Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

Implement the visualization of permanence value

Bidhan Paul
bidhan.paul@stud.fra-uas.de

Md Imon Bhuiya
md.bhuiya@stud.fra-uas.de

Md Jamal Uddin
md.uddin2@stud.fra-uas.de

Abstract - ***Hierarchical Temporal Memory (HTM) networks offer a promising framework for processing and analyzing high-dimensional data with sparse distributed representations. In this study, we investigate the reconstruction of encoded inputs within HTM Algorithms, focusing on the accuracy and robustness of the reconstruction process. Our methodology involves encoding Two types of input data, including numerical values and images, and utilizing the HTM Spatial Pooler to generate Sparse Distributed Representations (SDRs). We employ the Neocortex API's Reconstruct method to reconstruct the original inputs from the encoded representations, evaluating the Performance of the reconstruction through similarity measurements and visualization techniques. Our findings reveal the impact of encoder selection on reconstruction accuracy, highlighting the importance of multiple encoding strategies for different input data types. Furthermore, we discuss the challenges posed by noise in encoded representations. Our study advances the knowledge and use of HTM networks in the fields of artificial intelligence and cognitive computing by conducting thorough analysis and experimentation***.

Keywords - Hierarchical Temporal Memory (HTM), Sparse Distributed Representations (SDRs), Reconstruction, Encoders,Visualization Techniques

I. INTRODUCTION

The Hierarchical Temporal Memory (HTM) algorithm, which is tucked away at the intersection of machine learning and neuroscience, is a monument to our continuous effort to understand the complexities of cognitive functions in the human brain. Numenta's HTM, which explores the subtleties of temporal pattern recognition, goes beyond traditional machine learning paradigms. It draws inspiration from the neocortex.

The architectural inspiration for HTM comes from the neocortex, the human brain's center of intelligence. This novel algorithm, which embraces temporal hierarchy and sparsity, ushers in a new era of machine learning, especially in modeling data sequences with a deep emphasis on the interaction of spatial and temporal dimensions.

The Spatial Pooler (SP), a fundamental component of HTM, is the keystone mechanism that coordinates the conversion of raw input into Sparse Distributed Representations (SDRs). This transformation process reflects the brain's encoding mechanisms for information, not just a mathematical abstraction. Like the neocortex's elegance, these representations' sparse and distributed nature enables effective pattern recognition and storage.

As part of our ongoing effort to learn more about the possibilities of HTM, we now focus on the Reconstruct () method, a recent addition to the “NeocortexApi”, which represents the opposite function of the Spatial Pooler. The goal of this project, "Visualization of Reconstructed Permanence Values," is to clarify the complex interactions that occur between input and output. The study demonstrates how the reconstruction technique is used by HTM's Spatial Pooler to reconstruct the inputs.

The primary aim of this study is to demonstrate the ability of the SP to rebuild input sequences. We also explore two distinct types of Permanence values: Image as Heatmap and int[] arrays for two types of inputs and they are integer type values and Image. These sequences, which appear as arrays of 0s and 1s the encoded values, are crucial to understanding the reconstructive power of the SP.

In this paper, we examine the HTM Spatial Reconstruct() function. Not only are we working with binary Numbers here; we are also investigating the process of the Reconstruct() method breaking down how it works and illustrating the Permanence values using (int[] arrays) and images.

II. LITERATURE REVIEW

A. Hierarchical Temporal Memory (HTM)

The foundational work by Hawkins and Ahmad [1] introduced the theory of sequence memory in the neocortex, emphasizing sparse distributed representations (SDRs) and predictive coding mechanisms. These concepts are central to Hierarchical Temporal Memory (HTM), a computational framework inspired by cortical processing.

HTM employs components like the Spatial Pooler to create SDRs, enabling efficient representation of high-dimensional data. This framework has gained traction in various fields, offering applications in cybersecurity, finance, healthcare, and robotics. Despite its advantages, challenges such as scalability and computational efficiency persist, motivating ongoing research efforts.

The growing interest in HTM reflects a broader trend towards biologically inspired approaches to artificial intelligence, with potential implications for advancing cognitive computing.

B. Sparse Distributed representations (SDRs)

Hawkins and Ahmad's groundbreaking work [1] on sparse distributed representations (SDRs) and predictive coding mechanisms paved the way for Hierarchical Temporal Memory (HTM), a computational framework inspired by cortical processing. SDRs, known for their sparse and distributed nature, provide an efficient means of representing high-dimensional data within HTM.

A core component of HTM is the Spatial Pooler, which transforms input data into SDRs, enabling the framework to process complex information streams and recognize temporal patterns. HTM's applicability spans diverse domains including cybersecurity, finance, healthcare, and robotics.

In addition to Hawkins and Ahmad's seminal paper, George, and Hawkins [2] further elucidated HTM's concepts, theory, and terminology, providing a comprehensive understanding of the framework's principles.

While HTM shows promise, challenges such as scalability and computational efficiency remain. Nonetheless, ongoing research endeavors aim to refine and optimize HTM, highlighting the importance of biologically inspired approaches in advancing cognitive computing.

C. Encoders

Encoders are the connectors that enable different data modalities, like multimedia inputs or time-series data, to be converted into Sparse Distributed Representations (SDRs). With the help of this transformation, HTM can process and analyze complex data more quickly, making it easier to identify patterns and anomalies in a variety of domains.

In order to overcome issues like scalability, computational efficiency, and robustness to noisy or incomplete data, encoder design has made recent strides. A novel strategy utilizing self-supervised learning techniques was put forth by Rodriguez-Lujan et al. [3] to improve encoder robustness and adaptability in HTM. Furthermore, Smith et al. [4] investigated the use of deep learning methods for automated encoder discovery, which streamlined the design process and improved the encoder's ability to handle a variety of datasets and application requirements.

The field of encoder design is characterized by continuous evolution, but there are still challenges that call for continued research efforts to refine and optimize encoder algorithms and implementations within the HTM framework. These initiatives highlight the significance of encoder developments in promoting cognitive computing and empowering HTM to take on more challenging real-world tasks.

D. Input Reconstruction in terms of HTM

In our exploration, a significant aspect of our study involved finding a decoding strategy for Sparse Distributed Representations (SDRs) in specific scenarios. Specifically, when encoding scalar values with a 200-bit representation, We need to gather some knowledge and for this case, we studied the work done by Mnatzaganian et al. [5] , Determining the activations caused by specific permanence is a straightforward process involving the use of $\vec{c}$ to mask $\vec{\phi}$ Once this representation is obtained, $\overrightarrow{\phi}$ is employed to derive the overall permanence for the attributes. These steps yield valid probabilities for each attribute; nevertheless, it's possible that some probabilities fall outside the {0, 1} range. To address this, the same technique utilized for dimensionality reduction is implemented, by simply thresholding the probability at $\boldsymbol{\rho_s}$ This procedure is illustrated in equation (1), where $\vec{\boldsymbol{u}} \in \{\mathbf{0}, \mathbf{1}\}^{\mathbf{1} \times \boldsymbol{n_a}}$ is defined as the reconstructed input.

