Hierarchical Temporal Memory 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 Memory for Anomaly Detection in Videos

A subtitle of your thesis

Vladimir Monakhov

Hierarchical Temporal Memory 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 accidents on a 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 advanced deep learning models, where the latter has experienced increased popularity in the past few years.

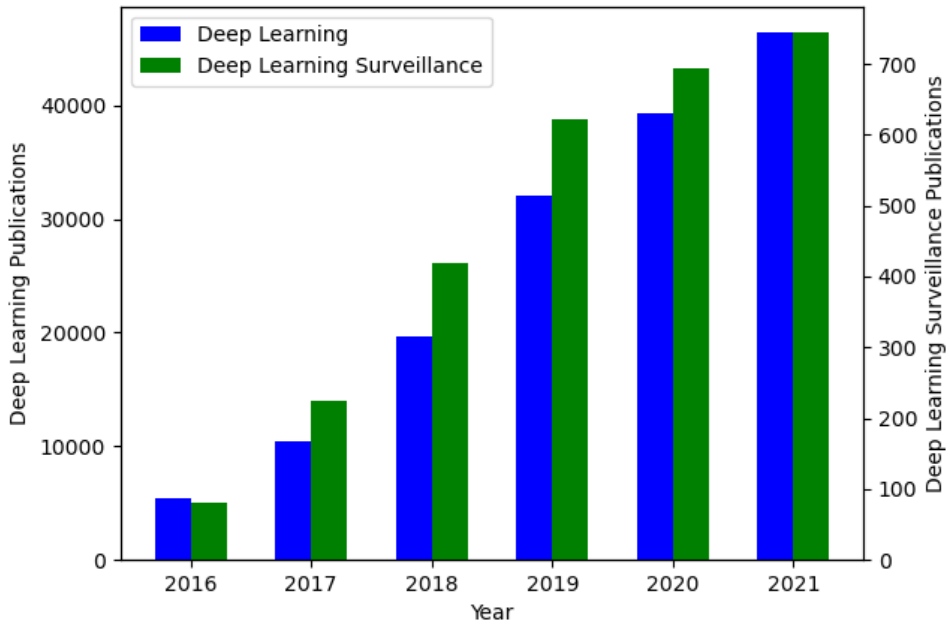


Figure 1.1: The increase in publications mentioning the terms "deep learning" and "deep learning surveillance" [2]

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 tasks 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 tasks, 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 this thesis. This thesis will therefore be relatively limited in scope.

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 and comparison.

Last but not least, the HTM theory described in this thesis is not the first generation [3], which was probabilistic in nature. The HTM theory described in this thesis is actually the third generation[4, 5], which builds upon the second generation[6, 7]. The second generation is often referred to as Cortical Learning Algorithms (CLA), which made the move from probabilistic modelling to Sparse Distributed Representation. The third generation builds upon the second generation by introducing concepts such as sensorimotor inference and The Thousand Brains Theory. The first generation had fundamentally different inner workings, 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. FINISH LATER.

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, supported with visualizations and a simpler language, making it easier for people with a machine learning background to understand. It is also important to mention that not only does this thesis include the main HTM research published, but it also contains little tidbits of community research and ideas which are otherwise hard to come by.

Additionally, during the development of the software used in 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 [8].

1.5 Thesis Outline

Thesis summary goes here?

Chapter 2

Background

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

2.1 Deep Learning

As mentioned in the introduction, deep learning has seen increased popularity in the past few years. Surprisingly enough, the field of deep learning can be traced back to 1958 when the Perceptron[9, 10] was first introduced.

2.1.1 Perceptron

The perceptron[9, 10] was a machine learning algorithm that was based on a simplification of the theory, at the time, about the inner workings of a neuron. The perceptron consists of three parts; the inputs $\mathbf{x}$, the weights $\mathbf{w}$, and the unit step activation function.

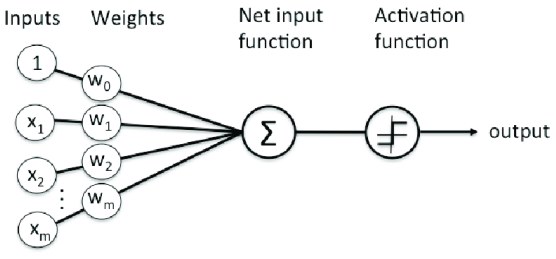


Figure 2.1: Perceptron

It also requires a label y corresponding to the input $\mathbf{x}$. The perceptron predicts the label of the input $\mathbf{x}$ to be $\hat{y} = sign(\mathbf{x} \cdot \mathbf{w})$. If $\hat{y} \neq y$ then it updates the weights $\mathbf{w} = \mathbf{w} + y\mathbf{x}$, otherwise it leaves $\mathbf{w}$ unchanged. This is performed until the perceptron reaches convergence, which would happen when all the inputs can be correctly classified. In other words, the perceptron requires that the inputs are linearly separable.

As shown by Minsky and Papert [11], the perceptron was able to solve linearly separable problems such as the OR function, but was unable to solve the XOR function. For the latter, it was theorized by others that stacking perceptrons in multiple layers, known as a Multilayer Perceptron (MLP) which is shown in Figure 2.2, would be able to solve more complex problems such as the aforementioned XOR function[12].

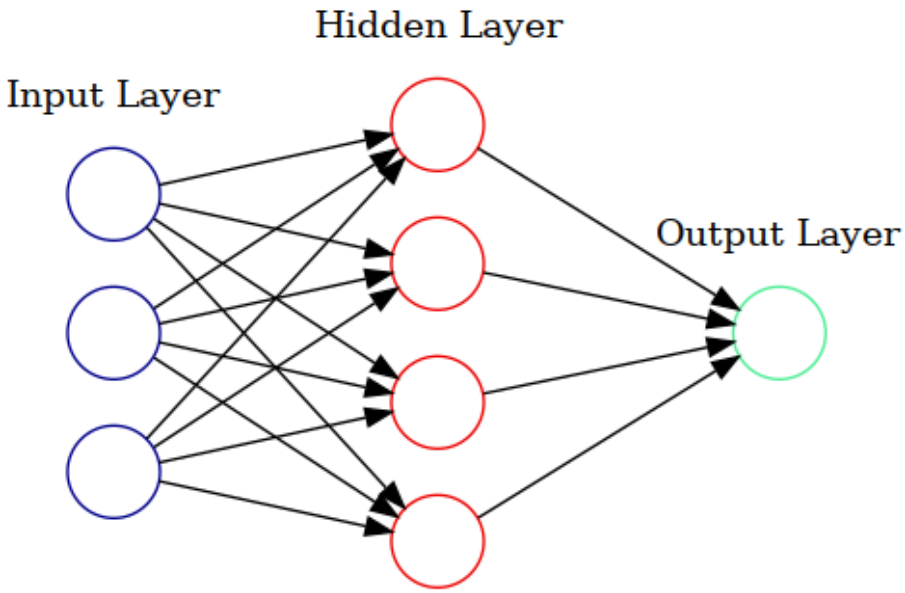


Figure 2.2: Example of a MLP.

Unfortunately, the work by Minsky and Papert [11] lead to the misconception that a MLP would have the same limitations as a single perceptron, and that further progress was impossible, which lead to the perceptron and other neuron-based approaches being largely abandoned[12]. This misconception, combined with a lack of computing power and no good approaches to train a MLP, lead to a decline in the research of neuron-inspired approaches.

It was not until the late 1980s[12] that the MLP approach experienced a revival, led by the increase in computational power and the introduction of backpropagation by Rumelhart, Hinton, and Williams [13], which allowed multilayer networks to be easily trained, and gave birth to modern neural networks.

2.1.2 Backpropagation

Backpropagation[13], which is shorthand for "backward propagation of errors", is a method for efficiently calculating the gradient of the error function, so that it is possible to adjust each individual weight with the purpose of lowering the error. The error function is a function that

quantifies the difference between a prediction and the corresponding label, and is differentiable. An example of such an error function, is the Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The result is that no longer did researchers have to manually engineer features, but could instead apply backpropagation to have the neural network automatically learn internal representations that expressed nontrivial features. An example is that no longer did researcher have to use line detection algorithms, instead the MLP could learn to represent a collection of pixels as a line automatically.

2.1.3 Neural Networks

Modern MLP networks, often referred to as just "neural networks", are similar to the MLP networks made in the 80s. The main difference is that each neuron in a MLP can have an arbitrary activation function, such as the sigmoid function and the Rectifying Linear Unit (ReLU)[14] function, which adds nonlinearity into the neural network and allows it to solve complex problems. They also contain more hidden layers, as seen in Figure 2.3, hence the term *deep* learning.

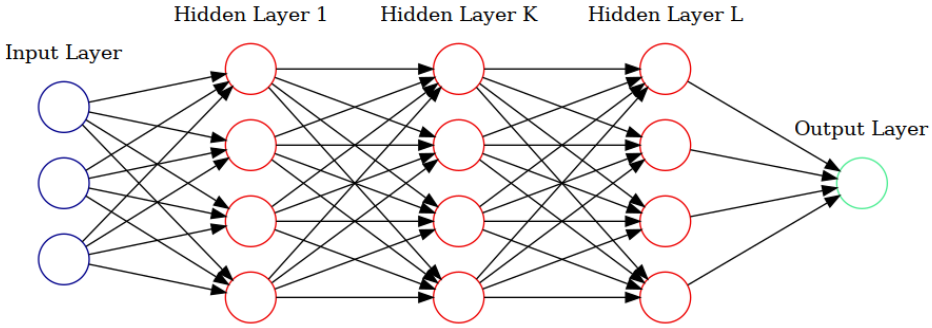


Figure 2.3: Example of a neural network.

2.1.3.1 Learning

Neural networks use gradients to learn. The gradients are first calculated using backpropagation which are then used to update the weights with the goal of reducing the error. This can be achieved using gradient descent, where one would calculate the gradient and descend towards the optimum using the entire dataset, but this is computationally expensive. An alternative is to use an optimizer such as Stochastic Gradient Descent (SGD)[15], which calculates the gradient and descends using a random subset of the dataset. A visual comparison between the two methods can be observed in Figure 2.4:

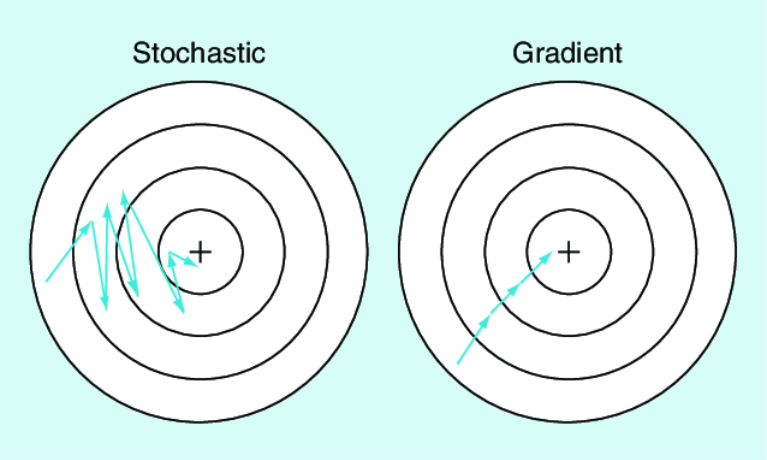


Figure 2.4: Comparison between SGD and gradient descent [16]

Yet with all the advancements within the field of neural networks, it still did not have the popularity that can be observed today. There are several reasons for this, one of them arguably being the introduction of the modern Support Vector Machine (SVM)[17], which stole the highlight.

The other reason is that neural networks did not have a lot of applications at the time, this being due to a lack of invariances which caused poor performance in machine vision, and no proper way to represent features across time which caused poor performance in time-series analysis. There was also a lack of regularization methods, which caused the models to overfit and generalize poorly. Additionally, large networks suffered from vanishing or exploding gradients.

In order to improve upon regularization, techniques such as Dropout[18], Batch Normalization (BN)[19], and Data Augmentation[20] were introduced. New neural network architectures were invented, such as the Autoencoder[21], that forces the model to construct an internal representation under constraints, which has a regularizing effect.

As for representing features across time, architectures such as Recurrent Neural Networks (RNNs)[22, 23] were invented. Later, attention[24]-based models, such as the Transformer[25] model, were introduced.

The vanishing and the exploding gradient problem was alleviated by implementing, among other techniques, residual connections[26] between layers.

2.1.4 Convolutional Neural Networks

In an attempt to solve the problems regarding invariance and to improve the performance in machine vision tasks, Convolutional Neural Networks (CNNs) were introduced by Lecun et al. [27] in 1998. A CNN offers several properties that are useful, such as translational invariance [27]. 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.

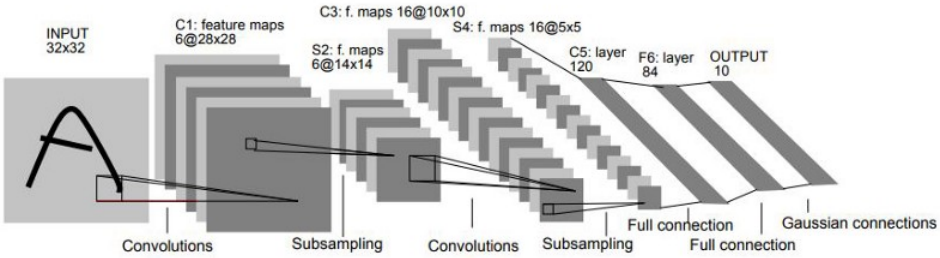


Figure 2.5: LeNet[27], an example of a CNN.

However, the progress within the context of machine vision stagnated due to a lack of computing power when training large models as well as a lack of sufficiently large datasets. This stagnation lasted until 2012 when AlexNet was introduced by Krizhevsky, Sutskever, and Hinton [28], which marked a major turning point in the history of deep learning, as their record-breaking results proved that deep learning was they way forward for solving complex machine learning problems.

