Software Technology
by Damric Dobric / Andreas Pech

Image Classification With Spatial Pooler

Mandeep Singh (1228409)
Mandeepsingh.sherry@gmail.com

***Abstract*-- This paper presents an emerging machine learning algorithm Hierarchical Temporal Memory (HTM) which is using Spatial pooler for learning and classifying visual data. The SP models how neurons learn feedforward connections and form efficient representations of the input. It converts arbitrary binary input patterns into sparse distributed representations (SDRs), using a combination of competitive Hebbian learning rules and homeostatic excitability control, and we also describe a key parameters of the SP.**

Keywords: Hierarchical Temporal Memory, spatial pooler, sparse distributed representations.

1. INTRODUCTION

The HTM (Hierarchical Temporal Memory) spatial pooler incorporates several computational principles of the cortex. It relies on competitive Hebbian learning, homeostatic excitability control, topology of connections in sensory cortices, and activity-dependent structural plasticity. The HTM spatial pooler is designed to achieve a set of computational properties that support computations with SDRs (Sparse Distributed Representations)**[1]**. These properties include (1) preserving topology of the input space by mapping similar inputs to similar outputs, (2) continuously adapting to changing statistics of the input stream, (3) forming fixed sparsity representations, (4) being robust to noise, and (5) being fault tolerant. As an integral component of HTM, the outputs of the SP can be easily recognized by downstream neurons and contribute to improved performance in an end-to-end HTM system.

The related works in this area of forecasting that have used in Machine Learning (ML) or statistical modelling, the emphasis here is to enable the reader to understand on some of the various ML or statistical techniques actively in use in the times past and till this present moment.

2. METHODOLOGY

The SP is a core component of HTM networks (Figure 1). In an end-to-end HTM system, the SP transforms input patterns into SDRs in a continuous online fashion. The HTM temporal memory learns temporal sequences of these SDRs and makes predictions for future inputs. A single layer in an HTM network is structured as a set of mini-columns, each with a set of cells (Figure 2). The HTM neuron model incorporates dendritic properties of pyramidal cells in neocortex, where proximal and distal dendritic segments on HTM neurons have different functions (Figure 3). Patterns detected on proximal dendrites lead to action potentials and define the classic receptive field of the neuron. Patterns recognized by a neuron's distal synapses act as predictions by depolarizing the cell without directly causing an action potential.

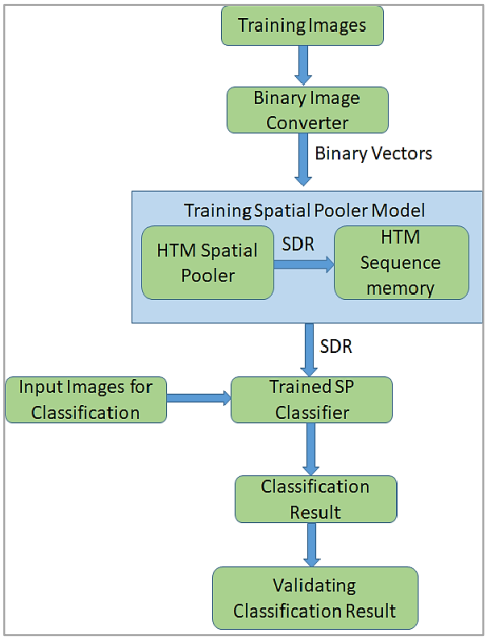

Figure 1. Flow chart of Image classification model

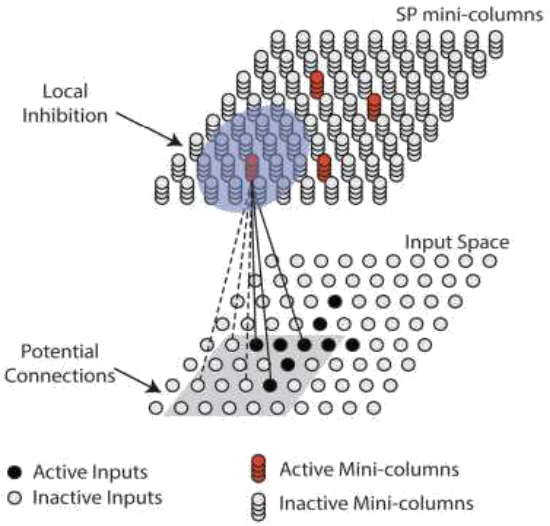

Figure 2. The illustration of spatial pooling.[3]

