**Abstract**

The introduction of neural networks has revolutionized a variety of classification tasks. This paper explores the potential of using neural networks to detect interesting cellular automata rules. A cellular automaton is a collection of colored cells on a grid that evolves according to a set of neighborhood rules. The rules are applied iteratively for as many times steps as desired to generate new grid configurations. Cellular automata are particularly useful for modelling complicated nonlinear systems in computational science, physics, chemistry, and biology.

A challenge that two-dimensional cellular automata face is a huge parameter search space to generate patterns within. The number of total possible combinations of parameters in the rules can easily exceed several billion. Furthermore, existing research has not discovered a clear pattern among “interesting” rules.

Manually searching for these patterns would be unrealistic as users may have to randomly go through hundreds if not thousands of random rules before finding an interesting one. In this paper, we will discuss our approach to detect rules and patterns that feature gliders (small patterns that move across the grid) using image processing, Convolutional Neural Network (CNN), NASNetLarge, and other techniques. This paper focuses on interesting rules with gliders, not other rules that can be subjectively labeled as interesting, such as intricate still life and oscillating patterns.

**Acknowledgements**

Throughout my years at UC Berkeley, I met many amazing friends, peers and mentors, all of whom have contributed to my accomplishments and progress. It is important for me to recognize all the people who have helped me during my graduate studies and the completion of this paper.

First and foremost, I would like to thank my faculty advisor, Dan Garcia, for all the support, guidance and opportunities he has given me during my time at UC Berkeley. I have taken a course taught by Dan during my sophomore year and was impressed by his enthusiasm for teaching. Later in my senior year, Dan trusted my research abilities and accepted me as his one of his fifth-year master’s students.

He was always interested and involved in my research.

I would also like to thank my friend, Randy Fan,

Additionally, I would like to thank Professor Gerald Friedland,

I learned from him how to think imaginatively,

**Chapter 1**

**Introduction**

A cellular automaton is a collection of cells on a grid of a specified shape that evolves through a discrete time steps according to a set of rules. The grid can be in any finite number of dimensions, and the rules consists of constraints on the current state of the cell and the states of the cells in the neighborhood.

The arguably most famous example is The Game of Life found by John Conway, which is a two-dimensional cellular automaton with two possible states (live and death) and the following set of rules:

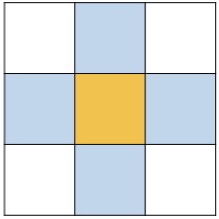
* Any live cell with two or three live neighbors survives.
* Any dead cell with three live neighbors becomes a live cell.
* All other live cells die in the next generation. All other dead cells stay dead.

Different types of patterns occur in cellular automaton. Some of the most common examples are

* Still life, which are static patterns that do not change between generations.
* Oscillators, which are periodic patterns that return to their initial state after a finite number of generations.
* Gliders, which translate themselves across the grid in one or more directions.

Gliders, in particular, are useful for modelling complicated nonlinear systems in computational science, physics, chemistry, and biology. Under most circumstances, it is impossible to tell whether the rule is interesting or boring just by looking at the parameters. Even more daunting, under the assumption that we use Moore neighborhood (shown in Figure 1.1) and a maximum of 10 possible states, there are 29­ survival rules, 29­ born rules, and 210­ states, which leads to a total of 228­ combination of rules. Because of this gigantic number of possibilities, automatic detection of interesting rules will be very helpful, which makes sure that users do not have to manually go through the process.

A picture containing chart

Description automatically generated

**FIGURE 1.1: (Left) Moore Neighborhood: the surrounding eight cells of a center cell.**

**(Right) Neuman Neighborhood: the surrounding four cells of a center cell.**

In this paper, we define interesting rules specifically as the rules which result in patterns of clear gliders. All the remaining rules which lead to stasis, noise with no discernible patterns moving across the screens, or some patterns other than gliders, are classified as boring. In Figure 1.2 are some of the famous gliders that have been discovered so far.

A picture containing text, light

Description automatically generated A picture containing light, traffic, traffic light

Description automatically generated

A picture containing shape

Description automatically generated A picture containing text, device

Description automatically generated

**FIGURE 1.2: Four examples of discovered glider patterns.**

**(Top Left) Brian’s Brain. (Top Right) Burst.**

**(Bottom Left) Brain6. (Bottom Right) Star Wars.**

After manually inspecting 40,000 patterns, it became reasonable to make the assumption that interesting rules will produce interesting results regardless of the initial configuration. Similarly, boring rules are expected to always produce boring results.

