

Expanding Convolutional Tsetlin Machine for Images with Lossless Binarization

JENS MARTIN HÅSÆTHER

SUPERVISOR

Lei Jiao

Ole-Christoffer Granmo

University of Agder, 2021

Faculty of Engineering and Science

Department of Engineering Sciences

Acknowledgements

I wish to show my gratitude to my supervisors for help, guidance and troubleshooting. I wish to thank several academic staff and personal of the University of Agder that helped with various issues ranging from academical to technical as well as for use of their computational power. Finally I wish to express my deepest gratitude to my family, for being supportive during the pandemic. Without the combined help and assistance of all these this thesis would not have been possible.

Abstract

Deep convolutional neural networks (CNN) is known to be efficient in image classification but non-interpretable. To overcome the **black box** nature of CNN a derivative of the Tsetlin automata, the convolutional Tsetlin machine (CTM) which is transparent and interpretable, was developed. As CTM handles binary inputs, it is important to transform the input images into binary form with minimum information loss so that the CTM can classify them correctly and efficiently. Currently, a relatively lossy mechanism, called adaptive Gaussian thresholding, was employed for binarization. To retain as much information as possible, in this thesis, we adopt an adaptive binarization mechanism, which can offer lossless input to the CTM. In more details, we employ up to 8 bits in three colour channels and arrange them in a certain way so that the CTM can handle all bits as input. In addition, we can also select certain significant bits and ignore others to increase efficiency of the system. Numerical results demonstrate that the newly proposed mechanism has potential to achieve better results than those of adaptive Gaussian mechanism when enough data is given after enough training epochs. The code used for this thesis is available at the [11].

Contents

Acknowledgements

Abstract

| | |
|--|----------|
| 1 Introduction | 1 |
| 1.1 Project Description | 1 |
| 1.1.1 Previous Work | 2 |
| 1.1.2 Problem Statement | 2 |
| 1.1.3 Assumptions | 2 |
| 1.1.4 Hypothesis | 2 |
| 1.1.5 Scope | 2 |
| 1.2 Motivation | 3 |
| 1.3 Goal | 3 |
| 2 Background & State of the Art | 4 |
| 2.1 Brief Explanation of Artificial Intelligence | 4 |
| 2.1.1 Archetypes of AI/ML | 5 |
| 2.2 Brief Overview of Tsetlin Technologies | 6 |
| 2.2.1 Tsetlin Machine | 6 |
| 2.2.2 Convolutional Tsetlin Machine | 7 |
| 2.3 Notes on Neural Networks | 7 |
| 2.3.1 Output Relation | 7 |
| 2.3.2 Convergence Time | 8 |
| 2.3.3 Inference Time | 8 |
| 2.3.4 Computing Power | 8 |
| 2.3.5 Dataset | 8 |
| 2.4 Thresholding | 9 |
| 2.5 Parameters for Performance Evaluations | 9 |
| 2.5.1 Accuracy | 10 |
| 2.5.2 Overfitting | 10 |
| 2.6 Output Interpretation | 10 |
| 2.6.1 100% and 0 learning curves | 10 |
| 2.6.2 Statistics | 11 |

| | | |
|----------|--|-----------|
| 3 | Proposed Method | 12 |
| 3.1 | Method overview | 13 |
| 3.2 | Initialising | 13 |
| 3.2.1 | Version Control | 13 |
| 3.3 | Importing Dataset | 13 |
| 3.4 | Pre-Process Binarization | 14 |
| 3.4.1 | Flowchart of Binarization Process | 14 |
| 3.4.2 | Binarization | 16 |
| 3.4.3 | Modelling 2s Complement as a Threshold | 17 |
| 3.5 | Train Model | 18 |
| 3.5.1 | Convolutional Tsetlin Machine | 18 |
| 3.5.2 | Hyperparameters | 18 |
| 3.6 | Visual Comparison of Techniques | 19 |
| 3.6.1 | Image Information Breakdown | 19 |
| 3.6.2 | Example Image from Dataset | 20 |
| 4 | Performance Evaluation | 21 |
| 4.1 | CTM Hyperparameters and Manual Loss | 21 |
| 4.1.1 | Curves and Axis | 21 |
| 4.1.2 | Observations | 21 |
| 4.2 | Results | 22 |
| 4.2.1 | Difference in Input-size | 23 |
| 4.2.2 | Comparing LL4 to AGauss | 23 |
| 4.2.3 | Scaling to Full Dataset | 24 |
| 4.2.4 | Linear Input/Clauses Ratio Assumption | 25 |
| 4.2.5 | Manual Approximation for the Logistic Function | 26 |
| 4.3 | Convergence and Factor Analysis | 27 |
| 4.3.1 | Speculation on the Slope Difference | 28 |
| 5 | Discussion | 29 |
| 5.1 | Observation on the RBG Channels | 29 |
| 5.2 | Experience and Epiphanies | 29 |
| 5.3 | Retrospect | 29 |
| 5.4 | Future Work | 30 |
| 5.4.1 | Check Breakpoints/Equilibrium Methods | 30 |
| 5.4.2 | Expand System for \mathbb{R}^N | 31 |
| 5.4.3 | Expand System for Per-pixel Labelling | 31 |
| 6 | Conclusions | 33 |
| A | Code Snippets & Misc | i |
| A.1 | Manual Import of CIFAR10 | i |
| A.2 | Library Import of CIFAR10 | ii |
| A.3 | Readout of "nvidia-smi" - Jupyterlab GPU | ii |
| A.4 | Terminal Readout of CUDA and CPU capacity | ii |

Bibliography & Sources

List of Figures

| | | |
|-----|---|-----|
| 2.1 | A Tsetlin Automaton for two-action environments [18]. | 6 |
| 2.2 | Thresholding techniques comparison [26]. | 9 |
| 3.1 | Overview over the new proposed system. | 12 |
| 3.2 | Flowchart of the binarisation process using Diagrams [15]. | 14 |
| 3.3 | Pixel Extraction and per pixel data | 16 |
| 3.4 | Pixel Extraction and per pixel data | 17 |
| 3.5 | Comparison of techniques over an 32x32 image with hue-maps of Red, Green, Blue, and RGB respectively | 19 |
| 3.6 | Comparison of techniques on Cifar-10_Train_1 in multicolour RGB | 20 |
| 3.7 | Comparison of techniques on Cifar-10_Train_1 in monochrome G | 20 |
| 4.1 | Graph of mean of the accuracy per epoch of the mean accuracy per image. | 21 |
| 4.2 | Comparison of techniques on Cifar-10_Train_23099 in monochrome Yellow/Purple. | 22 |
| 4.3 | Comparison of AGauss and LL4 at 5000 images, 800 clauses. | 23 |
| 4.4 | Difference in Accuracy/ $\Delta a(x)$ for AGauss and LL4. | 24 |
| 4.5 | Comparison of AGauss and LL4 at 50000 images, 800 clauses | 25 |
| 4.6 | Juxtaposition of AGauss and LL4, where LL4 has 4xClauses | 26 |
| 4.7 | Left side showing Logistic Curve approximation, and right side combines it with Figure 4.5. | 26 |
| 4.8 | Convergence sequences of different valued L and k for the logistics formula using GeoGebra [31]. | 27 |
| 4.9 | Comparison of techniques on Cifar-10_Train_1123 in monochrome Yellow/Purple | 28 |
| 5.1 | Visualisation of the Tsetlin network | 32 |
| A.1 | Terminal showing CUDA Implementation being locked at 100% CPU Use | ii |
| A.2 | Terminal Showing CPU superseding 100% CPU use for the parallel implementation, i.e. using more than one CPU | iii |

List of Tables

| | | |
|-----|--|----|
| 1 | Acronyms & Abbreviations | |
| 2.1 | Output Relation-grid | 7 |
| 3.1 | Versions used in current project of certain modules | 13 |
| 3.2 | Shapes of the imported CIFAR10 dataset. | 14 |
| 3.3 | Decimal and Binary representation of numbers | 17 |
| 3.4 | Binarization Scheme for the proposed system | 18 |
| 4.1 | Values and differentials from Figure 4.3 and Figure 4.5. | 25 |
| 5.1 | Transpose of the binary representation. | 31 |

| # | Explanation |
|-------|---|
| 2d/3d | 2-dimensional/3-dimensional |
| AI | Artificial Intelligence |
| AMT | Adaptive Mean Thresholding |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| CTM | Convolutional Tsetlin Machine |
| DAG | Directed Acyclig Graph |
| DCNN | Deep Convolutional Neural network |
| FFN | Feed Forward Neural network |
| GAN | Generative Adversarial Networks |
| GPU | Graphics Processing Unit |
| LSTM | Long-Short Term Memory |
| ML | Machine Learning |
| NN | Neural Network |
| OS | Operating System |
| RGB | Red-Green-Blue (Computer primary colours) |
| RNN | Recurrent Neural Network |
| SotA | State of the Art |
| TA | Tsetlin Automata |
| TM | Tsetlin Machine |
| VGG | Visual Geometry Group |

Table 1: Acronyms & Abbreviations

Chapter 1

Introduction

Research on the Tsetlin machine [18] has the potential to bring forth many new innovations within the field of Machine Learning (ML). These include concepts such as more explainable Artificial Intelligence (AI), from which we as human beings- more easily can make intuitive guesses, as to why and how such systems learn patterns, and make decisions. With this novel technology comes many new and interesting challenges to overcome in order for the Tsetlin and its derivatives to compete with the other state of the art methods. The Tsetlin machine requires input in binary form. Now you may ask yourself, isn't all data on a computer stored binary, but there is an important difference in certain data structures. For example if you have a letter lets say "e", this can be represented as the value $65_{16} = 101_{10} = 01000001_2$. If we want to actually represent the decimal number 101, the format would be the same $65_{16} = 101_{10} = 01000001_2$. To make a long story short, there is a sort of header to the data, that tells it what *type* this form of data is, where you can have strings, integers, floats, etc. However its not always clear how to go between them. E.g. if we had the decimal number 101 in integer form, and wanted it in string form- what should be the correct output e?, 101?, one hundred and one?. This ambiguity is what creates some of the interesting problems for what seems like an arbitrary decision to read a decimal number as a binary number. There still needs to be a method to decide how to convert the number. The state of the art method that is used today suffers from loss and compression during the conversion of data. These losses may make it harder for the Tsetlin to make accurate predictions.

In this master thesis we will explore another option to remove or minimise loss in raw image format conversions. We will attempt to implement a new method that is lossless and compare the results against the current lossy implementation.

1.1 Project Description

The current solution for handling computer images is by the use of Adaptive Gaussian Thresholding to do a lossful compression of [0-255] down to [0-1]. The thesis seeks to find an alternative solution that does not remove information from the given image.

1.1. PROJECT DESCRIPTION

1.1.1 Previous Work

The precursor to this thesis was a project called "Analysis of binarization techniques and Tsetlin machine architectures targeting image classification" [39]. Some of the findings of this paper will be used to further benchmark the quality of the new model.

1.1.2 Problem Statement

The TM and CTM only accepts 0 and 1 as input. Most common image formats has 0-255 (8-bit) for all 3 RGB channels. Creating a method or solution that enables lossless input of 8-bit images, into a convolutional Tsetlin machine (CTM) [20] container that can only read binary input.

1.1.3 Assumptions

- A strict one to one comparison of previous work, may not be representative as the size of the input, and resulting convergence space will differ.
- A lossless version as opposed to a lossy version will provide information that the lossy version does not.
- As this thesis focuses on the Artificial Intelligence part and not the mathematics part specifically, some approximations may be necessary, as opposed to perfect solutions.

1.1.4 Hypothesis

As the current method uses Gaussian thresholding which is a very lossful format, expanding it to govern more of the data should be able to increase accuracy. By keeping more data will mean higher training times, but the end result in the peak accuracy is expected to be higher given enough training time. However as with anything, this effect might not kick in, before the amount of data reaches a certain size. As we are taking in more information, we will need more time to find the distinguishing features for a given function. We will have room to find more than the baseline version, whether the effect of this is significant enough will show in time. How much longer/extra we need to train before this effects that to show is up for speculation.

1.1.5 Scope

This project seeks complete #1 and then work down the list as time allows for.

1. Enable the CTM to accept lossless 8 bit colour-depth images.
2. Compare the lossless method with Adaptive Gaussian Thresholding
3. Optimise the lossless function
4. Compare both methods to see, what problems each method excels on.

Challenges

Knowing exactly what an AI model would need to optimally solve a classification task, would be the same as a perfect solution to the classification problem. However from a bottom-up standpoint, this solution is for most problems infeasible to calculate. The solution is then to try to limit what the AI model needs to learn, by giving it something that can be referenced against, e.g., labels for a classification task.

Procedure

As the input needs to be in binary, we think it goes without saying that binarisation would have to be involved at one stage or another in the process. Beyond this one could speculate that as we are adding more information, the input needs to be larger, and potentially training time needs to be extended. Some argument then needs to be made, for when and/or if the new method will beat the old one.

1.2 Motivation

Being able to have a general conversion method between 8 bit colour-depth images and binary, would enable the CTM to accept most image databases, assuming they employ standard 8-bit colour-depth images.

1.3 Goal

Create a baseline that enables input of non binary coloured computer images, for the CTM.

The remainder of the thesis is organised as follows. In Chapter 2, the background and the related studies are summarised. The newly proposed binarization scheme and the corresponding CTM system is detailed in Chapter 3. The performance of the proposed mechanism is evaluated in Chapter 4 before the discussions and the conclusions are given.

Chapter 2

Background & State of the Art

In this chapter we will try to introduce the reader into the relevant field(s) within AI/ML research. We will present several key aspects of the state of the art, but focus mainly on the ones that pertain to this thesis. We will also focus on how we will proceed in troubleshooting and trying to get the method to work, as well as some limitations towards what we will be focusing on.

