translate these discoveries into improved robotics and computational technology. Models that can define global coordination out of local level interactions can serve as a valuable tool for biologists, helping to deepen their understanding of morphogenesis and its underlying mechanisms. Additionally, by using these models to study the development of organisms, it may be possible to create more advanced computational technology that can replicate the complexity and functionality of natural systems.

1.3 Objective

The objectives are:

1. To develop an extension of the methodology described in the paper "Growing Neural Cellular Automata" to facilitate complex structure generation in three dimensions
2. To simulate the growth and development of multicellular structure starting from a single cell.

1.4 Scope & Limitations:

The model developed in this project aims to simulate the growth and development of multicellular structure from a single seed cell using Cellular Automata and Neural Cellular Automata models. By implementing the power of Convolutional Neural Networks, the model can capture how the shape and structure of a system is specified at a cellular level. The project could have applications in robotics, medicine, and biotechnology.

The developed model may not be able to accurately capture all the complexities of the biological morphogenetic process, as this process involves a multitude of factors that interact in a highly intricate and dynamic way. There may be limitations in the computational resources required to implement the model, particularly if the size of the grid or the number of iterations required is large. As with any simulation or model, the accuracy of the results is dependent on the quality and completeness of the input data and the assumptions made in the model design. There may be challenges in verifying the accuracy of the model's predictions, particularly in cases where there is limited experimental data available for comparison.

1.5 Developmental Methodology

The developmental methodology for this project involves conducting a thorough literature review on Neural Cellular Automata (NCA) and its applications, designing a 3D voxel-based extension of the methodology, collecting and preparing relevant data, implementing the model using deep learning frameworks, evaluating and validating the model's performance, iteratively refining the model based on feedback and experimentation, and documenting the entire development process in a comprehensive report. While not strictly adhering to Agile principles, the methodology emphasizes a systematic and structured approach to achieve the objectives of the project.

1.6 Report Organization:

The report is organized as follows:

Chapter 1 introduces the concept of cellular automata and their ability to simulate complex behavior. It focuses on the use of cellular automata to model biological morphogenesis, and proposes the use of Neural Cellular Automata to achieve this. The chapter outlines the problem statement, objectives, scope and limitations of the project, and highlights its potential applications in various fields. Overall, Chapter 1 sets the foundation for the rest of the report by providing background information and outlining the goals of the project.

Chapter 2 of the report contains a background study and literature review on the topic of using neural cellular automata (NCA) for simulating self-organizing systems, including soft robot growth and regeneration. The literature review covers various studies on NCAs, including their application in 3D environments, simulation of morphogenesis, and regenerative capabilities of soft robots. The chapter provides a summary of the main findings and contributions of each study.

Chapter 3 of the report is dedicated to system analysis. This section covers several aspects such as requirement analysis, feasibility analysis, and project analysis. The requirement analysis includes both functional and non-functional requirements, while the feasibility analysis examines technical and schedule feasibility. In addition, the analysis section presents a flow diagram that illustrates the project's workflow. Specifically, a flow diagram depicts the step-by-step process of the project, providing a clear understanding of how the project works.

Chapter 4 of this report focuses on the design aspect of the project. It begins with a sequence diagram that outlines the sequence of interactions between the system and its users. Following that, an activity diagram is provided to illustrate the various activities that occur within the system. Additionally, this chapter includes a detailed description of the algorithms used in the project.

Chapter 5 is all about implementation and testing. This chapter includes the tools used for building the system i.e the programming language as well as the test cases for unit and integration testing performed for the system are included within this chapter.

Chapter 6 provides us with the conclusion and future recommendations.

CHAPTER 2

BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

The concept of cellular automata was first introduced by John von Neumann in the 1940s as a method to model self-replicating machines [10]. In his book "Theory of Self-Reproducing Automata," von Neumann presented a cellular automaton that had 29 potential states for each cell and featured a "von Neumann" neighborhood, in which every cell was linked to the cells above, below, left, and right. He demonstrated that this type of cellular automaton displays dynamics that are comparable to the biological mechanisms associated with self-reproduction and evolution [11].

Several other cellular automata were created, including one by Edgar Frank Codd in 1968, which varied in the number of potential states, connected neighbors, and algorithms employed to determine new states [12]. Later, in the 1970s, Stephen Wolfram introduced the idea of using cellular automata to model complex systems and patterns[3]. However, it was not until the 1990s that neural networks were combined with cellular automata to create GNCA.

The first GNCA model was developed by Hiroki Sayama in 2007, who used it to simulate plant growth. Since then, several variations of GNCA have been proposed, including 2D and 3D models, and models that incorporate genetic algorithms and reinforcement learning [13].

2.2 Literature Review

An article "Growing Neural Cellular Automata" describes an approach for using neural cellular automata (NCAs) [1]. The author of this study performs three experiments where the model types are growing, persistent, and regenerating. In the Growing experiment, the authors are training a neural cellular automaton (CA) to achieve a target image after a random number of updates.

The authors of this study trained their growing models to generate patterns, but found that they did not have the ability to maintain their structure over time. Some patterns would explode or decay, while others were almost stable or even able to regenerate parts. Similarly in the persistent experiment, the authors trained their models to generate patterns that persist for a prolonged period of time using a sample pool strategy. They found that these persistent models

often developed regenerative capabilities without being explicitly instructed to do so. Likewise in the regenerating experiment, the authors subjected their regenerating models to pattern damages during training, which resulted in the models developing much stronger regenerative capabilities, particularly in the central area.

In a study by Sudhakaran et al. extension of NCAs is proposed to the 3D domain, using 3D convolutions in the neural network architecture [7]. They apply this approach to the simulation of structures in the video game Minecraft where Minecraft structure is represented as a 3D grid of cells, each with a state vector containing information about the type of block in the cell, whether the cell is "alive" or "dead," and additional "hidden" information carried through steps of the NCA process. The block type is represented as a one-hot vector, and cells are considered "alive" if they or their neighbors have an alpha value greater than 0.1. The authors of this study evaluated the ability of their Neural Cellular Automata (NCA) to regenerate structures in Minecraft by measuring the ratio of regenerated blocks after the structure was cut in half. They found that an NCA trained for regeneration was able to regrow 99% of the blocks in a Caterpillar functional machine, compared to only 30% for an NCA not trained for regeneration. This work suggests that NCAs may have potential as a tool for simulating complex, self-organizing systems in 3D environments.

In this study, Mordvintsev et al. analyses the ability of the Neural Cellular Automata (NCA) model to simulate morphogenesis, or the local interactions between cells that lead to the development of multicellular organisms [9]. They argue that the original Growing NCA model has a limitation in that it is not invariant to rotation, meaning that it cannot produce differently oriented instances of the same pattern on the same grid. To address this limitation, the authors propose a modified Isotropic NCA (IsoNCA) model that is invariant to rotation. They demonstrate that the IsoNCA model can be trained to grow accurate asymmetrical patterns using either structured seeds or a rotation-reflection invariant training objective. The model was improved in a way that it can now extrapolate to unseen orientations. These findings may help to improve the ability of NCA models to simulate morphogenetic processes.

In another study by Horbie et al. an approach is prepared for simulated soft robots to regenerate damaged parts of their morphology using a Neural Cellular Automata (NCA) [8]. This approach relies on local cell interactions and allows the simulated soft robots to partially regenerate their original shape and regain some of their locomotion abilities. This genetic

algorithm is used to train deep neural networks and uses truncation selection, in which the top T individuals from the current generation become the parents for the next generation. To create the offspring for the next generation, parents are selected randomly and their weight vectors (genotypes) are mutated by adding Gaussian noise. The authors believe that this work takes a step towards giving artificial systems regenerative capacities and could potentially allow for more robust operations in a variety of situations and environments.

CHAPTER 3

SYSTEM ANALYSIS

3.1 System Analysis

System analysis is the process of studying a system to identify its components, functions, and relationships in order to improve its efficiency, effectiveness, and overall performance.

3.1.1 Requirement analysis

Requirement analysis is the process of thoroughly understanding the objectives and expectations of a project or system, gathering information, and identifying the necessary features and functionalities to meet those objectives.

