Predicting Stability within the Mandelbrot Set

Wes Jones, Zack Lamb
CS-435 Final Project Report
11/30/23

Introduction:

In this project, we aimed to use a 1D CNN to predict stability within the mandelbrot set. The Mandelbrot set is a beautiful fractal that resides on the imaginary plane. (More details in the background). The goal of this project is to train a model that would be able to recognize instability within the iterations of the Mandelbrot. This is important because of how it relates to the field of chaos theory. Many things like weather, populations, the stock market, and planetary orbits fall under the area of chaos theory. Mandelbrot himself used chaos theory to predict that a market crash would happen roughly every ten years, despite the fact that they should theoretically never happen, and he has been mostly correct. In the medical field, chaos theory has been used to analyze sleep disorders, and heart disease, as well as control brain seizures. In the Social Sciences, chaos theory has been used to gather information on social phenomena and theorize about beneficial amounts of disorder. In the data section below, we will show a few theoretical examples showing exactly how this ties to real world applications. We chose the Mandelbrot set for a few reasons. The first is because it's a well-known fractal and there's a lot of information on it. The second reason is because of its relative simplicity. The Mandelbrot set is the easiest way to observe chaos theory, and its simple data has low noise and is easy to understand. This is a good Big Data project because of the large amount of computing power it takes to generate this fractal. Combined with the digital expense of training a neural network, this scaled into an appropriately large big data project. After our final testing, our model was able to successfully predict stability in fewer iterations, but not in the way we expected.

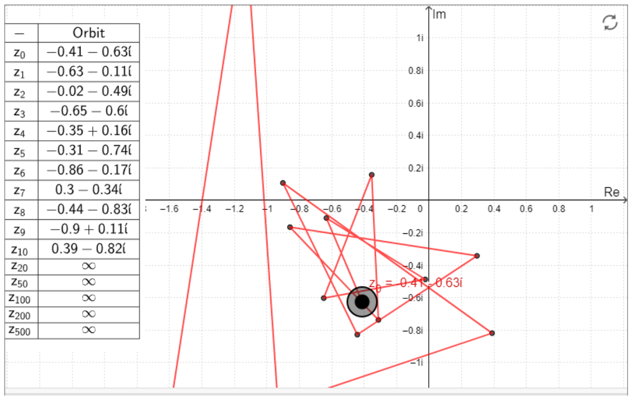
Background:

Although it has been observed in several instances throughout history, modern Chaos Theory was discovered by Edward Lorenz in 1961 by accident. While attempting to create a weather prediction model at MIT, he noticed that minimal changes at the beginning of his model would create drastically different outcomes as time went on. Almost minuscule differences in beginning states would lead to completely different weather patterns down the road. This phenomenon went on to be named Chaos Theory. The almost cliche phrase “Does the flap of a butterfly's wing in Brazil set off a tornado in Texas?” summarizes the theory well.

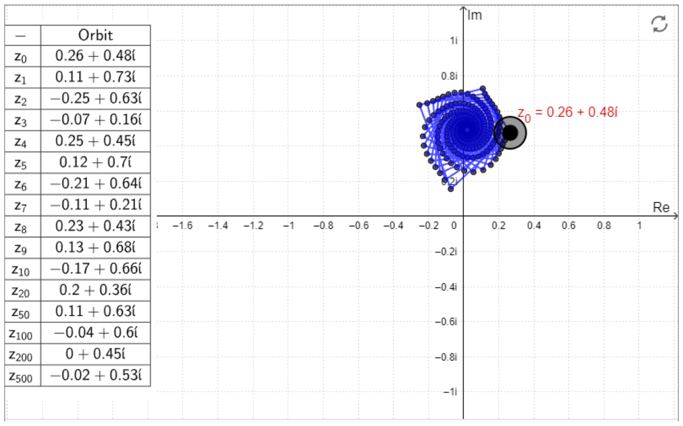
In 1980 the mathematician Benoit Mandelbrot made his discovery of chaos theory in the field of mathematics. The basic idea of this came from simple iteration on the imaginary plane. Given a point on the imaginary plane, and input it into the recursive formula:

$$Z_{n+1} = Z_n{}^2 + C.$$

Where Z is the iterating number and C is the complex number to bring it into 2 dimensions. The beauty of this comes from its chaotic nature. Some points when passed into the iteration will spin off into chaos while others will converge neatly. See the images below:

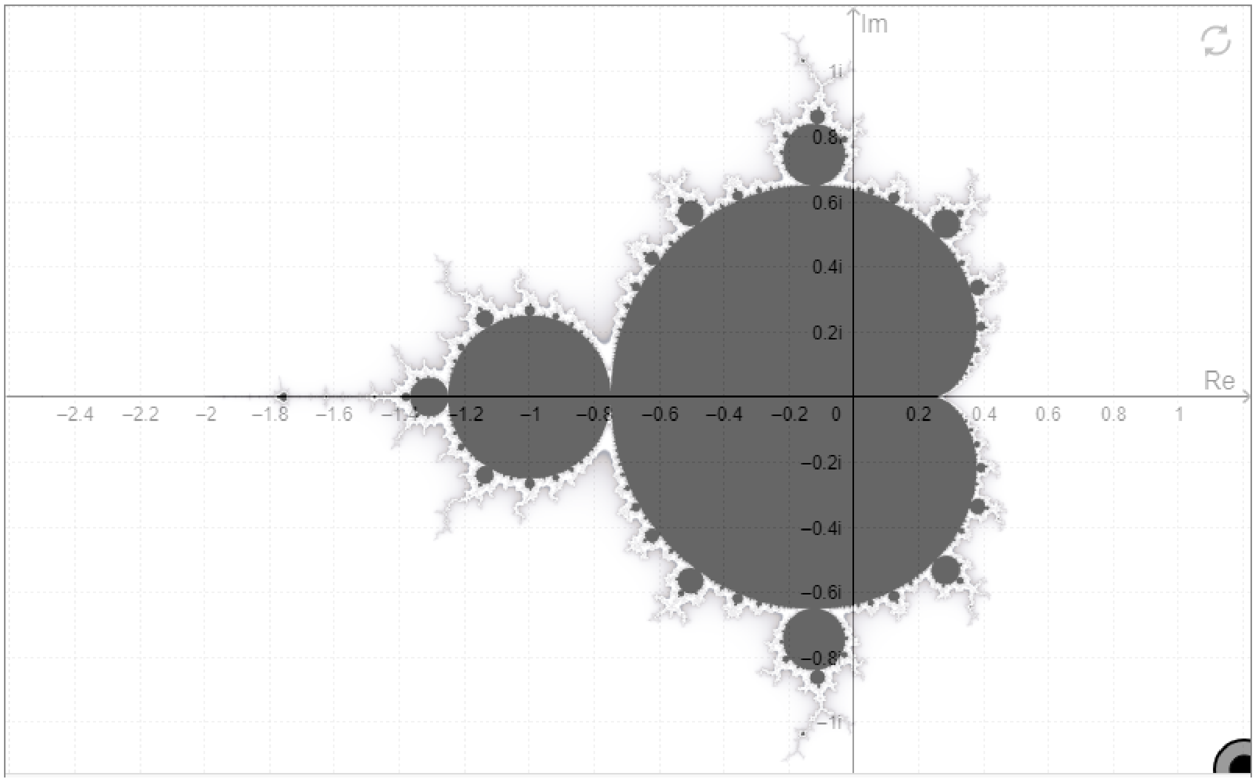


–	Orbit
z_0	$-0.41 - 0.63i$
z_1	$-0.63 - 0.11i$
z_2	$-0.02 - 0.49i$
z_3	$-0.65 - 0.6i$
z_4	$-0.35 + 0.16i$
z_5	$-0.31 - 0.74i$
z_6	$-0.86 - 0.17i$
z_7	$0.3 - 0.34i$
z_8	$-0.44 - 0.83i$
z_9	$-0.9 + 0.11i$
z_{10}	$0.39 - 0.82i$
z_{20}	∞
z_{50}	∞
z_{100}	∞
z_{200}	∞
z_{500}	∞



–	Orbit
z_0	$0.26 + 0.48i$
z_1	$0.11 + 0.73i$
z_2	$-0.25 + 0.63i$
z_3	$-0.07 + 0.16i$
z_4	$0.25 + 0.45i$
z_5	$0.12 + 0.7i$
z_6	$-0.21 + 0.64i$
z_7	$-0.11 + 0.21i$
z_8	$0.23 + 0.43i$
z_9	$0.13 + 0.68i$
z_{10}	$-0.17 + 0.66i$
z_{20}	$0.2 + 0.36i$
z_{50}	$0.11 + 0.63i$
z_{100}	$-0.04 + 0.6i$
z_{200}	$0 + 0.45i$
z_{500}	$-0.02 + 0.53i$

The graph on the left shows an unstable point and the graph on the right shows a stable point. If a point converges within the given amount of iterations it is classified into the mandelbrot set. (There is a great explanation from Numberphile on youtube) As you generate more and more points within these bounds and color them based on stability vs instability you get an image as shown:



Note: The gray areas are stable points and are part of the Mandelbrot set. An interactive aid can be found here and here.

The beautiful thing about the Mandelbrot is its infinite complexity. If you were to zoom in and keep calculating points, you would find that there is no true boundary between stable and unstable. You would also find miniature Mandelbrots within the Mandelbrot. To perfectly capture the full beauty of the Mandelbrot you would need an infinite amount of points and an infinite amount of iterations to find its "true" boundary.

