UiA University of Agder

Master

Analysis of binarization techniques and Tsetlin machine architectures targeting image classification

ERIK MATHISEN
HALVOR S. SMØRVIK

SUPERVISORS
Ole-Christoffer Granmo, Jivitesh Sharma

University of Agder, 2020
Faculty of Engineering and Science
Department of Information and Communication Technology

Abstract

The Tsetlin Machine is a constantly evolving and developing machine learning technique with ever-increasing success. However, for every success, the Tsetlin Machine achieves, a new set of challenges are put ahead. To sufficiently bring the Tsetlin Machine to a broadly used standard, these challenges must be completed. This thesis focuses on the challenge of doing color image classification and will provide an introductory description of how this is possible through the usage of an older technique, namely binarization. A comparison with the various Tsetlin Machine adaptations made public in recent times is also present after the achieved color image classification. The results of both the initial color image classification experiment and the comparison between the varying adaptations show that the Tsetlin Machine, with a little extra work, can achieve high accuracy color image classification without image augmentation or pre-training.

Preface

This thesis summarizes the culmination education received at the University of Agder, Grimstad. The thesis encompasses work done over the last semester. The task was proposed by Prof. Ole-Christoffer Granmo in early January 2020. Both the authors of this thesis have a great interest in the Tsetlin Machine and its potential capabilities. Therefore when Granmo proposed working on a proof of concept towards color image classification along with a likely comparison of the various architectures, which had yet to be done, we were highly interested.

Throughout this thesis, we have received a great deal of motivation and inspiration under the supervision of Prof. Ole-Christoffer Granmo, and PhD Candidate Jivitesh Sharma and would like to extend our gratitude towards them both.

This thesis was written during a tumultuous time during Spring 2020, in which the Covid-19 pandemic forced substantial workflow changes. To this end, it would be unjust not to extend our gratitude towards Sr Engr. Sigurd Brinch and Tech. Bendik Egenes Dyrli for providing the invaluable infrastructure on which we performed most of the experimentation necessary for this thesis.

This project was part of the single semester IKT590 course which culminates in the writing of a master's thesis. Over the course of a few months, we have received a great deal of understanding of the Tsetlin Machine and the fundamental mechanics, and hope through this thesis that others can as well.

List of Figures

List of Tables

Contents

Part I

Introduction

Chapter 1

Introduction

1.1 Motivation

Tsetlin Machine is an up and coming machine learning technique published by Prof. Ole Christoffer Granmo in 2018, in the midst of what can only be described as an AI summer. While still considered new and state-of-the-art, this Machine Learning Technique has resulted in multiple revisions and adaptions, which in turn only strengthens its overall capabilities[1]. With stark contrast in the increasingly complex black-box nature of neural networks, the Tsetlin Machines shows promise in achieving a scaleable transparent classification algorithm. Especially in a time where the focus is shifting towards interpretable machine learning models[2].

While the Tsetlin Machine has proven itself capable of handling large datasets and severe classification problems[3]. It is still working its way up the various challenges required to acquire a fuller acceptance. While achieving high accuracy on both the MNIST and Fashion MNIST datasets[1] it has yet to prove capable of high accuracy classification on color images.

This is, in essence, the motivation behind this thesis. By utilizing older image binarization techniques, this thesis examines the possibility of color image classification with relatively low amounts of data pre-processing. Should this be achievable, the various Tsetlin Machine Architectures would then require to be compared.

1.2 Thesis Definition

Through this section, we will go through the research question that compelled the completion of this thesis. Following the research question, we will explain a few goals and hypotheses close-knit to the question as mentioned earlier.

1.2.1 Research Question

The main focus of this thesis will be examine to what extent can the Tsetlin Machine classify color images and which state-of-the-art Tsetlin Machine architecture provides the highest accuracy.

1.2.2 Hypotheses

- Baseline Tsetlin Machine can provide color image classification without image augmentation or pre-training.
- Similar to how adaptations to the Neural Network model can provide higher accuracy for classification, so can the adaptations of the Tsetlin Machine.

The initial hypothesis states that the Tsetlin Machine can, in fact, be used to classify color images.
The latter hypothesis states that while the standard Tsetlin Machine can provide color image classification, a more recent adaption of the Tsetlin Machine, might provide higher accuracy than the standardized version.

1.2.3 Thesis Goals

With the Research Question and hypotheses in mind, the following are the goals thought necessary to pursue in this thesis.

- **Goal 1:** Provide a fundamental understanding of how the binarization of images can contribute to increased classification accuracy
- **Goal 2:** Provide a working understanding of the Tsetlin Machine Algorithm and the various adaptations
- **Goal 3:** Provide an environment in which the Tsetlin Machine can classify color images
- **Goal 3:** Implement the various adaptations into the environment for further testing
- **Goal 4:** Run multiple adaptions of the Tsetlin Machine with the intent of comparing empirical results.

1.2.4 Scope & Limitations

To date, multiple adaptions, variations, or revisions have been made to the Tsetlin Machine. While still a heavily developed machine learning algorithm with multiple published papers within the last few months, this thesis will focus on mostly adaptations published last year[1][4][5] with the singular exception of the layered Tsetlin Machine. Which has at the time of writing yet to be published.

1.3 Contributions

The Tsetlin Machine is an interpretable machine learning technique with which capabilities are continually growing. One step which has yet to be achieved is the classification of color images.

This thesis introduces a new environment that enables color image classification for the Tsetlin Machine and its various adaptations; Weighted, Convolutional, and Layered. This environment enables multiple types of datasets to be utilized through simple data processing methods, some of which have been available since the fax machine's days[6].

1.4 Thesis Outline

The thesis is divided into the following chapter with the topics as follows:

- **Chapter 2: Background** explain the fundamental mechanics and theories related to this thesis.
- **Chapter 3: Tsetlin Machine** Introduce and examine the state-of-art machine learning algorithm that is the Tsetlin Machine and the adaptions that is utilized through this thesis
- **Chapter 4: Environment** The proposed architecture of the environment in which the Tsetlin Machine can achieve classification of color images.
- **Chapter 5: Experiments** The results needed to test the color image capabilities of the Tsetlin Machine and its various adaptations
- **Chapter 6: Conclusion** Conclusion and discussion of the results and solution to this thesis. Will also contain potential future work based on this thesis

Part II

Background

Chapter 2

Background

With the ever-expanding field of machine learning, there are a vast plethora of different algorithms or techniques, most of which are tested against the same problems as a threshold to be regarded as an acceptable example of machine learning. One of the most common types of techniques is the neural network and its ever more complexing black box of nodes. In 2018 Prof. Ole-Christoffer Granmo introduced the Tsetlin Machine as a fundamentally more straightforward machine learning technique at its core and scaled to the size of a deep neural network. This follows a trend in which there is a strong desire to understand the reasoning behind machine learning outcomes.[1]
In this chapter, we examine the theory behind the fundamentals of both the Tsetlin Machine and image processing techniques used as data pre-processing before it is utilized for the final experimentation.

2.1 Image Processing

While the Tsetlin machine is both simple and robust, it can at this time only accept binary strings. Therefore, to help with image classification, we utilize a few different binarization methods to provide the required binary input structure. The act of binarization is done with a threshold value or algorithm, either grayscale or colorized is then put through such that the resulting image will yield a binarized version of either black or white. Previously over the MNIST dataset testing, Ole-Christoffer Granmo utilized a simple static value threshold for every image and achieved up to 99.33% accuracy using the Convolutional Ttsetlin Machine.[4] However, through this thesis, multiple other variations of binarizations will be utilized on the CIFAR10 dataset, which is a 3x32x32 RGB image dataset compared to the MNIST 28x28 grayscale.
Binarization is not something new; many different variations of binarization has been created for older technology such as image transfer over fax machine[6]. However, since other image classification techniques such as Neural Networks, Convolutional NN, Naive Bayes function without the use of binarization, this form of image pre-processing has been more or less forgotten. Through this thesis, we will explore a few of these older algorithms for binarizations and test them over using the CIFAR10 colorized image set.

2.2 Color Scheme

This section's primary focus is on different image processing methods utilized in this thesis. The section starts with a quick explanation of fundamental technology, then moves onto differing color schemes that was used throughout this thesis.

2.2.1 Grayscale Image

A grayscale image is a collection of variables sorted in a two-dimensional matrix. Each variable represents a pixel and is an integer in the range from 0 (black) to 255(white).

2.2.2 Histogram

The histogram of a grayscale image shows distribution of the values of each pixel spread out. Some images have very distinct peaks, while others are more evenly distributed between the values. It is constructed by mapping each of the pixels in the grayscale image to a histogram. The x-direction represents the value of the pixel from 0 to 255. The y-axis represents how many pixels are present of that particular value.

2.2.3 RGB

A color image coded in RGB is primarily made up of three grayscale images, called channels, each stacked on top of each other. Each channel represents one of the colors in the RGB color model. Another way of looking at RGB coded images is to look at single pixels. They have 3 values: Red, Green, and Blue. Each has a range from 0 to 255 (a byte). A single pixel can, therefore, have 255*255*255 = 16581375 different colors shown. In a geometric representation, these colors form a cube with red in one direction, blue in another and green in the last, and any color can be picked in the cube's space. As an example, a picture coded in RGB of a red cat would have overall increased values in the red channel, as there would be higher occurrence of red compared to the two other colors.

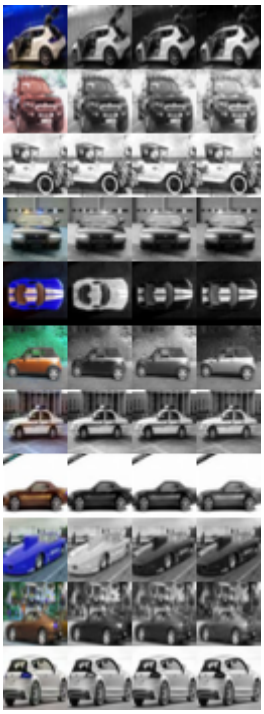


Figure 2.1: First image column shows 11 color images from the CIFAR 10 dataset with the car class. The three columns next to them, are the B, G and R channels respectively in grayscale form.

2.2.4 HSL

HSL is another encoding of the color scheme in an image. Here there are 3 variables: Hue(H), Saturation(L) and Lightness(L). Hue is measured as an angle and represents the primary and secondary colors. Lightness is how much white or black that are mixed in the color; Saturation is how gray the color is. Changing the RGB image to HSL can be done through the following equation, which is the method of the OpenCV library[7], which is utilized in Chapter 4.

$$
\begin{aligned}
Vmax &\leftarrow \quad max(R,G,B) \\
Vmin &\leftarrow \quad min(R,G,B) \\
L &\leftarrow \{ \quad \tfrac{Vmax+Vmin}{2} \\
S &\leftarrow \begin{cases} \frac{Vmax-Vmin}{Vmax+Vmin} & if L < 0.5 \\ \frac{Vmax-Vmin}{2-(Vmax+Vmin} & if L \geq 0.5 \end{cases} \\
H &\leftarrow \begin{cases} 60(G-B)/S & if Vmax = R \\ 120+60(B-R)/S & if Vmax = G \\ 240+60(R-G)/S & if Vmax = B \end{cases}
\end{aligned}
\tag{2.1}
$$

2.2.5 HSV

HSV is very similar to HSL, but have two key differences. It uses "Value" instead of light, which decides how much black should be blended in the color, and the formula for calculating Saturation is different. This change in color scheme from RGB to HSV is done by equation 2.2. The process of change is also available through the OpenCV library[7].

$$
\begin{aligned}
V &\leftarrow \{ \quad max(R,G,B) \\
S &\leftarrow \begin{cases} \frac{V-min(R,G,B)}{V} & ifV \neq 0 \\ 0 & otherwise \end{cases} \\
H &\leftarrow \begin{cases} 60(G-B)/V - min(R,G,B)) & ifV = R \\ 120 + 60(B-R)/(V - min(R,G,B)) & ifV = G \\ 240 + 60(R-G)/(V - min(R,G,B)) & ifV = B \end{cases}
\end{aligned}
\tag{2.2}
$$

2.2.6 Image convolution

Convolution is a mathematical operator for two functions. With one dimentional countinuining functions, it calculates the area ($c(x)$) that two functions($a(x)$ and $b(x)$) shares, when one of them is reversed and sent trough the other and is written as $c(x) = (a*b)(x)$. The easiest way to calculate this property is by taking the fourier transform of both functions, multiply them, and inverse fourier transform them (eg $c(x) = (a * b)(x) = \mathcal{F}^{-}1(\mathcal{F}(a(x))\mathcal{F}(b(x)))$. Convolution with images and kernels use the same concept but the math is easier, as it boils down to multiplication and addition.

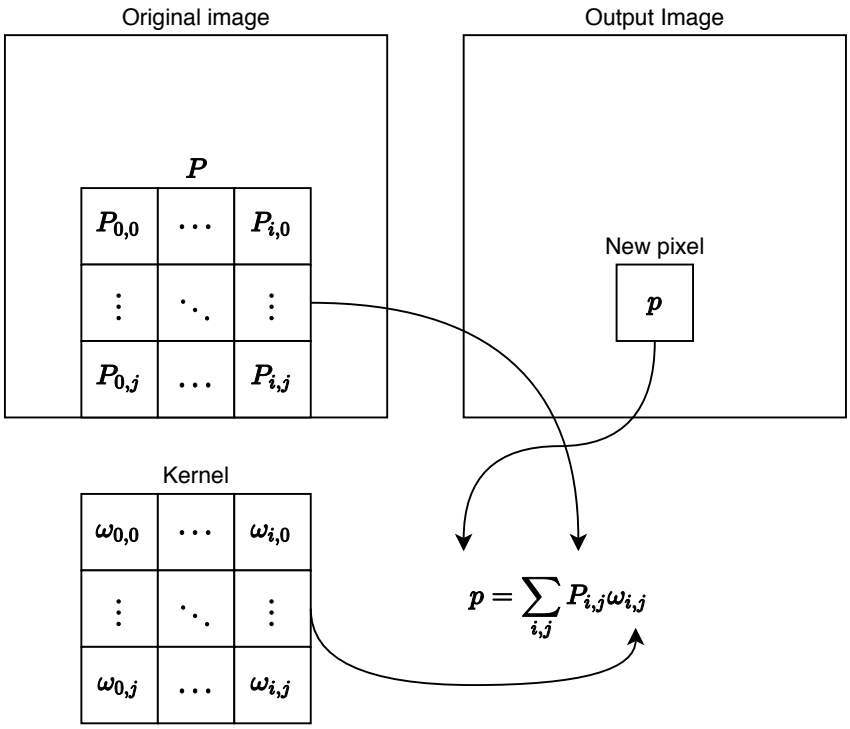


Figure 2.2: Image convolution with kernel

Convolution of images with a kernel is a common way of processing images. The technique is used for blurring, edge detection, sharpening images, gradient detection, and more. Figure 2.2 shows how the calculation of one pixel of an image using this method. The kernel is centered over a pixel in the original image. Each of the pixels it overlaps with multiplies by that kernel pixel value. The pixel value of the output image is then calculated by adding each of the products together. This process is then repeated for all the pixels in the image. Note that the kernel needs to have an odd-numbered kernel size, as there needs to be only one pixel in the middle of it, and not four. This whole process can be written mathematically as stated in equation 2.3, where $f(x, y)$ is the input image, ω is the kernel and $G(x, y)$ is the output.[8] Figure2.3 is an example, which uses the 3 by 3 averaging filter kernel.

$$G(x, y) = f(x, y) * \omega \tag{2.3}$$

Kernel

$$\omega = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Pixels kernel overlaps

$$P = \begin{bmatrix} 128 & 90 & 128 \\ 134 & 100 & 10 \\ 54 & 226 & 30 \end{bmatrix}$$

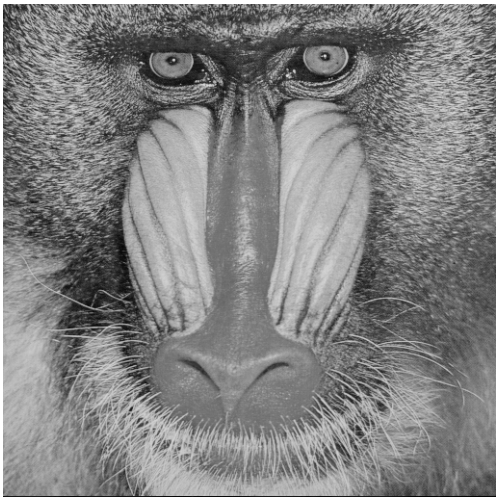
Figure 2.3: 3 by 3 averaging filter kernel and what it is applied to

Each of the kernel pixels is multiplied with the corresponding overlap and added. The output pixel value is therefore $p = 1/9(128+90+128+134+100+10+54+226+30) = 100$. This is also the average value of P, hence the name of the kernel.