In addition to creating a unique CNN architecture, they also made it run on a GPU and made the code publicly available. This was not the first case of running deep learning on a GPU[29], but it can be argued that it was this paper that popularized it. Using GPUs meant that a vast amount of computational power, for the purpose of training deep learning models, became unlocked. This also paved the way for frameworks such as PyTorch[30] and Tensorflow[31], which have democratized deep learning and made it into what it is today.

2.1.5 Generative Models

Some of the latest progress within deep learning can be attributed to generative models, such as Generative Adversarial Networks (GANs)[32] and Variational Autoencoders (VAEs)[33]. Generative models work by taking some input, and then generating realistic output. The input could be anything from random noise to a real sample.

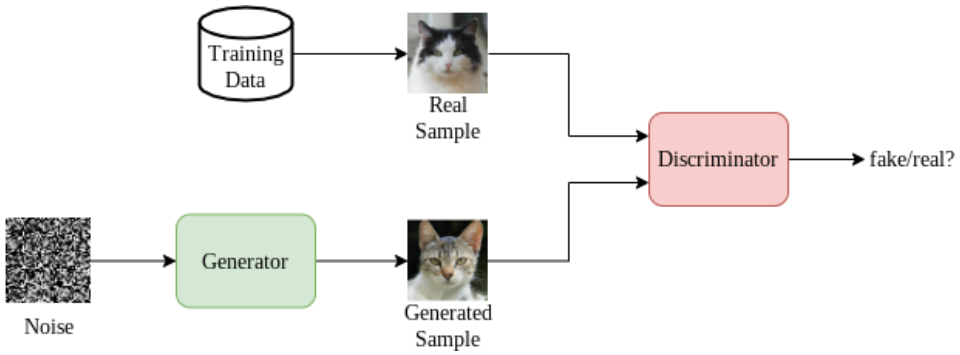


Figure 2.6: Example of a GAN.

The idea is that the models learn a distribution which describes the data domain represented by the training data, and can then generate synthetic data from that distribution by simply sampling it.

2.1.6 Disadvantages

A disadvantage for deep-learning models in general is that they are susceptible to noise in the dataset [34, 35], which leads to decreased classification accuracy and poor prediction results.

Due to the nature of training deep learning models, they are also in most cases not self-supervised and therefore require constant tuning in order to stay effective on changing data. Not to mention that they require a lot of data before they can be considered effective, and that performance increases logarithmically based on the volume of training data[36].

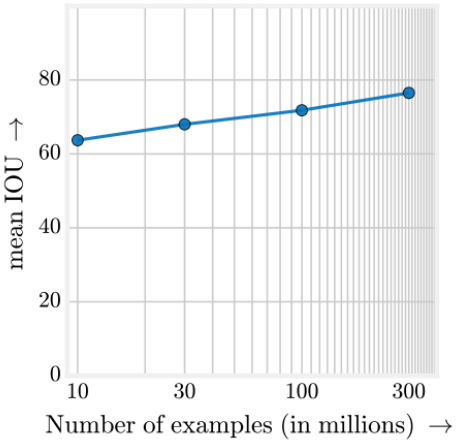


Figure 2.7: Accuracy increase in relation to size of dataset [36].

They also suffer from issues with out-of-distribution performance, where a model might perform great on the dataset it is tested on, but performs poorly when deployed in real life. This could be caused by selection bias

in the dataset or when there are differences in the causal structure between the training domain and the deployment domain[37].

2.1.6.1 Explainability

As stated by Barredo Arrieta et al. [38], 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 make those predictions. 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.

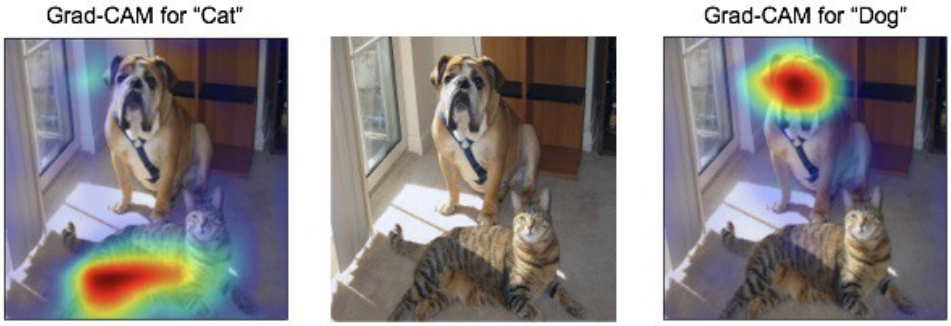


Figure 2.8: Grad-CAM visualization for cat and dog.

Approaches such as Grad-CAM [**GradCAM**] and Guided Backpropagation [39] 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 [38].

2.2 Anomaly detection

As reviewed by Pang et al. [40], 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. Michałowska et al. [41] and Fan et al. [42] 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. Devi, Biswas, and Purkayastha [43] 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 the one introduced by Datta, Muthiah, and Ramakrishnan [44] have been proposed to alleviate the problem.

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

2.2.1 Deep Learning and Anomaly Detection

The current state-of-the-art algorithms for anomaly detection are numerous, but the main approach is achieved by using deep learning [40]. 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 [40, 45, 46] 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.

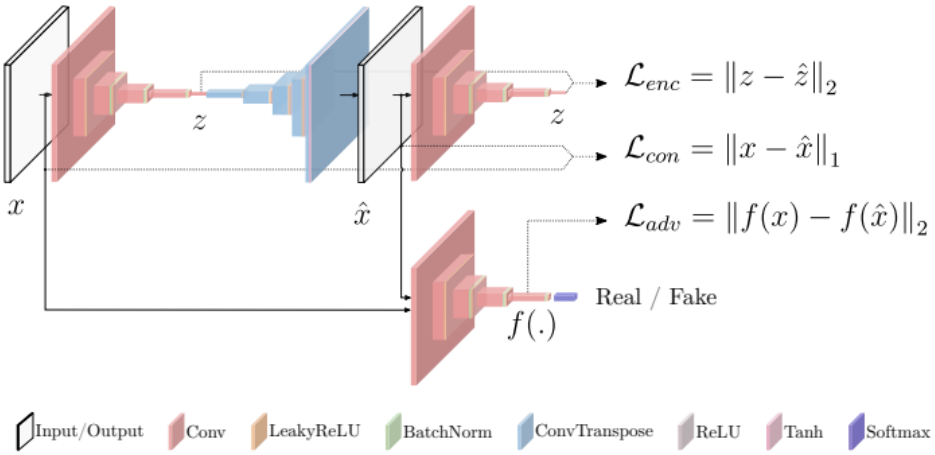


Figure 2.9: GANomaly [45], a variation of GAN for Anomaly detection

Another common approach is autoencoders, which aim to minimize the reconstruction error from a learned feature representation space [47, 40, 46]. 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.

As previously stated, generative models suffer from a lack of explainability, and since generative models make up most of the state-of-the-art approaches in anomaly detection, it becomes safe to say that there is a lack of explainability in the field of anomaly detection in videos.

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.

2.2.2 Smart Surveillance

Smart surveillance, which is the use of automatic video analysis specifically in surveillance, has seen rapid development since its inception. Zhu, Chen, and Sultani [46] present and summarize recent progress for anomaly detection in video for surveillance purposes, where the most promising methods are achieved by using convolutional Autoencoders (AEs) and GANs. The results show that the deep learning approaches have a high degree of accuracy, and are consistently improving. This is not surprising given that generative models are popular in anomaly detection, as mentioned in Section 2.2.1.

The paper also discusses problems with using deep learning approaches for anomaly detection, and further emphasizes the complexities mentioned in Section 2.2. One of the examples that it uses is about a bicycle on campus being wrongly classified as an anomaly, which is related to the aforementioned issue with the non-realistic requirement that the training data must contain all possible classes of non-anomalous events.

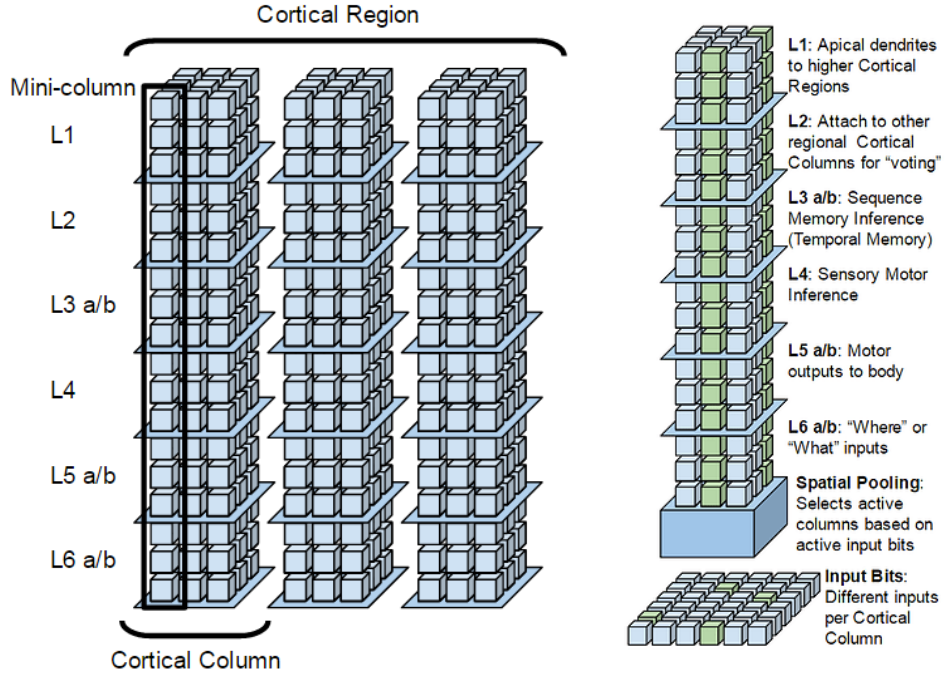
2.3 Hierarchical Temporal Memory

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. Hawkins et al. [48] 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 [49].

2.3.1 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.



(a) Cortical Region.

(b) Cortical Column.

Figure 2.10: Visualization of the cortical region structure in the neocortex according to HTM theory. Note that current HTM implementations only model L2/L3, but can be extended to model L4 as well [50]. Source: [51]

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.

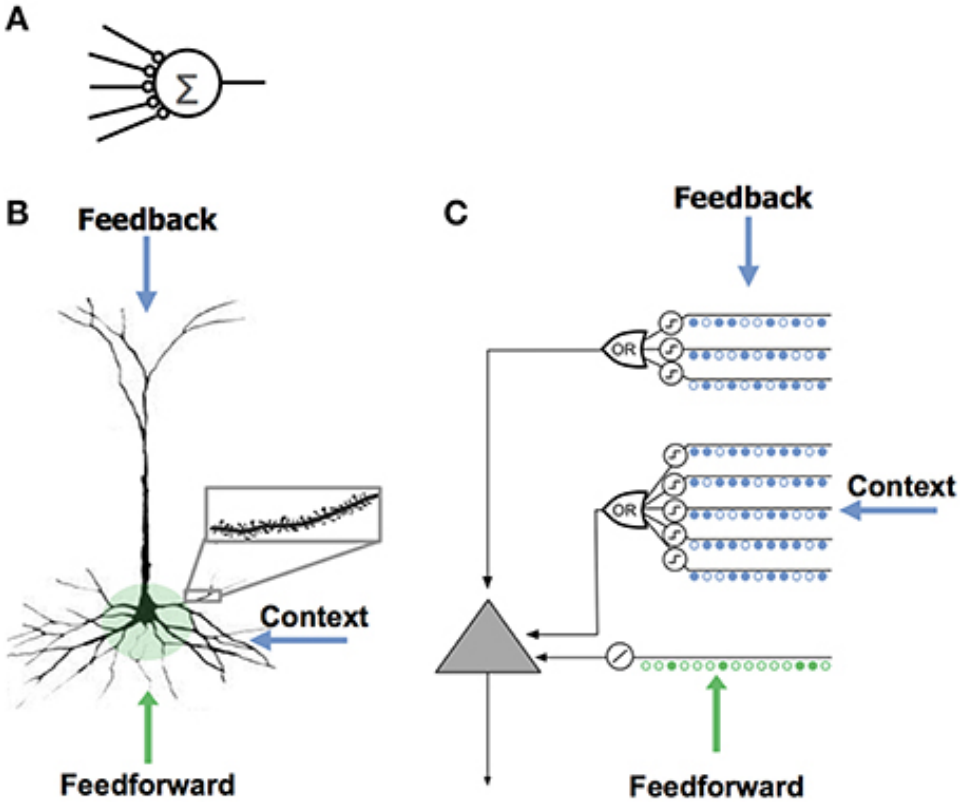


Figure 2.11: Comparison of neurons [52]: Traditional Machine Learning (A), Biological Neuron (B), HTM Neuron (C). Source: [48]

The HTM neuron has three inputs [52]:

- 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.

2.3.2 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.

2.3.3 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.

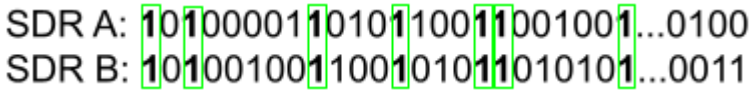


Figure 2.12: Semantic similarities between the two SDRs A and B

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.

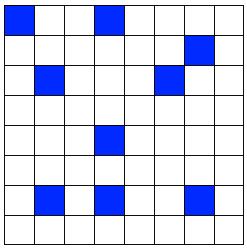


Figure 2.13: Example representation of an SDR with a length of 64 and a sparsity of 14.1%, visualized as a 2D grid.

2.3.4 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 2.14 shows the cyclical date encoder.