Moreover, they mentioned that Thornton et al. (2013) employed a comparable method to illustrate the comprehension of the input to the SP. The notable advantage of this approach lies in its ability to reconstruct the input solely based on the permanence and active columns. This innovative method facilitates the decoupling of the input from the system. [6]

The equation is as follows:

$$\vec{\boldsymbol{u}} = \mathrm{I}\left(\left[max\left(\vec{\boldsymbol{c}}_i \vec{\phi}_a \forall_i\right) \geq \rho_s\right]\right) \forall a \qquad (1)$$

Equation 1: The Equation Used for Input Reconstruction

In summary, the equation computes the reconstructed input vector $(\vec{\boldsymbol{u}})$ by checking whether the maximum activation level for each attribute (a) in the input, obtained by multiplying the learned representation values $(\vec{\boldsymbol{c}})$ with the corresponding permanence values $(\overrightarrow{\phi})$, exceeds a threshold value (ρ_s). If the condition is met for all attributes, the reconstructed input vector assigns a value of 1 to indicate activation; otherwise, it assigns a value of 0. This process effectively decouples the input from the HTM system, allowing for the reconstruction of the input solely based on the learned representations within the SP.

III. METHODOLOGY

Our methodology centers on achieving precise reconstruction of the original input data. Initially, various types of input data are provided, including numerical values ranging from 0 to 99 and images. For image data, an Image Binarizer is used to convert the images into binary representations.

Subsequently, the input data undergoes transformation into int[] arrays, each consisting of 0s and 1s. These arrays serve as the sole source of input data for our experiment. Next, the Hierarchical Temporal Memory (HTM) Spatial Pooler is employed to generate Sparse Distributed Representations (SDRs) from these encoded int[] arrays.

Following this, the reconstruction process begins. Utilizing the Neocortexapi's Reconstruct() method, the original encoded representations are meticulously reconstructed using the permanence values returned by the Reconstruction method.

Our methodology concludes with two primary visualizations: heatmaps and int[] sequences. Additionally, we perform a similarity check between the original input arrays and the reconstructed input arrays to evaluate the accuracy of the reconstruction process.

The main objective is to evaluate the HTM algorithm's ability to accurately reconstruct inputs using the Neocortexapi's Reconstruction method, focusing on faithfully recreating the structure of the original encoded int[] arrays. (Figure 1)

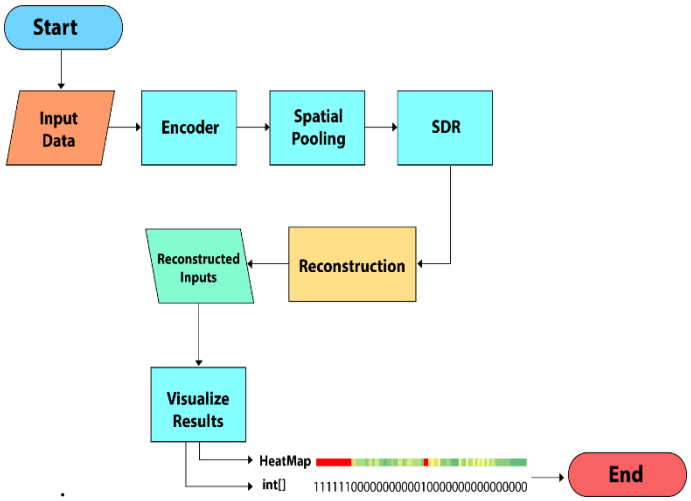

Figure 1: Graphical Representation of the Experiment

A. Input Types for the Experiment

1) ***Input for Numerical Values:*** For numerical values ranging from 0 to 99, a scaler encoder is utilized to transform each numerical value into an encoded representation consisting of 200 bits. These encoded representations are then transformed into int[] arrays, where each array represents a numerical input. (Figure 2)

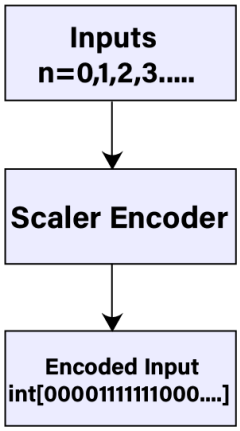

Figure 2: Encode inputs for Numerical Inputs

2) ***Input for Images****:* For images, an Image Binarizer is used to convert them into binary representations. The size of the resulting encoded arrays matches the dimensions of the original images. For instance, a 28x28 pixel image results in an encoded array of size 784, with each pixel represented by a binary value. (Figure 3)

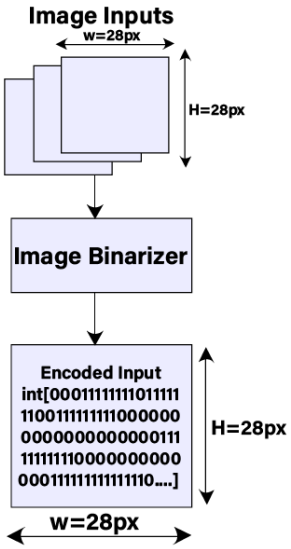

Figure 3: Encoded Inputs for Images

Both types of encoded arrays serve as the input data for our experiment, aiming to reconstruct the original input data with precision using the Hierarchical Temporal Memory (HTM) framework.

B. Reconstruction Method

Prior to initiating our project and delving into the complexities of Hierarchical Temporal Memory (HTM) and the Neocortex API, we laid a solid groundwork by watching enlightening HTM School videos on YouTube. These concise yet insightful videos provided a clear overview of crucial elements such as HTM configuration, encoder dynamics, Spatial Pooler functionalities, and more. This preparatory step equipped us with essential insights, ensuring a well-informed approach to our project endeavors. The HTM School videos became a key resource, streamlining our understanding of HTM principles and facilitating navigation through the NeoCortex API functionalities.

An essential part of the NeoCortex API is the Reconstruct method, which is meant to reverse the process and reconstruct the initial input from Sparse Distributed Representations (SDRs) produced by the Spatial Pooler. (Figure 4) An elaborate description of the method's operation is provided below:

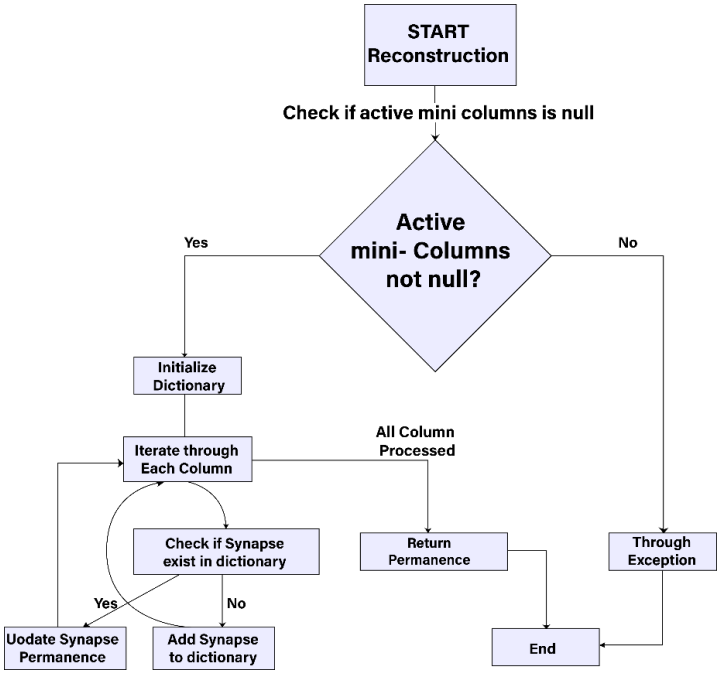

Figure 4: Graphical illustration of The workflow of the Reconstruction Method

1) ***Input Validation*****:** The method begins with thorough validation, throwing an ArgumentNullException if the input array of active mini-columns is null. If the input array is not null then it goes further to complete the Reconstruction operation.