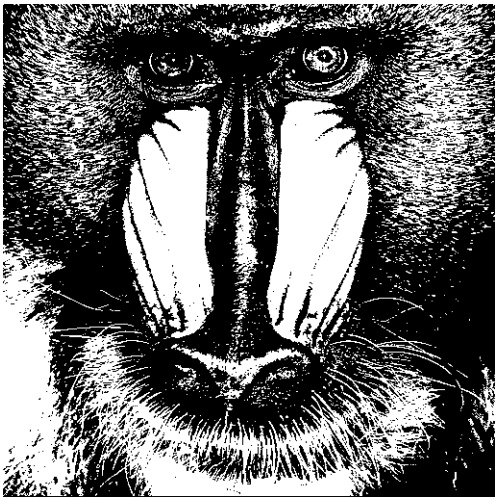
2.3 Binarization

Binarization is the act of taking input data that contains a varying degree of variables and translating them to a simple string of binary data. Image Binarization is then the act of translating the image channels, as explained in section 2.2.3, into a simple list containing only binary data. The primary reason why binarization is required is due to the usage of the Tsetlin Machine in this thesis, and further explanation is shown in section 2.7.1.

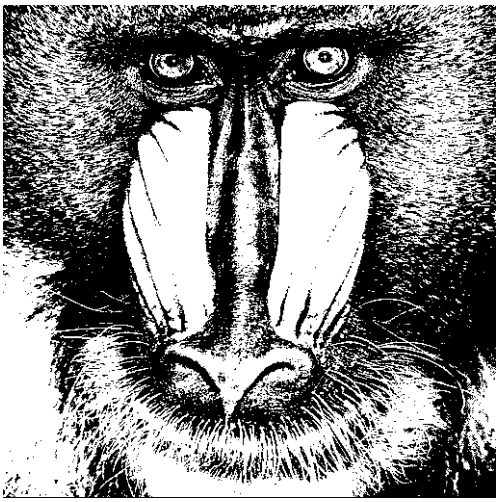
There are multiple techniques of how this binarization is achieved. Through this section, an introductory description is presented for those used in this thesis. Following is an example of the different types of binarization and the effect they have when applied to a grayscale image.



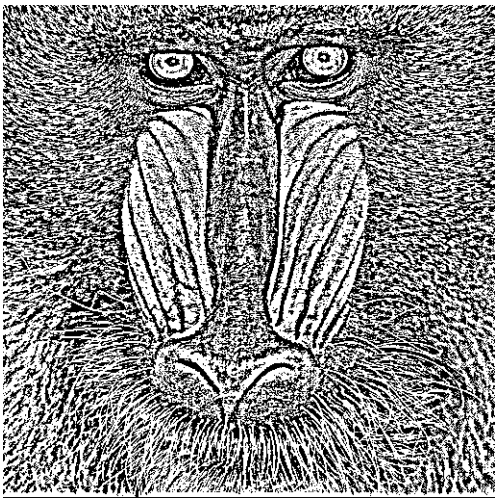
(a) Original grayscale



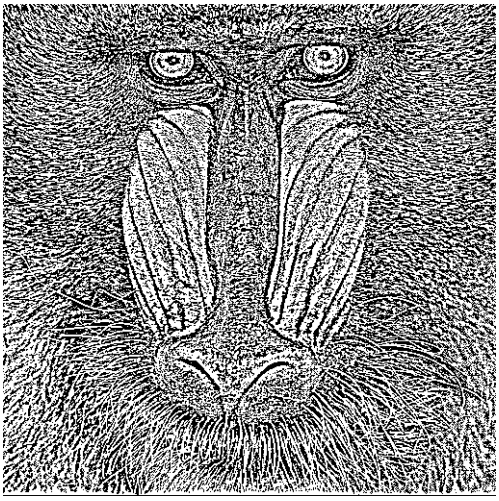
(b) Static global treshold



(c) Otsu's method



(d) Adaptive tresholding



(e) Gaussian adaptive tresholding

(f) Canny

Figure 2.4: Binarization techniques compared

2.3.1 Global Tresholding

Originally the binarization used by Ole-Christoffer Granmo when testing the Tsetlin machine against the MNIST dataset[1]. The purest form of binarization. It works as shown in equation 2.4, by taking a threshold function $T(x, y)$ on a channel. The function takes a static threshold value T_r on each pixel (x, y) in a channel, and makes a new channel with the new pixels. This is a global binarization technique, and will, therefore, work best on images that have the same brightness throughout the image. Therefore, images containing varying degrees of lighting could potentially lose essential features as part of the binarization, or in extreme cases, could attain features that would lead to miss-classification. This type of binarization might not be suited for complex images, such as those used in this thesis.

This binarization technique can be efficient on datasets where each image has defined divides in the channel intensity histogram, and each image does not have shadows. The MNIST dataset works fine with this, but a more complex image would, with a higher potential, lose data with this transformation.

$$T(x,y) = \begin{cases} 1 & if(x,y) > T_r \\ 0 & else \end{cases} \tag{2.4}$$

2.3.2 Otsu

Otsu is a global binarization method that focuses on classifying the background and foreground of an image. It does this by minimizing the weighted class average (σ_w^2). If there are two well-defined peaks in the histogram, this equation results in a set threshold between the two.[9][10]

$$\sigma_w^2 = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t) \tag{2.5}$$

$$q_1(t) = \sum_{i=1}^{t} P(i) \;\; \& \;\; q_2(t) = \sum_{i=t+1}^{I} P(i) \tag{2.6}$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \;\; \& \;\; \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)} \tag{2.7}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \;\; \& \;\; \sigma_2^2(t) = \sum_{i=1+t}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)} \tag{2.8}$$

The weighted class average is calculated by the equation (2.5), where t is the threshold value, $q_1(t)$ and $q_2(t)$ are the cumulative sum of the classes and σ_1^2 and σ_2^2 is their respec-

tive variances. For the cumulative sum of the classes equation (2.6) is used, where the histogram of the image of the image is split up into I bins (255 bins) and normalized. The normalized histogram can now be used as a probability mass function, as the sum of all the normalized bins is 1. $P(i)$ is the probability that the value is in the bin i. In equation (2.7) the mean of the classes are calculated, the variance is calculated in equation (2.8). The weighted class average is calculated for all threshold values. The threshold value that ends up being used with the global threshold algorithm (T_r in equation 2.4) is the one that produces the lowest weighted class average (σ_w^2).

2.3.3 Adaptive

"The threshold value T(x,y) is a mean of the blockSize*blockSize neighborhood of (x,y) minus C " - OpenCV[9]

$$T(x,y) = \begin{cases} 1 & if\ (x,y)\ >\ G(x,y) - C \\ 0 & else \end{cases} \tag{2.9}$$

$$G(x,y) = \omega * f(x,y) \tag{2.10}$$

$$\omega = \frac{1}{\beta} \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}_{k_s \times k_s} \tag{2.11}$$

This binarization technique calculates each pixel locally instead of globally. It takes the average of a square around the pixel minus a constant. If the resulting value is over the threshold value, the pixel is set to true (1); if not, it is set to false (0).
The threshold function described in equation (2.9) uses an average filtered image $G(x,y)$ of the original, minus a constant C to find if the pixel should be 1 or 0. Equation 2.10 describes a general convolution of a kernel ω and the input image $f(x,y)$. The kernel described in equation 2.11 $G(x,y)$ becomes an filtered image of the original with a window of kernel size (k_s) times kernel size (blocksize in OpenCV). The sum of the averaging kernel pixels adds up to 1, this is achieved by normalizing it with β.
Compared to the global threshold algorithm, this one is not as much affected by the shadows and complexity of saturation in an image. It can translate critical information into a binarized version of the original image. Thus compared to Global Threshold, it

retains more features even when presented with an image containing various light levels.

2.3.4 Gaussian

"the threshold value T(x,y) is a weighted sum (cross-correlation with a Gaussian window) of the blockSize*blockSize neighborhood of (x,y) minus C . The default sigma (standard deviation) is used for the specified blockSize . " -OpenCV[9].

This algorithm is very similar to adaptive thresholding. The only difference is that it uses a gaussian blurring kernel instead of an averaging kernel in the calculation of $G(x, y)$ in equation(2.10).

$$\omega = \frac{1}{\beta} A A^T \tag{2.12}$$

$$A = \begin{bmatrix} A_0 \\ \vdots \\ A_i \\ \vdots \\ A_{k_s-1} \end{bmatrix} \tag{2.13}$$

$$A_i = \alpha e^{-(i-(k_s-1)/2)^2/(2\sigma^2)} \tag{2.14}$$

$$\sigma = 0.3((k_s - 1)0.5 - 1) + 0.8 \tag{2.15}$$

$$\tag{2.16}$$

The Gaussian blurring kernel(ω) is produced as shown in equation 2.12, where β is calculated by the sum of all the elements of AA^T. This results in the sum of all elements in ω becomes 1. This is done to normalize the kernel, so that the image it is used on will not become brighter. A is a one dimensional Gaussian kernel (equation 2.13) filled with values calculated with equation 2.14 and 2.15, where α is adjusted so that $\Sigma_i A_i = 1$.
Compared to the averaging filter, the Gaussian blur filter keeps more edges in an image. This is because the pixels closer to the pixel calculated are more valued than in the edges of the window. This potentially keeps more data from the image after thresholding.

2.3.5 Canny

Canny is a binarization algorithm that focuses on the edges in the image instead of focusing on whenever something has a value over or under a threshold[11]. Canny proceeds in the following four steps:

1. Gaussian blurring, to remove noise in the image. This filter kernel is described in Gaussian Thresholding (section 2.3.4). The kernel size used here is 5.

2. Using the Sobel operator, which is an edge detection algorithm that produces two sets of images with kernels that makes it so that the result approximates derivation in the x- and y-direction. These two are then combined to produce the gradient of the blurred image. The direction of the gradient is also calculated.

3. non-maximum suppression reduces the image down to thin edges. Here a pixel is kept at its current value if it's a local maximum in the gradient direction, otherwise to zero.

4. Hysteresis thresholding finds what edges are connected and keep them. As the name of this algorithm suggests, there are two values set, maxVal and minVal, which defines the hysteresis boundaries. Any edge over maxVal is set to true. The hysteresis kicks in as any edge between minVal and maxVal that is connected to an edge over maxVal are set to true. The rest is set to false. The point of doing this is only to keep the dominant edges, e.g., the shape of an object instead of the texture.

2.4 Logic Gates

A fundamental mechanic in computing is the application of logic gates. Understanding these basic bit manipulation mechanics will further help in understanding the fundamentals of the Tsetlin Machine which will be explained later in the thesis. Two main gates are of interest in this thesis and they are the AND and OR Gates. The following are tables which explain how these logic gates work.

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.1: Logic AND gate

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

Table 2.2: Logic OR Gate

2.5 Accuracy Measurements

How one measures the correctness of a machine learning technique is extremely important. The two methods of correctness that are used in this thesis are the standard accuracy for the final output of the Tsetlin Machine, and the confusion matrix, which is necessary to understand in order to calculate accuracy.

2.5.1 Confusion matrix

Confusion Matrix is a technique to show the various states the potential output of a machine learning algorithm can have. This is done by comparing the output of the machine learning algorithm with the true class value. This comparison causes the output to fall within four states, which are as follows:

- **True Positive (TP)**: The algorithm predicts correctly that the image is in the class
- **True Negative (TN)**: The algorithm predicts correctly that the image is not in the class
- **False Positive (FP)**: The algorithm predicts wrongly that the image is in the class
- **False Negative (FN)**: The algorithm predicts wrongly that the image is not in the class

These states are normally presented within a Confusion Matrix, which is as follows:

		Actual classs	
		Positive	Negative
Class Prediction	Positive	TP	FP
	Negative	FN	TN

The table presented above is only for a single class classifier. For multi-state classifiers, this table extends outwards in a similar fashion.

2.5.2 Accuracy

Accuracy is defined as shown in equation 2.17, and can be denoted as the ratio of correct classifications over total classifications. However, with unbalanced data, this evaluation of the classifier should not be used, as it can give skewed results. If an algorithm first classifies a dataset with 5 cats and 5 cars (this is a balanced dataset) and gets one car wrong, the accuracy would be $ACC = \frac{5+4}{10} = 90\%$. The same algorithm is then again tasked to analyze 9 pictures of cats, and 1 picture of a car, which is a very unbalanced dataset. This time it classifies all pictures as cats and gets the accuracy of $ACC = \frac{9+0}{10} = 90\%$, which is misleading. It gives no indication of learning the classification of a car. Other evaluation techniques are used for unbalanced datasets such as the F1 score and Matthews Correlation Coefficient to deal with this issue[12]. In this thesis, all the datasets are balanced, so accuracy can be used to evaluate the results. Accuracy can be extended to calculate multi-class predictions with equation 2.18 [13].

$$ACC = \frac{TP + TN}{P + N} \tag{2.17}$$

$$ACC = \frac{True\ Predictions}{All\ Predictions} \tag{2.18}$$

2.6 CIFAR10

The CIFAR dataset is a set of 32x32 color images divided into 10 classes. Each class contains a set of 5000 training images and a set of 1000 testing images[14]. It has been used in multiple publications to measure how well an algorithm can classify color images. It is the dataset in which we test the capabilities of the Tsetlin Machine and its various adaptions.
The dataset encompasses 10 classes, numbering from 0 to 9, and is denoted in the following table.

class	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
label	0	1	2	3	4	5	6	7	8	9

2.7 Tsetlin Automaton

In stark contrast to the increasing complexity of the deep neural network stands the Tsetlin machine. However, to adequately explain the Tsetlin Machine, one needs to understand the fundamentals of it accurately. Namely, the Tsetlin Automaton. While the neural network nodes are simple in nature, they also gain massive complexity as the networks grow in size. Comparably the Tsetlin Automatons simplicity allows it to scale up without significantly adding to the overall complexity.
The Tsetlin Automaton is, at its core, a straightforward decision-making mechanism that has attracted interest due to its ability to learn optimal actions when operating in unknown stochastic environments [1]. Each Automaton performs a binary action, and through rewards or punishments, it helps define the accuracy of these actions onward.

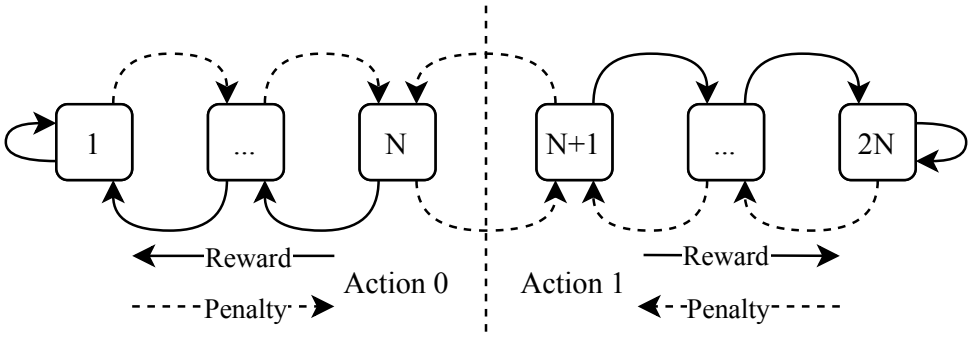


Figure 2.5: Two action Tsetlin Automaton

Shown in the figure 2.5, the decision making of a automaton is that of a two state Markov chain with 2N steps. Depending on the automatons state location in the chain. The final output will follow thereafter. Following figure 2.5 if the automaton state is located between 1 to N, the action selection (A) will result in 0. If the automaton is on a state between N+1 to 2N the final action A will result in 1. This is further shown in the following equation.

$$A(\varnothing_u) = \begin{cases} \alpha_0 & \text{if} \quad 1 \leq u \leq N \\ \alpha_1 & \text{if} \quad N+1 \leq u \leq 2N \end{cases} \tag{2.19}$$

2.7.1 Learning Automaton

While the simplicity of the Automaton does lend well to its scale-ability, it does require the input to be simple, in that it can only process single discrete-binary data as both its training and evaluation data. This, in turn, leads to a potentially large amount of pre-processing of the data before it can be utilized. How this is achieved for this thesis is explained further in section(2.3). Initialization of the Automaton happens with the number of states N being set. After N states are set, the Automaton chooses a random integer from 1 to 2N as its placement $\emptyset_u$ where $u \in 2N$. After the initialization, the only external events v the Automaton receives are either punishments or rewards, as shown in figure 2.6.

$$F(\emptyset_u, \beta_v) = \begin{cases} \emptyset_{u+1}, & if\ 1 \leq u \leq N \text{ and } & v = Penalty \\ \emptyset_{u-1}, & if\ N+1 \leq u \leq 2N \text{ and } & v = Penalty \\ \emptyset_{u-1}, & if\ 1 < u \leq N \text{ and } & v = Reward \\ \emptyset_{u+1}, & if\ N+1 \leq u < 2N \text{ and } & v = Reward \\ \emptyset_u, & Otherwise & \end{cases} \tag{2.20}$$

Figure 2.6: Reward / Penalty equations for Tsetlin Automatons

2.7.2 Automaton Teams

The previous section provided a fundamental description of the Tsetlin Automaton. Through this section, we explain how these singular Automatons can achieve more significant decision making. Namely the mechanics of gathering multiple Automatons into teams or collectives.