2.1 Brief Explanation of Artificial Intelligence

The idea that you can teach a system to perform a task through mathematics is not a new concept. However in recent years there have been great advances in this area, which culminated in technologies such as the Learning Automata [40]. This technology, and others like it, serves as a predecessor for what we today call Artificial Intelligence (AI) and Machine Learning (ML). As there seems to be a looming possibility that AI/ML might one day be on par with humans, there needs to be a convention for how it is developed. Philosophers seem to agree that a top-down approach, that follow virtue ethics would be the most nominal case [3][23][53], while not all- most state of the art AI/ML today follows a bottom up approach[36][6]. The reason the bottom-up approach has gotten a sort of reincarnation in interest in the last years, is as Moore's Law [44] has steadily increased the computing power in an exponential rate. Comparing it to earlier theories of Machine Learning on checkers from 1959 [43], to take an example. Today such tasks as solving checkers can be seen on as trivial (computing wise), even in a brute force sense. What this exponential increase in computing power has effectively done, is making tasks that would be seen as unfeasible a few decades prior such as the GPT-3 [5], to enter the realm of possibilities. Despite Moore's Law eventually coming to an end, as micro-transistors are getting closer to the point where a single transistor, barely has a few atoms to check the charge of, and quantum physics kicks in [12], there is still room to increase output by use of technologies such as parallel programming and clustering [28]. This means that even for the foreseeable future we can expect computing power to increase to new limits, further increasing the likelihood a bottom-up AI/ML method, will manage to converge meaningfully for a given task.

2.1.1 Archetypes of AI/ML

Depending on what you are trying to predict, and what you are using for your prediction, there are several options. Do note that we will be generalising some concepts. Even though a technology is said to be good at handling images, there exists methods to present certain data such as the 2d vector map representation of Navier-Stokes equations as images [9], even though they explicitly aren't.

FFN and Perceptron

If you can assume that your data points, have relatively low noise and that one input can correspond to another (non cyclic), then a perceptron type network [45] could be a proper fit. E.g. you are trying to predict whether someone is walking, cycling or driving based on the relative speed/acceleration of a given object. While today, the advent of technologies such as the Feed Forward Network [16] has become more popular than the general perceptron, they are based on similar concepts.

LSTM/RNN

If you have either time-stepped information, or anything that can be represented as continuous (cyclic) data, then Long-Short-Term Memory (LSTM) [24] is commonly used. Just as with the perceptron, there has been a predecessor in the Recurrent Neural Networks (RNN) [54]. Weather or stocks is great examples of typical cyclic data, and that sometimes you want to predict more than one unit ahead.

Finite Markov Decision Process and GAN

If you have a more complex task, that has multiple solutions, and multiple states- you have two general options: If you can assume that within reason it is brute-force-able and/or can be expressed as a Finite Markov Decision Process [46], then you can use archetypes such as the Monte Carlo Tree Set [7] or Q-learning [47]; if on the other hand you may need to change decision-state-space underway, but you can assume that the task has a general pattern, then technologies such as a Generative Adversarial Networks [17] is more common. Examples of tasks here is chess for Finite Markov Decision Process and image generation of faces for variable decision state space.

Convolutional Neural Network

The last archetype we will go through, and the one that is most relevant to this thesis, is the Convolutional Neural Network (CNN) [34]. It takes in portions or *sliding windows* of a given image, and looks at how it will affect the function of the entire image, through mathematical convolutions [27]. Examples of tasks/datasets that CNNs are commonly used for is the ImageNet [14] or CIFAR [33] of which the latter will be used as a benchmark. Despite narrowing down the archetypes to only one, there is still a lot of competing technologies within CNN. What this type of network does is try to segment the images up into frames or sliding windows, and see if the corresponding pattern(s) of data has a

structure. More mathematically speaking its how a sub-function over a given area, effects the overall function (rest) of the image [27]. A more standard version of a CNN using the Tensorflow platform [50] and the Keras API [29] would look something like this [10]. There are limitations with some of these technologies that becomes apparent, first when the sizes become very large, such as for gigapixel imagery. Just as there are competing technologies for different problems, large scale image processing is done by Inception-Resnet [48], VGG [22], DAG [49], DeepLab [8] and DenseNet [25] to mention a few. We will however focus on the CTM [20] for this thesis.

2.2 Brief Overview of Tsetlin Technologies

One of the precursors to modern day AI/ML that has until recently, mostly gone unnoticed is the Tsetlin Automata, proposed by Michael Lvovitch Tsetlin in his 1963 paper *"Finite Automata and models of simple forms of behaviour"* [51]. An example of how the Tsetlin Automaton processes information:

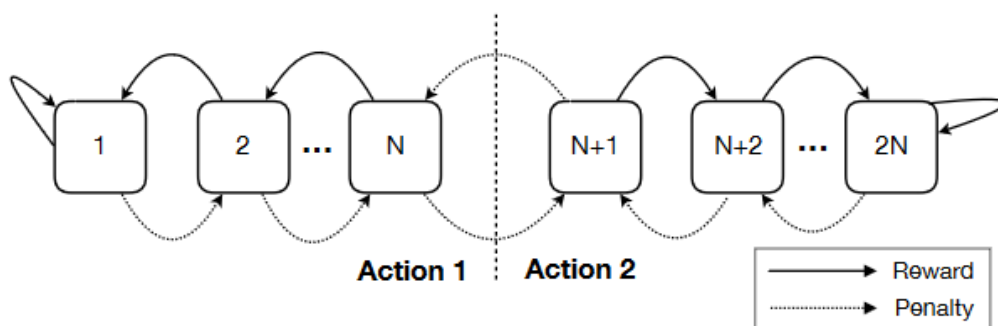


Figure 2.1: A Tsetlin Automaton for two-action environments [18].

Above in Figure 2.1 we can see that the Automata learns by changing states, based on rewards and penalties. The further away from the cutoff point between Action 1 and Action 2 the system is, the further its confidence is increased that it has the right Action for the its given state [19]. This technology has since been expanded upon to create several automaton, working together to become what has been referred to as a Tsetlin Machine [18].

2.2.1 Tsetlin Machine

While the properties on the Tsetlin Automata are interesting, they can not solve complex problems. However, using Boolean logic, and binary arithmetic allows for chaining together several Tsetlin Automata to what can be described as an array/several Automata, which we will henceforth refer to as a Tsetlin Machine (TM) [18]. TM has been employed in many applications [13, 55] and the theoretical analysis of it can be found [56]. For image processing, the technology has been tested on, and has showed promise for, is the dataset MNIST [21]. However to be able to tackle more complex tasks that don't directly

translate to a binary representation, there are further advancements to be made. It is one of these advancements, of this technology called the CTM [20] that we will focus on in this thesis.

2.2.2 Convolutional Tsetlin Machine

As mentioned in Section 2.1.1, one of the baselines for image processing is the use of CNNs. Being able to determine how an area effects the overall function of an image, has been crucial for the state of the art of image classification. While still having the limitation of taking in only 0 or 1, the Tsetlin system has now the ability to perform mathematical convolutions with the new CTM [20] framework. What distinguishes it however from standard CNN architectures is that instead of having standard convolutions, it instead utilises convolution clauses, as well as storing information about the coordinates of said clause. One of the upsides of doing it in this way, is that the CTM is transparent and interpretable, which sets it apart, from almost any state of the art DCNN/CNN currently employed. With systems slowly but surely being able to do more complex tasks, the need to be able to see what led to a certain decision, becomes more and more valuable, at least if you want to employ that technology among humans. One of the downsides, is as mentioned the strictness about what the CTM can take as an input. To try to solve this limitation is what this project will focus on.

2.3 Notes on Neural Networks

2.3.1 Output Relation

Output relation is how often the model(s) guess gets right, in relation to what the real answer is. While most commonly used for binary output, (e.g. as seen in the the 2x2 grid below) output relation can also be used for some non-binary outputs.

| | True | False |
|----------|------|-------|
| Positive | 92% | 1% |
| Negative | 1% | 6% |

Table 2.1: Output Relation-grid

To extrapolate on this table: 92% of the time the model guessed a positive output, it corresponds with the real positive classification, and is wrong on this guess 1% of the time; 1% of the time it guesses a negative output, it corresponds with the real negative classification, and is wrong 6% of the time. For most applications this might be a very good model, but if we take a viral disease test as an example, means that 6 of every hundred patients walk out of the ward, thinking they are well(or at the very least something else is the issue), when they are actually sick. The ability to reduce False Negatives, might have a larger impact than increasing/reducing other factors in these situations. For non-binary values you can use abstraction i.e. a stock rose when it actually fell, as opposed to using the exact value the stock rose, or fell by. Alternatively see which categories it performs good on, and which it performs bad on. What this helps you accomplish in terms

of over-fitting, is being able to determine the discriminant relation, i.e. what is hard to learn, what is hard to differentiate, and what categories has overlapping features.

2.3.2 Convergence Time

There is also some factors that is not related to the data itself, but how the model handles the data. One of these factors is how long it takes for the model to train, until it reaches whatever accuracy/threshold set for the given task. Depending on the deadline of the project, be that a product you sell a client, or annual updates that need to happen to the model, if the time it takes to train/retrain your model is longer than this set time, then the model might as well be useless.

2.3.3 Inference Time

Inference time refers to how long it takes a trained model to translate an input, to a prediction. This can be important, especially if you have a continuous stream of data/input, as you can end up with a bottleneck, if the inference time per data-point is longer than the time between them.

2.3.4 Computing Power

If we take GPT-3 [5] as an example, even after training is done, performing a single inference, requires a cluster of computers to compute [5]. System specs the model is supposed to work on can therefor be an important factor.

2.3.5 Dataset

The dataset is usually what the model takes in as input. If the dataset has labels, the training is said to be supervised, as there are definite categories the NN is working towards. If the dataset is without labels, it is said to be unsupervised, and the goal is usually to find groupings, of similar data. This project does not seek to test capabilities on a dataset specifically, but will use one as a benchmark, to talk comparatively about other technologies. Since the previous project used the CIFAR-10, this is the dataset that will be used for this purpose. CIFAR has 2 datasets respectively namely CIFAR10 and CIFAR100 which refers to the datasets having 10 and 100 categories/labels respectively [32]. The CIFAR10 dataset consists of 50000 training, and 10000 testing images, and has a roughly even split between its 10 categories.

2.4 Thresholding

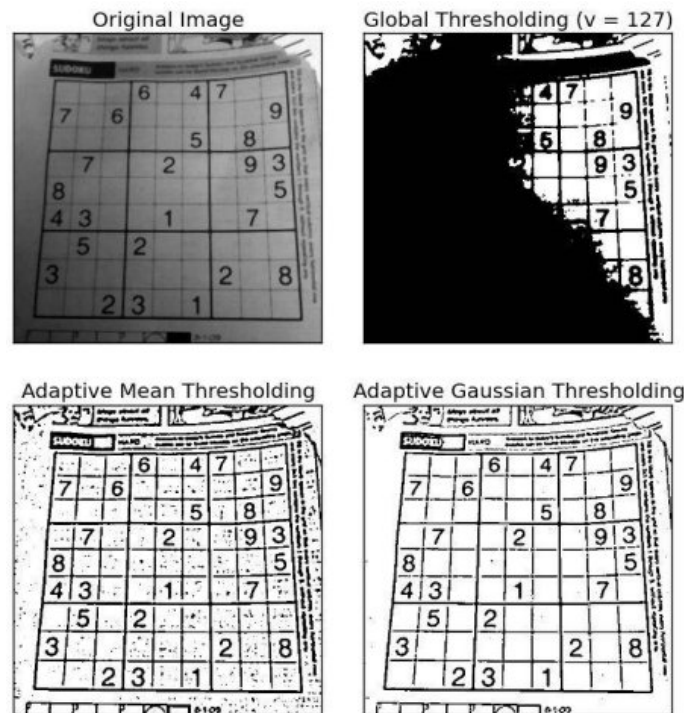


Figure 2.2: Thresholding techniques comparison [26].

A standard thresholding technique would be to split the range of the data into a variable V brackets, and then optionally do some operation to account for different weighting of the values [42]. However as described best by Figure 2.2 standard thresholding has its limitations when it comes to image noise. The optional operation mentioned earlier can for example be Adaptive Mean Thresholding (AMT). The AMT does the following "The threshold value is the mean of the neighbourhood area minus the constant C " [26], or Adaptive Gaussian which instead does "The threshold value is a Gaussian-weighted sum of the neighbourhood values minus the constant C " [26]. The different results can be viewed in the bottom left and bottom right images respectively of Figure 2.2.

2.5 Parameters for Performance Evaluations

When considering performance evaluations there are a few things to take into consideration. There are many different ways in which a model may be evaluated upon. Some of the factors are: Accuracy, Over-fitting, Output relation, Convergence Time, Inference Time and Computing Power. While these may affect the model to varying degrees, being able to distinguish them help you combat them easier.

2.5.1 Accuracy

Accuracy is one of the most common ways to measure a models effectiveness. What this enables you to see, is how often you are able to make a correct output/guess, for a given input/data point. What the metric accuracy effectively measures, is the neurons ability to translate the input space, onto the output space. While at first glance it seems to be a encompassing metric, it does not account for variables such as overfitting.

2.5.2 Overfitting

While a model might perform very well on its training/data-set, the real value of an AI/ML model comes from being able to perform well on similar data, and not just the exact dataset you have provided it. Overfitting refers to a situation where you perform well on the dataset you trained on, but are unable to transfer the results to similarly labelled and structured datasets. This is usually a symptom of having either too many epochs on too little data, or too many neurons, for too few data-points. The result is that you make the model learn too many/too few, features of a given data-point. Net result is the model having no leeway in terms of similarity, despite two data-points being the same reference/ground-truth. This metric is very difficult to get rid of all-together, as it is hard to beforehand know the exact extent, of a given real-input-space vs model-input-space. There exists methods to at least reduce the chances for overfitting, (beyond the aforementioned epochs, and amount of data), such as output relation.