The HTM spatial pooler converts inputs (bottom) to SDRs (top). Each SP mini-column **[2]** forms synaptic connections to a subset of the input space (gray square, potential connections). A local inhibition mechanism ensures that a small fraction of the SP mini-columns that receive most of the inputs are active within the local inhibition radius (shaded blue circle). Synaptic permanences are adjusted according to the Hebbian rule: for each active SP mini-column, active inputs (black lines) are reinforced and inactive inputs (dashed lines) are punished.

3. Mathematical Formulation [3]

In HTM, spatial pooling is performed using the notion of SDRs followed by competitive Hebbian learning rules, a Homeostatic excitability control, and an overlapping mechanism for deriving candidate or winner SDR patterns via inhibition. SDRs are formed by activating or deactivating a set of potential synapses or connecting neural links. These synapses are grouped into a set of mini-columns and are spread out in a hypercube based on a set of predefined rules.

Initialize connections:
Consider a group of mini-columns with a set of potential connecting logical synapses or neurons; these potential connections may be initialized accordingly as:

$$\prod_i = \{j \mid I(X_j ; X_i^c, \gamma) \text{ and } Z_{ij} < p\}$$

where,
j = HTM neuron location index in the mini-column.
i = mini-column index.
x_j = location of the jth input neuron (synapses) in the input space.
x^c_i = location centre of potential neurons (synapses) of ith mini-column in a hypercube of input space. γ = edge length of x_j.
ρ = fraction of inputs within the hypercube of input space that are potential connections.
Z_{ij} = represents a uniformly distributed random number between 0 and 1.
I = an indicator function.

Permanence Activation Rule:
A set of connected synapses are described by a binary matrix.
W, which is formulated by conditioning the synapses to a permanence activation rule as:

$$W_{ij} = \begin{cases} 1 & if\ Dij \geq \theta c \\ 0 & otherwise \end{cases}$$

where,
D_{ij} = independent and identically distributed (i.i.d) dendrite synaptic permanence values from the jth input to the ith mini-column,
θ_c = synaptic permanence threshold

$$D_{ij} = \begin{cases} U(0,1)\ if\ j \in \prod i \\ 0\ otherwise \end{cases}$$

Where a natural topology exists, neighbourhood mini-columns may be inhibited in accordance to the relation given , otherwise a global inhibition parameter is simply used.

$$N_i = \{ j \mid \| Yi - Yj \| < \emptyset , j \neq i \}$$

where,
y_i = is the ith HTM-SP mini-column,
y_j = is the jth HTM-SP mini-column,
i,j = mini-column indexes.

Computing Overlap:
For creating associations with input patterns, feed-forward inputs to each of the generative spatial mini-columns are computed using a matching technique called the overlap. The overlap is computed as:

$$O_i = b_i \sum_j WijZj$$

Where,
bi = is a positive boost factor for exciting each HTM-SP mini-column.
zj = input pattern vector seen by the generative HTM neuron

SP mini- column Activation rule:
SP mini-column becomes active if the feedforward input is above a stimulus threshold θstim and is among the top s percent of its neighbourhood,

$$a_i = \begin{cases} 1\ if Oi \geq Z(Vi, 100 - s) and\ oi \geq \theta stim \\ 0\ otherwise \end{cases}$$

Where,
s = target activation density (sparsity).
Z = a percentile function.
θ_{stim} = a stimulus threshold.

