Hierarchical Temporal Networks for anomaly detection in surveillance

A subtitle of your thesis

Vladimir Monakhov

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

Department of Informatics
The Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2022

Hierarchical Temporal Networks for anomaly detection in surveillance

A subtitle of your thesis

Vladimir Monakhov

Hierarchical Temporal Networks for anomaly detection in surveillance

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

Printed: Reprosentralen, University of Oslo

Abstract

Bla Bla Bla ...

Contents

Chapter 1

Introduction

As the global need for security and automation increases, many entities seek to use video anomaly detection systems. Their uses can vary from detecting faulty products on an assembly line or detecting car incidents on the highway, and everything in between. Leveraging modern computer vision, modern anomaly detection systems play an important role in increasing monitoring efficiency and reducing the need for expensive live monitoring.

Despite the major progress within the field of Machine Learning, there are still many tasks where humans outperform algorithms. The reason for this probably lies in the difference between how humans and machine learning algorithms represent data and learn. Most machine learning algorithms use a dense representation of data and apply back-propagation in order to learn. Human learning happens in the neocortex, where the most popular theory is 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 thesis proposes an architecture which combines both deep learning image segmentation and HTM in order to perform anomaly detection on surveillance videos.

Chapter 2

Background

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

2.1 Hierarchical Temporal Memory

Today's machine learning algorithms aim to solve complex problems by simulating a substantial amount of mathematically defined neurons. These neurons are vastly simplified compared to the neurons in the brain and therefore do not have the complexity required to solve complex problems with an accuracy and level of generalizability comparable to the brain. [1] 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.

2.1.1 Structure

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

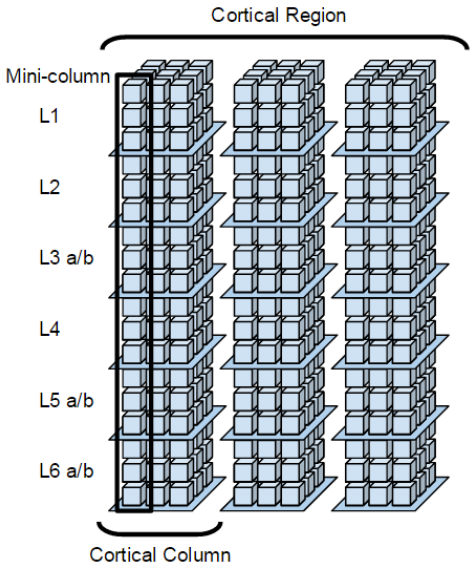


Figure 2.1: Visualization of the HTM Cortical Region structure [2]. Note that current HTM implementations only model layer 2/3.