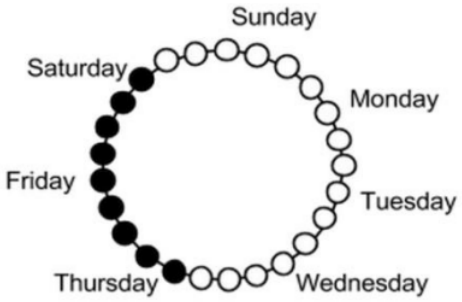


Figure 2.14: Visualization of a cyclical date encoder, which is currently encoding Friday. Note that also Thursday and Saturday are included in the encoding of Friday in order to emphasize that Thursday and Saturday are both equally distanced from Friday. Source: [48]

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

2.3.4.1 Encoding Visual Data

Several approaches for encoding visual data have been proposed. An example is a neuroscientifical approach which replicates how the eye works[53], another example is the approach by Fallas-Moya and Torres-Rojas [54] which uses scale-invariant feature transform (SIFT) to find points of interest in images and encode that information as an SDR. A simple threshold, as seen in Figure 2.15, can also be applied to extract information into an SDR-friendly format.

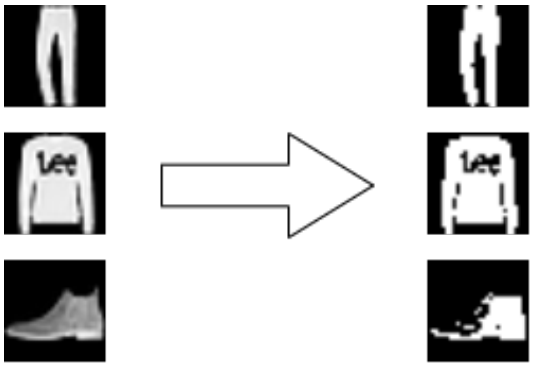


Figure 2.15: Example thresholding on the Fashion MNIST dataset TODO.

There are also deep learning approaches such as the one proposed by Zou et al. [55] which uses a Convolutional Neural Network (CNN) as part of the encoder. It creates an SDR by storing the top n-features in a feature map as ones and set the rest to zeros in order to achieve binary values suitable for the SDR.

The reason for why a direct binary threshold 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 threshold encoders also lead to loss of information, and is hard to perform for complex objects.

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. Figure 2.16 shows only a small subset of all the feature maps in a single layer, which highlights how intangible this approach is.

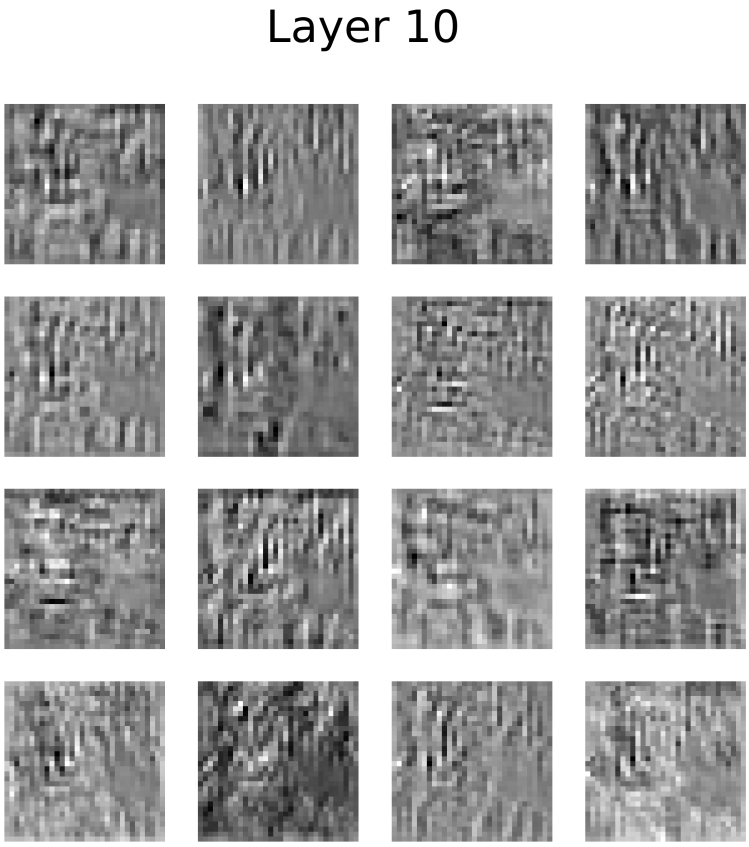


Figure 2.16: Select few feature maps in the 10th layer of ResNet18[26] trained on ImageNet[56].

Additionally, minor variations in the feature map would cause major variations in the resulting SDR. Alternatively, one could follow Zou et al. [55] 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 in the feature maps.

2.3.5 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

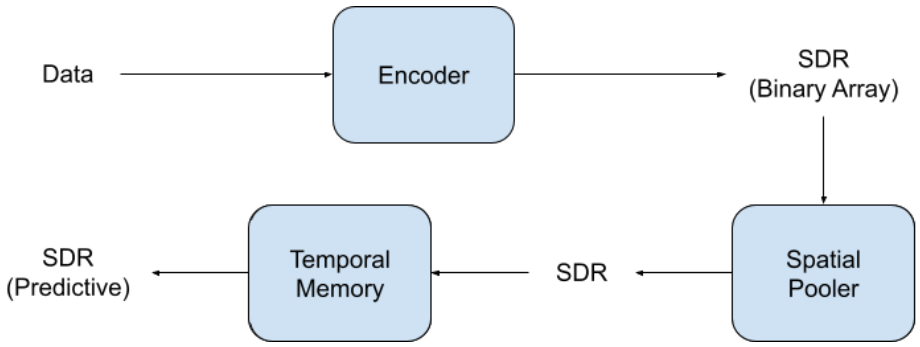


Figure 2.17: The HTM Pipeline. A common next-step could be to use a classifier to convert the predictive SDR into a classification.

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.

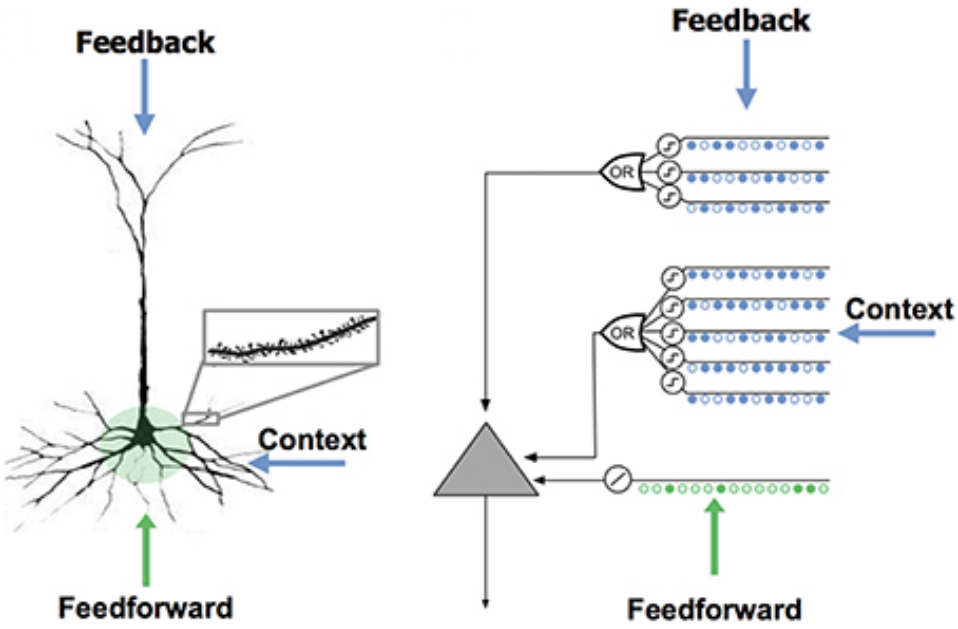


Figure 2.18: Real neuron and HTM neuron.

Figure 2.18 shows that 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.

2.3.5.1 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.

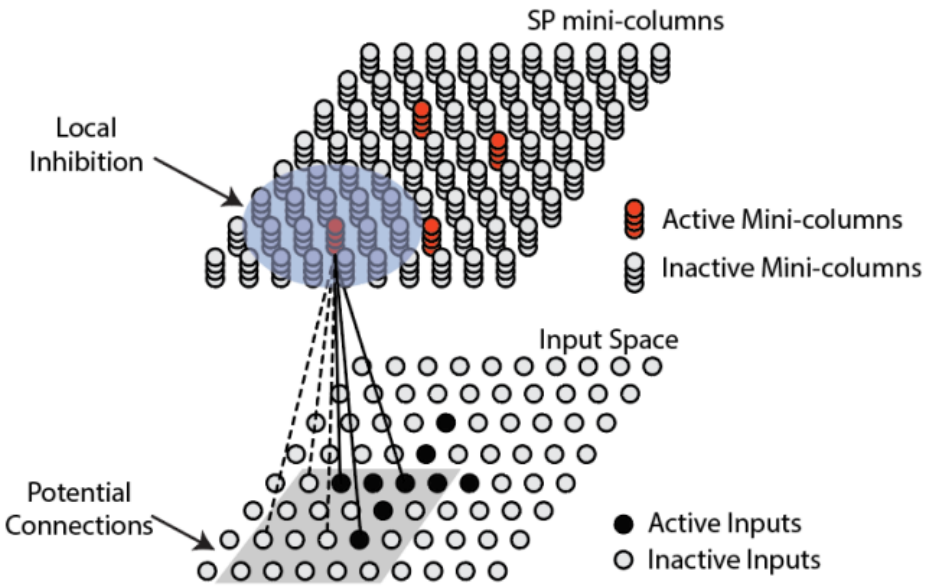


Figure 2.19: Visualization of the Spatial Pooler (SP) and the potential pool of one of its columns. Source: [57]

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 2.20 visualizes the mechanism that determines the output in the SP.

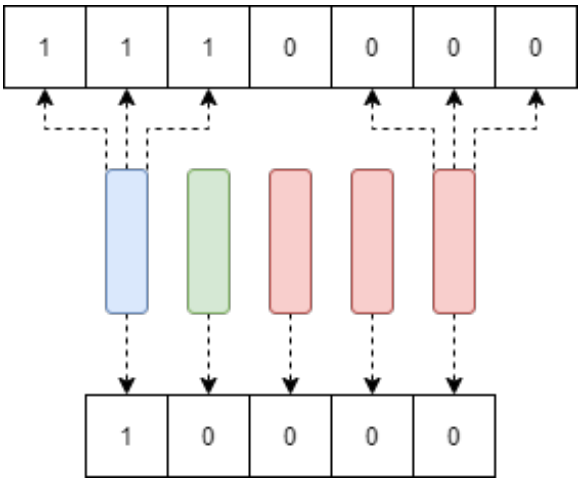


Figure 2.20: This figure illustrates how a spatial pooler works. All connections are above the stimulus threshold. The receptive field is 3 bits wide for each column. Overlap score threshold is 1, and $n = 1$. Red means inactive, green means active, and blue means active and selected.

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.

2.3.5.2 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 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 Temporal Memory (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.

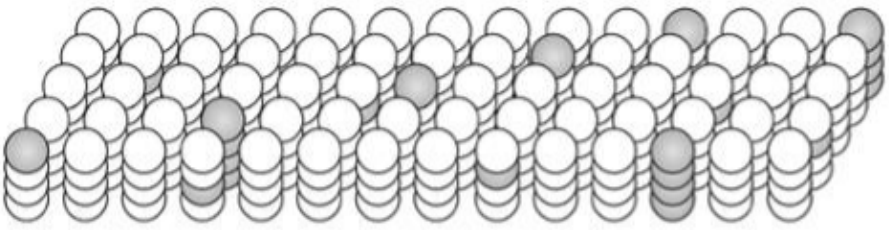


Figure 2.21: Visualization of the TM, with number of cells equal 4. Some columns are bursting. Source: [48]

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 (see Figure 2.22). Otherwise, a cell is inactive.

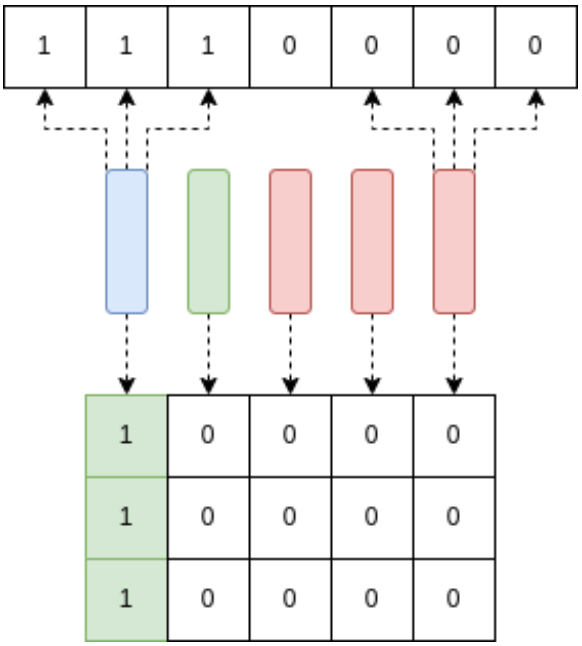


Figure 2.22: Expanded SP example with TM component where the number of cells is set to 3. The leftmost column is bursting (all 3 cells activated in green) due to the active SP output and due to containing no predictive cells.

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.

Since the TM can only grow synapses to cells that were previously active, the TM struggles to express sequences of patterns over multiple timesteps, as has been discussed on HTM forums [58]. The solution has been to also encode some temporal information, such as the time of day [59, 58], so that it can use timestamps as anchor points for its contexts.

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.

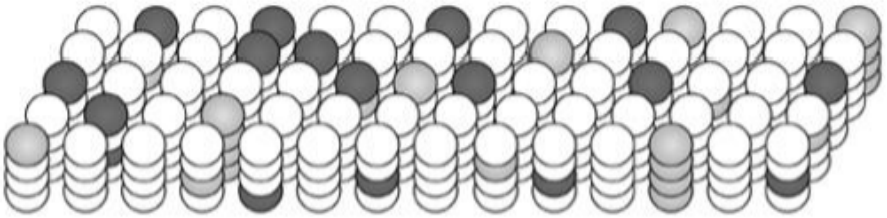


Figure 2.23: Visualization of the TM and the three states. Active in black, predictive in gray, and inactive in white. Source: [48]

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_{t-1} it made in the previous time step with the actual pattern A_t 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 [60]. 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.

