# Hierarchical Temporal Networks for Anomaly Detection in Videos

*A subtitle of your thesis*

Vladimir Monakhov


Thesis submitted for the degree of
Master in Informatics: Robotics and Intelligent Systems
60 credits

Department of Informatics
The Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2022

# Hierarchical Temporal Networks for Anomaly Detection in Videos

*A subtitle of your thesis*

Vladimir Monakhov

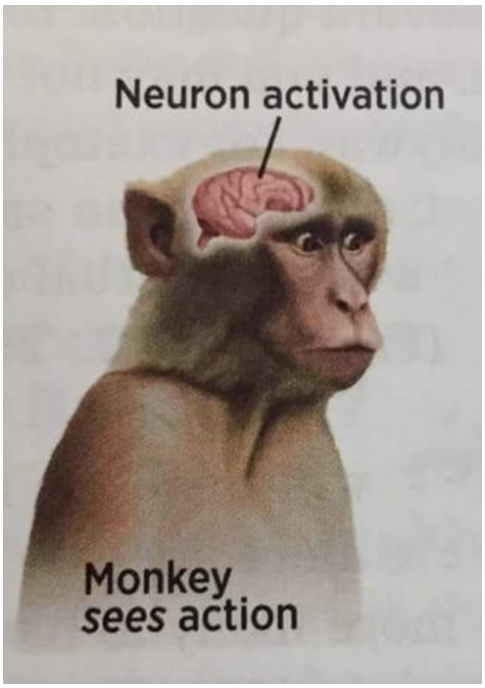
Hierarchical Temporal Networks for Anomaly Detection in Videos

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

# Abstract

Bla bla bla...



# Contents

# Chapter 1

# Introduction

## 1.1 Background and Motivation

As the global demand for security and automation increases, many seek to use video anomaly detection systems. In the US alone, the surveillance market is expected to reach $23.60 Billion by 2027 [1]. Leveraging modern computer vision, modern anomaly detection systems play an important role in increasing monitoring efficiency and reducing the need for expensive live monitoring. Their use cases can vary from detecting faulty products on an assembly line or detecting car incidents on the highway, and everything in between.

The most important component in video anomaly detection systems is the intelligence behind it. The intelligence ranges from simple on-board algorithms to dedicated servers hosting complex deep learning models, where the latter has seen increased popularity the past few years.

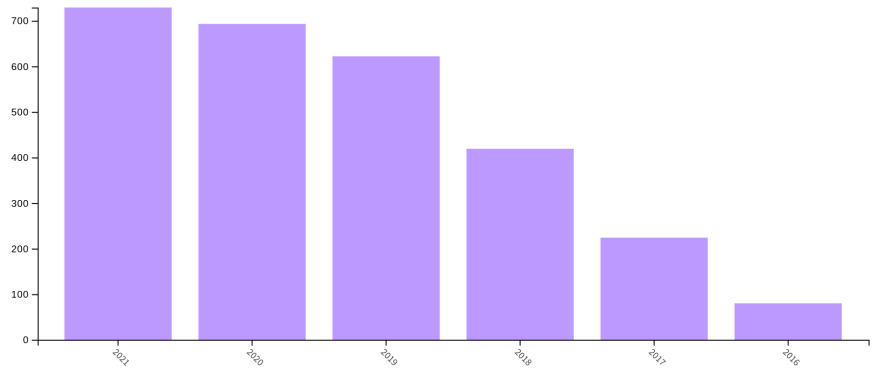



Figure 1.1: The increase in publications mentioning the terms "deep learning" and "surveillance". [2]

Yet despite the major progress within the field of deep learning, there

are still many tasks where humans outperform models, especially in anomaly detection where the anomalies are often undefined. Deep learning approaches also perform poorly when dealing with noise and concept drift.

The cause for the discrepancy lies in the difference between how humans and machine learning algorithms represent data and learn. Most machine learning algorithms use a dense representation of the data and apply back-propagation in order to learn. Human learning happens in the neocortex, where evidence points to that the neocortex uses a sparse representation and performs Hebbian-style learning. For the latter, there is a growing field of machine learning dedicated to replicating the inner mechanics of the neocortex, namely Hierarchical Temporal Memory (HTM) theory. This theory outlines its advantages over standard machine learning, such as noise-tolerance and the ability to adapt to changing data.

With the advantages of HTM and the rise of video anomaly detection in mind, a natural question one could pose is whether it is possible to apply HTM for anomaly detection in videos. Combined with a lack of related works, it is this very question that is the motivation behind this thesis.

## 1.2 Problem Statement

Based on the background and motivation, the problem statement can be boiled down to a simple question: **Is HTM viable for use in video anomaly detection?**

This thesis will introduce three different experiments that will help answer the question and also showcase the performance of HTM. These experiments will vary in difficulty, complexity, and will focus on different use cases. This thesis will also cover all required knowledge. To summarize, this thesis will cover three objectives:

1. Introduce HTM and give a deep understanding of the inner workings, the strengths, and the weaknesses. While also being friendly to readers with a machine learning background.
2. Develop and outline a theoretically sound pipeline so that HTM can be applied for anomaly detection in complex videos.
3. Perform experiments, discuss the results, and lay out potential future work.

## 1.3 Limitations

HTM is a complex topic not part of the curriculum in most educations, if any at all. It is also based on neurological research, lending terms and concepts from the biological field, which significantly raises the level of entry for people with a machine learning background. This makes learning and understanding HTM a process which takes up a sizable chunk out of

the total time spent on the thesis. This thesis will therefore mainly focus on only one approach, leaving other approaches for other papers.

Additionally, HTM for video anomaly detection is a novel topic and is therefore naturally limited on several fronts. One of the main limitations is the lack of labeled anomaly data that suits the nature of HTM, because most datasets are made for use with deep learning approaches. Another problem is the lack of works related to applying HTM on video-based problems. Finally, while there are other methods that can be used for video anomaly detection, none of them are based on the same premises as HTM. This means that there is a major lack of methods to use for the purpose of benchmarking.

Last but not least, the HTM theory described in this thesis is not the first generation, it is actually the third generation which builds upon the second generation. The first generation had fundamentally different inner workings [3], but shared a lot of the terms with the current generation. This has made researching HTM challenging as there are many research papers published that refer to the first generation.

## 1.4 Contributions

This thesis contributes in multiple ways. Not only does it present a novel way to apply HTM on video-based problems, it also uncovers the reasoning behind the design decisions that were made as well as providing thorough analysis. This thesis also acts as an organization of HTM related research backed up with visualizations and a simpler language, making it easier for people with a machine learning background to understand.

During the writing of this thesis, contributions have been made to the HTM community in the form of uncovering and reporting a bug related to the technical implementation of HTM [4]. TODO.

## 1.5 Thesis Outline

Thesis summary goes here.

# Chapter 2

# Background

The following is relevant background information on HTM Theory, Deep Learning, and Video Anomaly Detection.

## 2.1 Hierarchical Temporal Memory

Today's machine learning algorithms aim to solve complex problems by simulating a substantial amount of mathematically defined neurons. These neurons are vastly simplified compared to the neurons in the brain and therefore do not have the complexity required to solve complex problems with an accuracy and level of generalizability comparable to the brain. [5] introduces HTM theory which aims to outline a machine learning algorithm which works on the same principles as the brain and therefore solves some of the aforementioned issues.

The brain consists of layers that have been added throughout evolution. The inner layers are responsible for primal intelligence such as hunger, sex and instincts. HTM theory specifically aims to simulate the neocortex which is the outer layer of the brain tasked with advanced logic. It is important to note that HTM only attempts to estimate the activity in the brain, unlike Spiking Neural Networks and others which aim to accurately simulate the activity of the brain [6].

### 2.1.1 Structure

HTM aims to replicate the structure of the neocortex which is made up of cortical regions. Cortical regions consist of cortical columns, where each column is divided into layers height-wise. These cortical columns are made up of mini-columns, which in turn are made up of neurons.

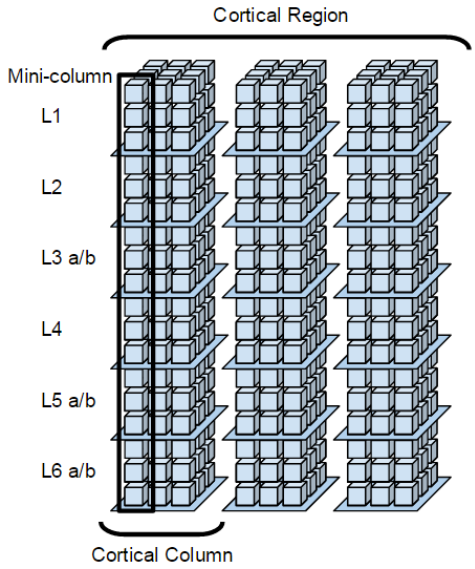



Figure 2.1: Visualization of the HTM Cortical Region structure [7]. Note that current HTM implementations only model layer 2/3.

Neurons in HTM Theory are different from neurons in traditional machine learning. The term neuron in traditional machine learning is very misleading and since it is mathematically derived, has actually very little in common with a biological neuron. A biological neuron does not perform back propagation but learns by strengthening and weakening inter-neural connections (synapses), which is something that the HTM neuron attempts to model through Hebbian like learning.

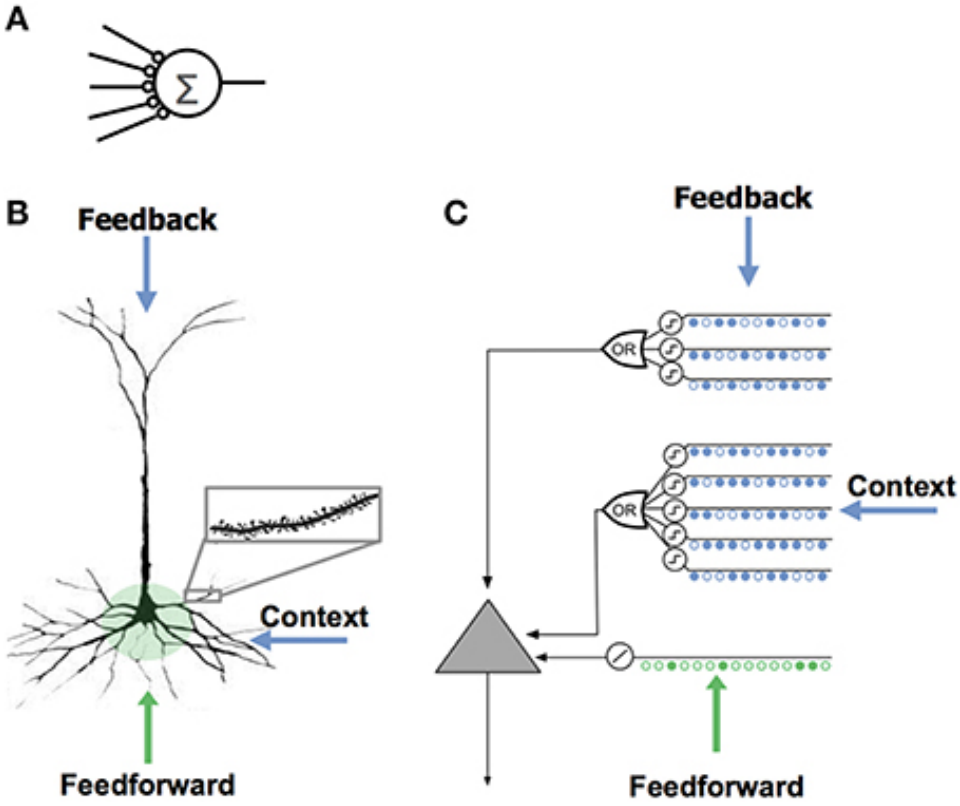



Figure 2.2: Comparison of neurons [8]: A) Traditional Machine Learning, B) Biological, C) HTM. Source: [5]

The HTM neuron has three inputs [8]:

- Feedforward, which is the input data
- Context, which is data from neighboring neurons and acts as a prediction mechanism for the next feedforward input
- Feedback, which is feedback from other neurons in the hierarchy and acts as a prediction mechanism for a sequence of feedforward inputs

How this type of neuron operates will be covered in greater detail later in this section.