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

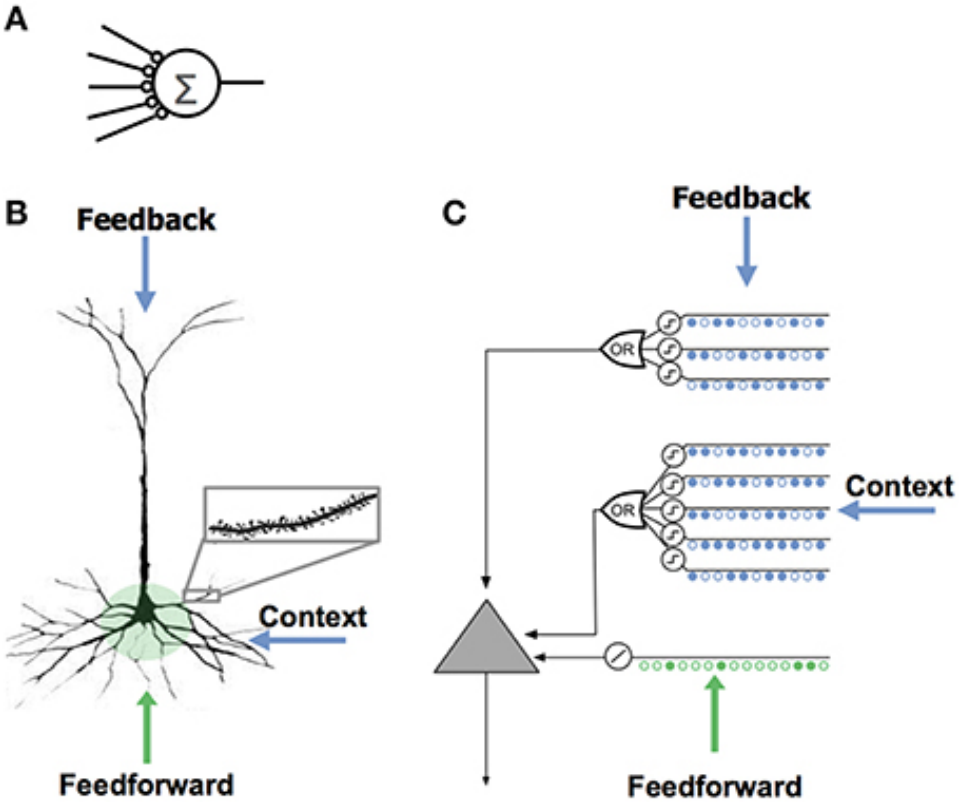


Figure 2.2: Comparison of neurons [3]: A) Traditional Machine Learning, B) Biological, C) HTM

The HTM neuron has three inputs [3]:

- Feedforward, which is the input data
- Context, which is data from neighbouring 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

2.1.2 Common Algorithm

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

2.1.3 Sparse Distributed Representation

HTM Theory introduces Sparse Distributed Representation (SDR) as a way of representing data in HTM and can be thought of as a bit-array. Each bit theoretically corresponds to a neuron in the neocortex and also represents some semantic information about the current data. This opens up for all

kinds of mathematical operations, for instance it is possible to compare the semantic similarities between two SDRs by simply performing a binary AND operation.

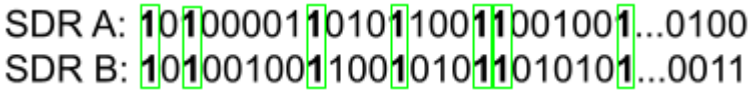


Figure 2.3: Semantic similarities between the two SDRs A and B

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

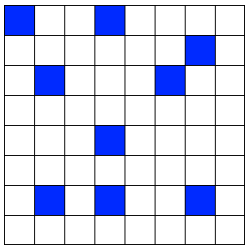


Figure 2.4: Example representation of an SDR with a length of 64 and a sparsity of 14.1%, visualized as a 2D grid

2.1.4 Encoders

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

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

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

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

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

Several approaches for encoding visual data have been proposed, such as [4] which uses a neuroscientifical approach by replicating how the eye works, and [5] which uses scale-invariant feature transform (SIFT) to find points of interest in images and encode that information as an SDR. There are also deep learning approaches such as [6] which uses a Convolutional Neural Network (CNN) as part of the encoder, specifically they use the top n-features in a feature map as ones and set the rest to zeros in order to construct their SDRs.

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

An encoder which transforms convolutional feature maps into SDRs could help solve this, but the issue is converting the dense representation of the feature maps into SDRs. Directly encoding them into SDRs by treating each value in the feature map as a float and converting it into its own SDR quickly becomes intangible due to the processing and memory requirements. Additionally, minor variations in the feature map would case major variations in the resulting SDR. Alternatively one could follow [6] 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 unclear in models trained for complex tasks.

2.1.5 Learning

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

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

The spatial pooler takes in 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.

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

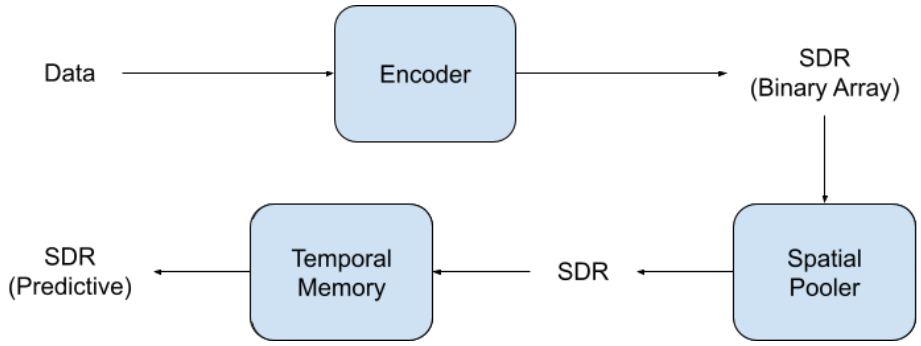


Figure 2.5: The HTM Pipeline. A common next-step could be to use a classifier to convert the predictive SDR into a classification.

