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

Improve Samples and Documentation for SDR representation

Md Tanzeem Hasan Mahmud
mahmud.mdtanzeemhasan@stud.fra-uas.de

Khan Mushfiqur Rahman
khan.rahman@stud.fra-uas.de

Abstract*— Sparse Distributed Representations (SDRs) are fundamental to the architecture of Hierarchical Temporal Memory (HTM) systems, mimicking the sparse and distributed nature of neural activity observed in the brain. An individual pixel in the SDR of digital representation of an image or graphic is called a bitmap. This paper presents a method for visualizing Sparse Distributed Representations (SDRs) as bitmap images, leveraging the DrawBitmap utility within Hierarchical Temporal Memory (HTM) systems. SDRs, akin to the brain's data processing technique, offer a robust and efficient way to handle diverse information types. By converting SDRs into bitmap images, this can visually interpret the encoded patterns and assess the functionality of encoders and spatial pooling processes in HTM models. This paper explains the working procedure of DrawBitmap methods and an understanding of SpatialPooler and Encoder's capabilities in visual representation and data encoding.

Keywords—HTM, SDR, Bitmap, DrawBitmap

I. INTRODUCTION

Sparse Distributed Representations (SDRs) are pivotal in computational models that mimic the human brain's processing, such as Hierarchical Temporal Memory (HTM). Brain works on continuous stream of input patterns which represent input sequences based on the input stream's recursive pattern [1]. Representing data in a sparse and distributed manner allows for efficient and robust information processing. However, understanding and interpreting the complex patterns encoded within SDRs can be challenging. This paper introduces a visualization method that transforms SDRs into bitmap images, providing an intuitive way to analyze and comprehend the information encoded by HTM systems.

Through the customization of *DrawBitmap* methods parameters such as dimensions, colors, and additional text, this approach allows users to personalize visualizations to suit their unique analysis requirements, promoting a more profound comprehension of SDR properties and behaviors. This paper demonstrates the practicality and flexibility of bitmap representations in understanding the complexities of SDRs, using examples from Date Time, Scalar, and Geospatial Encoders, Binary Encoders, Spatial Pooler etc.

A machine learning technique called Hierarchical Temporal Memory (HTM) was developed after studying how the neocortex, a region of the human brain, arranges and analyzes data. It simulates the way that our brain processes several kinds of information, such as vision, sound, and behavior. The HTM network can identify patterns in input data, learning from them, and using that knowledge to predict future events. HTM is a structural and algorithmic characteristic of the neocortex [2].

The Hierarchical Temporal Memory term suggests its three main features: hierarchy, temporal, and memory. In terms of hierarchy, an HTM network is structured in multiple levels organized hierarchically. Information from lower levels is combined at higher levels, creating more complex components. Regarding temporal aspects, an HTM network can learn both spatial (related to space) and temporal (related to time) patterns from a continuous stream of data. This means the network's output at a given time not only depends on the current input but also on previous ones.

In Hierarchical Temporal Memory (HTM) systems, the Spatial Pooler is a crucial component that processes input patterns and produces Sparse Distributed Representations (SDRs). Through this paper, it can be learned about the functioning of the Spatial Pooler and how it contributes to pattern recognition in the cortex by using the Draw Bitmap technique. The HTM temporal memory learns temporal sequences of the SDRs and makes predictions for future inputs [3].

Also in this project, encoder is an important component which responsible for converting the raw data from its original format into a suitable representation. Encoders generally apply scaling, binning, mathematical transformations, or other methods to transform input data into a format that reduces dimensionality and duplication while maintaining pertinent information.

In this program, Encoders generates SDRs with a constant number of bits 'N' and a fixed number of active (1's) bits 'W', regardless of what they represent. An encoder should always provide the same output for the specific input. If the encoders offered various SDR bit lengths, comparisons and other operations would not be possible.

The output of the Spatial Pooler can be seen visually due to the *DrawBitmap* function in the cortices. By using this technique, this can be able to comprehend the process by which the Spatial Pooler converts input patterns into organized representations, which in turn allows the HTM

temporal memory to learn temporal sequences and anticipate future events.

II. METHODOLOGY

In this section, this will describe the approach which has been followed to improve samples and documentation for SDR representation. The objective was to visualize Sparse Distributed Representations (SDRs) as bitmap images, utilizing the *DrawBitmap* utility within Hierarchical Temporal Memory (HTM) systems. To achieve this, first performed *DrawBitmap* method [4]. The *DrawBitmap* method transforms a two-dimensional array representing an SDR into a visual bitmap image. By specifying the dimensions, colours, and additional text, this function can customize the visualization to suit the analysis as per the needs. The method scales the SDR array to fit the specified bitmap dimensions, allowing for a clear and adjustable representation of the SDR's structure.