How this could theoretically apply to real world applications is shown below. In this example, X and Y would represent the initial atmospheric conditions like temperature, humidity, or pressure. z1 through z25 would represent a time interval such as 1 day. Each interval is a prediction of the weather at that time. The stability may represent whether or not the prediction is within a certain range of confidence. Example:

Temp(F) Humidity(%)	Mon	Tue	Wed	Thur	Fri	Sat	Sun	stability(0/1)
94.0,39.0	47.8	81.2	…	…	…	…	13.5	1
97.0,51.0	32.4	15.0	…	…	…	…	47.5	1
53.0,88.0	68.0	44.9	…	…	…	…	9.2	0

Of course, the imaginary plane (z1-25) does not have to be as linear as time. In the case of analyzing infection spread, rather than using time as the interval for z1 through z25, we could use each interval to account for a change such as a new policy (social distancing), a new development in the disease, etc. In this scenario, stability might represent whether or not the disease is considered "controlled" depending on how you define it.

Hypothesis:

Considering the unique nature of the Mandelbrot set and its intricate relationship with chaos theory, we hypothesize that by introducing a set level of randomness into the data generation process and focusing on the border regions of the Mandelbrot set, our neural network model will be able to identify and predict stable points with a higher level of accuracy, using fewer number of iterations. We anticipate that this approach will allow the model to recognize patterns and variations characteristic of the chaotic behavior found in the Mandelbrot set. This is based on the assumption that the edge regions, where stable and unstable points are nearest to each other, hold critical information and patterns that the model can train on for more accurate predictions. The inherent chaotic nature of the Mandelbrot set, specifically centered around the border areas, creates an interesting testing ground for our model to train and predict chaotic data, and will hopefully be able to mirror real-world scenarios dealing with chaotic systems.

Data:

An interesting caveat to this project is how we sourced our data. Our starting data is a CSV file with a list of points on the imaginary plane. Due to this, we were able to generate as many points as we needed. This allowed us to go back and make changes to our original data as our project evolved.

As we stated in our proposal, we were able to successfully generate 1 gigabyte of data with over 25 million (x,y) coordinate points. For each point, we recorded the first 25 iterations and the stability of the 25th iteration as well as the stability of the 100th iteration for our network to train on. After generating 25 iterations for each point, we ended up with over 25 gigabytes of data to use for our model.
Our initial data followed this structure:

X	Y
X	Y
X	Y

The final dataset we used to train our model followed this structure:

Initial x,y	z1_x	z1_y	z2_x	z2_y	…	z25_x	z25_y	stability(0/1) 25th iteration	stability(0/1) 100th iteration
Initial x,y	z1_x	z1_y	z2_x	z2_y	…	z25_x	z25_y	stability(0/1) 25th iteration	stability(0/1) 100th iteration
Initial x,y	z1_x	z1_y	z2_x	z2_y	…	z25_x	z25_y	stability(0/1) 25th iteration	stability(0/1) 100th iteration
Initial x,y	z1_x	z1_y	z2_x	z2_y	…	z25_x	z25_y	stability(0/1) 25th iteration	stability(0/1) 100th iteration

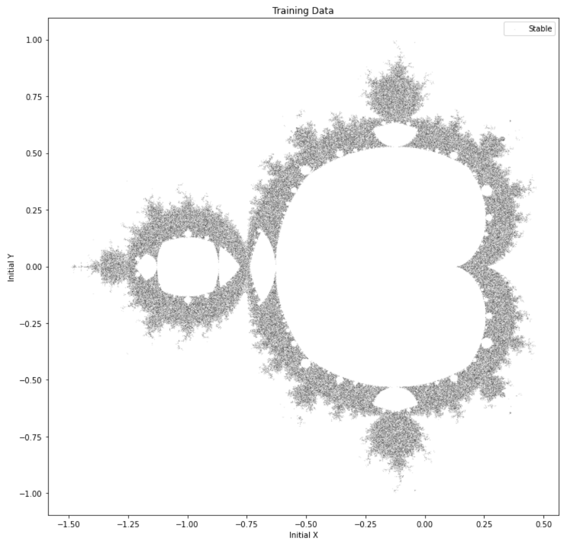
Our final data set was shaped “# of rows x 52 columns”