This gives HTM systems the property of on-line learning, meaning they learn as they go. There is no batch training because each input into the HTM system will update the system. The system effectively builds a predictive model of the data and learns by trying to minimize the error between the true values and the predicted values. This means that the system will continuously adapt to a changing environment. A spatial pooler followed by a temporal memory creates the HTM neuron described in 2.2, where the color green indicates the responsibility of the Spatial Pooler and blue indicates the responsibility of the Temporal Memory.

Spatial Pooler

The spatial pooler consists of 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 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.

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 or not 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).

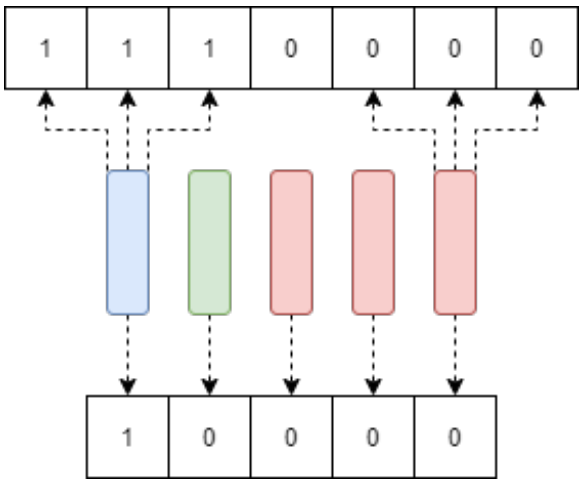


Figure 2.6: This figure illustrates how a spatial pooler works. All connections are above the stimulus threshold. The receptive field is 3 bits wide for each column. Overlap score threshold is 1, and $n = 1$. Red means inactive, green means active, and blue means active and selected.

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 overperforming 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.

It is also possible to **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 of the aforementioned concepts are configurable in technical implemen-

tations.

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 consists of cells and can contain an arbitrary **number of cells** which defines the capacity of sequences and contexts that the temporal memory can express. Each cell in a column can connect to other cells in other columns using segments (more specifically, distal dendrite segments), where each segment consists of synapses connecting to other cells.

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

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

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

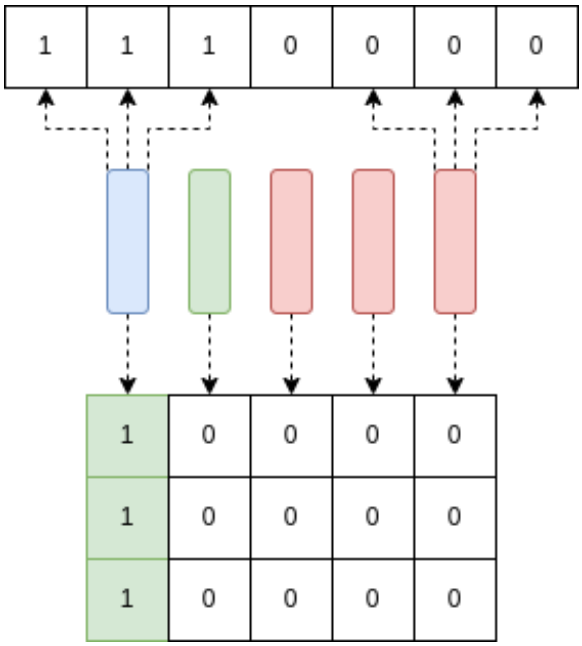


Figure 2.7: Expanded SP example with TM component where the number of cells is set to 3. The leftmost column is bursting (all 3 cells activated in green) due to the active SP output and due to containing no predictive cells.

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

- Active, if the column is bursting or the cell was in a predictive state in the previous timestep.
- 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.

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 timestep, then the number of cells can be set to 2.