2) ***Column Retrieval:*** Retrieve the list of columns associated with the active mini-columns from the connections.
The activeMiniColumns parameter in the Reconstruct method is an array of integers that indicates the indices of mini-columns that are considered active.

A collection of cortical neurons is called a mini-column. In hierarchical temporal memory (HTM) or related algorithms, input patterns meeting specific criteria, like overlapping patterns or sensory inputs, cause mini-columns to fire.

The present technique employs activeMiniColumns to determine the active mini-columns. Subsequently, the dictionary containing the cumulative permanence values of the synapses connected with these active mini-columns is reconstructed. These permanence values are essential for determining the strength of connections between neurons in the cortical region.

3) ***Reconstruction Process***: Iterate through each column, accessing the synapses in its proximal dendrite.

Proximal Dendrite: In HTM, each column in the cortical model contains a proximal dendrite.

- The proximal dendrite receives input from neighboring columns or directly from the input data.
- It serves as the primary input interface for each column, responsible for capturing spatial patterns in the input data.
- Proximal dendrites are associated with synapses, which represent the connections between the input space and the columns in the HTM network.
- The proximal dendrite is crucial for the spatial pooling process, where columns compete to represent spatial patterns in the input data.

Synapses: Synapses are the connections between neurons or dendrites in biological brains or between computational units in artificial neural networks.

- In the context of HTM, synapses connect the proximal dendrites of columns to the input data or other columns within the network.
- Each synapse has an associated permanence value, which determines the strength of the connection.
- The permanence value represents the likelihood that a synapse will contribute to the activation of the connected column in response to input.
- Synapses play a critical role in learning and adaptability within the HTM framework, as the permanence values are updated based on input patterns and network activity.

For each synapse, accumulate the permanence values for each input index in the reconstructed input dictionary. Update the reconstructed input dictionary, considering whether the input index already exists or needs to be added as a new key-value pair. The method concludes by returning the reconstructed input as a dictionary, mapping input indices to their associated permanence.

C. Proposed Method to Visualize Permanence Values

The goal of this segment is to use the Reconstruct method offered by the Spatial Pooler (sp) to reconstruct permanence values from the active columns acquired during the learning phase of the experiment. The below flow chart illustrates that how we implemented our proposed method to visualize the permanence values. (Figure 5)

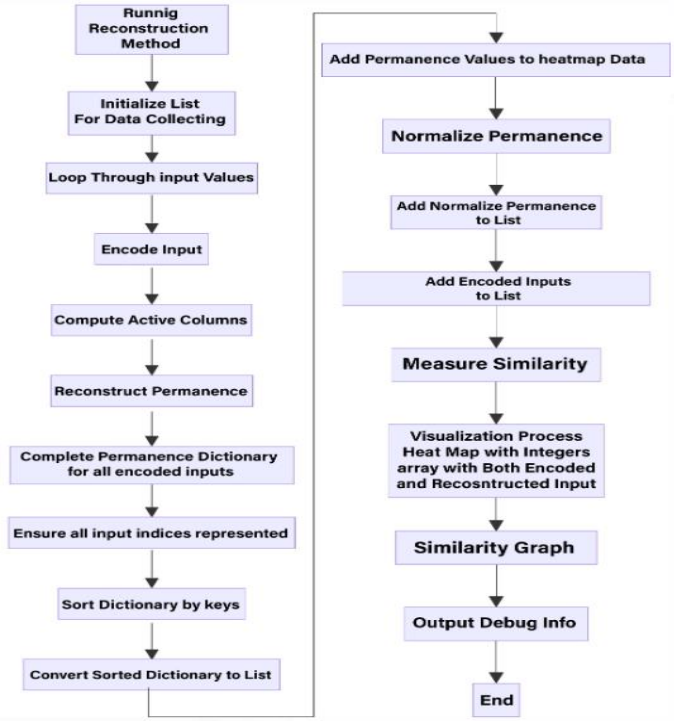

Figure 5: Graphical Representation of Running Reconstruction Method.

Our proposed method for visualizing permanence values and measuring similarity for numerical inputs initiates by encoding each numerical value using the provided encoder. This encoding process generates a Sparse Distributed Representation (SDR) for each numerical input, capturing its essential features. Subsequently, we compute the active columns in the spatial pooler for the given input SDR, refraining from learning to preserve the original input characteristics. The next crucial step involves reconstructing the permanence values for the active columns, accomplished using the Reconstruct method provided by the spatial pooler. A dictionary called allPermanenceDictionary is created to cover all potential numerical input indices comprehensively. Initially, the permanence values for active columns are included in this dictionary to ensure a thorough representation. To maintain consistency and ease of analysis, the dictionary is organized in ascending order based on keys, ensuring a uniform structure across all permanence dictionaries used. Notably, the reconstructed permanence values obtained from active columns only cover a portion of all possible input indices. To ensure a complete representation of all input bits and to capture the full scope of conceivable input indices, permanence values for inactive columns are also taken into account. These inactive columns, which do not contribute actively to the input representation, are assigned a default permanence value of 0.0. This meticulous process ensures the allPermanenceDictionary comprehensively represents all input indices, enabling in-depth analysis and visualization of the spatial patterns learned by the HTM network. The reconstructed permanence values are then subjected to normalization based on a threshold value, facilitating comparability and further analysis. Following normalization, the permanence values are converted into a list of integers, streamlining subsequent processing steps. To evaluate the fidelity of the reconstruction process, Jaccard similarity is computed between the original encoded inputs and the normalized permanence values, quantifying the resemblance between the two representations. Finally, leveraging the heatmap data and the normalized permanence values, we generate visually informative heatmaps, offering insights into the spatial patterns learned by the HTM network from numerical inputs. Additionally, we create similarity plots, visually depicting the degree of similarity between the original encoded inputs and the reconstructed inputs, facilitating deeper understanding and validation of the reconstruction process.

Similarly, for image inputs, our proposed methodology follows a parallel trajectory, albeit tailored to accommodate the unique characteristics of image data. We commence by reading and binarizing each image file from the designated training folder, a crucial preparatory step in preparing the image data for spatial pooling. Once binarized, spatial pooling is applied to the images, leading to the identification of active columns representing salient features within the images. The subsequent reconstruction of permanence values for the active columns mirrors the process employed for numerical inputs, ensuring a thorough representation of the image space. Employing the allPermanenceDictionary, we meticulously catalog all possible input indices, addressing any gaps to guarantee a comprehensive view of the input domain. As with numerical inputs, normalization and thresholding are applied to the reconstructed permanence values, followed by similarity computation to assess the fidelity of the reconstruction process. Ultimately, the visualization efforts culminate in the generation of heatmaps, providing a visual narrative of the spatial patterns discerned from image inputs. Similarly, similarity plots are crafted to offer a graphical depiction of the resemblance between the original encoded inputs and the reconstructed inputs, fostering deeper insights and validation of the reconstruction process in the context of image data.