This paper explores the possibility of using deep neural networks to detect interesting cellular automata rules. Specifically, we will focus on outer totalistic generations of two-dimensional cellular automata, which means the state of the cell at time *t* depends on both its own state and the total of its neighbors at time *t – 1*. We will use Moore neighborhood for the update rules.

Deep neural networks, a branch of machine learning, are computational algorithms that can extract information from complicated data to detect patterns or trends which are too convoluted for human brains and other computer techniques. The most unique property of neural networks is that once trained, they are able to learn and adapt to new situations on their own. In this way, their learning process resembles the cognitive development of human brain, which are made of neurons, the fundamental building unit for information transmission. These characteristics make neural networks better candidate than humans to distinguish the interesting rules of cellular automata.

**Chapter 2**

**Related Work**

There have been many approaches to find interesting two-dimensional cellular automata rules. The most naïve is to repeatedly create random neighborhood rules and wait to see if it generates an interesting result. This method is inefficient, random, and the experiment became a matter of luck due to its gigantic parameter search space. Another approach that works for lower dimension cellular automata (specifically one-dimensional), whose parameter search space is small, is simply brute forcing all the possible sets of rules and manually inspecting which ones are interesting after they have been generated and the frames are saved. But brute forcing is impossible for higher-dimensional cellular automata. An alternative naïve approach is to tweak existing interesting rules, such as John Conway’s Game of Life rules, to generate similar or refined patterns. However, usually just changing a single parameter value could lead to a widely different result. This means that the interesting results are not necessarily clustered together in the search parameter space.

There exist other more computational methods used to determine the type of cellular automata generated. For example, Chris Langton created a cellular automata lambda value that is computed based on the number of cells that have been born at that time step and dividing it by the total number of cellular automata cells. This gives a decimal value between 0 and 1. The endpoints of the interval correspond to the Still Life and Chaos respectively. Based on his classification, a lambda value within 0.1 and 0.15 indicates a good rule that could require further investigation. The most well-known classification of cellular automata is introduced by Stephan Wolfram, which consists of four classes representing automata in which patterns stabilize into homogeneity, automata in which patterns evolves into mostly stable or oscillating structures, automata in which patterns evolves into chaos, and automata in which patterns become extremely complex. Based on his classification, the fourth class is considered to be potentially computational universal and worth investigating. But Wolfram did not establish a connection between the classifications and the rules themselves.

None of these described methods have been able to reliably detect rules that may be subjectively determined as interesting (e.g., they usually find static noise). Thus, detecting interesting gliders in two-dimensional outer totalistic cellular automata is an unsolved problem that we attempt to tackle in this project.

**Chapter 3**

**Evaluation Method**

We first collected the two-dimensional cellular automata data needed for training and testing. This entailed a data collection pipeline from scratch to generate a sequence of raw frames and then stitching the selected images to represent each of the patterns. Later, several models were tested and analyzed for the best results. We trained the data with a CNN, and performed hyperparameter tuning, image feature extraction via NASNetLarge, and entropy analysis.

**3.1 Frame Generation and Image Stitching**

To obtain boring rules, we manually went through random examples and collected rules that died out immediately, generated static noise, or boring non-glider patterns. For interesting rules, we used Cellular Automata Rules Lexicon, and example provided in Visions of Chaos and recorded those with gliders. We ended up 105 boring and 35 interesting sets of rules.

**Algorithm 1** Cellular Automata Generation

**function** GENERATE(center\_cell, max\_state, survive\_arr, born\_arr)

**if** center\_cell == max\_state **then**

**for** num\_neighbors in survive\_arr

**if** total - 1 == num\_neighbors **then**

             return center\_cell

**end if**

     return center\_cell – 1

**else if** center\_cell != 0 **and** center\_cell != max\_state **then**

     return center\_cell – 1

**else**

     for num\_neighbors in born\_arr:

**if** total == num\_neighbors **then**

             return max\_state

**end if**

         return 0

**end if**

**end function**

We created the cellular automata generation and used the algorithm to generate frames and create images and ran the rules using a random initial configuration. Next, we applied data augmentation due to the limited number of rules that we identified. The boring rules were reused 10 times, and the interesting rules 30 times with different initial configuration. 140 images are generated for each of the pattern.

