Approve prediction for multisequence learning

Poonam Dashrath Paraskar
poonam.paraskar@stud.fra-uas.de

Pratik Prakash Desai
pratik.desai@stud.fra-uas.de

Ankita Talande
ankita.talande@stud.fra-uas.de

***Abstract*—** Multisequence learning is the approach used to investigate implicit learning, where the model learns temporal patterns of sequences one by one during the experiment and provide the matching sequences as the predicted output. Model also predict the next element of the predicted sequence. List of sequences with double data-type are stored in an excel file, which worked as an input sequence file for model to train itself by storing these values in temporal memory. This experiment demonstrates how to learn sequences using *RunPredictionMultiSequenceExperiment()* method and save all the sequences as a dictionary object which is then considered as training data.

After that temporal memory will quickly learn cells for patterns and memorized the sequences. By using HtmClassifier's prediction method, where we must pass one test data sequence, by comparing each value of test data sequence with the trained data, model will predict which sequence data is matching with test data, it also predicts the next element of that sequence. This experiment also focuses on calculating the accuracy of matched sequences and writing the resulted accuracy into CSV file. In this experiment the additional implemented functionality helps user to pass both input sequences and test data sequences through excel file, which helps user to modify the sequence values directly through an external file instead of adding code changes

Keywords—: Multisequence, Temporal Memory, Prediction.

I. Introduction

Hierarchical temporal memory (HTM) is a type of machine learning model inspired by the structure and function of the neocortex in the human brain. Its purpose is to learn sequences of information and make predictions based on them. The HTM model consists of several components, including an encoder, a spatial pooler, a temporal memory, and an HTM classifier. Each element in a sequence is represented by a Sparse Distributed Representation (SDR). The temporal memory learns the sequences of these SDRs and makes predictions about future input SDRs. [1]

During the learning phase, the temporal memory predicts the next element in the sequence at every step.
After the last prediction, a learning cycle ends, and the temporal memory starts again with the same sequence.
The goal is for every cycle to have correct predictions for every element, resulting in an accuracy of 100%.
The aim is to achieve 100% accuracy for as many consecutive cycles as possible.
Sequence-to-sequence learning is a general framework used for machine translation, where the encoder maps the input sequence to a fixed-length vector. [2]

II. Methods

The components we are using to caried out this experiment are:

1. SDRs
2. HTM (Hierarchical Temporal Memory)
3. Spatial Pooler
4. Encoders

A. **SDRs***:* Sparse Distributed Representation (SDR), a system for information organization in HTM, is effective. A small portion of the big, interconnected cells are only partially active at any given time, which is referred to as "sparse" in this context. The word "distributed" means that the active cells, which are used to represent the activity of the region, are dispersed throughout the territory. HTM employs a binary SDR, which comes from a particular encoder and is more computationally and biologically reasonable. Because to the binary SDR's crucial properties, functional information is not lost even though the number of possible inputs surpasses the number of possible representations. [3]

B. **HTM***:* Hierarchical temporal memory (HTM) provides a theoretical framework that models several key computational principles of the neocortex. The HTM architecture is made up of layers of hierarchically stacked processing nodes, also referred to as neurons. Each neuron interacts with other neurons in a dispersed, sparse network to process the input it receives. The HTM architecture is designed to perform pattern identification, anomaly detection, and prediction tasks using time-series data. It accomplishes this by using a memory approach called temporal pooling, which gives it the ability to learn and recognize patterns over time even in the presence of noise and variance. [4]

Information is transferred up the hierarchy in layers in the HTM design, with each layer producing progressively more intricate representations of the input. An HTM system typically receives its input as a stream of high-dimensional sensory data, such audio or video. Depending on the input, the output consists of several predictions or actions. Overall, the HTM architecture provides a solid and flexible framework for building intelligent systems that can adapt to and learn from new data over time. [4] This makes it suited for a wide range of applications, including robotics, computer vision, and natural language processing.