2.3.6 Use Cases

The general use case for HTM is to perform anomaly detection. More specifically, Numenta has made example applications showcasing how HTM can be used in practice [61]:

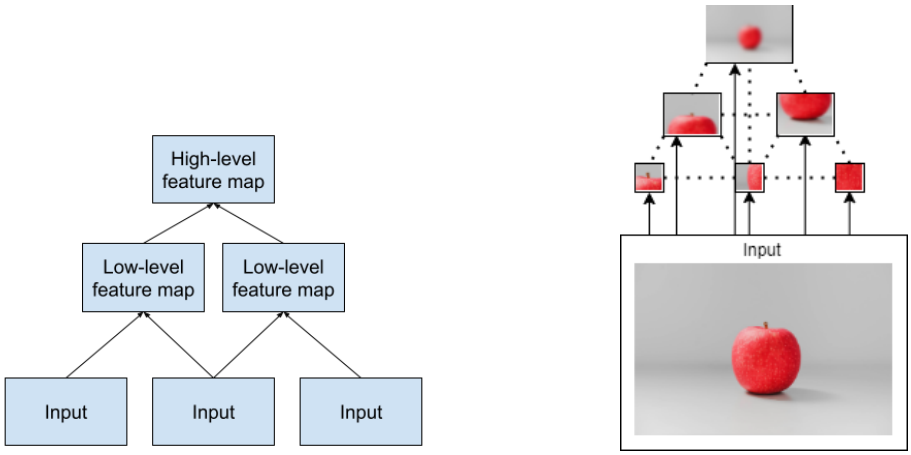
- **Rogue Behavior Detection** which models normal behavior and detects anomalies, such as unusual use of files in a network [62].
- **Geospatial Tracking** which detects anomalies in the movement of people, objects, or material, using speed and location data [63].
- **Financial Monitoring** which detects anomalies in publicly traded companies by continuously modelling stock price, stock volume, and Twitter volume [64].

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 [65].

2.3.7 The Thousand Brains Theory

One of the newest advancements in HTM Theory is the introduction of the Thousand Brains Theory. Hawkins et al. [5] 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 Thambawita et al. [66]. 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 2.24: Comparison of classical hierarchy and the hierarchy introduced by the Thousand Brains Theory

This new type of hierarchy is also quite similar to some state-of-the-art image recognition deep learning architectures such as InceptionNet[67] and Feature Pyramid Networks[68], 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.

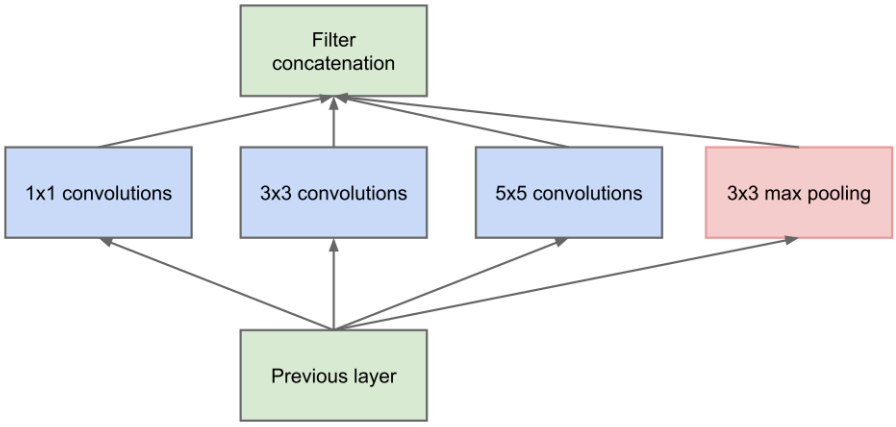


Figure 2.25: How the Inception [67] architecture combines multiple filters

While the Thousand Brains Theory is not yet technically 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.

2.3.8 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.

Ahmad et al. [59] 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.

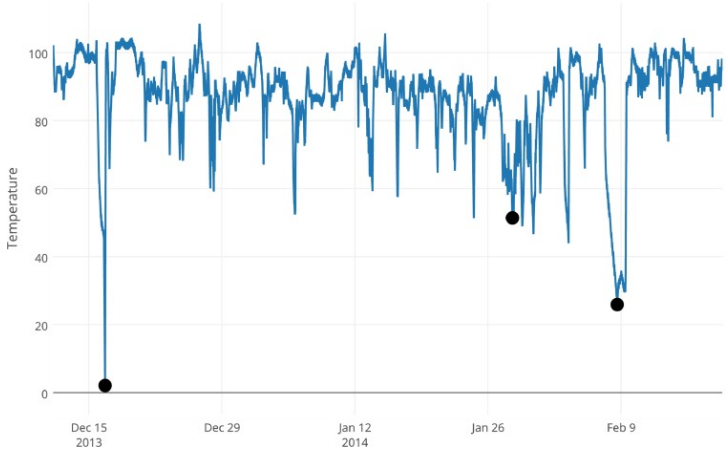


Figure 2.26: Temperature anomaly detection from Ahmad et al. [59].

For high-dimensional anomaly detection; Daylidyonok, Frolenkova, and Panov [69] used a HTM system to find anomalous frames in videos of motions (Figure 2.27). 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.

Figure 2.27: Example motion frame[69].

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.

2.4 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 ORB[70] 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 [71] dataset.

Figure 3.1: Segmentation result of cars, which is suited to be used as an SDR. Original frame taken from [71].

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.

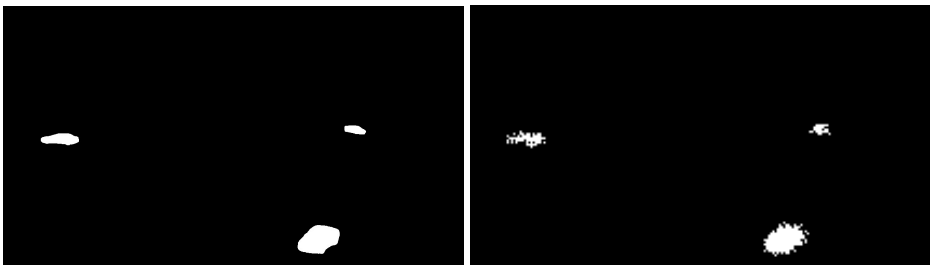


Figure 3.2: The SDR and its corresponding SP representation. Note that the SP is untrained.

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 Improvements

3.2.1 Invariance

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.

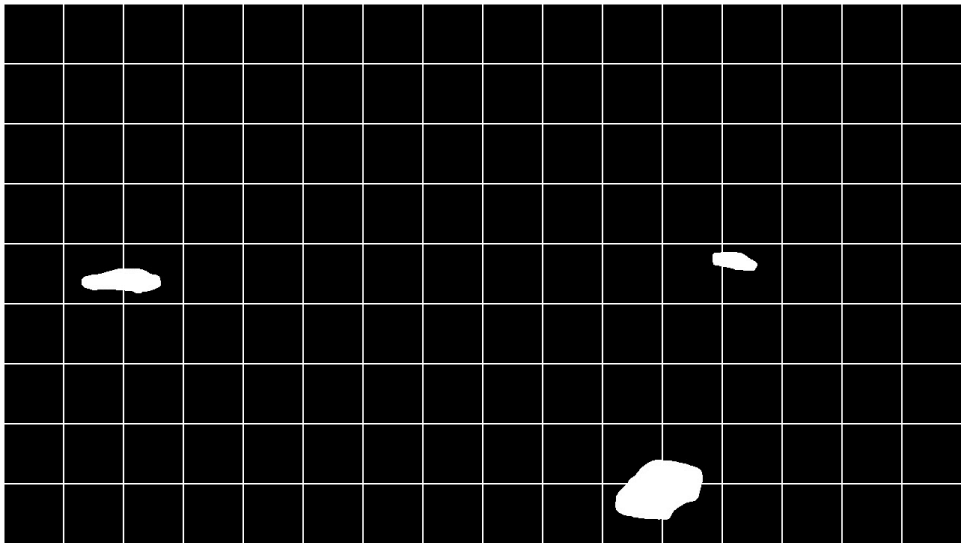


Figure 3.3: The encoder output divided into a grid.

3.2.1.1 Aggregation Function

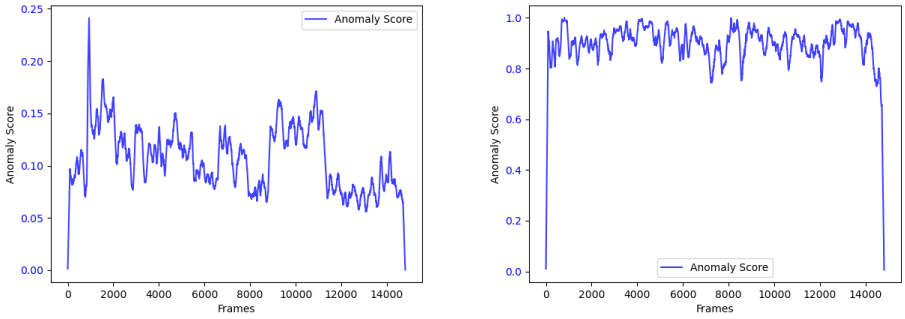
Selecting the correct aggregation function is important because it affects the final anomaly output. For instance, it might be tempting to use the mean of all the anomaly scores as the aggregation function, but this leads to problems with normalization, meaning that an overall anomaly score of 1 is hard to achieve due to many cells having a zero anomaly score. In fact, it becomes unclear what a high anomaly score is anymore. Using the mean also means that anomalies that take up a lot of space will be weighted more than anomalies that take up a little space, which might not be desirable.

To solve the aforementioned problem and if the data has little noise, a potential aggregation function could be the non-zero mean:

$$X : \{x \in \mathbb{R} : x > 0\}$$

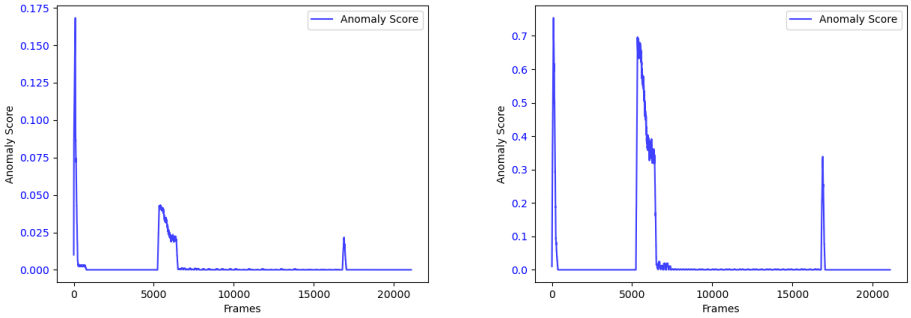
$$anomScore = \begin{cases} \frac{\sum_{x \in X} x}{|X|} & \text{if } |X| > 0 \\ 0 & \text{otherwise} \end{cases}$$

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



(a) Mean. (b) Non-zero mean.

Figure 3.4: How the two aggregation functions perform on the same noisy data.



(a) Mean. (b) Non-zero mean.

Figure 3.5: How the two aggregation functions perform on the same clean data.

Figure 3.4 illustrates the effect of an aggregation function for noisy data, where the non-zero mean is rendered useless due to the noise. On the other hand, Figure 3.5 shows how the non-zero mean gives a clearer anomaly score when the data is clean. Especially regarding how, unlike the mean, the non-zero mean has a clearly defined range between 0 and 1.

3.2.2 Explainability

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.

3.2.3 Flexibility and Performance

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 the frame into smaller cells makes it possible to run each cell in parallel for increased performance.

3.2.4 Reviewing Encoder Rules

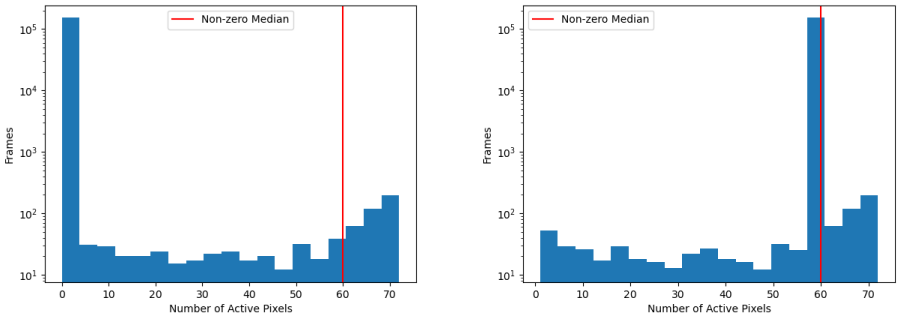
That being said, a potential problem with this approach is that the previously mentioned rules for creating a good encoder, see Section 2.3.4, may not be respected and therefore should be reviewed:

- **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 distributions seen in Figure 3.6:



(a) Without empty pattern.

(b) With empty pattern.

Figure 3.6: Distribution of number of active pixels within a cell of size 12×12, it can also be observed that it would benefit from having a minimum sparsity requirement of ~ 5.

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 run as a part of the calibration process.