Finally, boost updating in HTM-SP follows the homeostatic excitability control mechanism comparable to that observed in cortical neurons. Boosting is accomplished in HTM-SP using the following model equations.

$$\overline{ai}\,(t) = \frac{(T-1) * \overline{ai}(t-1) + ai(t)}{T}$$

$$< \overline{ai}\,(t) > = 1/|\mathrm{Ni}| \sum_{j \in Ni} \overline{aj}\,(t)$$

$$\mathrm{bi} = e^{-\beta(\overline{ai}\,(t) - < \overline{ai}\,(t) >)}$$

where, ai = time averaged activation over the last T SDR inputs,
T = an integer number denoting the number of Monte Carlo trials to obtain a reasonable activation estimate, $a_i t$ = the current activity of the *i*th mini-column at time step *t*.
β = a positive parameter that controls the strength of the adaptation effect.
As mentioned in, such calculations have been used in previous models of homeostatic synaptic plasticity.

4. Code for Testing Algorithm

Step 1: Project Structure
In this project we have different libraries**[5]**, classes with different specifications.

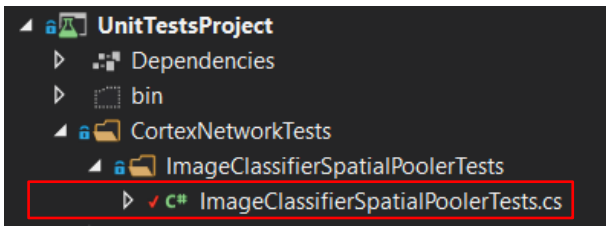

Figure 3: UnitTestProject

In the UnitTestProject we have main unit test method in ImageClassificationSpatialPoolerTest.cs which gives the information about Implementation of this project.

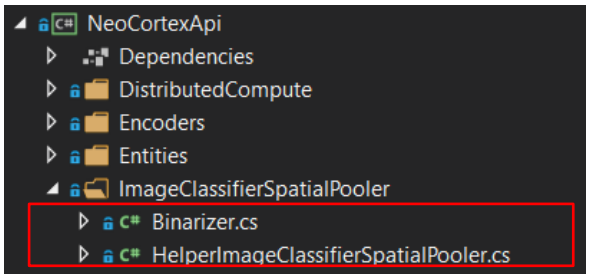

Figure 4: Helping classes

Helping classes like Binarizer.cs and HelperImageClassificationSpatialPooler.cs have methods defined which are used in UnitTestProject class.

Step 2: Defining input and output parameters of Images.

```
// Input Image parameters
var imageWidth = 50;
var imageHeight = 50;

// Output images parameter
int outputWidth = 100;
int outputHeight = 100;
```
Figure 5: input and output parameters of Images.

Step 3: Setting input space dimensions and Column space dimensions.

```
parameters.setInputDimensions(new int[] { imageWidth, imageHeight });
parameters.setColumnDimensions(new int[] { outputWidth, outputHeight });
```
Figure 6: input space dimensions

Step 4: Setting the parameters of Potential connection.

```
parameters.setNumActiveColumnsPerInhArea(0.02 * outputWidth * outputHeight);
```
Figure 7: parameters of Potential connection**.**

Step 5: TrainingImagePath is the image path contains images file name.

```
String[] trainingImagesPath = Directory.GetFiles(Path.Combine(Directory.GetCurrentDirectory(),
    "TrainedImages"), "*.jpg");
```
Figure 8: Training Image Path

Step 6: Training Data

a) In ReadImageData method reads the data of training image in binary form.

```
int[] activeArray = new int[outputWidth * outputHeight];
int[] inputVector = ReadImageData(trainingImagePath, imageHeight, imageWidth);
int[] newActiveArray = new int[outputWidth * outputHeight];
sp.compute(inputVector, activeArray, true);
var activeCols = ArrayUtils.IndexWhere(activeArray, (el) => el == 1);
int[] oldActiveArray = activeArray;
```
Figure 9: Training Process

b) So, Input vector having the data of images in binary form.
c) Compute function contains input vector which has input patterns that shows a large number of data, grouped together into a common output representation that is active array.
d) Active column represent the activation of mini column, which have active input =1.
e) Training Images as below Figure 10.