The purpose of generating bitmaps to represent Sparse Distributed Representations (SDRs) in Hierarchical Temporal Memory (HTM) is to provide a visual representation of the encoded information and the processing performed by HTM algorithms such as encoders, spatial poolers, and temporal memory. By visualizing the SDRs as bitmaps, one can gain insights into how the input data is transformed and processed throughout the different stages of the HTM system. Bitmaps allow for easy interpretation of the sparsity and patterns within the representations, facilitating analysis and debugging of the HTM algorithms. Additionally, visualizing SDRs as bitmaps enables researchers and developers to observe the effects of parameter changes or optimizations on the encoded representations, aiding in the refinement and improvement of HTM algorithms. Bitmaps of SDRs serve as a valuable tool for understanding the inner workings of HTM algorithms and optimizing their performance for various applications.

In the context of an encoder, bitmaps visually depict how raw input data is transformed into sparse binary patterns. Each bit in the bitmap represents the presence or absence of a feature or characteristic in the input data. By generating bitmaps of the encoded representations, one can observe how different input signals are represented sparsely in the binary space. This visualization aids in understanding how the encoder is capturing relevant information from the input data and converting it into a format suitable for further processing by the HTM network.

For the spatial pooler, bitmaps illustrate the activation patterns of columns in response to input data. Each column's activation state is represented by a bit in the bitmap, where active columns are indicated by set bits and inactive columns by unset bits. By generating bitmaps of the spatial pooler's output, one can visualize how the input patterns are distributed and transformed across the columns of the spatial pooler. This visualization helps in analyzing the sparsity and distribution of active columns, as well as understanding the spatial pooling process and its impact on the input representations.

In temporal memory, bitmaps depict the active cells and connections within the network over time. Each bit in the bitmap corresponds to the activation state of a cell or connection, representing the temporal sequence of patterns learned by the HTM network. By generating bitmaps of the temporal memory's activity, one can observe how the network learns and predicts sequences of input patterns, as well as analyze the stability and adaptability of the learned representations. This visualization aids in understanding the temporal processing capabilities of the HTM network and assessing its performance in sequence learning tasks.

Turning an SDR into a visual bitmap involves a few straightforward steps as shown in the flowchart. Also, brief description of these steps defined after that for understanding those properly.

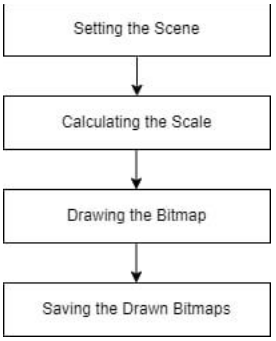

Fig 1: Steps of DrawBitmap method

Description of these steps given below.

1. *Setting up the Scene:*

First Determine the width and height for the bitmap, scaling the image to fit your needs. Then, choose colors for the active and inactive cells to make the SDR's structure clear.

2. *Calculating the Scale*:

A scale factor is calculated based on the ratio of the bitmap's width to the SDR array's width. This helps adjust the cell sizes in the bitmap to fit the entire SDRs.

3. *Drawing the Bitmap*:

The *DrawBitmap* function converts the two-dimensional array into a bitmap image, with each cell in the array corresponding to a pixel in the bitmap. Active cells in the SDR are represented in yellow, while inactive cells are represented in black. The resulting bitmap image provides a visual representation of the SDR, allowing for easy interpretation and analysis of the encoded patterns.

4. *Saving the drawn bitmaps:*

Once every cell is colored, the bitmap is saved to the location specified in file Path. This method simplifies analyzing and understanding SDR patterns by providing a visual representation.

There are three functions of DrawBitMaps in this project. Two DrawBitMaps functions take a two dimensional array and one DrawBitMaps function takes a list of two dimensional arrays.