Finally, the TM compares the predictions P it made in the previous timestep with the actual pattern A in the current timestep 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 [7]. 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 because in reality the two patterns may have overlapping bits in their representations.

2.1.6 History

The HTM theory described in this thesis is not the first one to have been made, this one is actually the third generation which builds upon the second generation. The first generation had fundamentally different inner workings [8], but shared a lot of the terms with the current generation. This has made researching HTM challenging as there are still many research papers published that refer to the first generation.

2.1.7 Use cases

The general use case for HTM is to perform anomaly detection. More specifically, Numenta has made example programs showcasing how HTM can be used in practice [9]:

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

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 [10].

2.2 Convolutional networks

A key element to understand is why one would want to use convolutional networks as a part of the encoder for an HTM network. The main reason is that a CNN is able to extract important spatial features from high dimensional data, effectively lowering the amount of dimensions. A CNN can also apply several properties that are useful, such as translational and distortional invariances [11]. These properties stem from the three architectural ideas in a CNN:

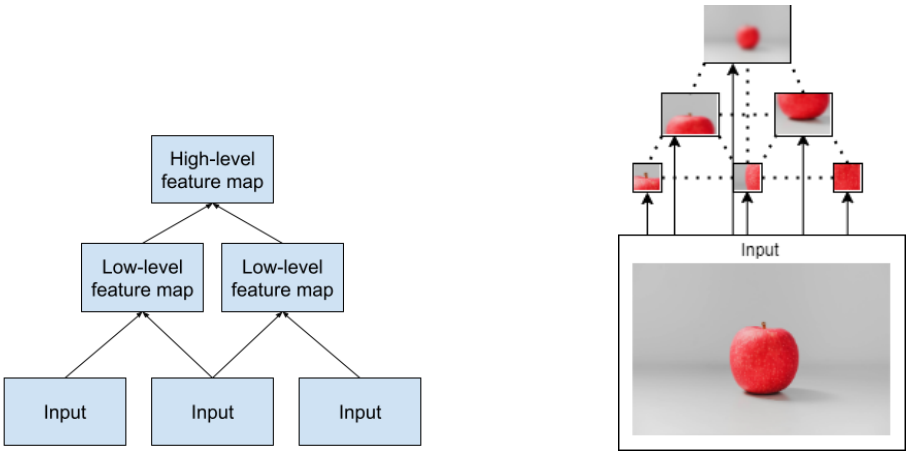
- Local receptive fields
- Shared weights
- Spatial sub-sampling

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 invariance.

2.3 The Thousand Brains Theory

One of the newest advancements in HTM Theory is the introduction of the Thousand Brains Theory. [12] 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 rely on different inputs. 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 2.8: Comparison of classical hierarchy and the hierarchy introduced by the Thousand Brains Theory

This new type of hierarchy is coincidentally similar to some state-of-the-art image recognition deep learning architectures such as [13] and [14], 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.

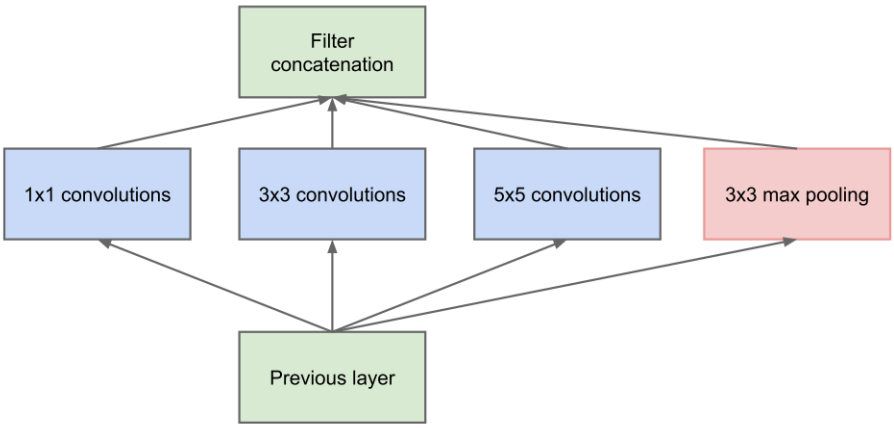


Figure 2.9: How the Inception [14] architecture combines multiple filters

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

2.4 Anomaly detection

As reviewed by Pang et al.[15], 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 of the complexities are as follows:

- **Unknowness** Anomalies are associated with many unknowns which do not become known until the anomaly happens.
- **Rarity and class imbalance** Anomalies are by definition rare instances, which means that it becomes difficult to create a balanced dataset.
- **Heterogenosity** Anomalies can take form in many different ways, and as such one class of anomalies can be vastly different from another.