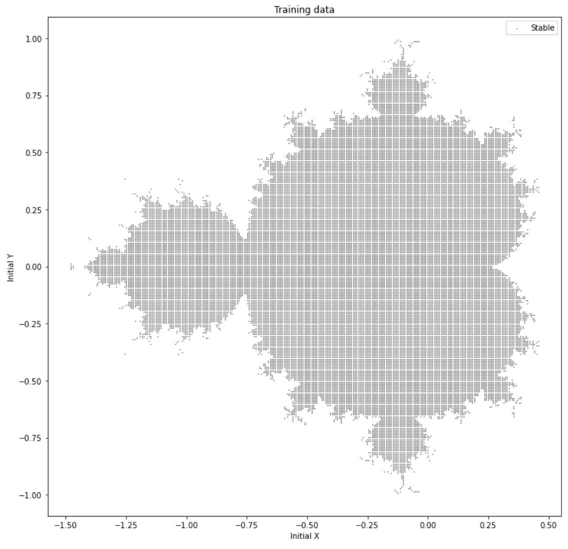
Strategy:

Data Generation/Sorting:

A major concern with generating our own data was making the data too uniform thereby making it too easy for our model to achieve a high accuracy. The standard way of generating Mandelbrot points is to create a grid of points over the complex plane. But after consideration we decided that we wanted to add a bit of noise into the data. To create a more natural set of data points, we opted to go the route of randomly picking a point on the imaginary plane within the coordinate boundary of x(-1.5, .5), y(-1,1), iterating that point for stability. After defining a distance *delta*, we would check all 4 neighbors *delta* distance away for their stability. If the point had a neighbor of opposite stability, then we would keep that point. The neighboring points were also tested to better determine if that initial random point is in the mandelbrot set or not.

This random sampling continued until there were 10 million points in the mandelbrot set, 10 million points not in the mandelbrot set, and 15 thousand random points to help introduce more noise into the dataset. This method resulted in an extremely computationally expensive operation. Our initial border *delta* was .01, this however took around 14 hours to generate 20 million points. Therefore, we opted to increase the *delta* to .06, which cut down the total processing time to around 9 hours for the initial data generation. The edge model and grid model can be seen in the figures below.





In order to generate our initial dataset of x,y coordinates on the complex plane, we wrote a python script that takes advantage of the PySpark API to accomplish distributed computing. Using the PySpark framework allowed us to take advantage of the clusters already configured on the CS machines as well as the Hadoop File System to temporarily store the generated data. Due to the large file sizes we generated, it was necessary to configure the PySpark driver and executor to use 2 gigabytes of memory. It also proved necessary to break up the data generation into three separate events. During the development phase of this project, we found that the maximum number of points we could generate and write to the Hadoop File System in a single instance was around 10 million points. To overcome this obstacle we simply broke up the execution into three separate parts, the first two generating 10 million points and the last generating 7 million. Each execution was written to a different Hadoop File directory to avoid running out of memory. This was a successful strategy to generate the total one gigabyte of data we needed. After the initial points were generated we wrote an additional script to pull the data to the local machine's file system and then combine all of the generated CSV files into a single combined CSV file. From this point we had our initial data that we then took each point and iterated it using the $Z_{n+1} = Z_n^2 + C$ iteration. During this iteration process we recorded the first 25 iterations, appending the resulting stability of the 25th iteration and then the 100th iteration at the end of each row. Again, due to the large amount of data this would generate we decided to use the same PySpark methodology. The combined CSV file that contained the initial points was uploaded back into the Hadoop File System so that the iteration python script could use it for additional data generation. The PySpark driver and executor used two gigabytes of data to handle the large amount of initial data. Curiously, the program was able to successfully write over 25 gigabytes of data back into the Hadoop File System without the need to break up the job into different file writing executions. Immediately after attempting this iteration process we ran into an error that the $\boldsymbol{Z_n^2}$ value was too large and caused a memory overflow error. To fix this we determined that if a point reaches instability we would mark the rest of the iterations as a 0, both to avoid memory overflow errors and to maintain uniformity in our CSV file. After this job was completed the resulting CSV files were then downloaded and combined into a single CSV file. Prior to using the data for the model we discovered the data needed to be reformatted to be easier to work with. Rather than regenerating the 25 gigabytes of data, we wrote a preprocessing script that iterated through each row of data and formatted it to our needs.

Models:

After we iterated our data we started work on our model. For this application we decided to use a 1D CNN. We chose to use this type of model because of its general success with time series data. Our data can be classified as time series data because of its iterative nature. To transform our data into data that our model can recognize we reshaped our data frame into a tensor. We reshaped the iteration data and labels as follows:

Model Input	**Iteration Data**							
		Initial point 1		Initial point 2		Initial point 3		…
Node 1	←	z1_x	z1_y	z1_x	z1_y	z1_x	z1_y	…
Node 2	←	z2_x	z2_y	z2_x	z2_y	z2_x	z2_y	…
Node 3	←	z3_x	z3_y	z3_x	z3_y	z3_x	z3_y	…
…	←	…	…	…	…	…	…	…
Node 25	←	z25_x	z25_y	z25_x	z25_y	z25_x	z25_y	…

Labels			
Stability at 100 Iterations (0/1) for point 1	Stability at 100 Iterations (0/1) for point 2	Stability at 100 Iterations (0/1) for point 3	…