The main function of DrawBitMaps is:

public static void DrawBitmap(int[,] twoDimArray, int scale, String filePath, Color inactiveCellColor, Color activeCellColor, string text = null)

The working flowchart of this function is:

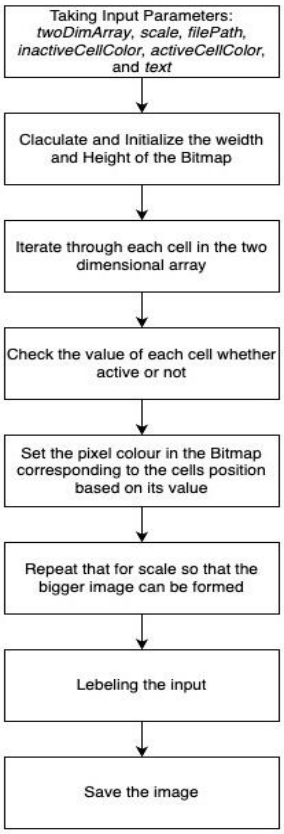

Fig 2: Flowchart of DrawBitmap function using scaling parameters

This function takes a two-dimensional array twoDimArray representing a binary image, along with parameters for scale, file path, inactive cell color, active cell color, and an optional text label. First, it determines the dimensions of the array w and h. Then, it creates a new bitmap image with dimensions w * scale by h * scale. It iterates over each element in the array, and for each active cell (value of 1), it sets the corresponding pixel in the bitmap to the active cell color, while for inactive cells (value of 0), it sets the pixel to the inactive cell color. This process is repeated at scale times to scale up the image. Additionally, if a text label is provided, it draws the text onto the bitmap. Finally, the bitmap is saved to the specified file path as a PNG image. The function effectively converts a binary array into a bitmap image with customizable colors and text. This function is mainly for generating the SDRs for active columns. This same function can be used for encoder also with transposing the two dimensional array that this function takes as an input.

Another function is written and that is also for active columns with small changes.

public static void DrawBitmap(int[,] twoDimArray, int width, int height, String filePath, Color inactiveCellColor, Color activeCellColor, string text = null)

The working flowchart of this function is:

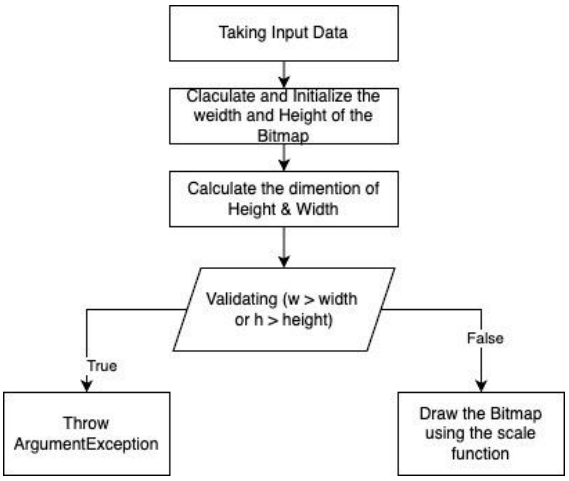

Fig 3: Flowchart of DrawBitmap function with validating W & H

This function also takes input as a two dimensional array and additionally it takes height and width which is validated by two dimensional array's row length and column length. Here, 'w' is the row's length and 'h' is the column's length. If the condition met then it throws an argument exception else the previous method of draw bitmap function is used.

Another function was written which takes a list of two dimensional arrays.

public static void DrawBitmap(int[,] twoDimArray, int scale, String filePath, Color inactiveCellColor, Color activeCellColor, string text = null)

The working flowchart of this function is:

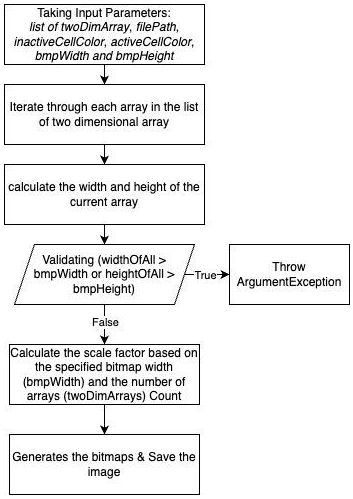

Fig 4: Flowchart of DrawBitmap function with list of two-dimensional array

At first, this function takes a list of two dimensional arrays then iterates through each two dimensional array. After this, the two dimensional array also validated like the previous method of draw bitmap function. Here the different thing is, it is calculating the scale value based on the specified bitmap width and the number of arrays (two dimensional arrays) count. At the end, it generates the bitmaps and saves the image.