2.6 Output Interpretation

If a given model is producing weird output/predictions, it is usually (but not always) problems with either the dataset, or the code. A consideration is that a model can not output, something which you did not implement/input, either explicitly or implicitly. With this in mind we can approach the problem of finding the reason for weird predictions, in a much more constricted sense. While none of these methods are a catch-all-solve-all solution, they give you the ability to eliminate one or more factors.

2.6.1 100% and 0 learning curves

There are many ways to force both 100% accuracy, and a gated learning curve. However assuming that the task is complex enough, to warrant using AI in the first place, and the dataset is sufficiently large, to do a 60/40 cutoff without compromising learning, we can reach the following conclusions:

- 100% accuracy should never happen, and is either neuron space close to or equal the permutation/combination space, of the input or overfitting through epochs.
- You have either capped the potential of the model, be that the surjective limit(too small onto space), either via layers, neurons or output space.
- While local minima/maxima can be hit, you rarely hit the same minima/maxima multiple times for non wave-form-data tasks.
- You data is literally random noise.

2.6. OUTPUT INTERPRETATION

Information at hand we would gather to troubleshoot this would be:

1. Input Space (Model)
2. Neuron Space
3. Output Space (Model)
4. Epochs

It might also be an idea to change where the 60/40 cutoff is for the data and/or shuffle the data.

2.6.2 Statistics

If applicable it can be a good idea to run general statistics, over all or some of the data points. Some of the important values to get out are: Average Value, Mean Value, Variance, Standard Deviation, Minimum Value, Maximum Value, and sometimes $Q1+Q3$. What these values enables you to do is to say something about their general distribution, and if there might be anomalies in the dataset. Sometimes though it might not be sufficient with raw data, and graphing the data for visual inspection is another option.

Graphing

Visualisation can be a great tool to look for boundary values that differentiate highly from the rest of the set. Sometimes there may be 2 big clusters, which can tell you that you might have 2 different classes, while other times it might be stray values. These stray values might interfere with training, so either removing them, or using a method to prune them, such as to cut extremal values beyond a certain threshold might be the best approach.

Cutting Extremal Values

Sometimes there may be either mislabel-ed data, data that is too far off the average value, or errors occurring in reading/collecting. An example of this could be a accelerometer measuring walking/jogging speed that you attach to your arm, and you suddenly wave at a passerby. This might create a spike in the registered speed, while the whole set might still be label-ed *jogging*. One way to deal with these extremal values is to take advantage of the empirical rule [41] and prune data further than 3σ /std from mean. What this means, is that the overall range of the distribution gets reduced, and you only encompass 99.973...% of the actual data, (which gives roughly a 1 in 370 of it being wrong), but has the upside of being more specific on average data.

Normalising

As input/data can sometimes come from multiple sources, it might be wise to normalise values between 0 and 1. What this helps with is to make sure that even though systems might be calibrated differently, the ratios are preserved. However if it is not used in junction with other points mentioned in 2.6.2 they might sometimes produce problems.

Chapter 3

Proposed Method

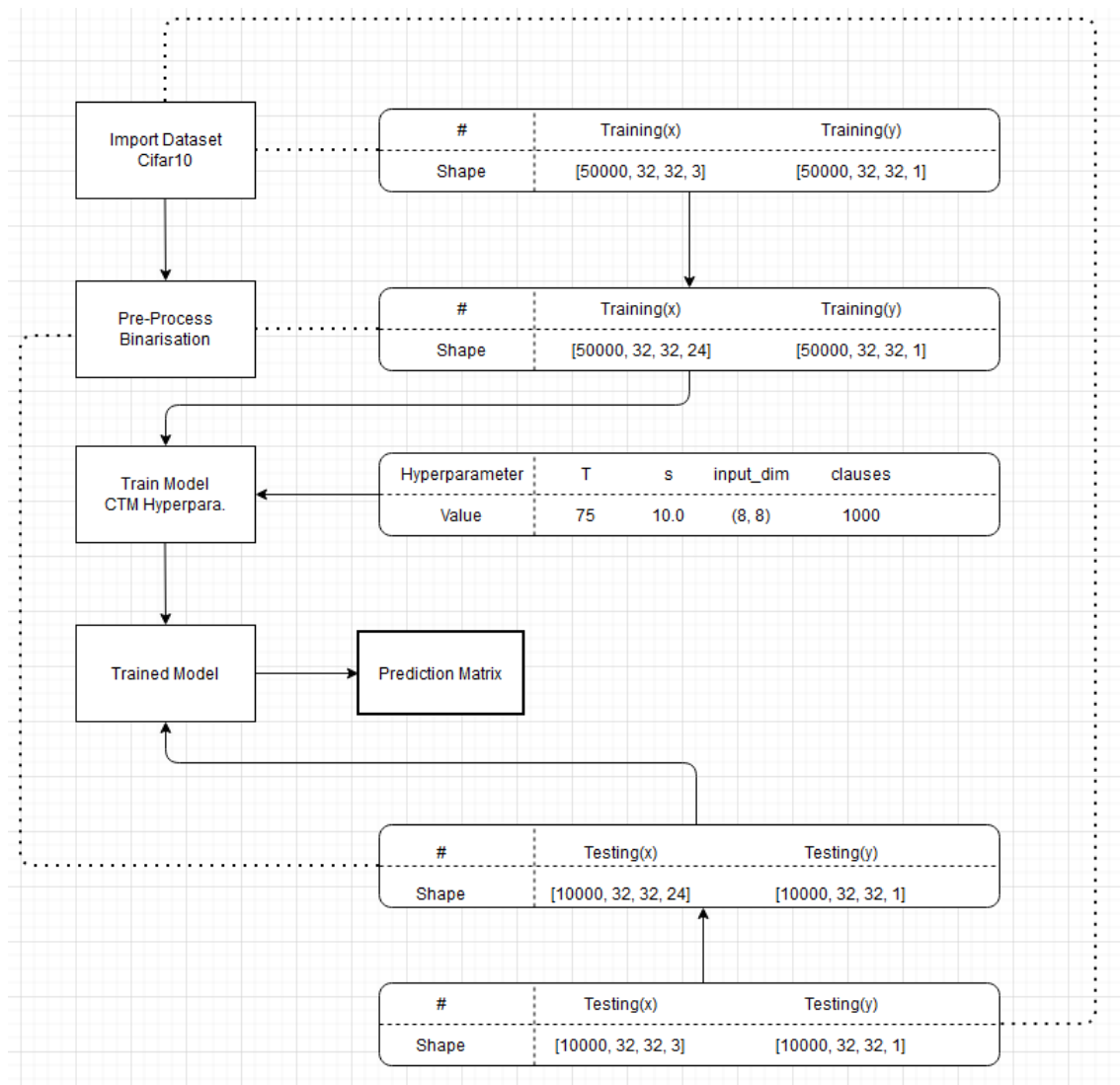


Figure 3.1: Overview over the new proposed system.

In this chapter we will focus on attempting and working on the proposed new method. One of the first things to take into account when one wants to set up any system, is what the requirements for the modules that one need to import/implement are.

3.1 Method overview

Going from Figure 3.1 we can see in general terms what the method will do. It starts by importing the dataset Cifar-10, which will then have a certain size (Chapter 3.3). In this case, the training input set of the Cifar 10 has 50000 images, and the images are 32x32, with one value for Red, Green, and Blue respectively giving it the final shape of [50000, 32, 32, 3]. Most of the values are the same for the training output set, albeit having 1 value as the label, as opposed to 3 for colours giving it the final shape [50000, 32, 32, 1]. The testing set only has 10000 images, but are otherwise of same shape as the training set, giving it the final shape x/input = [10000, 32, 32, 3] and y/output = [10000, 32, 32, 1]. The next step is then to pre-process it, to fit the goals presented in Chapter 1.3. The fine details of this pre-processing process will be explained in Chapter 3.4.2. The main takeaway for now is that the shape will change. The next step is to train the model on the training set, which now is pre-processed, with given hyper-parameters. Finally use the trained model so see how good it is at predicting from the testing set which has also undergone the same pre-processing. Finally we receive a matrix/list about how well it performed and we can then analyse this data (Chapter 4).

3.2 Initialising

One of the first things to take into account when you want to set up any system, is what the requirements for the modules that you need to import/implement are. Starting with a version control, of what the system looks like, to easier replicate it.

3.2.1 Version Control

| Machine | OS | Python | Tensorflow |
|------------|--------------------|--------|------------|
| Desktop | Windows 10 | 3.7.4 | 2.3.0 |
| Jupyterlab | Ubuntu 18.04.3 LTS | 3.7.4 | 1.15.2 |

Table 3.1: Versions used in current project of certain modules

3.3 Importing Dataset

There are 2 ways of importing the CIFAR-10 dataset, one requires you to download the modules, and construct each image from their respective colour channel, as can be seen a solution for in Appendix A.1. The other arguably more convenient way is by the use of either baseline Keras [29], or as a add-in from newer Tensorflow versions [30] as such, as seen in Appendix A.2.

3.4. PRE-PROCESS BINARIZATION

| | Input | Output |
|----------|--------------------|--------------------|
| Training | [50000, 32, 32, 3] | [50000, 32, 32, 1] |
| Testing | [10000, 32, 32, 3] | [10000, 32, 32, 1] |

Table 3.2: Shapes of the imported CIFAR10 dataset.

3.4 Pre-Process Binarization

3.4.1 Flowchart of Binarization Process

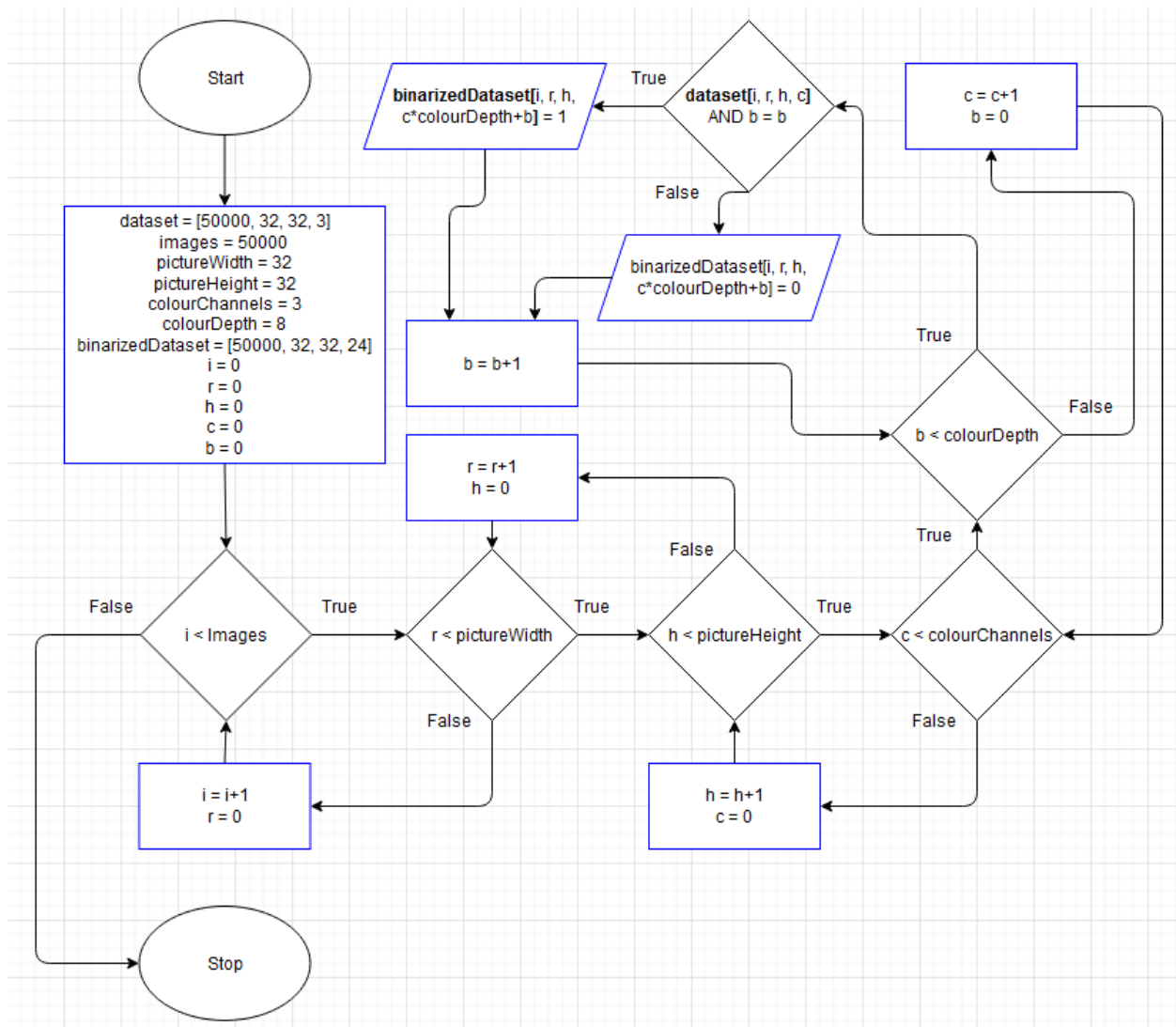


Figure 3.2: Flowchart of the binarisation process using Diagrams [15].