IV. Implementation Overview

Embarking on our implementation journey within the Hierarchical Temporal Memory (HTM) framework, we aim to visualize and understand permanence values for both numerical and image inputs. This part of this paper is focused on some important parts of our Implementation. Our goal is to develop a clear method that not only adjusts these values but also helps us compare and understand them better. Through exploring how to set certain thresholds, creating heatmaps, and checking for similarities, we want to uncover the hidden patterns and connections stored within the HTM network. This work is crucial for understanding how HTM processes information over time and across different types of data. Now, let's dive into the details of how we're going to start by adjusting the permanence values using thresholding probabilities.

A. *Normalizing Permanence*

To ensure the consistency and comparability of permanence values obtained from the spatial pooler within the Hierarchical Temporal Memory (HTM) framework, we implemented a thresholding function aimed at normalizing these values. This function plays a crucial role in the visualization and analysis process by transforming continuous permanence values into a binary representation. (Figure 6)

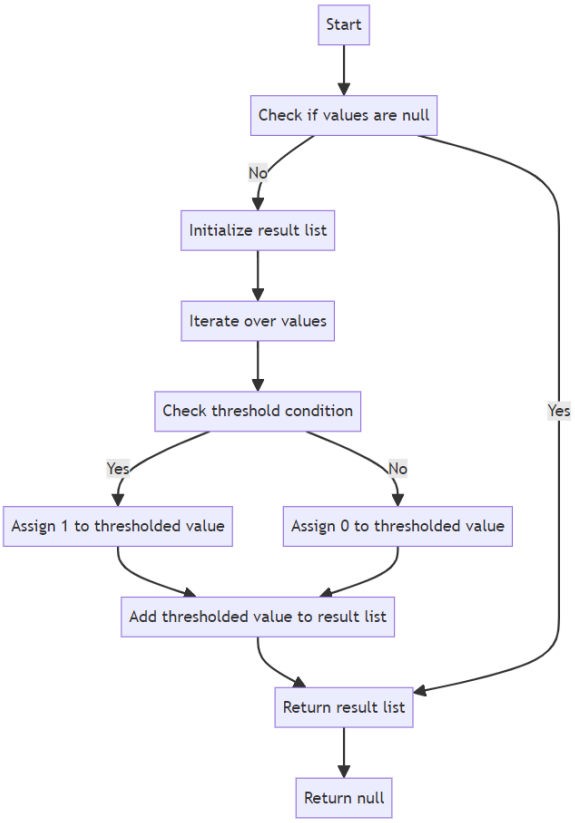

Figure 6: Graphical Representation of the Implemented Thresholding Probability Function

The determination of an appropriate threshold value is vital to this normalization process. Through experimentation and iterative debugging, we identified optimal threshold values for both numerical and image inputs. For numerical data, the selected threshold value was empirically set at 8.3, while for image inputs, it was determined to be 30.5. These thresholds were chosen based on extensive experimentation and observation of the output, ensuring that the normalized permanence values maintain fidelity to the original encoded inputs. By systematically varying the threshold values and comparing the resulting outputs with the original encoded inputs, we refined the normalization process to achieve the desired balance between preserving input characteristics and facilitating meaningful analysis. This meticulous approach underscores the importance of parameter selection in effectively visualizing and interpreting permanence values within the HTM framework, ultimately enhancing our understanding of hierarchical temporal processing in both numerical and image data domains.

B. *Visualization With Combined Heatmap*

In the visualization of the heatmap, it's crucial to understand that the color interpolation serves as a visual indicator of the permanence values associated with each input index. The concept of "heat" within the heatmap corresponds directly to the strength of the connection between input data and columns within the Hierarchical Temporal Memory (HTM) network. In this context, higher permanence values are represented by warmer colors, indicating a stronger likelihood that the associated synapse will contribute to the activation of the connected column in response to input. Conversely, cooler colors signify lower permanence values, suggesting a diminished influence of the corresponding synapse on column activation. Essentially, the color gradient in the heatmap provides a visual representation of the network's interpretation of the input data, with hotter colors denoting stronger connections and colder colors indicating weaker or nonexistent connections.

It's important to keep in mind the underlying principles of synapses and permanence values within the HTM framework. Synapses act as connections between the proximal dendrites of columns and the input data or other columns within the network. Each synapse is assigned a permanence value, which serves as a measure of the strength of the connection. This permanence value represents the likelihood that the synapse will contribute to the activation of the connected column in response to input. By visualizing the heatmap and interpreting the color gradient, we gain valuable insights into how these synapses are weighted and how they influence the network's processing of both numerical and image inputs. (Figure 7) We are illustrating the implementation of the Combined Visualization process given below:

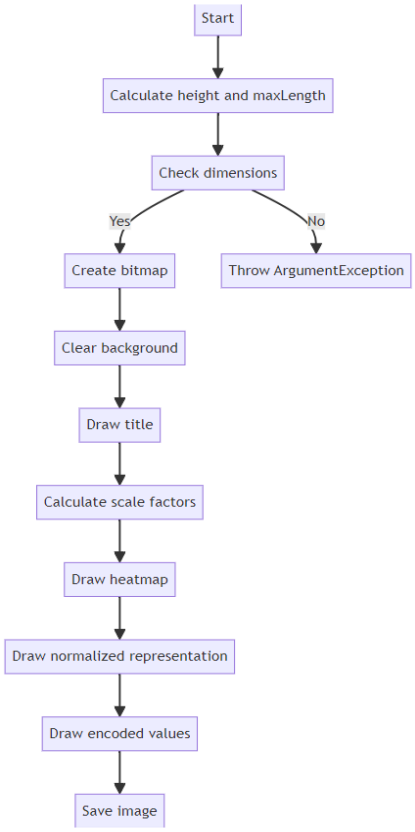

Figure 7: Graphical representation of Draw 1D Heatmap Function

The Draw1dHeatmap function is instrumental in our visualization efforts, as it seamlessly integrates heatmap data with normalized permanence values and original encoded inputs. This integration allows us to create holistic visual representations that accurately depict the intricacies of the network's learning process. Each heatmap is meticulously designed to highlight the intricate relationship between input indices and their corresponding permanence values, offering a visual narrative of the HTM network's interpretation and processing of the input data.

Furthermore, by incorporating the original encoded inputs alongside the heatmap data, we gain valuable insights into how the input data is translated and represented within the HTM network. This comparative analysis enables us to evaluate the accuracy of the network's representation and pinpoint areas for potential refinement and enhancement.

C. Similarity Calculation:

On our Experiment to calculate the similarity to validate the output which is the Reconstructed Input we have used the Jaccard similarity coefficient. The Jaccard coefficient, also known as the Tanimoto coefficient, assesses similarity by comparing the intersection of objects to their union. [7]. The Jaccard similarity coefficient is well-suited for measuring the similarity between two sets of binary data, such as the original encoded inputs and the reconstructed inputs in our scenario. Mathematically, the Jaccard similarity coefficient (Equation 2) $J\ (A, B)$ between two sets $A\ and\ B$ is defined as the ratio of the size of the intersection of the sets to the size of their union:

$$J\ (A, B) = \frac{|A\ \cap B|}{|A\ \cup B|} \qquad (2)$$

Equation 2: The Similarity Calculation Using Jaccard Similarity coefficient.