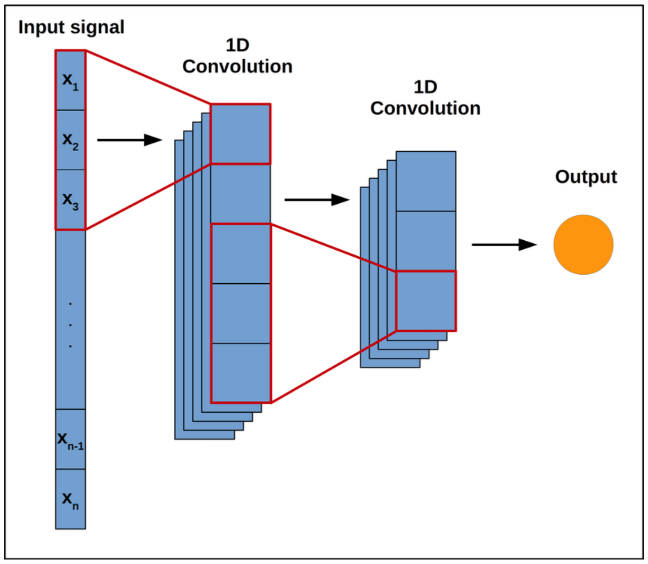
Our model has a column of 25 input nodes for each iteration, and at each node there are two channels for each iteration's x,y coordinates. A general visualization of how our model was designed is shown here on the right. The final architecture we used is as follows on the left:

Input: 25 Iterations with 2 input channels each

Conv_layer_1: 2, 16
Conv_layer_2: 16, 32
Conv_layer_3: 32, 64
Conv_layer_4: 64, 128
Conv_layer_5: 128, 256
Conv_layer_6: 256, 512

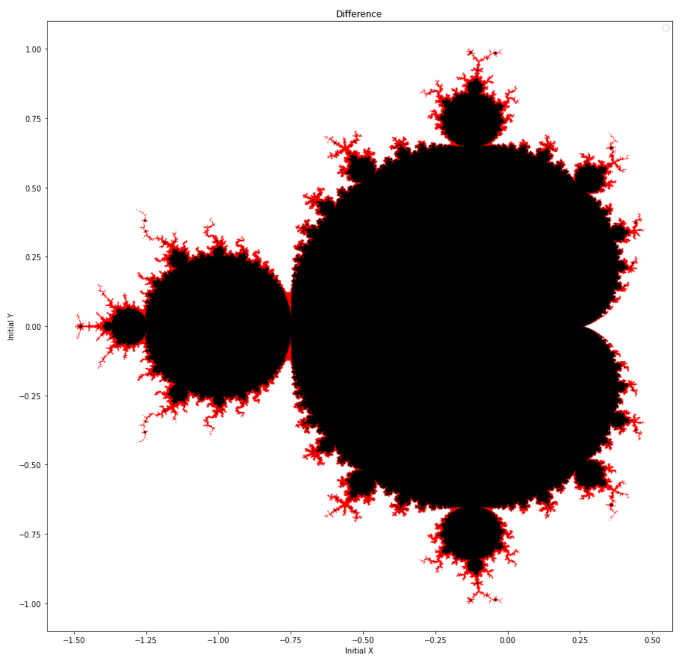
FC_layer_1: 512, 256
FC_layer_2: 256, 128
FC_layer_3: 128, 1

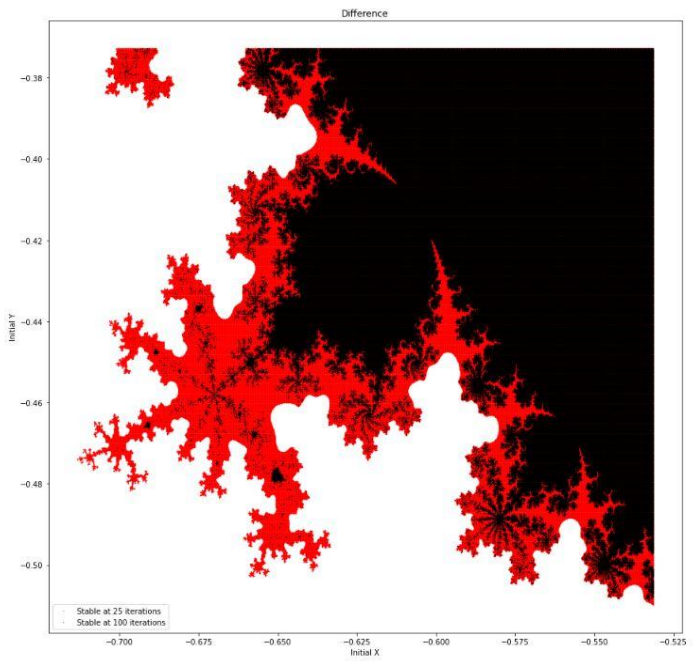
Output Node: 1 (0/1 for stability)



To test our model we generated new data using the grid method. We chose to create two testing data sets, the first one is the full mandelbrot and the second is a zoomed in portion where the difference between 25 and 100 iteration stability can be seen more clearly.