There are no bitmap functions of a one-dimensional array. That's why, here created a function for representing the SDRs for an one dimensional array.

public static void Draw1DBitmap(int[] array, string filePath, int scale = 10)

The working flowchart of this function is:

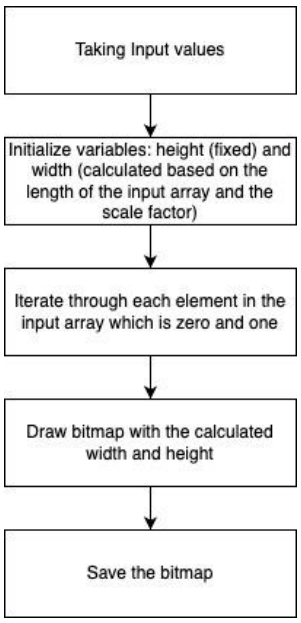

Fig 5: Flowchart of DrawBitmap function using 1DBitmap

This Draw1DBitmap method creates a 1D bitmap image representing a binary array. Each element in the array corresponds to a bit in the bitmap image. Active bits (with a value of 1) are represented as black rectangles, while inactive bits (with a value of 0) are represented as white rectangles. The dimensions of the bitmap are determined by the length of the input array and a specified scale factor. The resulting bitmap image is saved to the specified file path in PNG.

III. RESULTS

The paper presents a series of bitmap images generated from encoded SDRs, showcasing how different data types, including numbers, geographical coordinates, and temporal information, can be visually represented. Here illustrates the efficacy of the *DrawBitmap* method in highlighting the active and inactive bits within SDRs, providing insights into the encoding mechanisms of HTM models.

Generating of Bitmap using different Encoder:

In the experiment where used binary encoder, the implemented method showed how to encode the taken values. Subsequently, it will configure the SDR encoder settings, encode the value using those settings, and establish the two-dimensional array. The output can be found as follows.

For the value of 40148:

Fig 6: Example of visualizing a number 40148

For the value of 50149:

Fig 7: Example of visualizing an integer number 50149

The following result showcases DrawBitMap with Binary Encoder for 1D image generation. Output image follows -

Fig 8: Example of DrawBitMap with Binary Encoder 1D image

For the experiment of DateTime encoder, here used the *DrawBitmap* method for visualizing DateTimeEncoder outputs. By utilizing this encoder, this can be able to convert the datetime values into its corresponding SDR. In Date time encoder the value which has been passed are dates and precisions. Precisions are based on different parameters like days, hours & minutes etc. After passing this value in the encoder, it returned the SDRs only for the days as mentioned in the method.

This was employed by the DrawBitMaps method to visualize this SDR. This process allowed us to effectively represent and comprehend the temporal data in a binary format. Output image can be found below.

For this example given below, select the values as "08/03/2024 21:58:07" and send it through datetime encoder to get SDR [5].

This process allowed us to effectively represent and comprehend the temporal data in a binary format. Output image as follows –

Fig 9: DrawBitmap example for DateTime Encoder

As next, the work is done for AQI which stands for Air Quality Index. It is a measure used to communicate the quality of the air in a specific location and its potential impact on health. The AQI is calculated based on several key air pollutants, including particulate matter (PM2.5 and PM10), ground-level ozone, carbon monoxide, sulfur dioxide, and nitrogen dioxide.

The Scalar Encoder converts AQI levels into SDRs, capturing the essence of air quality in a binary format. For instance, the AQI levels are segmented into: 0-49: Good 50-149: Moderate 150-249: Unhealthy for Sensitive Groups 250-349: Unhealthy 350-449: Very Unhealthy 450-500:

Hazardous. For this example, here generated a bitmap to visualize the encoded AQI values to get SDR [6].

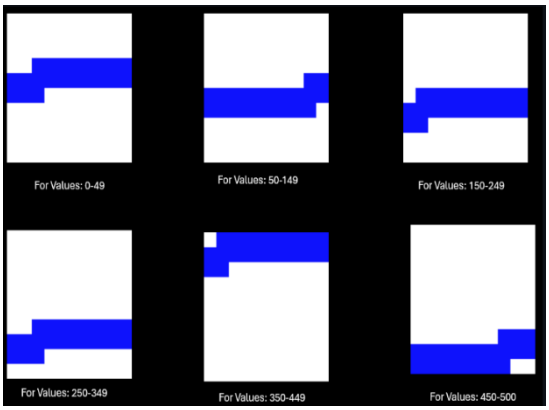

Fig 10: Drawing AQI Values with Scalar Encoder

Geospatial encoder is used to encode latitude or longitude values into Sparse Distributed Representations (SDRs) for Hierarchical Temporal Memory (HTM) systems. In the exploration of geospatial data through Sparse Distributed Representations (SDRs), here utilize the *DrawBitmap* method to translate encoded geographical coordinates into visually interpretable bitmap images. This approach allows for the visualization of spatial information encoded within SDRs, offering insights into the encoded geographical regions.