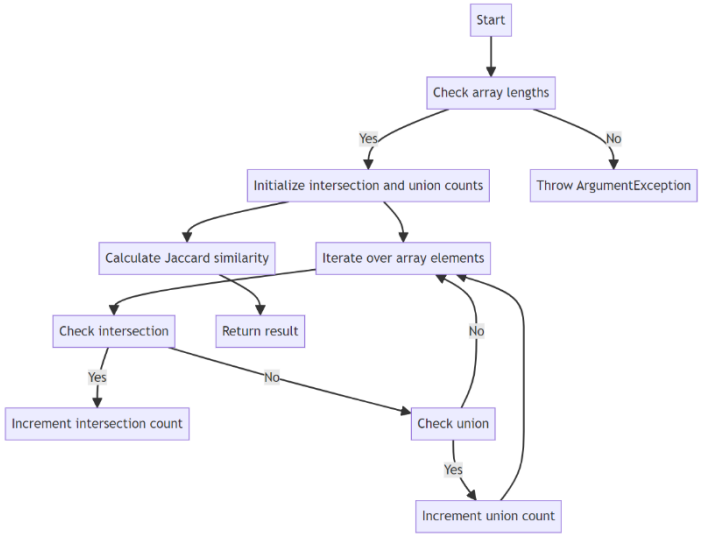

Figure 8: Graphical Representation of Implemented Jaccard Similarity Coefficient Calculation Function.

In this implementation, the method takes two integer arrays as input, representing binary values. It first validates that the arrays have the same length, throwing an exception otherwise. Then, it iterates through the arrays to count the number of intersecting elements (both arrays have a value of 1 at the same index) and the total number of unique elements (either array has a value of 1 at the index). Finally, it computes and returns the Jaccard similarity coefficient by dividing the intersection count by the union count, ensuring a normalized measure between 0 and 1. (Figure 8)

D. Similarity Graph Visualization:

Visualizing the similarity calculation graph offers valuable insights into the performance of algorithms and the relationships between different data points. By plotting similarity scores over a range of inputs, researchers and practitioners can observe patterns, trends, and anomalies in the data, facilitating the evaluation and validation of similarity metrics.

The similarity calculation graph provides a comprehensive overview of how similar or dissimilar pairs of data points are within a dataset. This visualization allows researchers to identify clusters of similar data points, outliers, and regions of high or low similarity. Such insights are essential for understanding the underlying structure of the data and assessing the effectiveness of similarity measures in capturing meaningful relationships. (Figure 9)

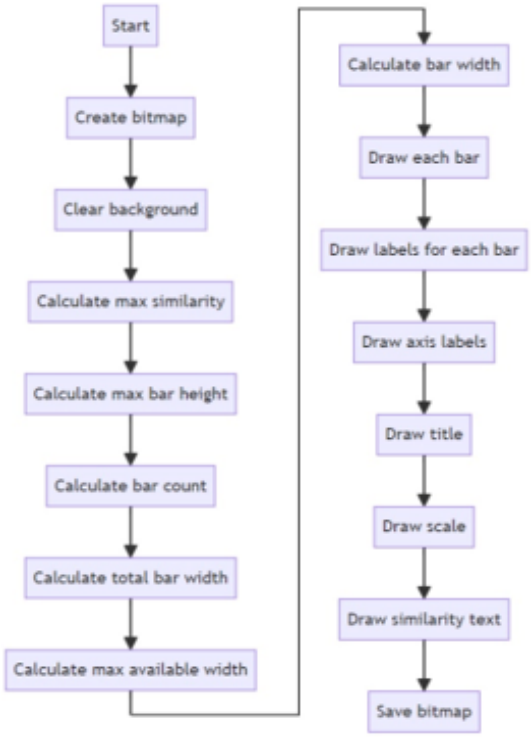

Figure 9: Graphical Representation of Implemented Draw Combined Similarity Plot Function

The "DrawCombinedSimilarityPlot" function is an essential component for visualizing the similarity between encoded inputs and reconstructed inputs. This function generates a graphical representation of the similarity values calculated for each input, allowing for a comprehensive understanding of the network's performance in reconstructing input data.

This function creates a bitmap image with bars representing the similarity values. Each bar corresponds to an input, and its height reflects the similarity between the original encoded input and the reconstructed input. The function takes several parameters, including the list of similarity values, file path for saving the image, and dimensions of the image.

Internally, the function calculates the maximum similarity value from the provided list to scale the bar heights accordingly. It then determines the width and spacing for each bar based on the specified dimensions. Next, it iterates through the similarity values, drawing a bar for each input. The height of each bar is proportional to its similarity value, providing a visual indication of how closely the reconstructed input matches the original encoded input.

Furthermore, the function adds axis labels, a title, and a scale to the image to provide context and facilitate interpretation. It ensures that the generated image is informative and visually appealing, enabling users to assess the performance of the HTM network in reconstructing input data briefly.

V. Unit-Test of SDR-Reconstructor Class

The Unit-Test of the SDR Reconstructor Class is an essential component of the software validation process within the Hierarchical Temporal Memory (HTM) framework. This unit testing suite focuses on thoroughly assessing the functionality and reliability of the SDR Reconstructor, a critical module responsible for reconstructing Sparse Distributed Representations (SDRs) from active columns identified by the spatial pooler Through systematic testing of edge cases, boundary conditions, and typical use cases, we can verify the correctness and effectiveness of the SDR reconstruction process. The Unit-Test of the SDR Reconstructor Class serves as a vital quality assurance mechanism, providing confidence in the integrity and performance of this fundamental component within the HTM framework. In this segment, we will discuss in detail the unit test.

A. Valid Input Reconstruction

Objective: This test scrutinizes the behavior of the Reconstruct method under standard operating conditions, ensuring accurate generation of a dictionary containing permanence values for the provided active mini columns. It also validates the method's handling of missing keys in the dictionary.

Test Scenario: The test begins by retrieving the HTM configuration and initializing the necessary objects, including the spatial pooler and the SDR reconstructor. A set of active mini columns is defined for reconstruction. The Reconstruct method is then invoked to generate permanence values for the specified mini columns.

Rationale for Delta Parameter: The delta parameter, specified as the tolerance for comparing double values, is crucial for evaluating the equality of expected and actual permanence values. It allows for a flexible comparison, accommodating slight variations due to computational precision.

Outcome Verification: The test asserts the presence of the reconstructed dictionary and checks for the existence of expected keys and corresponding permanence values within a specified tolerance range. Additionally, it ensures that the dictionary does not contain keys that were not present in the input.

Benefit: This test verifies the fundamental functionality of the Reconstruct method, validating its capability to accurately reconstruct permanence values for active mini columns. By confirming the method's adherence to expected behavior, it instills confidence in the reliability of the SDR reconstructor component.

B. Handling Null Input:

Objective: This test evaluates the error-handling mechanism of the Reconstruct method when invoked with a null input parameter. It ensures that the method correctly throws an ArgumentNullException in such scenarios.

Test Scenario: The test initializes the necessary objects and attempts to reconstruct permanence values using a null input parameter.

Rationale: By deliberately passing a null input, this test assesses the robustness of the SDR reconstructor in handling invalid inputs. The expected exception ensures that potential errors arising from null inputs are appropriately handled.

Outcome Verification: The test verifies that the expected exception type (ArgumentNullException) is thrown when the Reconstruct method is invoked with a null input parameter.