To visualize the data and and how our model performed we plotted the data on-top of eachother. The figures below show our testing data at 25 iteration stability in red and 100 iteration stability in black. For clarity, these plots are only stable points.





To test our models accuracy we established a baseline accuracy using the stability at 25 iterations vs the stability at 100 iterations. For example: if our baseline accuracy was 85% that means that 15% of the points were incorrectly classified at 25 iterations. Using this metric as a baseline we can judge if our model was able to learn insightful information about stability from the training data. For example: If our baseline accuracy was 85% and our models accuracy was 89% we could say that our model performed better than just calculating stability at 25 iterations. To make this easier to understand we had a final score that's the difference between baseline and predicted accuracy. For example: If our baseline was 85% and our model accuracy was 89% the evaluation score would be 4.0. The higher this score is, the more our model was able to learn from our data.

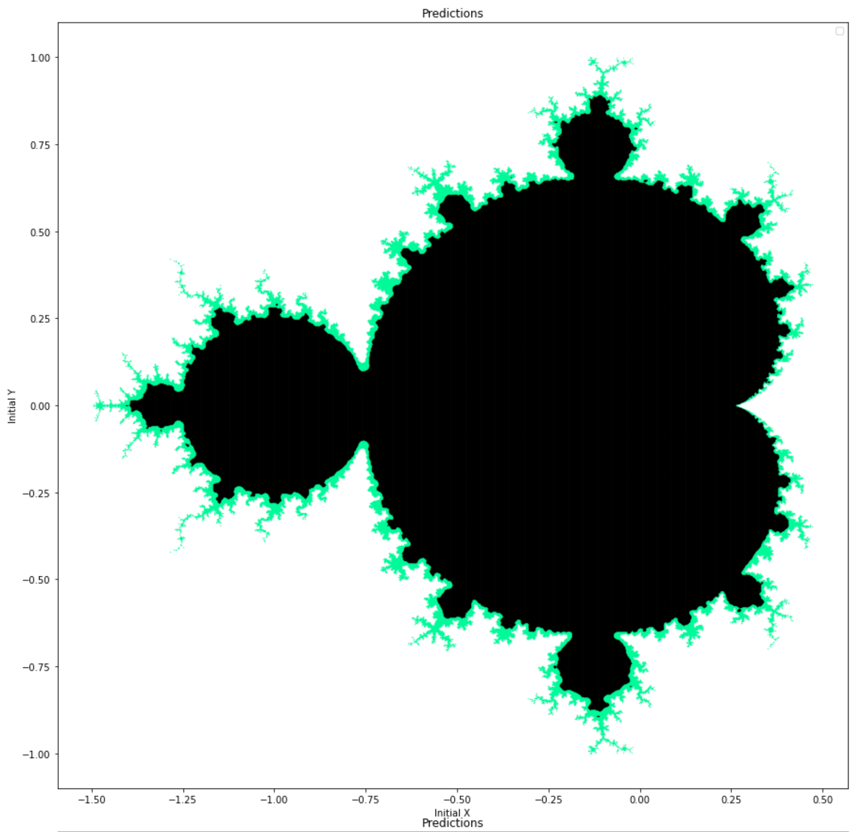
An interesting note on the training of our models. Generally when training models, lower learning rate and a large amount of epochs will tend to train better models. But we found the opposite when training on this data. When training we found that a 1 or 2 epochs with a large learning rate gave us better accuracy. We think this is happening because our gradient has a lot of local minimums. This would explain why the lower learning rate and more epochs would get stuck, but one large shot down the gradient tended to be better.

To test our hypothesis we trained identical models on the edge data and the regular grid data. Then we compared the scores to see which model was able to perform better.