3.1.1.i Functional requirement:

The functional requirement of the system include:

- An actor and a 3D NCA system is present.
- User inputs data then validates, trains and visualizes the model.
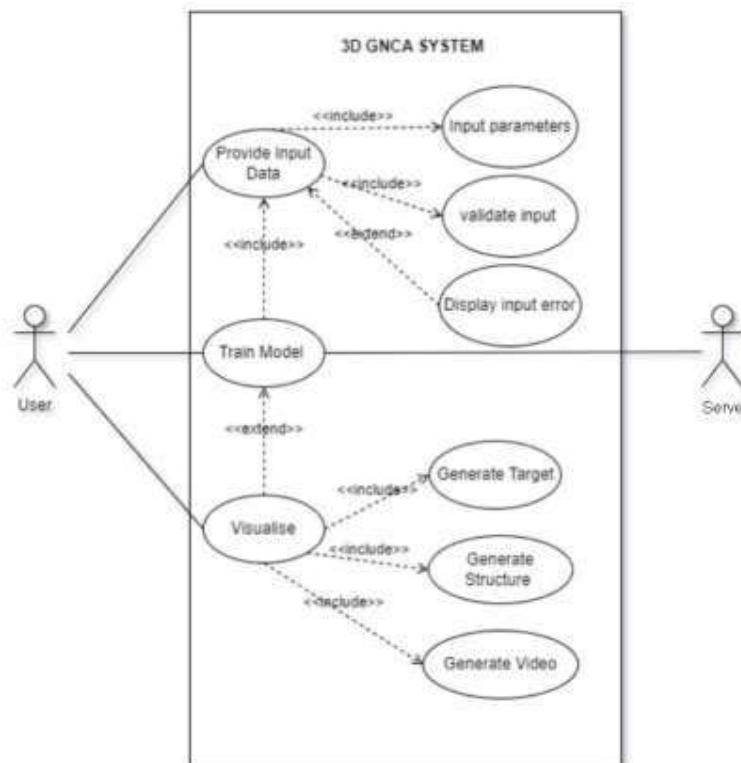- The 3D NCA system trains the model.

Figure 1: Use Case Diagram of system

The "3-D GNCA System" allows users to interact with a Growing Neural Cellular Automata (GNCA) model. Users can input data, train the GNCA model, and visualize the output. The system consists of two actors: User and System.

Actors:

User: Interacts with the 3-D GNCA System.

System: Executes the GNCA model and handles processing.

Use Cases:

Input Data:

- User provides input data for processing.
- Includes: Input Parameter, Validation of Input.
- Extends: Display Input Error (for invalid input).

Train GNCA Model:

- User trains the GNCA model using the provided input data.

Visualize Output:

- User views the generated output of the GNCA model.
- Includes: Generate Target, Generate Structure, and Generate Video.

3.1.1. ii Non-functional requirement:

The non-functional requirement of the system may include:

- **Performance**: The model should be able to generate complex 3D structures efficiently and within a reasonable amount of time. It should be able to handle different sizes of data and run smoothly on the target hardware.
- **Usability**: The user interface should be easy to use and understand, with clear instructions and feedback provided at all times. It should also be intuitive and responsive, allowing users to interact with the system in a natural and efficient manner.
- **Reliability**: The system should be reliable and stable, with minimal errors. It should be able to recover from errors and failures quickly and gracefully, without compromising the integrity of the data or the overall performance of the system.

3.1.2 Feasibility analysis

3.1.2.i Technical feasibility

This project is carried out with the help of a free version of Google colab and in visual studio code if needed to run the code locally.

3.1.2.ii Schedule feasibility

With the necessary resources including hardware, software, and personnel, the project can be successfully completed within a three-month timeframe. A Gantt chart in figure below provides a tentative schedule outlining the project's timeline.

PROCESS	MONTH 1				MONTH 2				MONTH 3			
	Week1	Week2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
Research	1 Jan - 14 Jan											
Developing Model			15 Jan - 7 Feb									
Model Refinement							7 Feb- 4 Mar					
Making Interactable GUI										5 Mar-25 Mar		
Documentation										5 Mar- 25 Mar		

Figure 2*: Gantt chart

3.1.2.iii Economic feasibility

This project is flexible in terms of economic expense. All the software tools used for developing the project are free. The project was developed with the help of the personal devices of the team members and no other computational infrastructure was bought, rented or used throught the project.

3.1.3 Analysis

Analysis in a flow diagram for a software project involves examining and understanding the system's requirements, components, and interactions. It helps visualize the flow of information and identify the inputs, processes, and outputs of the software system. This

analysis aids in the design and development process by providing a comprehensive understanding of the system's functionality.

3.1.3.i Flow diagram

The flow diagram represents a software process consisting of data loading, seed definition, target specification, iterative training, and model evaluation. It begins by loading data, defining a seed, and specifying the target. The system then enters a training loop, generating an output model. The model is evaluated for satisfaction, and if satisfactory, the system continues the training loop; otherwise, the process ends.
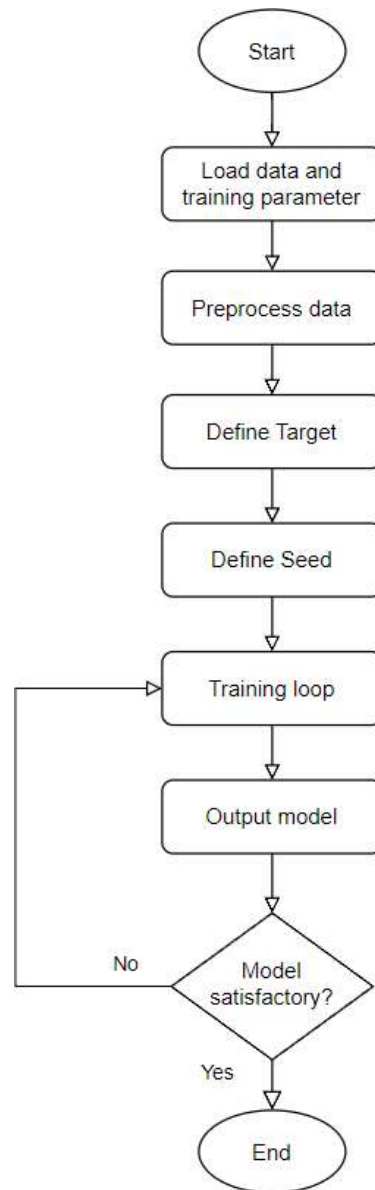

Figure 3: Flow Diagram of system

11

CHAPTER 4

SYSTEM DESIGN

4.1 Design

4.1.1 Sequence diagram

The sequence diagram of the system describes the sequence of activities that occurs between the server system, client system, and the user. In the sequence diagram below the User actor initiates the process by uploading a CSV file containing the information about the 3D model. The client receives the file, and validates the file with the server, and then contacts the Server actor to create the 3D model. Once the Server has created the model, it sends it back to the Website, which then allows the User to either save or visualize the model.
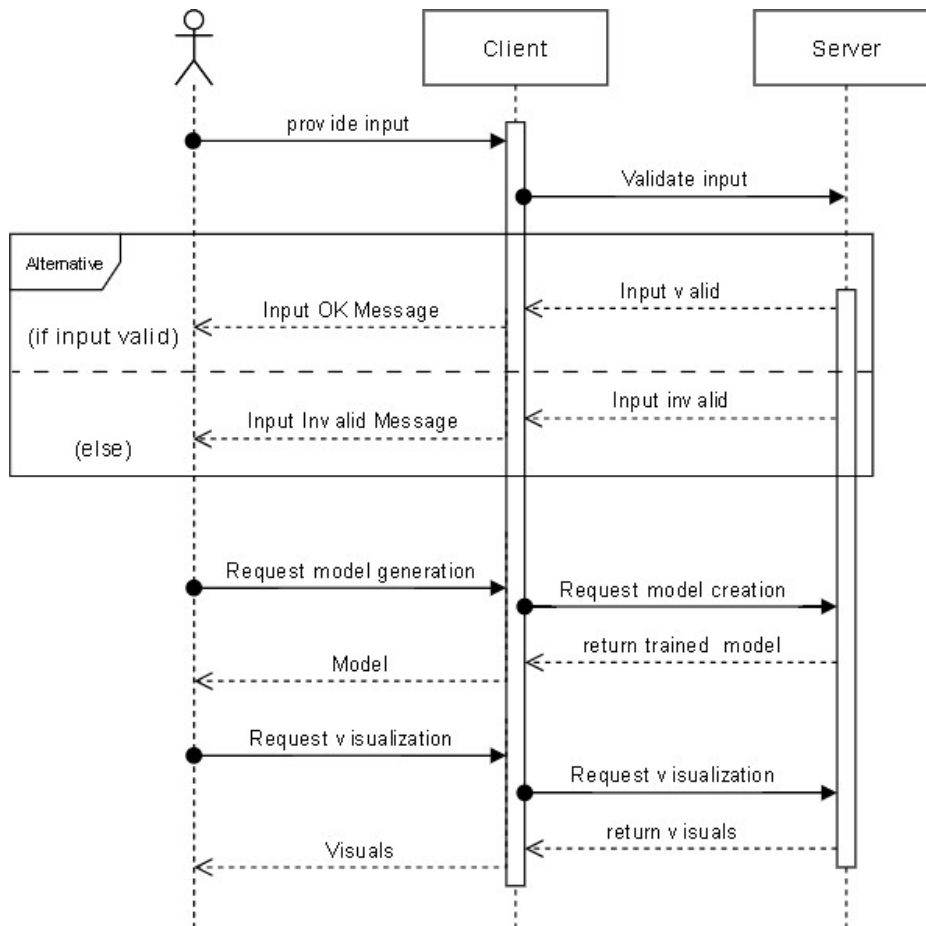