Benefit: This test reinforces the reliability of error handling within the SDR reconstructor, ensuring that the system gracefully handles unexpected scenarios and maintains stability during runtime.

C. Empty Input Reconstruction:

Objective: This test validates the behavior of the Reconstruct method when provided with an empty input array of active mini columns. It ensures that the method returns an empty dictionary as the result.

Test Scenario: The test initializes the required objects and attempts to reconstruct permanence values using an empty array of active mini columns.

Rationale: Handling edge cases, such as empty inputs, is essential for ensuring the robustness of the reconstruction logic. This test verifies that the SDR reconstructor gracefully handles scenarios where no active mini-columns are provided.

Outcome Verification: The test verifies that the returned dictionary is not null and contains zero key-value pairs, indicating an empty result as expected.

Benefit: By confirming the correct handling of empty inputs, this test enhances the reliability of the SDR reconstructor, ensuring consistent behavior across a variety of input scenarios.

D. Reconstruct All Positive Permanences Returns Expected Values:

Objective: This test ensures that the Reconstruct method returns permanence values that are all non-negative when all mini-column indices provided as input are positive integers.

Test Scenario: The test initializes the required objects, defines a set of active mini-columns with positive indices, and calls the Reconstruct method.

Rationale: Verifying the correctness of the reconstructed permanence values, especially when all inputs are positive, is essential for validating the reconstruction logic.

Outcome Verification: The test confirms whether all permanence values returned by the Reconstruct method are non-negative, as expected for input consisting of positive mini-column indices.

Benefit: By ensuring that all reconstructed permanence values are non-negative, this test validates the accuracy and reliability of the SDR reconstructor, preventing potential errors or inconsistencies in the reconstructed data.

E. Reconstruct Adds Key If Not Exists:

Objective: This test verifies whether the Reconstruct method adds a key to the dictionary if it does not already exist, ensuring that the method handles missing keys correctly.

Test Scenario: The test initializes the required objects, defines a set of active mini-columns, and calls the Reconstruct method to obtain the reconstructed permanence values.

Rationale: Testing the behavior of the Reconstruct method when encountering missing keys helps ensure proper handling of such scenarios to prevent data loss or inconsistencies.

Outcome Verification: The test verifies whether the dictionary returned by the Reconstruct method contains the expected keys, including any keys that were added during reconstruction.

Benefit: By validating the handling of missing keys, this test enhances the reliability and completeness of the reconstructed data, ensuring that all relevant information is included in the output dictionary.

F. Reconstruct Returns Valid Dictionary:

Objective: This test evaluates whether the Reconstruct method returns a valid dictionary containing integer keys and double values, ensuring the integrity of the reconstructed data structure.

Test Scenario: The test initializes the required objects, defines a set of active mini-columns, and calls the Reconstruct method to obtain the reconstructed permanence values.

Rationale: Verifying the validity of the reconstructed dictionary helps ensure that it conforms to expected data types and structures, preventing data corruption or misinterpretation.

Outcome Verification: The test checks whether the reconstructed dictionary contains keys and values of the expected data types (integer keys and double values), confirming its validity.

Benefit: By validating the integrity of the reconstructed dictionary, this test ensures that the output data structure is suitable for further processing and analysis, maintaining data consistency and reliability.

G. Reconstruct Negative Permanences Returns False:

Objective: This test examines the behavior of the Reconstruct method to ensure that it does not return any negative permanence values, verifying the correctness of the reconstruction logic.

Test Scenario: The test initializes the required objects, defines a set of active mini-columns, and calls the Reconstruct method to obtain the reconstructed permanence values.

Rationale: Preventing the generation of negative permanence values is essential for maintaining the integrity and interpretability of the reconstructed data, as negative values may indicate errors or inconsistencies.

Outcome Verification: The test verifies whether the reconstructed permanence values are all non-negative, ensuring that the method produces valid and interpretable output.

Benefit: By ensuring the absence of negative permanence values, this test enhances the reliability and accuracy of the reconstructed data, preventing potential issues in downstream processing or analysis tasks.

H. At Least One Negative Permanence Returns False:

Objective: This test validates the behavior of the Reconstruct method when at least one permanence value is negative, ensuring that it correctly handles such scenarios without compromising the integrity of the output.

Test Scenario: The test sets up the necessary objects, defines a set of active mini-columns that includes at least one negative permanence value, and calls the Reconstruct method.

Rationale: Testing the handling of at least one negative permanence value is essential for verifying the robustness and correctness of the reconstruction logic under diverse input conditions.

Outcome Verification: The test verifies whether the reconstructed dictionary contains at least one negative permanence value, confirming that the method behaves as expected in scenarios involving negative values.

Benefit: By validating the handling of negative permanence values.

I. Reconstruct Invalid Dictionary Returns False:

Objective: This test assesses the behavior of the Reconstruct method when provided with an invalid dictionary, ensuring that it correctly identifies and handles invalid data structures.

Test Scenario: The test initializes the required objects, defines a set of active mini-columns, and calls the Reconstruct method to obtain the reconstructed permanence values. Subsequently, it evaluates the validity of the reconstructed dictionary.

Rationale: Detecting and handling invalid data structures is essential for maintaining the consistency and reliability of the reconstructed data, as unexpected formats or contents may lead to errors or misinterpretations.

Outcome Verification: The test examines whether the reconstructed dictionary is considered invalid based on specific criteria, such as the presence of NaN values or negative keys. It verifies whether the method correctly identifies and flags invalid data structures.

Benefit: By validating the handling of invalid dictionaries, this test enhances the robustness and error resilience of the Reconstruct method, ensuring that it produces valid output even in the presence of unexpected data formats or anomalies.

VI. Result Analysis

The accurate reconstruction of encoded inputs by the HTM network is examined in the input reconstruction evaluation. Although the algorithm performs well in this task overall, small differences show how difficult it is to capture high-dimensional sparse distributed representations. Heatmap visualization offers important insights into how permanence values are distributed throughout the network. Despite its effectiveness, color interpolation schemes can be difficult to interpret, requiring careful qualitative analysis in addition to quantitative measures.

In the below figure (10) we can see our combined Heatmap which has three separate parameters. The Generated Permanence visualization image consists of three parts. It has Heatmap, and it generates and interpolates colors according to the Permanence values that are Restructured by using the Spatial Pooler Reconstruction Method. In the second property, we visualize the original encoded Input bits for numbers and images for two types of inputs. The third and final property is the Reconstructed input. We keep all three things together for

better understanding and comparing purposes to understand the specific input by comparing it with the original encoded input bits. In this figure (10) we used a split-cropped part of a random image from our input.

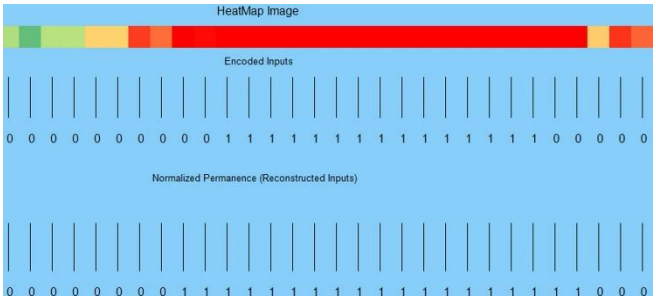

Figure 10: The Image for the final output.