Figure 10: Training Data images containing different types of apples [9] [10] [11] [12]

Step 7: GetHammingDistance [6]

This method used to compare, the reference array which we have to compare with array which is given.

```
public int GetHammingDistance(int[] referenceArray, int[] givenArray)
{
    int unmatchedIndex = 0;

    for (int i = 0; i < referenceArray.Length; i++)
    {
        if (referenceArray[i] != givenArray[i])
        {
            unmatchedIndex++;
        }
    }

    return unmatchedIndex;
}
```

Figure 11: Hamming Distance Function

Step 8: Deciding Model Trained or not:

The GetHammingDistance method is in the loop, this is how the model of common receptive field is learned from the input. In this loop, defined new active array always gives value to old active array till then model is not trained.

```
while (isTrained == 0)
{
    sp.compute(inputVector, newActiveArray, true);
    activeCols = ArrayUtils.IndexWhere(newActiveArray, (el) => el == 1);

    if (GetHammingDistance(oldActiveArray, newActiveArray) == 0)
    {
        isTrained = 1;
    }
    else
    {
        isTrained = 0;
        oldActiveArray = newActiveArray;
    }
}
```

Figure 12: Training Model

Step 9: Input Images for Classification Prediction

a) Prediction **[7]** follows the same procedure as training with same compute function.

```
int[] inputVectorPrediction = ReadImageData(predictionImagePath, imageHeight, imageWidth);
int[] activeArrayPrediction = new int[outputWidth * outputHeight];
sp.compute(inputVectorPrediction, activeArrayPrediction, false);
var activeColsPrediction = ArrayUtils.IndexWhere(activeArrayPrediction, (el) => el == 1);

var strPrediction = Helpers.StringifyVector(activeColsPrediction);

Debug.WriteLine("Active Column: "+strPrediction);
```

Figure 13: Prediction Process

b) Prediction Images **[8]**

Figure 14: Input images for Prediction [13] [14] [15] [16]

Step 10: Converting Prediction Images into Binary

a) Reading Data from binary images to identify the prediction and active column

```
public int[] ReadImageData(String imagePath, int height, int width)
{
    Binarizer bizer = new Binarizer(targetHeight: height, targetWidth: width);

    bizer.CreateBinary(imagePath, Path.Combine(Path.Combine(Directory.GetCurrentDirectory(),
        $"OutputBinarizedImages"), $"bin-{Path.GetFileName(imagePath)}.txt"));

    var binaryString = bizer.GetBinary(imagePath); // we will get binary string from it
```

Figure 15: Binary Conversion

b) It returns binarized image in integer array and creates file of binarized Image

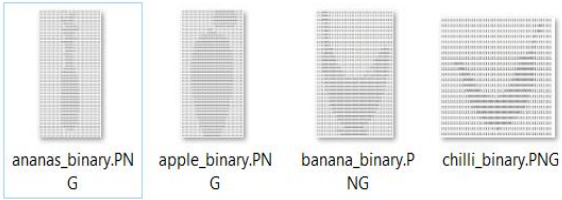

Figure 16: Binary Images

Step 11: Computing Prediction Percentage

```
public double GetMatchingPercentage(int[] trainedArray, int[] predictionArray)
{
    var activeColsTrained = ArrayUtils.IndexWhere(trainedArray, (el) => el == 1);

    var activeColsPredicted = ArrayUtils.IndexWhere(predictionArray, (el) => el == 1);

    var intersection = activeColsTrained.Intersect(activeColsPredicted).ToArray();

    double percentage = (double)intersection.Length * 100 / activeColsPredicted.Length;

    return percentage;
}
```

Figure 17: Computing Prediction Percentage

GetMatchingPercentage method compares the active column of trained Images and active column of Prediction Images. Percentage of Intersection of these active columns gives us a result.

```
var strPrediction = Helpers.StringifyVector(activeColsPrediction);
Debug.WriteLine("Active Column: "+strPrediction);
double predictionPercentage = GetMatchingPercentage(activeArrayTrained, activeArrayPrediction);
```

Figure 18: Percentage Matching

Our Model is trained for Apple because we are giving different Images of Apple Only. So, it gives more percentage prediction towards Apple image rather than other images.