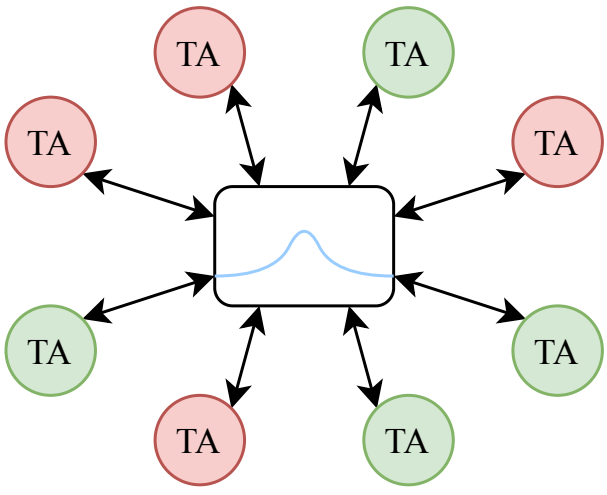


Figure 2.7: The Gur Game representation

Through the use of an experiment called The Gur Game presented by M.L Tsetlin [15], one can adequately show the ability for an Automaton team to operate within unknown stochastic environments. The Gur Game is a game of individual cooperation between agents in which they have no knowledge of each other and are strictly unable to communicate with one another. The goal of the game is for a set amount of the agents participating in returning a active signal while the remaining are inactive. To achieve this optimal ratio of active and inactive, we utilize a reward normal distribution probability function ($R(f) < 1$), which peaks at the desired ratio of automatons being active. As long as the reward probability at the desired ratio is above 0.5, the Automatons, eventually, achieve and secure the desired ratio by moving all inactive automatons to the state 1 and active towards 2N[16].

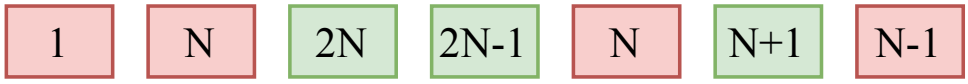


Figure 2.8: Automaton team with individual states ranging between 1 and 2N.

In classification examples such as one shown in figure 2.8, the team needs to form a consensus to decide upon the team's final vote. This is done through the Majority Vote. The Majority Vote is a simple tally on final action counts. In the case of figure 2.8, the majority vote would result in Action 0 because of the majority of the Automatons currently assigned to states N or below.

2.7.3 Conclusion

Through this Chapter, various fundamental mechanics that are crucial to this thesis were introduced. Initially, with the introduction of image processing mechanics, a thorough description of how these techniques show how image pre-processing is needed for the Tsetlin Machine. The various techniques, ranging from Static, adaptive, gaussian, otsu, and canny, have notable influences on the final output and changes the feature sets used. A further detailed description of the fundamental theory on which the Tsetlin Machine is built was produced. These simple Markov Chains, when grouped, can defeat even the prisoners' dilemma[1].

This concludes the first goal of the thesis, which was to provide an introductory description of how these older techniques are used in tandem with the state-of-the-art technique Tsetlin Machine. This section was merely theoretical in nature, and a description of the actual implementation will be further in the thesis.

Part III

Tsetlin Machine

Chapter 3

Introducing the Tsetlin Machine

Through section 2.7 we explained fundamental mechanics of the Tsetlin Automaton, through this chapter, we go more in the depth of how the primary Tsetlin Machine operates and further variations of the Tsetlin architecture.

In April 2018, Prof. Ole-Christoffer Granmo published the Tsetlin Machine. Tsetlin Machine is a machine learning technique that builds upon an older learning mechanism, namely the Tsetlin Automaton, which was created by Mikhail Lvovitsj Tsetlin in the soviet union in 1960[1]. Through this paper, Prof. Granmo describes the in-depth mechanics of the Tsetlin Automaton and how such a simple mechanism such as the Tsetlin Automaton can be used for complex tasks such as the multi-armed bandit problem[1].
In short succession to the publication of this paper, multiple revisions and adaptions have been submitted. Multiple of which are utilized through this thesis such as, Multi-Class-, Convolutional-, Weighted-, and Layered Tsetlin Machine. All of which have an introductory explanation through this chapter.

3.1 Multi Class Tsetlin Machine

In this section, we explain the mechanics of the Standard as well as the Multi-Class Tsetlin Machine proposed by Prof. Granmo[1]. Section 2.7 explained one of the fundamental mechanics of the Tsetlin Machine, namely the automatons, here we explain further how the automaton teams can be utilized with a classification algorithm to provide a competitive machine learning technique. Initially, the paper explained only the Standard Tsetlin Machine, which classifies either single class or a class pair, however with revisions, it ended up including the Multi-Class Tsetlin Machine, which has the added capability of classifying multiple subjects.

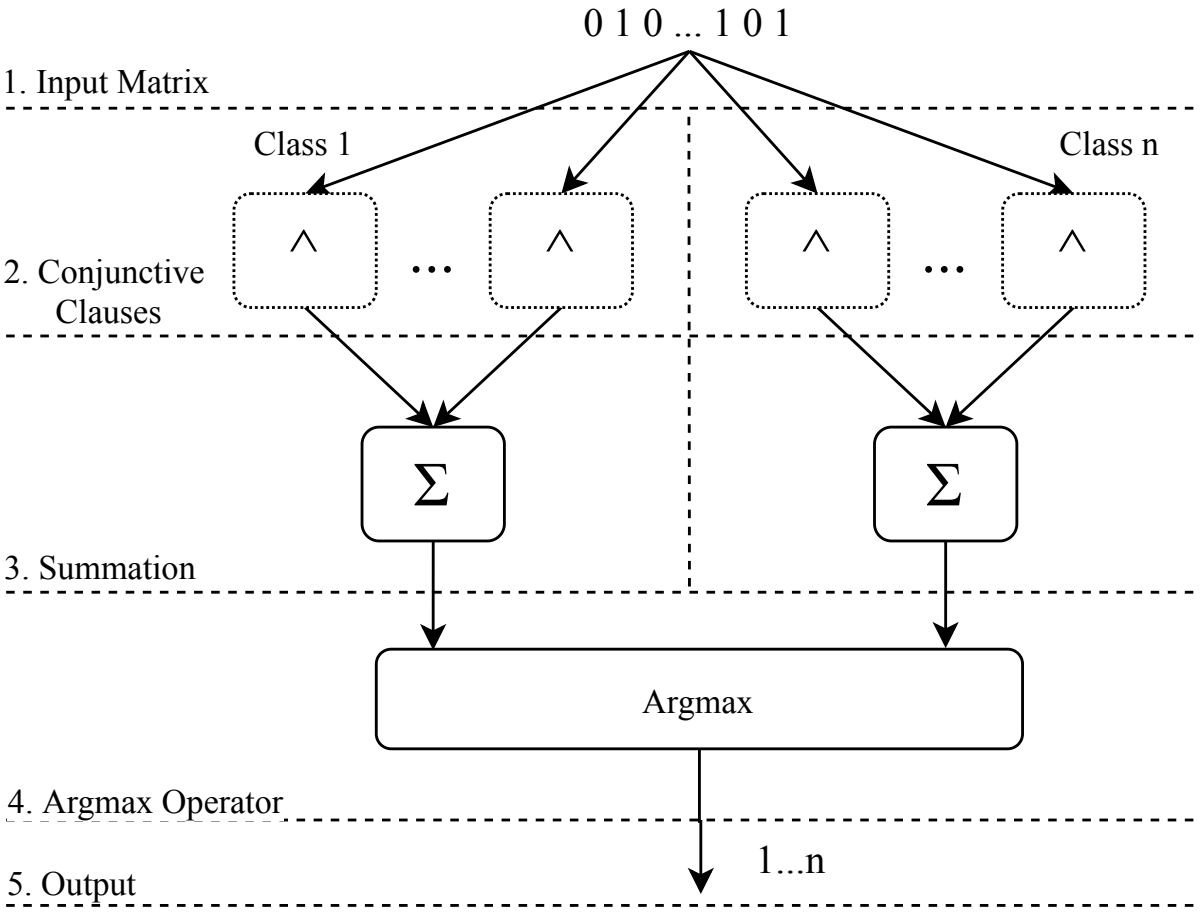


Figure 3.1: Step by Step overview of a Multi-Class Tsetlin Machine

Figure 3.1 shows a graphic step-by-step process that the Multi-Class Tsetlin Machine goes through for every classification process. These five steps are as follows:

1. **Input:** The initial input to the Tsetlin machine is a 1D matrix consisting solely of binary values.

2. **Conjunctive Clauses:** Clauses are comparison matrices used by the Tsetlin machine to evaluate every input matrix. For every positive Clause, there is a negated Clause to help classification. They output a simple binary statement that denotes whether the subject correlates to the Clause or not (1/0).

3. **Summation:** Through the summation section, the various clauses are summated in what is explained in section 2.7.2 as the majority vote.

4. **Argmax Operator:** After the summation of the clauses per class. The index number is collected for every class, and the most likely class is presented.

5. **Output:** Finally, the output is based on the presented most likely class and is then output as whatever arbitrarily defined variable the developer wishes. In our case, the class number.

This figure is similar to the one proposed by Prof. Granmo in his paper[1], where he correlates the process the Tsetlin Machine takes as primarily a game.

3.1.1 Input

While the Tsetlin Machine does very little pre-processing of input data, what it does do is extremely important to the entire process. As explained earlier, the input for the Tsetlin Machine is a 1D matrix containing only binary features. Before this data can be used, the Tsetlin Machine replicates it and negates it. Pairing this new negated data with the standard one, which doubles the number of features for the input matrix. This set of new features is also frequently named a set of literals. A literal l is the feature elements within the now doubled input matrix, and denotes as a set of literals L.
This is generally written as: $l_k \in L_j^i$

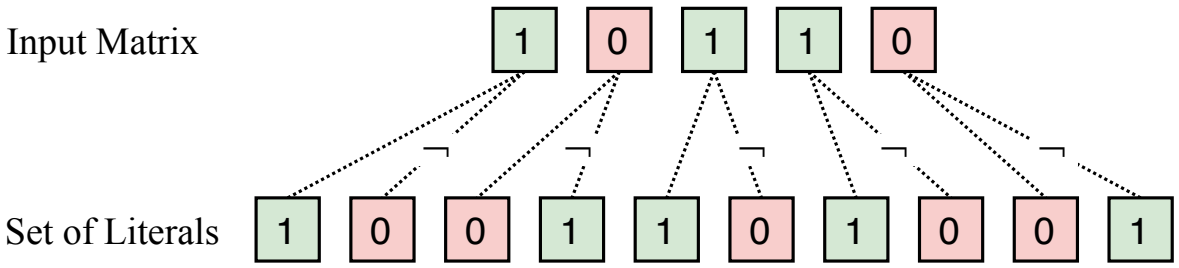


Figure 3.2: Conversion of a input matrix to a set of literals

3.1.2 Conjunctive Clauses

Understanding literals and their essential role in the Tsetlin Machine, they can now be combined with the automaton teams, which were explained in section 2.7.2. These two fundamental mechanics then combine into what now presents as a clause, with which the Tsetlin Machine classifies. In its purest form, a clause can be explained as an Automaton Team where every literal correlates to a unique automaton.

$$C_j^i(X) = \bigwedge_{l_k \in L_j^i} l_k \tag{3.1}$$

As shown in equation 3.1, a Clause C_j^i consists of a subset of the set of literals L where each unique automaton has to decide whether the corresponding literal l_k should be included or excluded in its assigned Clause. Initially, this is done randomly, as explained in section 2.7.2, every automaton has a random set state. If this set state is $N + 1$ or higher, the corresponding literal is included. As the Tsetlin Machine goes on, it updates the included and excluded literals through a feedback system explained in section 3.1.4. After grouping up all the automatons and having decided whether their given literal should be included or excluded, the final Clause would be similar to the following figure.

$$\begin{array}{cccccccccc}
0 & 0 & * & * & * & * & * & 0 & * & 0 \\
0 & 0 & * & * & 1 & 1 & 0 & * & 0 & 0 \\
0 & 0 & * & * & 1 & * & * & 0 & * & 0 \\
0 & * & * & * & 1 & * & * & * & * & 0 \\
0 & 0 & * & 0 & * & 1 & * & 0 & * & 0 \\
0 & * & 0 & * & 1 & * & * & * & 0 & 0 \\
0 & 0 & * & * & 1 & 1 & * & 0 & 0 & 0 \\
0 & * & 0 & * & 1 & * & 0 & * & 0 & 0
\end{array}$$

Figure 3.3: Example of a bit pattern depicting the gathered literals in a clause that might be used to classify written 1.

Shown in figure 3.3 is an example of a bit pattern from a potential image classification test. In this example, the 0 or 1 denotes automatons that have chosen to include their literals, 0 being negated, and 1 being positive literals. The asterisks depict excluded literals that function as a wildcard. This bit pattern effectively works as a stencil over the input data. The following figure is another example of how this stencil is then used to help with the classification.

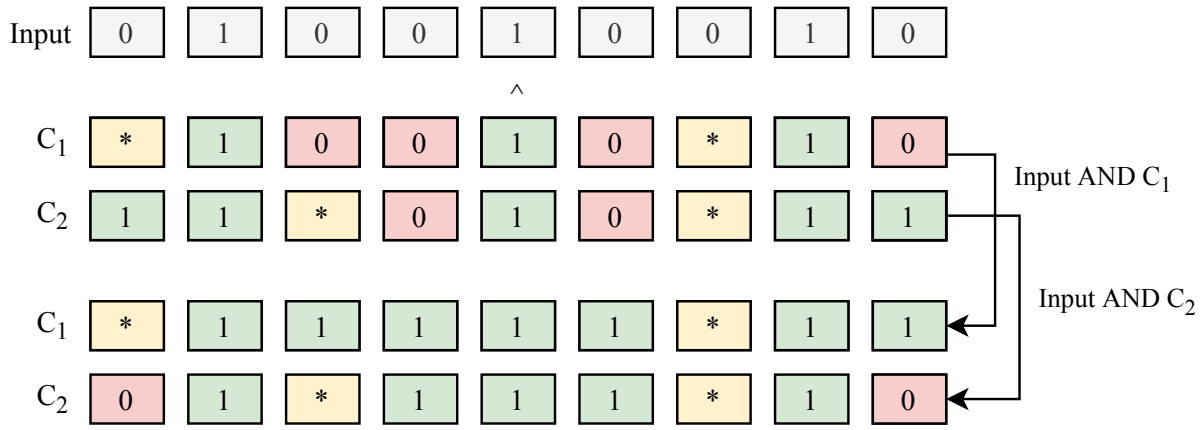


Figure 3.4: A Input Matrix used together with two different clauses for classification comparison that uses the AND logic gate.

The only way a clause outputs as True is if every literal included in the Clause correlates with the equivalently indexed input element.

3.1.3 Summation

Explained previously under section 3.1.2 every clause is given a unique index number. The odd-numbered clauses are counted as negative votes, effectively learning what is not the wanted class, while the even-numbered clauses are positive. This includes the first Clause, C_0. This is further shown in the following equation.

$$\widehat{y}^i = u\left(\sum_{j=1}^{n/2} C_j^{i+}(X) - \sum_{j=1}^{n/2} C_j^{i-}(X)\right) \tag{3.2}$$

$$\widehat{y}^i = argmax_{i=1,\ldots,m}\left(\sum_{j=1}^{n/2} C_j^{i+}(X) - \sum_{j=1}^{n/2} C_j^{i-}(X)\right) \tag{3.3}$$

In this equation the C_j^{i+} are even indexed clauses that output true if they classify the image as the given class I.E the image IS a cat
While C_j^{i-} are odd indexed clauses that output true if they classify the image as NOT the targeted class, I.E the image IS NOT a cat, $\widehat{y}^i$ is the final summation and defines the final output. $argmax_{i=1,...,m}$ is the common denotation of finding the index i with the highest value to calculate the output, and is used to the Multi-Class Tsetlin Machine.

3.1.4 Feedback

This is the last section on the standard Tsetlin Machine where we explain a fundamental mechanic and before we start going into detail about the other adaptations. Through this section, we explain how the Tsetlin Machine achieves learning by increasing or decreasing the literals within the clauses explained in section 3.1.2.
Looking back to figure 3.1 we have essentially explained the structure through sections 3.1.1 - 3.1.3. What remains is the mechanic in which the Tsetlin Machines achieve learning after each round of its game. This process takes place after every single round of the Tsetlin Machine Game and before starting again. In this step, it potentially changes the included or excluded literals in a clause by Clause basis.
There are two types of main feedback tables below. There is also a third feedback possibility not listed. This feedback type is ignore, do not change. In short, we have the following feedback with their given area of influence.