Figure 4: Sequence Diagram

4.1.2 Activity diagram

An activity diagram is a graphical representation of a system's workflow, which portrays the sequence of activities that take place in the system. In the given scenario, the activity diagram depicts the workflow of a system that involves an actor and a 3D NCA system. The first activity is user input, where the actor enters the necessary data into the system. The second activity is validation, where the system checks the user input to ensure that it meets the required standards. The third activity is training, where the system trains the model using the validated data. The fourth activity is visualization, where the system provides a visual representation of the data once the model is trained and the fifth activity in the diagram involves generating a video output by the system that depicts a 3D object.
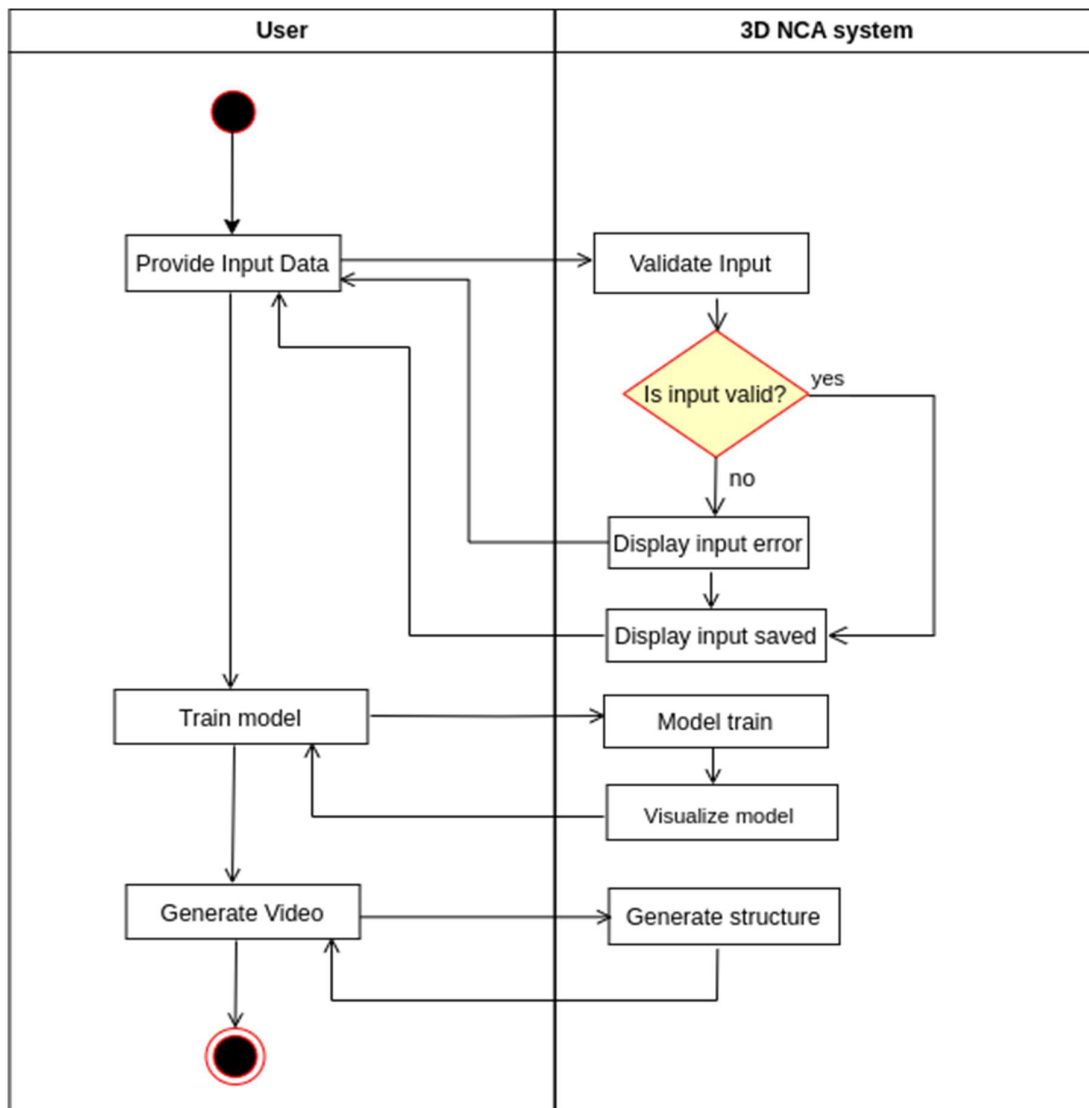

Figure 5: Activity Diagram

4.2 Algorithm Details

Defining Target Object

The process starts by representing the 3d object in a format suitable for convolutional operation. Prior to training the model to learn the update rule defining the global structure of the 3d object, the object is represented as a stack of RGBA values. Every layer in the stack holds information about the 3d object for a particular z coordinate, where the layer also represents the corresponding x and y coordinates of the voxel along with the RGB value. The data is finally presented as a 4d array for further processing. Subsequently, the resulting matrix can be visualized using the matplotlib library.
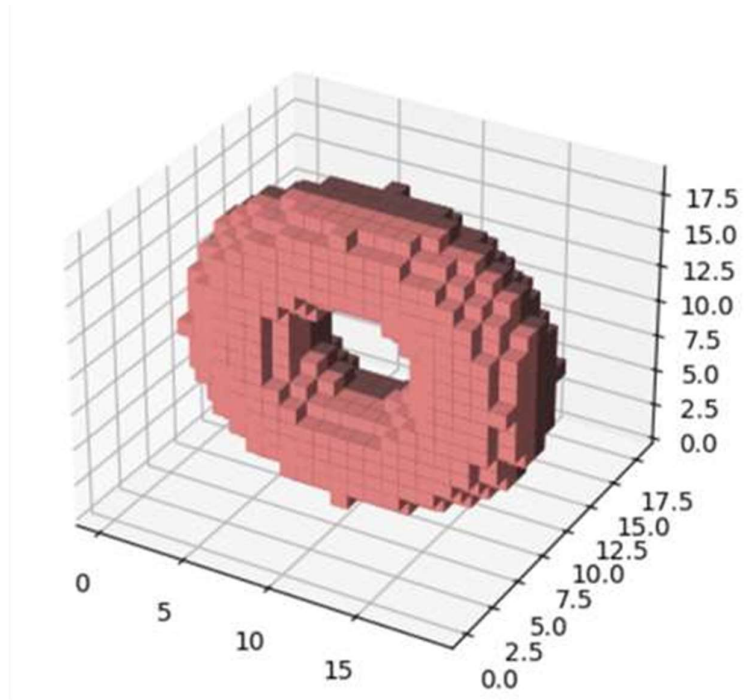
Figure 6: Defining target

The target is defined as a 4D numpy array, where the first three dimensions represent the x, y, and z coordinates of a 3D object, and the last dimension represents the RGB and alive channel. The RGB values for each voxel are normalized to a range of 0 to 1 by dividing them by 255.