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

2.4.1 State of the Art Algorithms

The current state of the art algorithms are numerous, but the main approach is achieved by using deep learning. 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 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 sub-optimal results given that it tries to generate good synthetic data rather than directly detect anomalies.

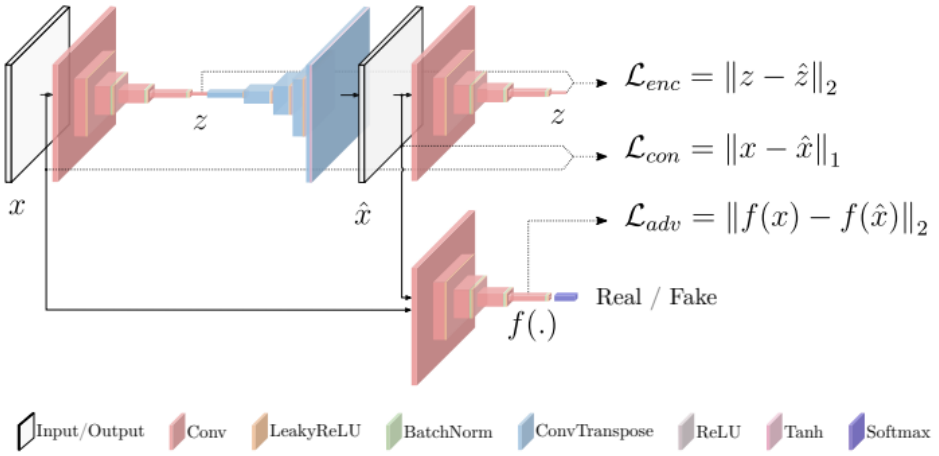


Figure 2.10: GANomaly [16], a variation of GAN for Anomaly detection

Another common approach is autoencoders, which aim to minimize the reconstruction error from a learned feature representation space. 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. The main disadvantage is that the learned feature representation can be biased by outliers and extremities in the data.

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

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

2.4.2 Smart Surveillance

Smart surveillance, which is the use of automatic video analysis in surveillance, has seen rapid development since its inception. [17] presents and summarizes recent progress for anomaly detection in video for surveillance purposes, where the most promising methods are achieved by using convolutional auto-encoders (AE) and General Adversary Networks (GAN). The results show that the deep learning approaches have a high degree of accuracy, and are consistently improving.

The paper also discusses problems with using deep learning approaches for anomaly detection. One example is that in most instances a bicycle on a campus is not an anomaly, but if it is rare enough then it may be wrongly classified as so.

2.4.3 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.

[18] 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.

For high-dimensional anomaly detection, [19] used a HTM system to find anomalous frames in videos of motions. The anomalies were artificially created by swapping certain frames between different motion videos in the dataset. The results show that the HTM system was able to correctly detect some of the anomalies, but not an impressive amount. One thing to note is that direct binary representations of the video frames were used as SDRs, therefore no proper encoding was performed which might have led to the poor results. This hints at the fact that HTM by itself is not capable of handling high dimensional data, and is instead reliant on an encoder to lower the dimensionality by extracting important spatial features.

2.4.4 Deep Learning HTM Encoder

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

2.5 Summary

Chapter 3

Methodology

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. The video to be used is part of the VIRAT dataset.

Figure 3.1: Segmentation result of cars

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 considered equal while trucks and motorcycles will be different. 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.

One issue that becomes evident is the lack of invariance. Because the TM is learning the global patterns, 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.