- **Type 1 Feedback** is given to clauses if they produce either a false negative or true positive. I.E a positive clause outputs a 0 or 1 if the target class is a 1.
- **Type 2 Feedback** is given to clauses if they produce a false positive. I.E a positive polarity clause outputs a 1 when the target class is a 0.
- The final feedback given to clauses is **Ignore**. This only happens if the Clause produces a True Negative, and the final Clause does not change the state structure in this case.

As well as having to satisfy the requirements listed above per feedback table, for both types of feedback, there is also a probability of whether or not they receive feedback. This probability has the intent of making the clauses distribute themselves equally over the sub-patterns in the data towards a target summation T. In the following equations, the clamp functions as a limit of $-T$ to T. Equation 3.4 shows the probability of generating Type I Feedback, while equation 3.5 shows the probability of generating Type II Feedback.

$$\frac{T - \textbf{clamp}\left(\sum_{j=1}^{n/2} C_j^{i+}(X) - \sum_{j=1}^{n/2} C_j^{i-}(X),\ -T, T\right)}{2T} \tag{3.4}$$

$$\frac{T + \textbf{clamp}\left(\sum_{j=1}^{n/2} C_j^{i+}(X) - \sum_{j=1}^{n/2} C_j^{i-}(X),\ -T, T\right)}{2T} \tag{3.5}$$

Type I Feedback - False Negative

As shown above, Type I Feedback table is given when a clause produce either a false negative or a true positive towards their target class.
$C_j^{i+}(X) = 1$ *or* $C_j^{i+}(X) = 0$ *where* $y^i = 1$ **OR** $C_j^{i-}(X) = 1$ *where* $y^i = 0$.
Type I feedback tries to leverage the Clause to include more literals from the input matrix to improve the accuracy of the True Positive clauses. This is shown in the table where, to include literals has a higher chance of reward compared to exclude. While False Negatives are leveraged slightly towards exclusion.

Truth Value of Clause (C_j^{i+})		1		0	
Truth Value of Literal (l_k)		1	0	1	0
Include Literal ($l_k \in L_j^{i+}$)	P(Reward)	$\frac{s-1}{s}$	NA	0	0
	P(Inaction)	$\frac{1}{s}$	NA	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	P(Penalty)	0	NA	$\frac{1}{s}$	$\frac{1}{s}$
Exclude Literal ($l_k \notin L_j^{i+}$)	P(Reward)	0	$\frac{1}{s}$	$\frac{1}{s}$	$\frac{1}{s}$
	P(Inaction)	$\frac{1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	P(Penalty)	$\frac{s-1}{s}$	0	0	0

Table 3.1: Type I Feedback

Type II Feedback - False Positive

Type II Feedback is given whenever the Clause produces a False Positive. I.E the input matrix is NOT the class this Clause is trained on, yet it still outputs a positive state. This could potentially be an image of a cat is classified as NOT cat by a negative affixed clause. $C_j^{i+}(X) = 1$ *where* $y^i = 0$ **OR** $C_j^{i-}(X) = 1$ *where* $y^i = 1$.
Type II feedback tries to leverage the Clause to include literals from the input matrix to reduce the chance of triggering false positives by adding harsher stencils.

Truth Value of Clause (C_j^{i+})		1		0	
Truth Value of Literal (l_k)		1	0	1	0
Include Literal ($l_k \in L_j^{i+}$)	P(Reward)	0	NA	0	0
	P(Inaction)	1.0	NA	1.0	1.0
	P(Penalty)	0	NA	0	0
Exclude Literal ($l_k \notin L_j^{i+}$)	P(Reward)	0	0	0	0
	P(Inaction)	1.0	0	1.0	1.0
	P(Penalty)	0	1.0	0	0

Table 3.2: Type II Feedback Table

3.2 Tsetlin Machine Parallel

In December 2019, Prof. Granmo added another adaption to the Tsetlin Machine family, namely the Tsetlin Machine Parallel [17]. By running multiple inputs, each running on its own thread. The Tsetlin Machine can massively increase the speed of training with minimal loss. When a clause is done, it might update the states based on the outcome. While this update is being done, it locks down the clause state from other threads. The principal loss occurs if there is a change in the clause state after another thread has already initiated it. Because it cannot be changed after the process has been started, there is potential for loss of learning in these steps. However, the massive reduction in time compared to running a completely sequential single thread Tsetlin is so large that the small possible loss is deemed acceptable since overall loss appears be minimal[17][18].

3.3 Convolutional Tsetlin Machine

The Convolutional Tsetlin Machine is the first adaption of the Tsetlin Machine we explain. It was initially theorized in the revised publication of the Original Tsetlin Machine in 2018[1] and was given a full publication in 2019[4]. Motivated by the impact convolution made to deep neural networks, this adaption came to.

The changes from the standard Tsetlin Machine can be generalized into the following three items.

1. **Input Data** The Convolutional Tsetlin Machine does additional pre-processing of the data compared to the standard.
2. **Clauses** as a consequence of the change to the input data, the clauses must also be changed.
3. **Feedback** cascading effect of the other two changes causes the feedback loop to change.

While these three items contain the bulk of the foundations of the Tsetlin Machine, the actual changes are not as severe. Therefor with a baseline in what was explained in section 3.1 we will, through this section, explain the main difference in this adaption.

3.3.1 Input

The Standard Tsetlin Machines utilizes the entire input matrix together with the negation, equating to the set of Literals L, which results in clauses becoming $X \times Y \times Z \times 2$ and potentially growing massive. With the adaption of the Convolutional Tsetlin Machine, these input matrices are reduced to a fixed size denoted as $W \times W$ and shown further in the following figure.

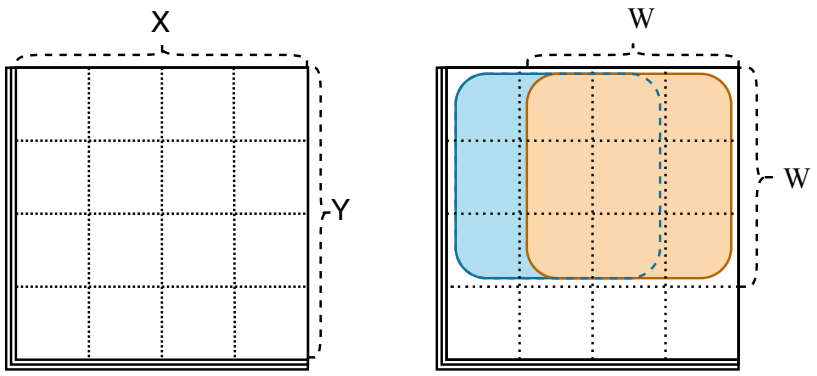


Figure 3.5: Tsetlin Input comparison between Standard and Convolutional with patches. Image size is set to X for Width, Y for height and Z for depth. While W for the length of the equal sized patch

As shown in the figure above, the new input will therefor be cut down from the clause length as explained in section 3.1.2 to fixed patch size, but will in turn generate far more input matrices per image. As equation 3.6 shows the estimated amount of patches generated per image X Y W are all noted on figure 3.5 while Z denotes the amount of channels, d is the step size between patches.

$$B = B_X \; x \; B_Y \; x \; Z \; where \; B_X = \left[\frac{X - W}{d}\right] + 1 \; and \; B_Y = \left[\frac{Y - W}{d}\right] + 1 \tag{3.6}$$

With each patch is also a coordination vector to point to the correct placement within the image.

3.3.2 Clauses

Understanding the change to the input matrices is important for understanding the changes to the clauses. Considering the change from the standard Tsetlin with a single input matrix per image to now have the amount is shown in equation 3.6, the changes to the recognition through clauses have to be done. Each clause now outputs potentially B values per image. To alleviate the large increase in extra values a logical OR gate is added to the gathered output of both positive and negative clauses. This is shown in equation 3.7. B and b denote the patch index numbers.

$$c_j^+ = \bigvee_{b=1}^{B} c_j^{b,+} \tag{3.7}$$

3.3.3 Feedback

With the added increase in input matrices per Tsetlin Machine Game loop, some slight changes have to be made to the Feedback loop. For every input matrix put through the Tsetlin Machine Game, it contains B patches, which in turn results in B literal inputs (literal inputs are explained in section 2.7.1 as the reward of punishment feedbacks) $L^b, 1 \leq b \leq B$. Because of this, to decide the patch for which it will use to update the clause, it randomly chooses a patch in which the clause outputs to 1. $\left\{X^b | c_j^{b,+} = 1, 1 \leq b \leq B\right\}$ should however no clauses fit this requirement, feedback table Ib will be called. Ib leverages the clause to exclude literals.

$$\mathrm{I}_{C+}^{Ia} = \left\{(j,k) | l_k^b = 1 \wedge c_j^+ = 1 \wedge r_{j,1}^+ = 1\right\} \tag{3.8}$$

$$\mathrm{I}_{C+}^{Ib} = \left\{(j,k) | \left(l_k^b = 0 \vee c_j^+ = 0\right) \wedge r_{j,1}^+ = 1 \wedge q_{j,k}^+ = 1\right\} \tag{3.9}$$

$$\mathrm{I}_{C+}^{II} = \left\{(j,k) | l_k^b = 0 \wedge c_j^+ = 1 \wedge r_{j,0}^+ = 1\right\} \tag{3.10}$$

Feedback table Ia reinforces the Include Literal actions to increase pattern fineness. In contrast, Ib reinforces the exclude literals to combat overfitting, which is a consequence of oversaturating the clauses with literals. $r_{j,1}^+$ and $q_{j,k}^+$ are both explained in the following equations and are approximations of equation 3.4 - 3.5 in which the probability of feedback to their respective clause is based on the approach to T.

$$r_{j,1}^+ = \begin{cases} 1 & with\ probability\ \ \frac{T - clamp(v,-T,T)}{2T} \\ 0 & otherwise \end{cases} \tag{3.11}$$

$$q_{j,k}^+ = \begin{cases} 1 & with\ probability\ \ \frac{1}{s} \\ 0 & otherwise \end{cases} \tag{3.12}$$

3.4 Weighted Tsetlin Machine

The second adaptation of the Tsetlin Machine to be explained is the Weighted Tsetlin Machine. It was publicized in late November 2019[5] and is a simple adaptation of the Tsetlin Machine. With more and more complicated unique patterns for classification, the potential need for large scale clause numbers ever grows. As a consequence of this is the increasing complexity, computation requirement, and memory usage, and to help negate and reduce these ever-increasing requirements came the Weighted Tsetlin Machine adaptation. In this adaptation, they present the usage of adding weights to clauses, and through this section, we examine and detail how this adaptation is different from the standard Tsetlin Machine.

3.4.1 Clauses

The main difference between the standard Tsetlin Machine is the addition of weights per clause. This gives the potential to increase the variety of the clauses. This is done by adding a weight value to each clause and also a large increase in the T variable which governs the coarseness of the clauses (explained in section 3.1.4).

$$w_j^+ \leftarrow 1.0 \tag{3.13}$$

$$w_j^- \leftarrow 1.0 \tag{3.14}$$

As the Tsetlin Machine is initialized, a neutral weight is added to every clause. This weight of 1.0 equates to a non-weighted Tsetlin Machine if left alone. Then as the Tsetlin Machine Game completes and feedback is applied, these weights are updated. w_j^+ and w_j^- correlate to weights attributed to positive and negative clauses, respectively.

3.4.2 Classification

Because every clause is now given a separate weight, the simple summation equation 3.2 must be updated as well. The summation equation presented earlier is now changed to equation 3.15. In this equation, $s'(x)$ was originally the summation where the final tally defined whether or not an input matrix correlated to a given class, is now a true value generalization of the Tsetlin Machine. The last equation 3.17 is the change when Multi-Class Tsetlin is added with weights. The argmax is explained in section 3.1.3.

$$s'(x) = \sum_{j=1}^{n/2} w_j^+ C_j^+(X) - \sum_{j=1}^{n/2} w_j^- C_j^-(X) \tag{3.15}$$

$$\widehat{y} = u\left(\sum_{j=1}^{n/2} w_j^+ C_j^+(X) - \sum_{j=1}^{n/2} w_j^- C_j^-(X)\right) \tag{3.16}$$

$$\widehat{y} = \underset{i}{argmax}\left\{s_i'(X)\right\} \tag{3.17}$$

3.4.3 Weight Update

Similar to how each clause is updated through the feedback tables at the end of every Tsetlin Machine Game round, the newly introduced weights must also be updated. The weight update is controlled by the learning rate $\gamma \in [0, \infty)$

$$w_j^+ \leftarrow w_j^+ \cdot (1+\gamma),\ if\ C_j^+(X) = 1 \tag{3.18}$$

$$w_j^- \leftarrow w_j^- \cdot (1+\gamma),\ if\ C_j^-(X) = 1 \tag{3.19}$$

$$w_j^+ \leftarrow w_j^+ / (1+\gamma),\ if\ C_j^+(X) = 1 \tag{3.20}$$

$$w_j^- \leftarrow w_j^- / (1+\gamma),\ if\ C_j^-(X) = 1 \tag{3.21}$$

$$w_j^+ \leftarrow w_j^+\ if\ C_j^+(X) = 0 \tag{3.22}$$

$$w_j^- \leftarrow w_j^-\ if\ C_j^-(X) = 0 \tag{3.23}$$

Each of the feedback tables has been segmented above. Equation 3.18 - 3.19 is used whenever Feedback Type I is called. Respectively Equations 3.20 and 3.21 are both called upon Feedback Type II. Finally, as explained in section 3.1.4, the final feedback type is Ignore, upon which the weights won't be changed. This, again, always happens on true negative outputs.

3.5 Layered Tsetlin Machine

The final adaptation of the Tsetlin Machine to be examined in this chapter is the Layered Tsetlin Machine. No singular publication has been made on it yet, but it was initially theorized in the revision of the paper by Prof. Granmo in 2018[1][18].
The Layered Tsetlin Machine has its inspiration from Convolutional Neural Networks. With Convolutional Neural Networks each kernel is convolved with the input vector so that each kernel produces a feature map. A feature map describes where certain features are in the input image. A future map can be a cat ear or a dog's nose. Usually, in the first layer, these features appear as simple edges. These feature maps are then usually downsampled using a pooling operator to reduce the input vector to the next layer. The output of the pooling operator is then used in the next layer. This new layer will analyze features that appear in the image and combine them. This layering can be done multiple times so that the feature map can be complex shapes. If multiple complex shapes are triggered, say human eyes and human teeth, a classification will favor it therefore, classifying it as a human in this case.[19]

3.5.1 Layered Overview

As shown in figure 3.6, the Layered Tsetlin Machine is built up in multiple layers, where each layer uses the feature map of the previous layer, except the first, which uses the original input vector. Like Convolutional Neural Networks, the features of the image are sampled using Convolutional Tsetlin Machine and generates a feature map. This next layer only needs to be a normal Tsetlin Machine. This Tsetlin Machine will then evaluate the images based on a combination of its features.

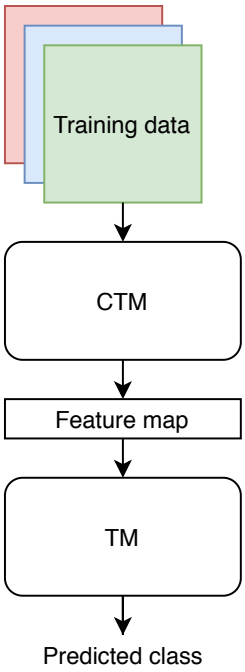


Figure 3.6: Layered Multi-Class Tsetlin

3.5.2 Feature map

The feature map is, in this case, what every clause evaluates to, with each image after the Convolutional Tsetlin Machine is finished training. The feature map for each training image can be downsampled using a pooling operator to reduce the input vector size used in the next layer. The feature map generation of a single image can be seen in figure 3.7. Here the Tsetlin Machine has finished its training on the training dataset. The clauses are then again exposed once to the training dataset, but this time the feature map is generated by storing what each of the clauses evaluates to and pooled.