Figure 7: Layers of voxel

Define seed

After defining the target, an initial seed must be established for the structure to grow from. This seed comprises of a single cube situated at the center of the 3d space.

Figure 8: Seed value

Model Architecture

The 2D convolution used in [1] is replaced with a 3D convolution to allow for cell awareness of its neighbors in the 3D space. In contrast to [1], a learnable perception network akin to 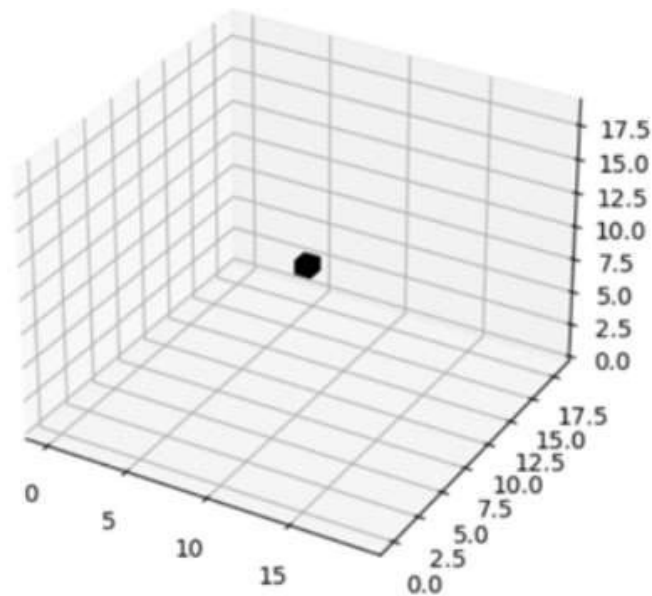[7] is used rather than a static perception network with a Sobel filter. The dynamic perception network is realized as a 3D convolutional layer with a kernel size of 3, a stride of 1, output channels equal to the cell state channels multiplied by 3, and an activation function of tf.nn.relu. Similar to [1], a stochastic update approach is adopted, in which the output of the CA is multiplied by a randomly generated binary matrix with half of its values being 0. Then, the "alive mask" is applied to the updates, which leverages the interaction with the living channel mentioned earlier. This is accomplished by multiplying the updates with the results of a MaxPool layer and a Boolean mask where the value is 1 if it exceeds 0.1 and 0 otherwise.
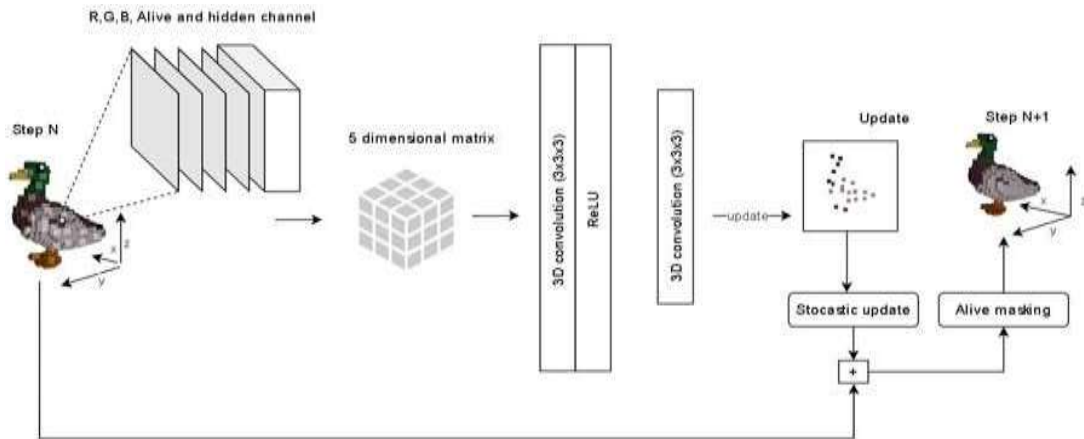

Figure 9: Single update step of the model

Figure above shows how deep learning is used to learn the set of rules that produce a desired multicellular pattern.
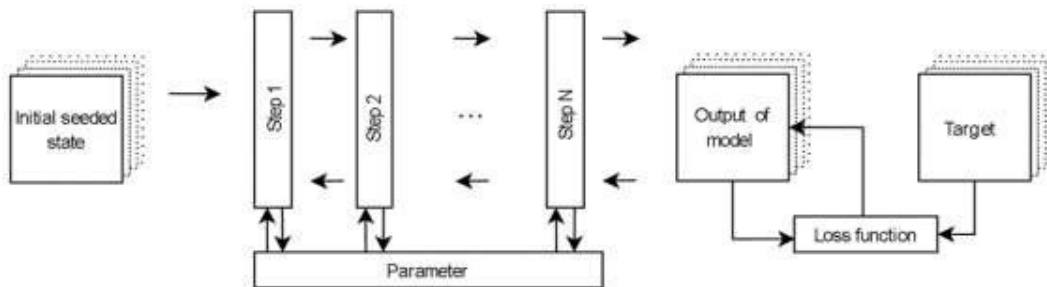
Training procedure

Figure 10: Training scheme for learning target pattern

Using a differentiable update rule, a set of rules can be found through numerical optimization techniques, such as gradient descent, to produce a desired multicellular pattern in 3D space. The approach employed in this work is reminiscent of the method outlined in the paper "Growing Neural Cellular Automata" [1], where the local rules are learned through reconstruction tasks. Similar to previous studies [1][7], the NCA (Neural Cellular Automata) endeavors to generate a target entity from a single living cell and optimize the reconstruction process using supervised learning and a reconstruction loss. Unlike [7], the structure reconstruction task cannot be treated as a multiclass classification problem since there is no fixed number of classes to consider.

To compute the loss, the seed value is initialized and the loss function is determined using mean squared error (MSE). Specifically, the loss is calculated by computing the mean of the sum of the squared differences between the first 4 channels of the output and the target structure:

$$Loss\ (l)\ =\ \frac{1}{n} \sum_{k=0}^{4} (outputt_{k} -\ targett_{k})^2$$

Here n represents the total number of cubes and k represents the channel.

CHAPTER 5

IMPLEMENTATION AND TESTING

5.1 Implementation

Implementation is the part of action to put all the plans into working using different tools and programming language

5.1.1 Tools used

For the implementation of this project, Python was selected as the primary programming language. In addition to Python's native functionality, several supporting libraries were utilised to facilitate the development process. The following libraries were particularly instrumental in the project's successful completion:

1. Numpy
2. Matplotlib
3. Tensorflow
4. PIL library
5. Moviepy
6. Pickel
7. Gradio

These libraries helped in achieving the desired functionality and allowed for efficient coding practices.

5.2 Testing

5.2.1 Testing cases for unit testing

The system undergoes white box unit testing to ensure the accuracy and reliability of each module. During this process, each module is tested independently to verify that its functions are producing the expected output. This approach allows for a thorough examination of the system's internal operations and helps to identify any potential errors or inconsistencies. By conducting white box unit testing, the overall quality and effectiveness of the system can be enhanced, providing a robust and reliable solution.