The visual output resulting from these changes, which is an equally important output as the aggregated anomaly score, can be seen in Figure 3.7:

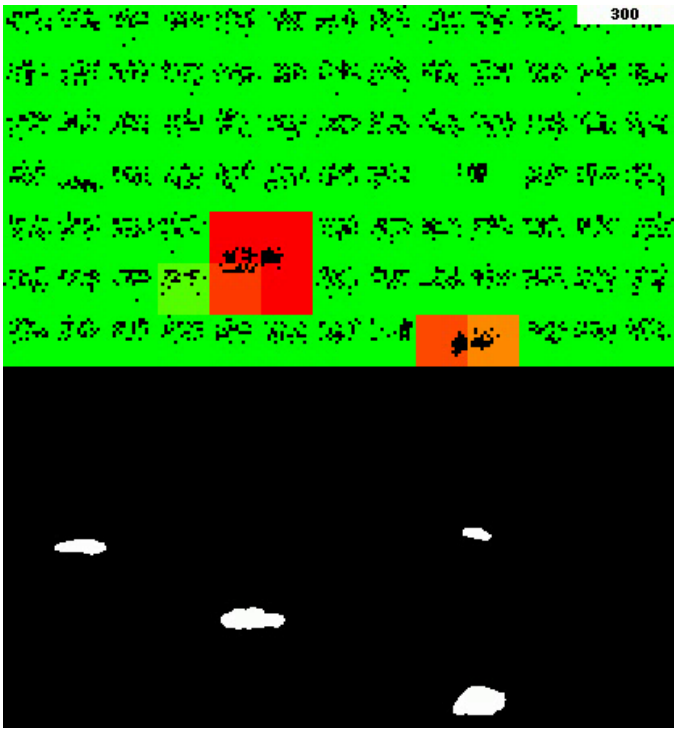


Figure 3.7: Example Grid HTM output and the corresponding input. The color represents the anomaly score for each of the cells, where red means high anomaly score and green means zero anomaly score. Two of the cars are marked as anomalous because they are moving, which is something the Grid HTM has not seen before during its 300 frame long lifetime.

Since there are now cells that are observing an empty pattern for a lot of the time in sparse data, boosting is recommended to be turned off, otherwise the SP output for the empty cells would change back and forth in order to adjust the active duty cycle.

3.2.5 Stabilizing Anomaly Output

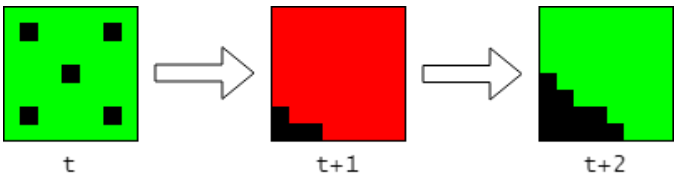


Figure 3.8: High anomaly score when an empty cell (represented with an empty pattern with a sparsity value of 5) changes to being not empty, as something enters the cell.

Another issue with the grid based approach is 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, since it does not see outside its own cell, and therefore the first frame that a car enters a cell will cause a high anomaly output. This is illustrated in Figure 3.8 where it can be observed that this effect causes

the anomaly output to needlessly fluctuate. 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, which is illustrated in Figure 3.9.

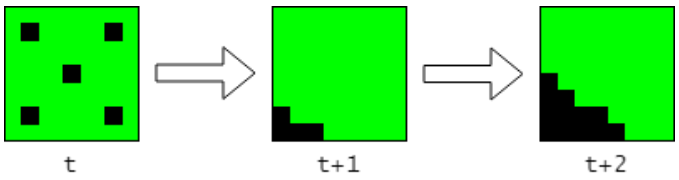


Figure 3.9: In this case, the anomaly score is ignored (set to 0) for the frame in which the cell changes state from empty to not empty.

A more proper solution could be to allow the TM to grow synapses to the TMs in the neighboring cells, but this is not documented in any research papers and might also hinder invariance.

3.2.6 Multistep Temporal Patterns

Since the TM can only grow segments to cells that were active in the previous timestep, as was mentioned in Section 2.3.5.2, it will struggle to learn temporal patterns across multiple timesteps. This is especially evident in high framerate videos, where an object in motion has a similar representation at timestep t and $t+1$, as an object standing still.

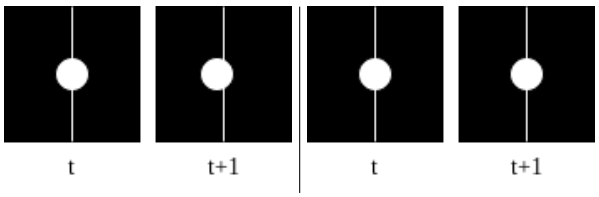


Figure 3.10: Comparison of a moving object (left) and a still object (right)in a high framerate video. They are very similar and could actually end up being represented identically by the SP.

This could cause situations where an object that is supposed to be moving, suddenly stands still, yet the TM will not mark it as an anomaly due to it being stuck in a contextual loop. A contextual loop is when one of the predictions at t becomes true at $t+1$, and then one of the predictions at $t+1$ is almost identical to the state at t, which becomes true if the object is not moving, causing the TM to enter the same state that it was in at t.

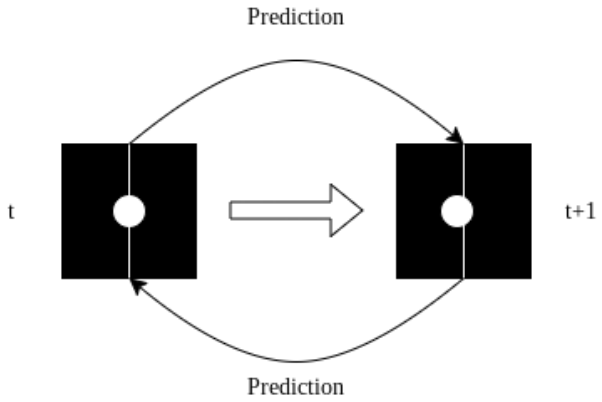


Figure 3.11: Example of a contextual loop.

A solution is to concatenate the past n SP outputs as input into the TM, which is made possible by keeping a buffer of past SP outputs and shifting its contents out as new SP outputs are inserted. This follows the core idea behind encoding time in addition to the data, which makes time act as a contextual anchor. However, in this case there are no timestamps that are suitable to be used as contextual anchors, so as a replacement, the past observations are encoded instead.

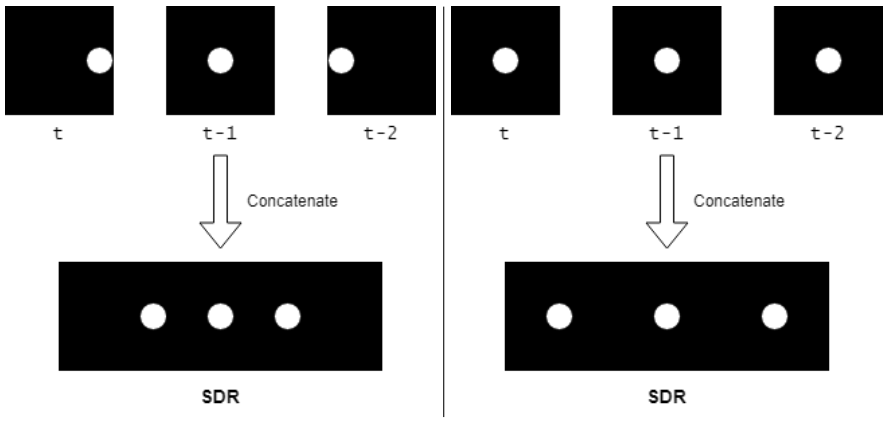


Figure 3.12: Example of concatenation with $n = 3$ when an object is moving from left to right, compared to when an object is not in motion. It can be observed that the SDRs are vastly different.

This will force the TM input, for when an object is in motion and when an object is still, to be unique. High framerate videos can benefit the most from this, and the effect will be more pronounced for higher values of n. One could feed the past n encoder outputs into the SP instead, but this will lead to a much higher computational requirement, as well as make both the SP and TM parameters to be dependent on n.

A potential side effect of introducing temporal patterns, is that because the TM is now exposed to multiple frames at once, it will be more tolerant to temporal noise.

Adding support for multistep temporal patterns to the method used by Daylidyonok, Frolenkova, and Panov [69], which is mentioned in Section 2.3.8, could improve their results.

3.3 Use Cases

The most intuitive use case is to use Grid HTM for semi-active surveillance, where personnel only have to look at segments containing anomalies, leading to drastically increased efficiency.

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.

Grid HTM could also be used to help automate labeling of anomaly datasets for deep learning. This would be similar to how older deep learning networks are used to help automate creating new labeled image datasets, where the model proposes a label for an image, which is then further refined by a human if needed.

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.

4.1.1 Data

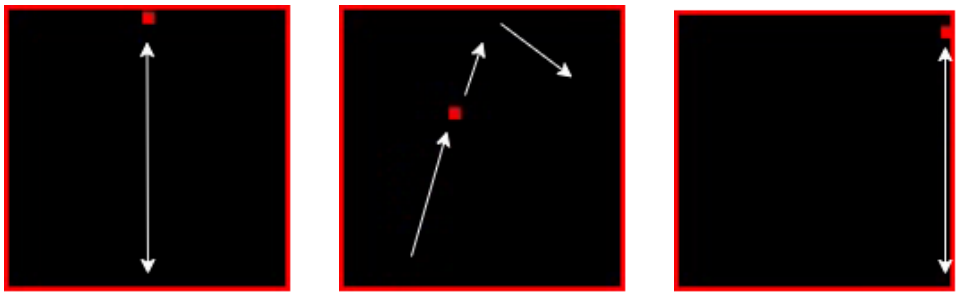


Figure 4.1: The bouncing ball test, and its three stages

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.

4.1.2 HTM

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

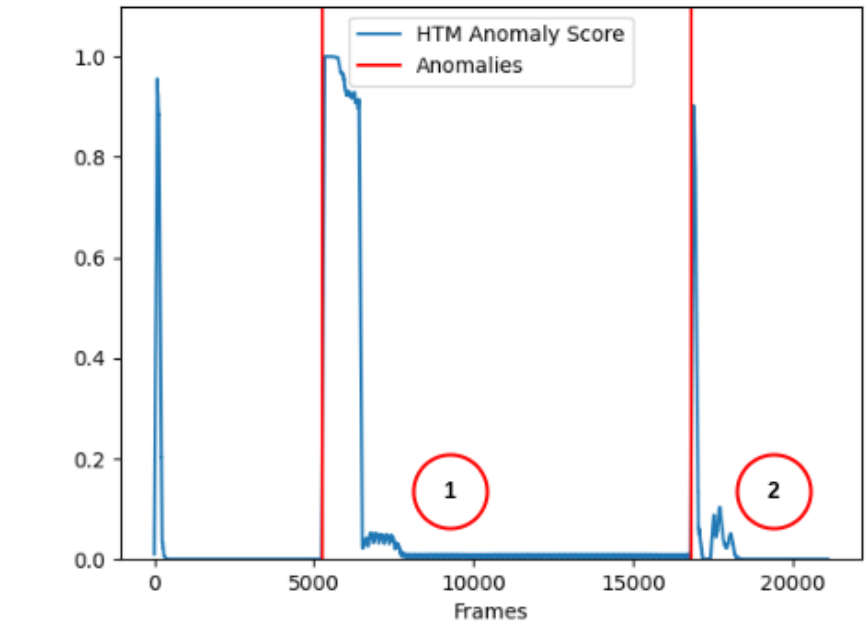


Figure 4.2: The 100-point moving average of the anomaly score in the bouncing ball experiment.

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 (1) and the anomaly spikes towards the end close to (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.

4.1.2.1 Boosting

The reason for the oscillations is due to the spatial pooler being dominated by a lucky few columns. The solution is to enable boosting, as explained in Section 2.3.5.1. This also helped with the spikes towards the end, as can be seen in Figure 4.3.

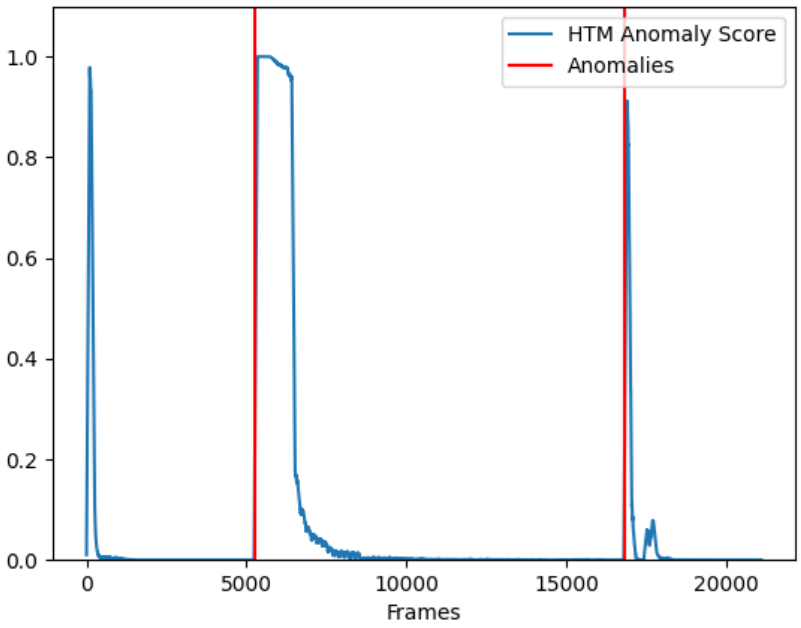


Figure 4.3: Bouncing ball with boosting enabled.

4.1.2.2 Zero Permanence Decrement

The reason for the anomaly spikes towards the end is because the spatial pooler has found an optimal representation when the ball is 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", as can be seen in Figure 4.4. That being said, the ability to decrement permanence is important in HTM systems, therefore disabling it is not always feasible.