The Prediction percentage can be calculated by

```
double percentage = (double)intersection.Length * 100 / activeColsPredicted.Length;
```

If the Percentage Prediction is more than or equal to 90%, then it is apple and if it is less than 90% then it is not an apple.

```
Debug.WriteLine("prediction percentage related to trained data:{0}", predictionPercentage);
if (predictionPercentage >= 90.00)
{
    Debug.WriteLine("Result: It is apple");
}
else
{
    Debug.WriteLine("Result: It is not Apple");
}
Debug.WriteLine("----------------------------------------------");
```

Figure 19: Classification Result

Step 12: Output of Classification Prediction

If we give an image of ananas then the prediction percentage related to trained data is less than 90%, so it is telling us it is not an apple.

```
Name of image for Prediction : ananas.jpg
Active Column: 4, 8, 14, 17, 24, 28, 29, 30, 34, 41, 44, 45, 46, 47,
prediction percentage related to trained data:72.95373665480427
Result: It is not Apple
----------------------------------------------
```

Figure 20: Output for Ananas

If we give an image of an apple then the prediction percentage related to trained data is more than or equal to 90% , so it is telling us it is Apple.

```
Name of image for Prediction : apple.jpg
Active Column: 4, 8, 14, 17, 24, 28, 29, 30, 34, 41, 44, 45, 46, 47,
prediction percentage related to trained data:90.00884173297966
Result: It is apple
----------------------------------------------
```

Figure 21: Output for Apple

If we give a image of banana then the prediction percentage related to trained data is less than 90%, so it is telling us it is not Apple.

```
Name of image for Prediction : banana.jpg
Active Column: 4, 8, 14, 17, 24, 28, 29, 30, 34, 41, 44, 45, 46, 47,
prediction percentage related to trained data:77.37556561085972
Result: It is not Apple
----------------------------------------------
```

Figure 22: Output for Banana

If we give an image of Chilli then the prediction percentage related to trained data is less than 90% , so it is telling us it is not Apple.

```
Name of image for Prediction : chilli.jpg
Active Column: 4, 8, 14, 17, 24, 28, 29, 30, 34, 41, 44, 45, 46, 47,
prediction percentage related to trained data:67.26786907147628
Result: It is not Apple
----------------------------------------------
```

Figure 23: Output for Chilli

5. Testing Approach

Scope of the Testing: SP Model is been trained using four different types of apple images as explained in following steps (1 to 4). Then 4 different images like Banana, Ananas, Apple and Chilli are given as an input to SP model for classification as explained in step 5. Expected result is for Apple, model should give more than 90 % as Prediction Percentage related to trained data i.e. Classifying it as an Apple and on the other hand , for Banana, Chilli and Ananas model should give less than 90 % Prediction Percentage related to trained data i.e. classifying it as not an Apple. The result of this test is depicted in table below Table 1.

5.1 Training image

Training is using four different types of Apples as shown in figure fig. 4.1.Considering **apple2.png** for further explanation.