### 2.1.2 Common Algorithm

HTM Theory states that there is a common algorithm for the intelligence. That the signals from hearing, vision, and touch are at the core processed by the same common algorithm. By extension, this means that HTM networks should be able to solve all kinds of logical tasks.

### 2.1.3 Sparse Distributed Representation

HTM Theory introduces Sparse Distributed Representation (SDR) as a way of representing data in HTM and can be thought of as a bit-array. Each bit

theoretically corresponds to a neuron in the neocortex and also represents some semantic information about the current data. This opens up for all kinds of mathematical operations, for instance it is possible to compare the semantic similarities between two SDRs by simply performing a binary AND operation.

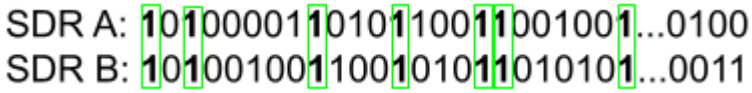



Figure 2.3: Semantic similarities between the two SDRs A and B

Observations of the brain has found that at any given point in time, a small percentage of neurons are activated and an SDR aims to keep this property by having a small percentage of bits be 1 at any given point. A common value is 2% in order to mimic the sparsity of active neurons in the neocortex. Having this property means that the chance of two bit-patterns with different semantic meanings coinciding, for instance due to bit-flips caused by noise in the data, is astronomically low and is what makes HTM robust to noise.

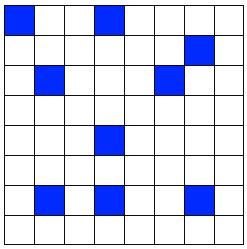


Figure 2.4: Example representation of an SDR with a length of 64 and a sparsity of 14.1%, visualized as a 2D grid.

### 2.1.4 Encoders

To convert real-world data into an SDR, there is a need for an encoder in the pipeline. These encoders can be designed to take potentially any data and convert it into an SDR with an arbitrary sparsity. Given the fact that they may have an arbitrary sparsity, the output SDRs created by the encoder are sometimes referred to as just binary arrays.

Writing an encoder is no easy task as it is important to keep semantic similarities between values. This also means that the encoder is perhaps the most important part of an HTM pipeline to get right as it is the part that can limit the system the most.

A biological example would be an eye that takes in visual information and converts it into an SDR so that it can be processed by the neocortex. This is the most important part for this thesis as creating a high dimensional encoder for video is still being researched.

There are principles that should be followed in order to create a good encoder:

- Semantically similar data should result in SDRs with overlapping active bits
- The same input should always produce the same SDR
- The output must have the same dimensionality (total number of bits) for all inputs
- The output should have similar sparsity (similar amount of one-bits) for all inputs and have enough one-bits to handle noise and subsampling

As of now, there exists encoders for numbers, categories, geospatial locations, and dates.

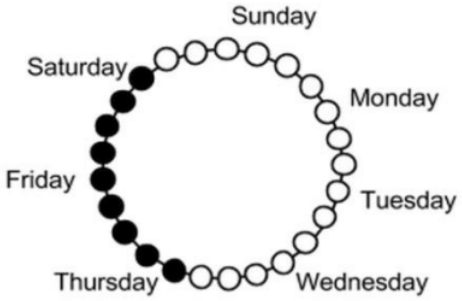



Figure 2.5: Visualization of a cyclical date encoder, which is currently encoding Friday. Source: [5]

Some applications may require anomaly detection on multiple values at once, the correct approach then is to encode the values one by one and then concatenate them into a single SDR before passing it to the HTM.

Several approaches for encoding visual data have been proposed, such as [9] which uses a neuroscientifical approach by replicating how the eye works, and [10] which uses scale-invariant feature transform (SIFT) to find points of interest in images and encode that information as an SDR. There are also deep learning approaches such as [11] which uses a Convolutional Neural Network (CNN) as part of the encoder, specifically they use the top $n$-features in a feature map as ones and set the rest to zeros in order to construct their SDRs.

The reason for why a direct binary encoding might not perform well is due to the fact that it is neither position nor scale invariant and as such breaks the first principle of creating a good encoder. For instance, if it is desired that two pictures of the same object, but in different scales have more or less the same semantic meaning, then a direct binary encoding is not going

to work. Direct binary encodings also lead to loss of information, and is hard to perform for complex objects.

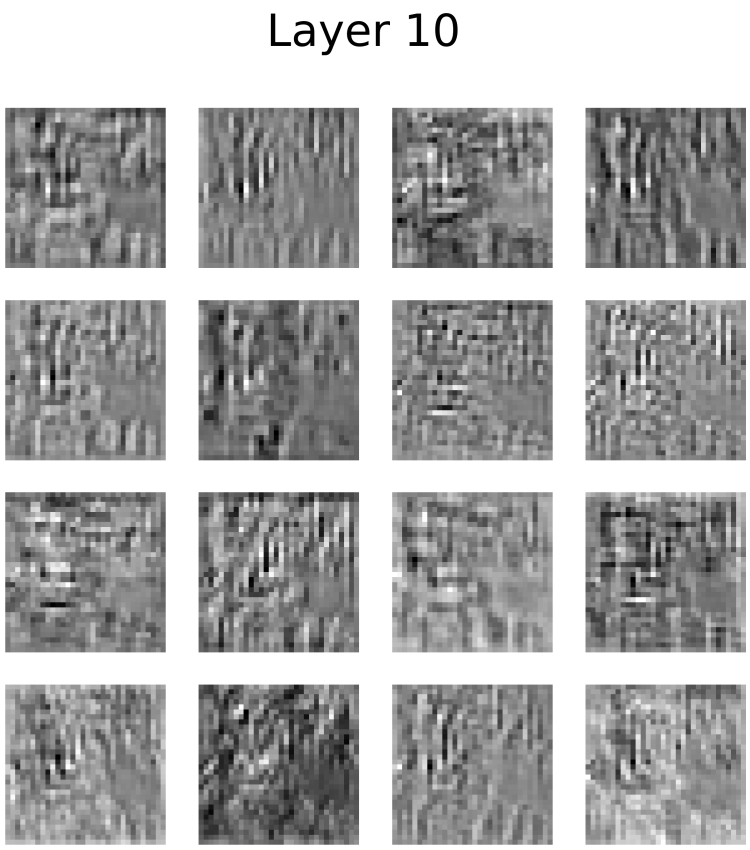



Figure 2.6: The feature map activations in the 10th layer of ResNet18. This is only 16 of them, there are many more in the layer that are not shown.

An encoder which transforms convolutional feature maps into SDRs could help solve this, but the issue is converting the dense representation of the feature maps into SDRs. Directly encoding them into SDRs by treating each value in the feature map as a float and converting it into its own SDR quickly becomes intangible due to the processing and memory requirements. Additionally, minor variations in the feature map would cause major variations in the resulting SDR. Alternatively one could follow [11] and binary threshold the top-$n$ features, but this leads to its own problems such as loss of information and that the information contained in the top-$n$ features is often undefined in models trained for complex tasks. It is also undefined what the top-$n$ features represent when there are no strong activations.

### 2.1.5 Learning

Similar to biological beings, HTM is designed to work on streaming data. It does not operate with batches like traditional machine learning, but rather with streaming data that may be changing over time.

The learning mechanism consists of two parts; the Spatial Pooler (SP) and the Temporal Memory (TM) algorithm. The latter is also commonly referred to as Sequence Memory. Together they make up the HTM neuron.

The spatial pooler takes SDRs produced by the encoder, and uses Hebbian like learning to extract semantically important information into output SDRs. These output SDRs usually have a fixed sparsity of about 2% due to the fact the spatial pooler aims to produce SDRs that have similar sparsity to what has been observed in the neocortex, but this can be configured at will and is dependent on the problem at hand.

The temporal memory algorithm, on the other hand, simulates the learning algorithm in the neocortex. It takes the SDRs formed by the spatial pooler and does two things:

- Learns sequences of SDRs formed by the spatial pooler
- Forms a prediction, in the form of a predictive SDR, given the context of previous SDRs

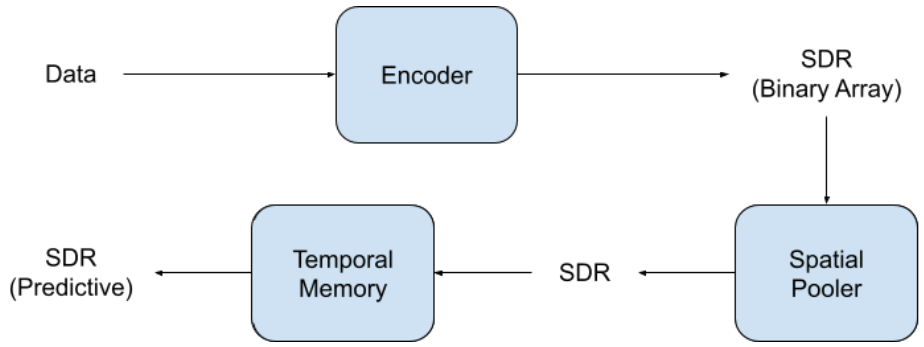



Figure 2.7: The HTM Pipeline. A common next-step could be to use a classifier to convert the predictive SDR into a classification.

This gives HTM systems the property of on-line learning, meaning they learn as they go. There is no batch training because each input into the HTM system will update the system. The system effectively builds a predictive model of the data and learns by trying to minimize the error between the true values and the predicted values. This means that the system will continuously adapt to a changing environment. A spatial pooler followed by a temporal memory forms the HTM neuron, where the color green indicates the responsibility of the Spatial Pooler and blue indicates the responsibility of the Temporal Memory:

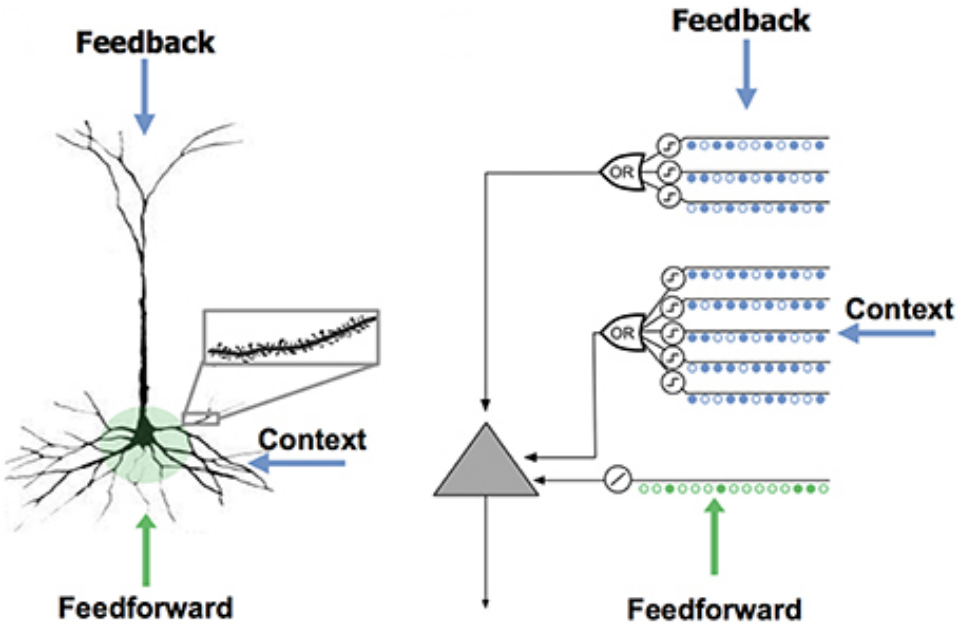



Figure 2.8: Real neuron and HTM neuron.