C. **Spatial Pooler***:* By assigning active cells to columns, the Spatial Pooler creates a Sparse Distributed Representation (SDR) input. Synapses connect each column to the next region of input bits; despite the fact that multiple columns may look same, they are all distinct from one another. Various patterns generate different levels of activation, and in the columns, stronger activation suppresses weaker activation levels. The size of the columns can be changed to accommodate little or huge areas. In order to restrict the representation of input, an inhibitory mechanism is put in place. The HTM develops connections between cells based on the input. Synapse permanence updating is a type of learning. Whereas inactive columns have a lower persistence value, active columns have a higher one. [5]

D. **Encoders***:* An HTM's evolution is influenced by the data it receives and the way that data is shown. An encoder is used to transform arbitrary input into a format that an HTM can comprehend, allowing the HTM to interpret the input. Each bit in this Sparse Distributed Representation (SDR) format indicates the activation state of a column in the HTM's prior area. The following region of the HTM then uses the SDR as a feedforward input. [6]

Newly Introduced methods: In the existing implementation of Multisequence learning project, in all the methods inputs were hardcoded so if user wants to change the input, then it’s necessary to change the input sequences from the code. so, to resolve the issue we have tested different methods to take the inputs from the file.

1. *GetInputFromTextFile(*): Team has implemented GetInputFromTextFile() method to take the inputs from the Text file. We have tried 2 approaches to split the multiple input sequences by using comma ‘,’ to separate each digit of the input sequence and using special character at the end of each sequence for splitting it from other input sequences. In this case we used semi-colon ‘;’ to split. The significant issue we faced by using this approach is we had to add both comma ‘,’ and semi-colon ‘;’ at the end of each input sequence, which is not a feasible solution and by which text file also looks inappropriate.

   To resolve issue, team has encountered in the first approach we used regular expression to split multiple sequences based on detecting the enter ‘/r/n’. Using this approach wherever we added enter for next input, is getting detected by our regular expression logic. For this we had to read all the rows together using reader.ReadToEnd( ) method and then split it by detecting the enter keyword. This can cause an issue in real time working environment.

2. *GetInputFromCsvFile( ):*Team has implemented GetInputFromCsvFile( ) method to take the inputs from the CSV file. CSV stands for "Comma-Separated Values". It is a file format used for storing and exchanging tabular data, such as spreadsheets or databases. In a CSV file, each line represents a row of data and each field within a row is separated by a comma. CSV files are simple and widely supported, making them a popular choice for data exchange between different systems and applications.

   The problem with CSV file is we need to add one non double character at end of each row to terminate the row/sequence and take the next sequence. This can cause an issue in real time working environment.

3. *GetInputFromExcelFile( ):*In the GetInputFromExcelFile() method we are using .xlsx file type to take the input sequences. Which are referred as training data sequences. Here we overcame the issues of the previous methods GetInputFromCsvFile() and GetInputFromTextFile() where we need to add any non-double value to terminate the row/sequence and to jump to the next row/sequence and any special in case of text file to jump over the next input sequence. To implement this feature we used the string.IsNullOrWhiteSpace( ) property.

4. *GetSubSequencesInputFromExcelFile( ):* Team has implmented the GetSubSequencesInputFromExcelFile() method to take the subsequence test input from the .xlsx file. We are passing the TestSubSequences to the SubSequences list of type double. After reading all the TestSubSequences we are returning SubSequences.

5. Accuracy Logs: Team used *StreamWriter()* class to create the file. If the file exists, it can be overwritten or appended to. If the file does not exist, this constructor creates a new file. The true flag appends to the file instead of overwriting it. Here we are generating logs for *sequenceKeyPair.Key* and accuracy. Ex. Sequence: 1 is having accuracy 30%

6. Encoder Settings: For encoder settings we have modified the value from 0-99. We have added input validation for the same in our program.cs file.

7. FilePath: For reading inputs, test subsequences and writing accuracy to .xlsx file we are using the relative filePath of the input file, subsequence file and Final Accuracy file. We are using *Environment.CurrentDirectory* property in C# that gets or sets the current working directory of the

application. It returns a string that represents the absolute path of the current working directory.

8. Final Accuracy: We have implemented a logic that writes the final accuracy in .xlsx file along with the current date and time so if we need to check the accuracy in future we can get to know when the file is generated. For that we are using DateTime.Now property in c# that returns the current local date and time on the computer where the code is running. It returns a DateTime object that contains the current date and time.
   string fileName = string.Format("Final Accuracy ({0:dd-MM-yyyy HH-mm-ss}).csv", DateTime.Now);

IV. Implementation

This part of the report explains stages of the experiment. This experiment broadly carried out into two stages.