Figure 24: apple2.png [10]

5.2 Converting input image into binary image

```
var binaryString = bizer.GetBinary(imagePath);
```

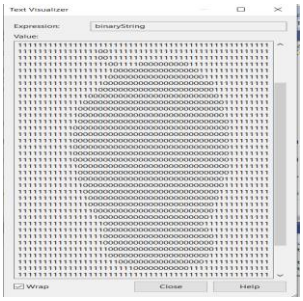

Figure 25: Binary image of apple2.png

5.3 Generating active column

```
var str = Helpers.StringifyVector(activeCols);
```

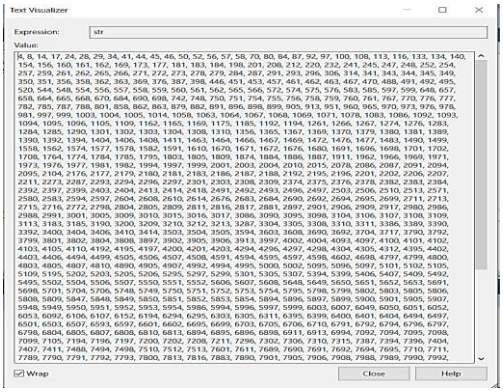

Figure 26: Active Column of apple2.png

5.4 Training completed

When Hamming Distance is zero that indicate modal is trained

```
if (GetHammingDistance(oldActiveArray, newActiveArray) == 0)
```

5.5 Giving input images for classification

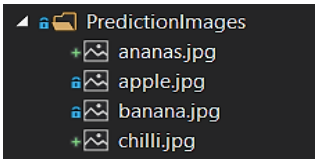

Figure 27: Input images for classification

5.6 Image Classification Prediction

Further testing is done using more images for classification e.g. Ananas, Apple, banana, Chilli. In order to classify whether input image is apple or anything other than apple. Model is classifying input image as an Apple if Prediction percentage is more than 90% and not an apple otherwise. Table 1 summarizes corresponding Binary Image, Active column, Prediction percentage, and classification result for each given input image like Ananas, Apple, Banana and Chilli. Result for each input image is depicted as below in Table 1.

Table 1: Summary of prediction for images

Actual input image for classification	Expected Result	Binary Image	Active column	Prediction Percentage	Classification result
Ananas	< 90 %			72.9537%	It is not Apple
Apple	> =90 %			90.0088%	It is Apple
Banana	< 90 %			77.3755%	It is not Apple
Chilli	< 90 %			67.2678%	It is not Apple

6. Result

For evaluating a classifier quality and accuracy, we are using Confusion Matrix [17]. A confusion matrix is a summary of prediction results on a classification problem, in which number of correct and incorrect predictions are summarized with count values and broken down by each class. For constructing Confusion matrix for above 4 input images as shown in Figure 14, Above test case has been executed for 10 times and getting following result (10 Test run) .Table 2 explains Prediction result for each input image e.g. Apple, Ananas Banana ,Chilli.

Table 2: Result of 10 Test Runs

Test Run Nr.	Input Image	Expected Result	Predicted Result	
1	Apple	> =90 %	90.00881326	Apple
2	Banana	< 90 %	77.37551245	Not Apple
3	Ananas	< 90 %	72.95373444	Not Apple
4	Chilli	< 90 %	67.26332167	Not Apple
5	Apple	> =90 %	90.00884173	Apple
6	Banana	< 90 %	77.37556561	Not Apple
7	Ananas	< 90 %	72.95373665	Not Apple
8	Chilli	< 90 %	67.26786907	Not Apple
9	Banana	< 90 %	76.6599086	Not Apple
10	Chilli	< 90 %	67.36332167	Apple
11	Ananas	< 90 %	72.95389367	Apple
12	Apple	> =90 %	89.89000432	Not Apple
13	Apple	> =90 %	91.89760065	Apple
14	Apple	> =90 %	90.23454432	Apple
15	Apple	> =90 %	92.34543211	Apple
16	Apple	> =90 %	90.41255099	Apple
17	Apple	> =90 %	90.43255678	Apple
18	Apple	> =90 %	90.34213394	Apple
19	Apple	> =90 %	90.44500982	Apple
20	Banana	< 90 %	77.6599086	Not Apple

Based on above 10 test runs result, Confusion matrix has prepared as shown below, Table 3 summarizes result of 10 test runs as shown in Table 2. First row indicates result for total 10 Apple images. For 10 actual Apple images, SP model correctly classify it as an Apple 9 times and fail to classify correctly 1 time. Second row indicates, For total 10 not-Apple images including Ananas, Banana and Chilli, SP model classifies all 10 correctly as Not-Apple.