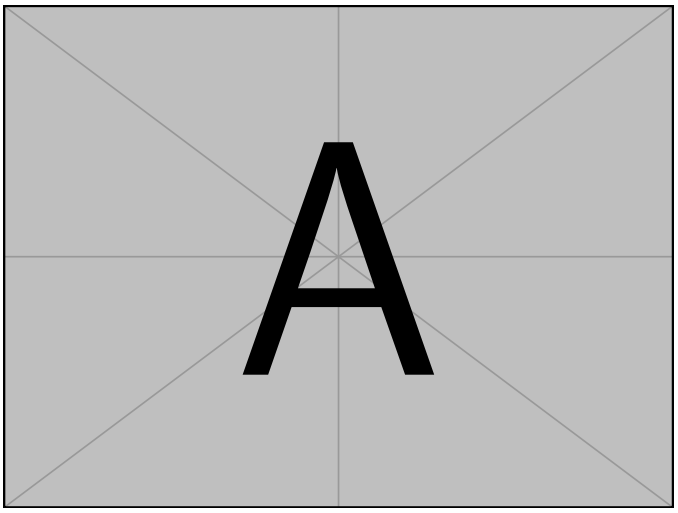


Figure 3.2: The encoder output divided into a grid.

Another problem is that the previously mentioned rules for creating a good encoder may not be respected, and therefore should be reviewed:

- **Semantically similar data should result in SDRs with overlapping active bits**. In this case, a car at the 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 produces a deterministic output given the same input.
- **The output must have the same dimensionality (total number of bits) for all inputs**. The segmentation 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:

- Pick a cell size so that the expected amount of active pixels when a cell contains car(s) is roughly the same.
- 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.

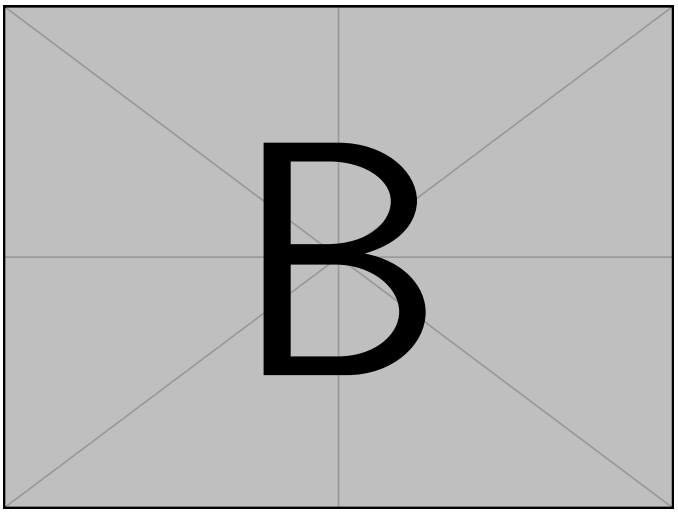


Figure 3.3: Encoder output.

Chapter 4

Results

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 Bouncing Ball Test

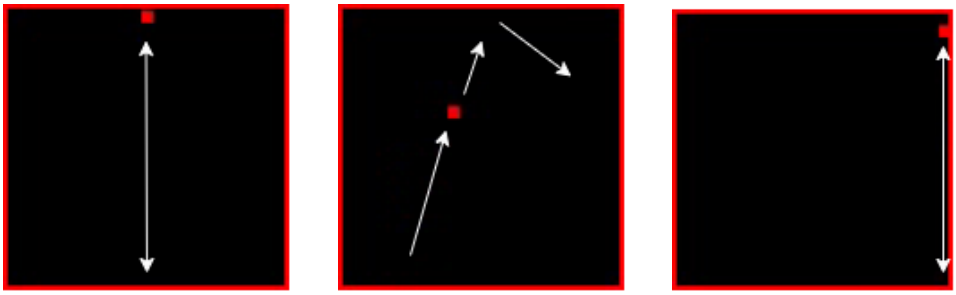


Figure 4.1: The bouncing ball test, and its three stages

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 goes back to 0 and the ball goes back to bouncing up and down in-place.