Table 1: Table of Unit Testing

Module	Input	Expected Output	Observed output	Test result
load_file	"test_file_3x3x6.csv"	[3, 3, 6]	[3, 3, 6]	Passed
load_file	"test_file_3x3x6(1).csv"	[3,3,6]	ValueError: invalid literal for int() with base 10: '6(1)'	Failed
return_Layer	((([[0, 0, 1, 255, 0, 0], [1, 1, 1, 0, 255, 0], [2, 2, 2, 0, 0, 255]]), 2)	([[0, 0, 0], [0, 0, 0], [0, 0, 0]]) ([[0, 0, 0], [0, 0, 0], [0, 0, 0]]) ([[0, 0, 0], [0, 0, 0], [0, 0, 255]]) ([[0, 0, 0], [0, 0, 0], [0, 0, 1]])	([[0, 0, 0], [0, 0, 0], [0, 0, 0]]) ([[0, 0, 0], [0, 0, 0], [0, 0, 0]]) ([[0, 0, 0], [0, 0, 0], [0, 0, 255]]) ([[0, 0, 0], [0, 0, 0], [0, 0, 1]])	Passed
define_seed	-	(1,3,3,6,20)	(1,3,3,6,20)	Passed
loss_function	-	1	1	Passed

5.2.2 Testing case for integration testing

Table 2: Table of Integration Testing
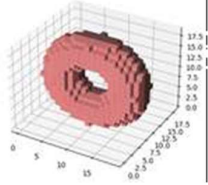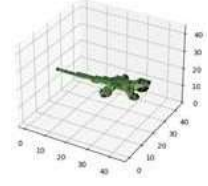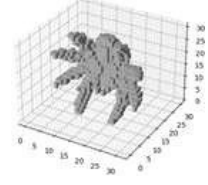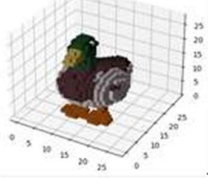
Test Name	Unit Test
Objective	The objective of integration testing is to verify that the different modules or components of the software system, including input data validation, z-layer stacking, seed definition, and Mean Squared Error calculation, are integrated correctly and working together seamlessly.
Expected Output	Unit test successful for checking input data Unit test successful for stacking up z-layers Unit test successful for defining seed for 3D Unit testing successful for Mean Squared Error
Original Output	Integration Testing Successful

5.2.3 Loss Analysis:

Dataset

The dataset comprises a CSV file that contains individual voxel coordinates and their corresponding color values. Each entry in the CSV file consists of six values. The first three values represent the X, Y, and Z coordinates of the voxel, while the remaining three values represent the RGB color values, ranging from 0 to 255. Here is an attached image of the 3D object we are currently working on, along with its associated properties.

Table 3: List of datasets

Entity	No. of Block	No of full Block	No. of Empty Block	Dimension	Plot
Torus	2166	1400	766	19x6x19	
Lizard	6688	443	6245	44x19x8	
Spider	8990	1342	7648	29x10x31	
Bird	11760	2911	8849	15x28x28	

The table above shows how different numbers of channels affect the training and loss. We train the model for 2000 iterations for the CA to generate the target object in 100 update steps.

Table 4: MSE Loss Calculation for different channels

Entity	Training Iteration	MSE Loss		
		16 Channels	**20 Channels**	**24 Channels**
Torus	2000	0.00178936	0.0016519437	0.0019017322
Bird	2000	0.00246781	0.001897112	0.0011209109
Spider	2000	0.00167612	0.001426785	0.0011232341
Lizard	2000	0.00036459	0.001234467	0.0011235789

The table contains information about the performance of a machine learning model for four different entities: Torus, Bird, Spider, and Lizard. The model was trained for 2000 iterations and the Mean Squared Error (MSE) loss was observed for different channel configurations: 16, 20, and 24 channels.

Loss plot of different entities

The plot represents the loss of different entities, specifically the torus, spider, lizard and bird during a training process. The training consisted of 2000 iterations, with each entity having 16 channels for their 3D representation. The plot visualizes the mean squared error (MSE) for each entity over 2000 iterations of training.
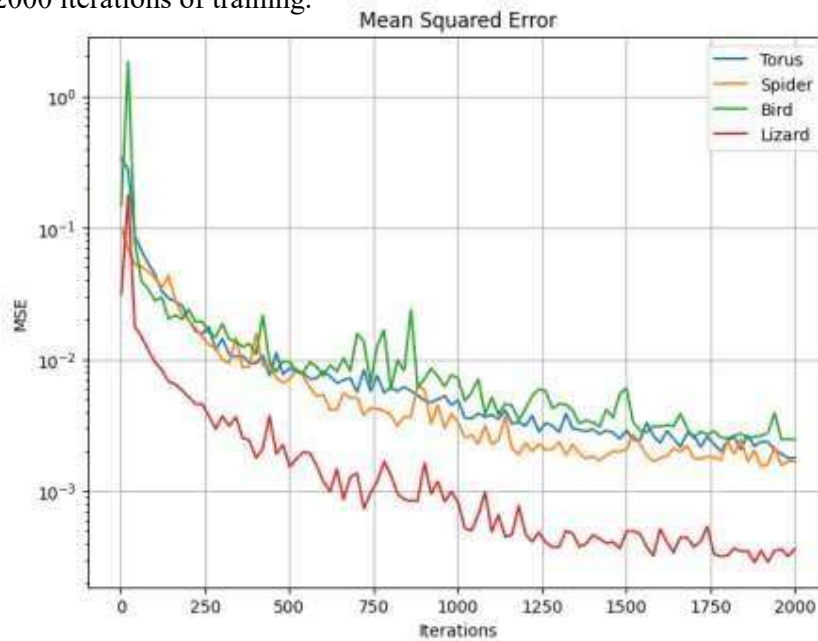
Fig 11: Loss plot of different entities

5.3 Result Analysis

5.3.1 Structure generated over time

The following diagram illustrates the temporal evolution of the structure based on the update rule that the cellular automaton (CA) model was trained to learn. It demonstrates how the structure develops and changes over time.