#### 2.1.5.1 Spatial Pooler

The spatial pooler consists of columns (mini-columns), where each column has a receptive field covering the input. In technical implementations of spatial poolers, the columns exist in name only and could be thought of as nodes instead. A column can cover parts of the input or the entire input, the range being referred to as the **potential radius**. During initialization, each column creates random connections to a percentage of the bits in the input space within its receptive field, this gives each column a unique **potential pool** when there are overlaps of receptive fields caused by a large potential radius.

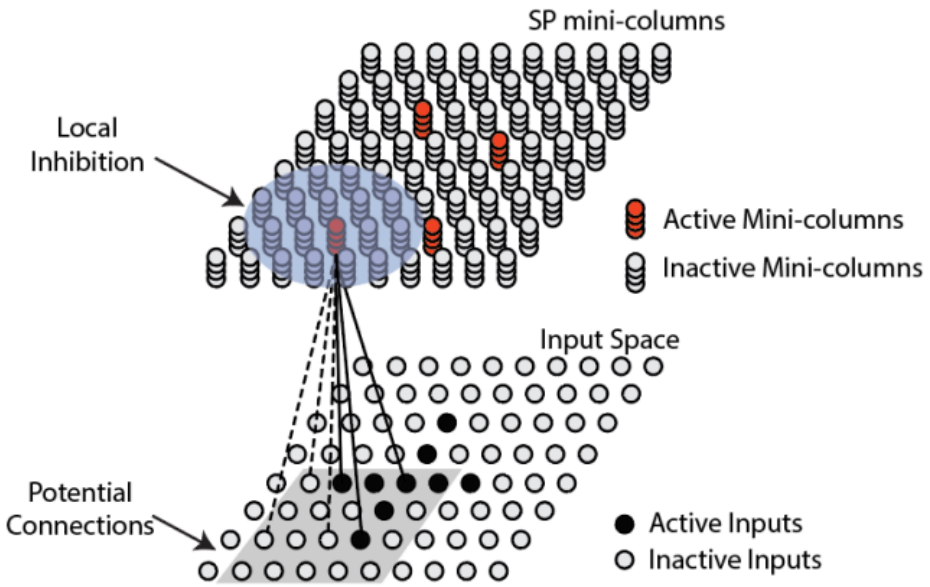



Figure 2.9: Visualization of the SP and the potential pool of one of its columns. Source: [12]

Each connection is described using a **permanence**-value which can be considered the "strength" of the connection, and it ranges between 0 and 1. During learning, the permanence value of the connections is increased or decreased depending on whether the corresponding bit in the input is active or inactive. When the permanence-value crosses above a **stimulus threshold**, the connection will be considered "active".

The amount of active connections for a given column is referred to as **overlap score**. If a column has a high enough overlap score which crosses the **overlap score threshold**, then the column will itself become active. The reason behind locking activation behind a minimum overlap score is to reduce the influence of noise in the input. Finally, out of all the active columns, only the top $n$ columns with the most overlap score will be selected to be included in the SP output. The value $n$ is chosen so that the SP output has a specific **sparsity**. Only the selected columns are allowed to learn (increase/decrease permanence).

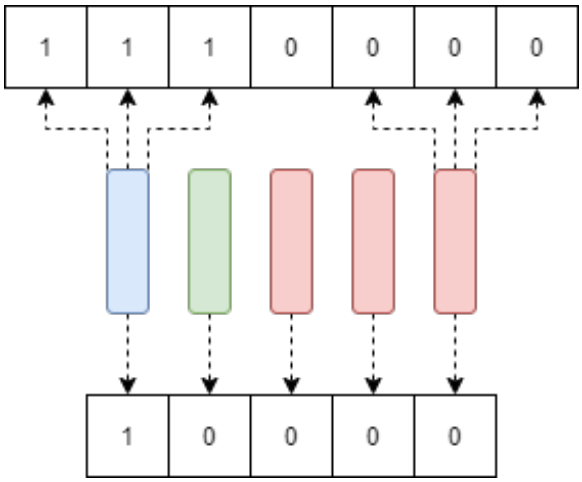



Figure 2.10: This figure illustrates how a spatial pooler works. All connections are above the stimulus threshold. The receptive field is 3 bits wide for each column. Overlap score threshold is 1, and $n = 1$. Red means inactive, green means active, and blue means active and selected.

Because only the active columns are allowed to learn, only a select few columns who got lucky during the random initialization will dominate the spatial pooler output and have a very high **active duty cycle**. Active duty cycle measures how often a column is active and ranges from 0 (never) to 1 (always).

To counter dominating columns, the spatial pooler uses **boosting**. The concept behind boosting it to "boost" the overlap score of underperforming columns and lower the overlap score of over performing columns. The result is that more columns learn and contribute to the output, which means that the spatial pooler can then process the input data with a finer granularity. One has to be careful with boosting, since it can cause instability in the spatial pooler output.

It is also possible to have **topology** in the output by selecting the columns to be included in the output by their local neighborhood, instead of comparing their overlap score globally.

All the aforementioned concepts are configurable in technical implementations.

#### 2.1.5.2 Temporal Memory

The temporal memory consists of the columns that a spatial pooler outputs, but treats them as actual columns instead of "nodes". These columns consist of cells and can contain an arbitrary **number of cells** which defines the capacity of sequences and contexts that the temporal memory can express. Each cell in a column can connect to other cells in other columns using segments (more specifically, distal dendrite segments), where each segment consists of synapses connecting to other cells.

Essentially, it takes the "node" based representation of the SP output, and turns it into a new representation which includes state, or context, from previous time steps. It achieves this by only activating a subset of cells per column, typically only one per column. This allows the temporal memory to represent a pattern in multiple contexts. If every column has 32 cells and the SP output has 100 active columns and only one cell per column is active, then the TM has $32^{100}$ ways of representing the same input. The same input will make the same columns active, but in different contexts different cells in those columns will be active.

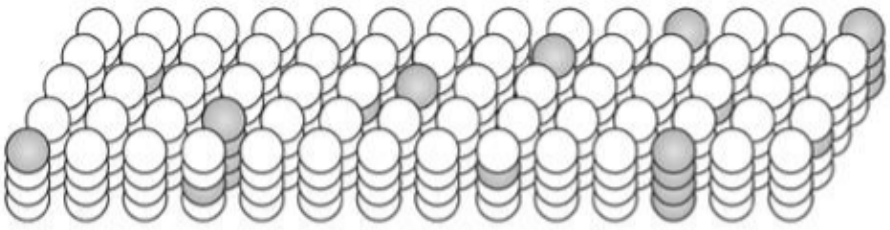


Figure 2.11: Visualization of the TM, with number of cells equal 4. Some columns are bursting. Source: [5]

The temporal memory algorithm consists of two phases. The first phase is to evaluate the SP output against predictions and choose a set of active cells. It does so by looking at the active columns and the cells they contain. If an active column contains predictive cells, then those cells are marked as active. If an active column has no predictive cells, usually caused by observing a new pattern for the first time, then the column "bursts" by activating all the cells that the column contains. Otherwise, a cell is inactive.

At this point, the active cells represent the current input in the context of previous input. For each active column we look at the segments connected to the active cell(s). If the column is bursting we look at the segments that contain any active synapses, if there is no such segment we grow one on the cell with the fewest segments. On each of the segments that we are looking at, we **increase the permanence** on every active synapse, **decrease the permanence** on every inactive synapse, and grow new synapses to cells that were previously active. The algorithm also punishes segments that caused cells to enter predictive state, but which did not end up being active.

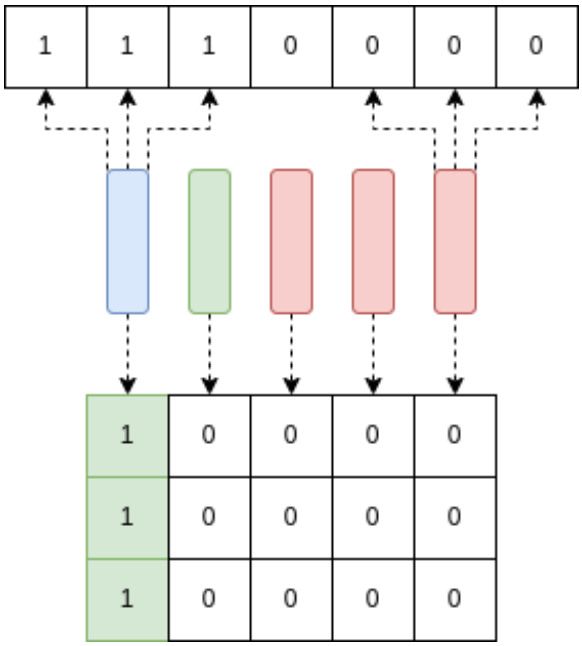



Figure 2.12: Expanded SP example with TM component where the number of cells is set to 3. The leftmost column is bursting (all 3 cells activated in green) due to the active SP output and due to containing no predictive cells.

The second phase is to form a prediction by putting cells into a predictive state. For every segment on every cell, the number of synapses connected to active cells are counted. If the number exceeds an **activation threshold**, then the segment is marked as active and all the cells connected to the segment enter the predictive state. To summarize, a cell has three possible states:

- Active, if the column is bursting or the cell was in a predictive state in the previous time step.
- Predictive, if a connected segment is active, which is in turn determined by the amount of active synapses.
- Inactive, if none of the other states apply.

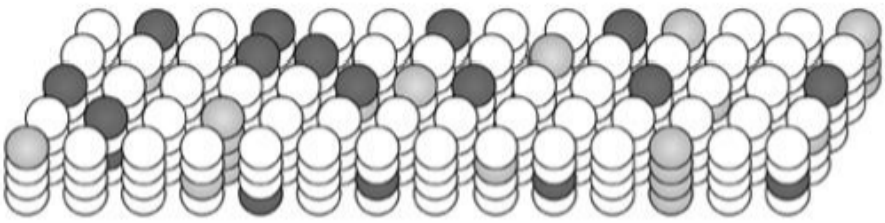


Figure 2.13: Visualization of the TM and the three states. Active in black, predictive in gray, and inactive in white. Source: [5]

One can configure how much the system can learn by setting the number

of cells and the values by which permanence should be increased or decreased. If it is desired that the TM does not "forget" at all, then the permanence value by which synapses are decremented can be set to 0. If it is desired that the TM can only express patterns in the current context and the context of the previous time step, then the number of cells can be set to 2.