1. Learning/Training phase
2. Prediction and Accuracy calculation phase.

Learning phase: In learning phase input data sequences are getting passed to *RunExperiment()* method. In *RunExperiment()* method training of input sequences is done using Cortex Layer, Spatial Pooler, Homeostatic Plasticity Controller which checks the stability of spatial pooler. Training of input sequences is required to get the stable state of Spatial pooler. Newborn cycles are generated for each input sequence till the time Spatial pooler reach the stable state. In newborn cycle, compute method of Cortex Layer is getting executed. Once Spatial pooler reaches to stable state the Temporal Memory algorithm is getting activated. At this stage, Spatial Pooler is trained completely. With pretrained SP and HPC, the TM learn cells for patterns.

The figure 2.1 explains how training phase is carried out:

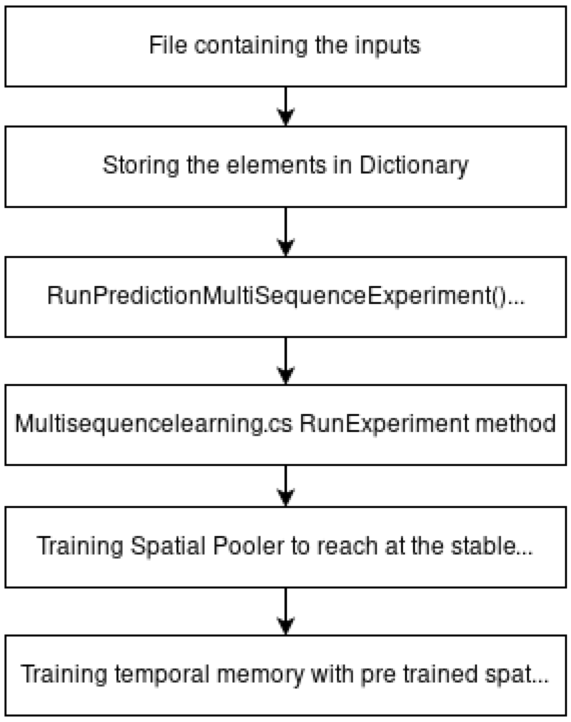

File containing the inputs
Storing the elements in Dictionary
RunPredictionMultiSequenceExperiment()...
Multisequencelearning.cs RunExperiment method
Training Spatial Pooler to reach at the stable...
Training temporal memory with pre trained spat...

Figure 1.1 Training Phase

Prediction and Accuracy calculation phase: *PredictNextElement()* method and Predictor class is used for prediction.
After the learning process, the algorithm returns the instance of Predictor class. This class provides *Predict()* method with a list of input elements.
For every presented input element, the predictor tries to predict the next element.
The more element provided in a sequence the predictor returns with the higher score then model produces a similarity matrix for all the classes.
After prediction of test sequence and next element, accuracy calculation is done.

The figure 2.2 explains how prediction phase is carried out:

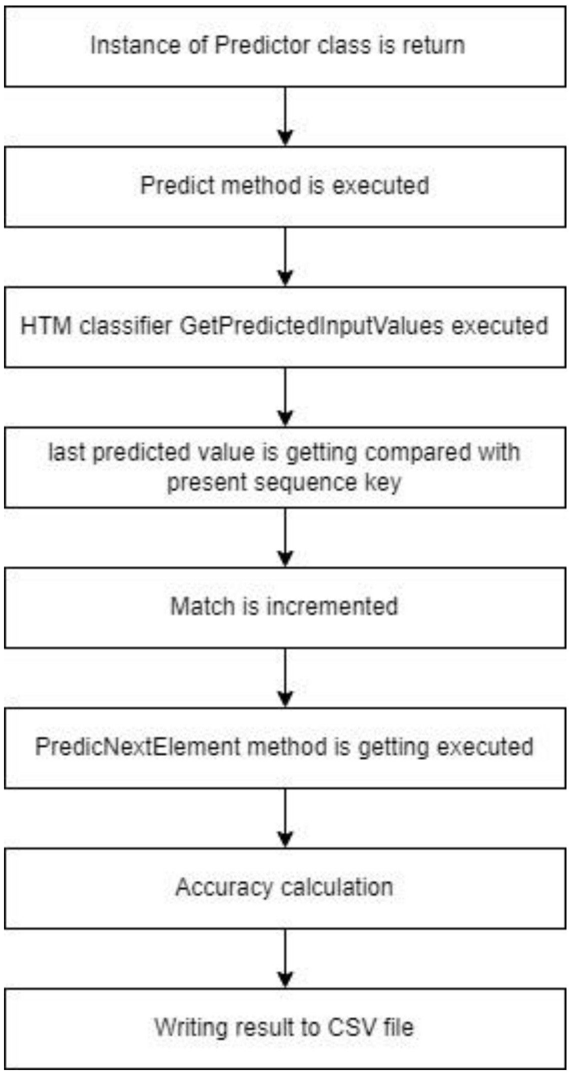

Instance of Predictor class is return
Predict method is executed
HTM classifier GetPredictedInputValues executed
last predicted value is getting compared with present sequence key
Match is incremented
PredicNextElement method is getting executed
Accuracy calculation
Writing result to CSV file

Figure 2.2 Prediction Phase

V. Results

We have divided our result into six sections.

i. Algorithm/Flow chart of the code
ii. Result of training phase
iii. Result of prediction phase
iv. Result of accuracy calculation phase
v. Exporting existing accuracy result into .csv file
vi. Performance testing and comparison with legacy code

I. Flow chart of the code:

Figure 2.3 explains the flowchart of MultiSequenceLearning experiment.

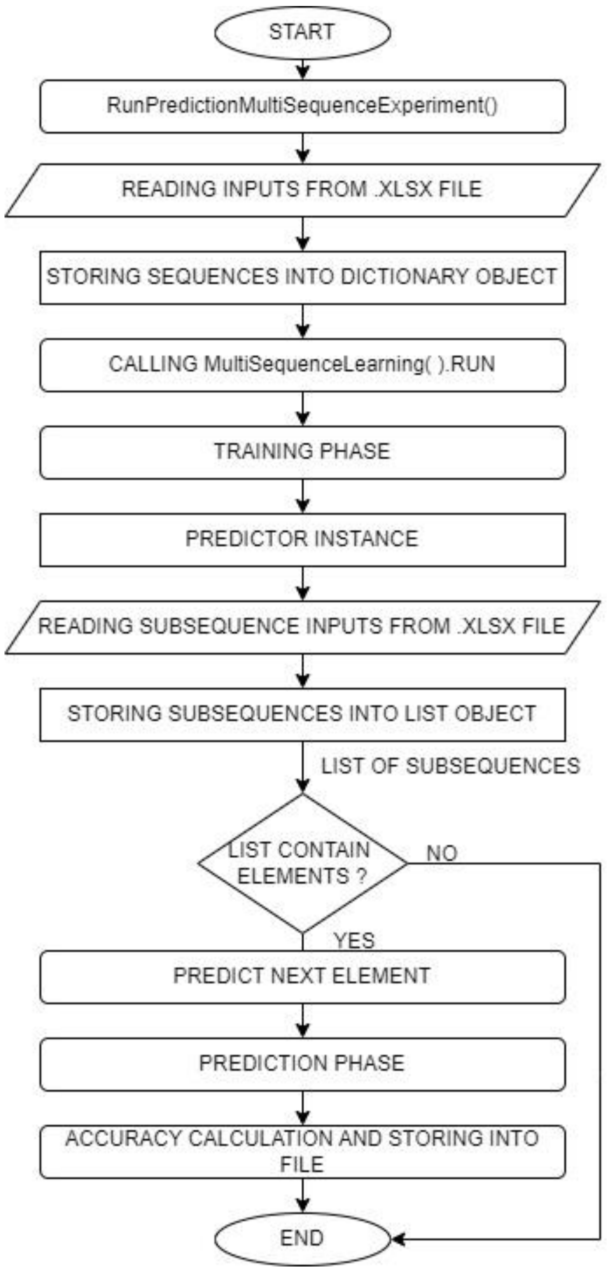
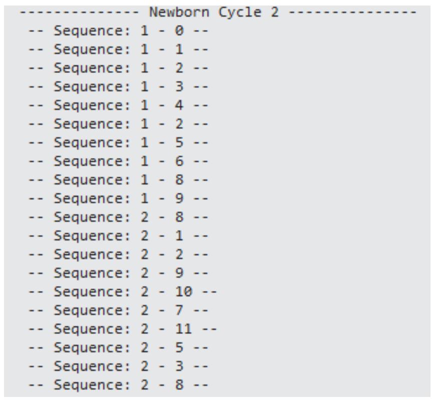

START
RunPredictionMultiSequenceExperiment()
READING INPUTS FROM .XLSX FILE
STORING SEQUENCES INTO DICTIONARY OBJECT
CALLING MultiSequenceLearning( ).RUN
TRAINING PHASE
PREDICTOR INSTANCE
READING SUBSEQUENCE INPUTS FROM .XLSX FILE
STORING SUBSEQUENCES INTO LIST OBJECT
LIST OF SUBSEQUENCES
LIST CONTAIN ELEMENTS ?
NO
YES
PREDICT NEXT ELEMENT
PREDICTION PHASE
ACCURACY CALCULATION AND STORING INTO FILE
END

Figure 2.3 Algorithm of Code.