Our next step is image stitching. Frames 100-108 of the evolution were stitched together in a 3×3 grid. This was designed to increase the algorithm’s robustness and to control better for interesting configurations that have some seemingly uninteresting frames interspersed throughout their evolutions. As raw data, these images were quite large given the RAM allocation. Running the notebook tended to crash the kernel so we settled for less resolution, down sampling the 1188×1188-pixel images to 300×300. This tradeoff allowed us to manipulate and do machine learning on the data without too much computational expense. The training data is assembled with a 50% interesting and 50% boring split.

As a final preprocessing step, the [0, 255] valued matrices representing the images were normalized using simple division to [0, 1]. This improved performance greatly in practice. Many of the CNN architectures we tried produced sub-baseline results before this step. The next part of the optimization process was a question of model architecture and hyperparameter tuning.

**3.2 Data Training with CNN**

For our model architecture, we implemented a CNN, which is a logical choice given the task being to classify images. The machine learner aims to predict whether the generated cellular automata will be interesting or boring given a set of rules and an initial configuration. We tried many architectural parameters and hyperparameters in order to create the best model, including convolutional filter size, dense layers at output, pooling kernel size, type of pooling, dropout, and batch normalization.

We found that the greatest improvements happened after adding dropout and batch normalization. There was also a significant increase in accuracy after increasing the convolutional filter size of the first convolutional layer to 5×5 from 3×3. We believed this is because 3×3 is too small to capture much of the complexity of the interesting configurations. Given a 3×3 window, many of the interesting shapes look like noise.

We tried many things that did not work in addition to those that did. Increasing the pooling kernel size, using average pooling instead of max pooling, increasing the number of filters in the convolutional layers (from 64 in each), and increasing the second convolutional layer’s filter size from 3×3 to 5×5, all resulted in worse performance by the validation accuracy metric. We found that increasing the epochs past 30 resulted in overfitting.

Eventually, we used the architecture described in Figure 3.1.

Table

Description automatically generated

**FIGURE 3.1: Structure of the CNN.**

The machine learner was able to achieve 93.44% training accuracy and 84.12% testing accuracy on the testing set with 10% interesting data. The test recall is 100%, indicating every interesting configuration has been correctly labeled as such.

**3.3 Feature Extraction**

**3.4 Entropy Analysis**

According to the random nature of cellular automata patterns, entropy is an adequate metric to compute since it is likely that there is correlation between the label (boring and interesting) and the statistical measure of randomness in the images.

**3.4.1 Cross Entropy Evaluation**

We first used cross-entropy as the default entropy function.

**Algorithm 3** Image Cross-Entropy Algorithm

**function** COMPUTE\_ENTROPY(signal)

lensig = signal.size

symset = list(set(signal))

probpab = [np.size(signal[signal == i]) / (1.0 \* lensig) for i in symset]

entropy = np.sum([p \* np.log2(1.0 / p) for p in propab])

return entropy

We iterated through all the stitched frames and computed their entropies using the cross-entropy algorithm, then plotted the entropy values of the boring and interesting images (Figure 3.2).

Chart, histogram

Description automatically generatedChart, histogram

Description automatically generated

**FIGURE 3.2: Entropy distribution of cellular automata patterns.**

Boring images had entropy values that spread roughly evenly from 0.0 to 3.5, with a small gap between 0.5 and 0.75. There are many which had entropy values close to 0.0, which is reasonable because they would likely correspond to Still Life. On the other hand, interesting images have entropy values concentrated in the 0.0 to 2.0 range, especially between 0.5 and 0.75. This is intuitively true because interesting images have less noise and entropy compared to boring images on average. It should be noted the minimum entropy for boring images was 0.0 while the minimum entropy for boring images was 0.0318. This is because frames that had no live cells were always labeled as boring. The entropy values suggest adding features identifying if the image entropy is above 2.0 or equals to exactly 0 may be beneficial for the model accuracy.

**3.4.2 Customized Entropy Evaluation**

Instead of using cross-entropy of the stitched images, we tried to use a customized entropy. We loaded the nine images 100-108 as arrays and computed an elementwise (pixel-by-pixel) difference. The mean norm of the difference is used as the new entropy.