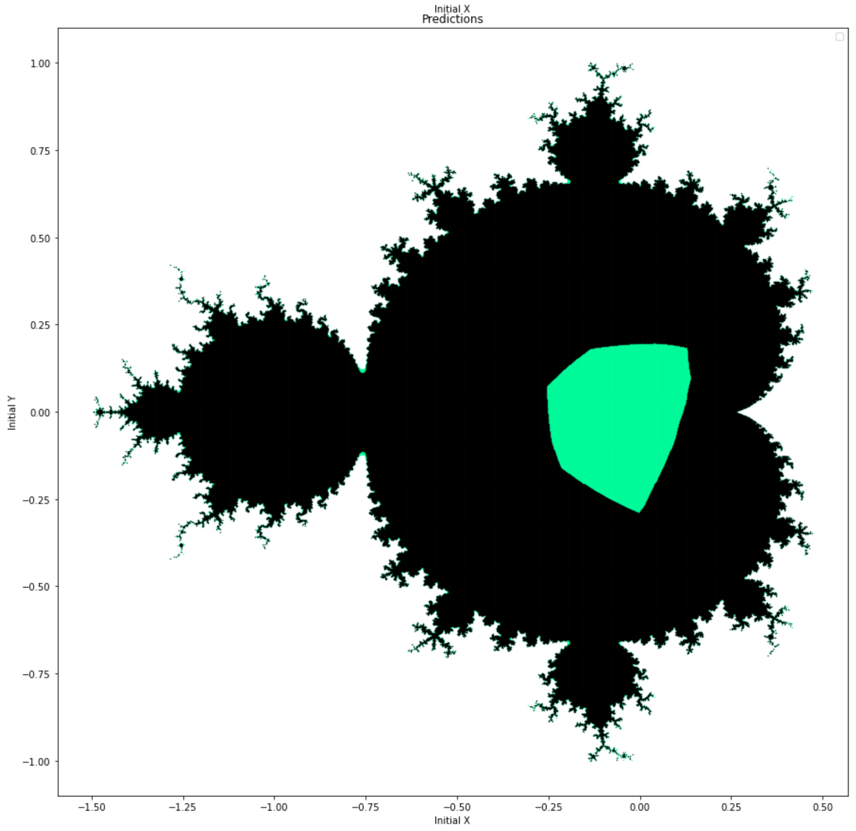
Results:

The following images and scores are the results of our testing. The first two images are the full Mandelbrot predictions and the last two are the zoomed in portion. The points in black are points that our model predicted to be stable, the points in green are points that are stable at 25 iterations. In simple terms, the areas in green are points that our model was able to predict past 25 iterations. So generally the more green on an image means the model performed better.