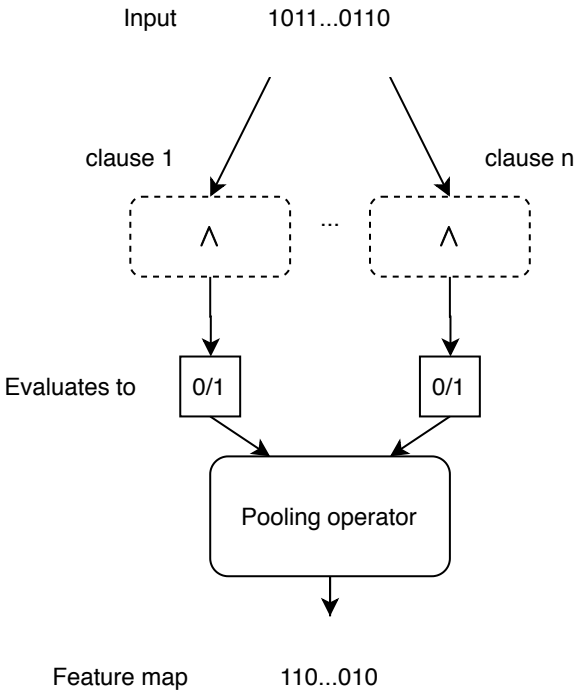


Figure 3.7: Feature map generation for a single image

3.5.3 Conclusion

Throughout this Chapter, an introductory description to the Tsetlin Machine and multiple of its adaptions was provided. The original Tsetlin Machine was introduced in depth together with the revision, Multi-Class Tsetlin Machine. The newer adaptions which are all used further in this thesis were also presented and explained along with the variations they bring. A fundamental understanding of these adaptions is crucial in understanding this thesis's anthesis.

This Chapter concludes Goal 2 of the thesis, which was to provide a thorough understanding of how the Tsetlin Machine works and how the adaptations differ and can improve the overall classification capabilities. A theoretical understanding was provided through this Chapter, and a technical implementation will be described further in the following Chapter.

Part IV

Environment for Tsetlin Machine-based color image classification

Chapter 4

Environment

Throughout this chapter, a description of the proposed environment in which both the hypotheses are tested and therefore making an environment in which the Tsetlin Machine and its adaptations utilized in the thesis can provide color image classification.

4.1 Proposed Architecture

This section describes how the experiment's environment is set up to analyze the CIFAR-10 dataset with various Tsetlin Machine architectures using various color schemes and binarization techniques.
Figure 4.1 gives a general overview of processes and information flow. The Architecture of the testing is set up in 4 parts: Image processing, logging, Machine learning, and log viewing.

- Image prepossessing handles downloading of the dataset and the various techniques to process images.
- Tsetlin Machine takes the binarized version of the color images for classification
- The output of the Tsetlin Machine is stored for future analysis
- Log viewer is used to analyze the logs produced by the logger.

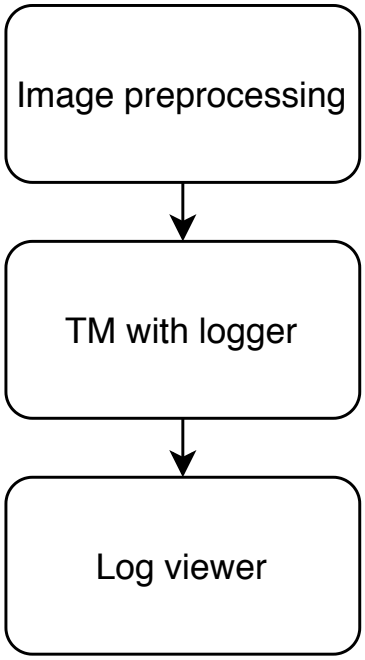


Figure 4.1: Architecture overview

4.2 Image pre-processing

The proposed pre-processing done to the images so they can be read with a Tsetlin Machine is done as described in figure 4.2.

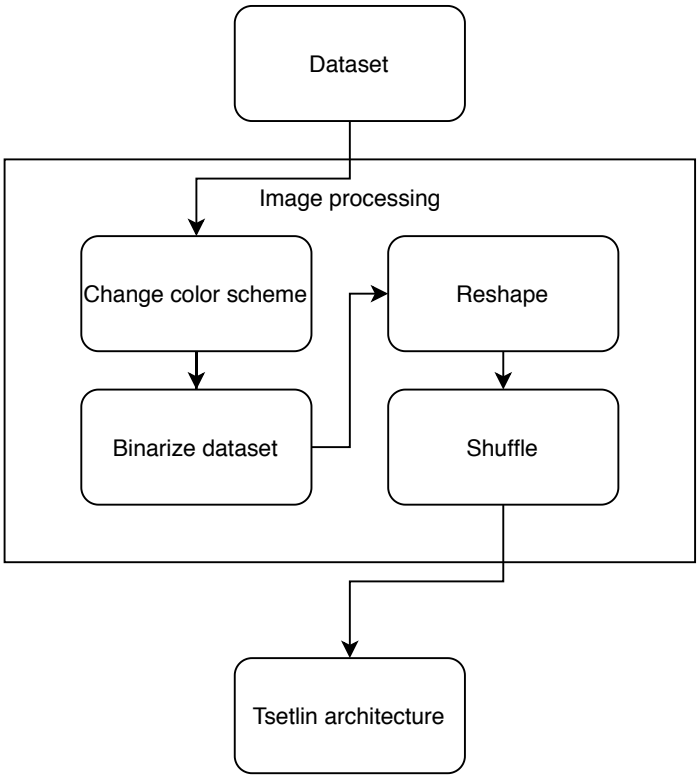


Figure 4.2: Image pre-processing architecture

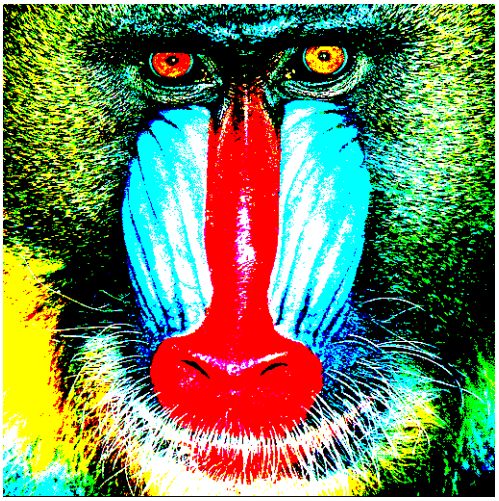
Change color scheme

The color scheme can be changed with an argument. The pictures are usually in RGB but can be changed to both HSV and HSL, based on which is being tested.

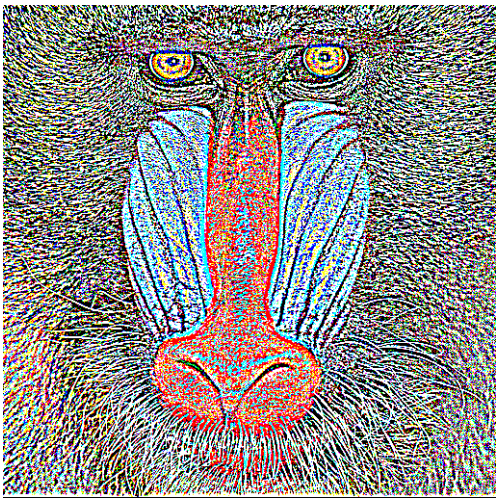
Binarize dataset

To binarize the dataset, each picture is split up in its channels. This is to keep as much data as possible throughout the binarisation process. Every channel is then binarized using a static global threshold, Otsu's method, adaptive threshold, Gaussian threshold, or Canny. This results in images that can contain $2^3 = 8$ colors. The colors are each of the primary and secondary colors in the RGB color model and black and white. Each channel, however, needs to be in boolean form. Some examples of how these binarization methods look like are shown in figure 4.3. Here the boolean values for True are set to 255 and false to 0 to make them perceivable to a human eye. One can see that Otsu's global threshold resembles the original. Adaptive Gaussian thresholding is very good at keeping texture, even in shadows, and canny is very good at detecting the edges. Figure 2.4 in section 2.3 also shows how they are perceived after being binarized.

(a) Original RGB



(b) Otsu's method



(c) Adaptive gaussian tresholding

(d) Canny

Figure 4.3: Binarization of 3 layers compared

The channels are combined again after the binarization but are done in two ways depending on the Tsetlin architecture. If it is a Convolutional Tsetlin Machine, then the channels can be combined to the shape of [row][column][color] like it is done for all of the images in figure 4.3.

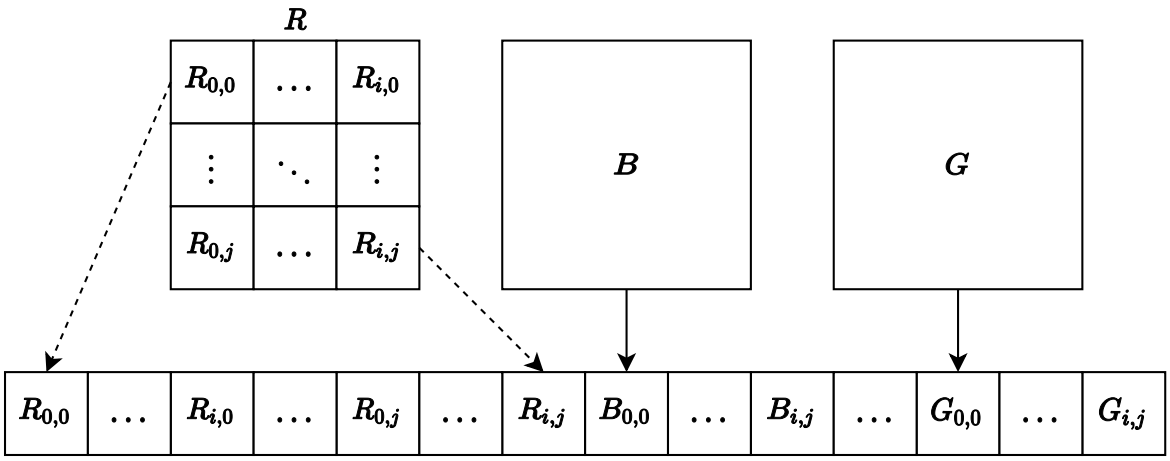


Figure 4.4: 3 channel matrices to 1d array

However, if it is a standard Tsetlin Machine architecture used, the binarized channels are stacked together in a single 1d array. This stacking algorithm is shown in figure 4.4. Here each channel is appended to the array, one row at the time.

Shuffle

Say that "cat" represents the positive class, and "not a cat" represents the negative, and there are 5000 images of each of them. In the first batch of 5000 images, the Tsetlin Machine will optimize to just recognize the "not cat" images, as it only trains on "not cat" images. In the second batch with just cat images, the Tsetlin Machine will optimize for cat images heavily. There are, in fact, so many images of cats in a row that the Tsetlin Machine over-trains, and will just predict cats for all images. This also shows that the first images in the training data set are less important than the last. To prevent this behavior, the training dataset is shuffled so that the classes don't come sorted out in the finalized binarized dataset.

Serialization

The last step in image processing is serialization. The images are stored to an object so that the same dataset doesn't need to be recreated every time it's going to be used.

4.2.1 Tsetlin Machine architecture

Tsetlin Machine

The implementation of the standard Tsetlin machine. The arguments passed in is the following:

Arguments

- Dataset - Choose dataset based on string
- Clauses - Defines how many clauses should be used
- T - Defines the threshold value
- s - defines s, used in the feedback tables

First, the binarized dataset from image pre-processing is de-serialized. For this implementation, it is important that the Tsetlin mode 1 is selected as an argument while creating the dataset. The Multi-Class Tsetlin Machine object and the logger object is created with the arguments. The Tsetlin machine runs for 400 epochs, where each epoch is recorded in the logger by predicting the whole dataset and storing the confusion matrix in the logfile.

Convolutional Tsetlin Machine

Similar to the standard Tsetlin machine, but with an extra argument for the mask. The image processing part needs to have Convolutional Tsetlin Machine specified to make the dataset work on this type of Tsetlin machine.

Layered Tsetlin Machine

The layered Tsetlin machine is set up in two parts. The first layer is a Convolutional Tsetlin Machine and set up as above with a few exceptions. It does not log the prediction, and in the end, it serializes the trained clauses and stores them.
The second layer is set up as a standard Tsetlin Machine, but is weighted and has to append_negated set to false in initialization. The labels are loaded from the original binarized dataset, and the serialized objects from layer 1 are loaded. The Tsetlin machine then runs as described above as a normal Tsetlin Machine with a logger.

4.2.2 Log viewer

The log viewer takes in log files and is used to analyze them. Here one can look at the graph of the accuracy a log. It is also implemented so one can read the average accuracy of the last 100 epochs and the top accuracy epoch of each log. For pairwise class accuracy, the log viewer produces two matrices. One that shows the average accuracy over the last 100 epochs for all 45 pairs, and one for the top accuracy for all pairs. The average of all pairs is also calculated here.

Part V

Experimentation Environment

Chapter 5

Experiments

The previous section (4.1) provided a theoretical description of the environment and capabilities used for this thesis. This section focuses on explaining how the two main hypotheses are tested in two separate experimentation rounds. The initial experiment focuses on prooving that the Tsetlin Machine can achieve color image classification, and then provide a search for the optimal parameters with which the second main experiment utilizes. The Second experiment then compares the various Tsetlin Machine adaptions.

5.1 Experiment 1

The initial experimentation correlates with the initial hypothesis of this thesis. Namely, whether or not the Tsetlin Machine can achieve color image classification with the various binarization techniques. The secondary task, a search for optimal parameters with which the next round of experimentation utilizes is also required.

Through this experiment, we examine the various combination of color schemes presented in section 2.2 and binarization techniques in section 2.3. For every combination of color scheme and binarization technique available in this thesis, an experimental sequential parameter search is performed. The starting parameters are those used to classify the MNIST dataset[17] with a slight increase in epochs.

Epochs	Clauses	T	s
400	Static	Adaptive	S

Table 5.1: Parameter Search initialisation variables

For every variance of pre-processed data, the sequential test begins with mask variance and work itself down table 5.1 . The overall result with the combination of color scheme, binarization and parameters will then be presented in the next section of this chapter.

Mask Variance	Clause Variance	T Variance	s Variance
32-30-28-...-10-8-6	500-1000-2000-4000-8000	25-75-100-200	3-5-10-20-40

Table 5.2: Parameter Search Sequence

5.2 Experiment 2

The second part of this thesis and experiment examines the capabilities of the various Tsetlin Machine adaptations when faced with the classification of color images. The color scheme and binarization technique used for this test are RGB Gaussian. Based on the results of the previous experiment, the variables presented in table 5.3 are those assumed optimal to use for this experiment. While near identical, there are a few variations that will be explained below. The overall results are presented within the next section (5.4).

Adaptation	Clauses	T	s	Epochs	Mask
Standard Tsetlin Machine	4000	75	10	400	-
Weighted Tsetlin Machine	4000	75*100	10	400	-
Convolutional Tsetlin Machine	4000	75	10	400	8
Weighted Convolutional Tsetlin Machine	4000	75*100	10	400	8
Layered (1) Tsetlin Machine	4000	75	10	200	8
Layered (2) Tsetlin Machine	4000	75*100	10	250	-

Table 5.3: Parameters for class pair classification

Multi-Class Tsetlin Machine

The Multi-Class Tsetlin Machine is not listed in the table above. However, it follows the exact same variables as the Convolutional Tsetlin Machine. Through this test, all the classes are the CIFAR10 dataset that will be used and is the only test in which the entire dataset is used.

Tsetlin Machine

As shown above, the Standard Tsetlin Machine utilizes the 4000, 72, 10 , 400 parameters which were the believed optimal set of parameters for this experiment, the main difference between the setup for the Tsetlin Machine and the optimal parameters from the previous example is the removal of the mask. This allows the Tsetlin Machine to classify the entire image as described in section 3 instead of by patches, as explained in section 3.3.

Weighted Tsetlin Machine

The variance between the standard parameters and the weighted is solely the T variable, which is $T \times 100$. This essentially gives every clause added weight and reduces the chance of large changes in clauses per epoch.

Convolutional Tsetlin Machine

Convolutional Tsetlin Machine was the adaptation used for the previous experiment. The parameters used are those assumed optimal for this test based on the previous results. The main

Layered Tsetlin Machine

The Layered Tsetlin Machine has the largest variance in parameters compared to the other adaptations. As explained in section 3.5. Every class comparison consists of two separate Tsetlin Machine runs. The initial run will provide a feature map that the second layer will utilize again for classification. The first layer consits of the same parameters as given to the Convolutional Tsetlin Machine, with the exception to the epochs, which is now set to 200. The second layer will introduce weights but no longer use a mask as it is based upon the feature maps of the previous layer.

5.3 Parameter Search

Throughout this section, we will reveal the results of the experiments presented above. These results have been divided into two main subsections, those focusing on the Parameter search for the initial setup and the Tsetlin Machine Architecture comparison. This section will focus mainly on the Parameter Search Experiment, and the concluded optimal parameters will then be used further on in the Tsetlin Machine comparison.

5.3.1 Empirical Results

The following are the results of the Parameter Search experiment with different color schemes and binarization techniques. The results will be shown in the order of RGB, HLS, HSV color schemes, and, finally, an overview of the top-performing parameters.

RGB

The first results are the RGB image binarization parameter search. This was the group that performed most optimally.