II. Result of training phase.

Model is getting trained for each input sequence and newborn cycle gets generated till the spatial pooler reach the stable state.



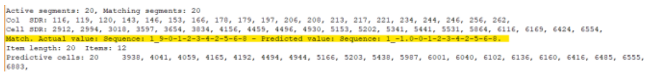
```
-------------- Newborn Cycle 2 ---------------
  -- Sequence: 1 - 0 --
  -- Sequence: 1 - 1 --
  -- Sequence: 1 - 2 --
  -- Sequence: 1 - 3 --
  -- Sequence: 1 - 4 --
  -- Sequence: 1 - 2 --
  -- Sequence: 1 - 5 --
  -- Sequence: 1 - 6 --
  -- Sequence: 1 - 8 --
  -- Sequence: 1 - 9 --
  -- Sequence: 2 - 8 --
  -- Sequence: 2 - 1 --
  -- Sequence: 2 - 2 --
  -- Sequence: 2 - 9 --
  -- Sequence: 2 - 10 --
  -- Sequence: 2 - 7 --
  -- Sequence: 2 - 11 --
  -- Sequence: 2 - 5 --
  -- Sequence: 2 - 3 --
  -- Sequence: 2 - 8 --
```

Figure 2.4 Result of Training Phase.

Figure 2.4 shows the result of training phase where newborn cycles are generated.

III. Result of prediction phase.

Prediction phase is carried out once training phase is completed. Prediction phase executes in three steps match detection, mismatch detection and next element prediction.

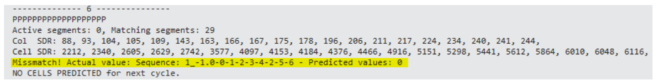


```
Active segments: 20, Matching segments: 20
Col  SDR: 116, 119, 120, 143, 146, 153, 166, 178, 179, 197, 206, 208, 213, 217, 221, 234, 244, 246, 256, 262,
Cell SDR: 2912, 2994, 3018, 3597, 3654, 3834, 4156, 4459, 4496, 4930, 5153, 5202, 5341, 5441, 5531, 5864, 6116, 6169, 6424, 6554,
Match. Actual value: Sequence: 1_9-0-1-2-3-4-2-5-6-8 - Predicted value: Sequence: 1_-1.0-0-1-2-3-4-2-5-6-8.
Item length: 20  Items: 12
Predictive cells: 20     3938, 4041, 4059, 4165, 4192, 4494, 4944, 5166, 5203, 5438, 5987, 6001, 6040, 6102, 6136, 6160, 6416, 6485, 6555,
6883,
```

Figure 2.5 Match detection

If the input sequence and test sequence are matching then predict element will detect a match and match counter is getting incremented. Which we are using for accuracy calculation.

Figure 2.5 is showing the match.



```
-------------- 6 ---------------
PPPPPPPPPPPPPPPPPPP
Active segments: 0, Matching segments: 29
Col  SDR: 88, 93, 104, 105, 109, 143, 163, 166, 167, 175, 178, 196, 206, 211, 217, 224, 234, 240, 241, 244,
Cell SDR: 2212, 2340, 2605, 2629, 2742, 3577, 4097, 4153, 4184, 4376, 4466, 4916, 5151, 5298, 5441, 5612, 5864, 6010, 6048, 6116,
Missmatch! Actual value: Sequence: 1_-1.0-0-1-2-3-4-2-5-6 - Predicted values: 0
NO CELLS PREDICTED for next cycle.
```

Figure 2.6 Mismatch detection

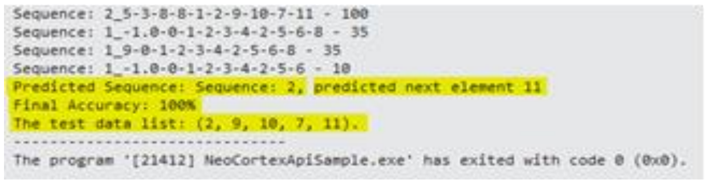
If any element of the input sequence does not match with the test data predictor will return a mismatch. The figure 2.6 is showing the mismatch.