| Torus step-10 | Torus step-25 | Torus step-50 | Torus step-100 |

Spider step-10 Spider step-25 Spider step-50 Spider step-100

Bird step-10 Bird step-25 Bird step-50 Bird step-100

Lizard step-10 Lizard step-25 Lizard step-50 Lizard step-100

Figure 12: Structural generation over time

5.3.2 Structural decay over time

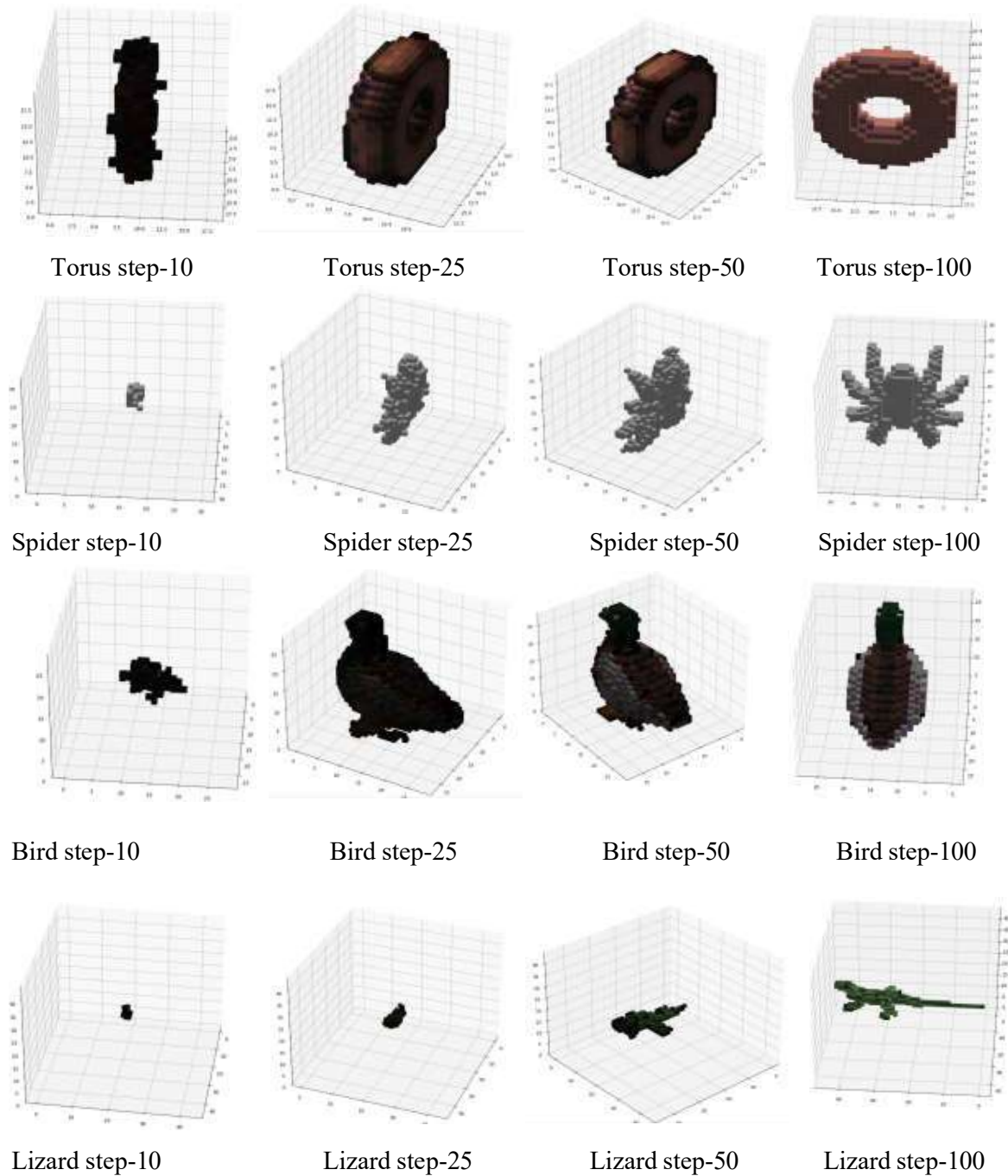In line with the findings of [1], our system encounters instability issues when generating samples beyond the number of steps it was trained for, this can be seen in the figure below. To mitigate this problem, we introduce a "sample pool" with a capacity of 10. The pool is constantly updated with the outputs of each batch, and it starts with a collection of "seed states" that consist of a single living cell.
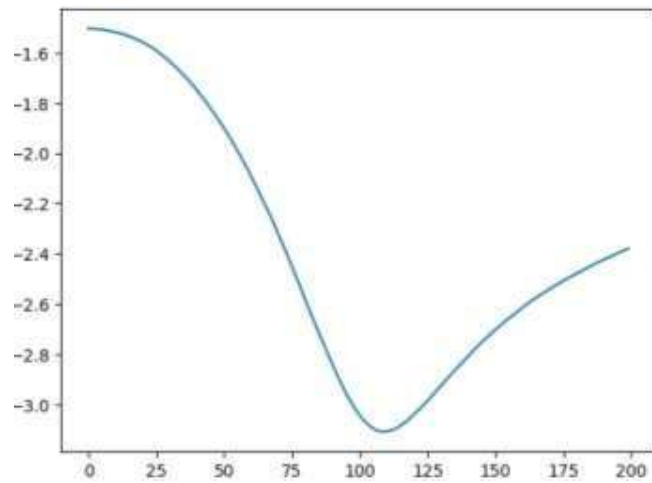
Figure 13: Loss plot after running CA for 200 without mitigating the instability issue
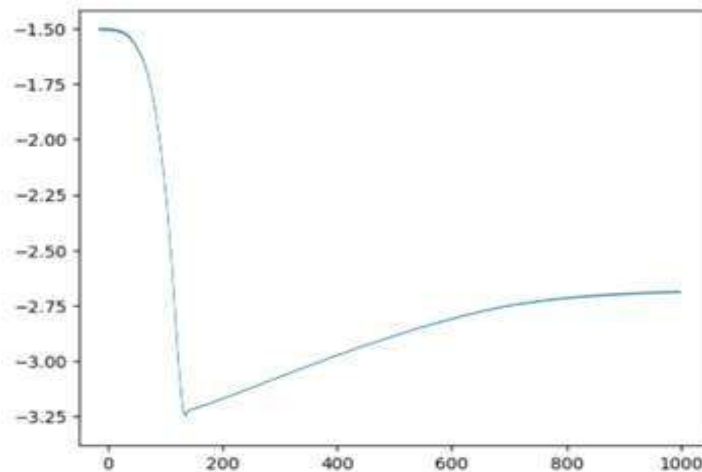
Figure 14: Loss plot for 1000 iteration after mitigating the instability issue

5.3.3 Regeneration

Step 10	Step 20	Step 40
Step 60	Step 60 cutting the lizard in half	Step 70
Step 80	Step 90	Step 100(regenerated tail)

Figure 15: Structural generation over time

The figure 15 demonstrates the regenerative property of the system. At the 60th update step, the tail of the lizard is intentionally severed, and our cellular automaton (CA) model successfully regenerates the amputated tail.

CHAPTER 6
CONCLUSION AND FUTURE RECOMMENDATIONS

6.1 Conclusion

In conclusion, this project attempts to create a model of morphogenesis using Neural Cellular Automata (NCAs) in 3D. The goal is to simulate the growth and development of multicellular organisms startin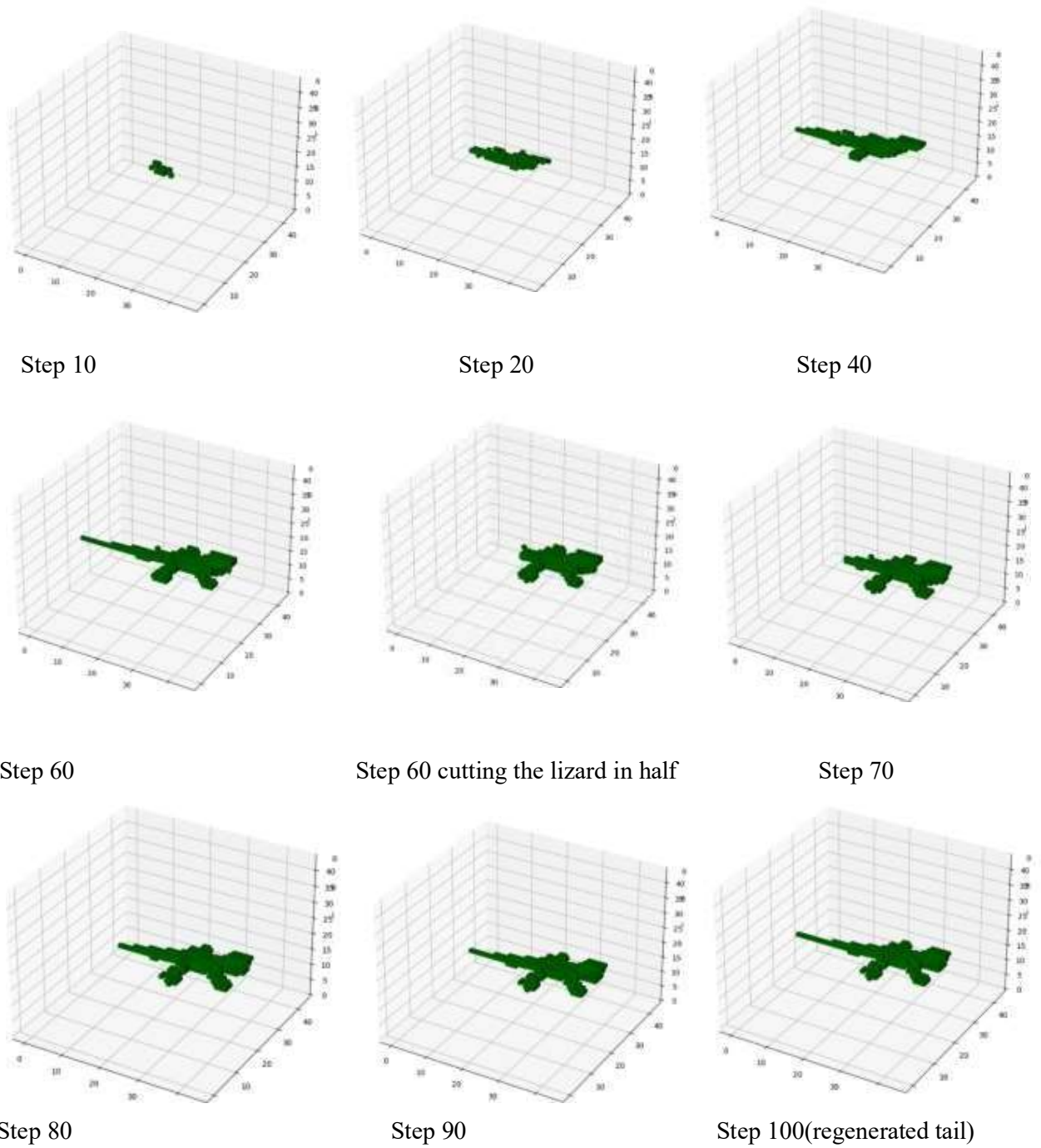g from a single seed, with the aim of enhancing our understanding of morphogenesis and its underlying mechanisms. The use of NCAs allows for the simulation of complex and seemingly intelligent behavior from a simple set of rules, making them a useful tool for studying self-organization and emergent behavior in complex systems.