Table 3: Confusion Matrix [17]

	Predicted Classification Result			
Actual Images for Prediction		**Apple**	**Not Apple**	**Total**
	Apple	9	1	10
	Not Apple	0	10	10
	Total	**9**	**11**	**20**

From this Confusion matrix, Accuracy of Classification can be determined. Classification accuracy is the ratio of correct predictions to total predictions made as follows,

$$\text{Accuracy} = \frac{(\text{Total Predicted Apple} + \text{total Predicted Not Apple}) * 100}{\text{Total Predictions made}}$$

$$\text{Accuracy} = \frac{(9 + 10) * 100}{20} = 95\ \%$$

We are getting Classification accuracy 95 % based on 10 test runs.

7. Conclusion:

In this paper, we followed testing approach by training SP model with 4 different types of Apples as a training dataset and testing classification capability of SP model based on 4 images including apple and non-apple images as testing dataset. It is observed that for apple image, SP model is predicting maximum percentage (> 90 %) and for non- apple images it predict comparatively less percentage. It is noticed that for images looking more like apple (Training data) has comparatively more prediction percentage e.g. ananas has more prediction Percentage than chilli.

From above results, we can conclude that SP model has Classification accuracy of 95% when 10 test runs are been carried out.

References:

[1] Kirtay, Murat, Lorenzo Vannucci, Ugo Albanese, Alessandro Ambrosano, Egidio Falotico, and Cecilia Laschi. "Spatial pooling as feature selection method for object recognition." In *ESANN*. 2018.

[2] Osegi, E. N. "Using the hierarchical temporal memory spatial pooler for short-term forecasting of electrical load time series." *Applied Computing and Informatics* (2018).

[3] W. Zhuo, Z. Cao, Y. Qin, Z. Yu and Y. Xiao, "Image classification using HTM cortical learning algorithms," Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), Tsukuba, 2012, pp. 2452-2455.

[4] http://nupic.docs.numenta.org/0.6.0/spatial-pooler.html

[5] D. Dobric, "NeoCortexApi," 2019. [Online]. Available:https://github.com/ddobric/neocortexapi/.

[6] https://github.com/UniversityOfAppliedSciencesFrankfurt/se-cloud-2019-2020/blob/Mandeep_Singh/Source/HTM/UnitTestsProject/CortexNetworkTests/ImageClassifierSpatialPoolerTests/ImageClassifierSpatialPoolerTests.cs

[7] https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/image-classification

[8] Mattsson, Olov. "Fruit recognition by hierarchical temporal memory." (2011).

[9] https://www.istockphoto.com/de/foto/roter-apfel-mit-blatt-gm683494078-125451195

[10] https://www.pexels.com/photo/healthy-apple-fruits-natural-102104/

[11] https://www.istockphoto.com/de/foto/roter-apfel-mit-blatt-isoliert-auf-wei%C3%9Fem-hintergrund-gm185262648-19966694

[12] https://i5.walmartimages.ca/images/Enlarge/094/514/6000200094514.jpg

[13] https://d2lnr5mha7bycj.cloudfront.net/product-image/file/large_fda9f2e7-b6f4-4fc6-bc27-355eece8b1fd.jpg

[14] https://image.shutterstock.com/image-photo/red-apple-on-white-background-600w-158989157.jpg

[15] https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRXe6A1vQhf-ZiW-zZ0o7W204cR_mRR80G52Mh_Wj57tmEe2OIL&usqp=CAU

[16] https://image.shutterstock.com/image-vector/realistic-green-hot-natural-chili-260nw-1027834957.jpg

[17] Arora, R., 2012. Comparative analysis of classification algorithms on different datasets using WEKA. *International Journal of Computer Applications*, *54*(13).