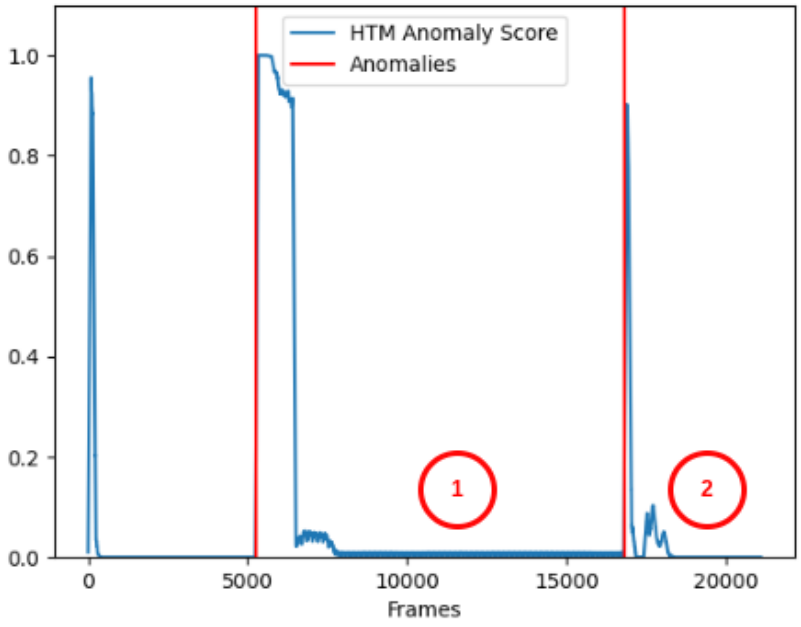


Figure 4.2: The moving average of the anomaly score in the bouncing ball experiment.

From the figure, it can be observed that it correctly detects anomalies and quickly adapts to them. On the other hand, the result is not perfect due to

the minor oscillations close to mark 1 and the anomaly spikes towards the end close to mark 2.

The reason for the oscillations is due to the spatial pooler being dominated by a lucky few columns. The solution is to enable boosting.

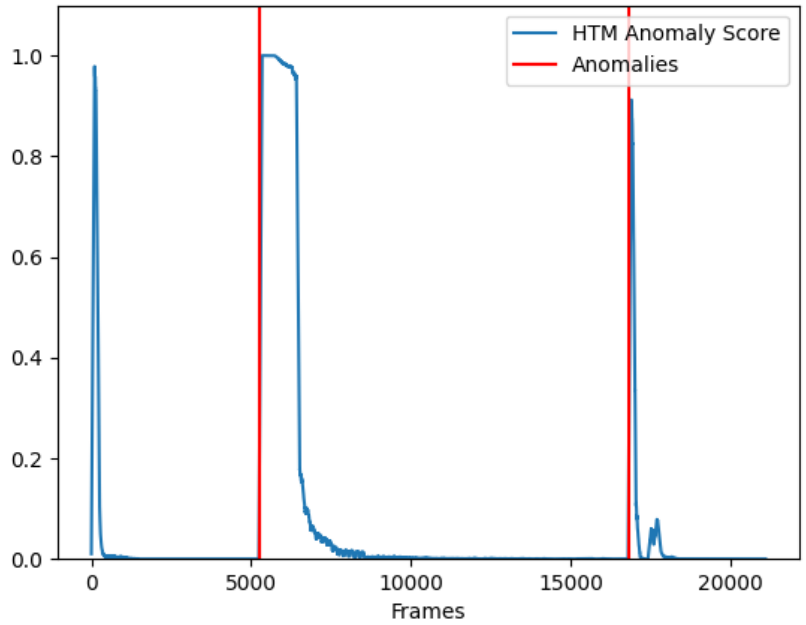


Figure 4.3: Bouncing ball with boosting enabled.

The reason for the anomaly spikes towards the end is because the spatial pooler had found an optimal representation when the ball was bouncing freely, but when the ball stops and starts bouncing in-place the spatial pooler ends up unlearning the old optimal representation while it learns the new optimal representation. This causes a sudden minor change in the SP output, which the TM reports as anomalous. The solution is to set the value by which permanence is decreased by to zero, effectively disabling the ability of the spatial pooler to "forget".