6.2 Future Recommendation

For further improvement of the system in the future, following features can be implemented:

- **Improve the architecture:** The architecture of the model can be improved to achieve better performance. This can include changing the number of layers, neurons, and convolutional filters used in the model.
- **Dataset expansion**: The dataset used to train the NCA model can be expanded to include voxel with higher resolution. The performance of the current model degrades with the increase in the model resolution.
- **Interactive Interface:** The NCA model could be deployed in an interactive interface for better visualization.

REFERENCES

[1] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin, "Growing Neural Cellular Automata," Distill, vol. 5, no. 2, p. e23, Feb. 2020, doi: 10.23915/distill.00023.

[2] "Morphogenesis," www.bionity.com. https://www.bionity.com/en/encyclopedia/Morphogenesis.html (accessed Jan. 04, 2023).

[3] S. Wolfram, A new kind of science. Champaign, Il: Wolfram Media, 2002.

[4] B. W.-C. Chan, "Lenia: Biology of Artificial Life," Complex Systems, vol. 28, no. 3, pp. 251–286, Oct. 2019, doi: 10.25088/complex systems.28.3.251.

[5] W. Gilpin, "Cellular automata as convolutional neural networks," Physical Review E, vol. 100, no. 3, Sep. 2019, doi: 10.1103/physreve.100.032402.

[6] W. Elmenreich and I. Fehérvári, "Evolving Self-organizing Cellular Automata Based on Neural Network Genotypes," Self-Organizing Systems, pp. 16–25, 2011, doi: 10.1007/978-3-642-19167-1_2.

[7] S. Sudhakaran, D. Grbic, S. Li, A. Katona, E. Najarro, C. Glanois, and S. Risi, "Growing 3D Artefacts and Functional Machines with Neural Cellular Automata," arXiv preprint arXiv:2103.08737, Jun. 2021. , Accessed: Jan. 05, 2023. [Online]. Available: https://arxiv.org/abs/2103.08737

[8] K. Horibe, K. Walker, and S. Risi, "Regenerating Soft Robots through Neural Cellular Automata." Accessed: Jan. 05, 2023. [Online]. Available: https://arxiv.org/pdf/2102.02579.pdf

[9] A. Mordvintsev, E. Randazzo, and C. Fouts, "Growing Isotropic Neural Cellular Automata," arXiv:2205.01681 [cs, q-bio], Jun. 2022, Accessed: Jan. 05, 2023. [Online]. Available: https://arxiv.org/abs/2205.01681

[10] The Embryo Project Encyclopedia (no date) John von Neumann's Cellular Automata | The Embryo Project Encyclopedia. Available at: https://embryo.asu.edu/pages/john-von-neumanns-cellular-automata (Accessed: March 14, 2023).

[11] Theory of self-reproducing automata : Von Neumann, John, 1903-1957 : Free download, Borrow, and streaming (1966) Internet Archive. Urbana, University of Illinois Press. Available at: https://archive.org/details/theoryofselfrepr00vonn_0 (Accessed: March 14, 2023).

[12] Codd, E.F. (1968) Cellular automata (1968 edition), Open Library. Academic Press. Available at: https://openlibrary.org/books/OL5615248M/Cellular_automata (Accessed: March 14, 2023).

[13] Theory of self-reproducing automata : Von Neumann, John, 1903-1957 : Free download, Borrow, and streaming (1966) Internet Archive. Urbana, University of Illinois Press. Available at: https://archive.org/details/theoryofselfrepr00vonn_0 (Accessed: March 14, 2023).

[14] E. Niklasson, A. Mordvintsev, E. Randazzo, and M. Levin, "Self-Organising Textures," Distill, vol. 6, no. 2, Feb. 2021, doi: https://doi.org/10.23915/distill.00027.003.

[15] S. D. Joshi and L. A. Davidson, "Epithelial machines of morphogenesis and their potential application in organ assembly and tissue engineering," Biomechanics and Modeling in Mechanobiology, vol. 11, no. 8, pp. 1109–1121, Aug. 2012, doi: https://doi.org/10.1007/s10237-012-0423-6.

[16] M. A. Kinney, T. A. Hookway, Y. Wang, and T. C. McDevitt, "Engineering three-dimensional stem cell morphogenesis for the development of tissue models and scalable regenerative therapeutics," Annals of Biomedical Engineering, vol. 42, no. 2, pp. 352–367, 2013.

APPENDIX I

APPENDIX II

Load Data

```
#@title Choose and load data from url
import pandas as pd
import numpy as np
import io
import requests

MODEL_3D = "lizard_44x19x8.csv" #@param ["torus_19x6x19.csv",
"lizard_44x19x8.csv", "bird_15x28x28.csv", "spider_29x10x31.csv",
"knight_18x8x15.csv"]
urls = {
    'knight': 'https://raw.githubusercontent.com/Aadityaza/3d-Growing-neural-cellular-au
    'bird': 'https://raw.githubusercontent.com/Aadityaza/3d-Growing-neural-cellular-auto
    'lizard': 'https://raw.githubusercontent.com/Aadityaza/3d-Growing-neural-cellular-au
    'torus': 'https://raw.githubusercontent.com/Aadityaza/3d-Growing-neural-cellular-aut
    'spider': 'https://raw.githubusercontent.com/Aadityaza/3d-Growing-neural-cellular-au
}
entity=MODEL_3D.split('_')[0]

url=urls[entity]
df = pd.read_csv(url,encoding= 'unicode_escape')
arr = np.array(df)
file_name = ((url.split('/')[-1]).split('?')[0]).split('.')[0]
dimension = [int(x) for x in file_name.split("_")[-1].split("x")]
max_dim=int(np.max(dimension))

print('Data loading successful')
print('File name:', file_name)
print('Data shape:', arr.shape)
SIZE_X=dimension[0]
SIZE_Y=dimension[1]
SIZE_Z=dimension[2]
```

Function to return single layer of voxel

```
def return_zLayer(arr, z):
    """Returns a single layer of voxels"""
    z_slice = np.array([arr[arr[:, 2] == z]])
    Zxy = z_slice[0][...,:6].astype(int)
    R = np.zeros((SIZE_X, SIZE_Y))
    G = np.zeros((SIZE_X, SIZE_Y))
    B = np.zeros((SIZE_X, SIZE_Y))
    Z = np.zeros((SIZE_X, SIZE_Y))
    for i in Zxy:
        Z[i[0]][i[1]] = 1
```

```
    R[i[0]][i[1]] = i[3]
        G[i[0]][i[1]] = i[4]
        B[i[0]][i[1]] = i[5]
    return R, G, B, Z
```

Function to stack up layers

```python
def stack_layers(arr):
    """Stacks up all the layers of voxels"""
    r_stack, g_stack, b_stack, z_stack = return_zLayer(arr, 0)
    for i in range(1, SIZE_Z):
        r, g, b, a = return_zLayer(arr, i)
        r_stack = np.dstack((r_stack, r))
        g_stack = np.dstack((g_stack, g))
        b_stack = np.dstack((b_stack, b))
        z_stack = np.dstack((z_stack, a))
    return r_stack, g_stack, b_stack, z_stack
```

Defining and plotting the target

```python
#Defining target
target=np.stack((r_stack/255,g_stack/255,b_stack/255,z_stack),axis=-1)
#Plotting the target
fig_t = plot_target(r_stack, g_stack, b_stack, z_stack, max_dim, SIZE_X,
SIZE_Y, SIZE_Z, flag, FACE_COLOR)
```

Function to create seed

```python
def create_seed(channels, size_x, size_y, size_z):
    """ This function creates a seed array """
    seed = np.zeros([1, size_x, size_y, size_z, channels], np.float32)
    seed[:, size_x//2, size_y//2, size_z//2, :4] = 1
    print("Shape of seed:")
    print(seed.shape)
    return seed
```

Function to return alive mask

```python
def get_living_mask(x):
    """Returns a mask that identifies living cells in the CA grid"""
    alpha = x[..., 3:4]
    return  tf.cast(tf.nn.max_pool3d(alpha,3,1,'SAME') > 0.1,tf.float32)
```