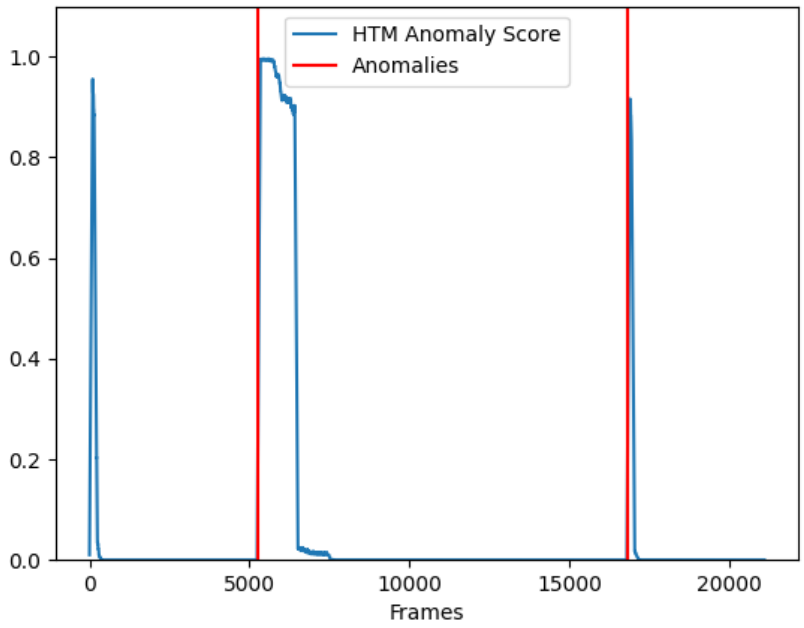


Figure 4.4: Bouncing ball without the ability of the SP to "forget".

4.1.2.3 Boosting and Zero Permanence Decrement

Finally, for the sake of interest, the bouncing ball test was performed with both boosting enabled and with zero permanence decrement. Results can be seen in Figure 4.5.

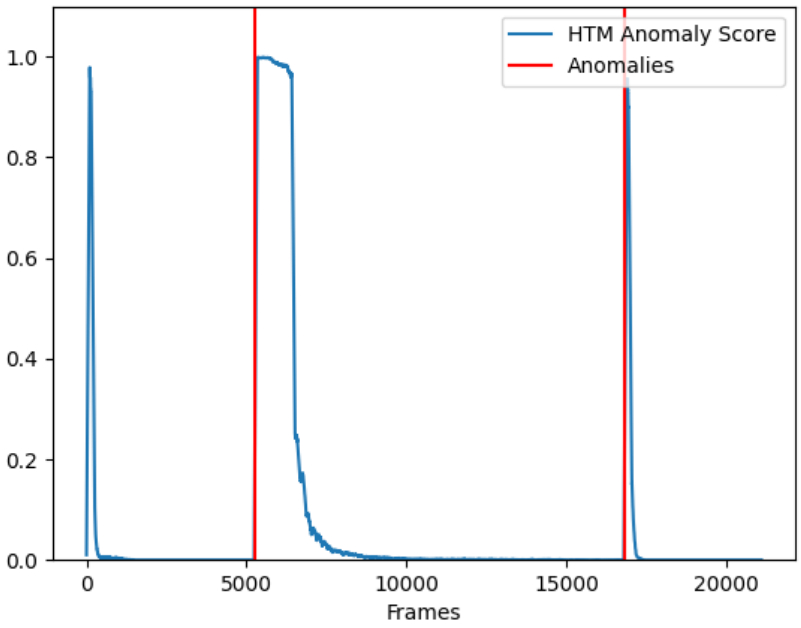


Figure 4.5: Bouncing ball without the ability of the SP to "forget" and with boosting enabled.

4.1.2.4 Parameters

Final list of parameters for reproducibility:

Parameter	Value	Notes
inputDimensions	120, 120	The shape of the entire frame
columnDimensions	60, 60	
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	60, 60	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

4.1.3 Grid HTM

This is a very simple problem which does not require invariances, making it unsuitable for Grid HTM. Grid HTM would be suitable if there were two or more independent bouncing balls, due to its improved invariance. Still, it is interesting to see how Grid HTM performs compared to normal HTM.

4.1.3.1 Results

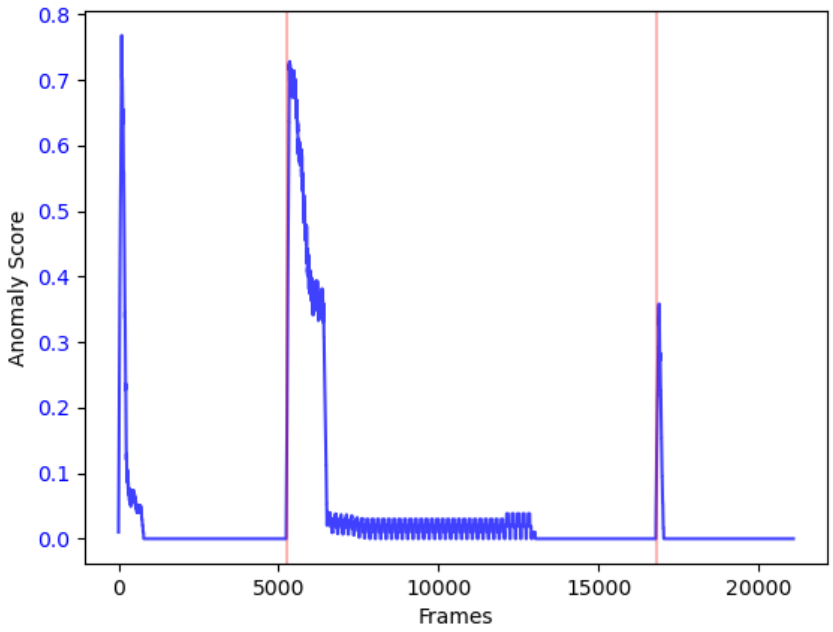


Figure 4.6: Grid HTM

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 given in Table 4.3, Table 4.4, and Table 4.5 are probably not optimal.

4.1.3.2 Parameters

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, because there is no noisy due to the controlled environment.

Parameter	Value	Notes
sp_grid_size	30, 30	Size of each grid
tm_grid_size	15, 15	Size of each SP grid output.
min_sparsity	1	
sparsity	A	Area of the bouncing ball with radius=3
temporal_size	1	Size of the multistep temporal pattern, 1 means it is effectively disabled

Table 4.3: Grid HTM specific parameters

Parameter	Value	Notes
inputDimensions	sp_grid_size	
columnDimensions	tm_grid_size	
potentialPct	0.5	Increased in order to compensate for the smaller potential pool
potentialRadius	5	
localAreaDensity	0.1	
globalInhibition	True	Set to False to enable topology
wrapAround	False	
synPermActiveInc	0.1	
synPermInactiveDec	0.001	
stimulusThreshold	2	
boostStrength	0	Causes instability in empty cells

Table 4.4: SP Parameters

Parameter	Value	Notes
columnDimensions	tm_grid_size	Same as the SP
predictedSegmentDecrement	0.003	
permanenceIncrement	0.1	
permanenceDecrement	0.001	
minThreshold	1	
activationThreshold	1	
cellsPerColumn	16	

Table 4.5: 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[71] video dataset, and was selected due to its long duration and stationary camera which can be seen in Figure 4.7.

The downside is that the video does not contain any non-technical anomalies, but consists of technical anomalies in the form of several segments with sudden frame skips in between. There is also a synthetic anomaly introduced in the form of a frame repeat lasting a couple of seconds, essentially "freezing" time, in order to test whether Grid HTM is able to understand how objects should be moving in time.

Figure 4.7: Example frames from the selected video.

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 PointRend[72] with a ResNet101[26] backbone, pretrained on ImageNet[56], and implemented using PixelLib[73].

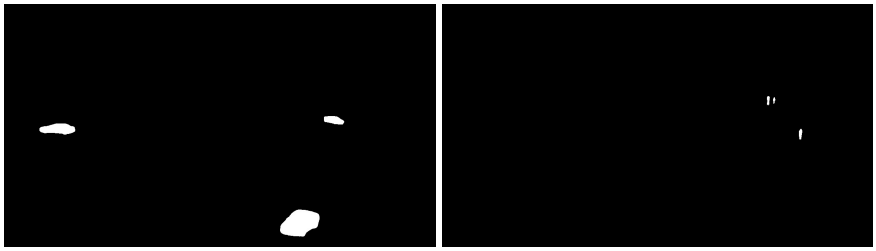


Figure 4.8: Example segmentation of cars and persons.

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.

4.2.1 Parameters

TODO

4.2.2 Results

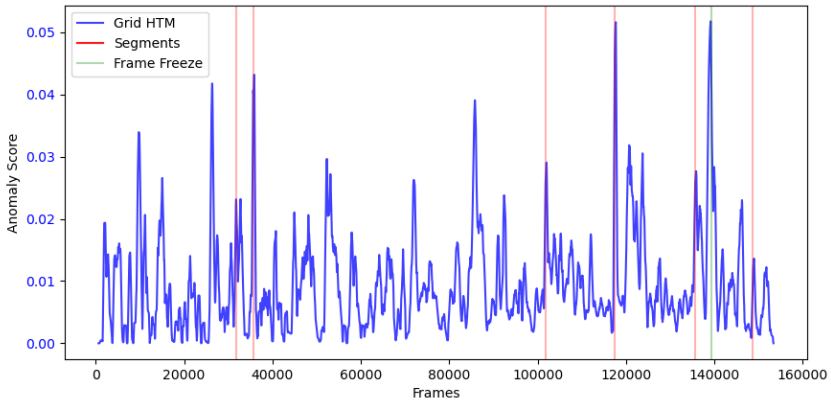


Figure 4.9: Anomaly Scores.

We can see in Figure 4.9 that the Grid HTM is detecting when segments begin and end, however it is not possible to use a threshold value to isolate them, and they also have vastly different anomaly scores between each other. This is due to the way the aggregation function works, which means that the anomaly output is dependent on the physical size of the anomaly.

With that in mind, with the aggregation functions presented in this thesis, it is safe to conclude that looking at the anomaly score output is meaningless for complex data such as a surveillance video. This does not mean that Grid HTM is completely useless in that regard, and this can be observed by looking at the visual output of the Grid HTM. The visual output during which the first segment anomaly occurs can be seen in Figure 4.10.

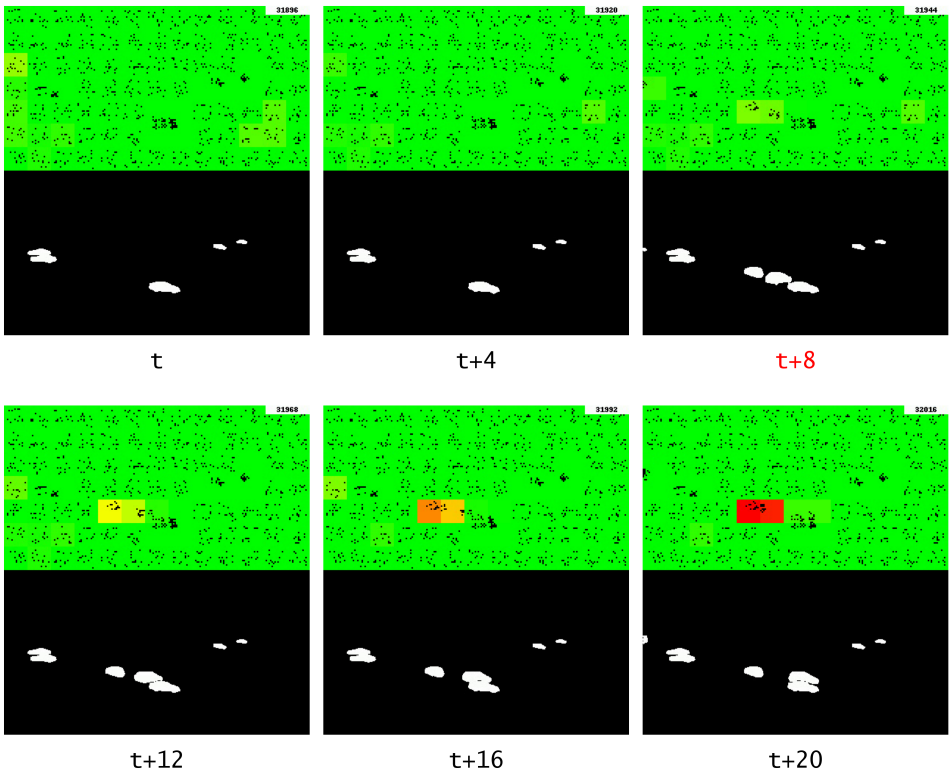


Figure 4.10: The first segment anomaly, which is marked with red text, and the corresponding changes detected by Grid HTM

4.2.2.1 Road

In the original video, there is a road on which cars regularly drive. By observing the visual output, it becomes evident that after some time the Grid HTM has mostly learned that behavior and does not report those moving cars as anomalies.

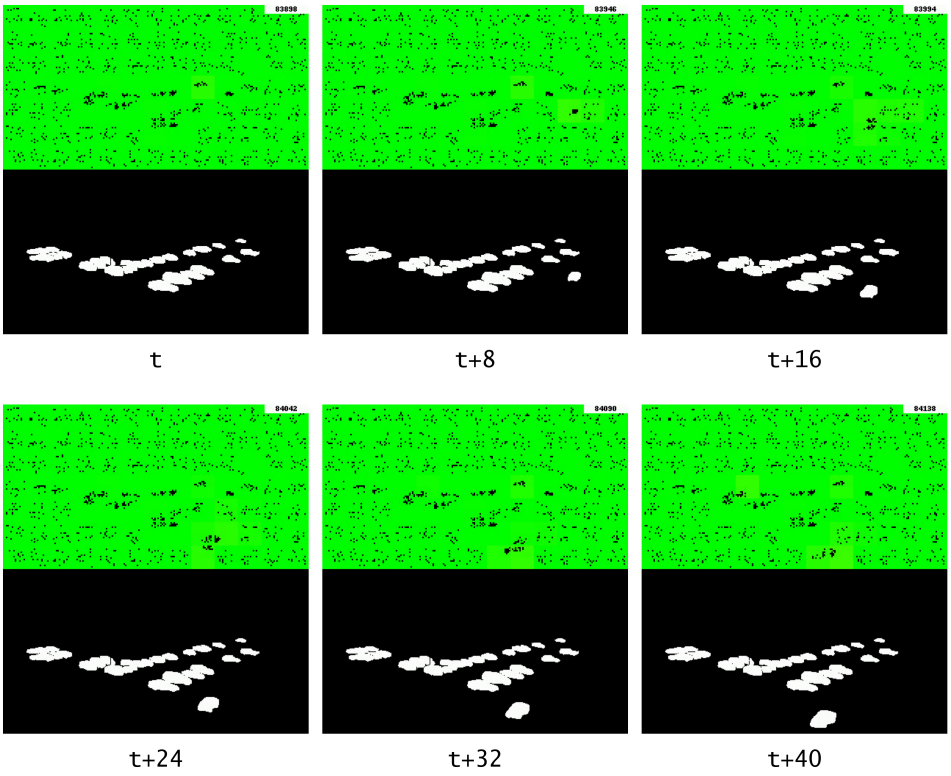


Figure 4.11: Road example

4.2.2.2 Frame Repeat

To prove that the Grid HTM has learned that cars on the road should be moving, it is possible to look at the visual output during the period when the video is repeating the same frame and observe if the system marks the cars standing still on the road as anomalies.