If we start in the upper-left corner and start the process there are a few things that need to be initialised:

3.4. PRE-PROCESS BINARIZATION

- The input/data of the system (shape shown in this example)
- How many images/objects are being processed
- What is the width of an image.
 - This system currently assumes all images have the same dimensions (which is also a requirement for the CTM).
- What is the height of an image.
 - This system currently assumes all images have the same dimensions (which is also a requirement for the CTM).
- How many colour channels.
 - Most standard imagery uses Red, Green and Blue channels to represent graphics. This means 3 one for red, one for green and one for blue respectively.
- What is the colour depth.
 - This number represents the amount of bits required to express the amount of hues in binary. For standard imagery using 0-255 or 256 unique hues, this results in $colourDepth = \lceil \frac{\log 256}{\log 2} \rceil = 8$ bits.
- Shape/Dimension of the binarized dataset.
 - This can be generalised to [images, pictureWidth, pictureHeight, colourChannels*colourDepth], but raw numbers were used for readability.
- Help variables
 - Generalising for any programming language, there will need to be set 5 help variables used in FOR loop handling.

After initialisation, the first thing it does is start the nested FOR loop chain, starting with the amount of images. Next follows the dimensions of the image in width and height. Finally there's is colour channels and depth. The reason for this schema is to account for the multidimensional array of both input and output. The deepest operation (trapeze) utilises the AND function to check one bit at a time of a given hue value if it is 0 or 1 thus binarizing the decimal number.

3.4.2 Binarization

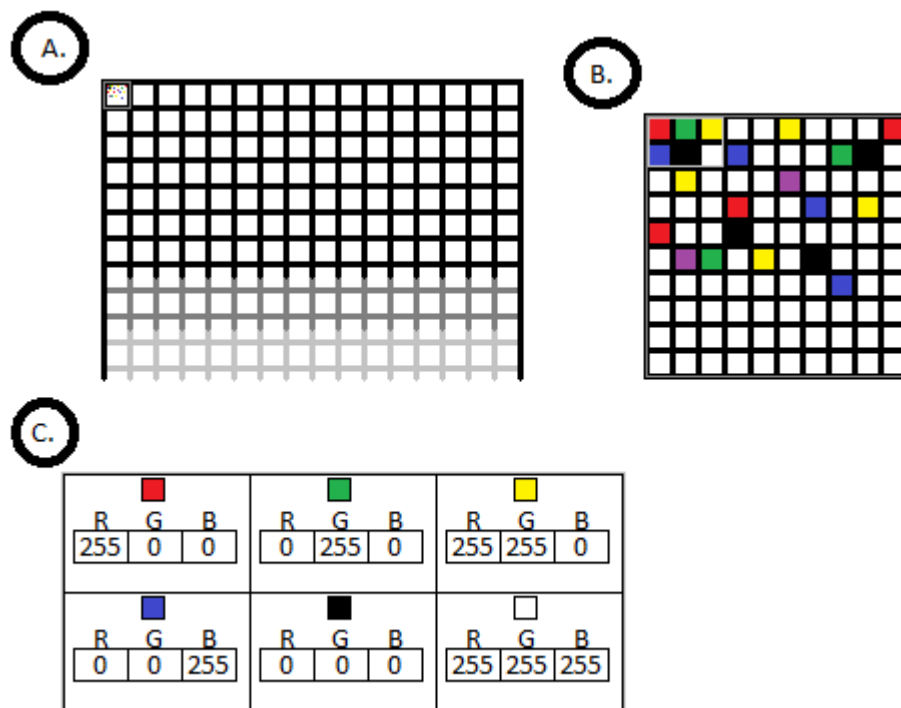


Figure 3.3: Pixel Extraction and per pixel data

From A. in Figure 3.3 imagine any image as raw input. From there crop out or use a sliding window to get a sub-set of the image B. which is easier to handle. On a per pixel basis of an image there is information regarding the **red**, **green** and **blue** colour saturation of that pixel. Assuming that the image has a 8-bit colour depth, means that the numbers range from 0-255.

3.4. PRE-PROCESS BINARIZATION

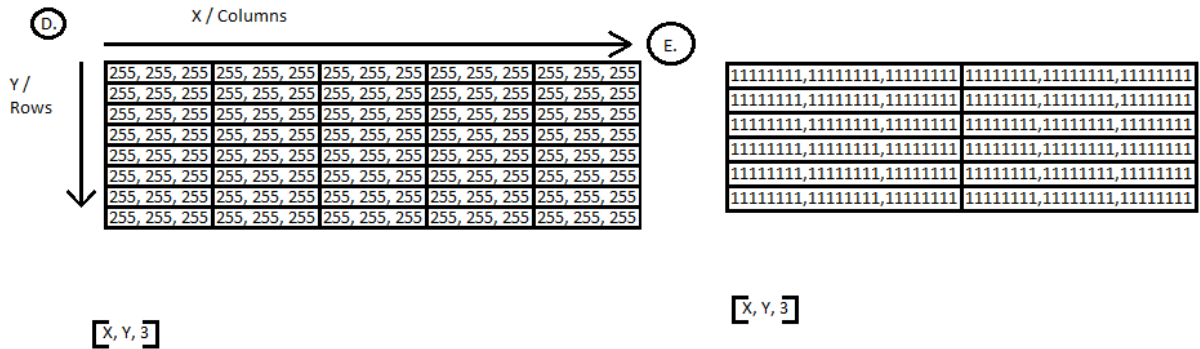


Figure 3.4: Pixel Extraction and per pixel data

From D. and E. in Figure 3.4 is how a computer would consider this crop/window. A list of sets of 3 numbers. Both for binary and decimal representation, but can be thought of as lists of [X, Y, 3] range.

| # | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |
|--------------|-------|-------|-------|-------|-------|-------|-------|-------|
| R 000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| G 150 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| B 255 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 3.3: Decimal and Binary representation of numbers

The CTM however does not take a decimal input, it takes sets of binary inputs. This means that at the very least, there needs to be some changes to make the system accept our image. One of the solutions is to simply use the raw binary representation and make the input $3 \times 8 = 24$ times bigger or [X, Y, Z] with colour depth α becomes [X, Y, $\alpha \times Z$]. While this has the downside that if you are using very large images, the input gets larger. This would still work feasibly from a categorical classification on a per small image basis, which is what CIFAR-10 is.

3.4.3 Modelling 2s Complement as a Threshold

What differentiates this new technique from normal thresholding is that instead of choosing one cutoff per threshold, it accounts for sets of thresholds as result of using the 2's complement translation of the value. This effectively gives us the following sets of thresholds T_{2^n} for the exponent of two n , represented by 1 if the formula $x \bmod 2^{n+1} \geq 2^n$ holds true:

3.5. TRAIN MODEL

| 2^n | 7 | 6 | 5 | 4 |
|-------|---|---|--|---|
| | $(256 > T_{2^7} > 128)$ | $(256 > T_{2^6} > 192) \vee$ $(128 > T_{2^6} > 64)$ | $(256 > T_{2^5} > 224) \vee$ $(192 > T_{2^5} > 160) \vee$ $(128 > T_{2^5} > 96) \vee$ $(64 > T_{2^5} > 32)$ | $(256 > T_{2^5} > 240) \vee$ $(224 > T_{2^4} > 208) \vee$ $(192 > T_{2^4} > 176) \vee$ $(160 > T_{2^4} > 144) \vee$ $(128 > T_{2^4} > 112) \vee$ $(96 > T_{2^4} > 80) \vee$ $(64 > T_{2^4} > 48) \vee$ $(32 > T_{2^4} > 16)$ |
| 2^n | 3 | 2 | 1 | 0 |
| | $\forall x \in T;$ $x \bmod 2^{3+1} \geq 2^3;$ $256 > x > 0;$ $x \in \mathbb{N}^0$ | $\forall x \in T;$ $x \bmod 2^{2+1} \geq 2^2;$ $256 > x > 0;$ $x \in \mathbb{N}^0$ | $\forall x \in T;$ $x \bmod 2^{1+1} \geq 2^1;$ $256 > x > 0;$ $x \in \mathbb{N}^0$ | $\forall x \in T;$ $x \bmod 2^{0+1} \geq 2^0;$ $256 > x > 0;$ $x \in \mathbb{N}^0$ |

Table 3.4: Binarization Scheme for the proposed system

By the time we reach 2^3 we are starting to see a pattern. The include-exclude schema is starting to get long we, so we switched to discrete notation. Lets take an example from the discrete notation above.

Assume you want to know if the 2^3 bit is 1 when x is 255 e.g:

$$x \bmod 2^{n+1} \geq 2^n \rightarrow 255 \bmod 2^{3+1} \geq 2^3 = 255 \bmod 16 \geq 8 = 15 \geq 8 \rightarrow \text{True.}$$

Next is the 2^1 bit 1 when x is 80 e.g:

$$x \bmod 2^{n+1} \geq 2^n \rightarrow 80 \bmod 2^{1+1} \geq 2^1 = 80 \bmod 4 \geq 2 = 0 \geq 2 \rightarrow \text{False.}$$

3.5 Train Model

3.5.1 Convolutional Tsetlin Machine

There are currently several versions implemented for the CTM. They are as following:

1. Convolutional Tsetlin Machine [20]
2. Paralell/CPU CTM [1]
3. GPU/CUDA CTM [2]

Depending on what your setup is they each have their own pros and cons. The Jupyterlab we used, had access to 42 CPUs and one GPU for which specs can be seen in Appendix A.3. Regardless having access to that many CPUs, makes the Paralell version better by default so it is the one we will be using. The CUDA implementation *should* be the fastest, assuming equal setup. In Appendix A.4 we show the CUDA implementation being CPU capped, and therefor bottle-necked, leading to the choice of Paralell Tsetlin CNN.

3.5.2 Hyperparameters

As the previously mentioned thesis [39] decided to use the following:

3.6. VISUAL COMPARISON OF TECHNIQUES

- $T = 75$
- $s = 10$
- $\text{patch_Dim} = (8, 8)$

We will be using this as a baseline. Below can be seen a flowchart explaining how the entire system should be set up.

3.6 Visual Comparison of Techniques

3.6.1 Image Information Breakdown

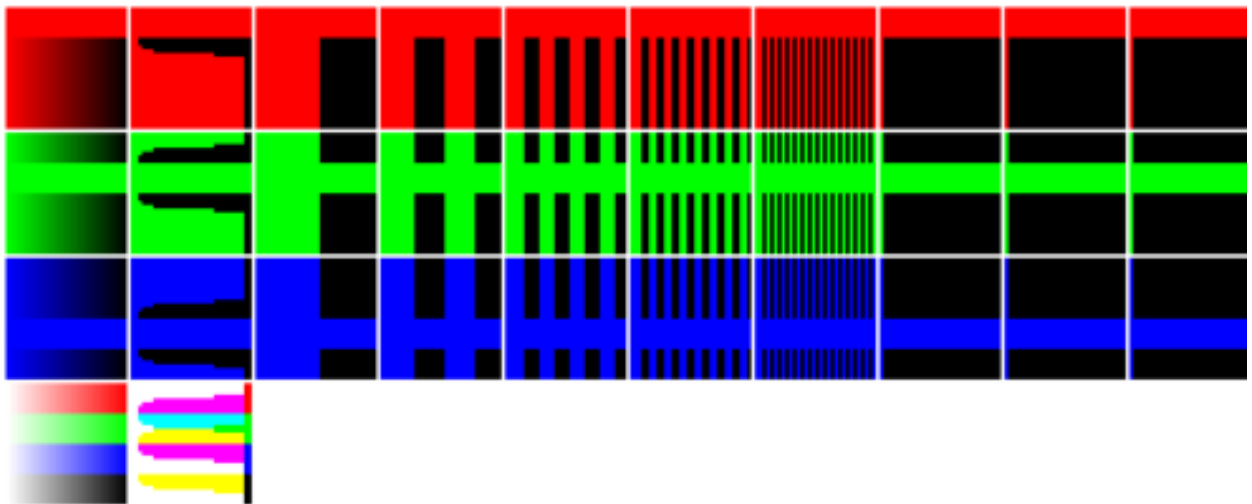


Figure 3.5: Comparison of techniques over an 32x32 image with hue-maps of Red, Green, Blue, and RGB respectively

Above in Figure 3.5 the ground-truth image (bottom left) has 4 rows of hues. The first-most column of images is ground-truth, second is Adaptive Gauss, and third to tenth is binarization in descending order of 2^s exponents $2^7 \rightarrow 2^0$. Rows split up from top to bottom: Red colour channel, Green colour channel, Blue colour channel and composite image respectively. Composite image not added for binarization techniques as the composite would technically just be ground-truth. Do notice that since the image used is 32x32, it is only possible to show 32 unique hues, for that image length. This means that there will exist no information beyond $\lceil \frac{\log 32}{\log 2} \rceil = 5$ bits, in the lossless binarization technique (apart from the base colour).

3.6.2 Example Image from Dataset

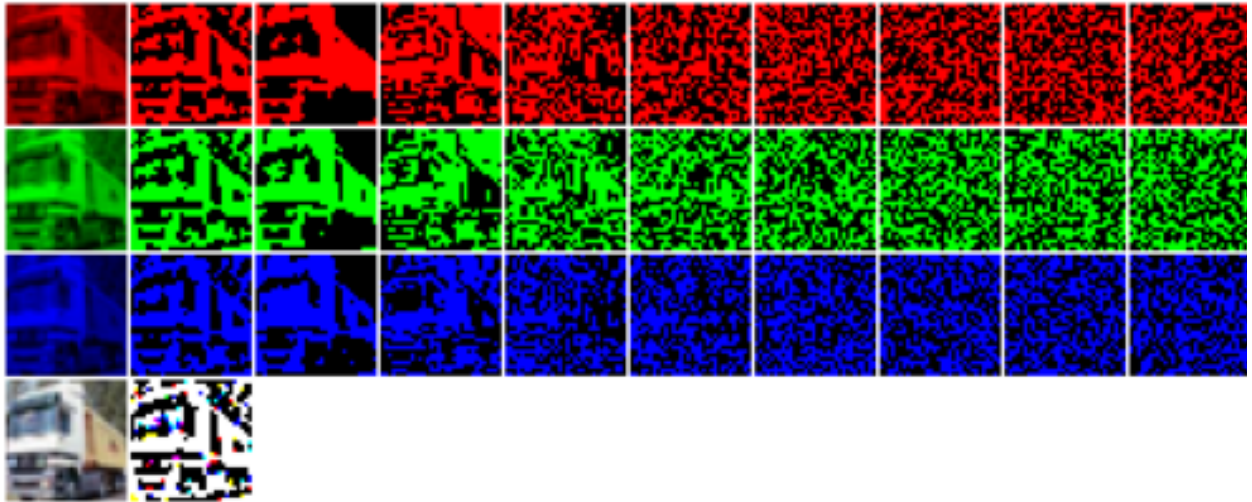


Figure 3.6: Comparison of techniques on Cifar-10_Train_1 in multicolour RGB

In Figure 3.6 all the techniques can be seen on an actual image from the dataset.