For the image input, the output is also the same category as this but different in size because the size of the dependent variable is due to the size of the encoder and types of the input data. For instance, the size of the output of the scaler encoder is 200. On the other hand, for the image input, the encoded bits depend on the size of the input image size both (Height and Width). Here in this below figure (11) we showed a full input of the reconstruction result and permanence values as a heatmap. In this figure (11) is clearly visible that the Reconstruction method can reconstruct the input, we presented the encoded input for each numerical input (0 to 1) as well to compare with the Reconstructed input to better understand to final outcome of the experiment.

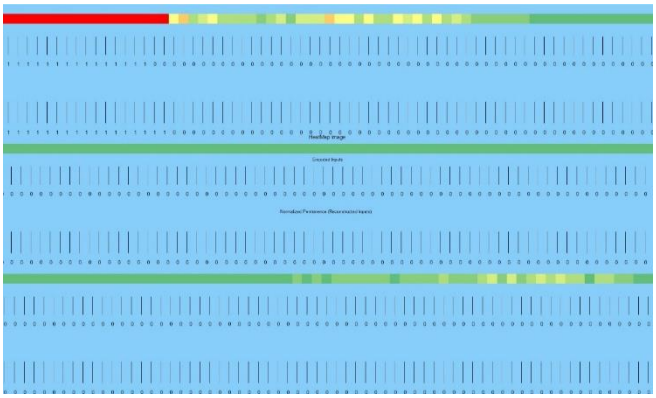

Figure 11: The output of the Result for Numerical input 1 (Heatmap, Encoded Inputs and Reconstructed Inputs)

Moreover, in our Experiment we have validated the result by checking the similarities between the Original encoded inputs and the Reconstructed Inputs by calculating the Jaccard similarities coefficient. (Equation 2) Here, we will present our Analyzed result by plotting the Similarity Bar graph for both Inputs data types. Here, we will present both data types, firstly, for the numerical Inputs here we calculated the similarities for all integer type inputs (0 to 99 total 100) the corresponding figures (12,13,14,15 and 16) will illustrate the similarity between Original encoded Inputs and Reconstructed Inputs.