RGB	Mask	Clauses	T	s	acc last 100
Static	8	8000	50	10	94.33
Otsu	10	4000	75	10	92.88
Adaptive	8	4000	75	10	95.20
Gaussian	8	4000	75	10	95.32
Canny	12	2000	75	3	92.50

Table 5.4: Table with the results of the various binarization methods together with RGB color scheme

Table 5.4 shows that RGB Gaussian performs slightly better than the RGB Adaptive binarization. Both achieved optimal results with the same parameters, while Canny and static both perform better with an increase in Clauses and reduced T.

HSL

HSL	Mask	Clauses	T	s	acc last 100
Static	30	2000	50	10	87.27
Otsu	10	2000	50	10	92.95
Adaptive	8	4000	75	5	94.67
Gaussian	8	2000	50	5	94.50
Canny	30	2000	50	3	88.10

Table 5.5: Table with the results of the various binarization methods together with HSL color scheme

The HSL parameter search showed a slight increase in accuracy for the adaptive binarization. In this experiment, adaptive performed optimally with the same parameters as in 5.4. However, while Gaussian performed slightly worse than adaptive, it performed with a significant change in parameters. The remaining binarization techniques performed optimally with a variety of parameter changes compared to RGB.

HSV

HSV	Mask	Clauses	T	s	acc last 100
Static	28	2000	75	10	88.73
Otsu	6	2000	50	10	92.16
Adaptive	8	4000	50	10	94.38
Gaussian	6	2000	50	10	94.51
Canny	30	2000	50	10	88.04

Table 5.6: Table with the results of the various binarization methods together with HSV color scheme

In this image scheme the HSV Gaussian performed best, while only slightly above Adaptive in accuracy. Both have slight variance in parameters. The remaining binarization techniques all performed less optimally with a larger variance of optimal parameters per technique.

5.3.2 Conclusion

The results from the initial experiment of the color schemes and binarization techniques show an exciting tendency between the variants. Because these results lay the baseline for the next section in which we compare the different adaptions of the Tsetlin Machine. It seems like the best color scheme is RGB, as both HSL and HSV fall below it overall. Static, Otsu and Canny are subpar compared to Adaptive and Gaussian, while it is up to chance whenever Gaussian or Adaptive is the best binarization algorithm with only a 0.12% difference in the RGB category. This is further stated as Adaptive is the better algorithm in the HSL color scheme. Canny seems to be the worst algorithm overall. This shows that the Convolutional Tsetlin Machine classifies better with the texture of objects, as the algorithm only keeps the edges of objects. Static does it surprisingly well in the RGB category. The threshold value was probably lucky in this case, but it shows that the Convolutional Tsetlin Machine can analyze color pictures that are binarized using only a global threshold if the threshold is set correctly.

5.4 Tsetlin Machine Architecture Comparison

Throughout this section, the results of the Tsetlin Machine Architecture comparison will be shown. As explained in section 5, the parameters used in this section are based on the results from the previous experiment. With the exceptions as explained in section 5.

Color Scheme	Binarization	Clauses	T	s	Mask
RGB	Gaussian	4000	x	10	x

Table 5.7: Experiment 2 Parameters

5.4.1 Multi-Class Tsetlin Machine

The first experiment for this section is the Multi-Class Tsetlin Machine. As explained in section 5, this experiment tries to classify every class in the CIFAR10 dataset. The graph below shows the accuracy curve over the 400 epochs of training, the average accuracy over the last 100 epochs is also calculated.

Average Accuracy: 60.74%

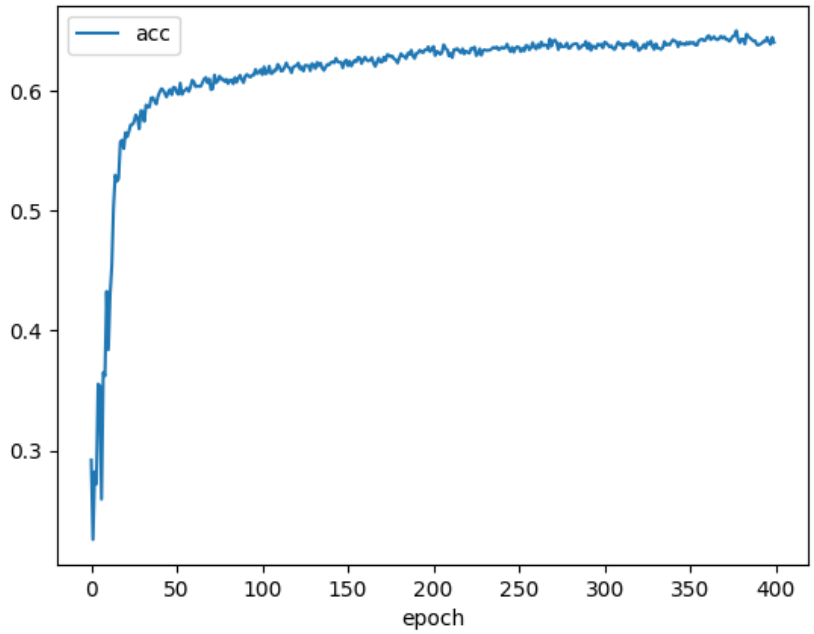


Figure 5.1: All classes with multiclass CTM

5.4.2 Standard Tsetlin Machine

The second round of testing is the Tsetlin Machine. Following the parameters specified in section 5 with the exception of Mask size as standard Tsetlin Machine runs on the entire image set per Tsetlin Machine Game round.

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship
automobile	88.16	–	–	–	–	–	–	–	–
bird	82.46	90.78	–	–	–	–	–	–	–
cat	87.79	87.19	68.32	–	–	–	–	–	–
deer	88.11	90.88	66.75	78.15	–	–	–	–	–
dog	91.04	89.26	68.61	61.82	81.35	–	–	–	–
frog	91.06	86.34	74.49	78.55	81.72	84.04	–	–	–
horse	90.12	92.48	74.77	81.24	78.81	79.17	88.41	–	–
ship	76.42	85.72	85.75	91.21	91.60	93.12	93.27	92.98	–
truck	85.30	69.95	88.31	85.77	90.43	89.85	89.66	88.54	86.10

Table 5.8: TM Result Matrix average last 100

The table above shows the Results for the Tsetlin Machine with the average over the last 100 epochs. The overarching average for this architecture was:
Average Accuracy: 84.13%.

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship
automobile	89.45	–	–	–	–	–	–	–	–
bird	84.45	92.75	–	–	–	–	–	–	–
cat	89.15	91.60	70.70	–	–	–	–	–	–
deer	89.65	92.85	69.80	80.65	–	–	–	–	–
dog	92.45	94.55	72.75	65.20	84.10	–	–	–	–
frog	92.40	92.35	77.05	81.15	84.40	86.25	–	–	–
horse	91.40	94.45	81.40	83.10	81.25	81.00	90.25	–	–
ship	80.70	87.30	90.20	92.50	93.00	94.24	94.10	94.25	–
truck	87.70	73.30	90.05	87.05	91.10	91.10	91.15	89.75	87.7

Table 5.9: TM Peak Result Matrix

The secondary table shows the average for the top 100 peaks. The overall peak average accuracy for the Tsetlin Machine was:
Average Peak Accuracy: 86.49%.

The following are the graph showing the accuracy of the Tsetlin Machine when classifying two classes. Figure 5.2 shows the graph depicting the accuracy over the 400 epochs for the two classes, which had the highest accuracy. Figure 5.3 shows the accuracy graph over 400 epochs for the two classes with the lowest accuracy. The variance between the accuracy of the two graphs equates to ≈ 31 percentage points with 3-5 barely achieving above random classification with only 61.82

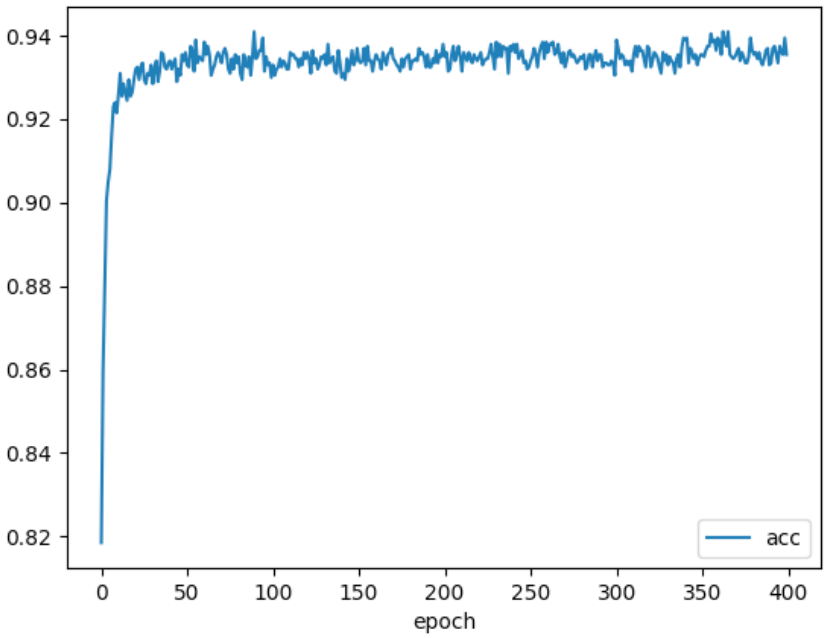


Figure 5.2: Tsetlin Machine Accuracy 6-8

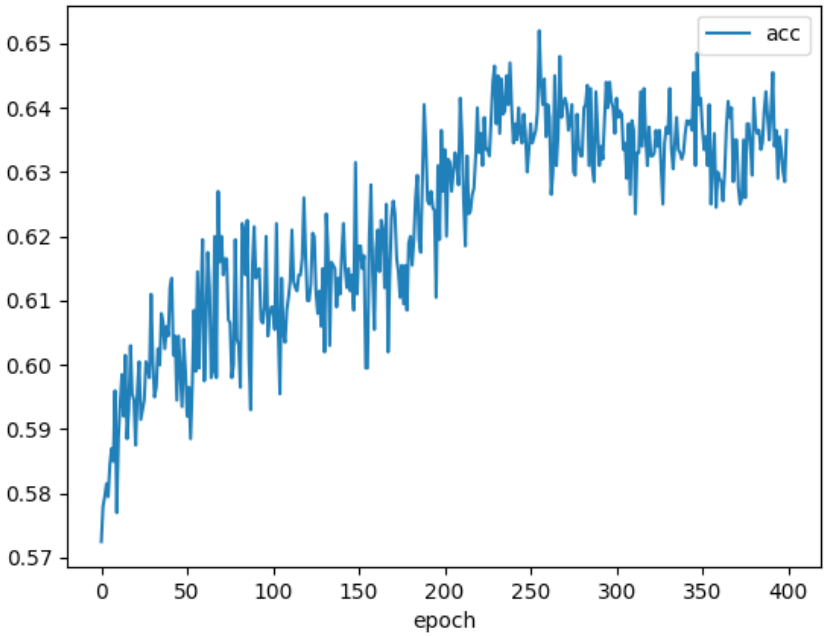


Figure 5.3: Tsetlin Machine Accuracy 3-5

5.4.3 Weighted Tsetlin Machine

The third round of testing was for the Weighted Tsetlin Machine; how the addition of the weights was done through the parameters is explained in section 5. The parameters used for these experiments were as follows:

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship
automobile	69.34	–	–	–	–	–	–	–	–
bird	63.13	69.73	–	–	–	–	–	–	–
cat	65.21	62.39	60.78	–	–	–	–	–	–
deer	65.80	62.91	63.03	67.74	–	–	–	–	–
dog	66.28	64.33	61.33	59.85	65.80	–	–	–	–
frog	65.56	61.76	63.65	66.98	71.79	72.28	–	–	–
horse	66.38	72.98	63.76	67.18	67.21	67.72	70.23	–	–
ship	64.52	76.20	67.62	65.38	66.25	67.98	66.21	67.32	–
truck	69.43	65.63	72.95	70.09	71.50	73.67	68.11	72.94	68.66

Table 5.10: WTM Result Matrix average last 100

With the parameters from section 5, the following table above (5.10) shows the average accuracy per classification of two different classes over the last 100 epochs. The total overarching average accuracy was: **Average Accuracy:** 67.10%
Table 5.11 shows the peak average accuracy achieved from the Weighted Tsetlin Machine architecture. The overarching average peak accuracy over the top 100 peaks was as follows: **Average Peak Accuracy:** 69.61%

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship
automobile	71.60	–	–	–	–	–	–	–	–
bird	66.00	77.40	–	–	–	–	–	–	–
cat	69.65	67.60	61.45	–	–	–	–	–	–
deer	71.20	69.30	63.70	68.55	–	–	–	–	–
dog	72.30	70.25	61.80	60.90	66.45	–	–	–	–
frog	72.50	65.90	64.35	67.90	72.50	72.95	–	–	–
horse	70.85	76.15	65.15	67.70	68.50	68.35	70.7	–	–
ship	65.05	77.25	71.05	67.25	68.95	70.35	67.8	69.35	–
truck	72.05	66.15	78.70	71.10	73.70	74.95	69.8	76.50	70.65

Table 5.11: WTM Peak Result Matrix

The following two graphs depict the accuracy over 400 epochs. The chosen graphs depict both the classes in which the Weighted Tsetlin Machine achieved the highest accuracy and the classes in which the accuracy was most poor. The overall difference in accuracy between them is ≈ 16 percentage points with classes 3-5 barely achieving above random selection (50%).

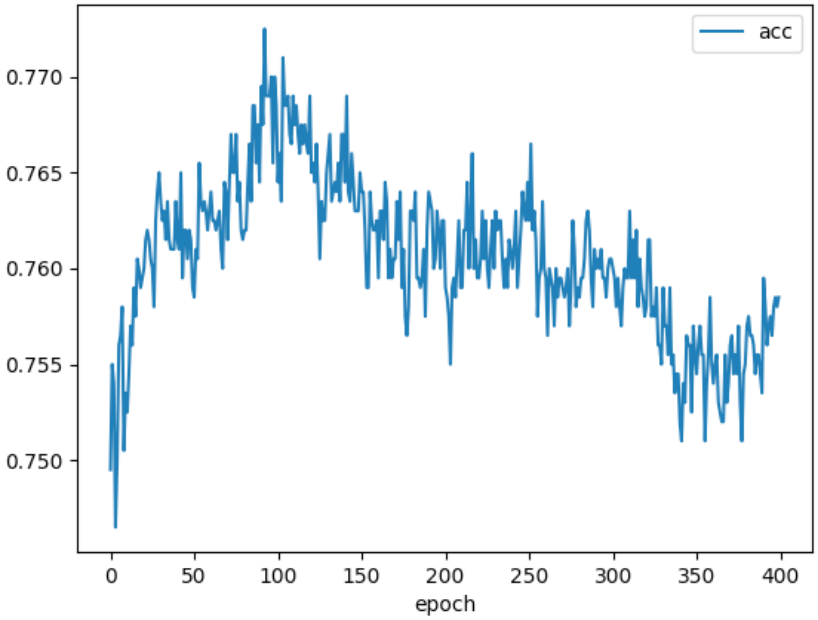


Figure 5.4: Weighted Tsetlin Machine Accuracy 1-8

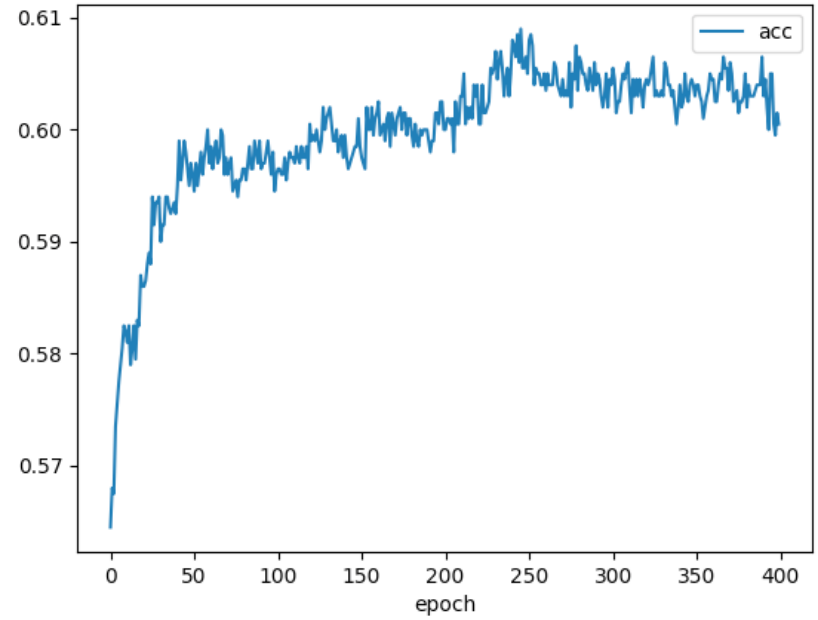


Figure 5.5: Weighted Tsetlin Machine Accuracy 3-5

5.4.4 Convolutional Tsetlin Machine

The Convolutional Tsetlin Machine was run by the parameters presented in section 5. Following the same presentation as Weighted and Standard Tsetlin Machine. The initial table presented below is the average of the last 100 epochs.
Average Accuracy: 91.66%

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship
automobile	94.65	–	–	–	–	–	–	–	–
bird	88.25	95.54	–	–	–	–	–	–	–
cat	92.50	95.19	81.95	–	–	–	–	–	–
deer	93.20	96.28	82.25	85.67	–	–	–	–	–
dog	93.92	96.44	83.34	73.43	87.74	–	–	–	–
frog	95.04	96.65	86.59	86.57	89.53	90.65	–	–	–
horse	94.51	97.39	87.93	87.96	86.39	87.67	94.27	–	–
ship	88.16	94.44	93.62	94.39	95.61	96.06	96.56	96.40	–
truck	93.30	89.03	94.90	94.50	95.48	95.24	96.17	95.09	93.57

Table 5.12: CTM Result Matrix average last 100

The second table is the average peak accuracy over the top 100 peak accuracy points. The overarching average for this table is as follows:
Average Peak Accuracy: 92.56%

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship
automobile	95.35	–	–	–	–	–	–	–	–
bird	89.50	96.25	–	–	–	–	–	–	–
cat	93.50	95.95	83.65	–	–	–	–	–	–
deer	94.10	96.85	83.80	86.90	–	–	–	–	–
dog	94.55	97.10	84.90	74.95	88.60	–	–	–	–
frog	95.95	97.10	87.95	87.80	90.55	91.55	–	–	–
horse	95.20	97.90	89.20	88.85	87.65	89.05	94.95	–	–
ship	89.60	95.15	94.25	95.15	96.20	96.70	97.05	97.05	–
truck	94.15	90.05	95.70	95.20	96.35	96.00	96.95	95.80	94.30

Table 5.13: CTM Peak Result Matrix

The two following graphs show the accuracy over 400 epochs, from start to finish. These graphs show the two classes presented with the best classification accuracy and those that presented with the worst accuracy. The difference between the averages is $\approx$ 23 percentage points.