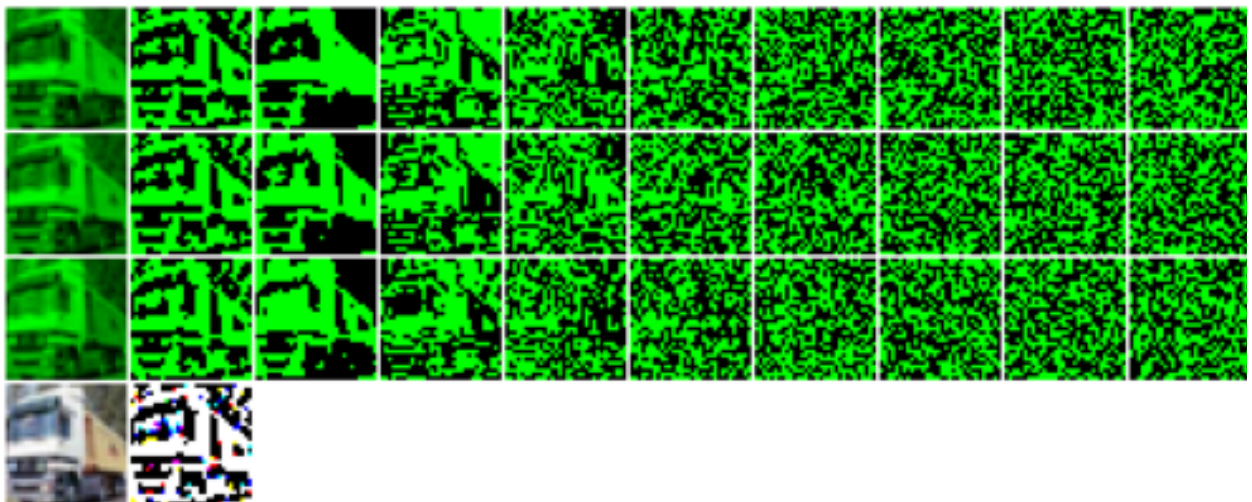


Figure 3.7: Comparison of techniques on Cifar-10_Train_1 in monochrome G

It can be hard to differentiate between what information is stored for each respective channel, for images of larger complexity, when they are also split by colours. Therefor we have in Figure 3.7 used a green representation for all the colour channels. Keep in mind here that green refers to 1, and black refers to 0.

Chapter 4

Performance Evaluation

4.1 CTM Hyperparameters and Manual Loss

4.1.1 Curves and Axis

The x axis/horizontal curve will represent epochs, or one session of training for the neural network per point. The y axis/vertical curve will represent the mean accuracy per epoch, of the mean accuracy per image in percentage per point. The epochs are 0 indexed, so the value at 0 will represent the value after one epoch/training across the input. AGauss will be represented in crosses x while LL4 will be represented as dots $*$.

4.1.2 Observations

Searching the hyper-parameter space is one of the things that allows you to fine tune an already working model. However graphing one of the results per epoch, shows something interesting.

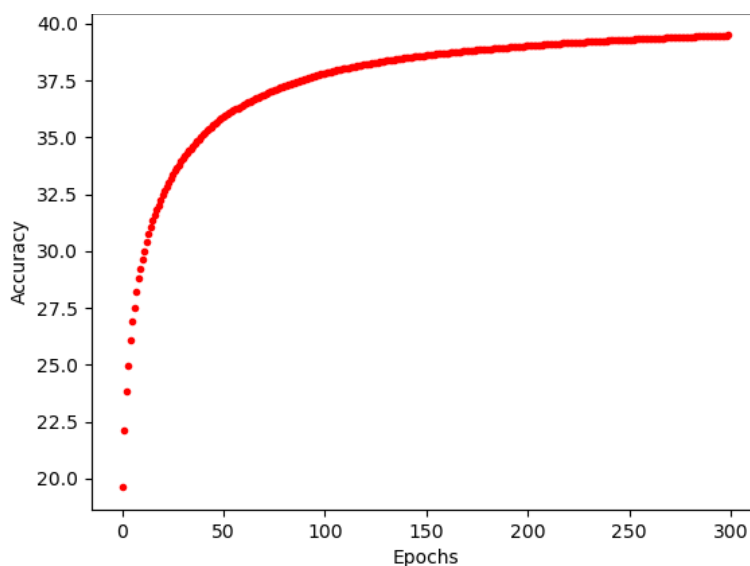


Figure 4.1: Graph of mean of the accuracy per epoch of the mean accuracy per image.

4.2. RESULTS

What one can observe from this curve is that the function in Figure 4.1 resembles a Sigmoid function. However it appears to converge before 1, which is more in line/similar to a logistics function. The reason this is important is it gives 3 factors to consider for convergence and not one. To reiterate, this means that just taking the hyper-parameter that throws the highest value for the last epoch is insufficient, as one also need to consider the slope of the function. A manual loss calculation to determine the best hyper-parameters would require at minimum, a 3 variable equation (and a 299 variable at maximum). Therefore, the hyper-parameter search will fall outside the scope of this thesis.

4.2 Results

The first/initial results of the proposed method referred to as LL8 or *lossless 8 bits* yielded 21% accuracy with 500 images, 800 clauses and 50 epochs. While at first this seems very low, keep in mind that random/chance would be 10%, and we are feeding the system an array that is almost the cube of the input size per image compared to the Adaptive Gauss or AGauss of [39]. The next thing to consider is to look at a few images to see if we can compress the amount of noise being given to the system, to see if we can make it converge faster.



Figure 4.2: Comparison of techniques on Cifar-10_Train_23099 in monochrome Yellow/Purple.

Above in Figure 4.2 we have expanded upon the idea from Chapter 4.1, but changed the binary representation to 2 colours that is easier to distinguish from each other. We use yellow to signify 1 and purple to signify 0, we have also added labels of explanation to each column. As shown in Figure 4.2 you can see that even in lower brackets, a lot of the information seems to be preserved. However it doesn't seem to add much beyond the first few spectrum's. It was therefor concluded to only utilise $2^7 - 2^5$ as input, as there seems to be mostly noise past this level in images such as Figure 3.7. For the next run we therefor omit $2^4 - 2^0$ and see if reducing the noise can make it converge *faster*. This method of using only the $2^7 - 2^5$ bits, will be referred to as LL4 or lossless 4 bits.

4.2.1 Difference in Input-size

The size of the input is vastly different for AGauss, LL4 and LL8. If we assume that the input from the AGauss is standard, we can refer to that as I .

The input size of the LL4, i.e. using only $2^7 - 2^4$ is equal to $I(I + 1) = I^2 + I$.

The input size of the LL8, i.e using the full $2^7 - 2^0$ is equal to $I(I^2 - 1) = I^3 - I$.

4.2.2 Comparing LL4 to AGauss

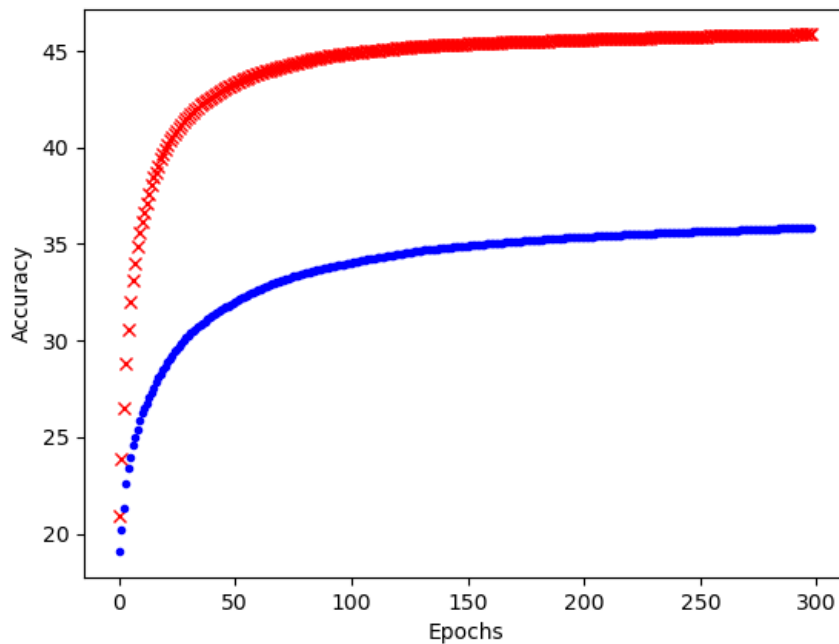


Figure 4.3: Comparison of AGauss and LL4 at 5000 images, 800 clauses.

Consider from Figure 4.3 the current input of the LL4 is bigger than the square of the input of the AGauss method. While it is not most obvious from the entire graph, if you pay close attention to the slope of the red and blue trajectories, it is a lot steeper for AGauss than for LL4.

4.2. RESULTS

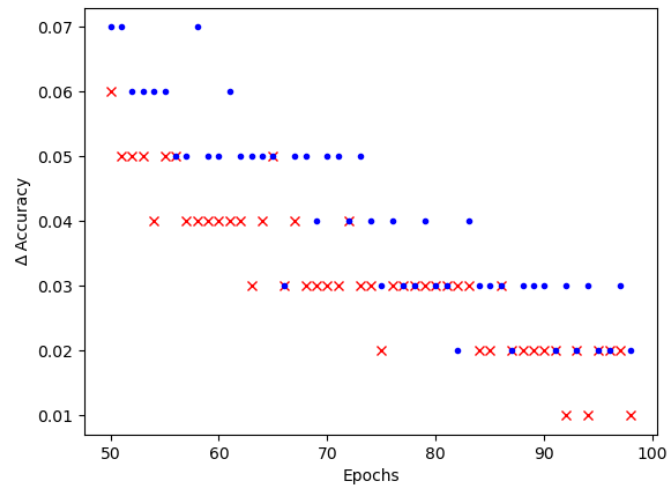


Figure 4.4: Difference in Accuracy/ $\Delta a(x)$ for **AGauss** and **LL4**.

Graphing the difference in epochs value, i.e. $\Delta Accuracy = \Delta a(x) = a(x+1) - a(x)$ and then zoom in x/epochs between 50 and 100 to get Figure 4.4. From Figure 4.4 it is observed that the blue values seem to be a little higher, however not only are they higher, they are almost double on average. The difference between the AGauss and the LL4 is slowly decreasing with epochs, as the slope of the AGauss is steeper. The LL4 is clearly learning faster, but as it has a lower initial value, it does not overtake AGauss within 300 epochs.

4.2.3 Scaling to Full Dataset

As the new system takes in more information, one way to check the latter part of the hypothesis from Section 1.1.4, is to see if the gap closes for more data, and more epochs. As epochs is somewhat already confirmed from Figure 4.4, the next focus will be on data. For the next run we then include the entire training set of 50000 images.

4.2. RESULTS

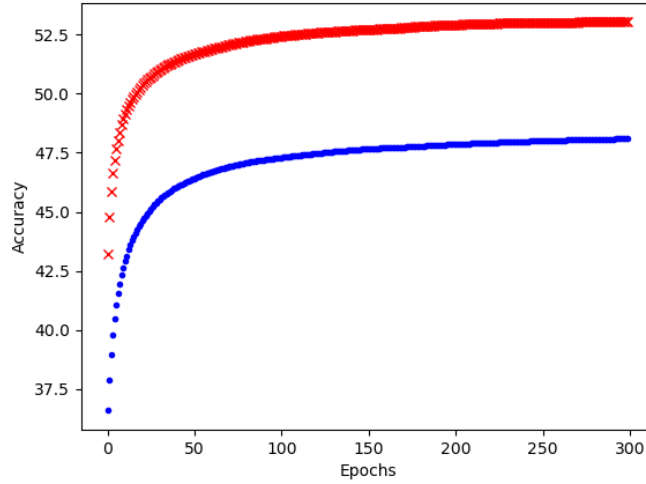


Figure 4.5: Comparison of **AGauss** and **LL4** at 50000 images, 800 clauses

From Figure 4.5 it is observed that the difference of AGauss and LL4 at the 300th epoch has been reduced to only 4.91% compared to the one from Figure 4.3 of 9.99% at the 300th epoch. While it is less obvious, the slope of AGauss is still steeper, and converges faster.

| | AGauss_5k | AGauss_50k | Δ _AGauss | LL4_5k | LL4_50k | Δ _LL4 |
|-----------|-----------|------------|------------------|--------|---------|---------------|
| Epoch 1 | 20.96% | 43.23% | 22.27% | 19.06% | 36.59% | 17.53% |
| Epoch 50 | 43.28% | 51.65% | 8.37% | 31.95% | 46.38% | 14.43% |
| Epoch 150 | 45.46% | 52.71% | 7.25% | 34.90% | 47.66% | 12.76% |
| Epoch 300 | 45.85% | 53.03% | 7.18% | 35.86% | 48.12% | 12.26% |

Table 4.1: Values and differentials from Figure 4.3 and Figure 4.5.