Finally, the TM compares the predictions $P$ it made in the previous time step with the actual pattern $A$ in the current time step and calculates an anomaly score:

$$anomalyScore = \frac{|A_t - (P_{t-1} \cap A_t)|}{|A_t|}$$

Which is a normalized value from 0 to 1. If the anomaly score is 1, then it means that none of the predicted columns matched the current active columns of the spatial pooler. If it is 0, then it means that all predicted columns matched the current active columns of the SP.

It is also possible to estimate the number of predictions being made by the TM at any time [13]. This is done by counting the number of predictive cells, and dividing them by the number of active bits required to express a pattern. As an example, if sparsity is set so that patterns have 60 active bits and the number of predictive cells is 120, then the estimated number of predictions is given as

$$numPredictions = \frac{predictiveCells}{activeBits} = \frac{120}{60} = 2$$

This is only an estimation, in reality the two patterns may have overlapping bits in their representations, and the number of active bits for each representation may have minor deviations.

### 2.1.6 Use cases

The general use case for HTM is to perform anomaly detection. More specifically, Numenta has made example programs showcasing how HTM can be used in practice [14]:

- **Rogue Behavior Detection** which models normal behavior and detects anomalies, such as unusual use of files in a network [15].
- **Geospatial Tracking** which detects anomalies in the movement of people, objects, or material using speed and location data [16].
- **Financial Monitoring** which detects anomalies in publicly traded companies by continuously modelling stock price, stock volume, and twitter volume[17].

There are also applications that are used in production, such as the model offered by cortical.io which builds upon HTM in order to perform language analysis. They made this possible by introducing Semantic Folding and Semantic Fingerprinting [18].

## 2.2 Convolutional networks

A key element to understand is why one would want to use convolutional networks as a part of the encoder for an HTM network. The main reason is that a CNN is able to extract important spatial features from high dimensional data, effectively lowering the amount of dimensions. A CNN can also apply several properties that are useful, such as translational invariance [19]. These properties stem from the three architectural ideas in a CNN:

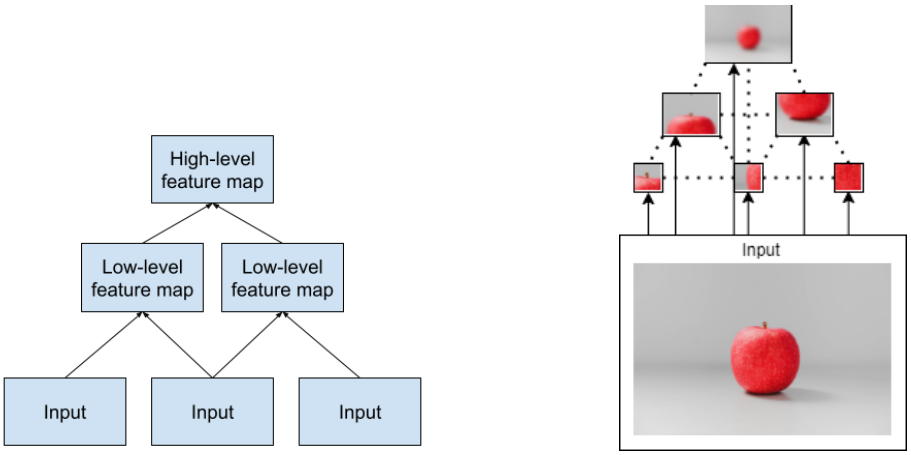
- Local receptive fields
- Shared weights
- Spatial sub-sampling

Through the use of data augmentation, it is also possible to not only improve the effectiveness of the aforementioned invariances, but to also introduce a degree of rotational and scale invariance.

## 2.3 The Thousand Brains Theory

One of the newest advancements in HTM Theory is the introduction of the Thousand Brains Theory. [20] introduces the Thousand brains theory as a way of redefining hierarchy in the brain based on recent neuroscientifical discoveries. Instead of our classical understanding of hierarchy in deep learning where each layer takes simple features and outputs complex features, we now have that every layer of the hierarchy sees the input at once but at different scales and resolutions. The different nodes in the hierarchy are now also connected and thus enable the network to use all available views of the object in order to create an understanding of that object.

To summarize, the object is learned by the brain using multiple models that may rely on different inputs, the models then vote to reach a consensus on what they are sensing. This is coincidentally similar to ensemble learning such as [21]. Each model can be thought of as a mini-brain, hence the name "The Thousand Brains Theory".




(a) Classical Hierarchy

(b) Hierarchy as in the Thousand Brains Theory

Figure 2.14: Comparison of classical hierarchy and the hierarchy introduced by the Thousand Brains Theory

This new type of hierarchy is also coincidentally similar to some state-of-the-art image recognition deep learning architectures such as [22] and [23], in the sense that they apply different sized convolutional filters, where each filter can be thought of as its own separate model, on the data and do predictions based on all of them at once. This also ensures scale invariance of objects fed in to the architecture.

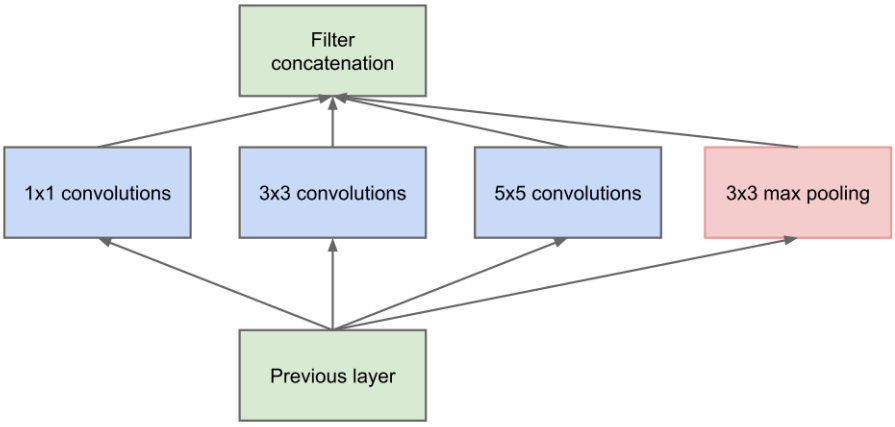



Figure 2.15: How the Inception [23] architecture combines multiple filters

While the Thousand Brains Theory is not yet implemented in any way in a standard HTM model, it does show that recent developments within deep learning for image analysis have similarities with HTM theory.

## 2.4 Anomaly detection

As reviewed by Pang et al. [24], anomaly detection is often defined as detecting data points that deviate from the general distribution of the data, this also often includes quantifying the level of deviation. Unlike other problems within machine learning and statistics, anomaly detection deals with unpredictable and rare events, therefore adding complexities to problems. Some complexities are as follows:

- **Unknowns** Anomalies are associated with many unknowns which do not become known until the anomaly happens. [25, 26] are works that address this.
- **Rarity and class imbalance** Anomalies are by definition rare instances, which means that it becomes difficult to create a balanced dataset. [27] reviews the current solutions to this problem.
- **Heterogeneity** Anomalies can take form in many ways, and as such one class of anomalies can be vastly different from another. Approaches such as [28] have been proposed to alleviate the problem.

This makes it hard to apply traditional deep learning methods for anomaly detection, because they are designed with pairs of $\{input, target\}$ in mind.

### 2.4.1 State-of-the-Art Algorithms

The current state-of-the-art algorithms are numerous, but the main approach is achieved by using deep learning [24]. As previously mentioned, traditional deep learning approaches are hard to apply for anomaly detection. Instead, a popular approach is to use generative deep learning models such as GANs [24, 29, 30] to generate synthetic data and compare it to real data in order to detect anomalies. This approach is based on the assumption that the model will only be able to generate data similar to what it has been trained on, and therefore fail when an anomalous event occurs.

The advantage is that GANs are generally good at generating realistic data, especially when it comes to images. The disadvantage is that GANs are very hard to train and may give suboptimal results given that it tries to generate good synthetic data rather than directly detect anomalies. The training data also needs to contain all possible non-anomalous classes of events, which may not be a realistic expectation.

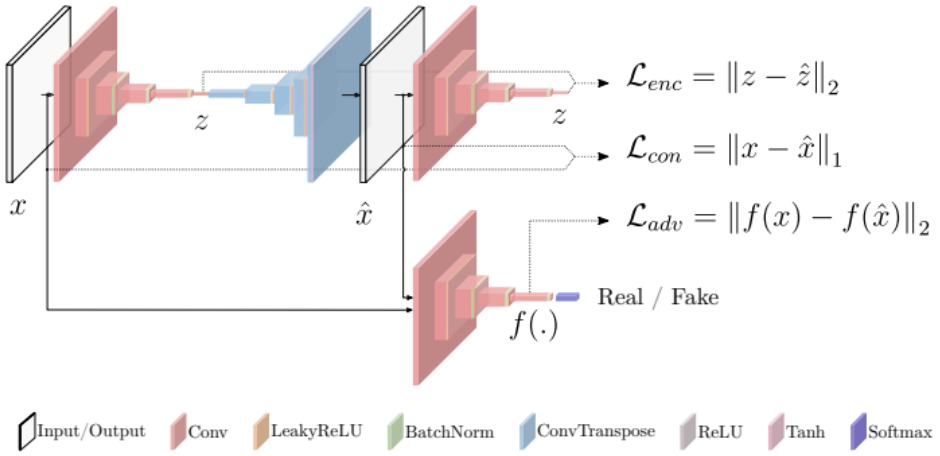



Figure 2.16: GANomaly [29], a variation of GAN for Anomaly detection

Another common approach is autoencoders, which aim to minimize the reconstruction error from a learned feature representation space [31, 24, 30]. The assumption is that anomalies are more difficult to reconstruct than normal data, hence the reconstruction error will be high and can therefore be used as a metric to detect anomalies.

A disadvantage for deep-learning models in general is that they are susceptible to noise in the dataset [32, 33]. They are also in most cases not self-supervised and therefore require constant tuning in order to stay effective on changing data. Additionally, they are very hard to design and train for the purpose of detecting anomalies in complex data such as surveillance.

There are also variations of the aforementioned approaches, such as Adversarial Autoencoders, but the core idea is the same; to get an anomaly measure using some sort of generated or reconstructed data.

### 2.4.2 Explainability

As stated in [34], as "black-box" approaches such as deep learning surged in popularity, many realized that they offered poor explainability. While it is known *how* the models make their decisions, their huge parametric spaces make it unfeasible to know *why* they decide as they do. Combined with the vast potential that deep learning offers in critical sectors such as medicine, has lead to an increase in focus on developing approaches that offer explainability.

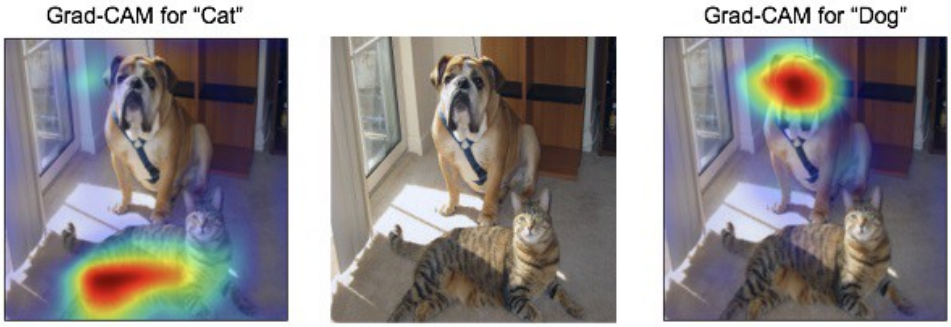



Figure 2.17: Grad-CAM visualization for cat and dog.