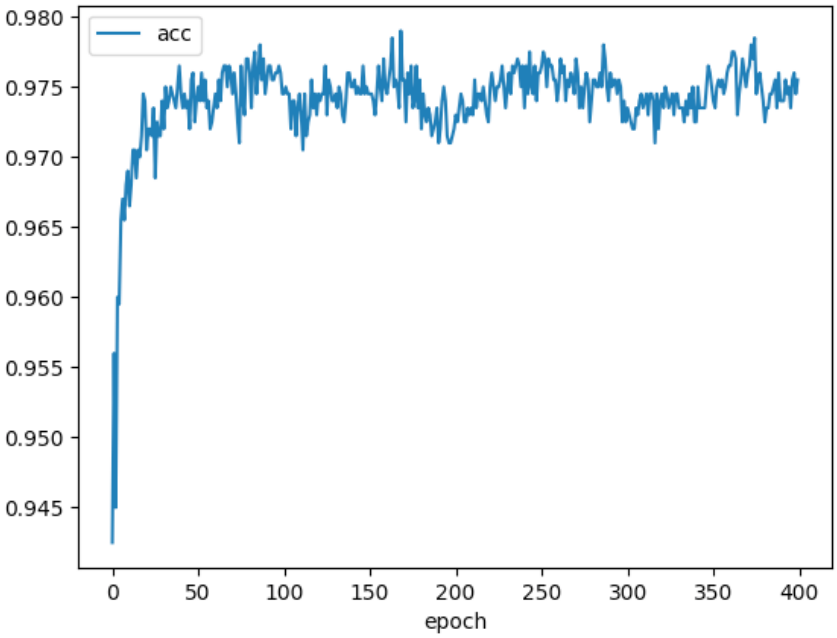


Figure 5.6: Convolutional Tsetlin Machine Accuracy 1-7

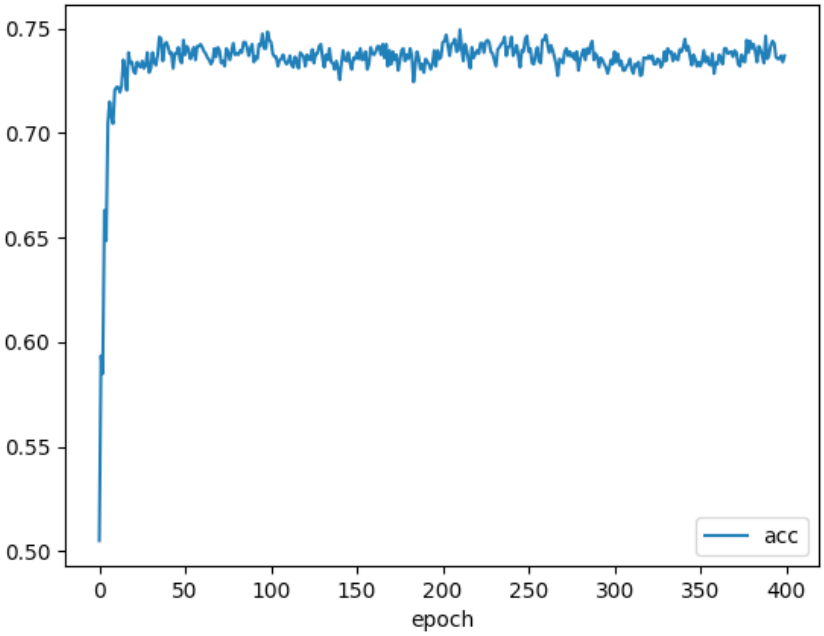


Figure 5.7: Convolutional Tsetlin Machine Accuracy 3-5

5.4.5 Weighted Convolutional Tsetlin Machine

Through this section is the results from the Weighted Convolutional Tsetlin Machine. These follow the same parameter deviations as the Weighted Tsetlin Machine vs the Standard Tsetlin Machine. These were again explained in section 5.
The initial table contains the average over the last 100 epochs. From these results, we can calculate the average accuracy between the classification pairs as:
Average Accuracy: 84.69%

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship
automobile	74.56	–	–	–	–	–	–	–	–
bird	76.00	95.78	–	–	–	–	–	–	–
cat	80.34	95.19	81.91	–	–	–	–	–	–
deer	77.25	96.44	82.59	86.26	–	–	–	–	–
dog	85.37	96.13	83.71	73.34	77.57	–	–	–	–
frog	78.09	96.46	87.05	87.01	76.86	77.57	–	–	–
horse	81.65	97.10	88.97	87.82	71.75	75.27	81.27	–	–
ship	74.17	94.13	93.73	94.75	81.99	88.33	79.30	85.31	–
truck	73.74	89.36	95.09	94.45	84.63	88.02	84.54	85.47	74.23

Table 5.14: WCTM Result Matrix average last 100

The secondary table contains the peak accuracy over the top 100 peaks of the Weighted Convolutional Tsetlin Machine. Via these results, the average peak accuracy over the classification pairs is:
Average Peak Accuracy: 86.03%

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship
automobile	76.65	–	–	–	–	–	–	–	–
bird	78.05	96.30	–	–	–	–	–	–	–
cat	84.05	95.70	83.45	–	–	–	–	–	–
deer	79.65	96.90	84.00	87.40	–	–	–	–	–
dog	87.20	96.50	85.10	74.80	78.10	–	–	–	–
frog	80.85	96.95	88.00	88.20	78.10	79.35	–	–	–
horse	84.20	97.45	90.15	88.80	72.20	76.25	83.15	–	–
ship	75.00	94.75	94.50	95.50	84.00	90.35	81.25	86.30	–
truck	77.00	90.35	95.85	95.20	85.10	89.30	85.30	86.35	77.80

Table 5.15: WCTM Peak Result Matrix

These two graphs show the accuracy of the classification pairs, which achieved the highest and lowest accuracy, respectively, through the Weighted Convolutional Tsetlin Machine Classification. As shown, there is a large difference in accuracy between them, $\approx$ 25 percentage points.

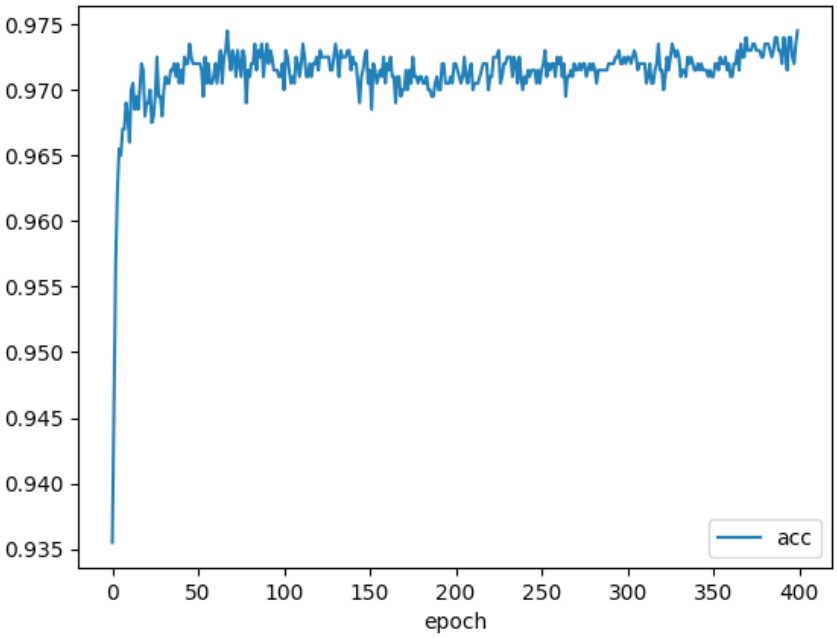


Figure 5.8: Weighted Convolutional Tsetlin Machine Accuracy 1-7

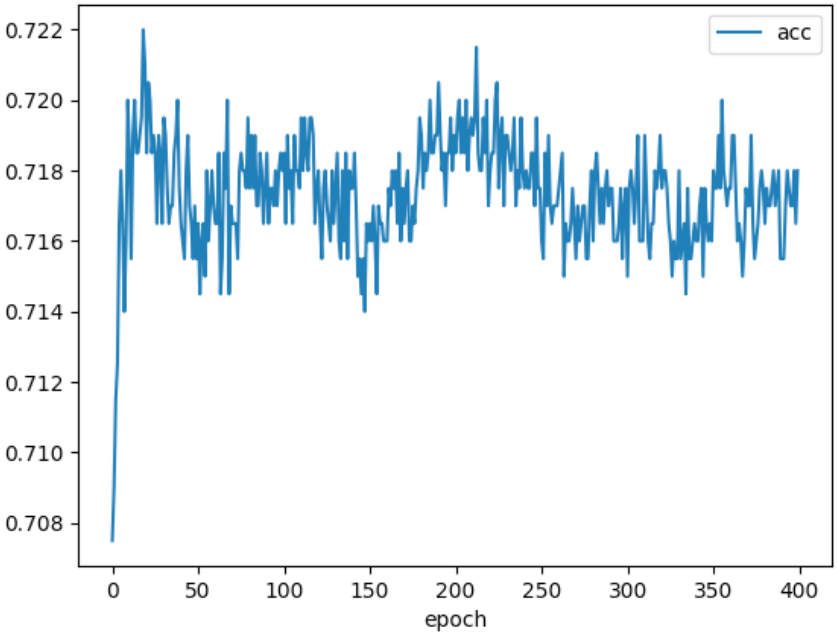


Figure 5.9: Weighted Convolutional Tsetlin Machine Accuracy 4-7

5.4.6 Layered Tsetlin Machine

The layered Tsetlin Machine consists of running the Tsetlin machine over multiple layers, as explained in section 3.5. However, this section will only show the results from the second layer. With some slight changes in parameters explained in section 5.
The first table (5.16) shows the average accuracy over the last 100 epochs while the second table (5.17) shows the average over the top 100 peaks of the 400 epochs. Through these two tables, we calculate the overarching accuracy for the classification pairs in the table.
Average Accuracy: 90.77% **Average Peak Accuracy:** 91.41%

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
automobile	94.39	–	–	–	–	–	–	–	–	
bird	84.81	95.84	–	–	–	–	–	–	–	
cat	92.77	95.26	80.76	–	–	–	–	–	–	
deer	93.25	95.76	80.78	84.98	–	–	–	–	–	
dog	95.11	96.22	81.67	73.46	87.33	–	–	–	–	
frog	94.91	91.55	86.51	86.49	88.11	89.83	–	–	–	
horse	94.73	97.01	86.95	87.51	86.56	87.67	94.69	–	–	
ship	80.08	90.36	93.46	94.48	95.65	96.16	96.53	96.42	–	
truck	92.73	80.47	94.93	93.82	95.37	94.96	95.77	94.46	94.18	

Table 5.16: Layered Average

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
automobile	94.80	–	–	–	–	–	–	–	–	
bird	86.45	96.05	–	–	–	–	–	–	–	
cat	93.20	95.50	81.70	–	–	–	–	–	–	
deer	93.80	96.00	81.65	85.85	–	–	–	–	–	
dog	95.55	96.50	82.25	75.35	88.05	–	–	–	–	
frog	95.25	92.85	87.60	87.00	89.15	90.60	–	–	–	
horse	94.95	97.25	87.50	87.95	87.30	88.45	95.20	–	–	
ship	82.65	91.15	93.75	94.75	95.85	96.35	96.75	96.75	–	
truck	93.40	82.50	95.25	94.25	95.65	95.25	96.10	94.80	94.55	

Table 5.17: Layered Peak

Graphs 5.10 and 5.11 show both the top and bottom accuracy pairs. There is a noticeable difference and the overall accuracy variation results in ≈ 21 percentage points for both variants.

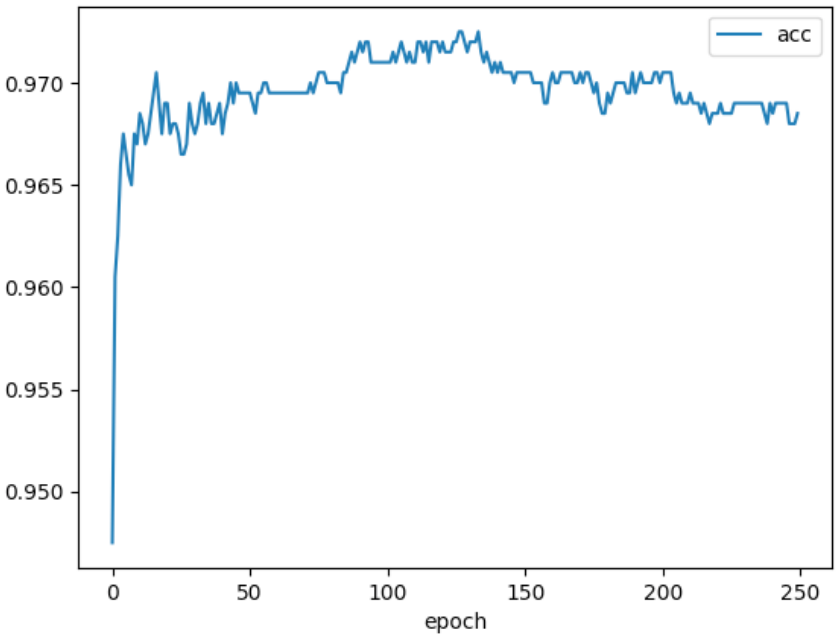


Figure 5.10: Layered Tsetlin Machine Accuracy 1-7

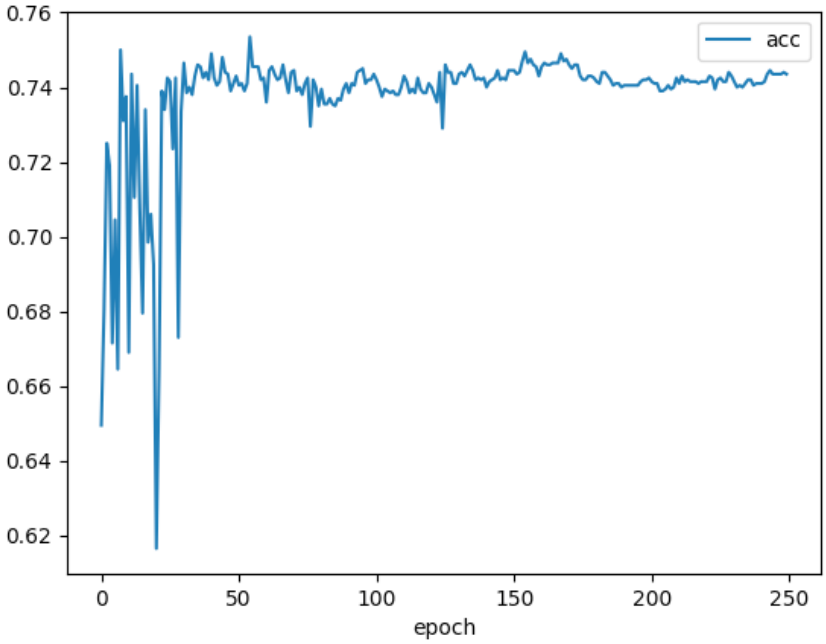


Figure 5.11: Layered Tsetlin Machine Accuracy 3-5

5.4.7 Conclusion

The results from the second experiment round of the Tsetlin Machine adaptation comparison show an interesting tendency. From the previous experiment, we learned the parameters we believed would be optimal for the architecture comparison. In the table below (5.18) are the various statistics from these results. The first round of the second experiment focused on the Multi-Class Tsetlin Machine, which was every class. While low, this simple implementation of the Multi-Class Tsetlin shows promise in its accuracy with 60.74%. The remaining results also show an interesting tendency in which the weighted variants of the Tsetlin Machine lose a large percentile of accuracy, dropping 15.03% accuracy between the standard Tsetlin Machine and the Weighted Tsetlin. Examining the graphs presented with each result, the overall accuracy of the weighted variants show very little overall learning over the 400 epochs. Thinking back to section 3.4 and 5 the simple implementation of weights by adding an increase in the T parameter could cause the clauses to update far too infrequently.