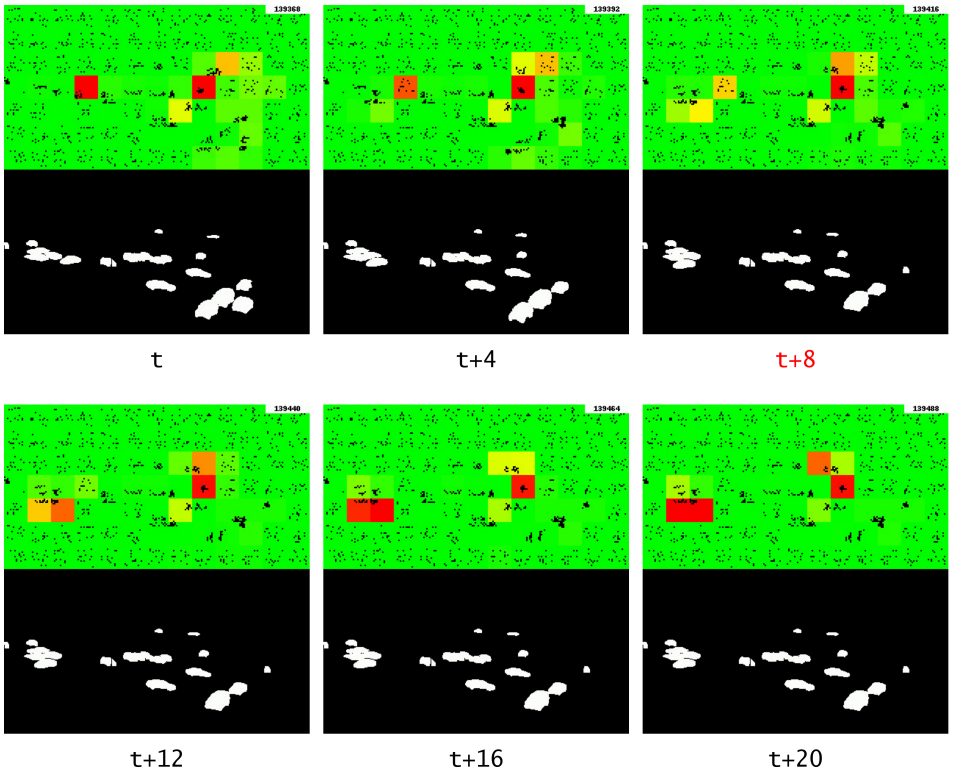


Figure 4.12: Anomaly output when the video is repeating a frame, the start of the frame repeat is marked with red text.

It can be observed that the cars along the main road are not marked as anomalies, but this could be attributed to the fact that there is a crossing there and that cars periodically have to stop at that point to let pedestrians cross.

On the other hand, when looking at the anomaly marked with a red circle, the car on the road in the parking lot is marked as an anomaly that increases in severity as the time goes on during the frame repeat. The reason why that car causes an anomaly is because, unlike the cars on the main road, a car is rarely observed as standing still at that position.

To prove that the anomaly was actually directly caused by the repeating frame, and not just due to repeating the anomaly in time, it should be compared to the anomaly output if there was no repeating frame.

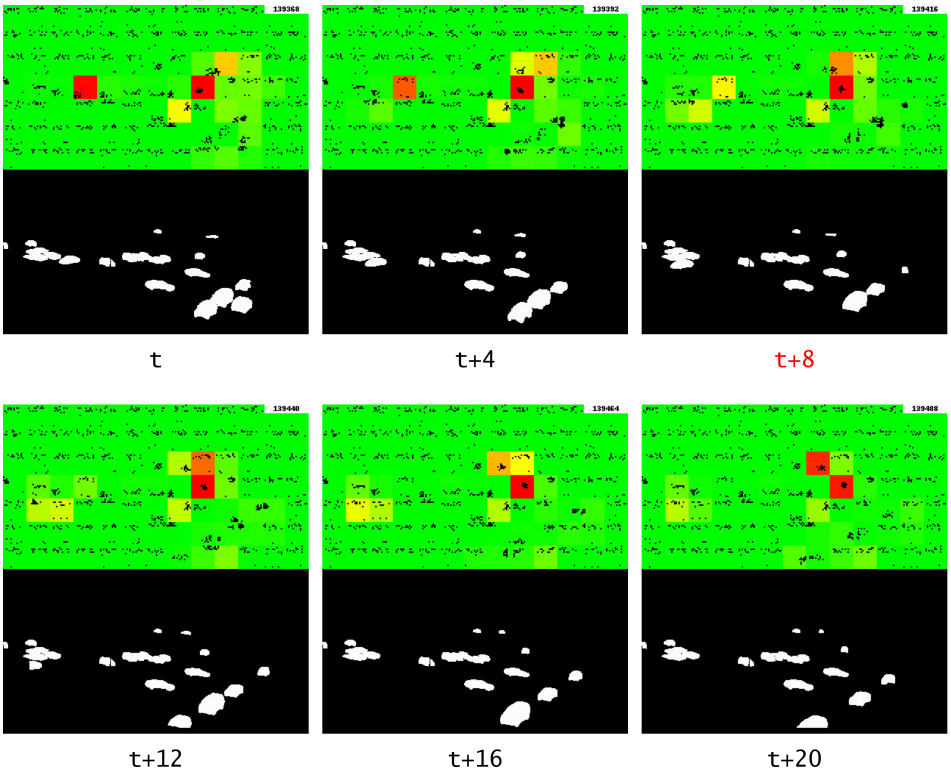


Figure 4.13: Anomaly output when the video is not frozen in time, when the freeze should have occured is marked with red text.

We can see that the anomaly output is minor compared to when there was a repeating frame, proving that the anomaly was indeed a product of the repeating frame and that the Grid HTM was able to learn how objects should be moving in time.

Finally, it is interesting to look at how the Grid HTM handles the repeating frames without multistep temporal patterns, which is shown in Figure 4.14.

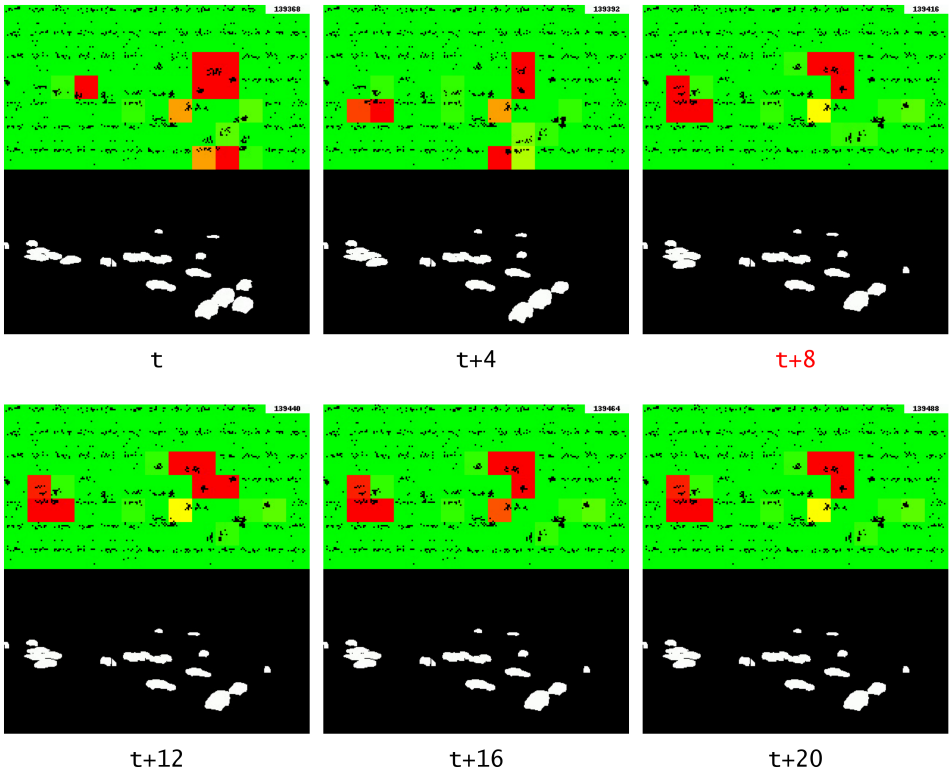


Figure 4.14: Anomaly output when the video is not frozen in time, when the freeze should have occured is marked with red text.

Unfortunately, simply disabling multistep temporal patterns without adjusting the other TM parameters causes the same car to be marked as an anomaly before and during the frame repeat. In fact, as mentioned in Figure 3.2.5, disabling multistep temporal patterns causes Grid HTM to be less noise tolerant which causes a lot more anomalies to be wrongly detected.

4.2.2.3 Points of Interest

Finally, it is interesting to look the various anomaly score spikes and observe in the visual output what caused them. The points of interest to be explored are marked in Figure 4.15

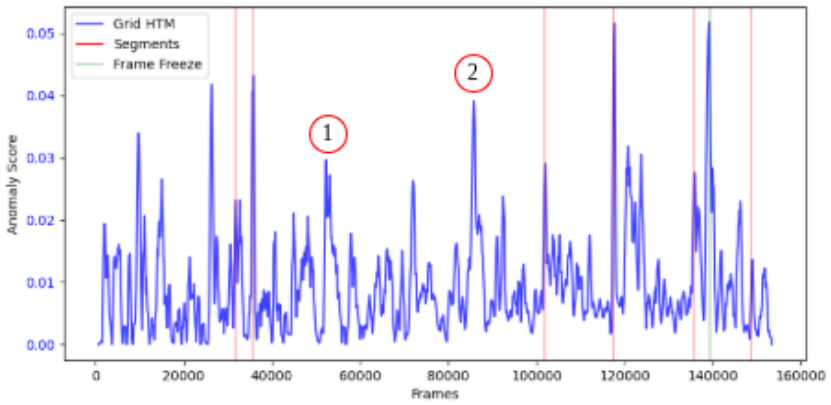


Figure 4.15: Points of interests.

The first point of interest, which can be seen in TODO, showcases the weakness of the aggregation function. single high anomaly Due to the nature of moving average, since this anomaly lasts for a long time, it

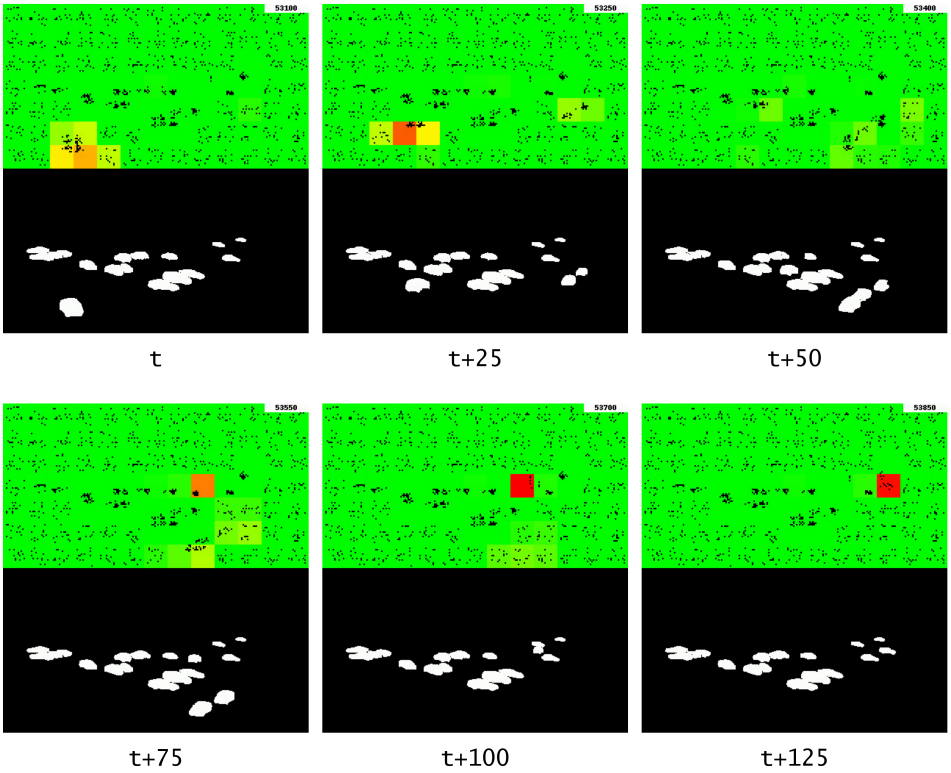


Figure 4.16: First

SECOND POI:

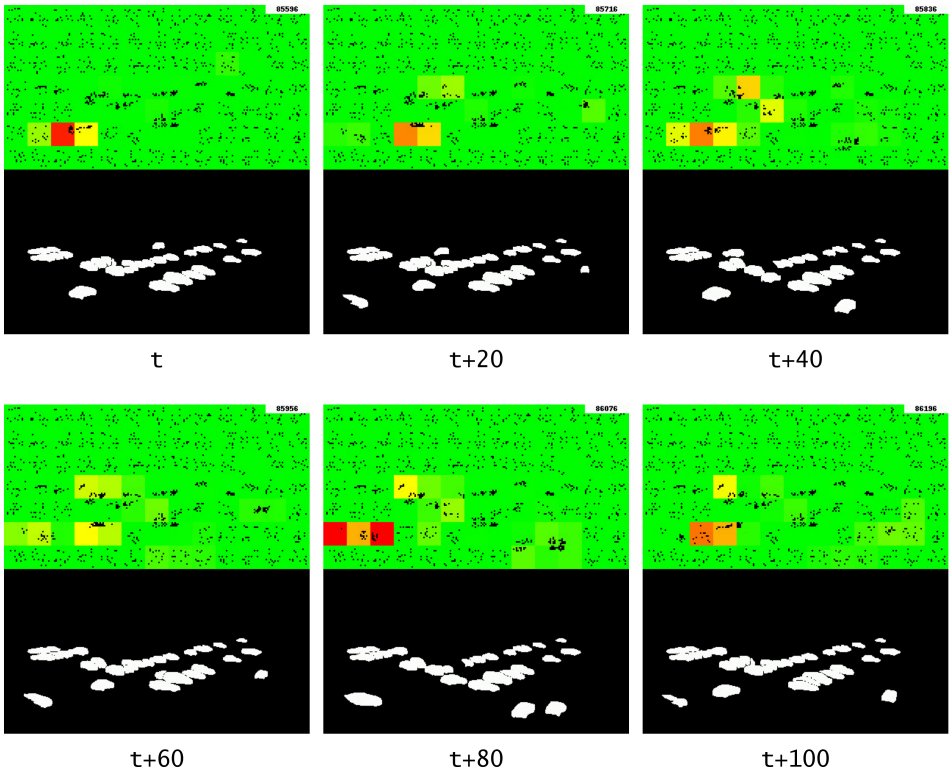


Figure 4.17: Second

4.3 Sperm example

As seen in the surveillance example, it seems the Grid HTM can detect when segments begin and end. This experiment will explore this ability in greater detail.