Approaches such as Grad-CAM [35] and Guided Backpropagation [36] offer improvements in that regard, but these approaches are not made with generative models in mind. In fact, there are very few explainable AI approaches for generative models [34]. Since generative models make up most of the state-of-the-art approaches in anomaly detection, this means that there is a lack of explainability in the field of anomaly detection in videos.

### 2.4.3 Smart Surveillance

Smart surveillance, which is the use of automatic video analysis in surveillance, has seen rapid development since its inception. [30] presents and summarizes recent progress for anomaly detection in video for surveillance purposes, where the most promising methods are achieved by using convolutional auto-encoders (AE) and Generative Adversarial Networks (GAN). The results show that the deep learning approaches have a high degree of accuracy, and are consistently improving.

The paper also discusses problems with using deep learning approaches for anomaly detection. One of the examples that it uses is about a bicycle on campus, which related to the aforementioned issue with the non-realistic requirement that the training data must contain all possible classes of non-anomalous events.

### 2.4.4 HTM Performance in Anomaly Detection

Knowing that deep learning approaches have a high degree of accuracy but suffer from problems related to generalizability, adaptability, and noise it stands to reason that HTM is a viable alternative for anomaly detection.

[37] explores the use of HTM for anomaly detection on low dimensional data such as temperature data from an industrial machine. The authors also discuss benchmarks for anomaly detection and compare different methods. The results show that HTM is very capable of performing anomaly detection, especially in a changing environment. HTM is able to outperform other anomaly detection methods and has the advantage of not requiring any per-problem parameter tuning.

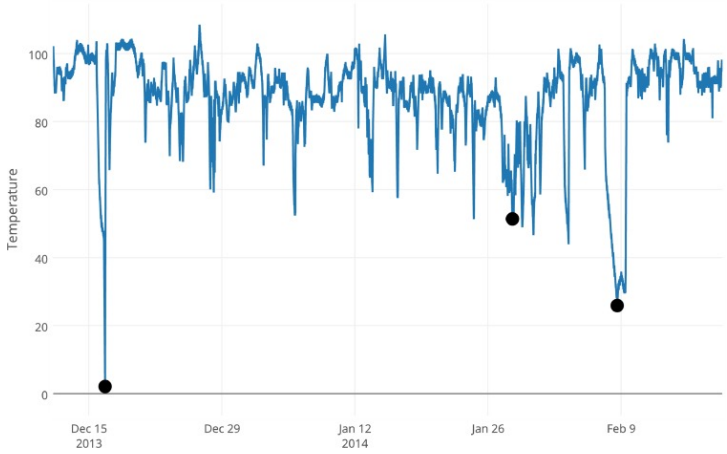



Figure 2.18: Temperature anomaly detection from [37].

For high-dimensional anomaly detection, [38] used a HTM system to find anomalous frames in videos of motions. The anomalies were artificially created by swapping certain frames between different motion videos in the dataset. The results show that the HTM system was able to correctly detect some anomalies, but not an impressive amount. One thing to note is that direct binary representations of the video frames were used as SDRs, therefore no proper encoding was performed which might have led to the poor results. This hints at the fact that HTM by itself is not capable of handling high dimensional data, and is instead reliant on an encoder to lower the dimensionality by extracting important spatial features.

Figure 2.19: Example motion frame used in [38].

### 2.4.5 Deep Learning HTM Encoder

A potential solution to most of the aforementioned issues could be to combine the deep learning approaches with an HTM network. This way it could be possible to leverage the self-supervision and noise resilience property of HTM, together with the powerful feature extraction and representation of deep learning approaches. Effectively combining the best of both approaches while eliminating the disadvantages that have been previously mentioned.

## 2.5 Summary

# Chapter 3

# Grid HTM

## 3.1 Introduction

When it comes to applying HTM on videos, this thesis proposes to use segmentation techniques to simplify the data into an SDR-friendly format. These segmentation techniques could be everything from simple binary thresholding to deep learning instance segmentation. Even keypoint detectors such as [39] could be applied. When explaining Grid HTM, the examples will be taken from deep learning instance segmentation of cars on a video from the VIRAT [40] dataset.

Figure 3.1: Segmentation result of cars, which is suited to be used as an SDR. Original frame taken from [40].

The idea is that the SP will learn to find an optimal general representation of cars. How general this representation is can be configured using the various parameters, but ideally they should be set so that different cars will be represented similarly while trucks and motorcycles will be represented differently.

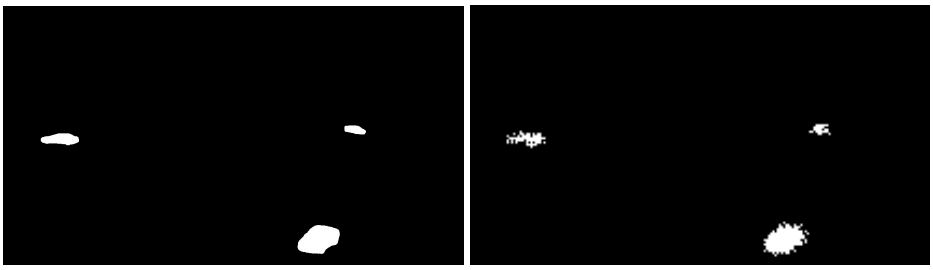


Figure 3.2: The SDR and its corresponding SP representation. Note that the SP is untrained.

The task of the TM will then be to learn the common patterns that the cars exhibit, their speed, shape, and positioning will be taken into account. Finally, the learning will be set so that new patterns are learned quickly, but forgotten slowly. This will allow the system to quickly learn the norm, even if there is little activity, while still reacting to anomalies. This requires that the input is stationary, in our example this means that the camera is not moving.

Ideally, the system will have a calibration period spanning several days or weeks, during which the system is not performing any anomaly detection, but is just learning the patterns.

## 3.2 Refinement

One issue that becomes evident is the lack of invariance. Because the TM is learning the global patterns, in our example it learns that it is normal for cars to drive along the road but only in the context of there being cars parked in the parking lot. It is instead desired that the TM learns that it is normal for cars to drive along the road, regardless of whether there are cars in the parking lot. This thesis proposes a solution based on dividing the encoder output into a grid, and have a separate SP and TM for each cell in the grid. The anomaly scores of all the cells are then aggregated into a single anomaly score using an aggregation function.

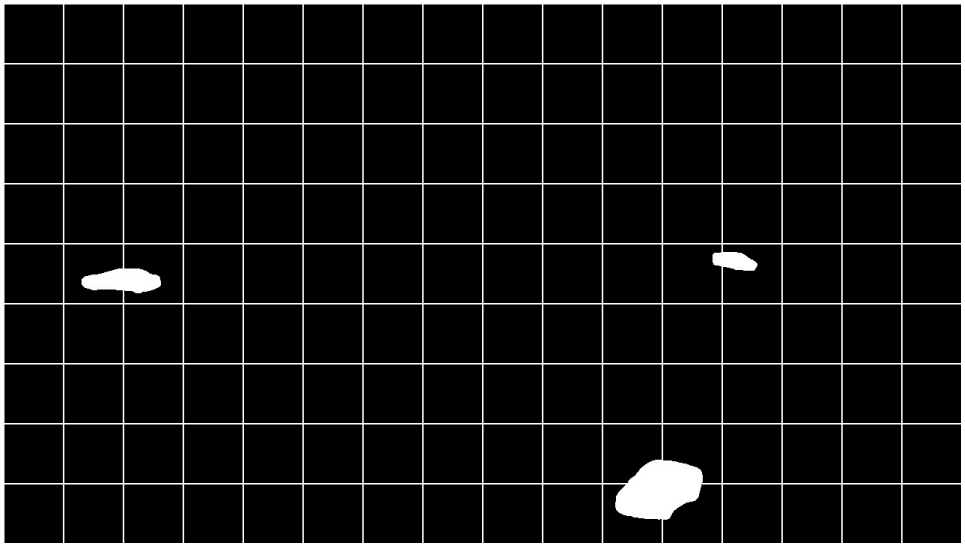


Figure 3.3: The encoder output divided into a grid.

It might be tempting to use the mean of all the anomaly scores as the aggregation function, but this leads to problems with normalization, meaning that an overall anomaly score of 1 is hard to achieve due to many cells having a zero anomaly score. In fact, it becomes unclear what a high anomaly score is anymore. An example is that one cell with a high anomaly score will lead to the same average anomaly score as every cell having a small anomaly score. That being said, the selection of an aggregation function depends on how noisy the input data is. For data with little noise, a potential aggregation function could be the non-zero mean:

$$X : \{x : x > 0\}$$
$$anomScore = \begin{cases} \frac{\sum_{x \in X} x}{|X|} & \text{if } |X| > 0 \\ 0 & \text{else} \end{cases}$$

Meaning that only the cells with a non-zero anomaly score will be contributing to the overall anomaly score, which helps solve the aforementioned normalization problem. On the other hand, this will perform poorly when the system is exposed to noisy data which could lead to there always being a cell somewhere with a high anomaly score.

Having the encoder output divided into a grid has the added benefit of introducing explainability into the model. By using Grid HTM it is now possible to find out where in the input an anomaly has occurred. Combined with the ability to estimate the number of predictions at any given time for any cell, makes this an attractive approach. In addition, it is also possible to configure the SP and the TM in each cell independently, giving the system increased flexibility. Last but not least, dividing it into smaller cells makes it possible to run each cell in parallel for increased performance.

That being said, a potential problem with this approach is that the previously mentioned rules for creating a good encoder may not be

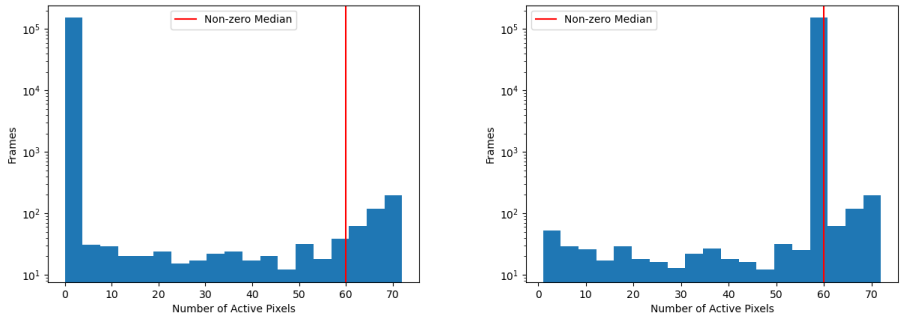
respected, and therefore should be reviewed:

- **Semantically similar data should result in SDRs with overlapping active bits**. In this example, a car at one position will produce an SDR with a high amount of overlapping bits as another car at a similar position in the input image.
- **The same input should always produce the same SDR**. The segmentation model produces a deterministic output given the same input.
- **The output must have the same dimensionality (total number of bits) for all inputs**. The segmentation model output has a fixed dimensionality.
- **The output should have similar sparsity (similar amount of one-bits) for all inputs and have enough one-bits to handle noise and subsampling**. The segmentation model does not respect this. An example is that there can be no cars (zero active bits), one car ($n$ active bits), or two cars ($2n$ active bits).