The two Tsetlin adaptations which produced the highest accuracy was Convolutional and Layered. Unsurprisingly, considering the parameter search experiment utilized the Convolutional Tsetlin Machine as its baseline. Layered interestingly produced a lower overall average accuracy by only 0.89%. With some parameter tweaking, it would not be surprising if the accuracy would rise above Convolutional Tsetlin.

Classification Algorithm	Average Accuracy	Peak Accuracy	Top Classes	Worst Classes
Multi-Class Tsetlin Machine	60.74%			
Tsetlin Machine	84.13%	86.49%	6-8	3-5
Weighted Tsetlin Machine	67.10%	69.61%	1-8	3-5
Convolutional Tsetlin Machine	91.66%	92.56%	1-7	3-5
Weighted Convolutional Tsetlin Machine	84.69%	86.03%	1-7	4-7
Layered Tsetlin Machine	90.77%	91.41%	1-7	3-5

Table 5.18: Overall Results from The second experimentation round.

Part VI

Contributions and Conclusion

Chapter 6

Conclusion

Through this thesis, both the hypotheses presented at the start, as well as the various goals linked together with these hypotheses, have been answered to various degrees. Following is the explanation of how we believe everything was achieved or examined.

Hypothesis 1:*Baseline Tsetlin Machine can provide color image classification without image augmentation or pre-training.*

Our experiments show that for pairwise color images, the only pre-processing done to them is binarization and restructuring the image array, which can, in fact, be classified by an untrained baseline Tsetlin machine.

Hypothesis 2:*Similar to how adaptations to the Neural Network model can provide higher accuracy for classification, so can the adaptations of the Tsetlin Machine.*

Comparing the results of the baseline Tsetlin Machine and the results of the various adaptations of the Tsetlin Machine, it is clear that certain adaptations, such as the Convolutional Tsetlin Machine provides higher accuracy in color image classification.

Goal 1:*Provide a fundamental understanding of how the binarization of images can contribute to increased classification accuracy*

Section 2.1 and 2.3 both go in-depth on how two older techniques used in image processing can be integrated into the Tsetlin Machine and color image classification.

Goal 2: *Provide a working understanding of the Tsetlin Machine Algorithm and the various adaptations*

Chapter 3 provides an in-depth understanding of the Tsetlin Machine, which in turn is based on the Learning Automatons described in Section 2.7. The chapter also goes in-depth on how the variations of the Tsetlin Machine differ while also providing an in-

troductory description.

Goal 3: *Provide an environment in which the Tsetlin Machine can classify color images*

Through Chapter 4, we go in-depth on how binarization and color schemes are utilized to provide an environment in which the Tsetlin Machine can provide color image classification.

Goal 4: *Implement the various adaptations into the environment for further testing*

As part of our experimentation, the various adaptations of the Tsetlin Machine was used to create a static parameter comparison. This was all done in the environment detailed in Chapter 4 and the varying implementation through the experiments was further shown in Section 5.

Goal 5:*Run multiple adaptions of the Tsetlin Machine with the intent of comparing empirical results.*

As part of the comparison experimentation, detailed in Chapter 5 and Sections 5.3 - 5.4. Multiple adaptations of the Tsetlin Machine was tested with the equivalent parameters.

Research Question:
The main focus of this thesis will be examined to what extent can the Tsetlin Machine classify color images and which state-of-the-art Tsetlin Machine architecture provides the highest accuracy?

Overall we believe in having provided an accurate environment in which the Tsetlin Machine can provide color image classification. In this environment, we also tested multiple adaptations of the Tsetlin Machine, and the results clearly show that the Tsetlin Machine is capable of color image classification. Even without the usage of image augmentation or pre-training. Based on the results from the experiments in this thesis. The Convolutional Tsetlin Machine provided the highest accuracy for color image classification. However with additional work or parameter tweaking, the other adaptations might achieve even higher accuracy. Especially considering Layered achieved accuracy very close to the Convolutional Tsetlin.

6.1 Discussion

6.1.1 Color and Binarization results

In this thesis, we have looked at different color schemes and binarization combinations applied to a dataset pair. These combinations are then analyzed by classifying with a Convolutional Tsetlin Machine. The experimental experiments show that using a binarization method that keeps both texture and shapes is most simple to classify for a Convolutional Tsetlin Machine. The best algorithms for this can be either Adaptive thresholding or Gaussian threshold, as the test with both techniques yielded about equally good results. The two global binarization methods (Static and Otsu) used in this thesis were about equally good, except for the RGB-static combination, which had an accuracy of 94.33%. This was higher than expected and could be just luck. For the comparison of the three color schemes, RGB does it better overall in almost all cases. The edge detecting algorithm Canny does it worse overall on all tests. This indicates that Convolutional Tsetlin Machine analyses are better with textures in the image. The combination of RGB color scheme and Gaussian adaptive thresholding wins by a small margin overall and is therefore chosen as the binarization method to use on all experiments after this.

6.1.2 Multi-Class

The Multi-Class Convolutional Tsetlin Machine experiment gives a somewhat disappointing result when all classes are classified at the same time using adaptive Gaussian binarization on the CIFAR-10 dataset. At only 60.74% average accuracy of the last 100 epochs. Compared to when only the automobile class and the cat class are analyzed with Convolutional Tsetlin Machine at 95.32% average accuracy of the last 100 epochs, the Multi-Class Convolutional Tsetlin Machine only achieves 60.74% average accuracy of the last 100 epochs.

The result isn't as bad as it seems, as classifying all classes as the same class in the automobile - cat pair gives a 50% accuracy while doing the same in the 10 classes example yield only 10% accuracy. They are in reality two very different datasets. The Multi-Class result is, however, not great.

This could be because instead of having 2000 clauses dedicated to 5000 automobile images and 2000 clauses dedicated to 5000 "not automobile" (cat) images, this algorithm dedicates the last-mentioned clauses on 45000 "not automobile" images divided into 9 classes instead. This makes it harder to identify images that are not automobiles, as each of the negative clauses now needs to both be more general at classifying "not automobile" and spread the 2000 clauses over 8 more classes compared to the pairwise experiment. This happens for all the negative clauses in the Multi-Class Convolutional Tsetlin Machine. As a result, half of all the clauses in the Multi-Class Convolutional Tsetlin Machine are

trained to identify images on a more general basis than in the pairwise test, where the negative clauses only needed to classify cat images.
Binary connect is a deep neural network, where the weights are binary. In its paper there are a CIFAR-10 test using no image augmentation, the network achieved an accuracy of 91.73%.[20] This is 30.99 percentage points above the result we achieved in this experiment.

6.1.3 Pairwise results

The best result from the pairwise test is from the Convolutional Tsetlin Machine. Interestingly the average of the peak epoch for each pair is 92.56%, which is very close to the average of all the pairs average accuracy for the last 100 epochs, which are at 91.66%. This shows that the algorithm is very stable. Predicatively, the algorithm has more trouble recognizing classes that looks more like each other than classes that differ. Cat and dog is an excellent example of this, reaching only 73.46% accuracy. Automobile and horse are the best pair, reaching 97.01% accuracy.
The rest of the pairwise implementations does not beat pairwise Convolutional Tsetlin Machine in overall accuracy. This could be because the parameters used are only optimized for the automobile - cat pair using Convolutional Tsetlin Machine.
The Standard Tsetlin architecture has an average accuracy of 84.13%. This was expected because standard Tsetlin analyzes the whole image using all channels with each clause instead of small parts like the Convolutional Tsetlin Machine. A wheel of a car must, for example, be in the same spot, with the same colors triggered in the binarization for it to be recognized as a car part, where the Convolutional Tsetlin Machine moves its kernel around to find it.
Layered Tsetlin Machine can probably beat out Convolutional Tsetlin Machine, as the first layer is a Convolutional Tsetlin Machine that uses the exact parameters as the winning Convolutional Tsetlin Machine. Doing a parameter search on the second layer would probably result in the layered having the best accuracy of all the pairwise implementations.
The weighted Tsetlin machine architecture has the worst accuracy of all the pairwise tests, reaching only 67.10% in the average accuracy measurement. This could be because of the parameters where set unfavorable for it, as a parameter search was never done for the standard Tsetlin Machine or weighted Tsetlin machine. It could also mean that adding weighing while analyzing the CIFAR-10 dataset worsen the result, which is doubtful.
The Weighted Convolutional Tsetlin machine gives the same tendencies as the weighted Tsetlin Machine, as its non-weighted counterpart does it better. With an average accuracy of 84.69%, it does it marginally better than the standard Tsetlin machine without weights.
The overall impression these results give is that all of the Tsetlin machine architectures that compare pairwise are stable and almost at peak accuracy for most of the last 100

epochs. Looking at the graphs confirms this, as most of them almost imminently flatten. This again means that when making the Layered Tsetlin Machine, the first layer is very close to peak efficiency when the feature map is extracted. The Convolutional Tsetlin Machine had better accuracy, but the Layered Tsetlin Machine can beat it.

6.1.4 The potential of Multi-Class

To test that the Multi-Class Convolutional Tsetlin Machine accuracy was indeed affected by how its algorithm was implemented, and not that it can't differentiate between some pairs, the pairwise Convolutional Tsetlin Machine experiment was executed. In theory, this average can indicate what a Multi-Class algorithm could potentially be close to achieving. This is, however, challenging.
One has to take into account that the pairwise dataset is easier to classify then the multi-class dataset, as it adds one layer of classification before beginning with machine learning as described in the multi-class discussion (section 6.1.2). All of the 45 pairs have data in them that they are of one or the other class, while it does not occur in the multi-class dataset. The clause division problem also stays in this version of the comparing problem, as it's really just an extension of one pair.
Trying to make a comparison is also hard, as taking an average of the pairwise provides problems one has when comparing only one pair to the multi-class. An algorithm that always classifies all images to one class yields 50% overall accuracy, while 10% accuracy for the multi-class. One way to try compare could be by mapping this with all values above 50% to all values above 10%, using $ACC_W(ACC) = 1.8ACC - 0.8$, 91.66% becomes 84.98%, which is still 24.24 percent points above the Multi-Class implementation.
This indicates that the Multi-Class algorithm has room for improvement.

6.2 Future Work

The final section of the thesis focuses on possible future revisions, adaptions, or additions to the work presented through this thesis.

6.2.1 Multilevel tresholding

In this thesis, Otsu binarization was used on each channel of an image, as a way of binarizing the image. It could be interesting to use a multilevel algorithm instead, and splitting the channel into more channels, as this would keep more data from the original image. An example of this is the multilevel version of the algorithm is described in the original paper for Otsu's method. Here it groups multiple classes within one grayscale image and sets multiple global thresholds [10].

6.2.2 Dither

Since the texture of the image seems to be important for classification, an exploration of different dithering techniques can be explored. Here a grayscale image is binarized, so it gives off the illusion to have grayscale.[21]

6.2.3 Filter bank

The Filter bank is a theoretical application of the binarization techniques used through this thesis. By utilizing multiple binarization techniques per image, then stacking these binarized images. One could potentially gain the features prominent in the output of all the binarization techniques. This would, however, increase the total pre-processing needed per image.

6.2.4 Non-sequential Parameter Search

The parameter search used in this thesis was used mainly as a pointer of whether the Tsetlin Machine would be able to provide acceptable accuracy for color image classification. The search for optimal parameters to use in the second experimentation round could also have been non-sequential. In that, every improvement in a changed parameter would result in a re-run of previously set parameters to search for a more optimal selection. A grid search could also be done to be thorough.

6.2.5 Multi-Class Layered Tsetlin Machine

The Layered Tsetlin Machine showed great promise inaccuracy for the pairwise classification. By utilizing the feature maps for every pairwise classification, one could potentially group them into a Multi-Class Layered Tsetlin Machine.

6.2.6 Mutli-Class Clause Division Problem

As described in section 6.1.4, there are indications that multi-class classification will suffer from a clause division problem. Without proper inspection and revision, multi-class classification might experience stunted accuracy as a consequence.

References

[1] O. Granmo, "The tsetlin machine - A game theoretic bandit driven approach to optimal pattern recognition with propositional logic," *CoRR*, vol. abs/1804.01508, 2018. arXiv: 1804.01508. [Online]. Available: http://arxiv.org/abs/1804.01508.

[2] C. Rudin, "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead," *arXiv*, Nov. 2018. eprint: 1811.10154. [Online]. Available: https://arxiv.org/abs/1811.10154v3.

[3] G. T. Berge, O.-C. Granmo, T. O. Tveit, M. Goodwin, L. Jiao, and B. V. Matheussen, "Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization with Medical Applications," *arXiv*, Sep. 2018. eprint: 1809.04547. [Online]. Available: https://arxiv.org/abs/1809.04547.

[4] O. Granmo, S. Glimsdal, L. Jiao, M. Goodwin, C. W. Omlin, and G. T. Berge, "The convolutional tsetlin machine," *CoRR*, vol. abs/1905.09688, 2019. arXiv: 1905.09688. [Online]. Available: http://arxiv.org/abs/1905.09688.

[5] A. Phoulady, O.-C. Granmo, S. R. Gorji, and H. A. Phoulady, *The weighted tsetlin machine: Compressed representations with weighted clauses*, 2019. arXiv: 1911.12607 [cs.LG].

[6] C. Satten, *Scanning Photographs with a Fax Machine*, [Online; accessed 26. May 2020], Aug. 1998. [Online]. Available: https://staff.washington.edu/corey/fax.html.

[7] *Miscellaneous Image Transformations — OpenCV 2.4.13.7 documentation*, [Online; accessed 27. Dec. 2019], Dec. 2019. [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=threshold.

[8] V. Podlozhnyuk, "Image convolution with cuda," *NVIDIA Corporation white paper, June*, vol. 2097, no. 3, 2007.

[9] *OpenCV: Image Thresholding*, [Online; accessed 9. Dec. 2019], Dec. 2019. [Online]. Available: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html.

[10] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[11] *OpenCV: Canny Edge Detection*, [Online; accessed 27. Dec. 2019], Dec. 2019. [Online]. Available: https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html.

[12] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using matthews correlation coefficient metric," *PLOS ONE*, vol. 12, no. 6, pp. 1–17, Jun. 2017. DOI: 10.1371/journal.pone.0177678. [Online]. Available: https://doi.org/10.1371/journal.pone.0177678.

[13] M. Döring, *Performance measures for multi-class problems*, Dec. 2018. [Online]. Available: https://www.datascienceblog.net/post/machine-learning/performance-measures-multi-class-problems/.

[14] *CIFAR-10 and CIFAR-100 datasets*, [Online; accessed 25. May 2020], Apr. 2017. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[15] M. L. Tsetlin, "FINITE AUTOMATA AND MODELS OF SIMPLE FORMS OF BEHAVIOUR," *Russ. Math. Surv.*, vol. 18, no. 4, pp. 1–27, 1963, ISSN: 0036-0279. DOI: 10.1070/rm1963v018n04abeh001139.

[16] B. Tung and L. Kleinrock, "Using finite state automata to produce self-optimization and self-control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 4, pp. 439–448, 1996.

[17] cair, *pyTsetlinMachineParallel*, [Online; accessed 19. May 2020], May 2020. [Online]. Available: https://github.com/cair/pyTsetlinMachineParallel.

[18] O.-C. Granmo, *Private Communication during supervision sessions*, 2020.

[19] S. Lawrence, C. L. Giles, Ah Chung Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

[20] M. Courbariaux, Y. Bengio, and J.-P. David, *Binaryconnect: Training deep neural networks with binary weights during propagations*, 2015. arXiv: 1511.00363 [cs.LG].

[21] R. A. Ulichney, "Dithering with blue noise," *Proceedings of the IEEE*, vol. 76, no. 1, pp. 56–79, Jan. 1988, ISSN: 1558-2256. DOI: 10.1109/5.3288.