The generated bitmap are as follows:

Fig 11: DrawBitmap sample for Geospatial Encoder

The bitmap images generated for geographical coordinates offer a unique view of the spatial patterns encoded within the SDRs. Now if after change the encoder settings and provide the below settings:

encoderSettings.Add("W", 21);

encoderSettings.Add("N", 40);

The output will be different for the same value. The bitmaps generated in this case are:

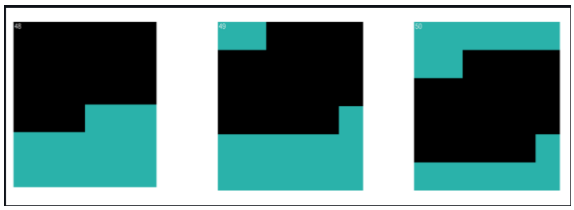

Fig 12: DrawBitmap sample for Geospatial Encoder after encoder changed.

DrawBitmap output for Spatial Pooler:

At first, bitmaps generated for numbers. For this, spatial pattern learning experiment is used. There is a variable called ***isInStableState*** which is responsible for to decide whether all the active mini columns of input is stable or not. Here is the output for input 20:

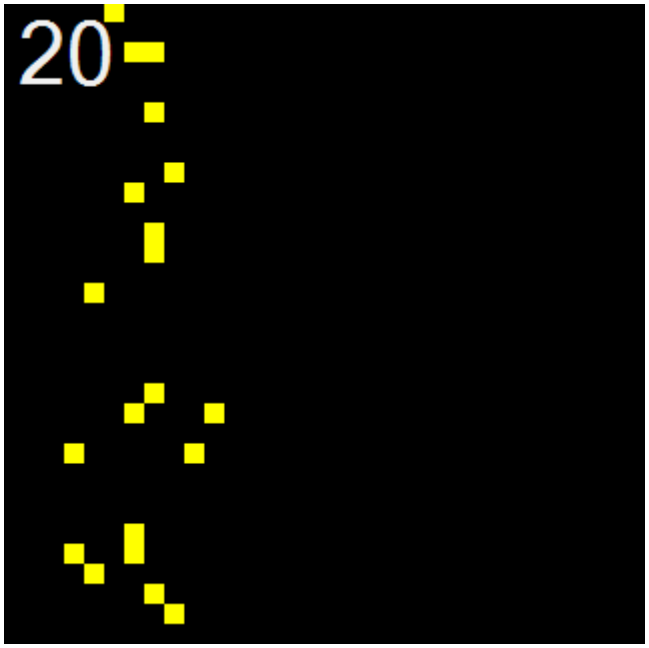

Fig 13: Bitmap of input 20

In that spatial pattern learning experiment, generated bitmaps are for input 0 to input 99.

In spatial pooler experiment [7], the *DrawBitmaps* utility function generates bitmap images that visually represent the active columns because of the SP's processing. The generated Images looks like this.

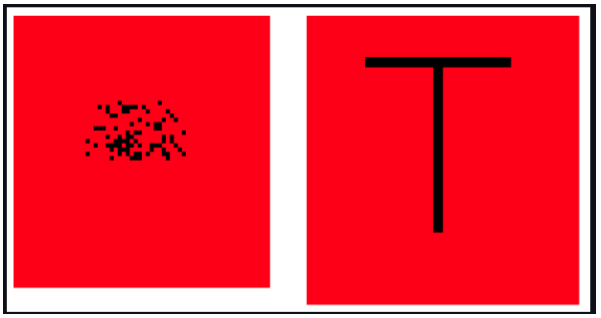

Fig 14: Bitmap representation of Image using Spatial Pooler

On the right side it is the image generated from the binarized file of the input training image, on the left is the SDR representation by SpatialPooler after feeding the training image.

Example representing Overlap (Intersection), Difference and Union for Alphabet T and Numeric 3 in Bitmap after computing in spatial pooler:

Here this can show the Alphabet T and Numeric 3 for spatial pooling.