The Table 4.1 was created to better observe the difference in convergence speeds. Just to reiterate Δ _AGauss = AGauss_50k - AGauss_5k. What this table shows is that for larger amounts of data, the LL4 closes the difference to AGauss, while maintaining a more gradual slope. Just increasing the data, reduced the difference between them with a substantial margin, and epochs keeps increasing the accuracy for longer for LL4.

4.2.4 Linear Input/Clauses Ratio Assumption

As the input is 4 times larger than the AGauss setup, what would happen if we ran LL4 with 4 times the clauses to compensate for this, and then compare methods.

4.2. RESULTS

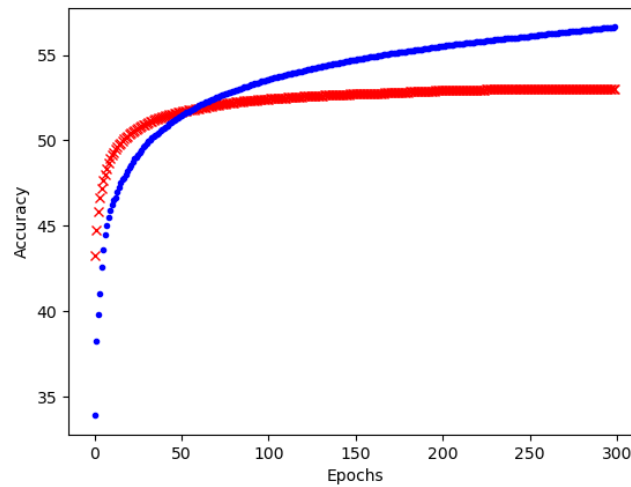


Figure 4.6: Juxtaposition of AGauss and LL4, where LL4 has 4xClauses

Keep in mind that clauses is strongly correlated to accuracy and convergence [56] so this linear assumption might have some flaws to it, but it highlights the importance of slope for a given function. Even though LL4 starts at a lower value, and converges slower, its maximum potential is higher relative to starting value.

4.2.5 Manual Approximation for the Logistic Function

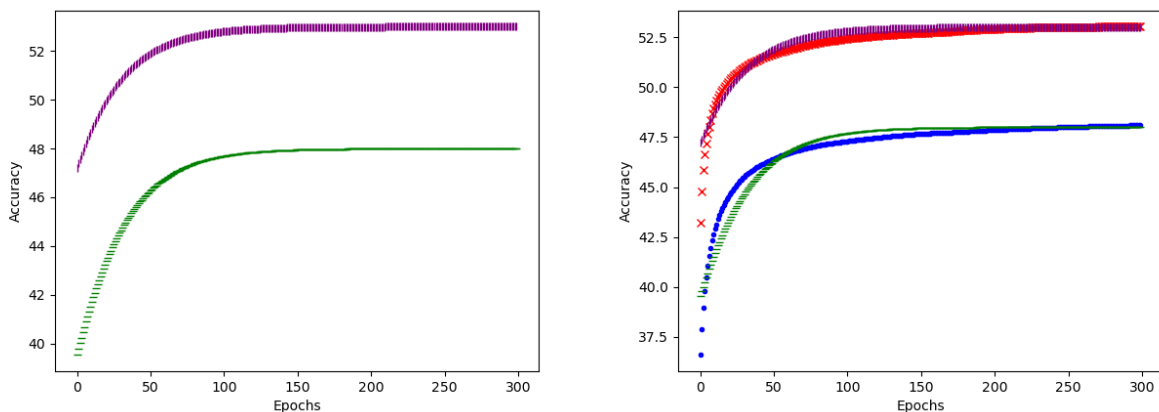


Figure 4.7: Left side showing Logistic Curve approximation, and right side combines it with Figure 4.5.

4.3 Convergence and Factor Analysis

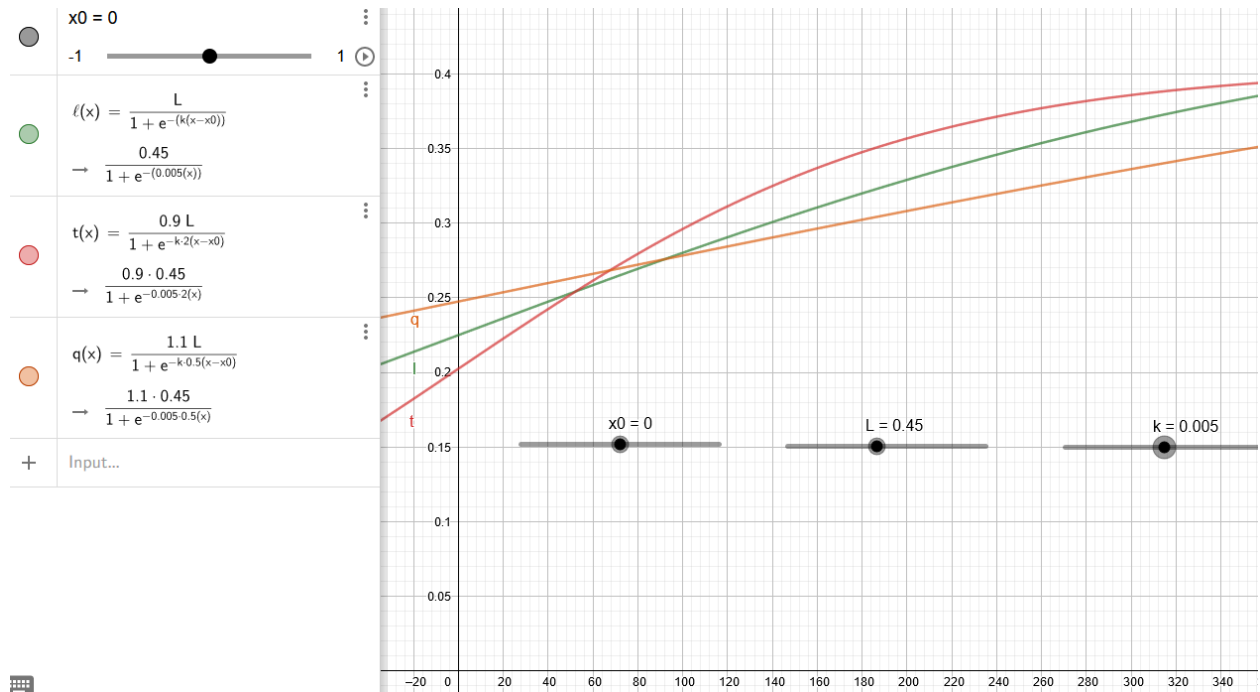


Figure 4.8: Convergence sequences of different valued L and k for the logistics formula using GeoGebra [31].

One of the reasons we put such emphasis on the LL4 increasing values for longer for example in Figure 4.4, is to do with the slope of the function. From the image in Figure 4.1 we can see that the shape of the curve roughly fits a sigma curve/sigmoid function. However as the max value may not approach 1, it then better fits the form of a logistic growth curve. While there exists many ways to get a better approximation to what the exact function of the curve is [52] that falls outside the scope of this thesis. We therefore assumed that the general formula for sigma curves is insufficient, i.e.:

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} = \begin{cases} 1 & \text{if } x \rightarrow \infty \\ 0 & \text{if } x \rightarrow -\infty \end{cases}$$

And instead used the logistic curve/logistic function defined as following.

$$l(x) = \frac{L}{1 + e^{-k(x-x_0)}} = \begin{cases} L & \text{if } x \rightarrow \infty \\ 0 & \text{if } x \rightarrow -\infty \end{cases}$$

where L is the maximum value, k is the growth/slope of the curve, and x_0 is the functions mid/center ($l''(x) = 0$). For my example x will be epochs. Both the value L and the value k is important here. The L value because its the maximum theoretical accuracy, while k tells us if worth training past a given number of epochs.

4.3. CONVERGENCE AND FACTOR ANALYSIS

From Figure 4.8 we have plotted 3 different graphs and take the assumption that we train for 300 epochs/ x and the y axis represents accuracy. The baseline graph is the green $l(x)$, and only takes the input as is from the image into the logistics formula. The red graph $t(x)$ assumes that the maximum accuracy L , is 10% lower, but the slope k is doubled. The orange line $q(x)$, assumes the maximum value L is 10% higher, but the slope k is halved. What we can see from the graph is that even though the $t(x)$ graph seems like the clear winner for this example, we can calculate that: $l(x)$ will overtake $t(x)$ for $x = 409$; $q(x)$ will overtake $t(x)$ for $x = 596$; and $q(x)$ will overtake $l(x)$ for $x = 868$. Now its not always the case that it is feasible to let x get to a certain number, however for examples sake, if x /epochs were doubled, $l(x)$ would win, and if tripled $q(x)$ would win, and stay the winner for all further epochs. If we then compare these to Figure 4.1 we can make assumptions about how steep the slope/ k value is for that given approximation. The reason this is important is how long it is necessary to train a model overall, and how much room for improvement there is.

4.3.1 Speculation on the Slope Difference

For most tests, the initial value of the AGauss is higher, but the learning rate is a lot slower. While going through the full prediction list we noticed that images such as Figure 4.2 were images that the Gauss method usually performed well on. Images that has little background noise, and and looks similar between the Lossy Gauss and the Lossless 2^7 for their respective channels.

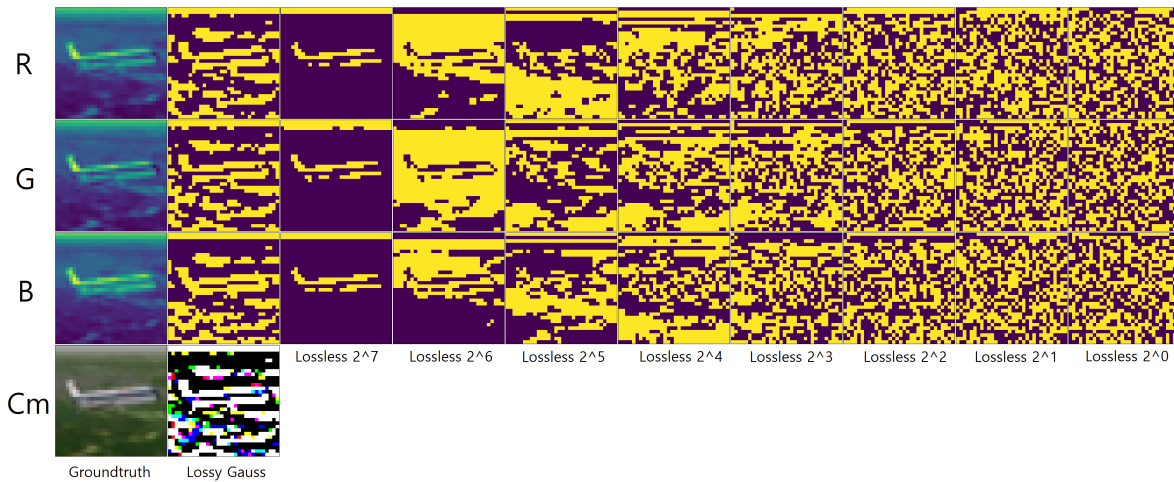


Figure 4.9: Comparison of techniques on Cifar-10_Train_1123 in monochrome Yellow/Purple

Images such as Figure 4.9, were some of the images the AGauss method performed worse on, comparative to LL4. One can see there is more noise in the AGauss, compared to Lossless 2^7 . One can therefore speculate that the Gauss method will encounter problems where the method *fails* to remove noise in a similar manner as presented in lower right Figure 2.2. We did not have the time test this, but it could be part some future work.

Chapter 5

Discussion

5.1 Observation on the RBG Channels

One thing to take note of is that from both Figure 3.7 and Figure 4.9 is the information between the RGB channels for the AGauss method is hardly noticeable, so it would be interesting to compare results using only one of the RGB spectrums.

5.2 Experience and Epiphanies

The goal of the thesis was to enable Tsetlin to have an option of lossless input for computer imagery. This was implemented successfully, however were reduced to account for the latter images being prone to a lot of noise, and to reduce training time. During the earlier stage of testing there were small differences that ended up in the prediction matrix having very different values between the standard deviation of the Gauss and lossless method, leading to some wrong conclusion. This culminated in a lot of writing, and testing that ended up null as the results were not comparable. In the comparison of the techniques we assume the size of input, and clause convergence space is linear and parallel i.e if we double the input size, we should also double clauses to create an even match-up between the 2 technologies. Solving this problem would require knowing the octo-equilibrium of hyper-parameters vs input size, which may not even exist, but a best fit should theoretically exist.

5.3 Retrospect

- *If you could do the project again, what would you do different?*
 1. **Get hold of version control and/or earlier code as early as possible**

We spent a lot of time trying to get this to run on a baseline setup of python 3.6 and tensorflow 2.3 as it was something we had pre-installed and knew were working. However as it turns out, while the code *may* work on this configuration, it requires quite a few modifications, in lieu of the Tsetlin system having a python parser via ctypes, and running baseline C.
 2. **Thoroughly read through any code obtained from other sources**

One of the things we received, was a known working solution for the AGauss

5.4. FUTURE WORK

method, that would, and did make troubleshooting a lot easier. However one minor detail, in how the method did the prediction matrix, was that it performed the mean calculation *after* the training was done, meaning that the final matrix included the prediction for every image. While not normally a problem, it did a standard deviation across all the images, as proposed to the mean or average of them. This bug ended up being a basis for which that we based assumptions on. Most if not everything of the work being done, between receiving the known working method, and discovering the bug ended up being scrapped, as it was based on the wrong conclusions.