The solution for the last rule is two-fold, and consists of imposing a soft upper bound and a soft lower bound for the number of active pixels within a cell. The purpose is to lower the variation of number of active pixels, while also containing enough semantic information for the HTM to work:

- Pick a cell size so that the distribution of number of active pixels is as tight as possible, while containing enough semantic information and also being small enough so that the desired invariance is achieved. The cell size acts as a soft upper bound for the possible number of active pixels.
- Create a pattern representing emptiness, where the number of active bits is similar to what can be expected on average when there are cars inside a cell. This acts as a soft lower bound for the number of active pixels.

There could be situations where a few pixels are active within a cell, which could happen when a car has just entered a cell, but this is fine as long as it does not affect the distribution too much. If it does affect the distribution, which can be the case with noisy data, then an improvement would be to add a minimum sparsity requirement before a cell is considered not empty, e.g. less than 5 active pixels means that the cell is empty. In the following example, the number of active pixels within a cell centered in the video was used to build the following distributions:




(a) Without empty pattern.

(b) With empty pattern.

Figure 3.4: Distribution of number of active pixels within a cell of size $12 \times 12$, it can also be observed that it would benefit from having a minimum sparsity requirement.

With a carefully selected empty pattern sparsity, **the standard deviation of active pixels was lowered from 3.78 to 1.88**. It is possible to automate this process by developing an algorithm which finds the optimal cell size and empty pattern sparsity which causes the least variation of number of active pixels per cell. This algorithm would be a part of the calibration process.

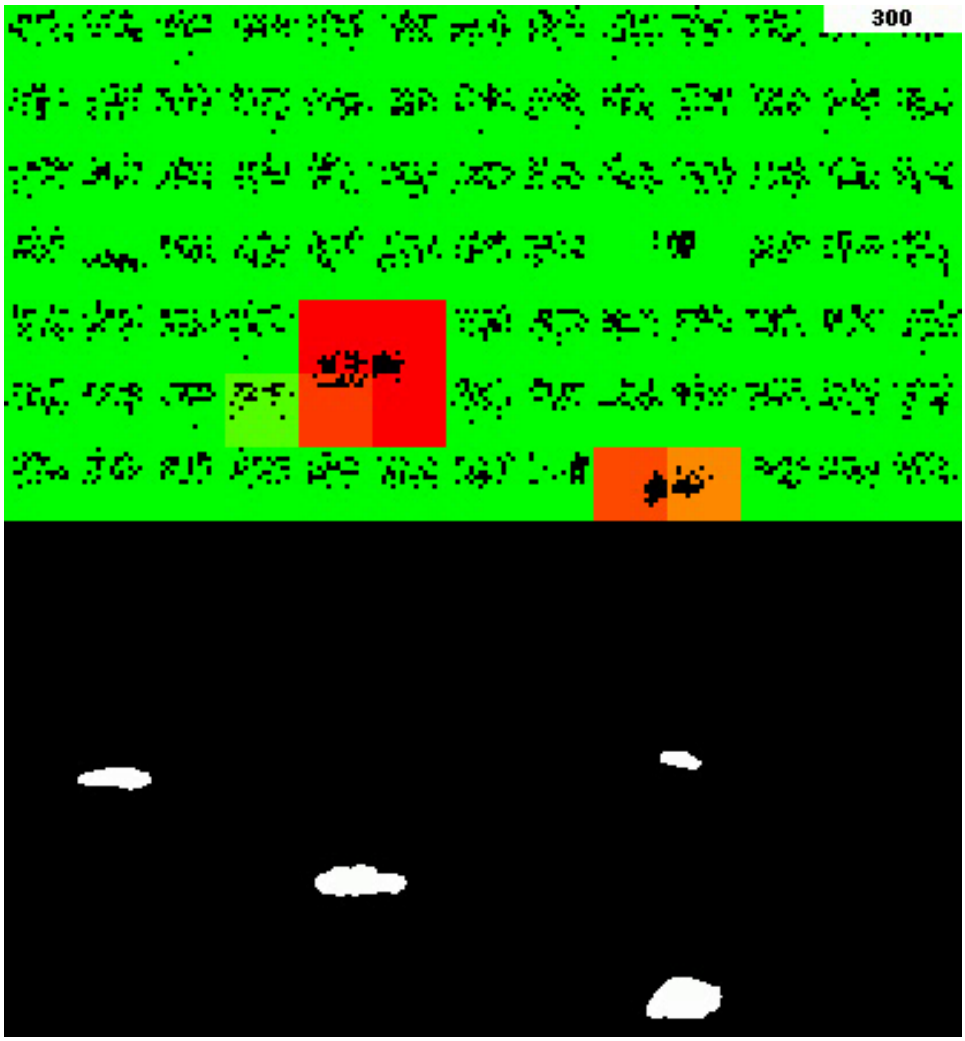



Figure 3.5: Example Grid HTM output and the corresponding input. The color represents the anomaly score for each of the cells, where red means high anomaly score and green means zero anomaly score. Two of the cars are marked as anomalous because they are moving, which is something the Grid HTM has not seen before during its 300 frame long lifetime.

Another issue occurs when a car first comes into a cell. The TM in that cell has no way of knowing that a car is about to enter, and therefore the first frame that a car enters a cell will cause a high anomaly output for that cell. The band-aid solution is to ignore the anomaly score for the frame during which the cell goes from being empty to being not empty.

## 3.3 Use Cases

The most intuitive use case is to use Grid HTM for semi-active surveillance. One example is making it possible to have an entire city be monitored by a few people. This is made possible by making it so that people only have to look at segments that the Grid HTM has found anomalous, which is what

drastically lowers the manpower requirement for active monitoring of the entire city.

## 3.4 Summary

# Chapter 4

# Experiments and Results

## 4.1 Bouncing Ball Test

To give credibility to the approach mentioned in Chapter 3, a simple test case to test the capabilities of HTM and confirm that they apply on a video is introduced.

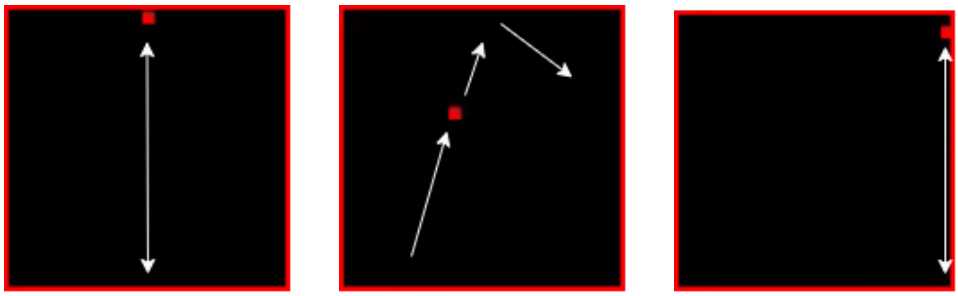


Figure 4.1: The bouncing ball test, and its three stages

The video consists of a ball bouncing up and down until an anomaly occurs in the form of a sudden introduction of a horizontal velocity. After a while this horizontal velocity is set back to 0 and the ball is once again bouncing up and down in-place.

The model used is a standard HTM model, which covers the entire input. This is equivalent to a single cell in a Grid HTM.

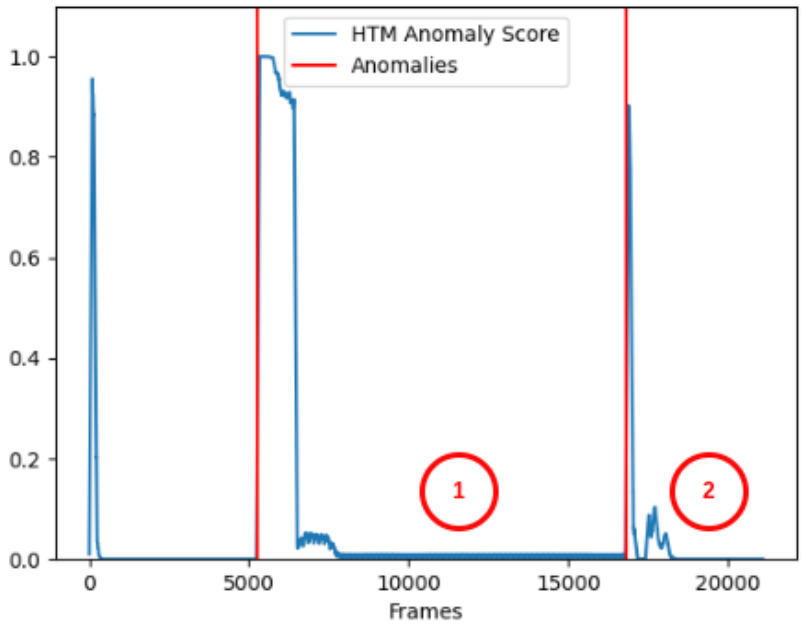



Figure 4.2: The 100-point moving average of the anomaly score in the bouncing ball experiment.

From the figure it can be observed that the HTM correctly detects anomalies and quickly adapts to them. On the other hand, the result is not perfect due to the minor oscillations close to mark 1 and the anomaly spikes towards the end close to mark 2. While the imperfections are not major and can be safely ignored, it is still important to understand their causes and what can be done to improve upon them.

The reason for the oscillations is due to the spatial pooler being dominated by a lucky few columns. The solution is to enable boosting. This also helped with the spikes towards the end.

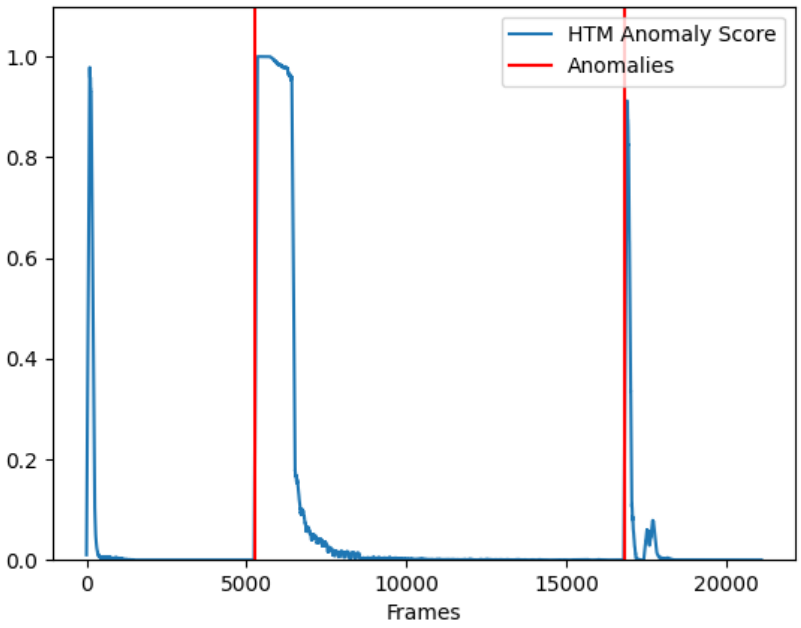



Figure 4.3: Bouncing ball with boosting enabled.

The reason for the anomaly spikes towards the end is because the spatial pooler had found an optimal representation when the ball was bouncing freely, but when the ball stops and starts bouncing in-place the spatial pooler ends up unlearning the old optimal representation while it learns the new optimal representation. This causes a sudden minor change in the SP output, which the TM reports as anomalous. The solution is to set the value by which permanence is decreased by to zero, effectively disabling the ability of the spatial pooler to "forget". That being said, the ability to decrement permanence is important in HTM systems, therefore disabling it is not always feasible.