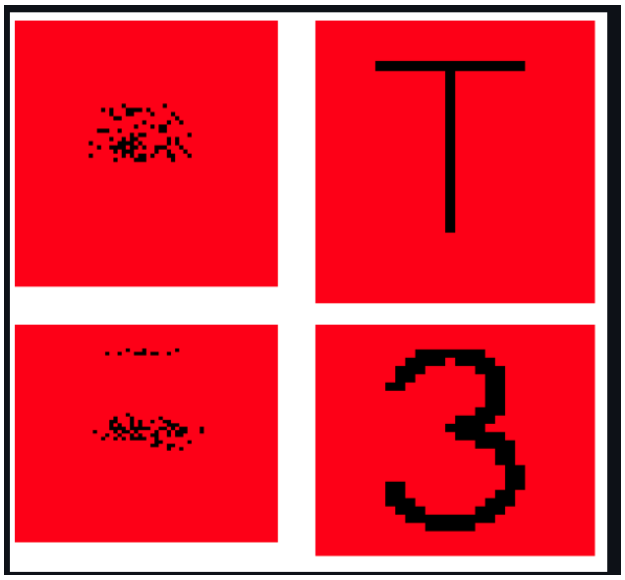

Fig 15: Example representing Overlap (Intersection), Difference and Union for Alphabet T and Numeric 3 in Bitmap

Additionally, here explored the visualization of SDRs generated by the spatial pooling process, demonstrating how input patterns are captured and represented in the model.

IV. Discussion

The set of methods presented for drawing bitmap images serves as a versatile process for visualizing binary data representations. Starting from the basic DrawBitmap function, which converts a two-dimensional array of binary values into a bitmap image, these methods are used for a wide range of visualization needs.

The DrawBitmap method efficiently converts two-dimensional binary arrays into bitmap images, offering flexibility in customization through parameters such as cell colors, dimensions, and optional text annotations. Its counterpart, Draw1DBitmap, specializes in creating one-dimensional bitmap images, suitable for visualizing binary sequences or linear data structures. Additionally, the DrawBitmaps method extends the functionality to visualize multiple binary arrays simultaneously, stitching them together into a single composite bitmap image.

Through the implementation of these methods, users can effectively visualize binary data representations, aiding in the analysis and interpretation of various data patterns and structures. Whether it's representing spatial patterns, encoding schemes, or other binary data, these methods provide a valuable toolset for generating insightful visualizations.

In conclusion, this project aims to interpret the encoded patterns and in HTM models by converting SDRs into bitmap images. Visualizing SDRs as bitmap images can interpret the complex patterns encoded in HTM systems. This approach not only aids in the analysis of HTM models but also paves the way for analyzing different types of data pattern from the input. Overall, this project can generate sparse distribution representations and all together a process to run the experiment with bitmap generation of SDRs whether it would be 1D or 2D.

V. References

[1] Ahmad, Subutai and a. J. Hawkins, "Properties of sparse distributed representations and their application to hierarchical temporal memory," 2015. [Online]. Available: arXiv preprint arXiv:1503.07469.

[2] J. Hawkins, S. Ahmad and D. Dubinsky, "Cortical learning algorithm and hierarchical temporal memory.," [Online]. Available: http://numenta.org/resources/HTM_CorticalLearningAlgorithms.pdf.

[3] Mnatzaganian, James, E. Fokoué and a. D. Kudithipudi, "A mathematical formalization of hierarchical temporal memory's spatial pooler.," *Frontiers in Robotics and AI : 81,* 2017.

[4] D. Dobric, "DrawBitMap Method," 2023. [Online]. Available: https://github.com/ddobric/neocortexapi/blob/be9a9e5d21965e7401f2ecb85d9d82275ca8ec22/source/NeoCortexUtils/NeoCortexUtils.cs#L50.

[5] M. T. H. Mahmud and K. Rahman, "DateTimeEncoderExample," 2023. [Online]. Available: https://github.com/TanzeemHasan/neocortexapi/blob/44772a45ac31c48e74a648ca9b1386fb82520590/source/UnitTestsProject/EncoderTests/DateTimeEncoderTests.cs#L78.

[6] M. T. H. Mahmud and K. Rahman, "ScalarEncoderExample," 2023. [Online]. Available: https://github.com/TanzeemHasan/neocortexapi/blob/f9953ba73c30c4dd24d979e1ca5eb52c5eab4137/source/UnitTestsProject/SdrRepresentation/ScalarEncoderTestOverBitmap.cs#L396.

[7] M. T. H. Mahmud and K. Rahman, "SpatialSpatialEncoderExample," 2023. [Online]. Available: https://github.com/TanzeemHasan/neocortexapi/blob/8de0bf4d823b94393381b63984c8c1c5a47e330d/source/UnitTestsProject/SdrRepresentation/SpatialPoolerColumnActivityTest.cs#L19.