4.3.1 Data

The dataset used is VISEM [74], 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.

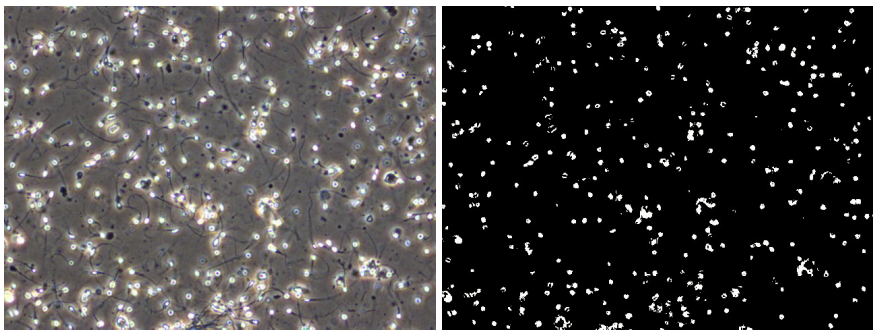


Figure 4.18: Example frame from a sperm video and its corresponding segmentation.

It is important to note that the data itself is noisy, and that it is not possible for the Grid HTM to learn any meaningful pattern. The individual videos are also relatively short, which makes it even harder to learn any meaningful patterns.

4.3.2 Benchmark

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. The L2 error could also have been used, but it would not matter since this experiment will be comparing relative values.

4.3.3 Parameters

For both results the following parameters were used:

Parameter	Value	Notes
sp_grid_size	16, 16	Size of each grid
tm_grid_size	8, 8	Size of each SP grid output.
min_sparsity	1	
sparsity	5	
temporal_size	1	Size of the multistep temporal pattern, 1 means it is effectively disabled

Table 4.6: Grid HTM specific parameters

Parameter	Value	Notes
inputDimensions	sp_grid_size	
columnDimensions	tm_grid_size	
potentialPct	0.2	
potentialRadius	5	
localAreaDensity	0.1	
globalInhibition	False	
wrapAround	False	
synPermActiveInc	0.01	
synPermInactiveDec	0.001	
stimulusThreshold	5	
boostStrength	0	Causes instability in empty cells

Table 4.7: SP Parameters

Parameter	Value	Notes
columnDimensions	tm_grid_size	Same as the SP
predictedSegmentDecrement	0.003	
permanenceIncrement	0.01	
permanenceDecrement	0.001	
minThreshold	1	
activationThreshold	3	
cellsPerColumn	16	

Table 4.8: TM Parameters

For the plots, a moving average with window size $n = 100$ was used to smooth the lines and reduce noise. Because the data is noisy, the mean was used as the aggregation function.

4.3.4 Results

As seen in Figure 4.19, the Grid HTM is able to outperform the L1 error benchmark. This can be deduced from the more prominent spikes in the anomaly score, compared to the L1 error. The reason might be that even though the data is very noisy, there is still something in it which makes Grid HTM able to learn something general about the current state. This could for instance be a single cell that is standing still or moving very slowly, which the Grid HTM anchors itself to and uses it to determine when segments start and end.

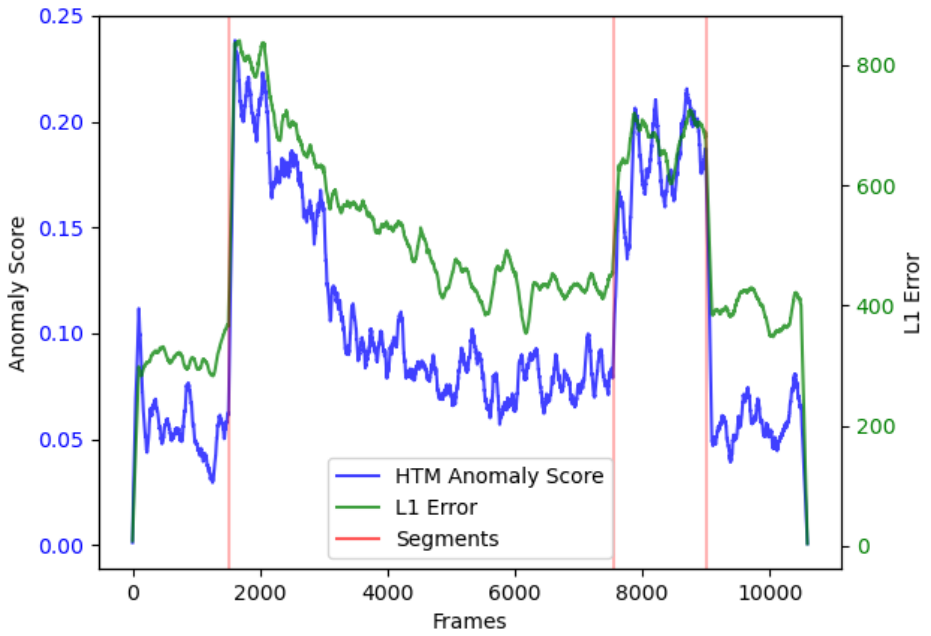


Figure 4.19: Results on a stationary sperm video.

That being said, the parameters for the Grid HTM were selected carefully to achieve the results seen in Figure 4.19, and are dependent on the contents of the data. Unfortunately, most of the videos in the dataset contain drift (in other words, the video is not stationary), which makes Grid HTM useless. This can be observed in Figure 4.20, where both the Grid HTM and the L1 error struggle.

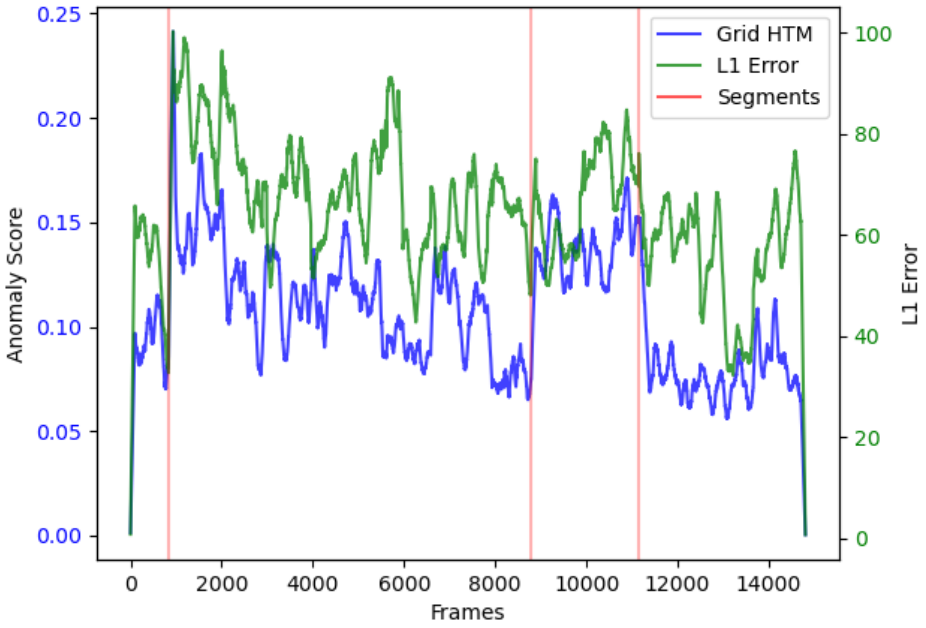


Figure 4.20: Results on a sperm video with drift.

Just like in the stationary video, the spikes in the anomaly score are more pronounced, but due to the drift in the video there is a constant high anomaly output which makes it impossible to find a threshold value.

4.4 Summary

Chapter 5

Conclusions & Future Work

5.1 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.

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] Jeff Hawkins, Subutai Ahmad, and Yuwei Cui. "A Theory of How Columns in the Neocortex Enable Learning the Structure of the World." In: *Frontiers in Neural Circuits* 11 (2017). ISSN: 1662-5110. DOI: 10.3389/fncir.2017.00081. URL: https://www.frontiersin.org/article/10.3389/fncir.2017.00081.

[5] 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.

[6] Yuwei Cui, Subutai Ahmad, and Jeff Hawkins. *The HTM Spatial Pooler – a neocortical algorithm for online sparse distributed coding*. Feb. 2017. DOI: 10.1101/085035.

[7] Jeff Hawkins and Subutai Ahmad. "Why Neurons Have Thousands of Synapses, A Theory of Sequence Memory in Neocortex." In: *Frontiers in Neural Circuits* 10 (Oct. 2015). DOI: 10.3389/fncir.2016.00023.

[8] 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.

[9] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65 6 (1958), pp. 386–408.

[10] Yoav Freund and Robert Schapire. "Large Margin Classification Using the Perceptron Algorithm." In: *Machine Learning* 37 (Feb. 1999). DOI: 10.1023/A:1007662407062.

[11] Marvin Minsky and Seymour Papert. "Perceptrons." In: (1969).

[12] Mikel Olazaran. "A Sociological Study of the Official History of the Perceptrons Controversy." In: *Social Studies of Science* 26.3 (1996), pp. 611–659. DOI: 10.1177/030631296026003005. eprint: https://doi.org/10.1177/030631296026003005. URL: https://doi.org/10.1177/030631296026003005.

[13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors." In: *nature* 323.6088 (1986), pp. 533–536.

[14] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. 2019. arXiv: 1803.08375 [cs.NE].

[15] Sebastian Ruder. "An overview of gradient descent optimization algorithms." In: *arXiv preprint arXiv:1609.04747* (2016).

[16] Kristy Carpenter et al. "Deep learning and virtual drug screening." In: *Future Medicinal Chemistry* 10 (Oct. 2018). DOI: 10.4155/fmc-2018-0314.

[17] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. "A training algorithm for optimal margin classifiers." In: *Proceedings of the fifth annual workshop on Computational learning theory*. 1992, pp. 144–152.

[18] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[19] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[20] Connor Shorten and Taghi M Khoshgoftaar. "A survey on image data augmentation for deep learning." In: *Journal of big data* 6.1 (2019), pp. 1–48.

[21] Dana H Ballard. "Modular learning in neural networks." In: *Aaai*. Vol. 647. 1987, pp. 279–284.

[22] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory." In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[23] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE].

[24] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[25] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

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

[27] 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.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems* 25 (2012).

[29] Rajat Raina, Anand Madhavan, and Andrew Y Ng. "Large-scale deep unsupervised learning using graphics processors." In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 873–880.

[30] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library." In: *Advances in neural information processing systems* 32 (2019).

[31] Martin Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." In: *arXiv preprint arXiv:1603.04467* (2016).

[32] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[33] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[34] Shivani Gupta and Atul Gupta. "Dealing with Noise Problem in Machine Learning Data-sets: A Systematic Review." In: *Procedia Computer Science* 161 (2019). The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia, pp. 466–474. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.11.146. URL: https://www.sciencedirect.com/science/article/pii/S1877050919318575.

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

[36] Chen Sun et al. "Revisiting unreasonable effectiveness of data in deep learning era." In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 843–852.

[37] Alexander D'Amour et al. *Underspecification Presents Challenges for Credibility in Modern Machine Learning*. Nov. 2020.

[38] 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.

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

[40] 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.

[41] 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.

[42] 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.

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

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

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

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

[47] 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.

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

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

[50] Jeff Hawkins, Subutai Ahmad, and Yuwei Cui. "Why does the neocortex have layers and columns, a theory of learning the 3d structure of the world." In: *bioRxiv* (2017), p. 162263.

[51] MRaptor. June 2016. URL: https://discourse.numenta.org/t/htm-cheatsheet/828.

[52] 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.

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

[54] 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.

[55] 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.

[56] 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.

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

[58] Oleg Iegorov. *My analysis on why Temporal Memory prediction doesn't work on sequential data*. Dec. 2017. URL: https://discourse.numenta.org/t/my-analysis-on-why-temporal-memory-prediction-doesnt-work-on-sequential-data/3141.

[59] 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.

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

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

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

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

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

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

[66] 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.

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

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

[69] 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.

[70] 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.

[71] 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.

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

[73] Ayoola Olafenwa. “Simplifying Object Segmentation with PixelLib Library.” In: 2021.

[74] 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.