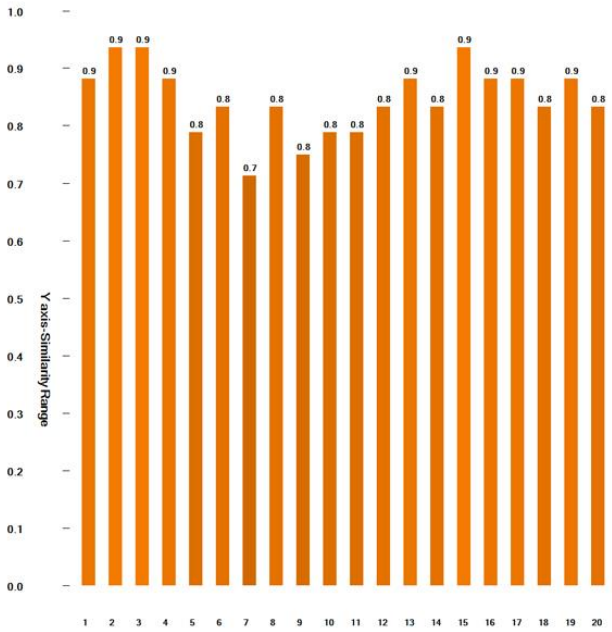

Figure 12: The similarity graph of input 1 to 20

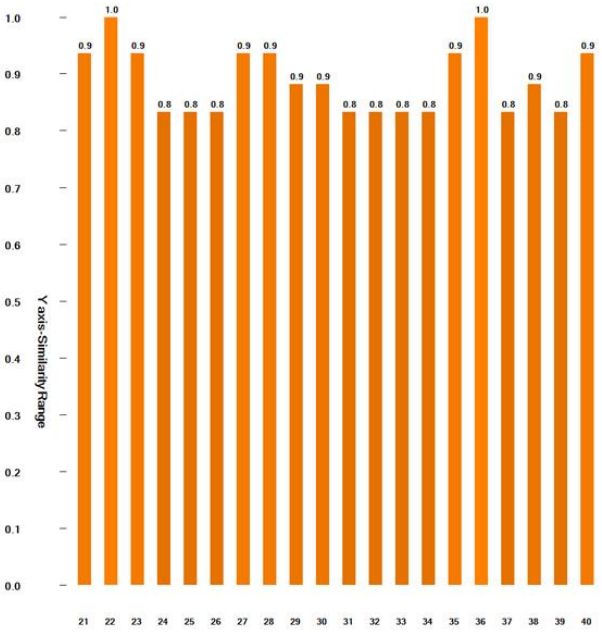

Figure 13: Similarity graph for input 21 to 40

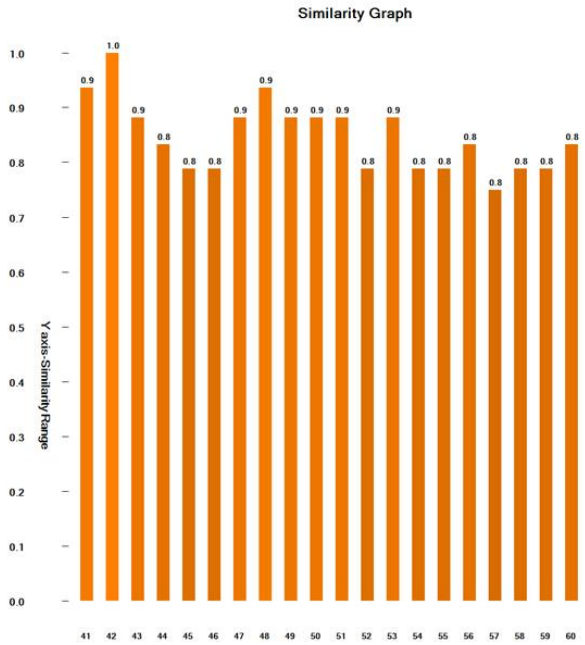

Figure 14: Similarity graph for input 41 to 60

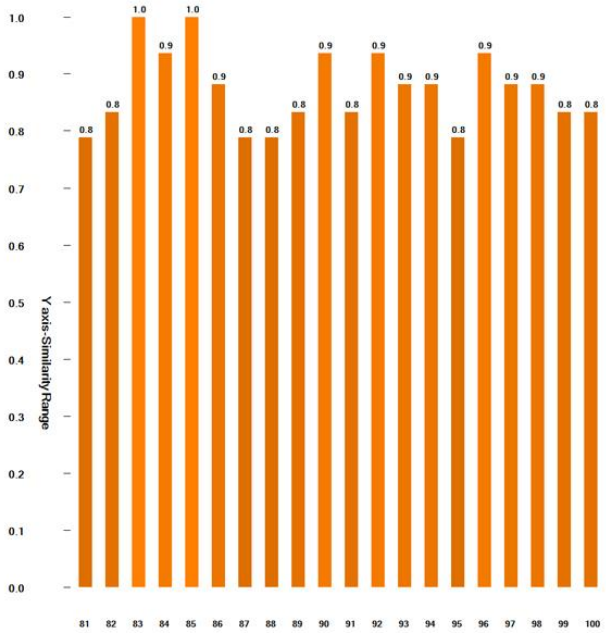

Figure 16: Similarity graph for input 81 to 100

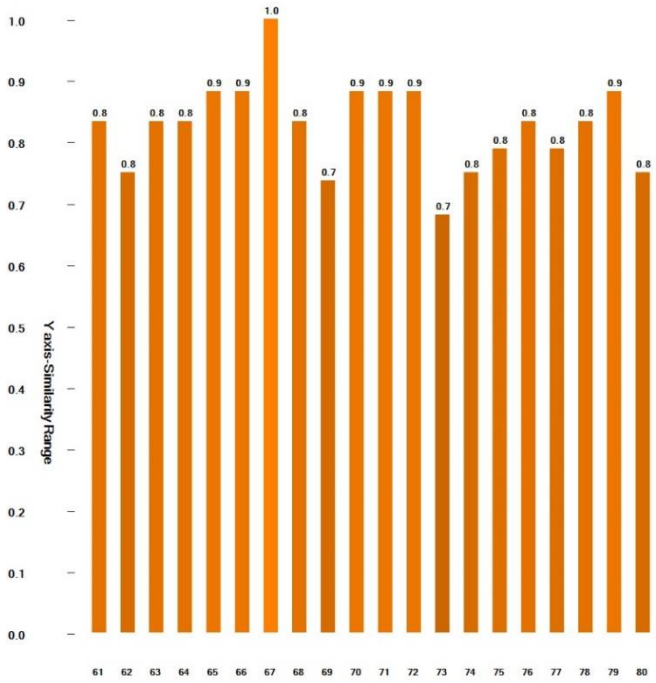

Figure 15: Similarity graph for input 61 to 80

This figure (17) here is the generated Similarity graph for validating output but we split this one to better visualization.

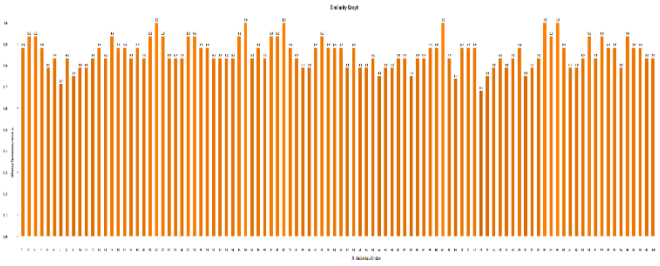

Figure 17: The similarity graphs of all numerical Inputs

Secondly, For the similarity graph for the Image input in figure (18) we took 7 images as inputs for the experiment, and we plotted the similarity graph between the binarized encoded Inputs and the Reconstructed Input.

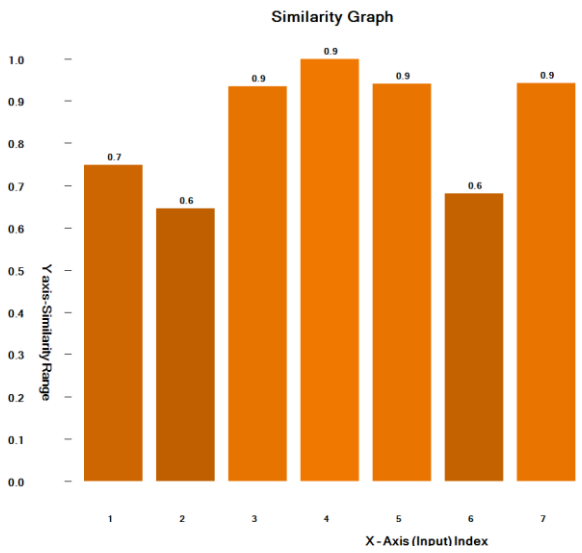

Figure 18: Similarity Graph for the Image Input.

VII. Discussion

The discussion section provides a thorough analysis of the study's findings, their implications, and the larger context in the field of Hierarchical Temporal Memory (HTM), highlighting some key points for further exploration. We examine the study's findings' significance in this section.

With this discussion, we hope to present on the contributions of our work, point out directions for future research, and provide ideas that improve our knowledge of and ability to use HTM networks along with the Reconstruction Process.

A major challenge could be in the presence of noise within the encoded Sparse Distributed Representation (SDR) poses a notable challenge in the reconstruction process, potentially distorting underlying patterns and compromising reconstruction accuracy. While our current study did not explicitly explore this scenario, it underscores a crucial area for future investigation. By delving into methods for enhancing noise robustness in decoding algorithms, we could bolster the reliability and practical utility of Hierarchical Temporal Memory (HTM) networks, particularly in real-world applications where data integrity is paramount.

Moreover, our observation regarding the impact of encoder choice on HTM network performance highlights an essential consideration in network design and optimization. The discrepancy in performance between using different encoders, such as the image binarize versus the scalar encoder for numerical inputs, underscores the significance of encoder selection in achieving optimal outcomes. Further exploration into the effects of diverse encoding strategies on reconstruction accuracy and robustness could yield valuable insights for tailoring HTM networks to accommodate a wide range of input data types and enhance their effectiveness in various applications.

VIII. Conclusion

To sum up, our research has yielded significant knowledge about how to reconstruct Sparse Distributed Representations (SDRs) in Hierarchical Temporal Memory (HTM) networks. We have investigated the complexities of the HTM framework and its use in Reconstruct encoded Input bits by putting different components into practice and analyzing them. These components include input encoding, reconstruction algorithms, similarity calculations, and visualization techniques.

The significance of encoder selection in affecting reconstruction accuracy and network performance is one of our study's main conclusions. Different encoding strategies produced different results, which emphasizes the importance of carefully selecting encoding techniques based on the type of input data.

Furthermore, our analysis of similarity computation techniques, including the Jaccard similarity coefficient, has yielded a numerical reconstruction fidelity measure, which makes the assessment of reconstruction precision and network efficiency easier.

Overall, our work lays the groundwork for further study and advancement in this area by improving our understanding of and ability to use HTM networks for Reconstructing encoded Inputs. HTM networks have the potential to provide strong solutions for a range of data processing and analysis tasks through further research and development, advancing the fields of cognitive computing and artificial intelligence.

IX. References

[1] J. &. A. S. (. Hawkins, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," Frontiers, pp. 10,23, 30 March 2016.

[2] D. &. H. J. George, "A Hierarchical Temporal Memory Model for Learning and Recognition of Sequential Data.," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[3] I. G.-C. L. &. S. A. Rodriguez-Lujan, "Enhancing Encoder Robustness in Hierarchical Temporal Memory with Self-Supervised Learning. Conference/Journal," January 2020.

[4] A. J. B. &. P. R. Smith, "Automated Encoder Discovery in Hierarchical Temporal Memory using Neural Architecture Search. Conference/Journal," January 2020.

[5] J. F. E. &. K. D. Mnatzaganian, " A mathematical formalization of hierarchical temporal memory's spatial pooler. Frontiers in Robotics and AI, 3," vol. DOI: 10.3389/frobt.2016.00081, January 2017.

[6] J. &. S. A. Thornton, "Spatial pooling for greyscale images. International Journal of Machine Learning and Cybernetics, 4(3), 207–216," Vols. DOI: 10.1007/s13042-012-0087-7, 2012.

[7] G. K. a. V. K. M. Steinbach, Similarity Measures for Text Document Clustering," in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, pp. 48-57.