3. **Explore back-end capabilities**

At the start of the project, the proxy used to connect to Jupyterlab were locked to a 2 hour session. This means that any task requiring more than 2 hours would be terminated, as the proxy session would be terminated. However as the system allowed for parentless running (& operator in bash), meant that processes could be left running in the background, despite the session ending. The proxy were eventually changed to be sessionless, but knowing about parentless running would have given a lot more time to train and test early on.

4. **Don't have a too low baseline for testing**

When exploring for potential methods, one of the mistakes made was to assume that most methods would be able to gain a few percent above chance, even for as small test as 50 images 800 clauses. This turned out to be enough for most compression/lossful methods, but not enough for lossless methods. This ended up causing some repeat tests needed to be done, to double check the viability of some methods.

5.4 Future Work

One of the main proponents that should be explored is what happens on larger image datasets. While very good for benchmark purposes, the CIFAR10 dataset does not in any way check if either technology works good on scale. While we suspect that both the AGauss, and the LL4/LL8 methods actually *works*, whether or not they translate their effectiveness to images with higher resolution and complexity is another topic.

5.4.1 Check Breakpoints/Equilibrium Methods

While we conclude that the new method will scale up better in the long run, there will of course exist a breakpoint for when you should use which method. At least in terms of clauses and data this can possibly be solved. One of the ways to cross reference, how much a input increase of $I^2 + 1$ with **I** as Gauss baseline input would effect the system, was to check what would happen if: You quadruple the input, and copied the original value for the remaining bits I.E. $(1 \rightarrow 1, 1, 1)$; You quadruple the input, and added random noise **r** to the last digits $(1 \rightarrow r, r, r)$; You quadruple the input, and swapped the last digits for the new system $(1 \rightarrow 2^6, 2^5, 2^4)$. While this wouldn't outright give you the breakpoint, it would give a good indicator as to the overall effect of increasing input size.

5.4. FUTURE WORK

5.4.2 Expand System for \mathbb{R}^N

Right now the system works specifically for 2 dimensional images, albeit being effectively 3d input because of the RGB split. However there exists images such as hyper-spectral imaging either in the form of multidimensional layering [37], data going beyond the visible spectrum of the electromagnetic field [38], or simply data being represented in image form [4]. For this reason it would be an idea to create a generator based on the number of dimensions needed for the problem.

5.4.3 Expand System for Per-pixel Labelling

| # | R 000 | G 150 | B 255 |
|-------|-------|-------|-------|
| 2^0 | 0 | 0 | 1 |
| 2^1 | 0 | 1 | 1 |
| 2^2 | 0 | 1 | 1 |
| 2^3 | 0 | 0 | 1 |
| 2^4 | 0 | 1 | 1 |
| 2^5 | 0 | 0 | 1 |
| 2^6 | 0 | 0 | 1 |
| 2^7 | 0 | 1 | 1 |

Table 5.1: Transpose of the binary representation.

Another solution for input handling, as showed above in Table 5.1, where the transpose of the binary representation is used. What this enables us to do is get 8 binary matrices that's 3 times as big as the original image. Each binary matrix consists of R, G and B 2^X representations, for example $R_0^{2^0}, G_0^{2^0}, B_0^{2^0}, R_1^{2^0} \dots$ and a final size of $[\alpha, Z, X, Y]$ or $[\alpha * Z, X, Y]$. While the current solution works good on a classification based solution, since one may have access to per pixel labelling such as the CAMELYON dataset[35]. It would be a waste to potentially lose information about position or dimensions. A third solution, and workaround for this, is to simply split the RGB into each of their own respective matrices, giving a total of 24 matrices for our example, or $\alpha * Z[X, Y]$ in general. This has the advantage that position is not lost, and no alteration needs to be done to the per-pixel labels.

Output Handling

At this point we have a 24-bit output, as opposed to a single bit. There are ways to interpret this output such as majority vote, or some weighted form of this. However one could hypothesise that the lower representation bits, might have less impact on deciding the outcome of the class, as a variation from 144-145 is less significant than 1-65. Speculating more, this would also mean that if all colours are very light, then some of the higher representation bits might simply be noise. The solution is that instead of using heuristics to try to determine the weight of each number, we will simply use a Tsetlin Machine[18] as this could be said to be a variation of the one-armed bandit problem, which the TM has shows great performance on [18]. The net result of all this, gives us a network that looks something like Figure 5.1 below.

5.4. FUTURE WORK

α = color depth in bits
 X = Input = [A, B, C]
 Y = Label/Output = [A, B, D]/[D]

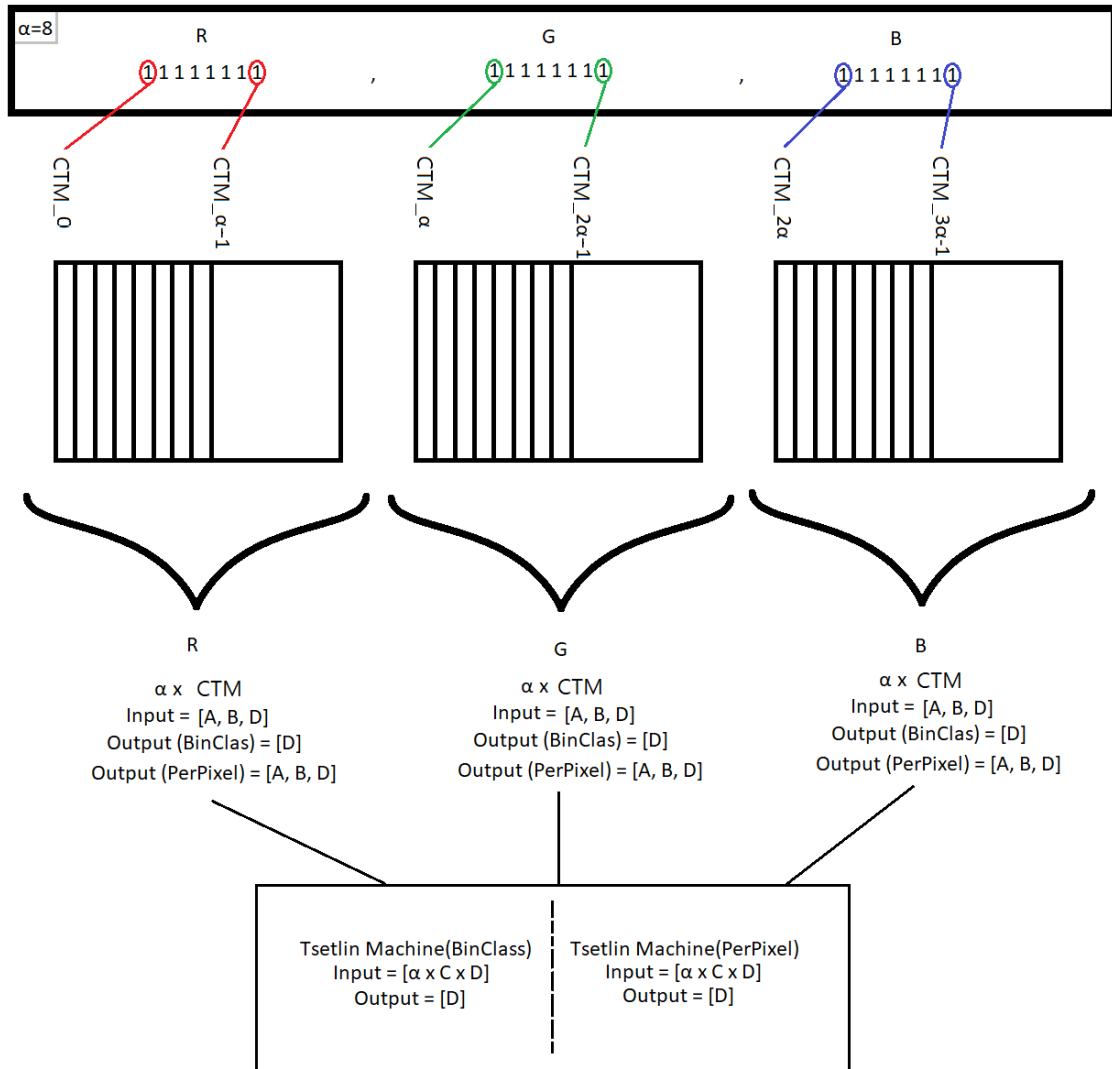


Figure 5.1: Visualisation of the Tsetlin network

What this form of neural network allows is not having to do any changes to the labels for the per-pixel classification. For Binary Classification there would need to be some form of heuristic that decides what is enough data to count one area being affected. The reason that the Tsetlin Machine template remains unchanged is that for both cases, you need to combine the binary representations back to a single output, and that is done by considering each output of each individual CTM. The main difference is that for BinClass you only need to perform this action once, while for PerPixel you need to do this operation for every single pixel. Also keep in mind that for non binary classification, you would need to change the output of the TCNN, to be either one-hot encoded, or binarized in a similar fashion to the one used for the colours, as the TM can only accept 0 or 1.

Chapter 6

Conclusions

The primary goal of the project was to create a method for lossless input of computer images, assuming that the input needs to be in a binary form, and one such method were presented in this thesis. Based on the findings from Section 4.2 and assumptions done in Section 4.3, it is enough to conjecture that the LL4 will eventually beat the AGauss for large enough datasets and/or enough epochs. Beyond this the project sought to see where the proper applications for this method, could be utilised, and while not giving a complete answer, we feel we covered a lot of ground for potential issues. The method should be implementable in other languages than Python, as it requires only nested lists and Boolean operators/mathematics. There is probably room for improvement to efficiency in the code, such as using a python parser down to C level in similar fashion to how the CTM does it. However as we are utilising NumPy already, the overall gain from a pure port is most likely minimal. We hope this thesis will contribute to enable the use of a bigger selection of datasets to be used for the Tsetlin framework, and possibly even make Tsetlin competitive for some of them.

Appendix A

Code Snippets & Misc

A.1 Manual Import of CIFAR10

```
1 import random as rand
2 import numpy as np
3
4 cifar_label_dict = {
5     0: 'airplane',
6     1: 'automobile',
7     2: 'bird',
8     3: 'cat',
9     4: 'deer',
10    5: 'dog',
11    6: 'frog',
12    7: 'horse',
13    8: 'ship',
14    9: 'truck'
15 }
16
17 def importCifarDataModule( file ):
18     import pickle
19     return pickle.load( open( file , "rb" ) , encoding= 'bytes' )
20
21 def exampleImageFromTheDataset( d ):
22     print( "\t Image Info " )
23     print( "Image Channels: \t\t", "RGB: [X, Y, 3]" )
24     print( "Color Depth: \t\t", "8bit: 0-255" )
25     randomIndice = rand.randint( 0, len( d[ 'data' ] ) - 1 )
26     imageArray = d[ 'data' ][ randomIndice ]
27     redChannel = imageArray[ 1023 ]
28     blueChannel = imageArray[ 1024:2047 ]
29     greenChannel = imageArray[ 2048: ]
30     help, help2 = 0, 0
31     image = np.zeros( ( 32, 32, 3 ) )
32     print( "\t Example Image " )
33     for x, y, z in zip( redChannel, blueChannel, greenChannel ):
34         if help == 32:
35             help2 += 1
36             help = 0
37             #Note as pyplot uses a normalisation of the 8 bit channels to fit between 0
38             #and 1, you need to normalise the input
39             image[ help2 ][ help ][ 0 ] = ( x - 0 ) / ( 255 - 0 ) #x
40             image[ help2 ][ help ][ 1 ] = ( y - 0 ) / ( 255 - 0 ) #y
41             image[ help2 ][ help ][ 2 ] = ( z - 0 ) / ( 255 - 0 ) #z
42             help += 1
43     print( "Label-Literal: \t\t", d[ 'labels' ][ randomIndice ] )
44     print( "Label-HumRead: \t\t", cifar_label_dict[ d[ 'labels' ][ randomIndice ] ] )
45     showImageAsPyplot( image )
46
47 cifarPackedLocation = "%DATAPATH%/cifar-10-batches-py/"
48 fullPath = cifarPackedLocation + "filename"
```

A.2. LIBRARY IMPORT OF CIFAR10

```
47 | imageSet = importCifarDataModule(fullFilePath)
48 | exampleImageFromTheDataset(imageSet)
```

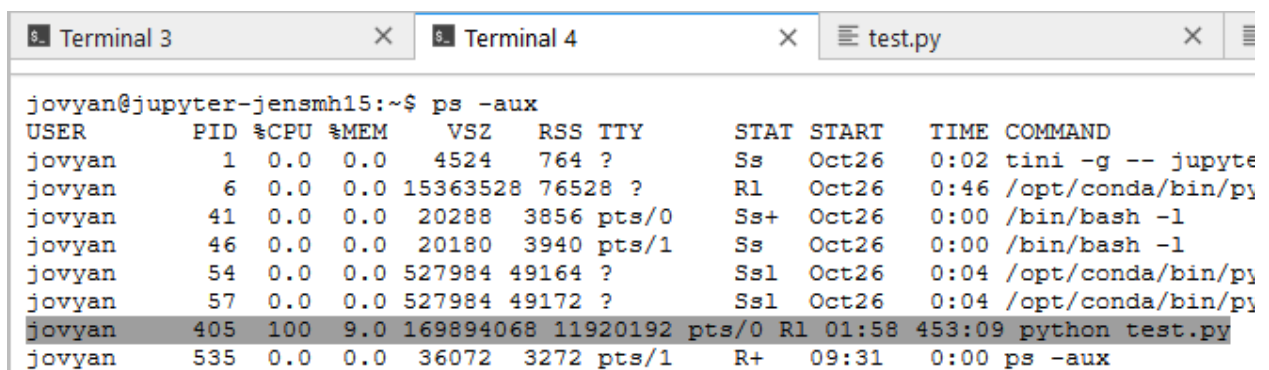
A.2 Library Import of CIFAR10

```
1 | #If you are using tensorflow
2 | from tensorflow.keras.datasets import cifar10
3 | #Or if you are using baseline keras
4 | from keras.datasets import cifar10
5 | #and finally
6 | (X_train, Y_train), (X_test, Y_test) = cifar10.load_data() # Load all data
```