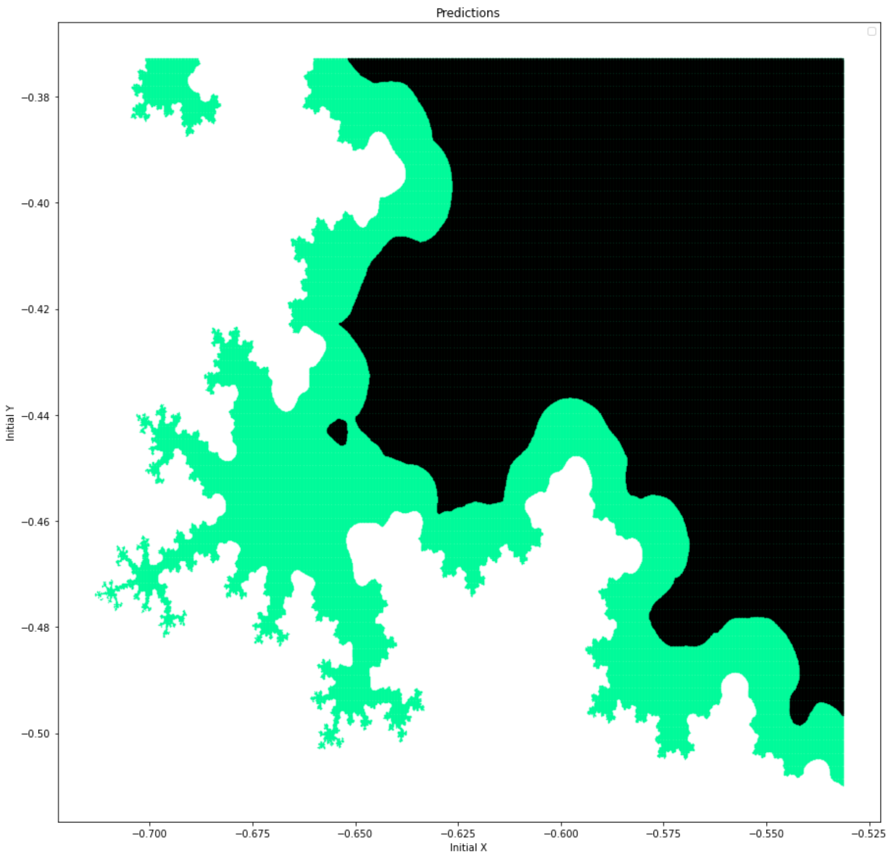
Grid Accuracy Score: 2.5



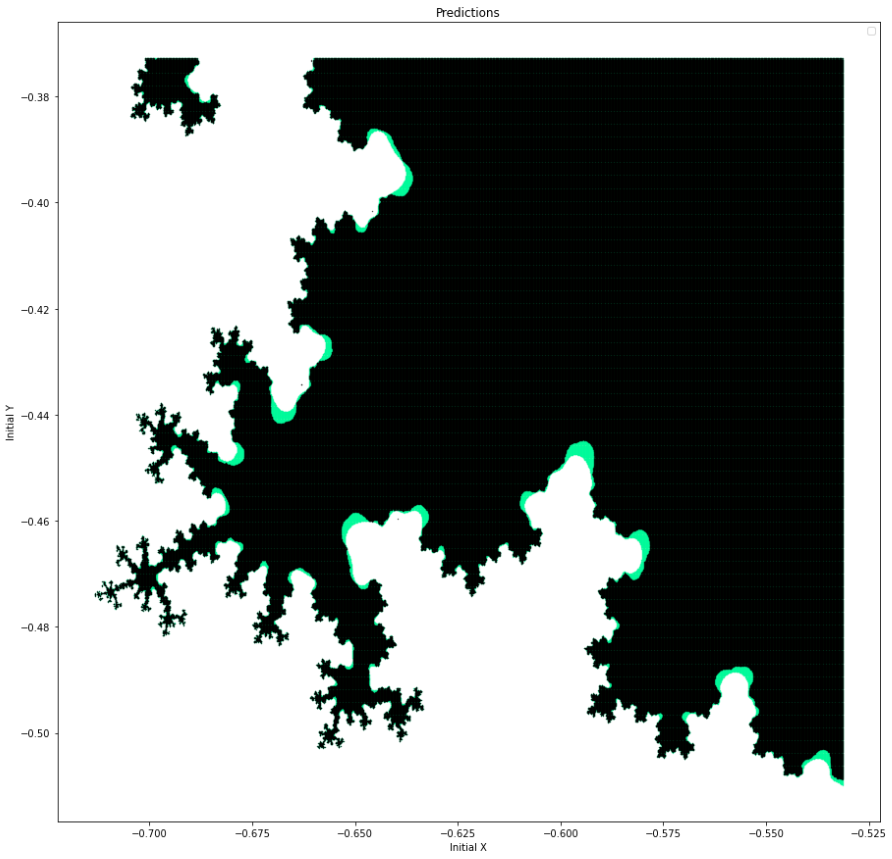
Edge Accuracy Score: -10.3



Grid Accuracy Score: 10.76



Edge Accuracy Score: 0.98



Discussion and Analysis:

After getting our results back we were surprised. We originally thought that the edge data would have performed better. However, our model performed way better on the grid data rather than the edge data. This was not what we were expecting.

When testing on the whole Mandelbrot, our edge data model had trouble classifying points around the 0,0 origin. We tried multiple times to get the model to recognize these points but to no avail. We have an idea on why that hole appears. We think that our edge data is too focused on the edge, leaving that area under represented in the dataset. All the points in this area are the most stable points in the mandelbrot. So these points being under-represented in our dataset skewed the model's idea of what stability is. This observation was again observed in the zoomed test.

Looking at how both models fared on the zoomed data, the model trained on the grid data again performed better. This interested us because we at least expected the edge data model to perform better along the edge. But this was not the case. The model trained on the grid data performed better in both test data sets. This furthered our idea that the edge data skewed the models idea of stability.

Interestingly enough, the grid model was able to predict stability at 100 iterations with 99% accuracy. This means that given the first 25 iterations, our model was able to predict stability 75 iterations out. We were very surprised by this. We expected our model to confidently predict stability at 10-20 more iterations. So 99% accuracy at 75 more iterations was huge. We were very happy with this.

Bibliography:

Simple 1D Convolutional Neural Network (CNN) Architecture with Two ..., www.researchgate.net/figure/Simple-1D-convolutional-neural-network-CNN-architecture-with-two-convolutional-layers_fig1_344229502. Accessed 26 Oct. 2023.

Ford, Dominic. “Online Mandelbrot.” *Online Mandelbrot Set Plotter - Sciencedemos.Org.Uk*, sciencedemos.org.uk/mandelbrot.php. Accessed 26 Oct. 2023.

Campuzano, Juan Carlos Ponce. “The Mandelbrot Set.” *Complex Analysis*, 26 Jan. 2019, complex-analysis.com/content/mandelbrot_set.html.

YouTube, YouTube, 18 Apr. 2019, https://www.youtube.com/watch?v=FFftmWSzgmk. Accessed 26 Oct. 2023.

Schmidt, Lasse. “Time Series Classification with Convolutions.” *Medium*, Analytics Vidhya, 2 May 2021, medium.com/analytics-vidhya/time-series-classification-with-convolutions-ed5cb33b1e3b.

Biswas, Hena Rani, Md Maruf Hasan, and Shujit Kumar Bala. "Chaos theory and its applications in our real life." *Barishal University Journal Part* 1.5 (2018): 123-140.