CA model

```python
class CA(tf.Module):
    """Defines the 3D Cellular Automaton model"""
    def __init__(self):
        self.model=tf.keras.Sequential([
            Conv3D(
                filters=Channel*3,
                kernel_size=3,
                padding='same',
                input_shape=(SIZE_X,SIZE_Y,SIZE_Z,Channel),
                activation=tf.nn.relu
                ),
            Conv3D(
                filters=Channel,
                kernel_size=3,
                padding='same',
                kernel_initializer=tf.zeros),
        ])
    @tf.function
    def __call__(self,x):          """Runs a forward pass of the model"""
        alive_mask= get_living_mask(x)
        update_mask = tf.floor(tf.random.uniform(x.shape) + 0.5)
        x= x+self.model(x)*update_mask
        x *= alive_mask
        return x
```

Training step

```python
def training_step(seed, target, ca, BATCH_SIZE, mse_loss, trainer, flag):
    """Performs a single training step"""
    with tf.GradientTape() as g:
        # Repeat the seed BATCH_SIZE times to create a batch of seeds
        x = tf.repeat(seed, BATCH_SIZE, 0)
        N = 100 # Number of iterations
        for i in range(N):
            x = ca(x)
            if flag:
                # Compute loss for RGBA channels
                loss = mse_loss(x[..., :4], tf.cast(target, tf.float32))
            else:
                # Compute loss for only alive channel
                loss = mse_loss(x[..., :4][...,3],
                tf.cast(target[...,3], tf.float32))

        # Compute gradients and update the model parameters
```

```
        params = ca.trainable_variables
        grads = g.gradient(loss, params)
        grads = [g / (tf.norm(g) + 1e-8) for g in grads]
        trainer.apply_gradients(zip(grads, params))
        return loss, x
```

Loss function

```python
# Define the training loop
def train_model(seed, target, ca, training_iterations, batch_size, flag):
    """Trains the 3D Cellular Automaton model"""
    # Create the optimizer, loss function, and learning rate schedule
    lr_schedule = create_lr_schedule()
    optimizer = create_optimizer(lr_schedule)
    mse_loss = create_mse_loss()

    # Track the loss over time
    loss_log = []

    for i in range(int(training_iterations)):
        # Take a training step
        loss, x = training_step(
                            seed, target, ca,
                            int(batch_size),
                            mse_loss,optimizer, flag)

        # Append the loss to the log
        loss_log.append(loss.numpy())

        # Print the loss every 20 iterations
        if i % 20 == 0:
            print(i, loss.numpy(), flush=True)

    # Create a plot of the loss over time
    fig_l, ax = plt.subplots()
    ax.plot(loss_log, '.', alpha=0.3)
    ax.set_yscale('log')
    ax.set_title('Loss over Training Iterations')
    ax.set_xlabel('Training Iterations')
    ax.set_ylabel('Loss')
    plt.close(fig_l)

    return fig_l
```

Structural decay mitigation

```python
#@title Training loop with pooling {vertical-output:true}


#defining optimizer
lr=tf.keras.optimizers.schedules.PiecewiseConstantDecay([1000],[1e-3,3e-4])
trainer=tf.optimizers.Adam(lr)
loss_log=[]
mse_loss = tf.keras.losses.MeanSquaredError()


#creating pool
pool=np.repeat(seed,BATCH_SIZE,0)


@tf.function
def training_step(x):
  with tf.GradientTape() as g:
    N="100"#@param[50,100,200]
    ittiration =int(N)
    for i in range(ittiration):
      x=ca(x)
    if flag:
      loss = mse_loss(x[..., :4], tf.cast(target, tf.float32))
    else:
      loss = mse_loss(x[..., :4][...,3], tf.cast(target[...,3],
tf.float32))

  params=ca.trainable_variables
  grads=g.gradient(loss, params)
  grads=[g/(tf.norm(g)+1e-8) for g in grads]
  trainer.apply_gradients(zip(grads,params))
  return loss,x


TRAINING_ITTIRATION="500"#@param[500,1000,3000,5000,10000]


for i in range(int(TRAINING_ITTIRATION)):
  batch_idx=np.random.choice(len(pool),8, replace=True)
  x0=pool[batch_idx]
  x0[:1]=seed
  loss,x=training_step(x0)
  loss_log.append(loss.numpy())
  if i%10==0:
    print()
    pl.plot(loss_log,'.',alpha=0.3)
    pl.yscale('log')
    pl.show()
    print(i,loss.numpy(),flush=False)
```

Sample input csv file for the system

	A	B	C	D	E	F	G
1	7	0	0	62	108	56	
2	8	0	0	49	78	37	
3	7	1	0	156	193	120	
4	8	1	0	137	168	91	
5	9	1	0	113	147	70	
6	5	2	0	119	155	104	
7	7	2	0	120	158	80	
8	8	2	0	123	160	80	
9	9	2	0	153	178	104	
10	5	3	0	49	78	37	
11	6	3	0	131	172	86	
12	7	3	0	49	78	37	
13	8	3	0	98	142	80	
14	9	3	0	74	97	38	
15	6	4	0	153	180	109	
16	7	4	0	120	167	92	
17	8	4	0	95	130	60	
18	18	4	0	67	101	52	
19	19	4	0	49	78	37	
20	20	4	0	110	151	71	
21	6	5	0	118	157	77	
22	7	5	0	165	187	126	
23	8	5	0	162	187	123	
24	17	5	0	106	148	84	
25	18	5	0	103	149	77	
26	19	5	0	120	162	85	
27	20	5	0	105	137	95	
28	7	6	0	148	180	90	
29	8	6	0	172	190	132	
30	16	6	0	126	176	103	
31	17	6	0	137	178	101	
32	18	6	0	63	93	62	
33	19	6	0	120	154	97	
34	20	6	0	49	78	37	
35	16	7	0	129	170	90	
36	17	7	0	124	164	81	
37	18	7	0	49	78	37	
38	19	7	0	49	78	37	
39	0	8	0	129	169	94	
40	0	9	0	137	178	101	
41	1	9	0	149	186	112	

lizard_24x19x8

User interface

Loss Plot and Images Generator

output 0

output 1

output 2

Flag

CSV File

Drop File Here
- or -
Click to Upload

Training Type
- Structure only,0
- structure with color,1
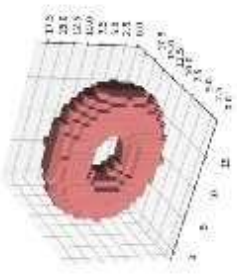
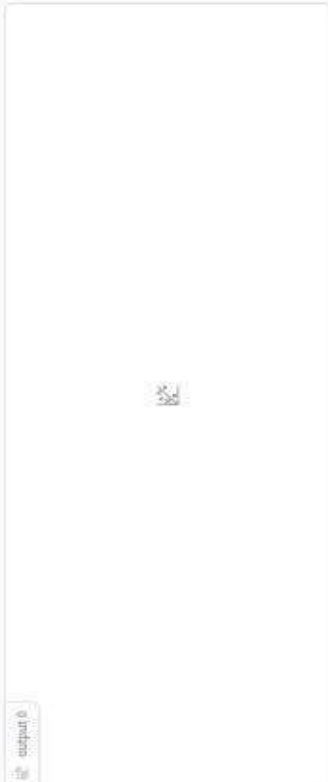Channels
16

Training Iterations
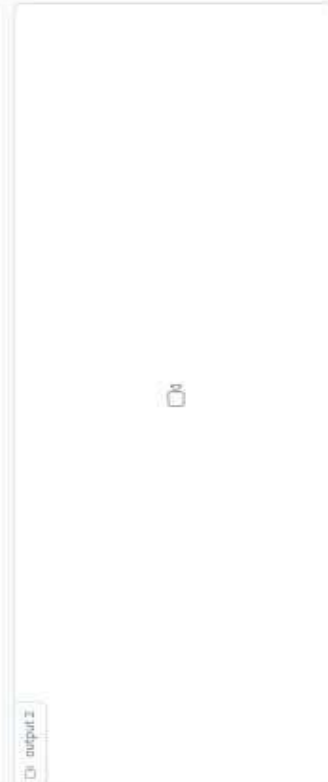1000

Batch Size
1

Color
silver

Clear

Submit

User interface after Training and visualization

Loss Plot and Images Generator