A.3 Readout of "nvidia-smi" - Jupyterlab GPU

```
1 | +-----+
2 | | NVIDIA-SMI 450.66          Driver Version: 450.51.06   CUDA Version: 11.0
3 | |-----+-----+
4 | | GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
5 | | Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M.
6 | |                                           |              | MIG M.
7 | |-----+-----+
8 | |    0   Tesla K80           On          | 00000000:84:00.0 Off   |    0
9 | | N/A   38C    P8             27W / 149W | 0MiB / 11441MiB |    0%      Default
10 | |                                           |              |
11 | |-----+-----+
12 | |
13 | | Processes:
14 | | GPU   GI    CI          PID    Type    Process name          GPU Memory
15 | | ID     ID     |             |          | Process name          Usage
16 | |-----+-----+
17 | | No running processes found
18 | |-----+-----+
19 | |
```

A.4 Terminal Readout of CUDA and CPU capacity



```
Terminal 3 x Terminal 4 x test.py x
jovyan@jupyter-jensmh15:~$ ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
jovyan     1  0.0  0.0   4524    764 ?        Ss   Oct26   0:02 tini -g -- jupyter
jovyan     6  0.0  0.0 15363528 76528 ?        Rl   Oct26   0:46 /opt/conda/bin/python
jovyan    41  0.0  0.0   20288   3856 pts/0    Ss+  Oct26   0:00 /bin/bash -l
jovyan    46  0.0  0.0   20180   3940 pts/1    Ss   Oct26   0:00 /bin/bash -l
jovyan    54  0.0  0.0   527984 49164 ?        Ssl  Oct26   0:04 /opt/conda/bin/python
jovyan    57  0.0  0.0   527984 49172 ?        Ssl  Oct26   0:04 /opt/conda/bin/python
jovyan   405 100  9.0 169894068 11920192 pts/0    Rl   01:58 453:09 python test.py
jovyan   535  0.0  0.0   36072   3272 pts/1    R+   09:31   0:00 ps -aux
```

Figure A.1: Terminal showing CUDA Implementation being locked at 100% CPU Use

A.4. TERMINAL READOUT OF CUDA AND CPU CAPACITY

```
Terminal 3 × Terminal 2 × gaussExpansion.ipynb × FullSetupTest.ipynb ×
jovyan@jupyter-jensmh15:~$ ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
jovyan     1  0.0  0.0   4524    756 ?        Ss   14:51   0:00 tini -g -- jupyter
jovyan     6  0.4  0.0 1722788 74068 ?        Rl   14:51   0:11 /opt/conda/bin/p
jovyan    41  0.0  0.0  20308   3948 pts/0    Ss   14:51   0:00 /bin/bash -l
jovyan    44  0.0  0.0  20312   3940 pts/1    Ss   14:51   0:00 /bin/bash -l
jovyan    54  0.0  0.0 527984 48516 ?        Ss1  14:52   0:00 /opt/conda/bin/p
jovyan    57  0.0  0.0 527984 48396 ?        Ss1  14:52   0:00 /opt/conda/bin/p
jovyan    60  0.0  0.0 527984 48936 ?        Ss1  14:52   0:00 /opt/conda/bin/p
jovyan    63  0.0  0.0 527984 48468 ?        Ss1  14:52   0:00 /opt/conda/bin/p
jovyan    66  0.0  0.0 527984 48924 ?        Ss1  14:52   0:00 /opt/conda/bin/p
jovyan   135  0.0  0.0  11592   3312 pts/0    S+   14:56   0:00 /bin/bash ./para
jovyan   143 1076 11.2 17546132 14903916 pts/0 Rl+  14:56 453:09 python test2.py
jovyan   493  0.0  0.0  36072   3172 pts/1    R+   15:38   0:00 ps -aux
```

Figure A.2: Terminal Showing CPU superseding 100% CPU use for the parallel implementation, i.e. using more than one CPU

Bibliography & Sources

- [1] K Darshana Abeyrathna et al. “Massively Parallel and Asynchronous Tsetlin Machine Architecture Supporting Almost Constant-Time Scaling”. In: *arXiv preprint arXiv:2009.04861* (2020).
- [2] K Darshana Abeyrathna et al. “Massively Parallel and Asynchronous Tsetlin Machine Architecture Supporting Almost Constant-Time Scaling”. In: *arXiv preprint arXiv:2009.04861* (2020).
- [3] Colin Allen, Iva Smit, and Wendell Wallach. “Artificial morality: Top-down, bottom-up, and hybrid approaches”. In: *Ethics and information technology* 7.3 (2005), pp. 149–155.
- [4] Alexander E Bondarev and Vladimir A Galaktionov. “Multidimensional data analysis and visualization for time-dependent CFD problems”. In: *Programming and Computer Software* 41.5 (2015), pp. 247–252.
- [5] Tom B Brown et al. “Language models are few-shot learners”. In: *arXiv preprint arXiv:2005.14165* (2020).
- [6] Eike Budinger et al. “Non-sensory cortical and subcortical connections of the primary auditory cortex in Mongolian gerbils: bottom-up and top-down processing of neuronal information via field AI”. In: *Brain research* 1220 (2008), pp. 2–32.
- [7] George Chang, Wayne C Guida, and W Clark Still. “An internal-coordinate Monte Carlo method for searching conformational space”. In: *Journal of the American Chemical Society* 111.12 (1989), pp. 4379–4386.
- [8] Liang-Chieh Chen et al. *Rethinking Atrous Convolution for Semantic Image Segmentation*. Accessed: 12.09.2020. 2017. URL: <https://arxiv.org/pdf/1706.05587.pdf>.
- [9] Alexandre Joel Chorin. “Numerical solution of the Navier-Stokes equations”. In: *Mathematics of computation* 22.104 (1968), pp. 745–762.
- [10] *CNN Basic setup @ Tensorflow*. URL: <https://www.tensorflow.org/tutorials/images/cnn>. Accessed: 03.12.2020).
- [11] *Codebase at Github*. URL: <https://github.com/Ikantra/losslessBinarizationTsetlin>. Accessed: 07.01.2021.
- [12] Robert Colwell. “The chip design game at the end of Moore’s law”. In: *2013 IEEE Hot Chips 25 Symposium (HCS)*. IEEE Computer Society. 2013, pp. 1–16.
- [13] K. Darshana Abeyrathna et al. “The regression Tsetlin machine: A novel approach to interpretable nonlinear regression”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2164 (2020), p. 20190165. DOI: [10.1098/rsta.2019.0165](https://doi.org/10.1098/rsta.2019.0165).

- [14] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
- [15] *Flowchart Draw.io - Diagrams*. URL: <https://app.diagrams.net/>. Accessed: 18.12.2020.
- [16] Marcus Frean. “The upstart algorithm: A method for constructing and training feedforward neural networks”. In: *Neural computation* 2.2 (1990), pp. 198–209.
- [17] Ian Goodfellow. “NIPS 2016 tutorial: Generative adversarial networks”. In: *arXiv preprint arXiv:1701.00160* (2016).
- [18] Ole-Christoffer Granmo. *The Tsetlin Machine – A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic*. 2018. URL: <https://arxiv.org/pdf/1804.01508.pdf>. Accessed: 10.09.2020.
- [19] Ole-Christoffer Granmo et al. “Learning automata-based solutions to the nonlinear fractional knapsack problem with applications to optimal resource allocation”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37.1 (2007), pp. 166–175.
- [20] Ole-Christoffer Granmo et al. *The Convolutional Tsetlin Machine*. 2019. URL: <https://arxiv.org/pdf/1905.09688.pdf>. Accessed: 11.09.2020.
- [21] Patrick Grother and Kayee Hanaoka. “NIST special database 19 handprinted forms and characters 2nd Edition”. In: *National Institute of Standards and Technology, Tech. Rep* (2016).
- [22] Kaiming He et al. *Deep Residual Learning for Image Recognition*. Accessed: 14 Nov 2020. 2015. URL: <https://arxiv.org/pdf/1512.03385.pdf>.
- [23] Bruce Headey, Ruut Veenhoven, and Alex Weari. “Top-down versus bottom-up theories of subjective well-being”. In: *Citation Classics from Social Indicators Research*. Springer, 2005, pp. 401–420.
- [24] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [25] Gao Huang et al. *Densely Connected Convolutional Networks*. Accessed: 14 January 2020. 2018. URL: <https://arxiv.org/pdf/1608.06993.pdf>.
- [26] *Image Thresholding – Python – OpenCV*. URL: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html. Accessed: 23.11.2020.
- [27] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. “A convolutional neural network for modelling sentences”. In: *arXiv preprint arXiv:1404.2188* (2014).
- [28] Anantharaman Kalyanaraman et al. “Efficient clustering of large EST data sets on parallel computers”. In: *Nucleic Acids Research* 31.11 (2003), pp. 2963–2974.
- [29] *Keras API Homepage*. URL: <https://keras.io/>. Accessed: 27.11.2020.
- [30] *Keras Library for Tensorflow*. URL: https://www.tensorflow.org/api_docs/python/tf/keras. Accessed: 04.10.2020.
- [31] *Knuth: Computers and Typesetting*. URL: <https://www.geogebra.org/>. Accessed: 01.09.2020.
- [32] Alex Krizhevsky. *The CIFAR10 dataset – Toronto Uni*. 2009. URL: <https://www.cs.toronto.edu/~kriz/cifar.html>. Accessed: 03.09.2020.

- [33] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).
- [34] Steve Lawrence et al. “Face recognition: A convolutional neural-network approach”. In: *IEEE transactions on neural networks* 8.1 (1997), pp. 98–113.
- [35] Geert Litjens et al. *1399 H&E-stained sentinel lymph node sections of breast cancer patients: the CAME- LYON dataset [Online]*. https://www.researchgate.net/publication/325479909_1399_HE-stained_sentinel_lymph_node_sections_of_breast_cancer_patients_the_CAMELYON_dataset. Accessed: 08 January 2020. 2017.
- [36] Xiaoping Liu et al. “A bottom-up approach to discover transition rules of cellular automata using ant intelligence”. In: *International Journal of Geographical Information Science* 22.11-12 (2008), pp. 1247–1269.
- [37] Guolan Lu and Baowei Fei. “Medical hyperspectral imaging: a review”. In: *Journal of biomedical optics* 19.1 (2014), p. 010901.
- [38] Dimitris Manolakis and Gary Shaw. “Detection algorithms for hyperspectral imaging applications”. In: *IEEE signal processing magazine* 19.1 (2002), pp. 29–43.
- [39] Erik Mathisen and Halvor S. Smørvik. *Analysis of binarization techniques and Tsetlin machine architectures targeting image classification*. Accessed: 3 Sept 2020. 2020.
- [40] Kumpati S. Narendra and M. A. L. Tthatthachar. *Tsetlin Automata*. 1974. URL: <http://www.dklevine.com/archive/refs4481.pdf>. Accessed: 10.09.2020.
- [41] Friedrich Pukelsheim. “The three sigma rule”. In: *The American Statistician* 48.2 (1994), pp. 88–91.
- [42] Prasanna K Sahoo, SAKC Soltani, and Andrew KC Wong. “A survey of thresholding techniques”. In: *Computer vision, graphics, and image processing* 41.2 (1988), pp. 233–260.
- [43] Arthur L Samuel. “Some studies in machine learning using the game of checkers”. In: *IBM Journal of research and development* 3.3 (1959), pp. 210–229.
- [44] Robert R Schaller. “Moore’s law: past, present and future”. In: *IEEE spectrum* 34.6 (1997), pp. 52–59.
- [45] I Stephen. “Perceptron-based learning algorithms”. In: *IEEE Transactions on neural networks* 50.2 (1990), p. 179.
- [46] Richard S Sutton. “Integrated architectures for learning, planning, and reacting based on approximating dynamic programming”. In: *Machine learning proceedings 1990*. Elsevier, 1990, pp. 216–224.
- [47] Richard S Sutton. “Integrated architectures for learning, planning, and reacting based on approximating dynamic programming”. In: *Machine learning proceedings 1990*. Elsevier, 1990, pp. 216–224.
- [48] Christian Szegedy et al. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. Accessed: 12 Jan 2020. 2016. URL: <https://arxiv.org/pdf/1602.07261.pdf>.
- [49] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. Accessed: 13.11.2020. 2015. URL: <https://arxiv.org/pdf/1512.00567v3.pdf>.

- [50] *Tensorflow Platform Homepage*. URL: <https://www.tensorflow.org/>. Accessed: 29.11.2020.
- [51] Mikhail L'vovich Tsetlin. "Finite automata and models of simple forms of behaviour". In: *RuMaS* 18.4 (1963), pp. 1–27.
- [52] Anastasios Tsoularis and James Wallace. "Analysis of logistic growth models". In: *Mathematical biosciences* 179.1 (2002), pp. 21–55.
- [53] Wendell Wallach, Colin Allen, and Iva Smit. "Machine morality: bottom-up and top-down approaches for modelling human moral faculties". In: *Ai & Society* 22.4 (2008), pp. 565–582.
- [54] Ronald J Williams and David Zipser. "A learning algorithm for continually running fully recurrent neural networks". In: *Neural computation* 1.2 (1989), pp. 270–280.
- [55] Rohan Kumar Yadav et al. "Human-Level Interpretable Learning for Aspect-Based Sentiment Analysis". In: *The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)*. AAAI. 2021.
- [56] Xuan Zhang et al. "On the Convergence of Tsetlin Machines for the IDENTITY-and NOT Operators". In: *arXiv preprint arXiv:2007.14268* (2020).