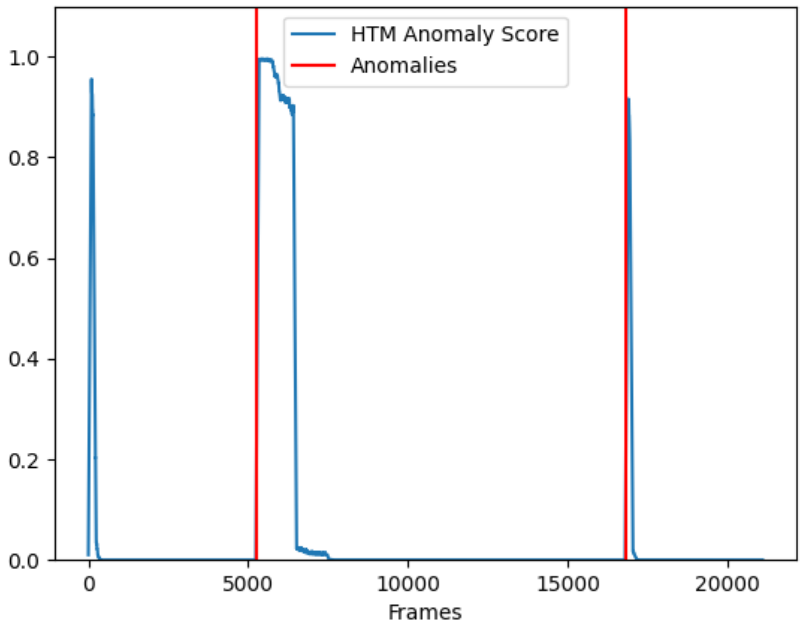


Figure 4.4: Bouncing ball without the ability of the SP to "forget".

Decrementing permanence is important in HTM systems, so disabling it is not always feasible.

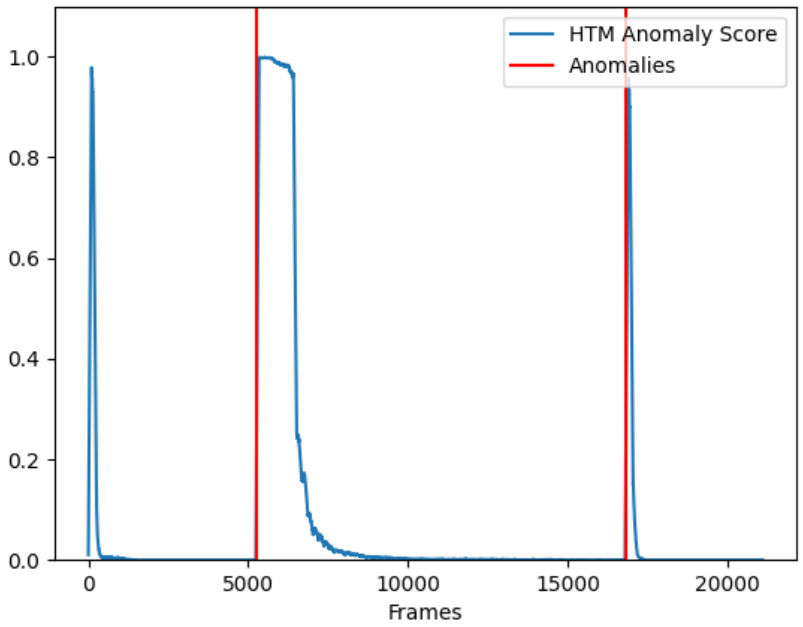


Figure 4.5: Bouncing ball without the ability of the SP to "forget" and with boosting enabled.

Final list of parameters for reproducibility:

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

Table 4.1: SP Parameters

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

Table 4.2: TM Parameters

4.2 Surveillance example

Chapter 5

Summary & Conclusions

Bibliography

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

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

[3] 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.

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

[5] 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.

[6] Y. Zou, Y. Shi, Y. Wang, 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.

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

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

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

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

[11] Y. Lecun, L. Bottou, Y. Bengio, 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.

[12] Jeff Hawkins, Marcus Lewis, Mirko Klukas, 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.

[13] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, et al. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV].

[14] Christian Szegedy, Wei Liu, Yangqing Jia, et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[15] Guansong Pang, Chunhua Shen, Longbing Cao, 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.

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

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

[18] Subutai Ahmad, Alexander Lavin, Scott Purdy, 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.

[19] 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.