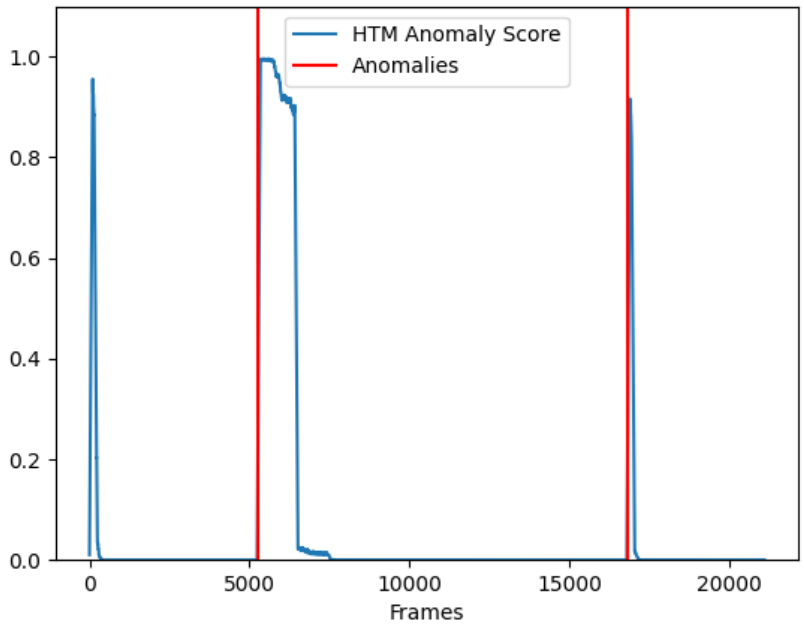



Figure 4.4: Bouncing ball without the ability of the SP to "forget".

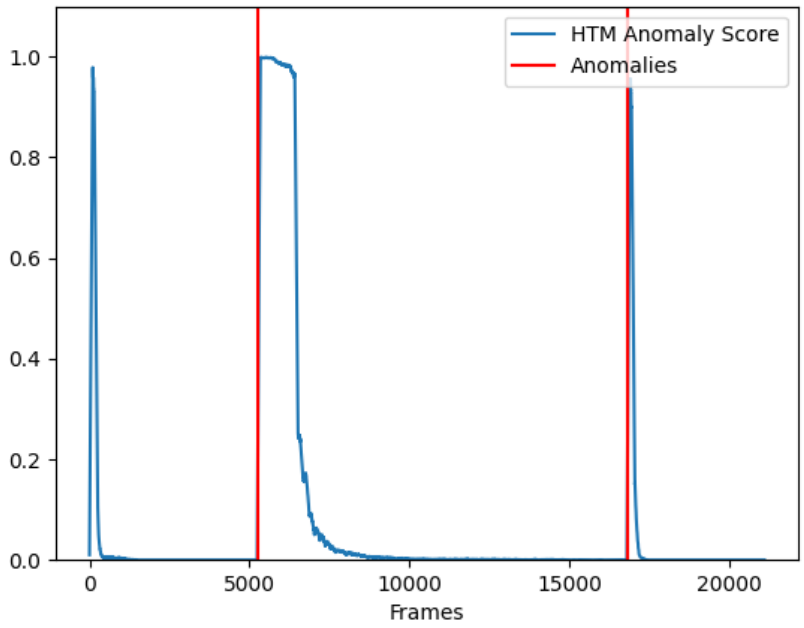



Figure 4.5: Bouncing ball without the ability of the SP to "forget" and with boosting enabled.

Final list of parameters for reproducibility:

| Parameter | Value | Notes |
|---|---|---|
| inputDimensions | frame_size, frame_size | |
| columnDimensions | frame_size/2, frame_size/2 | |
| potentialPct | 0.1 | |
| potentialRadius | 120 | |
| localAreaDensity | 0.02 | |
| globalInhibition | True | Set to False to enable topology |
| wrapAround | True | Allows the columns near the edges to "wrap around" and form connections on the other side |
| synPermActiveInc | 0.1 | |
| synPermInactiveDec | 0 | Set to >0 to enable the SP to "forget" |
| stimulusThreshold | 2 | |
| seed | 2 | |
| boostStrength | 0.1 | Set to 0 to disable boosting |
| dutyCyclePeriod | 250 | |

Table 4.1: SP Parameters

| Parameter | Value | Notes |
|---|---|---|
| columnDimensions | frame_size/2, frame_size/2 | Same as the SP |
| predictedSegmentDecrement | 0.003 | |
| permanenceIncrement | 0.1 | |
| permanenceDecrement | 0.001 | |
| minThreshold | 3 | |
| activationThreshold | 5 | |
| cellsPerColumn | 16 | |
| seed | 2 | |

Table 4.2: TM Parameters

This is a very simple problem which does not require invariances, making it unsuitable for Grid HTM. Grid HTM would be suitable if there was two or more independent bouncing balls. Still, it is interesting to see how Grid HTM performs compared to normal HTM. The SP and TM parameters were selected so that they were as close as possible to the normal HTM parameters. The non-zero mean was chosen as the aggregation function.

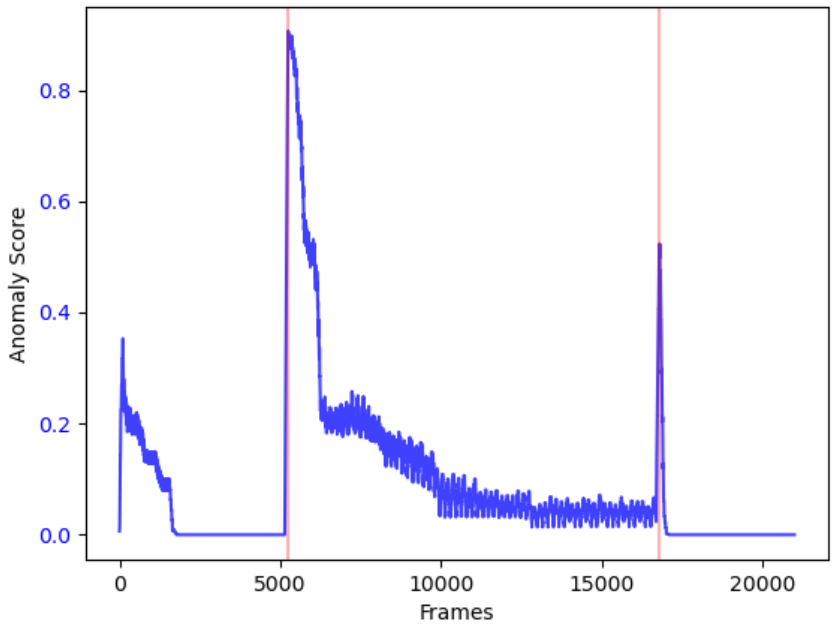



Figure 4.6: Grid HTM

It can be observed that Grid HTM performs worse than the normal HTM, but the result is still acceptable. This is to be expected since this problem is not suited for the Grid HTM and that the parameters are probably not optimal:

| **Parameter** | **Value** | **Notes** |
|---|---|---|
| inputDimensions | 30, 30 | |
| columnDimensions | 15, 15 | |
| potentialPct | 0.5 | Increased in order to compensate for the smaller potential pool |
| potentialRadius | 5 | |
| localAreaDensity | 0.02 | |
| globalInhibition | True | Set to False to enable topology |
| wrapAround | False | |
| synPermActiveInc | 0.1 | |
| synPermInactiveDec | 0 | Set to >0 to enable the SP to "forget" |
| stimulusThreshold | 2 | |
| seed | 2 | |
| boostStrength | 0 | Causes instability in empty cells |
| dutyCyclePeriod | 250 | |

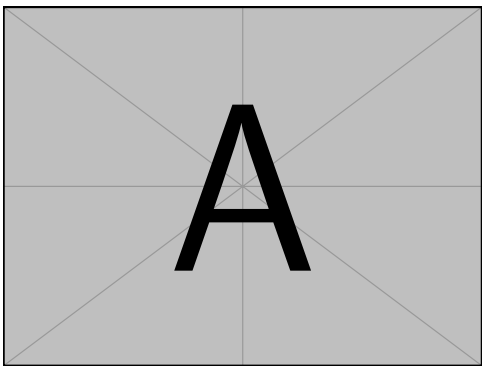
Table 4.3: SP Parameters

| Parameter | Value | Notes |
|---|---|---|
| columnDimensions | 15, 15 | Same as the SP |
| predictedSegmentDecrement | 0.003 | |
| permanenceIncrement | 0.1 | |
| permanenceDecrement | 0.001 | |
| minThreshold | 3 | |
| activationThreshold | 5 | |
| cellsPerColumn | 16 | |
| seed | 2 | |

Table 4.4: TM Parameters

## 4.2 Surveillance example

As stated earlier, one of the use cases of Grid HTM is anomaly detection in surveillance. This example will show how Grid HTM could perform. The video to be used is part of the VIRAT[40] video dataset, and was selected due to its long duration and stationary camera. The downside is that the video does not contain any anomalies, and also consists of several segments with sudden frame skips in between.




As previously mentioned, both binary thresholding and deep learning feature map extraction as encoders have their downsides. Therefore, this thesis proposes to use a combination of both, a segmentation model which can extract classes into their respective SDRs. Meaning that there could be an SDR for cars and an SDR for persons, that are then concatenated before being fed into the system. The segmentation model used is ResNet101[41] enhanced with PointRend[42], pretrained on ImageNet[43] and implemented using PixelLib[44].

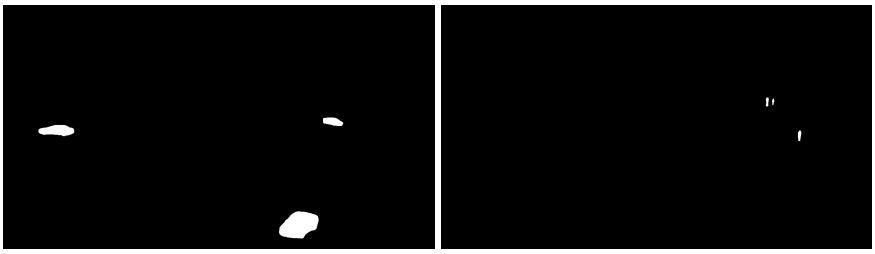


Figure 4.7: Example segmentation of cars and persons.

For the sake of simplicity, this experiment will focus only on the segmentation of cars.

While on the topic of segmentation, it is important to mention that the segmentation model is not perfect and that there are cases where objects are misclassified as well as cases where cars repeatedly go above and below the confidence threshold.

## 4.3 Sperm example

As seen in the surveillance example, it seems the HTM can detect when segments begin and end. This experiment will explore this ability in greater detail. The dataset is VISEM [45], a sperm dataset which consists of videos that are made up of several segments. The sperm cells will be segmented using a rough binary thresholding.

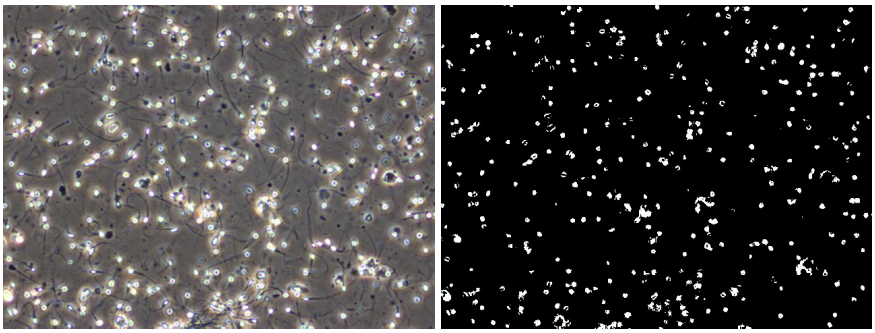


Figure 4.8: Example frame from a sperm video and its corresponding segmentation.