IV. Result of Accuracy Calculation Phase

Accuracy is calculated by dividing matches with total number of sequenceKeyPair multiply by 100.

Accuracy = count of matches / total number of predictions * 100

By performing this calculation, we are getting accuracy in percentage. This accuracy we are storing into the csv file and also printing this information into the debug window as shown in the fig.2.7.



```
Sequence: 2_5-3-8-8-1-2-9-10-7-11 - 100
Sequence: 1_-1.0-0-1-2-3-4-2-5-6-8 - 35
Sequence: 1_9-0-1-2-3-4-2-5-6-8 - 35
Sequence: 1_-1.0-0-1-2-3-4-2-5-6 - 10
Predicted Sequence: Sequence: 2, predicted next element 11
Final Accuracy: 100%
The test data list: (2, 9, 10, 7, 11).
--------------------------------
The program '[21412] NeoCortexApiSample.exe' has exited with code 0 (0x0).
```

Figure 2.7 Accuracy Calculation

V. Exporting existing accuracy result into .csv file

In MultiSequenceLearning.cs class accuracy is getting calculated by using formula

double accuracy = (double)matches / (double)sequenceKeyPair.Value.Count * 100.0;

This accuracy result we are exporting into csv file which is available in MySeProject. [8]

VI. Performance testing and comparison with legacy code.

In this phase of testing, we tested the code for 10 iterations with different range of training data and test data and compare the performance of newly implemented the code and legacy code. Result can be found in attached excel sheet. [9]

VI. CONLUSION

This project described that the Neocortex API's Multi Sequence learning reference model was used to perform multi-sequence learning for a sequence of numerical data sets. The HTM Prediction Engine was adjusted with various parameters to optimize the training process, and the data was transformed into an encoded value and stored in a dictionary using an Encoder and SDR input for training.

Team has developed new method that read learning sequences from a file by itself and learn them. After that the sample should read testing subsequences from another file and calculate the prediction accuracy.

Different methods are created to predict the trained sequences by comparing the generated similarity matrix with each of the SDRs of the learned sequence from the training phase, and the accuracy percentage of the predicted sequences was calculated and stored in a file. The accuracy of the predictions was found to increase with the number of cycles, indicating that the learning process improved over time.

This project provided insights into various aspects of the Neocortex API, including the functioning of encoders, how the Spatial Pooler generates SDR inputs and performs the learning phase in stabilizing the learning process.

VII. REFERENCES

[1] Cortical.io. [Online]. Available: https://www. cortical.io/science/sparse-distributed-representations/.

[2] S. A. a. J. H. Y. Cui, "The HTM Spatial Pooler —A Neocortical Algorithm for Online Sparse Distributed Coding," Vols. 11-2017, November 2017,.

[3] S. A. a. J. H. Yuwei Cui, "The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding," 29 November 2017. [Online]. Available:https://www.frontiersin.org/articles/10.3389/fncom.2017.00111/full.

[4] A. G. B. W. T. D. D. Pech, "Scaling the HTM Spatial Pooler,"*International Journal of Artificial Intelligence & Applications,* vol. 11, 2020.

[5] [Online].Available: https://github.com/pparaska/neocortexapi_Team_MSL/blob/Team_MSL/source/MySEProject/Accuracy_Testing.xlsx.

[6] [Online].Available: https://github.com/pparaska/neocortexapi_Team_MSL/blob/Team_MSL/source/MySEProject/Performance%20Testing.xlsx.

[7] P. Paraskar and P.Desai.[Online].Available: https://github.com/pparaska/neocortexapi_Team_MSL/blob/Team_MSL/source/MySEProject/Performance%20Testing.xlsx.

[8] "https://github.com/pparaska/neocortexapi_Team_MSL/blob/Team_MSL/source/MySEProject/Accuracy_Testing.xlsx," [Online].

[9] "https://github.com/pparaska/neocortexapi_Team_MSL/blob/Team_MSL/source/MySEProject/Performance%20 Testing.xlsx," [Online].