To ensure that the HTM does not just react to the sudden change in pixels but does something more, the L1 error will be used as a benchmark to compare against:

$$E_t = \sum |F_t - F_{t-1}|$$

Where $F_t$ denotes a segmented frame at time step $t$.

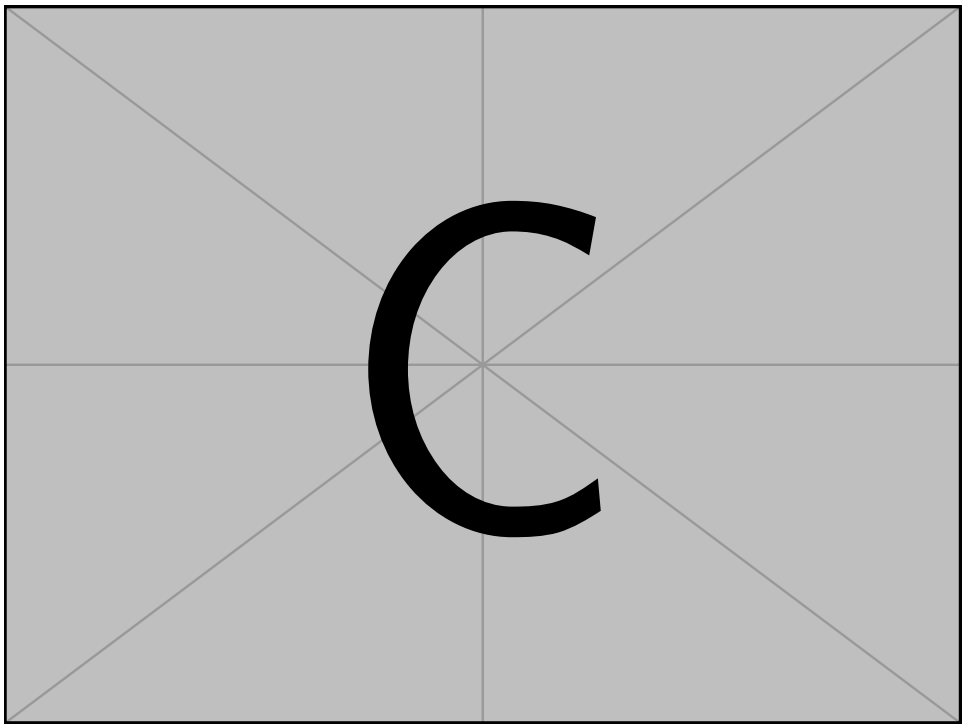


Figure 4.9: Results.

# Chapter 5

# Conclusions & Future Work

# Bibliography


[1] Divyanshi Tewari. *U.S. Video Surveillance Market by Component (Solution, Service, and Connectivity Technology), Application (Commercial, Military & Defense, Infrastructure, Residential, and Others), and Customer Type (B2B and B2C): Opportunity Analysis and Industry Forecast, 2020–2027*. Mar. 2019. URL: https://www.alliedmarketresearch.com/us-video-surveillance-market-A06741.

[2] Web of Science. Jan. 2022. URL: https://www.webofscience.com/wos/woscc/summary/f6ae0ce5-4319-416f-ab92-08d042bc3871-21874d31/relevance/1.

[3] Jeff Hawkins and Dileep George. "Hierarchical Temporal Memory Concepts , Theory , and Terminology." In: 2006.

[4] Vladimir Monakhov. *SP Topology is weird when input width and height are different*. Nov. 2021. URL: https://github.com/htm-community/htm.core/issues/961.

[5] J. Hawkins et al. "Biological and Machine Intelligence (BAMI)." Initial online release 0.4. 2016. URL: https://numenta.com/resources/biological-and-machine-intelligence/.

[6] Taki Hasan Rafi. *A Brief Review on Spiking Neural Network - A Biological Inspiration*. Apr. 2021. DOI: 10.20944/preprints202104.0202.v1.

[7] MRaptor. June 2016. URL: https://discourse.numenta.org/t/htm-cheat-sheet/828.

[8] Jeff Hawkins and Subutai Ahmad. "Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex." In: *Frontiers in Neural Circuits* 10 (2016), p. 23. ISSN: 1662-5110. DOI: 10.3389/fncir.2016.00023. URL: https://www.frontiersin.org/article/10.3389/fncir.2016.00023.

[9] David McDougall (ctrl-z-9000-times). Sept. 2019. URL: https://github.com/htm-community/htm.core/issues/259#issuecomment-533333336.

[10] Fabian Fallas-Moya and Francisco J. Torres-Rojas. "Object Recognition Using Hierarchical Temporal Memory." In: Jan. 2018, pp. 1–14. ISBN: 978-3-319-76260-9. DOI: 10.1007/978-3-319-76261-6_1.

[11] Y. Zou et al. "Hierarchical Temporal Memory Enhanced One-Shot Distance Learning for Action Recognition." In: *2018 IEEE International Conference on Multimedia and Expo (ICME)*. 2018, pp. 1–6. DOI: 10.1109/ICME.2018.8486447.

[12] Yuwei Cui. *HTM Spatial Pooler*. Feb. 2017. URL: https://numenta.github.io/numenta-web/assets/pdf/spatial-pooling-algorithm/HTM-Spatial-Pooler-Overview.pdf.

[13] Sam Heiserman (sheiser1). Jan. 2022. URL: https://discourse.numenta.org/t/htm-core-am-i-getting-prediction-density-correctly/9299.

[14] *HTM Legacy Applications*. URL: https://numenta.com/machine-intelligence-technology/applications/.

[15] *Whitepaper: HTM for Rogue Behavior Detection*. URL: https://numenta.com/assets/pdf/whitepapers/Rogue%20Behavior%20Detection%20White%20Paper.pdf.

[16] *Whitepaper: HTM for Geospatial Tracking*. URL: https://numenta.com/assets/pdf/whitepapers/Geospatial%20Tracking%20White%20Paper.pdf.

[17] *Github: HTM for Finance*. URL: https://github.com/numenta/numenta-apps.

[18] Francisco De Sousa Webber. *Semantic Folding Theory And its Application in Semantic Fingerprinting*. 2016. arXiv: 1511.08855 [cs.AI].

[19] Y. Lecun et al. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[20] Jeff Hawkins et al. "A Framework for Intelligence and Cortical Function Based on Grid Cells in the Neocortex." In: *Frontiers in Neural Circuits* 12 (2019), p. 121. ISSN: 1662-5110. DOI: 10.3389/fncir.2018.00121. URL: https://www.frontiersin.org/article/10.3389/fncir.2018.00121.

[21] Vajira Thambawita et al. "DivergentNets: Medical Image Segmentation by Network Ensemble." In: *Proceedings of the 3rd International Workshop and Challenge on Computer Vision in Endoscopy (EndoCV 2021) co-located with with the 17th IEEE International Symposium on Biomedical Imaging (ISBI 2021)*. 2021.

[22] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV].

[23] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[24] Guansong Pang et al. "Deep Learning for Anomaly Detection." In: *ACM Computing Surveys* 54.2 (Apr. 2021), pp. 1–38. ISSN: 1557-7341. DOI: 10.1145/3439950. URL: http://dx.doi.org/10.1145/3439950.

[25] Katarzyna Michałowska et al. "Anomaly Detection with Unknown Anomalies: Application to Maritime Machinery." In: *IFAC-PapersOnLine* 54.16 (2021). 13th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2021, pp. 105–111. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2021.10.080. URL: https://www.sciencedirect.com/science/article/pii/S2405896321014828.

[26] Wei Fan et al. "Using Artificial Anomalies to Detect Unknown and Known Network Intrusions." In: *Knowledge and Information Systems* 6 (Oct. 2001). DOI: 10.1007/s10115-003-0132-7.

[27] Debashree Devi, Saroj Biswas, and Biswajit Purkayastha. "A Review on Solution to Class Imbalance Problem: Undersampling Approaches." In: Aug. 2021.

[28] Debanjan Datta, Sathappan Muthiah, and Naren Ramakrishnan. *Detecting Anomalies Through Contrast in Heterogeneous Data*. 2021. arXiv: 2104.01156 [cs.LG].

[29] Samet Akcay, Amir Atapour-Abarghouei, and Toby P. Breckon. *GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training*. 2018. arXiv: 1805.06725 [cs.CV].

[30] Sijie Zhu, Chen Chen, and Waqas Sultani. *Video Anomaly Detection for Smart Surveillance*. 2020. arXiv: 2004.00222 [cs.CV].

[31] Tung Kieu et al. "Outlier Detection for Time Series with Recurrent Autoencoder Ensembles." In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 2725–2732. DOI: 10.24963/ijcai.2019/378. URL: https://doi.org/10.24963/ijcai.2019/378.

[32] Hossein Hosseini, Baicen Xiao, and Radha Poovendran. *Google's Cloud Vision API Is Not Robust To Noise*. 2017. arXiv: 1704.05051 [cs.CV].

[33] Dan Hendrycks and Thomas Dietterich. *Benchmarking Neural Network Robustness to Common Corruptions and Perturbations*. 2019. arXiv: 1903.12261 [cs.LG].

[34] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI." In: *Information Fusion* 58 (2020), pp. 82–115. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2019.12.012. URL: https://www.sciencedirect.com/science/article/pii/S1566253519308103.

[35] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization." In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL: http://dx.doi.org/10.1007/s11263-019-01228-7.

[36] Jost Tobias Springenberg et al. *Striving for Simplicity: The All Convolutional Net*. 2015. arXiv: 1412.6806 [cs.LG].

[37] Subutai Ahmad et al. "Unsupervised real-time anomaly detection for streaming data." In: *Neurocomputing* 262 (2017). Online Real-Time Learning Strategies for Data Streams, pp. 134–147. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.04.070. URL: http://www.sciencedirect.com/science/article/pii/S0925231217309864.

[38] Ilya Daylidyonok, Anastasiya Frolenkova, and Aleksandr Panov. "Extended Hierarchical Temporal Memory for Motion Anomaly Detection: Proceedings of the Ninth Annual Meeting of the BICA Society." In: Jan. 2019, pp. 69–81. ISBN: 978-3-319-99315-7. DOI: 10.1007/978-3-319-99316-4_10.

[39] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF." In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[40] Sangmin Oh et al. "A large-scale benchmark dataset for event recognition in surveillance video." In: *CVPR 2011*. 2011, pp. 3153–3160. DOI: 10.1109/CVPR.2011.5995586.

[41] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[42] Alexander Kirillov et al. *PointRend: Image Segmentation as Rendering*. 2020. arXiv: 1912.08193 [cs.CV].

[43] Jia Deng et al. "ImageNet: A large-scale hierarchical image database." In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[44] Ayoola Olafenwa. "Simplifying Object Segmentation with PixelLib Library." In: 2021.

[45] Trine B. Haugen et al. "VISEM: A Multimodal Video Dataset of Human Spermatozoa." In: *Proceedings of the 10th ACM on Multimedia Systems Conference*. MMSys'19. Amherst, MA, USA: ACM, 2019. ISBN: 78-1-4503-6297-9. DOI: 10.1145/3304109.3325814. URL: http://doi.acm.org/10.1145/3304109.3325814.